Improving Screen Power Usage Model on Android Smartphones

Ziling Lu, Chun Cao, XianPing Tao
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
Department of Computer Science and Technology, Nanjing University, Nanjing, China
nju10lzl@gmail.com, caochun@nju.edu.cn, txp@nju.edu.cn

***Abstract*—With user experience becoming richer and richer on Android smartphones, limited battery capacity has become a major concern for users. It is a good practice for Android to tell users where the battery power has gone. Android achieves this by a power profile provided by OEMs that specifies the power usage rate for each system component staying at each specific state. According to a recent study, screen is one of the most dominant power-hungry system components on Android smartphones. However, the accuracy of Android's screen power usage model is to be improved, taking not only the brightness, but also the displayed content into account. We modified Android source code to improve its screen power usage model and verified this improvement with experiments. The results show that our new screen power usage model is more accurate to a significant extent and introduces little overhead.**

KeyWords: Android Smartphones; Screen Power Usage; Power Model

I. Introduction

Android has become the world's most popular mobile operating system, especially on smartphones. However, due to various newly developed, power-hungry features and limited battery capacity, users are increasingly concerned about the power usage of each application and each system component, such as web browser [1], screen, Wi-fi [2], GPS [3], etc. In particular, screen is one of the most dominant power-hungry system components on Android smartphones, according to a recent study [4].

Android tells users where the battery power has gone with a *power profile* [5], provided by smartphone manufactures, for system components. This profile specifies the power usage rate required to keep each component staying at each specific state. Within a power profile, power usage rate is specified in milliamps (mA) of current at a nominal voltage. For example, to build a screen power usage model, this profile specifies the current required to keep the screen on at minimum brightness and at maximum brightness. To calculate the battery power drained by the screen, Android counts the time intervals the screen is kept on at each brightness level, then multiply these time intervals by corresponding interpolated current values and add up the products.

However, screen power usage rate may vary on account of different colors displayed. That is to say, despite the same brightness, the power usage rate still varies depending on what is displayed on the screen. Typically it requires much less energy to display darker colors, such as black, than lighter colors, such as white. An all-white screen can consume 6 times more energy than an all-black screen at maximum brightness according to our experiments. As a result, Android's screen power usage model is poor in accuracy, which will further affect the accuracy of the entire power usage statistics.

In this paper we propose a screen power usage model leveraging not only the brightness, but also the displayed content of the screen. We first reveal the influence of displayed content on screen power usage rate. We set the screen to display different colors and measure corresponding screen power usage rate and try to find the relationship. Then we modify Android source code to improve its screen power usage model making use of the relationship, more specifically, we measure and add some extra information (e.g. current required to keep an all-black screen on at minimum brightness) to Android's power profile and calculate screen power usage rate based on screen content and brightness.

There are several challenges to this work: First, modern smartphone screens may have millions of pixels and the screen power usage rate may be affected by the color value of each pixel. Second, Android typically refreshes its screen content at a rate of 30 or 60 frames per second (fps). If we want to seize every variation of screen content, we must instrument lowest levels of Android, which will make a heavyweight implementation. Third, there is no effective way to measure screen power usage over a period of time, so we lack a ground truth in our evaluation. We must find alternative ways to verify our improvement.

We make three contributions in this work: First, we construct a new and accurate screen power usage model that can be applied to most screens. Second, we implement this model on Android smartphones by modifying Android's source code, because smartphones are suffering from limited battery capacity and Android is open source. Third, we evaluated this new screen power usage model with a large number of experiments, which show that it is more accurate to a significant extent and introduces little overhead.

The rest of this paper is organized as follows: In Section 2 we discuss related work and describe the motivation of this work. Section 3 introduces original screen power usage model of Android and points out its deficiency. In Section 4 we derive the relationship between screen power usage rate and displayed cotent and present our new screen power usage model. Section 5 deals with its implementation on Android and Section 6 presents the evaluation. Certain threats to validity are discussed in section 7. Finally, conclusion and future work are discussed in Section 8.

