GoCAD: GPU-Assisted Online Content-Adaptive Display Power Saving for Mobile Devices in Internet Streaming

Yao Liu[1], Mengbai Xiao[2], Ming Zhang[2], Xin Li[3], Mian Dong[4], Zhan Ma[5], Zhenhua Li[6], Songqing Chen[2]

[1]SUNY Binghamton
yaoliu@cs.binghamton.edu

[2]George Mason University
{mxiao3, mzhang8,sqchen}@gmu.edu

[3]Samsung Telecommunications America
x.li@samsung.com

[4]AT International, Inc.
dongmian@gmail.com

[5]Nanjing University
zhan.ma@gmail.com

[6]Tsinghua University
lizhenhua1983@tsinghua.edu.cn

ABSTRACT

During Internet streaming, a significant portion of the battery power is always consumed by the display panel on mobile devices. To reduce the display power consumption, backlight scaling, a scheme that intelligently dims the backlight has been proposed. To maintain perceived video appearance in backlight scaling, a computationally intensive luminance compensation process is required. However, this step, if performed by the CPU as existing schemes suggest, could easily offset the power savings gained from backlight scaling. Furthermore, computing the optimal backlight scaling values requires per-frame luminance information, which is typically too energy intensive for mobile devices to compute. Thus, existing schemes require such information to be available in advance. And such an offline approach makes these schemes impractical.

To address these challenges, in this paper, we design and implement GoCAD, a GPU-assisted Online Content-Adaptive Display power saving scheme for mobile devices in Internet streaming sessions. In GoCAD, we employ the mobile device's GPU rather than the CPU to reduce power consumption during the luminance compensation phase. Furthermore, we compute the optimal backlight scaling values for small batches of video frames in an online fashion using a dynamic programming algorithm. Lastly, we make novel use of the widely available video storyboard, a pre-computed set of thumbnails associated with a video, to intelligently decide whether or not to apply our backlight scaling scheme for a given video. For example, when the GPU power consumption would offset the savings from dimming the backlight, no backlight scaling is conducted. To evaluate the performance of GoCAD, we implement a prototype within an Android application and use a Monsoon power monitor to measure the real power consumption. Experiments are conducted on more than 460 randomly selected YouTube videos. Results show that GoCAD can effectively produce power savings without affecting rendered video quality.

WWW 2016, April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4143-1/16/04.
DOI: http://dx.doi.org/10.1145/2872427.2883064.

Keywords

Internet Mobile Streaming; Liquid-Crystal Display (LCD); Power Consumption; Backlight Scaling; GPU; Storyboard

1. INTRODUCTION

Growth in the popularity of mobile devices, including smartphones and tablets, has changed the way users consume video content today. For example, mobile devices now comprise over half of YouTube's views [6]. The video streaming experience on mobile devices, however, is constrained by the on-device battery capacity. On a mobile device, both the wireless network interface card and the display consume a significant portion of the battery power for data transmission and content rendering, respectively. Many existing studies have proposed to reduce power consumption of the wireless network interface cards by carefully scheduling data transfers and putting them into low power sleep mode for as long as possible [15, 16, 18]. The display, however, cannot be put into sleep mode and must be kept active during the entire video playback. A previous study shows that the display subsystem is responsible for about 38% to 68% of the total power consumption during video playback [8]. This portion is expected to be larger on the high-resolution and larger-sized display panels.

Mobile devices are typically equipped with one of two types of display panels: (i) Liquid-Crystal Displays (LCD) or (ii) Organic Light-Emitting Diode (OLED) displays. The two types of displays operate on different principles and have vastly different power consumption characteristics. OLED displays usually require a special hardware control circuit to manage their power consumption [26]. Today, LCD still dominates the transparent display market as manufacturing OLED is typically an order of magnitude more expensive than LCD. In this study, we focus on saving the power consumed by LCD displays.

In the LCD display subsystem, a major power drainer is the backlight. Thus, backlight scaling has been proposed to reduce the power consumption of the display by dimming the backlight. At the same time, the brightness *perceived* by the human eye (intensity) is maintained by increasing the affected image's luminance. In this way, the original image can be rendered with little or no distortion. However, implementing a backlight scaling strategy with luminance compensation to save power *during video playback* is challenging. *First*, to maintain rendered video quality, the backlight level cannot be reduced below a point determined by the brightness characteristics of each frame. This constraint requires that the max-

imum pixel luminance of every frame be determined, which can be both time and energy intensive for mobile devices to compute. *Second,* luminance compensation must be performed for every pixel in every video frame. Thus, if increasing the luminance of one pixel takes only several tens of CPU cycles, increasing the luminance for the entire frame of a high resolution video on a high resolution display could consume tens of millions of CPU cycles. While a powerful CPU could complete this task in real time, on a mobile device, the corresponding power consumption overhead would negate the power savings achieved via dimming the backlight. As a result, some previous studies have suggested to pre-compute the backlight scaling schedule *offline*, e.g., [20, 22], and perform luminance compensation using external computing resources [10, 23], e.g., at a proxy server. Basically, such an offline approach requires the video be made available in advance so that the luminance information of every frame can be extracted and then the backlight scaling schedule can be determined for each frame based on the calculation. Although the external computations proposed in these methods allow power to be saved through backlight scaling, the offline approach taken and the additionally required infrastructure render these strategies impractical.

To overcome these challenges towards practical battery power saving, in this work, we propose a GPU-assisted Online Content-Adaptive Display power saving scheme, called GoCAD, for mobile devices in Internet streaming sessions. Instead of relying on external computing resources for offline processing, GoCAD operates in an online manner given small batches of video frames. To reduce the power consumption, GoCAD uses the Graphics Processing Unit (GPU) to extract per-frame luminance information from hardware-decoded video frames during video streaming playback. Then it employs a dynamic programming approach for determining the optimal backlight scaling schedule. To render a video frame with little or no distortion while dimming the backlight, GoCAD uses the RenderScript framework to interface with the GPU on mobile devices to adjust the pixel luminance of the frame in real-time. Compared to schemes using the CPU for luminance compensation, this allows GoCAD to achieve a net power savings in combination with the backlight scaling strategy.

