Content-Adaptive Display Power Saving for Internet Video Applications on Mobile Devices

YAO LIU, SUNY Binghamton
MENGBAI XIAO and MING ZHANG, George Mason University
XIN LI, Samsung Telecommunications America
MIAN DONG, AT International, Inc.
ZHAN MA, Nanjing University
ZHENHUA LI, Tsinghua University
LEI GUO, The Ohio State University
SONGQING CHEN, George Mason University

Backlight scaling is a technique proposed to reduce the display panel power consumption by strategically dimming the backlight. However, for mobile video applications, a computationally intensive luminance compensation step must be performed in combination with backlight scaling to maintain the perceived appearance of video frames. This step, if done by the Central Processing Unit (CPU), could easily offset the power savings via backlight dimming. Furthermore, computing the backlight scaling values requires per-frame luminance information, which is typically too energy intensive to compute on mobile devices.

In this article, we propose Content-Adaptive Display (CAD) for two typical Internet mobile video applications: video streaming and real-time video communication. CAD uses the mobile device's Graphics Processing Unit (GPU) rather than the CPU to perform luminance compensation at reduced power consumption. For video streaming where video frames are available in advance, we compute the backlight scaling schedule using a more efficient dynamic programming algorithm than existing work. For real-time video communication where video frames are generated on the fly, we propose a greedy algorithm to determine the backlight scaling at runtime. We implement CAD in one video streaming application and one real-time video call application on the Android platform and use a Monsoon power meter to measure the real power consumption. Experiment results show that CAD can save more than 10% overall power consumption for up to 55.7% videos during video streaming and up to 31.0% overall power consumption in real-time video calls.

CCS Concepts: • **Information systems → Multimedia streaming;**

Additional Key Words and Phrases: Internet mobile streaming, real-time video communication, LCD, display, power saving, backlight scaling

We appreciate constructive comments from anonymous reviewers. The work is partially supported by the High-Tech Research and Development Program of China ("863 - China Cloud" Major Program) under grant 2015AA01A201, by the National Natural Science Foundation of China (NSFC) under grants 61432002 (State Key Program), 61632020 (State Key Program) and 61471217, by the CCF-Tencent Open Fund under grants IAGR20150101 and IAGR20160101, and by the NSF under grant CNS-1524462. Preliminary versions of this manuscript are published in the proceedings of NOSSDAV 2015 and ISLPED 2015 [Liu et al. 2015; Xiao et al. 2015].

Authors' addresses: Y. Liu, Dept. of Computer Science, Binghamton University, State University of New York; M. Xiao, Dept. of Computer Science, George Mason University; M. Zhang, Dept. of Computer Science, George Mason University; X. Li, Samsung Telecommunications America; M. Dong, AT International, Inc. Z. Ma, Nanjing University; Z. Li, Tsinghua University; L. Guo, Dept. of Computer Science and Engineering, The Ohio State University; S. Chen, Dept. of Computer Science, George Mason University.

DOI: http://dx.doi.org/10.1145/2996461

ACM Reference Format:
Yao Liu, Mengbai Xiao, Ming Zhang, Xin Li, Mian Dong, Zhan Ma, Zhenhua Li, Lei Guo, and Songqing Chen. 2016. Content-adaptive display power saving for internet video applications on mobile devices. ACM Trans. Multimedia Comput. Commun. Appl. 12, 5s, Article 84 (November 2016), 19 pages.
DOI: http://dx.doi.org/10.1145/2996461

1. INTRODUCTION

With the ever-increasing network and mobility support, video applications, such as video streaming and real-time video communication, are gaining increased popularity among mobile users. However, these video applications are limited by the battery capacity on mobile devices. According to Carroll and Heiser [2010], the display subsystem is responsible for about 38% to 68% of the total power consumption during video playback. Unlike the wireless network interface cards, whose power consumption can be reduced by putting them into low power sleep mode for as long as possible, mobile display panels cannot be put into sleep mode and must be kept active while the video application is running.

To save power consumed by Liquid-Crystal Displays (LCD), backlight scaling has been proposed. This technique reduces power consumption by dimming the display backlight. Meanwhile, the brightness perceived by the human eye is maintained by increasing the affected image's luminance. By simultaneously scaling the backlight and increasing the luminance of the image, the original image can be rendered with little distortion.

However, implementing the backlight scaling strategy with luminance compensation to save power in Internet mobile video applications is challenging. First, backlight scaling values must be determined, subject to the following constraints: (i) To maintain image fidelity, the backlight level cannot be lower than a point determined by the brightness characteristics of an image. Any algorithm used to enforce this constraint must compute the maximum pixel luminance of every frame. This computation can be both time and energy intensive. (ii) It is infeasible to adjust the backlight level for every frame because the display hardware takes time to perform the adjustment. (iii) Large inter-frame backlight variation can cause flickering effects, constraining the range of such adjustments. Second, luminance compensation must be performed for every pixel in every video frame. Thus, increasing the luminance for the entire frame of a high resolution video on a high resolution display could consume tens of millions of Central Processing Unit (CPU) cycles. While a powerful CPU could complete this task in real time, the corresponding power consumption overhead could negate the power saved by dimming the backlight. Therefore, previous studies often suggested these tasks should be performed offline using extra computing resources [Cheng et al. 2007; Pasricha et al. 2003], making backlight scaling hardly practical.

In this article, we propose Content-Adaptive Display (CAD) power-saving mechanisms for reducing display power consumption for two typical Internet mobile video applications: video streaming and real-time video communication. Instead of using the CPU, CAD uses the Open Graphics Library for Embedded Systems (OpenGL ES) application program interface (API) to interface with the Graphics Processing Unit (GPU) on mobile devices to adjust pixel luminance for video applications, allowing a net power savings in combination with the backlight scaling strategy. For applications such as video streaming, where videos are available in advance, we propose a dynamic programming approach for determining backlight scaling assignments, which has a lower complexity than existing algorithms. The per-frame luminance information required by the dynamic programming algorithm is computed offline using external computing resources. For real-time video applications, such as video call, where video frames are generated on the fly, we propose a greedy algorithm that computes the backlight scaling level in an online manner.

We implement CAD in both a mobile video player application and a real-time video call application on the Android platform. The video player application sets the backlight according to the backlight scaling assignment computed externally and simultaneously compensates the brightness by increasing pixel luminance through GPU computations during video playback. We installed the video player application on three Android devices. A Monsoon power meter is used for real measurements of power consumption. Results show that on a randomly selected set of 1,000 YouTube videos at the resolution of 720P, CAD can achieve more than 10% overall power savings for up to 55.7% of the videos. The real-time video call application runs on all parties in the call. Receivers of video frames determine the backlight level using the greedy algorithm based on the per-frame pixel luminance information calculated by the senders and perform luminance compensation based on the selected backlight level. Experiment results using the Monsoon power monitor show that CAD can save up to 31.0% power consumption on average. Our evaluations of the video call quality based on frame rate, user perceived latency, and image fidelity in Peak-Signal-to-Noise Ratio (PSNR) and Structure SIMilarity (SSIM) show that CAD introduces negligible impact on the received video call quality.

The remainder of the article is organized as follows. Section 2 discusses the background. We present the backlight determination algorithms in Section 3 and the system design of our CAD power saving mechanism in Section 4. Section 5 describes the implementation details of our Android applications, and Section 6 discusses our evaluation results. We discuss related work in Section 7 and make concluding remarks in Section 8.

