Reducing Display Power Consumption for Real-time Video Calls on Mobile Devices

Mengbai Xiao*, Yao Liu†, Lei Guo‡, and Songqing Chen*
*Department of Computer Science, George Mason University
†Department of Computer Science, SUNY Binghamton
‡Department of Computer Science and Engineering, Ohio State University
Email: *{mxiao3, sqchen}@gmu.edu, †yaoliu@cs.binghamton.edu, ‡lguo@cse.ohio-state.edu

***Abstract*—The display subsystem of a mobile device usually consumes 38%-68% [1] of the total battery power in video streaming. Therefore, a few schemes have been designed to reduce the display power consumption. The basic idea is to dim the backlight level while properly compensating the pixel luminance to maintain image fidelity. The luminance compensation and proper backlight level calculation are computation intensive and demand per-frame luminance information. For these reasons, existing schemes only work for video-on-demand where each frame (and thus the luminance information) is available in advance. In addition, they demand additional computing resource support. Otherwise, if the computation is conducted on the mobile device, the power consumption due to such computation can easily offset the power savings from dimming the backlight.**

In this work, we set to investigate power saving for real-time video calls on mobile devices. Different from video-on-demand, real-time video calls are highly delay sensitive and the frame luminance information is not known in advance. Moreover, video calls often involve multiple streaming sources from multiple ($\geq$2) participants, making it more difficult. Because there are few background changes and the frame rate is usually small in video calls, we design a Greedy Display Power saving scheme, called LCD-GDP, which utilizes the commonly available GPU on mobile devices without demanding additional support. Our design is implemented on WebRTC, a popular real-time web browser based video call standard. Experiments show that our scheme can save up to 33% power consumption in video calls without affecting the video call quality.

***Keywords*—backlight scaling; power saving; mobile; video conferencing**

I. Introduction

With the ever-improving network and mobility support, recently, real-time communication software, being standalone or combined with social media, have found their way on mobile devices. Being able to support video calls while entirely free or at least cheaper than the traditional voice communications, these software, such as Skype [2], QQ [3], Apple's Facetime [4], and Google WebRTC [5], are quickly gaining popularity among common mobile users.

However, the limited battery power supply remains the Achilles' heel of mobile devices while real-time video communications are very power-hungry. Compared to video streaming, real-time video calls are more demanding. In real-time video communication, such as video conferencing, video contents are generated on the fly from multiple sources. Even for one-to-one video call, frames from two sources are generated and delivered, which is different from the one-way video-on-demand streaming. Thus, efficiently reducing the power consumption for video calls on mobile devices is imperative.

Since the display subsystem on a mobile device often consumes about 38% to 68% of the total energy during video streaming [1], a few schemes have been designed to reduce the power consumption of the display. For a Liquid-Crystal Displays (LCD) display, the dominant power consumption unit is the display backlight. Thus to reduce the power consumption, it is desirable to dim the backlight as much as possible while enhancing the luminance of the pixels to compensate the quality degradation due to backlight dimming. In this way, the modified images on the screen can maintain image fidelity to human eyes. Such a technique is called backlight scaling.

Backlight scaling often consists of three stages, i.e., 1) the histogram generation stage, 2) the backlight levels determination stage, 3) the pixel luminance compensation stage. The first stage and the last stage are both computation intensive due to per-pixel manipulation. Therefore, some previous design either ignore the luminance compensation [6], [7], leading to significant quality degradation and unsatisfactory user experience, or demand additional computing resources for compensation [8], [9], making the scheme impractical for mobile devices. In the second stage, some previous research applies the dynamic programming to find the globally optimal backlight levels without violating the hardware and user experience constraints [6], [7]. These schemes require that the video frames be available in advance. However, for real-time video calls, this is impossible because the video content is generated on the fly. Furthermore, being highly sensitive to delay, real-time video calls cannot tolerate the delay that may be caused by dynamic programming. In addition, since video calls often involve multiple participants and need to receive and display data from multiple sources simultaneously, the backlight scaling technique that works on a single video stream cannot be directly used for video calls.

