Lightweight Measurement and Estimation of Mobile Ad Energy Consumption

Jiaping Gui, Ding Li, Mian Wan, and William G. J. Halfond
University of Southern California
Los Angeles, California, USA
{jgui, dingli, mianwan, halfond}@usc.edu

ABSTRACT

Mobile ads are an important component of the app ecosystem. Typically, developers use ads to generate revenue and, in return, end users get a "free" app. However, recent work has shown that apps with ads actually have significant hidden costs to end users in terms of energy, network usage, and performance. These can affect the ratings and reviews an app receives. Therefore, it is desirable for developers to balance the usage of ads with these potential negative costs. However, for energy, developers lack techniques to help them measure ad costs to their apps. To address this problem, we propose and evaluate several lightweight statistical approaches for measuring and predicting ad related energy consumption. We evaluate our approaches on real-world market apps and find that they are able to accurately and quickly estimate the energy cost without requiring expensive infrastructure or extensive developer effort.

1. INTRODUCTION

Ads displayed in mobile apps have become an important part of the mobile app ecosystem. In 2014 alone, marketers spent 31 billion dollars on mobile advertising, up from 17 billion in 2013 [7]. Analysts expect this amount to continue to increase and predict that by 2017 revenue from mobile advertising will exceed that of TV advertisements [11] and account for one in three dollars spent on advertising [8].

Mobile ads have become a popular way for mobile app developers to earn revenue, and, as of 2014, over half of all apps contained mobile ads [24]. Typically, a developer releases his app to end users without charge and earns revenue when ads are displayed or clicked on by end users. This is perceived as a "win-win" situation by both parties, as users get a "free" app and developers can get a potentially large revenue stream. However, recent work has shown that this perception of a "free" app is misleading [13]. In fact, these "free" apps have significant hidden costs for both end users and developers. For users, the ads require a large amount of CPU processing time, slowing down the responsiveness of mobile apps; generate a large amount of network traffic, often doubling the amount of data used by an app; and consume a significant amount of energy, shortening battery life of the mobile device. For developers, these hidden costs can lead to negative reviews and poor app ratings from end users. Hence, it is important for developers to obtain information about ad hidden costs and avoid potential revenue loss.

Unlike the energy cost, performance and network cost can be easily obtained by off-the-shelf software or tools (e.g., *top* and *tcpdump*). This leaves developers with no easy way to determine the size or impact of the energy cost. Although recent work [13] developed techniques to measure energy cost, there are limitations to the applicability and general usage of their techniques. Namely, the techniques used custom built instrumentation and specialized measuring equipment. It is not reasonable to expect that a typical developer will have the resources or expertise to duplicate this infrastructure. Second, these techniques are only useful when the developer has fully implemented an app's integration with an ad network and implemented significant functionality of the app. At earlier stages of the development, such as design, developers are not able to obtain estimates of their app's energy consumption with respect to ads nor understand their potential impact. This means that for significant portions of an app's development lifecycle, developers have no insight into an important aspect of their app's behavior that can affect how their app will be received by end users. This increases the risk for developers that once they are able to ascertain the energy cost, they may need to redesign user interfaces or change ad related functionality. Since this is happening late in the process, it can lead to increased development costs and possibly delay the delivery of the app.

To address this problem, we propose and evaluate two lightweight statistical approaches for determining end user's ad energy cost. The first of these techniques is designed to provide feedback early in the app's development lifecycle. For this we build a statistical model that uses information from an ad's `unit profile`, which is the high-level configuration information (e.g., ad type and refresh rate) that a publisher requests from an ad network. This model allows developers to get early feedback, even without a working implementation, into how expensive their app will be with respect to the user energy cost. This can provide developers with an opportunity to compare their app with others early in the software lifecycle and possibly change their planned ad usage before investing development effort. Although this technique can provide useful guidance, it is possible to pro-

GREENS'16, May 16 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4161-5/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2896967.2896970

vide even more accurate estimates once functioning code has been written. This is because the energy cost is directly related to runtime information that varies based on implementation structure. Therefore, we introduce a second technique that can capture key metrics and use these to provide developers with more accurate information about an app's ad related energy consumption. Our evaluation focuses on demonstrating the capability of our proposed techniques for measuring and predicting user's ad related energy cost. In particular, we found that our techniques could predict the energy consumption of mobile ads to within 31% using the ad unit profile and within 14% using runtime metrics of ground-truth measurements. Overall, these results show that our techniques can provide actionable information and could form the basis for a technique that can guide developers in the usage of ads in their apps.