2. BACKGROUND

A visible image on an LCD display is produced by both backlight and the LCD panel which stores pixel color information. The perceptual luminance is the backlight intensity compensated by the pixels. Previous studies have pointed out that the backlight of an LCD display dominates the energy consumption of the display subsystem [Carroll and Heiser 2010; Simunic et al. 2001]. Therefore, power can be saved if we reduce the backlight intensity level of the LCD display.

Backlight scaling is a technique that exploits this characteristic. Figure 1 sketches the high-level idea. The power consumption of displaying an image can be reduced via dimming the backlight. If nothing else is done, it will lead to image distortion, i.e., a darker version of this image, which is normally defined as the resemblance between the original video image and the backlight-scaled image [Cheng et al. 2007; Tsai et al. 2009]. One way to resolve the problem is by simultaneously scaling the backlight level and increasing the luminance of every pixel [Chang et al. 2004; Cheng and Pedram 2004; Choi et al. 2002; Pasricha et al. 2003]. In this way, image fidelity can be preserved, and increasing the pixel luminance does not increase the power consumption of the display.

While the idea of increasing the luminance of video frames at runtime to compensate for a dimmed backlight is intuitive, a significant challenge arises when attempting to adjust the luminance of every pixel in every video frame in an energy efficient manner. Many existing techniques rely on the CPU to perform per-pixel manipulation [Cheng and Pedram 2004; Choi et al. 2002], but this may significantly offset the power savings achieved via backlight dimming. To avoid the computationally intensive step on the mobile device, Hsiu et al. [2011] and Lin et al. [2014] propose not to perform luminance compensation, but instead, simply to choose a critical backlight level for each frame as the scaling constraint. This, however, can lead to a large amount of distortion of the displayed frames. Instead of using the CPU, Ruggiero et al. [2008] consider a special multimedia processor, the Freescale i.MX31, which has an Image Process Unit (IPU) that can be used to perform the luminance adjustment task. Rather than a

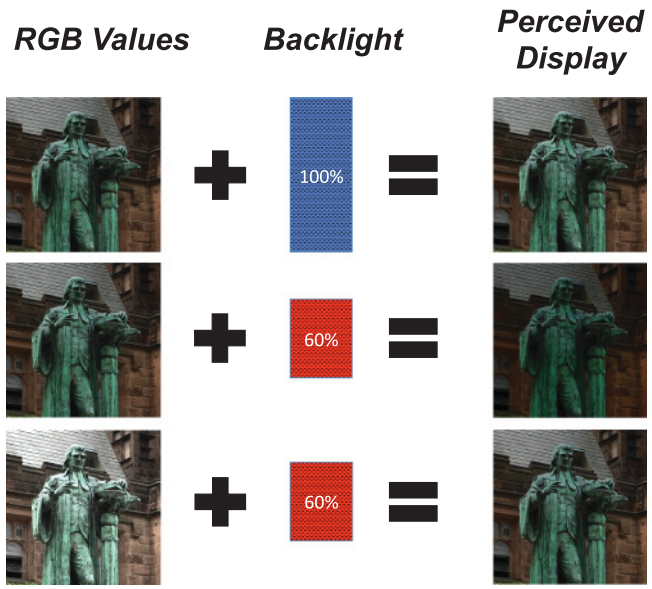


Fig. 1. Backlight scaling on LCD screen.

client-side solution, Pasricha et al. [2003] and Cheng et al. [2007] propose to migrate the computation to an intermediate proxy server that computes backlight scaling values and transcodes the original video to a luminance-adjusted version.

Compared to existing work, our proposed solution can save display consumption while maintaining image fidelity without relying on either a specialized processor for pixel luminance compensation or intermediate servers for video transcoding. Instead, CAD uses the GPU to perform the computation.

3. BACKLIGHT DETERMINATION ALGORITHMS

In this section, we first discuss a set of constraints associated with backlight scaling. Then we present an offline dynamic programming algorithm and an online greedy algorithm for computing backlight scaling levels in CAD.

3.1. Display Power Saving Constraints

CAD is based on the *Backlight Scaling* method [Choi et al. 2002], which saves power consumption by dimming the backlight level. Suppose the display backlight levels lie within the range $(0, 1]$, where a value of 0 indicates that the backlight is off, and a value of 1 indicates that it is set to maximum brightness. With backlight scaling, for every frame in the video, we would like to set the display backlight level to $b \in (0, 1]$. To avoid fidelity loss under reduced backlight levels, we adopt *Luminance Compensation* [Cheng et al. 2007; Tsai et al. 2009]. We increase the luminance of every pixel in the frame by a factor of $\frac{1}{b}$ (i.e., increasing the Y component of the YUV representation of every pixel). In this way, the *observed* pixel luminance, Y', is adjusted to a value that is the same as that of the original frame's: $b \times Y' = b \times Y \times \frac{1}{b} = Y$. With joint backlight scaling and luminance compensation, display power consumption can be reduced without any observable fidelity loss.

For every frame in a video, the backlight scaling value must be determined according to the following three constraints:

Distortion Constraint. The Y component of a pixel can not be scaled to higher than its maximum value, 255. Therefore, if b is chosen to be a value such that $Y \times \frac{1}{b} > 255$, then the observed luminance of the adjusted pixel with the reduced backlight 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" [Cho and Kwon 2009]. To avoid creating these "clipping artifacts," we must limit any backlight adjustments to $b \geq \frac{Y_{max}}{255}$, where Y_{max} is the maximum pixel luminance in the frame. This distortion-based constraint gives rise to a *lower bound* on the adjusted backlight level for every frame in the video.

User Experience Constraint. While setting the backlight level of every frame to its lowest possible value subject to the distortion constraint can maximize power savings, users could experience inter-frame brightness distortion, often perceived as flickering, if the variation in backlight scaling levels between two consecutive frames is too big. Therefore, we need to *limit* the brightness variation between two consecutive frames to reduce this flickering effect. We denote this constraint as $b_t \times (1 - \Delta_b) \leq b_{t+1} \leq b_t \times (1 + \Delta_b)$, where b_t denotes the backlight level at frame t, and Δ_b is the ratio of change limit in backlight level.

Hardware Constraint. The display hardware requires a minimum amount of time to apply any brightness adjustments. Therefore, it is impossible to adjust the backlight level promptly for every video frame. Instead, we must specify a *minimum interval* (in terms of numbers of frames), ℓ_{min}, where the backlight level must remain constant for all frames.

3.2. Compute Backlight Scaling Offline Using Dynamic Programming

Given the constraints described above, we have designed a dynamic programming algorithm to compute backlight scaling levels for maximized power savings. Given a sequence of maximum pixel luminance values of video frames, our dynamic programming algorithm computes the values of $B(t, b)$, the minimum cumulative backlight levels ending at frame t with the backlight level of frame t set to b. $B(t, b)$ can be computed by the following recurrence:

$$B(t, b) = \min_{t', b'}(b \times (t - t') + B(t', b')). \tag{1}$$

Given the hardware constraint, the backlight must be constant for an interval of at least ℓ_{min} frames. In the recurrence above, t' is the last frame of the constant backlight interval that immediately precedes the interval which frame t belongs to, subject to the constraint $t' \leq t - \ell_{min}$. b' is the backlight level of frame t', i.e., the backlight light level of the previous interval, and is subject to the Δ_b constraint discussed above. This 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 Section 6.