In addition, GoCAD also makes novel use of video storyboard information to predict if power savings can be achieved. Storyboard information today is pre-computed and made widely available by service providers. For example, YouTube added the storyboard feature in January 2012 to provide thumbnail previews of frames selected at regular intervals within the video [5]. Other video service providers, including Netflix and Hulu, were even earlier adopters of this feature than YouTube. Storyboards are often downloaded by the mobile video streaming application before video playback starts, which allows users to browse the video storyline. The availability of this storyboard information makes it possible for GoCAD to determine if backlight scaling should be used before the playback starts.

To evaluate the effectiveness of GoCAD, we implement it within a mobile video streaming application on the Android platform. During video playback, the mobile application sets the backlight according to the backlight scaling calculated online and simultaneously performs luminance compensation by increasing pixel luminance through GPU computations. We randomly selected more than 460 YouTube videos for evaluation. A Monsoon power monitor is used for real measurements of power consumption. Results show that with GoCAD, our video streaming application can save power for 76.3% videos with negligible (up to 5%) pixel distortion, and 88.7% videos if 10% pixel distortion is allowed. For 56.0% of the videos in our study, the GoCAD can achieve more than 500 mW (10%) of power savings.

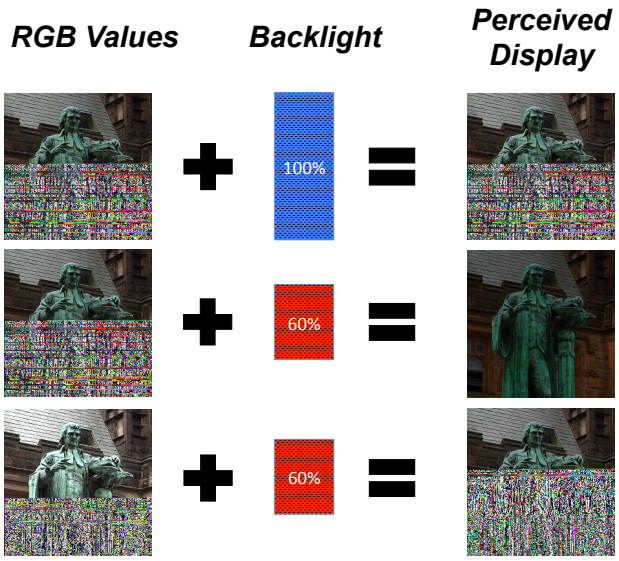


Figure 1: Backlight scaling and luminance compensation.

The remainder of the paper is organized as follows. Section 2 discusses the background. We present the design of GPU-assisted Online Content-Adaptive Display power saving mechanism in Sections 3 and 4. The implementation is described in Section 5, and the evaluation is discussed in Section 6. We discuss related work in Section 7 and make concluding remarks in Section 8.

2. BACKGROUND

2.1 YUV Color Space

Video decoders usually operate over video streams in the YUV color space. In the term "YUV", Y indicates the luminance component (the brightness) and U and V are the chrominance (color) components. Because display hardware uses the RGB color space, decoded video frames must be converted from the YUV color space to the RGB color space before they are displayed. Please refer to [1] for the conversions between YUV and RGB color spaces.

2.2 Saving Power in Liquid-Crystal Display

A liquid-crystal display (LCD) is a flat panel display that uses the light modulating properties of liquid crystals. Liquid crystals do not emit light directly. Instead, the backlight on the LCD panel illuminates the liquid crystals. A liquid crystal's transmittance can vary so that different pixels can have different luminance levels even though the intensity of the backlight is the same for all liquid crystals.

Previous studies have pointed out that the backlight of an LCD display dominates the energy consumption of the display subsystem and that its power consumption is different at different brightness levels [8, 27]. Therefore, power can be saved if we reduce the backlight level of the LCD display. However, simply reducing the backlight level can lead to image distortion, which is normally defined as the resemblance between the original image and the backlight-scaled image [10, 29]. For example, in Figure 1, the top of the figure shows the original image displayed with 100% backlight level. While reducing the backlight level can save display power consumption, it will result in a darker version of the image be rendered as shown in the middle of the figure.

2.3 Backlight Scaling and Luminance Compensation

One way to resolve the problem is by concurrently scaling both the backlight level and the pixel luminance [9, 11, 13, 23]. Suppose the display backlight levels lie within the range $(0, 1]$, where 0 indicates that the backlight is off, and 1 indicates maximum brightness. With backlight scaling, for every frame in the video, we set the display backlight level to $b \in (0, 1]$. To avoid distortion under reduced backlight levels, *luminance compensation* [10, 29] is used to increase the luminance (the Y component of the YUV representation) of every pixel in the frame by a factor of $\frac{1}{b}$. In this way, the *observed* pixel luminance is adjusted to a value that is the same as that of the original frame's: $Y' \times b = Y \times \frac{1}{b} \times b = Y$. With joint backlight scaling and luminance compensation, display power consumption can be saved without any observable fidelity loss. The idea is also depicted in bottom of Figure 1. If we increase the luminance of every pixel in the frame, the frame can be rendered with no distortion.

While the idea of increasing the luminance of video frames at runtime to compensate for a dimmed backlight is straightforward, significant challenges lie in determining the appropriate backlight scaling level, b, and adjusting the luminance of every pixel in the video in an energy efficient manner.

First, care must be taken when choosing the backlight scaling level, b, because we cannot scale up the Y component of a pixel to be higher than its maximum (i.e., 255). If b is chosen to be a value such that $Y \times \frac{1}{b} > 255$, then the observed luminance will be lower than the luminance under the original brightness level, distorting the displayed image. In previous work this distortion has been referred to as a "clipping artifact" [12]. To avoid creating these "clipping artifacts," we must limit any brightness adjustments to $b \geq \frac{Y^{(max)}}{255}$, where $Y^{(max)}$ is the maximum pixel luminance in the frame. However, this requires that we compute $Y^{(max)}$ of every frame in the video, which is both computation and time intensive. Given the stringent timing requirement in video streaming applications and the need for net power savings, much existing work has proposed to offload this task to external computing resources. In these schemes, external machines pre-process videos and extract the per-frame luminance information in an offline manner [17, 20, 22]. This makes these schemes hardly practical.