The rest of this paper is organized as follows: In Section 2, we introduce background information on mobile advertising and explain why simple techniques are not effective for determining the size and impact of the energy cost. We describe our approach in Section 3, and the generation of testing phase energy model in Section 4. We show that ad energy consumption can be modeled accurately for a set of real market apps in Section 5. We discuss related work and conclude in Sections 6 and 7.

Program 1 Example of adding ads to an app

```
 1 private static final String AD_UNIT_ID = "ca-app-pub
        -1234567890123456/1234567890";
 2 private AdView adView;
 3 public void onCreate(Bundle savedInstanceState)
 4 {
 5   super.onCreate(savedInstanceState);
 6   setContentView(R.layout.activity_main);
 7   adView = new AdView(this);
 8   adView.setAdSize(AdSize.BANNER);
 9   adView.setAdUnitId(AD_UNIT_ID);
10   LinearLayout layout = (LinearLayout) findViewById(R.id.
          linearLayout);
11   layout.addView(adView);
12   AdRequest adRequest = new AdRequest.Builder().build()
          ;
13   adView.loadAd(adRequest);
14 }
```

2. BACKGROUND & MOTIVATION

To embed ads into mobile apps, developers need to call special APIs provided by an ad library. Program 1 is an example of how an app loads banner ads from the Google ad library. In this program, developers use *AdView* to display ads. First, developers need to define the size of ads (SIZE) (line 8). In practice, SIZE has a finite number of values and these values are predefined in the AdSize class. In this example, we show the most common value of SIZE, which is banner size. However, developers can also select other values, such as *FULL_BANNER*, *LARGE_BANNER* and *SMART_BANNER*. After the size is specified, developers need to set the *AD_UNIT_ID*, a unique ID generated by an ad network that associates the app requests to an ad network with the corresponding ad unit profile (line 9). After binding the *AD_UNIT_ID*, developers can specify two other configurations, TYPE and RRATE. TYPE defines what kind of ads, text or image, need to be displayed. RRATE defines how often the app refreshes the contents of its ads. For AdMob (a Google mobile advertising network), it is an integer ranging from 30 to 120, which means that the app reloads the contents of ads every 30 to 120 seconds.

As we show in Section 3.1 and Section 5.3, the behavior of ads in an app can be statically approximated by the values of SIZE, TYPE, and RRATE. However, such a static model can only provide a first approximation of the behavior of mobile ads. To get more accurate estimates, we need more precise information about the execution environment of the app. Intuitively, the energy cost of ads depends on the state of the smartphone when mobile ads are executed and displayed. It consists of the energy consumed by different components on the smartphone, such as network and display: the network energy depends on network API usage; and display energy depends on the exact colors and background used by the app and ads. All of these conditions are affected by the structure of the apps, its implementation, and runtime interactions with the ad APIs. This motivates the development of techniques that can obtain precise information about the runtime behavior of an app with respect to ads so that the ad energy cost can be more accurately determined.

However, compared to some ad metrics (e.g., network throughput), which can be calculated with careful runtime monitoring techniques, the energy consumption is more challenging to estimate or measure. The reason for this is that although it is simple to use ads, the mechanisms behind the ad APIs are complex. For example, at line 13 of Program 1, when the API `loadAd` is called, the app will communicate with the Google Play Service to fetch data related to the ads. Thus, in this example the energy cost of ads include two parts: the first part is the cost of API calls, such as `loadAd`; the second part is the energy consumed by the Google Play Service to fetch the ad data. Once an ad is displayed, there are also costs incurred by the display component itself that are independent of the API used to send the data [30]. Current profiling based approaches (e.g., *eLens* [15] and *vLens* [19]) cannot measure the energy cost of the Google Play Service since it is a system level service. These challenges motivate the development of the statistical modeling based techniques that we describe in this paper.