We also add an additional constraint, ℓ_{max}, specifying the maximum length of an interval. This decision is motivated by two considerations. First, setting the algorithm to use fixed-length intervals of constant brightness produces suboptimal behavior. This is because large changes in maximum luminance exist at many positions within a video. These change points are unlikely to align with boundaries of any preset fixed-length interval. As a result, a large number of fixed-length intervals would cross changepoint boundaries, leaving a portion of these intervals assigned to higher backlight levels than necessary. On the other hand, if we were to consider all possible lengths of constant-brightness intervals, the algorithm would be optimal but computation would have a complexity of $O(T^2 \times |b|^2)$, where T is the total number of frames in the video, and $|b|$ is

```
▷ On input lum, an array of length T, indicating maximum luminance values for
each frame of a video containing a total of T frames, compute output, an array
containing minimum backlight brightness values subject to constraints.
▷ Compute the recurrence
for t in [ℓ_min, T] do
    for t' in [t − ℓ_max, t − ℓ_min] do
        for b in [max(lum[t' + 1 : t])/255, 1] do
            for b' in [b/(1 + Δ_b), b/(1 − Δ_b)] do
                if b × (t − t') + B[t', b'] < B[t, b] then
                    B[t, b] = b × (t − t') + B[t', b']
                    H[t, b] = (t', b')
▷ Backtracking phase
output = array(T)
t = T
b = arg min_{b'} B[t, b']
while t >= 0 do
    (t', b') = H[t, b]
    output[t' + 1 : t] = b
return output
```

Fig. 2. Computing the backlight scaling data with $O(T \times (\ell_{max} - \ell_{min} + 1) \times |b|^2)$ complexity, where $|b|$ indicates the number of possible values of b. In the algorithm, $B[t, b]$ indicates the minimum cumulative backlight levels ending at frame t with the backlight level of frame t set to b, T indicates the number of frames in the video, ℓ_{min} is the shortest allowed constant backlight interval (in frames) and ℓ_{max} is the longest, $lum[t' : t]$ indicates a subarray of the input lum, from video frame t' to t, Δ_b encodes the constraint specifying the allowable ratios of adjacent brightness intervals, and $H[t, b]$ is a history array that records how the minimum backlight array $B[t, b]$ was constructed.

the number of possible values of b,[1] which is unsuitable for our application. Motivated by the fact that long intervals of constant brightness can be expressed by concatenating shorter intervals, we can choose constant brightness intervals whose length is not fixed, but lie within a small range of values (i.e., between ℓ_{min} and ℓ_{max}). This allows regions of constant brightness intervals to align more closely with a video's luminance profile, thus achieving near-optimal total brightness levels. Therefore, the ℓ_{max} constraint can reduce the complexity of our algorithm at the cost of only a minimal increase in total brightness over the course of video frame rendering.

Our algorithm thus consists of a forward step where the values of $B(t, b)$ are computed and a backward step where the values of b_t, the backlight value at video frame t, are recovered. Pseudocode to compute the dynamic programming recurrence is shown in Figure 2.

3.3. Determine Backlight Level Online Using Greedy Algorithm

While a dynamic programming algorithm can maximize power savings while satisfying all constraints, it requires all frames of a video be available in advance. This can work for video streaming applications. However, in real-time video applications, such as video calls, frames are generated on the fly. Using dynamic programming to determine backlight level would require waiting for enough frames to be buffered (and thus, calculate the optimal backlight levels), inevitably increasing the user-perceived delay.

[1]This is similar to the algorithm proposed by Lin et al. [2014] for determining optimal backlight scaling levels with $O(N^2M^2(N + \ln M + d^2))$ complexity, where N is the number of frames, M is the number of backlight levels, and d is the minimum duration before a backlight level change.

```
▷ On input (t, Y_t^max, b', t'), where b' is the last adjusted backlight level and t' is the
  corresponding frame index, we generate b_t, the backlight level of the t^th frame.
if t = 1 then
    b' ← Y_t^max/255
    t' ← t
    b_t ← b'
    return b_t
if t − t' < l_min then
    return b'
b_t ← Y_t^max/255
if b_t < b' × (1 − Δ_b) then
    b' ← b' × (1 − Δ_b)
else if b_t > b' × (1 + Δ_b) then
    b' ← b' + (1 + Δ_b)
else
    b' ← b_t
b_t ← b'
t' ← t
return b_t
```

Fig. 3. Determining backlight level using greedy algorithm.

Given the stringent timing requirement in video calls, it is not possible to satisfy all three constraints. Therefore, we propose a greedy algorithm that attempts to relax the distortion constraint. This algorithm determines the current backlight level only based on the latest information from the last frame. In this way, little latency is introduced into the system. We expect our scheme will not lead to significant distortion based on the intuition that few scene changes are likely to exist in video call sessions.

The pseudocode of our greedy algorithm is shown in Figure 3. The backlight level of frame t, b_t, only depends on b'; the backlight level of the last constant backlight interval, t'; the last frame of the last constant backlight interval; and Y_t^{max}, the maximum luminance of frame t. During the adjustment, b_t still has to conform to the user experience constraint and the hardware constraint. Distortion may occur if there is significant change in Y_t^{max}, and b_t can not be adjusted to satisfy the distortion constraint. Assuming that there are no frequent scene changes in video calls, we expect such distortion is rare and will be corrected gradually in the next adjustment operations.

4. SYSTEM DESIGN

In this section, we present the design of two systems that leverage the offline and online CAD for reducing the display power consumption for mobile video streaming and mobile real-time video calls, respectively.

4.1. Mobile Video Streaming

In video streaming, all frames of the video are available beforehand. Therefore, we could compute the backlight scaling information offline using the dynamic programming algorithm and send this information to the mobile device before video streaming starts. The mobile devices can then use such information to dynamically adjust its LCD display backlight level during video playback.

Meanwhile, to compensate for the dimmed display and maintain image fidelity, the mobile device increases the luminance level for every pixel in every frame that is associated with a dimmed backlight level. That is, given the backlight scaling level b_f for rendering frame f, we increase the luminance of every pixel p in frame f from

Y_p, its original luminance, to $Y'_p = Y_p \times \frac{1}{b_f}$, and retain the original U_p and V_p values. This luminance scaling allows the video frame to be rendered on the display without fidelity loss. Note that this luminance compensation has to be performed on all frames that are rendered at a scaled backlight level to maintain consistent contrast levels. This computation must occur as long as the backlight intensity is not set to its default value.

This luminance compensation step requires a floating point data operation for every pixel in every frame. Therefore, the total computation load generated by operations on an entire frame would be infeasible for the CPU to perform, given the time constraints of live video playback. In addition, even if CPUs operating at a higher frequency could accomplish this task in time, the extra power consumed could offset the savings achieved by dimming the backlight. The GPU, on the other hand, is able to perform a large number of tasks in parallel, enabling it to compute adjusted luminance over many pixels in a timely manner. Therefore, we use the OpenGL ES API to interface with mobile device GPUs, enabling these GPUs to perform the luminance compensation task in an online manner.