Second, to increase the luminance of every pixel in the frame by a factor of $\frac{1}{b}$, many techniques rely on the CPU to perform per-pixel manipulation [11, 13]. However, this pixel-level manipulation is also time and computation intensive and may significantly offset, or even negate, the power savings achieved via backlight dimming. For example, for a 30 fps video, a single frame should be rendered approximately every 33.3 ms. However, according to our experiments on Android devices, the CPU usually needs on the order of hundreds of milliseconds to perform per-pixel luminance compensation on a 1280×720 frame. To avoid this time and computation intensive step on the mobile device, Hsiu et al. and Lin et al. propose to simply choose a critical backlight level for each frame as the scaling constraint [17, 20] rather than performing luminance compensation. This strategy, however, can lead to a large amount of distortion of the displayed frames. Ruggiero et al. consider a special multimedia processor, the Freescale i.MX31, which has an Image Processing Unit (IPU) that can be used to offload the luminance adjustment task from the CPU [25]. Rather than a client-side solution, Pasricha et al. [23] and Cheng et al. [10] propose to migrate the computation to an intermediate proxy server that computes the backlight scaling schedule and transcodes a luminance-adjusted version of the original video.

Compared to existing work, our proposed solution does not rely on external computational resources to pre-compute per-frame luminance information offline or to perform luminance compensation. Instead, GoCAD accesses raw video frames from the output of the mobile devices' hardware video decoder during video streaming playback. It uses the RenderScript framework to interface with the GPU to both extract luminance information and, later, to perform luminance compensation. Using the GPU for these per-pixel tasks in a highly parallelized manner consumes far less power than the CPU and satisfies the stringent timing requirement for real-time video frame rendering.

3. DESIGN OF GoCAD

3.1 Overview

GoCAD works in an online manner. It extracts luminance information of video frames as they are being decoded during video streaming playback. To determine the backlight scaling schedule that maximizes power savings under a set of constraints, we collect a batch of frames by buffering the decoded raw frames for a small amount of time before rendering them. Since video streaming applications already need to wait for enough video content to be downloaded before playback can start to avoid playback hiccups (the initial "buffering phase" [24]), temporarily buffering the decoded raw frames will not incur additional playback delay. After the backlight scaling schedule is calculated, we perform luminance compensation on the frames and render them on the display with lower backlight levels.

Logically, GoCAD consists of three modules: the *Scanning Module*, the *Dynamic Programming Module* and the *Rendering Module*. They work together in a pipeline fashion for processing decoded frames before rendering.

Scanning Module: As video frames are being decoded, the Scanning Module selects beacon frames at fixed interval. For every beacon frame, the Scanning Module extracts its pixel luminance information. It does so by cutting each frame into sub-images and computes their corresponding pixel luminance histograms, taking advantage of the highly parallel architecture of the GPU. The Scanning Module then collects all results together to get the histogram of the entire frame and forwards this information to the Dynamic Programming Module.

Dynamic Programming Module: Given an input sequence of maximum beacon frame luminance values and settings of variables associated with constraints (discussed later), the Dynamic Programming Module outputs a backlight scaling schedule that minimizes the backlight levels. To maximize power savings under constraints, this module runs only when the Scanning Module has forwarded pixel luminance histogram information from enough beacon frames to form a meaningful batch of frames. In this study, we buffer decoded frames for 4 seconds. That is, if a video is encoded at 30 frames-per-second, we can form a meaningful batch of 120 frames for backlight scaling schedule calculation.

Rendering Module: This module is responsible for synchronizing frames for rendering to the display during video playback. It sets the backlight level according to the schedule computed by the Dynamic Programming Module. Before rendering each frame with backlight scaling, the rendering module also performs luminance compensation for every pixel of the frame.

The core design piece of our GoCAD mechanism is the Dynamic Programming Module, which determines optimal backlight levels subject to a set of constraints associated with maximum lu-

minance values, user tolerance to display flickering, and responsiveness of display hardware. Next, we present the details of these constraints and the design of our dynamic programming algorithm that computes optimal backlight scaling schedule subject to these constraints.

3.2 Display Power Saving Constraints

Our dynamic programming algorithm enforces three constraints on computed backlight levels:

Distortion Constraint. To avoid creating "clipping artifacts", no adjusted luminance value in any pixel in any frame can exceed 255. This distortion-based constraint gives a *lower bound* on the adjusted backlight level for every frame in the video. The distortion constraint is expressed in our dynamic programming algorithm by restricting the backlight level of frame f, b_f, to be greater than $\frac{Y_f^{(max)}}{255}$, where $Y_f^{(max)}$ indicates the maximum luminance value of video frame f.

Variation Constraint. While it is tempting to set the backlight level of every frame to its lowest possible value subject to the distortion constraint to maximize power savings, changing backlight levels between frames could cause inter-frame brightness distortion, often perceived as flickering, when the variation between two consecutive backlight levels is above a certain threshold. Therefore, we need to *limit* the brightness variation between two consecutive frames to reduce this flickering effect. In our dynamic programming algorithm, the variation constraint is enforced by setting the value of Δ_b, a variable that limits the ratio of change in backlight level. That is, if the backlight is at level b_f at frame f and b_{f+1} at frame $f+1$, then, in the dynamic programming algorithm, we select values of b_{f+1} such that $b_f \times (1-\Delta_b) \leq b_{f+1} \leq b_f \times (1+\Delta_b)$.

Duration Constraint. In addition, due to limitations of hardware performance, it is impossible to adjust the backlight promptly for every video frame, as the underlying hardware requires a minimum amount of time to apply any brightness adjustments. This additional, hardware-dependent constraint makes per-frame backlight scaling infeasible in practice. As a result, we must specify a *minimum interval* (in terms of numbers of frames) where the backlight level must remain constant. In our dynamic programming algorithm, the duration constraint is enforced by setting the value of a variable d_{min}, which specifies the minimum number of frames the backlight must remain at the same level.

3.3 Dynamic Programming Algorithm

Our dynamic programming algorithm runs in an online manner, piecewise over small batches of video frames, as raw video frame information becomes available from the mobile device's hardware video decoder and maximum pixel luminance information of the frames are extracted by the Scanning Module using the GPU.