In this paper, we set to explore display power saving for video calls on mobile devices. Since there are often few background changes and the frame rate is usually small in practical video calls, we propose a Greedy Display Power saving scheme, called *LCD-GDP*. Our greedy algorithm must conform to the same set of constraints as the dynamic programming approach used for video streaming. The *first* is that the backlight variations between neighboring frames must be limited. Otherwise users will experience the flickering effect [10]. The *second* is that the backlight should not be

scaled down too much to cause image distortion. And the *third* is that the backlight cannot be adjusted too frequently because the hardware needs some time to respond [11].

To avoid processing the same frame repeatedly, in LCD-GDP, *first*, we migrate the per-frame pixel luminance histogram generation to the sender side for video calls. We embed the luminance information inside each frame without having to build an additional channel. Multiple pieces of this information are put in the frame to account for possible loss during the video encoding/decoding and the network transmission. *Second*, since GPU is commonly equipped on mobile devices, we take advantage of GPU to offload some tasks from the CPU. To render the received frames in a timely manner, we use the OpenGL ES 2.0 shaders to perform the pixel luminance enhancement. Moreover, the power saved at the display will not be offset by using the GPU since GPU is already used in the video conferencing for composing frames together and then rendering them.

To evaluate the performance of our design, we implement LCD-GDP in WebRTC and run experiments on a tablet and a smartphone. Experimental results show that by using the LCD-GDP, up to 33.2% power consumption can be saved on average. Our evaluations of the video call quality based on the frames per second received, the latency, and image fidelity using PSNR (Peak-Signal-to-Noise Ratio) and SSIM (Structure SIMilarity) also show that LCD-GDP introduces negligible impact on the received video call quality.

This remainder of the paper is organized as follows. We describe some background information for both backlight scaling and WebRTC in Section II. We present our design of LCD-GDP in Section III and the implementation details in Section IV. Evaluation results are discussed in Section V and we conclude our work in Section VI.

II. Background and Related Work

A. Backlight Scaling

A visible image on a LCD display is produced by both backlight and the LCD panel which stores pixel color information. The perceptual luminance is actually the backlight intensity compensated by the pixels. 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 a darker version of this image. This distortion can be compensated by concurrently increasing the luminance component of each pixel in this image [8], [12]–[14]. And increasing the pixel luminance does not increase the power consumption of the display.

However, directly applying backlight scaling to video playback faces several challenges. Both extracting and enhancing the pixel's luminance component frame by frame demand processing a mass of data, and is computation intensive. For this reason, deploying these tasks during the playback on the same CPU [12], [14] is not practical because of two reasons. First, the CPU may not have enough time to perform these computation intensive tasks without degrading the video quality (e.g., frames per second) of the video playback. Second, the achieved power saving by dimming the backlight can be offset by the power consumption due to these extra CPU operations. Hsiu et al. and Lin et al. [6], [7] proposed to skip the pixel compensation stage and use a critical backlight level for each frame to avoid severe image distortion. Ruggiero et al. suggested to offload the luminance adjustment tasks to the hardware image processing unit (IPU) integrated in Freescale's multimedia application processor [15]. It exploits in a smart and efficient way to implement a hardware assisted image compensation. Pasricha et al. [8] and Cheng et al. [9] suggested to compute the backlight scaling data on a proxy server and substitute the original video with a luminance-adjusted version. In short, existing schemes (1) only target video-on-demand where the video frame information is available in advance so that computation can be done before the playback, and (2) demand additional infrastructure support for compensation or do not consider quality degradation due to backlight dimming. So far, no scheme has been considered for real-time video calls.

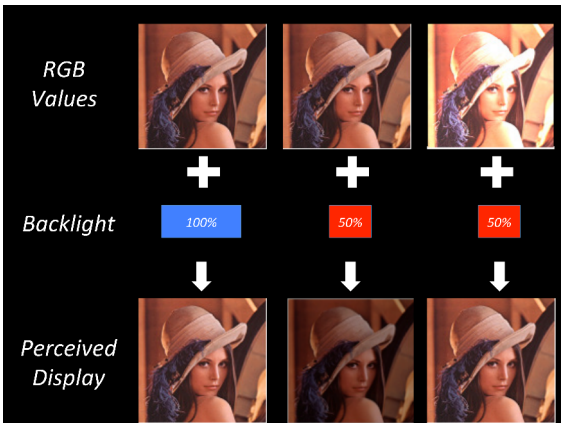

Fig. 1: Backlight Scaling on LCD screen

Compared to video-on-demand, real-time video calls are highly delay-sensitive and frames are generated on the fly. Furthermore, video calls often involve multiple participants, and thus multiple frames from different sources need to be combined in real-time, leaving little computing power for other tasks, such as pixel luminance compensation.