4.1.1. Deciding Whether to Use Display Adaptation. While using the GPU for luminance compensation, we need to take into account the additional power consumed by the GPU, i.e., we need to determine whether net power savings can be achieved by comparing GPU incurred power consumption overhead with display power that can be saved via backlight scaling. Our approach toward this problem involves three steps: (i) We build models to estimate the power consumption of different devices at different backlight brightness settings during video playback using power measurement results. We also estimate a fixed rate of power consumption of the GPU on these devices based on power measurement results. (ii) We use these models, given the input of a sequence of minimum backlight scaling values calculated using the dynamic programming algorithm, to estimate the display energy consumption during video playback. (iii) We compare these energy consumption values with baseline values to estimate the total energy saved, then compare this saved amount to our estimate of GPU energy consumption. If the value of display energy savings exceeds that of GPU energy consumption, then we can be reasonably confident that backlight scaling will save power when playing the video, and we can apply our scaling method. Although savings may not be possible for all videos, over the course of typical mobile device usage, significant savings could be achieved.

4.2. Mobile Real-Time Video Call

For real-time video communication, since the frames are not available in advance, we employ the online CAD mechanism and let the sender cooperate with the receivers for backlight scaling. Specifically, we organize the backlight scaling tasks into three modules: the *scanning module* at the sender side, the *adjustment module* and the *rendering module* at the receiver side. Figure 4 illustrates the organization and interaction of these three modules.

Sender-Side Scanning and Piggybacking: The scanning module extracts the per-frame pixel luminance information. In real-time video communication, frames are generated by the video capturer, e.g., a physical camera. Since it is impossible to access the whole video content in advance, the scanning module is necessary to generate the luminance histogram for later backlight level determination and luminance compensation. Typically, video call sessions involve multiple $N \geq 2$ participants, and the rendered frame is composed by all received frames (including the frame captured by the receiver itself). While it is possible to conduct the scanning after receiving the video content at the receiver side, in our design, the scanning module is located at the sender side. In

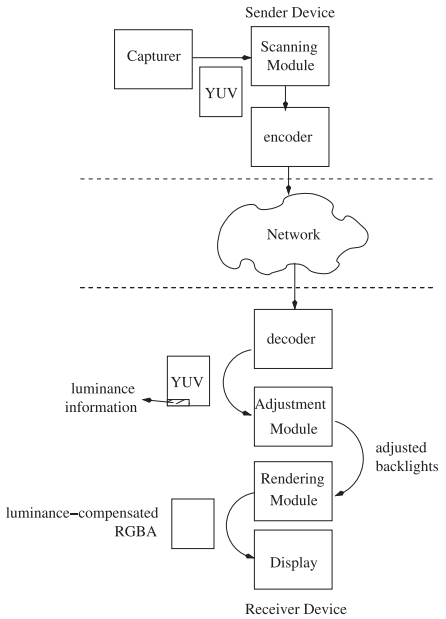


Fig. 4. Architecture and major components of our online greedy display power saving system.

this way, each generated frame will only need to be scanned once. (Otherwise, every receiver needs to conduct scanning for the same frame individually.) However, this brings another problem: how to transmit this luminance information to the receivers if scanning is done at the sender side. Building another channel is possible, but it introduces additional overhead and extra efforts must be made to synchronize the frames and the luminance information. Either of them may not arrive at the receiver on time.

To address this problem, we choose to encode this information with the frame data for more efficient luminance information transmission. The bottom corners are the best candidates for encoding this information. This position is either covered by frames from other participants, or negligible when the frame is resized to a smaller size. Since the encoded luminance information may get lost in the video encoding process or during network transmission, we propose to put this information at multiple positions that are known in advance by all participants. The receiver extracts the information and uses the maximum value among the candidates. It then passes the value to the adjustment module for determining backlight scaling level.

Greedy Backlight Adjustment: The adjustment module is located at the receiver side. This module determines the appropriate backlight levels from the luminance information using the greedy algorithm in Figure 3. After this, the new backlight levels associated with each frame are sent to the rendering module for backlight scaling and pixel compensation.

GPU-assisted Rendering: The rendering module for real-time video call is similar to GPU-assisted rendering in mobile video streaming, as discussed in Section 4.1: adjusting the display backlight level according to the adjustment module and using the commonly available GPU on today's mobile devices to enhance the pixel luminance.

For video calls, since the GPUs are already enabled for resizing and composing the received frames, little power consumption overhead can be expected from including an additional pixel compensation task. Therefore, there is no need to determine whether

display adaptation should be employed. Luminance compensated frames rendered with scaled backlight level allows users to see frames with no distortion from the original version.

5. IMPLEMENTATION

We have implemented two display power saving applications on Android. These two applications use the offline dynamic programming algorithm and the online greedy algorithm, respectively. Next, we present the implementation details of these two Android applications.

5.1. Mobile Video Streaming

5.1.1. Generating Backlight Scaling Data Offline. In mobile video streaming, since the video data is available in advance, we compute the backlight scaling schedule externally by a standalone application that aims to maximize power savings while satisfying the constraints. The external application first decodes the video using the FFmpeg library.[2] Then, it determines the minimum backlight level required by each frame using the dynamic programming algorithm described in Figure 2. We also compute backlight scaling information that results in greater power savings but causes a small number of pixels to be displayed with an observed luminance lower than the original luminance. We refer to this as "pixel distortion." If up to $d\%$ pixel distortion per frame can be tolerated, we can choose the $(100 - d)$ percentile luminance of every frame instead of the maximum luminance as input to the dynamic programming algorithm. The generated backlight scaling data is stored in a file with each value coupled with the corresponding frame index. This data is then sent to the mobile video player application before video playback.

5.1.2. Runtime Backlight Scaling and Luminance Compensation. We implement the concurrent backlight scaling and luminance compensation in an Android video player application. To program GPU to increase pixel luminance, we use OpenGL for Embedded Systems (OpenGL ES).[3] OpenGL ES is a cross-platform 2D 3D graphics API. It is designed for handheld and embedded devices such as mobile phones, PDAs, and video game consoles. Notable platforms supporting OpenGL ES 2.0 include iPhone 3GS and later versions, Android 2.2 and later versions, and WebGL.

In the video player application, we use the `MediaPlayer` provided by Android to decode the video stream. Instead of rendering the decoded video frames onto the default `Surface`, we create a `GLSurfaceView`, wrap it into a `Surface` object, and set the `MediaPlayer` to use this `Surface` as the video data sink so as to divert the video frames into the `GLSurfaceView` object.

To adjust the luminance of all pixels of the frame as a result of an adjusted brightness level, we set up a customized `Renderer` in the `GLSurfaceView` that implements the Vertex Shader and the Fragment Shader. Since pixels have been converted by the Android system from YUV to RGB color space for rendering, the Fragment Shader must convert the color space back to YUV before luminance compensation can be performed. The Vertex Shader just sets up the vertex positions without any transformation. Next, we scale the Y value of the pixel to Y' and convert the YUV representation back to RGB color space using Y', U, and V. By scaling the backlight brightness level and the pixel luminance simultaneously, power consumption is reduced and the image fidelity is maintained.

[2]http://www.ffmpeg.org/.
[3]http://www.khronos.org/opengles/.

5.2. Mobile Real-Time Video Call

We have also implemented a real-time video call application by integrating our online greedy algorithm-based CAD into the WebRTC open source project.[4] We link this component to the WebRTC Android application and use this application for evaluation.

The scanning module is implemented in C++ and is hooked after where the captured frames are generated. In practice, the camera on Android mobile devices produces frames in YUV format, and the scanning module directly extracts the Y component of each pixel in the frame, which represents the luminance. Then, we select the maximum value of Y among all pixels in a frame and encode it into the Y data panel. Because frame information may be lost due to packet loss during network transmission or due to video compression, we encode the maximum luminance value in multiple positions in the Y data panel. After that, the updated frames are sent to the encoder. There is one scanning module in every video call participant.