1530-1362/15 $31.00 © 2015 IEEE
DOI 10.1109/APSEC.2015.45

II. Related Work and Motivation

There are a lot of research efforts paid to calculate power usage of Android smartphones in recent literatures. Some of them focus on power modeling of applications. For example, Pathak et al. [6] constructed application power models based on system call tracing, while Hao et al. [7] estimated application power consumption using a combination of program analysis and per-instruction power modeling. Others focus on estimation of the power consumption of the whole system based on power modeling of each system component. For example, Dong et al. [8] presented a self-modeling method leveraging the built-in battery interface that provides current information, while Zhang et al. [9] proposed to build component power models based on SOD (state of discharge) readings of batteries. Yoon et al. [10] combined these two aspects of work and estimated application power consumption based on their component usage information. Still others focus on power modeling or saving of a certain system component, such as Wi-fi [2] or GPS [3]. However, none of them has ever paid attention to the power profile related functions provided by Android system.

In addition, there are some research efforts focusing on power modeling of smartphone screens. For example, Dong et al. proposed a method of power modeling of smartphone screens [11] and developed a color-adaptive web browser for Android smartphones based on the model [12], while Li et al. [13] presented an approach to transform the color schemes of web pages to improve energy efficiency of web applications on smartphones, based on the fact that smartphone screens consume less energy when display darker colors than lighter colors. However, they never paid attention to the construction of the screen power usage model and the implementation of the model on Android smartphones. What's more, they never focused on calculating power usage of the screen over time, which could be especially heavyweight when displaying a video with a high frame rate. We make up for those shortages mentioned above in this paper.

III. Android's Screen Power Usage Model

On Android smartphones, users can see such power usage information under Settings–>Battery interface as shown in Figure 1. In this interface, Android tells users the power usage of each application and each system component since last full charge. For example, screen has consumed 42% battery power and camera has consumed 7% battery power since last full charge in this case.

Android achieves this by a *power profile* [5], provided by smartphone manufactures, for system components. For example, in order to calculate screen power usage, this power profile contains two pieces of information: (1) additional power usage rate required to keep screen on at minimum brightness, compared to screen off, i.e., the value of item "screen.on", and (2) additional power usage rate required to keep screen on at maximum brightness, compared to at minimum brightness, i.e., the value of item "screen.full". Here, power usage rate is specified in milliamps of current since the output voltage of the battery remains nearly unchanged. Manufactures obtain these values by measuring and comparing the current drawn by the device in two different states, e.g, screen on at minimum

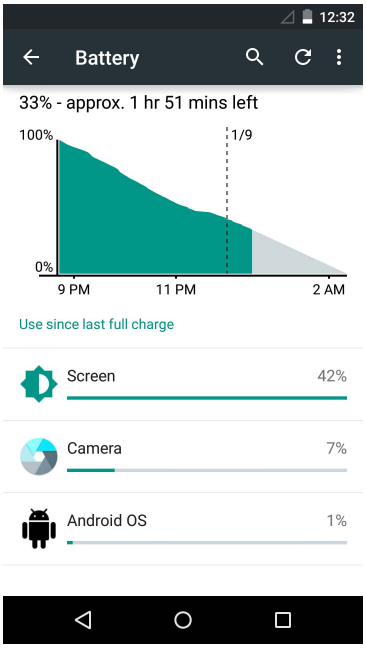

Figure 1. Screen shot of Settings–>Battery

brightness and screen off, the difference between which is the value of item "screen.on". Android leaves the measurement tasks to manufactures because these values may vary among different devices. Example values of item "screen.on" and "screen.full" provided by Android are 200mA and 300mA, respectively. Then, the power usage rate required to keep screen on at brightness x (x is from 0 to 255), denoted as "screen.x", is calculated as an interpolated value between "screen.on" and "screen.on" plus "screen.full", i.e.,

$$screen.x = screen.on + \frac{x}{255} * screen.full \qquad (1)$$

To simplify the calculation, Android divides screen brightness equally into five brightness levels, i.e., dark, dim, medium, light and bright, and the interpolation fraction is approximated to 0.1, 0.3, 0.5, 0.7 and 0.9, respectively. Afterwards, Android counts the time intervals the screen is kept on at each brightness level. Finally, screen power usage is calculated as the sum of the power usage at each brightness level, which is the product of corresponding power usage rate and time interval.

However, when measuring the current drawn to keep screen on at certain brightness, the displayed content has great influence on the result. Displaying darker colors tend to require much less energy than displaying lighter colors. For example, on an LG Nexus 5 device, an all-white screen at maximum brightness consumes energy at a rate of 586.5mA, while an all-black screen at maximum brightness only consumes energy at a rate of 99.5mA in our experiments. That is to say, an all-white screen can consume energy at a rate of approximately 6 times faster than an all-black screen at maximum brightness. But Android's screen power usage model approximates all these rates to a single value, i.e., the value of "screen.on" plus "screen.full". As a result, this model is actually poor in accuracy to an extent that is not negligible.