B. Video calls and WebRTC

Video calls/conferencing are gaining increasing popularity. On mobile devices, many applications, such as Skype [2], QQ [3], and Facetime [4], all support video calls/conferencing. The communication protocol of these applications is proprietary to commercial companies, which precludes the communication between different applications. The increasing categories of mobile devices, such as smartphones, tablets and emerging wearable devices, make the situation even more complicated. To solve this fragmentation problem in the real-time multimedia communication and also to provide a cross-platform solution, Web Real-Time Communication (WebRTC) [16] is proposed to enable video communications via web browsers and standardized by the W3C and IETF. Nowadays, mainstream browsers, e.g., Chrome, Firefox, Opera, all have integrated the WebRTC. The WebRTC component [5] implemented in Chrome provides Javascript-style APIs. This

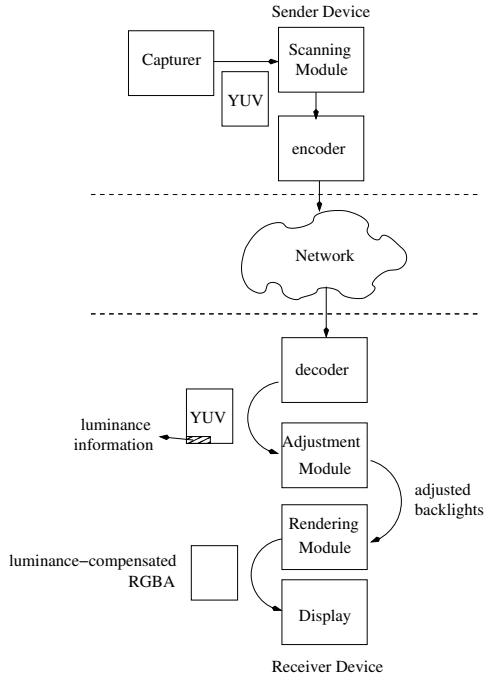

Fig. 2: LCD-GDP system architecture and major components.

component can also be linked to the native mobile apps as an external library. WebRTC is now widely used by mobile users. However, given its video streaming nature, the power consumption of video communication is high, which has slowed down its pervasion.

III. System Design

A. System Components

Backlight scaling generally involves three steps for 1) generating the luminance histogram, 2) determining the proper backlight levels and 3) compensating the pixel luminance. In our system, we organize them into the following three modules: the **Scanning** module, the **Adjustment** module, and the **Rendering** module. Next we present our new approaches in these three modules, respectively. Figure 2 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 communication (RTC), the frames are generated by the video capturer, which is usually a physical camera. It is impossible to access the whole video content in advance. So the scanning module is necessary to generate the luminance histogram for later backlight level determination and luminance compensation. Other than video on demand (VOD), RTC sessions involve multiple $N \geq 2$ participants, and the rendered image is composed by all received frames (including the frame captured by the receiver itself). While it is natural to conduct the scanning after receiving the image at the receiver side, in our design, the scanning module is installed at the sender side. In this way, each generated frame will only need to be scanned once. (Otherwise, every receiver needs to conduct scanning for the same image 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 this end, 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 the multiple positions that are known in advance by both ends. The receiver extracts the information and uses the maximum value among the candidates. It then passes the value to the **Adjustment** module for backlight scaling.

Greedy Luminance Adjustment: The **Adjustment** module is located at the receiver side. This module aims to find the appropriate backlight levels from the luminance information. In VOD, global luminance information is required to prevent the flickering effects, which is caused by frequently scaling the backlight level with great variation–VOD usually involves numerous scene changes. But real-time video communication is highly delay-sensitive, buffering a certain amount of frames for the purpose of waiting for enough luminance information to be accumulated (and thus calculate the optimal backlight levels) is not practical. In our design, we propose to use a greedy algorithm in the determination stage, as we discuss in the next subsection. 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. After this, the new backlight levels are sent to the **Rendering** module for pixel compensation and backlight scaling.