The adjustment and the rendering modules are both implemented at the JAVA layer. One adjustment module and one rendering module are required for each video stream, including the stream produced by the host itself, for processing and rendering frames of this stream. For example, if a video call involves two devices, there will be two adjustment modules and two rendering modules on each device to process two streams. For each stream, these two modules run on independent threads and are connected by a YUV frame queue.

The adjustment module receives the YUV frames from the decoder and reads the maximum frame luminance information that is encoded in each frame. Since this information is encoded in multiple places and some may be corrupted or lost, we conservatively select the greatest value among all the candidates. We use this maximum luminance information to determine the future backlight level based on our greedy algorithm when rendering this frame. After that, the frame is en-queued to be processed by the rendering module, and a three-tuple (`stream-id`, `frame index`, `adjusted backlight level`) is stored into a global hash table.

Given that a device has to render at least two streams (one from itself and the other from the other end of the call) in a video call, CAD collects the backlight level candidates by using the index of the next frame combined with its stream-id to look up the hash table. The candidate with the greatest value is selected, and backlight scaling and pixel compensation are performed based on the selected value. Video call frames are rendered by invoking the rendering modules sequentially. It fetches the YUV frame from the queue and uses the OpenGL ES shaders to do resizing, luminance compensation, and YUV to RGB conversion. Eventually, the framebuffer generated by the shaders is flushed to the screen.

6. EVALUATION

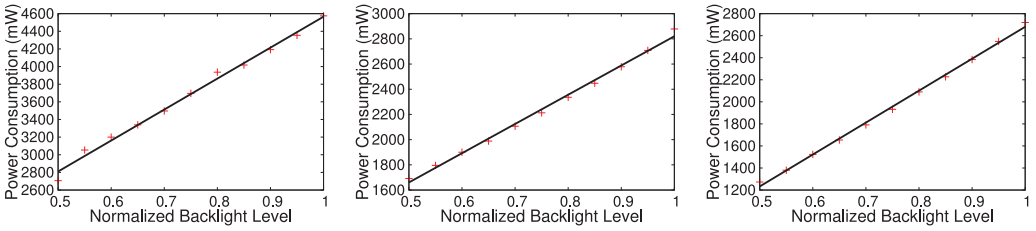
To evaluate our content-adaptive display power saving mechanisms for video applications, we have installed our video player application and video call application on Android devices. To accurately measure the power consumption during video streaming playback and video calls, we use a Monsoon power monitor[5] to supply power directly. Figure 5 shows the setup of our experiments.

6.1. Mobile Video Streaming

To evaluate our offline CAD during video streaming playback, we have installed our video player application on a Samsung Galaxy Tab 2 10.1-inch tablet, a Samsung

[4]http://www.webrtc.org/.
[5]http://www.msoon.com/LabEquipment/PowerMonitor/.

Fig. 5. Power measurement setup.



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

Fig. 6. Display power consumption model.

Galaxy Tab 2 7-inch tablet, and a Nexus 9 tablet. Both Samsung Galaxy Tab devices have the TI OMAP4430 SoC which includes the PowerVR SGX540 GPU, programmable using OpenGL ES 2.0. The Nexus 9 tablet uses the NVIDIA Tegra K1 SoC with a Kepler DX1 GPU. The LCD displays on these devices support 255 backlight levels. To build the supported backlight level table, we evenly pick 255 numbers in the range $(0, 1]$.

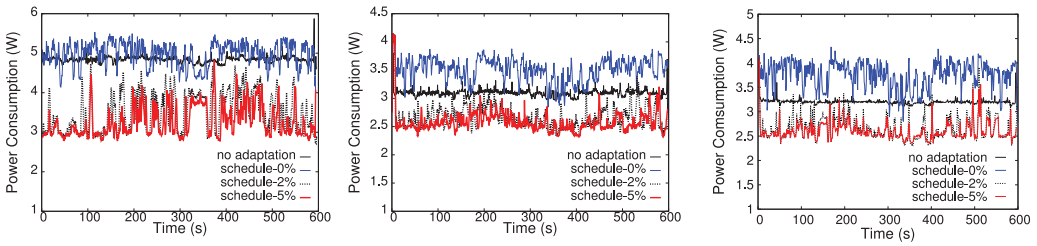
For experiments and analysis, we used 1,000 videos at the quality of 720P. These videos were randomly selected from YouTube using the random prefix sampling method proposed by Zhou et al. [2011].

6.1.1. Display and GPU Power Consumption Models. We build a display power consumption model as a function of brightness. We play a video with the "Gallery" application supplied by Android and measure its power consumption using the Monsoon power monitor. To maintain reasonable user experience, we restrict the minimum normalized backlight luminance level to 0.5. The results are shown in Figure 6. We find that the power consumption of the 10.1-inch Galaxy Tab can be best represented with the following linear model: $y = w_1 \times b + w_2$, where $b \in [0.5, 1]$ is the normalized display backlight level, $w_1 = 3512.7$, and $w_2 = 1053.4$, with $R^2 = 0.9928$. In addition, we also measure the GPU power consumption on the 10.1-inch tablet. We set the backlight to maximum level and compare power consumption when GPU is not used with when GPU is used to scale pixel luminance by 1.0 (no effect). Power measurement results show that when using GPU for luminance compensation to play videos at 30 frames per second, the GPU consumes a constant amount of power, around 578mW. The parameters for the linear display power consumption models for all three devices as well as the GPU power consumption overhead of our application are shown in Table I. We use our display and GPU power models in combination with computed backlight scaling data to decide whether the CAD mechanism should be employed to save power.

6.1.2. Power Savings with Backlight Scaling. To calculate backlight scaling schedule, we first determine appropriate values for parameters required in the algorithm, including ℓ_{min} and Δ_b. We set these parameters to different values and compute the

Table I. Parameters for Display Power Consumption Model: $p = w_1 \times b + w_2$, where $b \in [0.5, 1]$ is the Normalized Display Backlight Level, and the GPU Power Consumption Overhead of Our Application on Three Android Devices

	Display power model			GPU power consumption
	w_1	w_2	R^2	
10.1-in Galaxy Tab 2	3,512.7	1,053.4	0.993	578mW
7.0-in Galaxy Tab 2	2,321.5	499.2	0.995	558mW
Nexus 9	2,897.0	−216.4	0.999	867mW



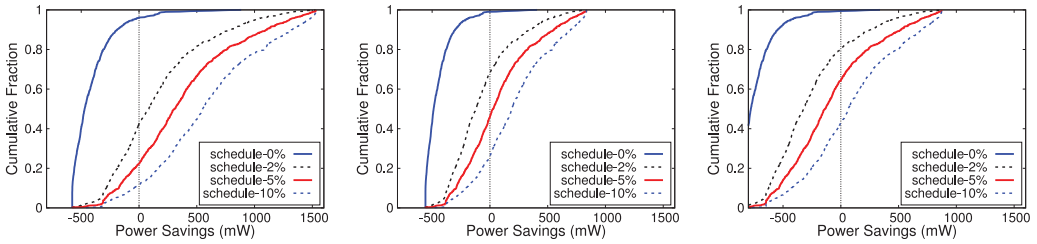
(a) 10.1-inch Galaxy Tab 2 (b) 7-inch Galaxy Tab 2 (c) Nexus 9

Fig. 7. Measured results: power consumed by the entire device with backlight scaling.

corresponding backlight scaling. Five users were asked to watch videos played using our CAD mechanism and report whether they noticed flickering or distortion during playback. We found that when we set Δ_b to 0.06, users did not perceive any flickering. We, thus, use 0.06 for backlight scaling data generation. Similarly, we found that setting $\ell_{min} = 5$ produces no observable display hiccups. We, therefore, enforce that the display backlight level remains stable for at least five frames.