IV. Screen Power Usage Model Leveraging Screen Content

To build a more accurate screen power usage model, we must take not only brightness, but also displayed content into account. Screen content consists of millions of pixels on modern smartphones, each of which is specified as an RGB value. To start with, we make the screen display a single color, that is, all of the pixels has the same RGB value. Moreover, this same RGB value is set from #000000, #100000, ..., to #f00000, #ff0000, i.e., from black to full red. Meanwhile we measure the current required to keep screen on at maximum brightness. The result is shown in Figure 2. The horizontal axis is the red value, i.e., from 00 (0) to ff (255).

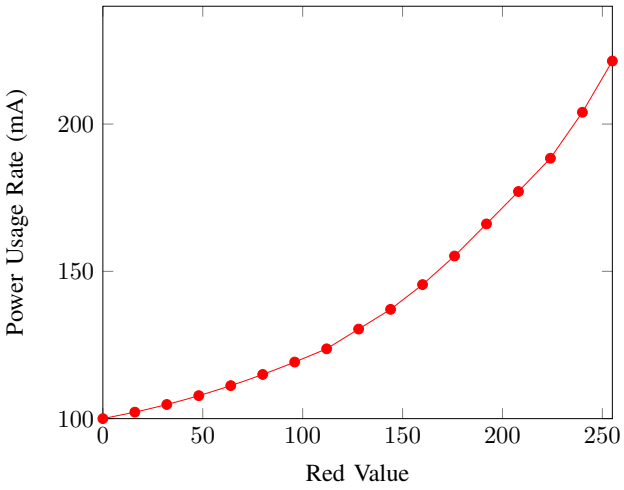

Figure 2. Power usage rate of a red screen(1)

Intuitively, screen power usage rate seems to be exponentially correlated with the red values. We conducted same experiments on green values (i.e., #000000, #001000, ..., #00ff00) and blue values(i.e., #000000, #000010, ..., #0000ff) and got similar results.

In fact, in order to optimize the usage of bits when encoding an image, there is an operation used to encode luminance into RGB values, called gamma correction [14]. It is based on the fact that humans are more sensitive to relative differences between darker colors than between lighter ones, following an approximate gamma or power function. In sRGB color space standard [15] used with most screens, luminance is encoded into sRGB values using formula (2). Let L_C be L_R, L_G, and L_B where L represents luminance, and C_{srgb} be R_{srgb}, G_{srgb} and B_{srgb}:

$$C_{srgb} = \begin{cases} 12.92\, L_C & L_C \leq 0.0031308 \\ 1.055\, L_C^{1/2.4} - 0.055 & L_C > 0.0031308 \end{cases} \quad (2)$$

C_{srgb} values are in the range 0 to 1. If values in the range 0 to 255 are required, simply multiply by 255. Inversely, sRBG values can be decoded back into luminance using formula (3):

$$L_C = \begin{cases} C_{srgb}/12.92 & C_{srgb} \leq 0.04045 \\ (\frac{C_{srgb}+0.055}{1.055})^{2.4} & C_{srgb} > 0.04045 \end{cases} \quad (3)$$

where C_{srgb} values can be calculated as sRGB values divided by 255.

We decoded red values back into red luminance using formula (3) and the result is shown in Figure 3. As we can see, generally there is a linear relation between power usage rate and luminance. Again, experiments on green values and blue values got similar results.

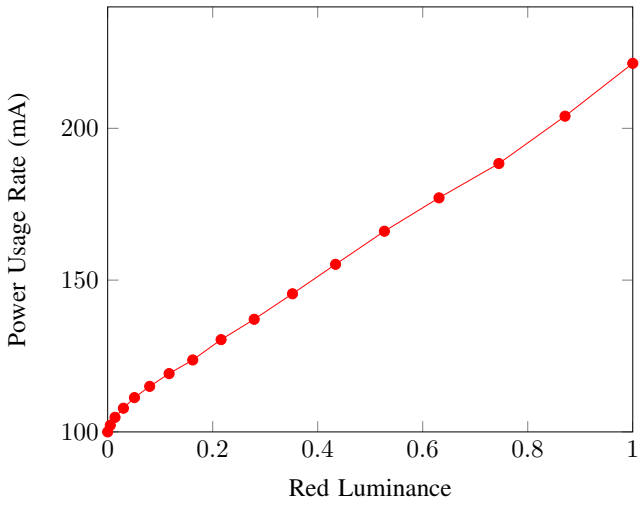

Figure 3. Power usage rate of a red screen(2)