The design of this algorithm represents a balance over two alternatives: a purely online algorithm to make backlight scaling decisions solely based on the current and previous frames can lead to either non-optimal backlight scaling level assignments, or image distortion, because it is difficult to estimate future luminance values that are needed for calculating optimal backlight scaling levels. On the other hand, a fully-offline backlight scaling algorithm would require that maximum luminance information be computed before the video is played on the mobile device, either requiring external computational resources or additional computation (and power consumption) on the mobile device before video playback can start.

Specifically, the algorithm computes the values of the function $B(f, b)$, the minimum cumulative backlight levels ending at frame f with the final backlight interval set to level b. $B(f, b)$ can be computed by the following recurrence:

$$B(f, b) = \min_{f', b'}(b \times (f - f') + B(f', b')), \quad (1)$$

where f' is the frame number of the end of the previous interval, subject to $f' \leq f - d_{min}$, and b' is the backlight level of the previous interval. b' is also subject to the variation constraint discussed above.

Our dynamic programming algorithm will minimize power consumption if a linear relationship exists between backlight levels and display power. We confirm that such a linear relationship exists in Figure 6. To calculate backlight values from the algorithm, we must first run a forward step to compute the values of $B(f, b)$. Then we must run a backward step where values of b_f, the optimal backlight value at video frame f, are recovered. The forward and backward steps are computed separately for each batch of frames.

The backlight setting at the beginning of each batch is set to the value computed for the end of the previous batch. A scaling factor, $C(b)$, is applied at the final frame of each batch for each value of b to make it less likely that the algorithm chooses a backlight setting for the final frame of the batch so that no feasible solution exists for the first frame of the next batch. (This situation could occur if a significant change in maximum luminance occurs at the boundary of batches.) If no feasible solution exists for a batch of frames, the dynamic programming algorithm is retried for the batch without applying constraints at the boundary between the first frame of the current batch and backlight value at the last frame of the previous batch.

Pseudocode to compute the dynamic programming recurrence is depicted in Figure 2. (Details of initialization and array bounds checking are omitted from the pseudocode.)

4. USING STORYBOARD FOR POWER SAVINGS PREDICTION

When using the GPU for computing luminance histogram and for luminance compensation, we still need to take into account the additional power consumed by the GPU. That is, we need to determine whether or not the display power that can be saved is greater than the power consumed by the GPU. It is only appropriate to apply our GoCAD mechanism when net power savings can be achieved.

In this section, we discuss how to improve our basic GoCAD mechanism by leveraging the video storyboard information for predicting if net power savings can be achieved. Storyboard is commonly available from many video streaming service providers (e.g., YouTube, Netflix, Hulu, etc.). The storyboard provides thumbnails of video frame images sampled at fixed intervals. These thumbnail images are pre-computed by the online video streaming services and are often made available via APIs used by the mobile video streaming applications. Storyboard information allows us to perform a significantly smaller amount of computation to obtain an estimate of luminance information than the computation required to extract exact luminance information by processing a full set of raw video frames.

Storyboard images cannot be directly used for backlight scaling because their lack of full video information and long sampling intervals (usually longer than 1 second) can result in significant visual distortion of the backlight-scaled video. Storyboard information, however, can be used to estimate the amount of power that will be saved in our online algorithm. Low cost estimation of whether GoCAD can save power can result in improved aggregate performance since GoCAD can be turned off when power may not be saved.

1: ▷ On input $Y^{(max)}[F_{begin} : F_{end}]$, indicating a sequence of maximum luminance values from frame F_{begin} to frame F_{end} in a video, compute L, an array containing minimum backlight levels subject to constraints.
2: ▷ Compute the recurrence
3: **for** f **in** $[F_{begin}, F_{end}]$ **do**
4: **for** f' **in** $[F_{begin}, f - d_{min}]$ **do**
5: **for** $b \geq \max(Y^{(max)}[f' + 1 : f])/255$ **do**
6: **for** b' **in** $[b/(1 + \Delta_b), b/(1 - \Delta_b)]$ **do**
7: **if** $b \times (f - f') + B[f', b'] < B[f, b]$ **then**
8: $B[f, b] = b \times (f - f') + B[f', b']$
9: $H[f, b] = (f', b')$
10: ▷ Backtracking phase
11: L = array(F)
12: $f = F_{end}$
13: $b = \arg\min_{b'} B[f, b'] \times C[b']$
14: **while** $f >= F_{begin}$ **do**
15: $(f', b') = H[f, b]$
16: $L[f' + 1 : f] = b$
17: $f \leftarrow f'$
18: $b \leftarrow b'$
19: **return** L

Figure 2: Computing the optimal backlight scaling level with $O(|F|^2 \times |b|^2)$ complexity, where $|F|$ indicates the number of frames in the batch and $|b|$ indicates the number of possible values of b. In the algorithm, $B[f, b]$ indicates the sum of the backlight levels ending at frame f whose final constant-brightness interval level is b, F_{begin} indicates the beginning frame and F_{end} the ending frame in a batch of video frames, d_{min} is the shortest allowed constant-backlight interval (in frames), $Y^{(max)}[f' : f]$ indicates the maximum luminance values over video frames f' through f, Δ_b encodes the constraint specifying the allowable backlight level changes between adjacent frames, $H[f, b]$ is a history array that records how the minimum-backlight-level array $B[f, b]$ was constructed, and $C[b]$ is a set of compensation values, reflecting prior knowledge about expected backlight values in most videos, to reduce the likelihood that backlight scaling values at the end of an interval will result in unsatisfiable constraints when the backlight scaling values for the next interval are computed.

Our modified storyboard-GoCAD scheme thus involves the following steps:

1. Extract maximum luminance values from the storyboard.
2. Use the maximum luminance values of storyboard images to estimate the amount of power that our GoCAD scheme can save.
3. If the estimated savings is positive, play the video using the GoCAD scheme. Otherwise run playback without backlight scaling.

4.1 Storyboard Quality and Availability

Figure 3 shows an example storyboard of a YouTube video. A storyboard is a series of reference images, typically of thumbnail size, that are scaled down from frames sampled from a video. Storyboards are usually provided as a viewing experience enhancement feature in the video player, providing users with the ability to browse the video's story line and locate content of interest with ease by dragging the playback progress bar. Storyboard images are

(a) Storyboards are provided as an enhancement feature in the YouTube video player. Both mobile and desktop YouTube applications make use of storyboards. (This figure shows the YouTube desktop video player.)