We use the Monsoon power monitor to measure real power savings. Figure 7 shows the overall power consumption of playing one video under different settings. This video is 9 minutes, 56 seconds long (596 seconds) and is encoded at 30 frames per second. In Figure 7, "schedule-0%" represents the setting where backlight scaling data is computed offline using maximum luminance data per frame, while "schedule-2%" represents the experiment setting where backlight scaling data is computed offline using 98-percentile luminance data per frame.

For the 10.1-inch Galaxy tablet, when our CAD mechanism is not used, the GPU is put into sleep mode, the average overall power consumption is 4,898mW. When we apply backlight scaling that yields no distortion, the average overall power consumption is 4,963mW, slightly higher than without adaptation. On the other hand, if up to 2% pixel distortion is allowed, the average power consumption can be reduced to 3,518mW, a 28% savings. If up to 5% pixel distortion is allowed, the average power consumption is further reduced to 3,264mW, a 33.4% savings. For the 7-inch Galaxy tablet, when the CAD mechanism is not used, the average overall power consumption is 3,091mW. When we apply joint backlight scaling and luminance compensation that yields no distortion, the average overall power consumption is 3,509mW. This is higher than without adaptation because the GPU power consumption is greater than the amount of power that can be saved at the display. On the other hand, if up to 2% distortion is allowed, the average power consumption can be reduced to 2,658mW, a 14% savings. If up to 5% distortion is allowed, the average power consumption is further reduced to 2,583mW, a 16.4% savings. For Nexus 9, when the CAD mechanism is not used, the average overall power consumption is 3,201mW. When no distortion is allowed, the average overall power consumption is 3,750mW, higher than without adaptation. On the other hand, if up to 2% distortion is allowed, the average power consumption can



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

Fig. 8. Estimated amount of power savings (accounting for GPU power consumption) for 1,000 720P videos downloaded from YouTube. Power savings is computed under different settings of pixel distortions.

be reduced to 2,663mW, a 16.8% savings. If up to 5% distortion is allowed, the average power consumption is further reduced to 2,603mW, an 18.7% savings.

To examine our CAD mechanism on a larger pool of videos, we run the dynamic programming algorithm on all 1,000 720P videos randomly selected from YouTube. For each video, we decode the video and extract the maximum, 98th percentile, 95th percentile, and 90th percentile pixel luminance for every frame. We use this data as input to our dynamic programming algorithm, which computes the backlight scaling level assignment for each frame that will yield no distortion, up to 2%, 5%, and 10% pixel distortion per frame, respectively. Given the backlight scaling level assignment data, we further use the power consumption models to examine if and how much net power savings can be achieved (i.e., display power savings being greater than GPU power consumption).

Figure 8 shows the results. For the 10.1-inch Galaxy tablet, results show that if no distortion is allowed, 38 out of 1,000 videos can save power with the CAD mechanism. If more distortion can be tolerated, more power can be saved: 57.8% of videos can save power with negligible (up to 2%) pixel distortion, and 77.7% of videos can save power with up to 5% pixel distortion. If up to 10% pixel distortion is allowed, 88.6% of videos can save power, and 55.7% of videos can save more than 490mW (10% on the 10.1-inch Galaxy) on average. For the 7-inch Galaxy tablet, Figure 8(b) shows that if no distortion is allowed, only 10 out of 1,000 videos can save power. However, if more distortion can be tolerated, 31.7%, 54.0%, and 73.9% of videos save power with up to 2%, 5%, and 10% pixel distortion, respectively. In addition, 34.7% of videos can save more than 309mW (10% on the 7-inch Galaxy) on average with up to 10% pixel distortion. For Nexus 9, if no distortion is allowed, only 6 out of 1,000 videos can save power. When more distortion can be tolerated, 20.1%, 35.7%, and 57.0% of videos can save power with up to 2%, 5%, and 10% pixel distortion, respectively. Under up to 10% pixel distortion, 27.6% of videos can save more than 320mW (10% on Nexus 9) on average.

For backlight scaling schedules that may produce pixel distortion, we calculate their PSNR and SSIM between the rendered video and the non-backlight-scaled video. We find that the PSNR values are always greater than 29dB, 24dB, and 20dB, when up to 2%, 5%, or 10% of pixels are rendered with lower observed luminance than their original luminance, 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 0.99, while existing schemes [Hsiu et al. 2011; Lin et al. 2014] that choose not to perform the computationally intensive luminance compensation step can only yield SSIM values around 0.9.

6.2. Mobile Real-time Video Call

To evaluate the performance of our online CAD power saving for real-time video calls, we installed our real-time video call application on two Android devices: a Nexus 4

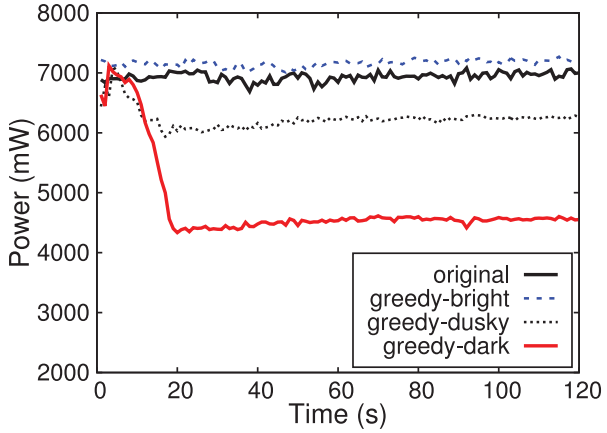


Fig. 9. Power consumption of the Samsung tablet during real-time video communication under different scenes.

smartphone and a Samsung Galaxy Tab 2 10.1-inch tablet. In our experiments, we set up video calls between the two devices in the same Local Area Network (LAN) in order to minimize the impact of the network. Parameters for the greedy algorithm are set according to experiments in Section 6.1.2. We measure real power consumption during video calls and examine if video call quality has been affected by our online CAD scheme.

6.2.1. Power Savings in Real-time Video Call. We measure the power consumption under scenes with different levels of brightness: *bright*, *dusky*, and *dark*. Under the *bright* scene, the maximum pixel luminance of frames captured by the camera is close to 255, which leaves our online CAD scheme very little space for backlight scaling. Under the *dusky* and *dark* scenes, the maximum luminance value is smaller, approximately 190 and 107, respectively. This allows more power to be saved by backlight scaling while maintaining user perceived brightness via luminance compensation.

We compare the power consumption of our online CAD scheme under different scenes with the power consumption of the original WebRTC application. Figure 9 shows the results measured from the 10.1-inch Galaxy tablet. Under *bright* scene, the original WebRTC applications consume 6,935mW power on average, while the power consumption of our application with online CAD is 7,160mW, slightly higher than the original application. This is expected because extra modules are used by our application, consuming extra power, while the backlight is never dimmed throughout the video call. Under *dusky* and *dark* scenes, since the original WebRTC application does not take brightness into consideration, its power consumption is almost the same as *bright* scene, 6,935mW. However, our application with online CAD mechanism gradually dims the backlight according to the greedy algorithm, reducing the power consumption shown as the beginning descendent gradient. As a result, the average power consumption is only 6,235mW and 4,783mW, respectively, saving 10.1% and 31.0% power compared to the original WebRTC application.