3. APPROACH

The goal of our approach is to provide energy cost information about mobile ads throughout the whole life cycle of app development. Our approach provides two types of information about the energy cost of mobile ads for app developers: feedback about the energy consumption of ads during the design and early implementation phase and the estimation of the runtime energy consumption after the app has been implemented. The flow of our approach is shown in Figure 1. Our approach takes the code of the App Under Development (AUD) as input and provides energy information about the ads in the AUD. In the implementation phase, before the AUD is implemented, our approach uses a static model to provide early feedback about the energy consumption of mobile ads based on the ad configurations. After the AUD has been implemented (i.e., in a post-implementation phase), our approach instruments the app to generate the with-ads and no-ads version, and then executes the same workload on both versions to measure the ad related metrics. Based on these metrics, our approach then calculates the energy consumption of ads with a runtime model.

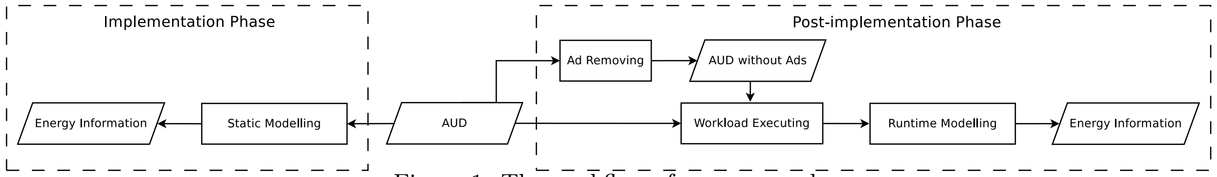

Figure 1: The workflow of our approach

3.1 The Static Model

The purpose of the static model is to provide early information about the energy consumption of ads for developers before the AUD has been fully implemented. Notice that before the AUD has been implemented, the only variables developers could see is the static configuration of ads, which includes the SIZE, TYPE, and RRATE. Thus, our static model takes a static ad configuration as input and returns a high level estimate of the energy consumption of mobile ads. Since the values of these three variables are defined within the app as constants, it is possible to build a static model that only takes the value of SIZE, TYPE, and RRATE as parameters to provide some information regarding the behavior of mobile ads.

To build the static model, we measured the energy consumption of ads under different configurations and performed statistical analysis on the data we collected. Specifically, we changed one specific configuration (e.g., RRATE) with the other parameters fixed, and ran the same workload on the app for a number of times during different time periods of a day. We found that the average energy consumption of mobile ads has a linear relationship with the value of RRATE, when SIZE and TYPE are fixed. With increasing RRATE (i.e., less frequent ad refresh), the average energy consumption of ads decreases linearly. This relationship is defined by Equation (1), where β is the average energy of ads with the lowest RRATE, which is 30, and α is the coefficient.

$$E = \beta - \alpha * (RRATE - 30) \tag{1}$$

3.2 Calculating Energy Cost With the Implemented AUD

After the AUD is implemented, developers could collect runtime information by executing the app. In this scenario, the goal of our approach is to provide more precise information about the energy consumption of mobile ads. We calculate the ad energy cost because it could provide more information about ads and may help developers optimize their app more effectively.

Our approach takes the implemented AUD and workloads (or tests) provided by developers as input, then it calculates the concrete energy consumption of ads while executing the workload. Our approach utilizes a systematic strategy to achieve the cost calculation: first, our approach removes ads from the implemented real world app; second, it runs both the with-ads and no-ads version of the app to get runtime ad metrics; third, the approach calculates the energy cost of mobile ads using the collected metrics. In the following part of this section, we introduce detailed information about each of the steps in our approach.