GPU-assisted Rendering: The **Rendering** module strengthens the original rendering function by adding pixel luminance compensation and backlight scaling. In our design, instead of using the CPU, we choose the commonly available GPU on today's mobile devices to enhance the pixel luminance. For video calls, typically the GPUs are already enabled for resizing, composing the received frames and performing RGB-YUV conversion. Thus, little power consumption overhead can be expected by the adding the pixel compensation task. Luminance compensated frames rendered with scaled backlight level allows users to see frames with no distortion from the original version.

B. Backlight Determination Algorithms

We next discuss the backlight determination algorithm used by the **Adjustment** module.

Ideally, to maximize power saving without affecting user experience, the algorithm should conform to the following constraints:

$$b \geq \frac{Y^{max}}{255} \tag{1}$$

$$b \in [\ b' \times (1 - \Delta_b),\ b' \times (1 + \Delta_b)] \tag{2}$$

Algorithm 1 The Greedy Algorithm

▷ On input (t, Y_t^{max}, b', t'), where b' is the last adjusted backlight level and the t' is the corresponding frame index, we generate the b_t, the backlight level of the t^{th} frame.

if $t = 1$ **then**
    $b' \leftarrow Y_t^{max}/255$
    $t' \leftarrow t$
    $b_t \leftarrow b'$ **return** b_t
end if

if $t - t' < l_{min}$ **then return** b'
end if

$b_t \leftarrow Y_t^{max}/255$
if $b_t < b' \times (1 - \Delta_b)$ **then**
    $b' \leftarrow b' \times (1 - \Delta_b)$
else if $b_t > b' \times (1 + \Delta_b)$ **then**
    $b' \leftarrow b' + (1 + \Delta_b)$
else
    $b' \leftarrow b_t$
end if

$b_t \leftarrow b'$
$t' \leftarrow t$
return b_t

$$t \geq \ t' + l_{min} \tag{3}$$

Equation 1 represents the distortion constraint. Y_t^{max} represents the maximum pixel luminance of the frame. Since pixel luminance cannot be scaled to be bigger than its maximum (i.e., 255), the backlight level b must be high enough. Otherwise, distortion will occur. The user experience constraint is quantified as the relationship between b and the most recent backlight level, b', in Equation 2. To prevent the flickering effects, the backlight level can only be scaled up or down by at most Δ_b for continuous frames. Equation 3 represents the hardware constraint. Since it takes time for the display hardware to apply the new backlight level, backlight levels must stay the same for at least l_{min} frames.

While a dynamic programming algorithm can maximize power savings while satisfying all constraints, it requires all frames be available in advance. In real-time video communication, however, frames are generated on the fly, and buffering frames would inevitably increase the user-perceived delay. 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 in Equation 1. We expect our scheme will not lead to significant distortion based on the intuition that few scene changes are likely to be found during the video call sessions. We evaluate if our conjecture holds in practice via experiments in Section V.

The pseudo code of our greedy algorithm is shown in Algorithm 1. The backlight level of the tth frame, b_t, only depends on the most recent backlight level b', its index t' and the maximum luminance of current frame Y_t^{max}. In the adjustment, b_t still conforms to the constraints represented in Equations 2 and 3. Distortion may occur if there is significant change in Y_t^{max} and b_t can not be adjusted to satisfy Equation 1. Assuming that there is no frequent scene changes in the video calls, we expect such distortion is rare and will be corrected gradually in next adjustment operations.

IV. Implementation

We integrated our LCD-GDP scheme into the WebRTC open source project [5], which is the WebRTC component of the Chrome browser implemented by Google. Then this component can be linked to the WebRTC app, an Android app, and then we use this app to do the 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 client that participates in the video call.

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, LCD-GDP 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 conversation. Eventually the framebuffer generated by the shaders is flushed to the screen.

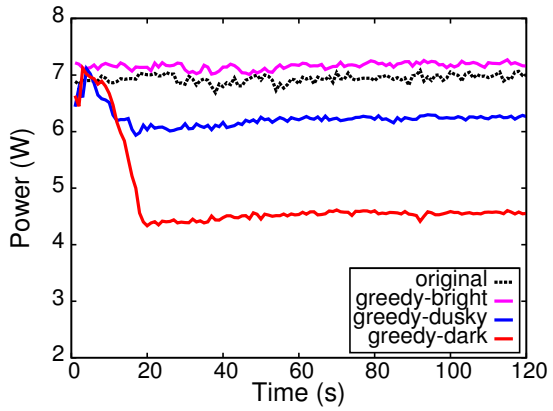

Fig. 3: Power consumption of the Samsung tablet during real-time video communication under different scenes.