6.2.2. Video Quality. We next examine the quality of video calls made under our online CAD scheme. We focus on four metrics: Frames Per Second (FPS), PSNR, structural similarity SSIM, and user-perceived video call latency. We compare our scheme with the original WebRTC app.

Frame Rate. The original WebRTC application collects statistics about the number of frames that are captured by the camera, the number of frames encoded at the sender,

Table II. FPS of Video Stream Originating from Nexus 4 Smartphone to Samsung Tablet

	Input		Sent		Output	
	E	δ	E	δ	E	δ
Bright scenario						
Original App	23.78	1.66	19.09	2.81	13.91	4.41
Online CAD	23.99	1.18	18.79	3.06	13.60	3.17
Dark scenario						
Original App	8.02	1.37	8.00	0.42	4.85	0.72
Online CAD	7.85	0.48	8.00	0.18	4.96	0.53

and the number of frames decoded and rendered at the receiver every second. We rely on these statistics to examine if the frame rate is affected.

We find that the number of frames that are encoded at the sender side varies significantly if there are moving objects. The frame rate is highest when the frame content is static. We, therefore, focus on static scenes in our frame rate experiments. This allows us to evaluate our scheme during a video call with high frame rate. We expect if our scheme will not impact the video call with high frame rate, it will not impact the calls with lower frame rate either. Each experiment video call lasted 5 minutes. The resolution of captured video is set to 640×480. During the experiments, we measured the frame rates under scenes with different brightness. Note that the original WebRTC application limits the frame generation rate to 30 frames per second, and at most 15 frames are rendered per second. Since we implemented our application based on the original WebRTC, our application is also subject to these limitations.

Table II shows the frame rate of the video stream originated from the Nexus 4 smartphone to the Galaxy tablet in different stages. The *Input* column indicates the rate of frames captured by the camera. The *Sent* column indicates the frame rate of the video stream encoded by the sender. This stream is sent over the network and decoded at the receiver side. The final rendered frame rate is shown in the *Output* column. In this table, E stands for the expectation of the result and δ is the corresponding standard deviation. The table shows that the frame rate is always lower in the dark scenario, even in the original WebRTC application. We find this frame rate degradation is due to the specific implementation of the original WebRTC application. Comparing the two extreme scenarios, i.e., *bright* and *dark*, yielding the highest frame rate and the lowest frame rate, we find there is no degradation of video quality in terms of frame rate.

Image Fidelity. Image fidelity loss could occur in the online CAD mechanism for two reasons: (i) the maximum pixel luminance of frames increases abruptly, causing the distortion constraint to be violated; and (ii) the luminance information piggybacked in the delivered frames is lost due to video compression or network transmission. To measure video quality, we calculate the PSNR and SSIM between the receiver-observed video stream and the original stream captured at the sender. To align the frames, we insert a black frame into the streaming every 10 frames as the anchor. Then, we record all the frames on both sides. We only compare the frames between two anchor frames if there are exactly 10 frames recorded on both sides. The results are shown in Table III.

We first run the original WebRTC application under scenes with different brightness. For example, under the *dusky* scene, due to video compression, the PSNR and SSIM between the rendered frame and the original captured frame are 41.79dB and 0.98, respectively. We use these values as a baseline and see if the greedy algorithm used in online CAD causes more fidelity loss. The results show that the PSNR and SSIM of online CAD under the same *dusky* scene has the same value of 41.79dB and 0.98, indicating video quality is not affected. Under the *bright* scene and the *dark* scene, the SSIM values are also the same. The PSNR values are slightly decreased when using

Table III. PSNR (dB) and SSIM between the Rendered Video Stream and the Original Captured Frame

	Original WebRTC App		Online CAD	
	E	δ	E	δ
Bright Scene				
PSNR	41.89	4.85	41.79	4.85
SSIM	0.98	0.01	0.98	0.01
Dusky Scene				
PSNR	41.79	4.85	41.79	4.85
SSIM	0.98	0.01	0.98	0.01
Dark Scene				
PSNR	44.86	6.88	41.63	9.78
SSIM	0.97	0.01	0.97	0.01

online CAD. However, the PSNR values are still above 40dB, indicating there is very little distortion.

User-perceived Video Latency. We also measure the user-perceived video call delay. To minimize the impact of wide area network dynamics, we conduct the experiments in the same LAN. To measure the end-to-end delay, we place the camera on the mobile device in front of a stop watch and compare the timestamps rendered on two devices using the method proposed by Yu et al. [2014]. In the original WebRTC application, we find the average latency is 261ms during the video call. When the online CAD scheme is applied, the average latency is increased to 302ms, indicating approximately 40ms delay is introduced due to the additional processing.

7. RELATED WORK

Two types of displays are primarily used on mobile devices: LCD and Organic Light-Emitting Diode (OLED) display. The sources of power consumption in these two types of displays differ significantly. While the majority of LCD power is consumed by its backlight, regardless of the displayed content, OLED displays do not have backlights and emit light directly from pixels. OLED power consumption is directly related to the color of the pixels displayed. Given these different characteristics, different mechanisms have been proposed for saving power depending on display type.

For OLED displays, existing work mainly focuses on changing the color of displayed content to save display power consumption. For example, Dong and Zhong [2012] proposed Chameleon, a mobile web browser for OLED displays. Chameleon can reduce display power consumption by adapting a webpage's color scheme to a more power-efficient one on OLED displays. Tan et al. [2013] proposed FOCUS that can save power by darkening portions of the display that are outside of users' current region of interest. For video streaming, Chen et al. [2015] proposed DaTuM to apply dynamic tone mapping functions to color compositions on OLED screens. These color-changing schemes, however, could significantly change the visual content, which could further lead to inferior user experiences.

For LCD displays, existing work focuses on how to effectively use backlight scaling to save power without compromising the user experience. For example, Hsiu et al. [2011] and Lin et al. [2014] applied backlight scaling without the energy-intensive luminance compensation step. Videos were analyzed beforehand, and every frame was assigned a "critical" backlight level to maintain a minimum acceptable level of video quality. Anand et al. [2011] focused on saving LCD power consumption during mobile game play. They proposed to apply backlight scaling while increasing image luminance using the Gamma function during the final rendering phase in 3D games. Yan et al. [2015]

conducted a user study to investigate how reduced backlights affect users' subjective viewing experience. However, the authors did not consider luminance compensation in their study.

Our CAD approach differs from the methods described above in that it uses the GPU to perform luminance compensation. This use of GPU computation yields larger net power savings and better video quality compared to backlight scaling-only approaches. This article builds on our preliminary conference versions [Liu et al. 2015; Xiao et al. 2015] in two ways. First, it presents a unified approach toward luminance compensation under different video display contexts. Second, compared to our preliminary version, we have evaluated our proposed CAD scheme on a more diverse set of mobile devices, across a larger set of videos. Our new results show that CAD can effectively save display power consumption on a variety of mobile devices that are equipped with different GPUs. In addition, our experiments on a newly selected set of 1,000 randomly selected YouTube videos at the high resolution of 720P demonstrate, with higher confidence, that CAD can be adopted to save power on mobile devices.