3.2.1 *Defining the Energy Model*

In our approach, the energy model is used to calculate the energy cost of mobile ads. It takes ad related metrics, which include the CPU time at different frequencies, the network usage, and the screenshots associated with timestamps, as input and returns a numeric value as the energy consumption of ads during execution. The model has three components as shown in Equation (2), where E_{system}, $E_{network}$, and $E_{display}$ are the energy consumption of the system, network, and display components, respectively. These components are essential for the execution of ad functionalities.

$$E = E_{system} + E_{network} + E_{display} \tag{2}$$

System Energy Model:
The system energy model in our approach takes the system information (e.g., CPU time at different frequencies) as input. The output of this model is the system energy consumed by ads. According to related work [33], the energy consumption of the CPU component can be modeled as a linear relationship of the CPU time at different frequencies. Hence, the value of the CPU energy model is shown as the first part of the right hand side (RHS) in Equation (3), where f is the CPU frequency of the target device, and n is the number of all CPU frequencies that the target device can have. T_f is the CPU time that the system spent running mobile ads when CPU is at the frequency f, and P_f is the coefficient, which represents the system power consumption at the frequency f. The second part of the RHS in the equation models the energy consumption of memory, where P_m denotes the energy consumption per unit memory usage and M is the average memory usage of ads.

$$E_{system} = \sum_{f=1}^{n} P_f T_f + P_m M \tag{3}$$

Network Energy Model:
The network energy model takes as input the total bytes transmitted by ads through the network, and calculates the network energy of ads. The energy model of the network component has been extensively studied in related work. To make the tool lightweight to developers, our approach takes an approach similar to Dong and colleagues' work [9] that adopts a linear model based on the total bytes transmitted. The model is shown in Equation (4), where P_n is the coefficient which represents the energy consumption per unit byte transmitted through the network component and B is the value of total bytes transmitted by ads.

$$E_{network} = P_n B \tag{4}$$

Display Energy Model:
The display model in our approach takes the screenshots of ads as input to calculate the display energy of mobile ads. It is defined by Equation (5), where $E_{display}$ is the display energy of the ads in the captured screenshots during the workload replay. This equation will just capture all the

energy consumption related to mobile ads. For either version, according to Equation (5), the display energy is the sum of all screenshots' energy, which is the product of the screenshot's power ($P(s)$), displayed time ($T(s)$), and s is the screenshot. Equation (6) shows that each screenshot's power is the sum of a cost function C, whose input is each pixel's RGB value. Prior work has shown that the power consumption of a pixel is based on its RGB value [10, 30], so the general form of the cost function C is shown in Equation (7). R, G, and B represent the red, green, and blue components of a pixel's color, respectively.

$$E_{display} = \sum_{s=0}^{n} P(s)T(s) \tag{5}$$

$$P(s) = \sum_{k \in |s|} C(R_k, G_k, B_k) \tag{6}$$

$$C(R, G, B) = rR + gG + bB + c \tag{7}$$

3.2.2 *Removing Ads*

To isolate the runtime ad information, our approach removes all ad related invocations in the original app to generate a no-ads version. To do so, we analyze the bytecode of each class of the app and identify ad related invocations by matching the signature of target methods with that of known ad networks. Ad related invocation signature can be collected by referring to the API documentation of these ad networks. For each ad related API invocation identified, we rewrite executable files to remove the invocation and repackage all files into an instrumented APK.

3.2.3 *Collecting Ad Information*

To estimate the energy cost, our approach needs to run workloads and measure ad related metrics. In this step, our approach takes both the with-ads and no-ads version of an app, and the workloads provided by the developer as input, and the output is the ad related runtime information, which includes the CPU time at different frequencies, the network traffic, and the ad screenshots with timestamps.

To collect the above ad information, our approach measures the corresponding metrics on both the with-ads and no-ads version of the app and then subtracts the metric of the no-ads version from the with-ads version. These metrics are measured under the same workload execution for each of subject apps. Specifically, we calculate the network usage of ads by taking the total bytes transmitted by the no-ads version of an app from the total bytes transmitted by the with-ads version of the app. We collect the system level information (e.g., CPU and memory) for both versions of an app and use their difference as ad related performance. We also take two sets of screenshots, one for the with-ads version, and the other no-ads. Then we collect all RGB values of each set of screenshots, and use them as input to calculate the display energy.

4. TESTING PHASE ENERGY MODEL GENERATION

In this section, we discuss how to generate the models for different devices. We generally expect such models to be provided as part of an SDK. So the process outlined in this section is for whoever is creating the SDK and not the end developer. To obtain an energy model, a power meter is needed to measure the power and energy consumption of the target device. At the high level, the coefficients of the energy model could be calculated by measuring the workloads and energy consumption of a set of benchmarks. There are two major steps to build the energy model. In the first step, our approach generates the display energy model. In the second step, it uses the display energy model to isolate the network and system energy from the display energy, and then build the other two models.