V. Evaluation

To evaluate the performance of LCD-GDP, we conducted experiments with two devices: one Nexus 4 smartphone and one Samsung Galaxy 10.1-inch tablet. We set up video calls between the two devices in the same LAN in order to minimize the impact of the network and we measure the real power consumption using the Monsoon power monitor [17]. In addition, we examine if video call quality has been affected by our LCD-GDP scheme.

A. Power Consumption

We measure the power consumption under scenes with different levels of brightness. These scenes are tagged with *bright*, *dusky* and *dark*. Under the *bright* scene, the maximum pixel luminance of frames is close to 255, which leaves our LCD-GDP scheme very little space for backlight scaling. Under the *dusky* and *dark* scenes, with smaller maximum luminance value (about 190 and 107, respectively), we can save more power by dimming the backlight and maintain the observed brightness via luminance compensation. We compare the power consumption of our LCD-GDP scheme under different scenes with the power consumption of the original version of WebRTC app. The results are shown in Figure 3. Since the power consumption of the original WebRTC app under different scenes is almost the same, we only plot the result under bright environment in this figure, which is 6.935 watts on average. This value is considered as the baseline. Under the same *bright* scene, the power consumption of our LCD-GDP is 7.160 watts, slightly higher than the baseline. This is because extra modules are integrated into the system, consuming more power, while the backlight is never dimmed throughout the conversation. In the *dusky* and *dark* scene experiments, the backlight intensity is gradually dimmed by the greedy algorithm, reducing the power consumption shown as the beginning descendent gradient. As a result, only 6.235 watts and 4.783 watts power is consumed on average, saving 12.92% and 33.20% power, respectively.

B. Video Quality

We next examine the quality of video calls made under our LCD-GDP scheme. We focus on four metrics: frames per second (FPS), peak signal to noise ratio (PSNR), structural similarity (SSIM), and user-perceived video call latency. We compare our scheme with the original WebRTC app.

TABLE I: 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
LCD-GDP	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
LCD-GDP	7.85	0.48	8.00	0.18	4.96	0.53

Frame Rate. To measure the number of frames that are captured, encoded at the sender side and decoded and rendered at the receiver side, we use the statistics reported by the WebRTC app directly. We find that at the sender side, the number of frames that are encoded varies significantly if there are moving objects. The frame rate reaches the highest value when the frame content is a static scene. Since our scheme is agnostic to frame content, we only consider static scene in our experiments. This allows us to evaluate our scheme during video call with high FPS. We expect if our scheme will not impact the video call with high FPS, it will not impact the calls with lower FPS either. We set the resolution of captured video to 640×480 and measured the FPS under scenes with different brightness. Note that in the WebRTC app where we incorporated our LCD-GDP scheme, the frame rate of generation is limited to 30 FPS and at most 15 frames are rendered per second. In each experiment, the video call lasted at least 5 minutes. During the video call, the FPS metric is recorded every second. Then we report the statistical results of the FPS in different stages for the video streaming originating from the Nexus smartphone to the Samsung tablet.

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

Image Fidelity. Fidelity loss could occur in LCD-GDP for two reasons: (i) the maximum pixel luminance of frames increases abruptly, causing the constraint shown in Equation 1 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 Peak Signal-to-Noise Ratio (PSNR) and Structural SIMilarity (SSIM) between the receiver-observed video stream and the original stream captured at the sender. To align the frames,

TABLE II: PSNR (dB) and SSIM between the rendered video stream and the original captured frame.

	Original WebRTC App		LCD-GDP	
	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

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 the Table II.

Using the original WebRTC app under Dusky Scene, for example, the PSNR and SSIM between the rendered frame and the original captured frame is 41.79 and 0.98, respectively. The difference is caused by video compression as expected. We use this value as baseline and see if using the greedy algorithm causes more fidelity loss. The result show that the PSNR and SSIM of LCD-GDP under the same scene has the same value of 41.79 and 0.98, indicating video quality is not affected. Under the Bright Scene and the Dark Scene, the SSIM values are always the same. The PSNR values are slightly decreased when using LCD-GDP. However, the PSNR values are still above 40 dB, 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 local area network (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. [18]. In the original WebRTC app, we find the average latency is 261 milliseconds during the video call. When the LCD-GDP scheme is applied, the average latency is increased to 302 milliseconds, indicating only about 40 milliseconds delay are introduced due to the additional processing.