(b) A storyboard consists of multiple thumbnail images.

Figure 3: The Storyboard of a YouTube video.

organized by time and put into a composite `.jpg` file as shown in Figure 3(b). In storyboards generated by YouTube, for example, each `.jpg` file may contain up to 10×10 storyboard thumbnail images. Depending on the number of storyboard images, multiple `.jpg` files may be needed. Storyboard images are provided at different quality levels. For example, we observed four storyboard image quality levels on YouTube (`L0`, `L1`, `L2` and `L3`), with the resolution of 48×27 (`L0`), 80×45 (`L1`), 160×90 (`L2`), and 240×180 (`L3`), respectively. These images are extracted at different intervals. The `default` storyboard (`L0`) consists of 100 images at the resolution of 48×27 (lowest quality) no matter how long the video is. Storyboards at other levels (`L1`, `L2`, and `L3`) consist of images that are sampled every 1, 2, 5, or 10 seconds, depending on the length of the video. Not all quality levels of storyboards are available for all videos: `L0` and `L1` are available for most videos, while only a very small percentage of videos have storyboards at `L3` quality level. Storyboards found on other video streaming websites, such as Netflix and Hulu, have similar characteristics to those on YouTube.

4.2 Using Storyboards to Estimate Backlight Scaling Power Consumption

Since storyboards only contain thumbnail images for frames selected at fixed intervals, they cannot provide exact per-frame maximum pixel luminance information needed by our dynamic programming algorithm. To use storyboard information effectively, we adapt our dynamic programming algorithm (Figure 2) to estimate power consumption over a full set of storyboard images.

This adaptation is straightforward. Rather than running the dynamic programming algorithm on the maximum luminance values

of each frame in a batch of video frames, we run the algorithm for all storyboard thumbnail images of the video. In addition, to compute the value of $Y_f^{(max)}$, the maximum luminance at frame f, rather than simply extracting the maximum luminance of the storyboard image at frame f, we take the maximum over the storyboard image at frame f as well as the storyboard images before and after it. This additional step allows us to minimize distortion under incomplete knowledge of both the full frame resolution and from subsampling frame at longer intervals. Once backlight scaling values are computed for each storyboard image, our algorithm expands the period of constant backlight level to cover all original video frames nearest in time to the storyboard image.

Meanwhile, we also take into consideration the minimum length the backlight must stay at the same level, d_{min}, and the maximum inter-frame backlight scaling variation constraint, Δ_b. We ensure that the change in backlight scaling levels assigned to two consecutive storyboard images is small enough such that the backlight can be scaled up or down to the desired level in time without violating the constraints.

To estimate the power savings, we first create a linear model relating backlight levels to the amount of power consumed by the display. Because the model is linear, the display power consumption over the full length of the video can be estimated by first computing the dynamic programming recurrence giving the sum of backlight scaling values over the entire video, then computing the power from the model associated with this sum. If the power that can be saved at the display is greater than the power consumed by the GPU, positive overall power savings is predicted.

Given that mobile video streaming applications today, such as YouTube, Netflix, and Hulu, already download storyboards before video playback starts, using storyboard data to estimate the power consumption of our online backlight scaling algorithm can be easily incorporated into video applications in real-world situations without incurring additional network transmission overhead. On the other hand, since a storyboard cannot provide complete luminance information of all frames in a video, we need to be careful about relying on the storyboard prediction for deciding whether to use the GoCAD scheme. We discuss how well storyboard predicts power savings in Section 6.3.

5. GoCAD IMPLEMENTATION

We implement the GoCAD power saving mechanism in an Android video streaming application. Figure 4 shows the organization of the application. The Scanning Module is connected to the Open MAX (OMX) [3] layer, which supplies raw video frames that are hardware-decoded. It selects beacon frames at fixed intervals for analysis. This interval is determined by the capacity of the GPU hardware. While selecting all frames as beacon frames for analysis can provide the dynamic programming algorithm with fine-grained information to maximize power savings and minimize distortion, the GPU may not be powerful enough to process all frames in time for playback. Therefore, we must find a balance between power saving and in time playback.

For selected beacon frames, the Scanning Module uses the RenderScript framework [4] to interface with the GPU to analyze the pixel luminance histogram of each beacon frame. Collecting luminance histogram information instead of simply the maximum pixel luminance per frame allows us to compute a backlight scaling schedule that results in greater power savings by allowing a small number of pixels to be displayed with an observed luminance lower than the original luminance. To do so, we can choose the $(100-d)$ percentile luminance of every frame as input to the dynamic programming algorithm, which produces **up to $d\%$ pixel distortion** per frame.

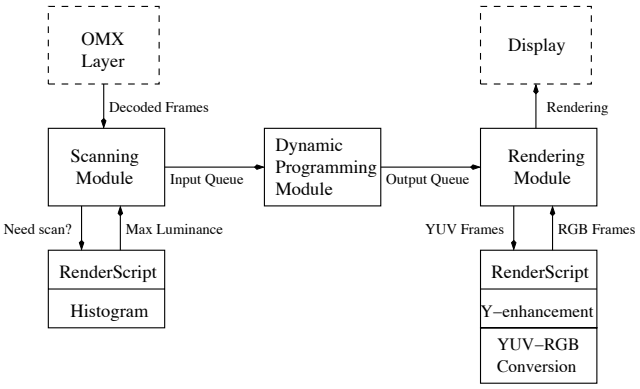


Figure 4: The organization of our GoCAD-enabled Android application.

The luminance information of batches of beacon frames are sent as input to the Dynamic Programming Module, which outputs the backlight scaling schedule for these frames. Next, all decoded frames (in YUV format) together with the backlight scaling schedule are sent to the Rendering Module. The Rendering Module sets the backlight level according to the computed schedule. Before a frame can be rendered, the Rendering Module uses the RenderScript framework to interface with the GPU to perform luminance compensation by increasing the Y component of each pixel in the frame. Finally, it converts the YUV representation to RGB color space using Y', U, and V for rendering.