We assume screen power usage rate is the sum of the power usage rate required to keep an all-black screen on and the power usage rate required by pixels emitting red, green and blue light. So, given brightness x, we establish our new screen power usage model as follows:

$$P = screen.x.black + L_{R\_AVG} * screen.x.red + L_{G\_avg} * screen.x.green + L_{B\_avg} * screen.x.blue \quad (4)$$

Here, P represents power usage rate of screen. "screen.x.black" represents power usage rate required to keep an all-black screen on. L_{Ravg}, L_{Gavg} and L_{Bavg} represent average red, green and blue luminance, which can be decoded from average red, green, and blue values of all the pixels using formula (3). "screen.on.red", "screen.on.green" and "screen.on.blue" represent power usage rate required by a full-red, green and blue screen emitting red, green and blue light.

Next, taking screen brightness x into account, we need to get the values of "screen.x.black", "screen.x.red", "screen.x.green" and "screen.x.blue". Take "screen.x.black" as an example, it should be an interpolated value between "screen.on.black" (where x equals 0) and "screen.on.black" plus "screen.full.black" (where x equals 255). That is to say,

$$screen.x.black = screen.on.black + \frac{x}{255} * screen.full.black \quad (5)$$

Similarly, we have

$$screen.x.red = screen.on.red + \frac{x}{255} * screen.full.red \quad (6)$$

$$screen.x.green = screen.on.green + \frac{x}{255} * screen.full.green \quad (7)$$

$$screen.x.blue = screen.on.blue + \frac{x}{255} * screen.full.blue \quad (8)$$

As a result, once we get the 8 values on the right sides of formulas (5)-(8), we can predict screen power usage rate given screen brightness x and screen content, i.e., the bitmap of the screen shot, from which we can calculate average red, green and blue values of all the pixels and decode back into average red, green and blue luminance.

V. Implementation

The next part of our work is applying our new screen power usage model to Android smartphones. Thanks to Android's open source nature, we can achieve this goal by modifying Android's source code directly.

First of all, we need to modify and get the values of the items in Android's power profile. As explained in Section IV, instead of items "screen.on" and "screen.full", we need 8 new items, i.e., "screen.on.black/red/green/blue" and "screen.full.black/red/green/blue", all of which can be measured through experiments.

To get the value of "screen.on.black", we subtract the current drawn of the device when screen is off from the current drawn when an all-black screen is on at minimum brightness. To get the value of "screen.on.red/green/blue", we subtract the current drawn when an all-black screen is on at minimum brightness from the current drawn when a full-red/green/blue screen is on which is also at minimum brightness. To get the value of "screen.full.black", we subtract the current drawn when an all-black screen is on at minimum brightness from the current drawn when an all-black screen is on at maximum brightness. To get the value of "screen.full.red/green/blue", we subtract the current drawn when an all-black screen is on at maximum brightness from the current drawn when a full-red/green/blue screen is on which is also at maximum brightness. These measurement tasks can be done by manufactures in advance.

Then, we divide red/green/blue luminance equally into 5 luminance levels just like brightness levels. That's because instead of counting the time intervals the screen is kept on at each brightness level, we need to count the time intervals the screen is kept on at each brightness & red/green/blue luminance level. That means we need 5^4 = 625 timers. Each time the screen brightness or screen content changes, we need to recalculate the new brightness & red/green/blue luminance levels. As long as one of these 4 levels changes, we need to switch to the corresponding timer.

However, Android typically refreshes its screen content at a rate of 30 or 60 frames per second (fps). If we want to seize every variation of screen content, we must instrument the kernel or HAL layers of Android, which will give rise to a heavyweight implementation. To solve this problem, instead of listening to red/green/blue luminance level changes and waiting for messages passively, we actively take screen shot in a new thread in the application framework layer at regular intervals, for example, every 1 second, and calculate its average red/green/blue luminance levels, and suspend/resume this thread when screen is turned off/on, which makes a relatively lightweight implementation. As the average red/green/blue luminance levels of the screen typically dose not change that frequently in everyday use of Android smartphones (possibly except when displaying a video), we believe checking screen content every 1 second is enough.