VI. Conclusion

Video communications are power-hungry. Real-time video calls add additional challenges because real-time video calls are highly delay sensitive and no video frames are available in advance. In this paper, we explore the possibility of using backlight scaling technique to save the display power consumption during real-time video calls. We have designed and implemented a greedy display power saving scheme, LCD-GDP. In LCD-GDP, the luminance scanning and generation is migrated to the sender side during video capturing, and the information is piggybacked to the receiver. Furthermore, we propose a greedy algorithm that is practical for backlight level determination in real-time video calls. Lastly, LCD-GDP leverages the GPU instead of the CPU to conduct real-time luminance compensation. We build LCD-GDP based on the WebRTC, and our evaluations on a smartphone and a tablet show that LCD-GDP can save up to 33% of power consumption without affecting the video call quality.

Acknowledgement

We appreciate constructive comments from anonymous referees. The work is partially supported by NSF under grant CNS-1117300.

References

[1] 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*, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 21–21.

[2] "Skype," http://www.skype.com/en/.

[3] "Tencent QQ," http://www.qq.com/.

[4] "Apple Facetime," https://www.apple.com/mac/facetime/.

[5] "WebRTC Project," http://www.webrtc.org/.

[6] P.-C. Hsiu, C.-H. Lin, and C.-K. Hsieh, "Dynamic backlight scaling optimization for mobile streaming applications," in *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, Aug 2011, pp. 309–314.

[7] C.-H. Lin, P.-C. Hsiu, and C.-K. Hsieh, "Dynamic backlight scaling optimization: A cloud-based energy-saving service for mobile streaming applications," *IEEE Trans. Computers*, vol. 63, no. 2, pp. 335–348, 2014.

[8] S. Pasricha, S. Mohapatra, M. Luthra, N. Dutt, and N. Venkatasubramanian, "Reducing backlight power consumption for streaming video applications on mobile handheld devices," in *In Proc. First Workshop Embedded Systems for Real-Time Multimedia*, 2003, pp. 11–17.

[9] L. Cheng, S. Mohapatra, M. El Zarki, N. Dutt, and N. Venkatasubramanian, "Quality-based backlight optimization for video playback on handheld devices," *Adv. MultiMedia*, vol. 2007, no. 1, pp. 4–4, Jan. 2007.

[10] A. Iranli, W. Lee, and M. Pedram, "Hvs-aware dynamic backlight scaling in tft-lcds," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 10, pp. 1103–1116, 2006.

[11] S. Pasricha, M. Luthra, S. Mohapatra, N. Dutt, and N. Venkatasubramanian, "Dynamic backlight adaptation for low-power handheld devices," *IEEE design & test of computers*, no. 5, pp. 398–405, 2004.

[12] W. Cheng, Y. Hou, and M. Pedram, "Power minimization in a backlit tft-lcd display by concurrent brightness and contrast scaling," in *Design Automation Conference*, 2004.

[13] N. Chang, I. Choi, and H. Shim, "Dls: dynamic backlight luminance scaling of liquid crystal display." *IEEE Trans. VLSI Syst.*, vol. 12, no. 8, pp. 837–846, 2004.

[14] I. Choi, H. Shim, and N. Chang, "Low-power color tft lcd display for hand-held embedded systems," in *Proceedings of the 2002 International Symposium on Low Power Electronics and Design*, ser. ISLPED '02. New York, NY, USA: ACM, 2002, pp. 112–117.

[15] 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*, ser. EMSOFT '08. New York, NY, USA: ACM, 2008, pp. 109–118.

[16] "WebRTC Standard," http://w3c.github.io/webrtc-pc/.

[17] "Monsoon Power Monitor," http://www.msoon.com/LabEquipment/PowerMonitor/.

[18] C. Yu, Y. Xu, B. Liu, and Y. Liu, ""can you see me now?" a measurement study of mobile video calls," in *INFOCOM, 2014 Proceedings IEEE*, April 2014, pp. 1456–1464.