6. PERFORMANCE EVALUATION

To evaluate the performance of GoCAD during video streaming playback, we install our video streaming application on 4 Android devices: one Samsung Galaxy Tab 2 7.0-inch tablet, one Samsung Galaxy Tab 2 10.1-inch tablet, one Nexus 7 tablet, and one Nexus 9 tablet. The two Samsung tablets use the TI OMAP4430 SoC that includes the PowerVR SGX540 GPU. The Nexus 7 tablet uses the Qualcomm Snapdragon S4 Pro SoC with an Adreno 320 GPU. And the Nexus 9 tablet uses the NVIDIA Tegra K1 SoC with a Kepler DX1 GPU. To build the supported backlight level table for these devices, we evenly pick 255 numbers in the range $(0, 1]$.

To test GoCAD, we randomly select 468 YouTube videos using the random prefix sampling method proposed by Zhou et al. [33]. We download all 468 videos and their corresponding storyboards from YouTube for experiments and analysis. Figure 5 shows the setup of our power measurement experiments. To accurately measure the power consumption during video playback, we use a Monsoon power monitor [2] to supply power to our testing devices directly.

We compare the power consumption of our GoCAD scheme with the baseline case where backlight scaling is not enabled. We do not compare our scheme with existing schemes discussed in Section 2 because these schemes either require offline pre-processing of the videos or rely on external computational resources.

6.1 Display and GPU Power Consumption Model

We build a display power consumption model as a function of backlight levels using the Monsoon power monitor. During the measurement, we close other applications and turn off the network interfaces on the mobile devices. We first measure the baseline

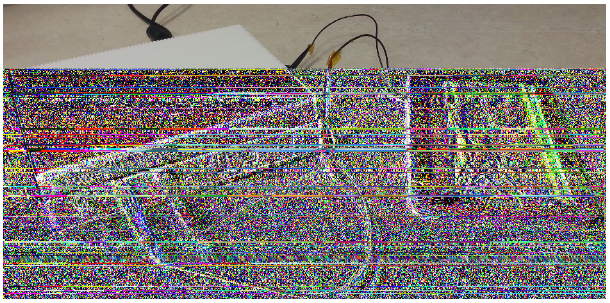
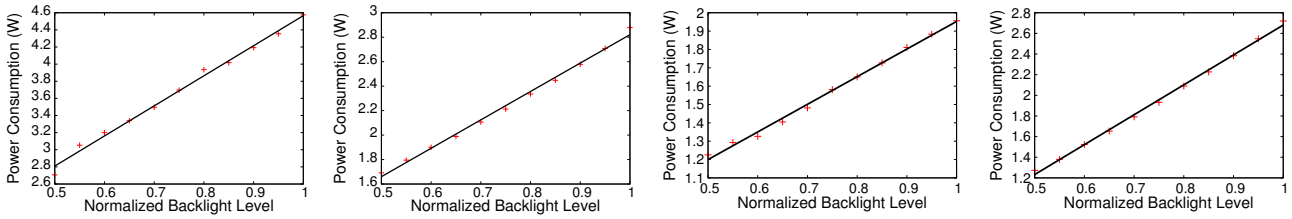


Figure 5: Power measurement setup.

power consumption when the backlight is set to its lowest possible level. Then, we gradually increase the backlight level and repeat the measurement. To maintain reasonable user experience, we restrict the minimum normalized backlight level to 0.5.

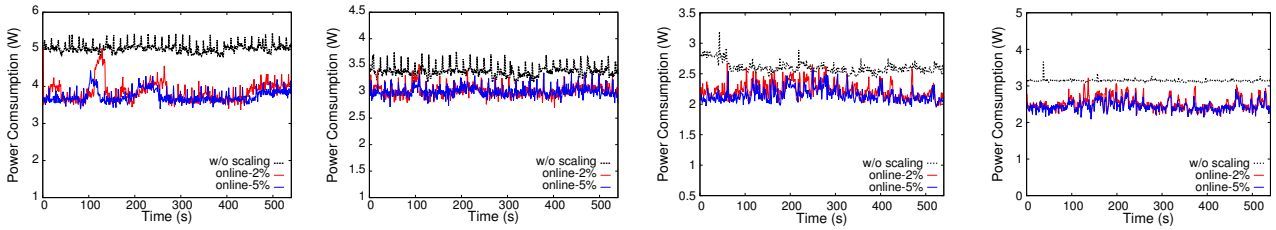
The results are shown in Figure 6. We find that the display power consumption of the 10.1-inch Galaxy Tab can be best represented with the following linear model: $p = w_1 \times b + w_2$, where $b \in [0.5, 1]$ is the normalized display backlight level, $w_1 = 3512.7$, and $w_2 = 1053.4$, with $R^2 = 0.993$. The fit is shown in Figure 6(a). To measure the GPU power consumption, we first play a video with the native video player application supplied by Android with the backlight set to the maximum level. We then compare the average power consumption of the native player with our video player application when GPU is used to scale pixel luminance by 1.0 (no effect). Results show that when using GPU for luminance compensation to play videos at 30 frames per second, the GPU consumes the power of 578 mW. The parameters for the linear display power consumption models for all 4 devices as well as the GPU power consumption overhead of our application are shown in Table 1.

the (po)25(w0e)-240(set)2TWOal(el)-306(W1)cw0e remains consumptiGPU)-198t setpiGPUGPUa(with)-289(heGPUplicatiormalized)vwn



(a) 10.1-inch Galaxy Tab 2 (b) 7.0-inch Galaxy Tab 2 (c) Nexus 7 (d) Nexus 9

Figure 6: Display power consumption model.



(a) 10.1-inch Galaxy Tab 2 (b) 7.0-inch Galaxy Tab 2 (c) Nexus 7 (d) Nexus 9

Figure 7: Measured results: power consumed by the entire device with online backlight scaling under different settings.

for 95 (22.4%), 162 (34.6%), 244 (52.1%) videos, respectively. For Nexus 9, GoCAD can save power for 108 (23.1%), 188 (40.2%), 273 (58.3%) videos if up to 2%, 5%, and 10% pixel distortion can be tolerated, respectively.