In addition, we can further reduce the overhead of our implementation by extracting only part of the pixels of each screen shot, for example, one single pixel out of every 100 pixels (the top left one out of a 10*10 square of pixels), and calculate the average red/green/blue luminance of the extracted pixels instead of all the pixels, which will introduce little inaccuracy to our implementation.

Afterwards, we calculate the screen power usage rate as follows:

$$\begin{aligned} P = {} & screen.on.black + xLvl * screen.full.black \\ & + L_R Lvl * (screen.on.red + xLvl * screen.full.red) \\ & + L_G Lvl * (screen.on.green + xLvl * screen.full.green) \\ & + L_B Lvl * (screen.on.blue + xLvl * screen.full.blue) \end{aligned} \tag{9}$$

Here, xLvl, L_RLvl, L_GLvl and L_BLvl represent the interpolation fraction for each brightness level and each red/green/blue luminance level representatively, whose value can be 0.1, 0.3, 0.5, 0.7 and 0.9.

Finally, screen power usage is calculated as the sum of the power usage at each brightness and red/green/blue luminance level, which is the product of corresponding power usage rate and time interval.

The whole procedure is implemented in one single sub project in the application framework layer of Android, i.e., /platform/framework/base. We modified 5 files (including 4 java files and 1 xml file) and approximate 300 lines of code in total.

VI. Evaluation

To make an empirical evaluation of our work, we performed a large number of experiments to answer 4 research questions:

RQ1: Is it accurate to calculate the average red/green/blue luminance of a screen shot based on part of the pixels instead of all the pixels?

RQ2: Is our screen power usage model accurate in predicting power usage rate of the screen displaying certain content at certain brightness?

RQ3: How does the time interval of checking screen content impact the accuracy of our implementation of our model?

RQ4: How much overhead will be caused by our implementation of our model on Android smartphones?

A. Experimental Setup

We use Monsoon Power Monitor [16] to measure power usage rate (current drawn) of a smartphone. It is a specialized battery-monitoring tool recommended by Android [5], which can supply a stable voltage to the smartphone and sample the current drawn by the smartphone at a rate of 5KHz. That is to say, it can act as power supply and meter at the same time.

We use an LG Nexus 5 and a Samsung Nexus S as our host devices. The former one is equipped with an IPS screen with a resolution of 1920*1080, i.e., over 2 million pixels, while the latter one is equipped with an OLED screen with a resolution of 800*480, i.e., approximate 0.38 million pixels.

When we measure the current required to keep screen on, we keep other components of the smartphone in idle states. We turn off Wi-Fi, GPS, bluetooth and turn on airplane mode. We shut down all running applications, including those running in background. We turn off automatic brightness control and automatic power off after a certain time period of inactivity. To get a precise control of screen brightness, we connect the smartphone to our laptop and use an ADB command "echo x > /sys/class/leds/lcd-backlight/brightness" (may be different among other devices) to set screen brightness to x (from 0 to 255). To control screen content we wrote a small Android application to display an image on the whole screen.

The current measurement environment is illustrated in Figure 4.