8. CONCLUSION

In this work, we have designed and implemented CAD power saving mechanisms for reducing display power consumption of Internet video applications on mobile devices. CAD improves on previous backlight scaling schemes by using the GPU instead of the CPU for online luminance adjustment. In video streaming, we have designed an offline dynamic programming algorithm to pre-compute the backlight scaling schedule externally. During video playback, we perform backlight scaling and luminance compensation only when net power savings can be achieved. In real-time video communication, we have proposed an online greedy algorithm to run at the video stream receiver to compute its backlight on the fly and perform luminance compensation accordingly. In addition, the luminance information of every frame is computed at the sender side, avoiding repeated computation at multiple receivers. Experiment results show that CAD can effectively save more than 10% overall power consumption in video streaming for more than 55.7% of test videos randomly selected from YouTube and up to 31.0% overall power consumption in real-time video communication while maintaining good streaming quality.

Future Work. In the future, we plan to recruit a large sample of mobile users and conduct a more comprehensive study to uncover the relationship between users' Mean Opinion Scores (MOS) and backlight scaling parameter settings in our algorithms. Subjective user studies can complement our current objective metric-based video quality assessment by establishing stronger connections between these existing measures and user-centric data.

REFERENCES

Bhojan Anand, Karthik Thirugnanam, Jeena Sebastian, Pravein G. Kannan, Akhihebbal L. Ananda, Mun Choon Chan, and Rajesh Krishna Balan. 2011. Adaptive display power management for mobile games. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*. 57–70. DOI: http://dx.doi.org/10.1145/1999995.2000002

Aaron Carroll and Gernot Heiser. 2010. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference (USENIX ATC'10)*. USENIX Association, 271–284.

Naehyuck Chang, Inseok Choi, and Hojun Shim. 2004. DLS: Backlight luminance scaling of liquid crystal display. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 12, 8 (Aug 2004), 837–846. DOI: http://dx.doi.org/10.1109/TVLSI.2004.831472

Xiang Chen, Yiran Chen, and Chun Jason Xue. 2015. DaTuM: Dynamic tone mapping technique for OLED display power saving based on video classification. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. DOI: http://dx.doi.org/10.1145/2744769.2744814

Liang Cheng, Shivajit Mohapatra, Magda El Zarki, Nikil Dutt, and Nalini Venkatasubramanian. 2007. Quality-based backlight optimization for video playback on handheld devices. *Advances in MultiMedia* 2007, 1 (Jan 2007). DOI: http://dx.doi.org/10.1155/2007/83715

Wei-Chung Cheng and Massoud Pedram. 2004. Power minimization in a backlit TFT-LCD display by concurrent brightness and contrast scaling. *IEEE Transactions on Consumer Electronics* 50, 1 (Feb 2004), 25–32. DOI: http://dx.doi.org/10.1109/TCE.2004.1277837

Hyunsuk Cho and Oh-Kyong Kwon. 2009. A backlight dimming algorithm for low power and high image quality LCD applications. *IEEE Transactions on Consumer Electronics* 55, 2 (May 2009), 839–844. DOI: http://dx.doi.org/10.1109/TCE.2009.5174463

Inseok Choi, Hojun Shim, and Naehyuck Chang. 2002. Low-power color TFT LCD display for hand-held embedded systems. In *Proceedings of the 2002 International Symposium on Low Power Electronics and Design*. 112–117. DOI: http://dx.doi.org/10.1109/LPE.2002.146722

Mian Dong and Lin Zhong. 2012. Chameleon: A color-adaptive web browser for mobile OLED displays. *IEEE Transactions on Mobile Computing* 11, 5 (2012), 724–738. DOI: http://dx.doi.org/10.1109/TMC.2012.40

Pi-Cheng Hsiu, Chun-Han Lin, and Cheng-Kang Hsieh. 2011. Dynamic backlight scaling optimization for mobile streaming applications. In *Proceedings of the 17th IEEE/ACM International Symposium on Low Power Electronics and Design*. 309–314. DOI: http://dx.doi.org/10.1109/ISLPED.2011.5993655

Chun-Han Lin, Pi-Cheng Hsiu, and Cheng-Kang Hsieh. 2014. Dynamic backlight scaling optimization: A cloud-based energy-saving service for mobile streaming applications. *IEEE Trans. Comput.* 63, 2 (Feb 2014), 335–348. DOI: http://dx.doi.org/10.1109/TC.2012.210

Yao Liu, Mengbai Xiao, Ming Zhang, Xin Li, Mian Dong, Zhan Ma, Zhenhua Li, and Songqing Chen. 2015. Content-adaptive display power saving in internet mobile streaming. In *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. 1–6. DOI: http://dx.doi.org/10.1145/2736084.2736087

Sudeep Pasricha, Shivajit Mohapatra, Manev Luthra, Nikil D. Dutt, and Nalini Venkatasubramanian. 2003. Reducing backlight power consumption for streaming video applications on mobile handheld devices. In *ESTImedia*. 11–17.

Martino Ruggiero, Andrea Bartolini, and Luca Benini. 2008. 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*. 109–118. DOI: http://dx.doi.org/10.1145/1450058.1450074

Tajana Simunic, Luca Benini, Peter Glynn, and Giovanni De Micheli. 2001. Event-driven power management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 20, 7 (2001), 840–857. DOI: http://dx.doi.org/10.1109/43.931003

Kiat Wee Tan, Tadashi Okoshi, Archan Misra, and Rajesh Krishna Balan. 2013. FOCUS: A usable & effective approach to OLED display power management. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. 573–582. DOI: http://dx.doi.org/10.1145/2493432.2493445

Pei-Shan Tsai, Chia-Kai Liang, Tai-Hsiang Huang, and H. H. Chen. 2009. Image enhancement for backlight-scaled TFT-LCD displays. *IEEE Transactions on Circuits and Systems for Video Technology* 19, 4 (2009), 574–583. DOI: http://dx.doi.org/10.1109/TCSVT.2009.2014022

Mengbai Xiao, Yao Liu, Lei Guo, and Songqing Chen. 2015. Reducing display power consumption for real-time video calls on mobile devices. In *Proceedings of the 2015 International Symposium on Low Power Electronics and Design*. 285–290. DOI: http://dx.doi.org/10.1109/ISLPED.2015.7273528

Zhisheng Yan, Qian Liu, Tong Zhang, and Chang Wen Chen. 2015. Exploring QoE for power efficiency: A field study on mobile videos with LCD displays. In *Proceedings of the 23rd ACM International Conference on Multimedia*. 431–440. DOI: http://dx.doi.org/10.1145/2733373.2806269

Chenguang Yu, Yang Xu, Bo Liu, and Yong Liu. 2014. "Can you SEE me now?" A measurement study of mobile video calls. In *Proceedings of the 33rd IEEE International Conference on Computer Communications (INFOCOM 2014)*. 1456–1464. DOI: http://dx.doi.org/10.1109/INFOCOM.2014.6848080

Jia Zhou, Yanhua Li, Vijay Kumar Adhikari, and Zhi-Li Zhang. 2011. Counting YouTube videos via random prefix sampling. In *Proceedings of the 2011 ACM Conference on Internet Measurement (IMC'11)*. 371–380. DOI: http://dx.doi.org/10.1145/2068816.2068851

Received August 2015; revised June 2016; accepted September 2016