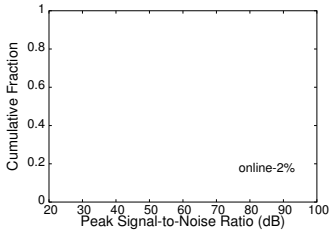
For backlight scaling schedule that may produce pixel distortion, we examine the quality achieved by calculating the peak signal-to-noise ratio (PSNR) and structural similarity (SSIM) [30] between the rendered video and the original, non-backlight-scaled video. Figure 9 shows that for all videos, their PSNR values are always greater than 24 dB if up to 2% or 5% pixel distortion is allowed, and always greater than 20 dB if up to 10% pixel distortion is allowed. In addition, for 80% videos, their PSNR values are greater than 40 dB, 33 dB, and 28 dB when rendered with up to 2%, 5%, and 10% pixel distortion, indicating good rendered frame quality.

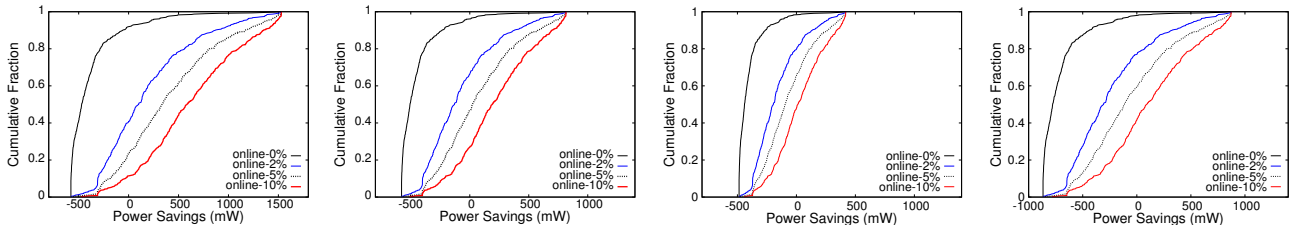
Our results also show that even with up to 2%, 5%, and 10% pixel distortion, the SSIM values are always greater than 0.998, while existing schemes [17, 20] that choose not to perform the computationally intensive luminance compensation step can only yield SSIM values around 0.9. This confirms that with GPU-assisted luminance compensation, our GoCAD mechanism can achieve much better video quality than these schemes.

6.3 Performance of Storyboard-based Prediction

Because storyboard does not contain complete information about pixel luminance of all frames in a video, we next study how well storyboard can predict if power can be saved via our GoCAD scheme.

For 468 selected we our namic programming algorithm using `L0`, `L1`, and `L2` (when available) storyboard images. For example, to predict how much power can be saved by GoCAD if we allow up to 10% pixel distortion per frame, we use the 90-percentile pixel luminance of every storyboard image as input to the adapted dynamic programming algorithm. We then calculate the corresponding storyboard-predicted power savings for each storyboard level using the display and GPU power consumption model. We assign positive label to a video if GoCAD can save power and negative label otherwise. We evalu-





(a) 10.1-inch Galaxy Tab 2 (b) 7.0-inch Galaxy Tab 2 (c) Nexus 7 (d) Nexus 9

Figure 8: Estimated amount of power savings (accounting for GPU power consumption) with online backlight scaling under different settings.

7. RELATED WORK

To effectively save the energy consumption on mobile devices during Internet streaming, plenty of existing research has proposed to reduce the power consumption during data receiving, video decoding, and video rendering. To reduce power consumed by the wireless network interface card (WNIC) (WiFi or cellular) for receiving streaming data, many schemes have been proposed to put the WNIC into the low power sleep mode for as long as possible. For example, BlueStreaming saves power by intelligently offloading frequent control traffic in P2P streaming to the low power Bluetooth interface, allowing the WiFi interface to enter sleep mode [21]. GreenTube is a system that carefully schedules video downloading based on expected user viewing time and current available bandwidth [18]. Instead of using a user's local view history to calculate the expected viewing time as GreenTube, Hoque et al. propose eSchedule that uses crowdsourced global video watching statistics for prediction [15]. To reduce power consumption during video decoding, content providers transcode videos to the resolution and format that best fit the decoding capabilities of mobile devices [19].

To reduce power consumption of mobile displays, researchers have focused on two types of displays: OLED and LCD. Since the power consumption of OLED depends on the color of the pixels displayed, existing work mainly focuses on reducing power by changing the color of displayed content. For example, Chameleon is a mobile web browser that transforms a webpage's color scheme to a more power-efficient one on OLED displays [14]. FOCUS saves power by darkening portions of the screen that is outside of the user's current region of interest (ROI) [28]. These schemes, however, significantly change the visual content and are not applicable to videos. Shin et al. propose to apply dynamic voltage scaling (DVS) on OLED displays [26]. However, special hardware is required for this scheme to work. For LCD displays, as the backlight consumes the majority of energy, backlight scaling has been used to save power in many mobile video applications. For example, Anand et al. propose to save display power consumption in mobile gaming while maintaining perceived gameplay quality by using backlight scaling with a tone mapping function available from graphics engines [7]. Xiao et al. use backlight scaling to save power in mobile video conferencing and leverage the GPU to perform pixel luminance enhancement [31]. Hsiu et al. and Lin et al. propose to use backlight scaling in mobile video streaming by assigning a critical backlight level to every frame in a video [17, 20]. However, applying backlight scaling without luminance compensation can adversely affect user-perceived video streaming quality. Different from these schemes, GoCAD leverages the GPU to efficiently perform luminance compensation and can achieve much better video streaming quality. Focusing on users' quality of experience (QoE), Yan et al. conducted a field study to investigate how reduced backlight affects QoE [32]. However, the authors did not adopt luminance compensation in their QoE study.

8. CONCLUSION

Internet mobile streaming services are boosting due to the pervasive adoption of all kinds of mobile devices. However, Internet mobile streaming is power-hungry while mobile devices are fundamentally constrained by the limited battery power. To reduce the battery power consumption and prolong the battery lifetime, in this work, we have designed and implemented a GPU-assisted Online Content-Adaptive Display (GoCAD) power saving mechanism for reducing display power consumption on mobile devices in receiving Internet streaming services. GoCAD improves on previous backlight scaling schemes by using the GPU instead of the CPU for practical online luminance adjustment and a backlight scaling algorithm that operates in an online manner over small batches of video frames. When a storyboard is available, GoCAD can use it to estimate the amount of power saved by the backlight scaling algorithm. The GoCAD scheme's efficiency is thus further improved as backlight scaling is applied only on videos where power savings from backlight scaling outweighs the additional GPU power consumption needed to run the scheme. We have implemented GoCAD on Android and experimented with more than 460 randomly selected YouTube videos. The results show that when backlight scaling is computed online, GoCAD can effectively reduce power consumption on mobile devices in receiving streaming services.