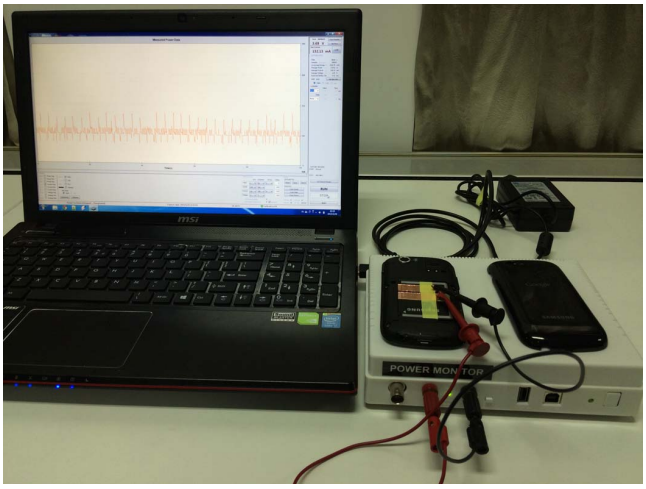

Figure 4. Environment for current measurement

B. RQ1

To answer RQ1, we collected 100 screen shots during everyday use of the host devices and calculated the average red/green/blue luminance of each of the 100 screen shots, based on (1) all the pixels, and (2) one out of every 100 pixels, respectively. The average absolute error of (2) compared to (1) of all the 100 screen shots was only 0.5%, which shows that calculating the average red/green/blue luminance of a screen shot based on part of the pixels (one out of every 100 pixels) will introduce little inaccuracy. In fact we can extract even less pixels to "represent" red/green/blue luminance of a screen shot according to our experiments, for example, one out of every 1,024 (10,000) pixels, which will cause an absolute error of approximately 1% (5%), to reduce even more overhead of our implementation.

C. RQ2

To answer RQ2, first of all we measured the values of screen.on.black/red/green/blue and screen.full.black/red/green/blue for our host devices following the instructions listed in Section V. Then we wrote a simple java program, which implements our screen power usage model leveraging the 8 measured values and calculates the average red/green/blue luminance of a screen shot based on one out of every 100 pixels, to predict the power usage rate of the screen of our host devices to display certain content (which is parameterized as a java.awt.image.BufferedImage) at certain brightness. Afterwards, we used the 100 screen shots collected in RQ1 and random generated brightness as input to run the program. We used Monsoon Power Monitor to measure the actual current drawn by the screen of our host devices displaying the same screen content and at the same brightness. Figure 5 shows the histogram of the absolute error of the 100 experiments. The average absolute error between the predicted and actual current drawn of the 100 experiments was only 2.7%, while the maximal absolute error was only 5.9%.

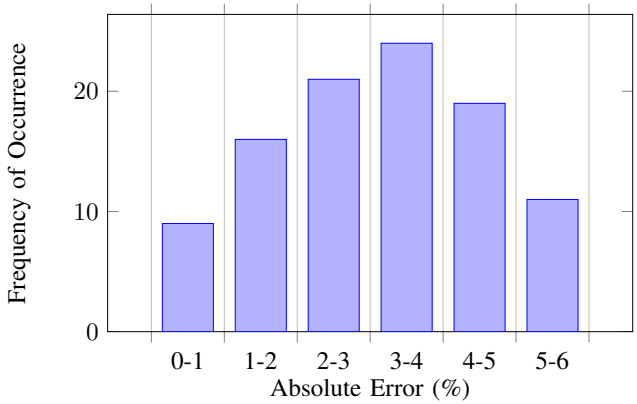

Figure 5. Absolute error of the 100 experiments

As a contrast we used the original screen power usage model of Android and "screen.on" and "screen.full" values in the power profile of our host devices to predict the current drawn, taking only screen brightness as a parameter, the average absolute error reached 45%, while the maximal absolute error reached 150%.

D. RQ3

To answer RQ3, we recorded 4 video clips during everyday use of the host devices, including when we (1)use them at will, (2)browse web pages, (3)watch a video and (4)play a mobile game. These 4 video clips are all 5 minutes long and their frame rates are all 30 fps. That is to say, they all contain 9,000 frames in total. For each video clip, we extracted some of their frames using JCodec [17] at certain time intervals, i.e., every 10 seconds, 5 seconds, 3 seconds, 2 seconds, 1 second, 1/2 second, 1/3 second, 1/5 second, 1/10 second, 1/15 second and 1/30 second, thus we got 30, 60, 100, 150, 300, 600, 900, 1500, 3000, 4500 and 9,000 "representative" frames. Then we calculated average red/green/blue luminance of each group of "representative" frames and compared with average red/green/blue luminance of all the 9,000 frames (all based on one out of every 100 pixels of each frame). We did not use Monsoon Power Monitor to measure the current drawn of the smartphone because part of the current would be drawn by the CPU processing user events. Table 1 to Table 4 show

time interval (s)	10	5	3	2	1	1/2	1/3	1/5	1/10	1/15	1/30
red	0.09571	0.04034	0.01312	0.01161	0.00182	0.00253	0.00057	0.00168	0.00066	0.00002	0.00000
green	0.10664	0.04811	0.00679	0.01347	0.00374	0.00176	0.00090	0.00223	0.00078	0.00003	0.00000
blue	0.14393	0.06779	0.01006	0.01524	0.00282	0.00118	0.00125	0.00252	0.00070	0.00005	0.00000
average	0.11543	0.05208	0.00999	0.01344	0.00279	0.00182	0.00091	0.00215	0.00071	0.00003	0.00000

Table 1. Absolute error of calculated average red/green/blue luminance when extracting frames of video clip (1) at different time intervals

time interval (s)	10	5	3	2	1	1/2	1/3	1/5	1/10	1/15	1/30
red	0.25054	0.10326	0.04771	0.01612	0.01044	0.00870	0.00432	0.00140	0.00154	0.00122	0.00000
green	0.32532	0.14569	0.06178	0.01130	0.01194	0.01078	0.00331	0.00021	0.00168	0.00133	0.00000
blue	0.23982	0.08674	0.05746	0.01008	0.01113	0.00948	0.00601	0.00027	0.00119	0.00128	0.00000
average	0.27189	0.11190	0.05565	0.01250	0.01117	0.00965	0.00455	0.00063	0.00147	0.00128	0.00000

Table 2. Absolute error of calculated average red/green/blue luminance when extracting frames of video clip (2) at different time intervals

time interval (s)	10	5	3	2	1	1/2	1/3	1/5	1/10	1/15	1/30
red	0.07088	0.03686	0.05585	0.01342	0.01044	0.01269	0.00504	0.00449	0.00081	0.00069	0.00000
green	0.00443	0.02967	0.02354	0.01005	0.00275	0.00537	0.00511	0.00202	0.00027	0.00052	0.00000
blue	0.02469	0.00867	0.01602	0.00569	0.00390	0.00327	0.00559	0.00109	0.00053	0.00014	0.00000
average	0.03333	0.02507	0.03180	0.00972	0.00570	0.00711	0.00524	0.00253	0.00054	0.00045	0.00000

Table 3. Absolute error of calculated average red/green/blue luminance when extracting frames of video clip (3) at different time intervals

time interval (s)	10	5	3	2	1	1/2	1/3	1/5	1/10	1/15	1/30
red	0.25150	0.02984	0.03793	0.01339	0.02896	0.01014	0.00778	0.00381	0.00035	0.00044	0.00000
green	0.45609	0.19565	0.05299	0.03687	0.04246	0.01769	0.01252	0.00538	0.00009	0.00095	0.00000
blue	0.40443	0.29722	0.00316	0.08010	0.03261	0.01685	0.01142	0.00472	0.00012	0.00079	0.00000
average	0.37067	0.17424	0.03136	0.04346	0.03468	0.01489	0.01057	0.00464	0.00019	0.00073	0.00000

Table 4. Absolute error of calculated average red/green/blue luminance when extracting frames of video clip (4) at different time intervals

the absolute error of the 4 video clips representatively. We list the results in 4 tables because they may vary in the pattern of luminance variation between frames.

As we can see, frames extracted every 1 second can generally "represent" the average red/green/blue luminance of the corresponding video clip, as the absolute error was only around 1%. So we conclude that checking screen content every 1 second is enough in the implementation of our model. We can set this time interval longer to reduce the overhead of our model or shorter to be more accurate.

E. RQ4

To answer RQ4, we monitored the CPU utilization rate of our host devices before and after we apply our new screen power usage model, checking screen content every 1 second and calculating average red/green/blue luminance based on one out of every 100 pixels. CPU utilization rate can be calculated from /proc file system for Linux devices. The difference between the CPU utilization rates was less than 1%, which indicated that our new model only introduced little overhead when applied to Android smartphones. Even when checking screen content every 1/5 second and calculating average red/green/blue luminance based on all the pixels, the extra CPU utilization rate introduced by our new model was only 3%. If the overhead is a major concern, we can set the time interval of checking screen content even longer, and calculate average red/green/blue luminance of a screen shot based on even less pixels, which will sacrifice the accuracy of our implementation of our new model to some extent, but still much more accurate than the original screen power usage model.

In addition, we measured the current drawn by our host devices using Monsoon Power Monitor before and after we apply our new screen power usage model. Figure 6 shows the current drawn when displaying a video clip before and after we apply our new model, illustrated by red and blue line respectively. As we can see, the difference was negligible.