9. ACKNOWLEDGEMENT

We appreciate constructive comments from anonymous referees. The work is partially supported by High-Tech Research and Development Program of China ("863 – China Cloud" Major Program) under grant 2015AA01A201, by China NSF under grants 61432002 (State Key Program) and 61471217, by CCF-Tencent Open Fund under grant IAGR20150101, and by NSF under grants CNS-1117300 and CNS-1524462.

10. REFERENCES

[1] Color Conversion. `http://www.equasys.de/colorconversion.html`.

[2] Monsoon Power Monitor. `http://www.msoon.com/LabEquipment/PowerMonitor/`.

[3] OpenMAX. `https://www.khronos.org/openmax/`.

[4] RenderScript. `http://developer.android.com/guide/topics/renderscript/compute.html`.

[5] YouTube player adds Netflix-like storyboard feature for easier navigation. `http://venturebeat.com/2012/01/13/youtube-player-storyboard-thumbnail-feature/`.

[6] YouTube Statistics. http://www.youtube.com/yt/press/statistics.html.

[7] B. Anand, K. Thirugnanam, J. Sebastian, P. G. Kannan, A. L. Ananda, M. C. Chan, and R. K. Balan. Adaptive display power management for mobile games. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 57–70. ACM, 2011.

[8] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, pages 21–21, 2010.

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

[10] L. Cheng, S. Mohapatra, M. El Zarki, N. Dutt, and N. Venkatasubramanian. Quality-based backlight optimization for video playback on handheld devices. *Adv. MultiMedia*, 2007(1):4–4, Jan. 2007.

[11] W.-C. Cheng and M. Pedram. Power minimization in a backlit tft-lcd display by concurrent brightness and contrast scaling. *Consumer Electronics, IEEE Transactions on*, 50(1):25–32, 2004.

[12] H. Cho and O.-K. Kwon. A backlight dimming algorithm for low power and high image quality lcd applications. *Consumer Electronics, IEEE Transactions on*, 55(2):839–844, 2009.

[13] I. Choi, H. Shim, and N. Chang. Low-power color tft lcd display for hand-held embedded systems. In *Low Power Electronics and Design, 2002. ISLPED '02. Proceedings of the 2002 International Symposium on*, pages 112–117, 2002.

[14] M. Dong and L. Zhong. Chameleon: a color-adaptive web browser for mobile oled displays. *Mobile Computing, IEEE Transactions on*, 11(5):724–738, 2012.

[15] 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. ACM, 2013.

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

[17] P.-C. Hsiu, C.-H. Lin, and C.-K. Hsieh. Dynamic backlight scaling optimization for mobile streaming applications. In *Proceedings of the 17th IEEE/ACM international symposium on low-power electronics and design*, pages 309–314, 2011.

[18] 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. ACM, 2012.

[19] Z. Li, Y. Huang, G. Liu, F. Wang, Z.-L. Zhang, and Y. Dai. Cloud transcoder: Bridging the format and resolution gap between internet videos and mobile devices. In *Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video*, pages 33–38. ACM, 2012.

[20] C.-H. Lin, P.-C. Hsiu, and C.-K. Hsieh. Dynamic backlight scaling optimization: A cloud-based energy-saving service for mobile streaming applications. *Computers, IEEE Transactions on*, 63(2), Feb 2014.

[21] Y. Liu, F. Li, L. Guo, Y. Guo, and S. Chen. Bluestreaming: towards power-efficient internet p2p streaming to mobile devices. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 193–202. ACM, 2011.

[22] Y. Liu, M. Xiao, M. Zhang, X. Li, M. Dong, Z. Ma, Z. Li, and S. Chen. Content-adaptive display power saving in internet mobile streaming. In *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 1–6. ACM, 2015.

[23] S. Pasricha, S. Mohapatra, M. Luthra, N. D. Dutt, and N. Venkatasubramanian. Reducing backlight power consumption for streaming video applications on mobile handheld devices. In *ESTImedia*, pages 11–17, 2003.

[24] A. Rao, A. Legout, Y.-s. Lim, D. Towsley, C. Barakat, and W. Dabbous. Network characteristics of video streaming traffic. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, page 25. ACM, 2011.

[25] M. Ruggiero, A. Bartolini, and L. Benini. Dbs4video: dynamic luminance backlight scaling based on multi-histogram frame characterization for video streaming application. In *Proceedings of the 8th ACM international conference on Embedded software*, pages 109–118. ACM, 2008.

[26] D. Shin, Y. Kim, N. Chang, and M. Pedram. Dynamic voltage scaling of oled displays. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 53–58. IEEE, 2011.

[27] T. Simunic, L. Benini, P. Glynn, and G. De Micheli. Event-driven Power Management. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 20(7):840–857, 2001.

[28] K. W. Tan, T. Okoshi, A. Misra, and R. K. Balan. Focus: a usable & effective approach to oled display power management. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pages 573–582. ACM, 2013.

[29] P.-S. Tsai, C.-K. Liang, T.-H. Huang, and H. Chen. Image enhancement for backlight-scaled tft-lcd displays. *Circuits and Systems for Video Technology, IEEE Transactions on*, 19(4):574–583, 2009.

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

[31] M. Xiao, Y. Liu, L. Guo, and S. Chen. Reducing display power consumption for real-time video calls on mobile devices. In *Low Power Electronics and Design (ISLPED), 2015 IEEE/ACM International Symposium on*, pages 285–290. IEEE, 2015.

[32] Z. Yan, Q. Liu, T. Zhang, and C. W. Chen. Exploring qoe for power efficiency: A field study on mobile videos with lcd displays. In *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference*, pages 431–440. ACM, 2015.

[33] J. Zhou, Y. Li, V. K. Adhikari, and Z.-L. Zhang. Counting youtube videos via random prefix sampling. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 371–380. ACM, 2011.