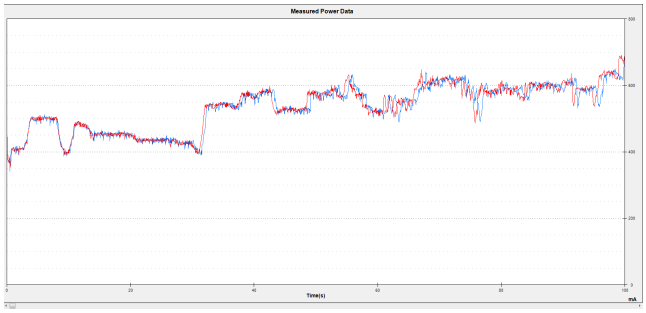

Figure 6. Current drawn by the host device displaying a video clip before/after we apply our new model

VII. Threats to Validity

In this section, we discuss a certain threat to validity of the results presented in this paper. As there is no effective way to measure total screen power usage over time, we lack a ground truth in our evaluation. So we must find alternative ways to verify the accuracy of our new model in calculating total screen power usage over time.

Since we have already verified the accuracy of our new model in predicting power usage rate of the screen displaying certain content at certain brightness, we only need to verify the accuracy of our implementation in calculating the time intervals of the screen staying at each brightness and red/-green/blue luminance level. So we wrote a small Android application to display an image on the whole screen for a certain period of time at a certain brightness and then switch to display another image for another certain period of time at another certain brightness and so on. We compared time intervals of the screen staying at each brightness and red /green/blue luminance level calculated by our implementation with the actual time intervals specified in the application, and add these time intervals together to compare with the total time the screen is on, the difference was negligible.

VIII. Conclusion And Future Work

In this paper, we derived the relationship between screen power usage rate and displayed content on smartphones and established a new and accurate screen power usage model based not only on screen brightness but also on screen content. We applied this model to Android smartphones by modifying Android's source code. We verified the accuracy of our new screen power usage model with a large number of experiments, which show that it is more accurate to a significant extent and introduces little overhead.

There are still several possible enhancements to our current work. First, we monitor screen content at regular intervals instead of every time screen content changes, this reduces the overhead of our implementation but sacrifices the accuracy to some extent. We need to achieve a balance. Second, we believe our screen power usage model applies to screens of other devices as well, as long as they use sRGB color space standard, such as tablets, PCs and TVs, etc, but we need more experiments. Finally, our work focuses only on the construction of our new screen power usage model on Android smartphones, yet how to reduce screen power usage is the most important thing.

Acknowledgment

This work was partially supported by the National 863 project under the Grant 2015AA01A203 and the NSFC under Grants 91318301, 61321491.

References

[1] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh, "Who killed my battery?: analyzing mobile browser energy consumption," in *Proceedings of the 21st international conference on World Wide Web*, pp. 41–50, ACM, 2012.

[2] K.-H. Kim, A. W. Min, D. Gupta, P. Mohapatra, and J. P. Singh, "Improving energy efficiency of wi-fi sensing on smartphones," in *INFOCOM, 2011 Proceedings IEEE*, pp. 2930–2938, IEEE, 2011.

[3] Z. Zhuang, K.-H. Kim, and J. P. Singh, "Improving energy efficiency of location sensing on smartphones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 315–330, ACM, 2010.

[4] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone.," in *USENIX annual technical conference*, vol. 14, 2010.

[5] "Power profiles for android — android developers." http://source.android.com/devices/tech/power/index.html.

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

[7] S. Hao, D. Li, W. G. Halfond, and R. Govindan, "Estimating mobile application energy consumption using program analysis," in *Software Engineering (ICSE), 2013 35th International Conference on*, pp. 92–101, IEEE, 2013.

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

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

[10] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "Appscope: Application energy metering framework for android smartphone using kernel activity monitoring.," in *USENIX Annual Technical Conference*, pp. 387–400, 2012.

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

[12] M. Dong and L. Zhong, "Chameleon: a color-adaptive web browser for mobile oled displays," *Mobile Computing, IEEE Transactions on*, vol. 11, no. 5, pp. 724–738, 2012.

[13] D. Li, A. H. Tran, and W. G. Halfond, "Making web applications more energy efficient for oled smartphones," in *Proceedings of the 36th International Conference on Software Engineering*, pp. 527–538, ACM, 2014.

[14] "Gamma correction - wikipedia." https://en.wikipedia.org/wiki/Gamma_correction.

[15] "srgb - wikipedia." https://en.wikipedia.org/wiki/SRGB.

[16] "Monsoon solutions." https://www.msoon.com/LabEquipment/PowerMonitor/.

[17] "Jcodec - home." http://jcodec.org/.