4.1 Display Energy Model

To generate the display model, our approach takes the same approach as in our previous work, *dLens* [30]. On the high level, our approach takes a set of screenshots as input and generates the coefficients of r, g, b, and c in Equation (7). The screenshots in the set are solid-colored images with different intensities for one color component (e.g., R) while holding the other two color components at zero. For each color component, there are 16 images whose intensity ranges from 0x0F to 0xFF. With this set, our approach builds the display energy model with the following steps: first, our approach measures the power consumption of a completely black screen to define the baseline power usage of an active screen. Second, our approach measures the power consumed by displaying each screenshot in the set. Third, our approach subtracts the black screen power from these measurements to obtain the power consumption for each R, G, and B component. After that, our approach applies a gamma correction to RGB values. Finally, our approach performs linear regression to determine the coefficients of Equation (7).

4.2 System and Network Energy Model

To generate the system and network model, our approach takes the display energy model and a set of with-ads market apps as input and calculates the coefficients of P_f in Equation (3) and P_n in Equation (4).

In the first step, our approach generates the no-ads version of benchmarks following steps described in Section 3.2.2. Then it runs both the with-ads and no-ads version of benchmarks through an automated UI testing framework, such as Monkey in Android SDK [6], and measures the energy consumption during the execution. At the same time, our approach also collects ad related information of benchmarks (as described in Section 3.2.3).

After collecting the energy cost of each of our benchmarks, our approach first removes the calculated display energy from the overall measured energy cost of the benchmarks. Then, it subtracts the energy consumption of the no-ads version of benchmarks from the with-ads version of benchmarks to get the estimated combined ad related energy of the network and system. For each of the benchmarks, our approach takes these numbers and uses them to define a system of equations that will be solved to find the network and system energy individually. To do this, our approach uses robust regression techniques [2] instead of the normal linear regression technique because they reduce the inaccuracy introduced by outliers representing system noise and measurement errors.

Table 1: Subject applications

Name	Category	Size (MB)
Restaurant Finder	travel & local	3.7
Smileys for Chat	communication	15.9
Polaris Navigation GPS	travel & local	7.8
Public Radio & Podcast	news & magazines	3.2
English for kids learning free	education	8.0
Lomo Camera	photography	29.5
Arcus Weather	weather	2.8
The Best Life Quotes	entertainment	2.6
Player dreams	music & audio	1.9
VLC Direct Streaming Pro Free	media & video	2.3
Translator Speak & Translate	travel & local	5.3
7Zipper	communication	7.6
Radaee PDF Reader	productivity	4.6

4.3 Generating the Model Without Power Meters

Even without the energy models, developers could still use our approach, but would have to generate the energy models on their own. To do this, developers could use the battery APIs provided by the Android OS to approximate energy consumption. Related work has demonstrated the feasibility of this technique for getting rough estimates of energy [34]. However, such a technique is less precise than hardware based power measurement devices and would impact the overall accuracy of the techniques.

5. EVALUATION

We conduct our evaluation on whether the energy cost of mobile ads, can be accurately estimated, without direct measurement, as a function of other easily collected ad metrics. In this study, we evaluate the completely static approach described in Section 3.1 and the approach based on collecting runtime metrics, such as the network and CPU usage, described in Section 3.2.1. The evaluation is conducted through two research questions below.

- RQ 1: How accurate is the completely static model?
- RQ 2: How accurate is the runtime metric based model?

5.1 Subject Applications

To evaluate our approach, we used a set of real-world subject applications, whose information is listed in Table 1. For each of these apps, we manually generated workloads and recorded all interactions with the RERAN tool [12], which enables us to replay the same workload on both the with-ads and no-ads version of an app. The runtime metrics of ads captured during the workload replay are used to generate the runtime model.

5.2 Implementation

We implemented a prototype version of our approach. Specifically, we first converted an app's APK file into corresponding Java bytecodes. We then wrote the instrumentation code to remove ad related bytecodes using the ASM library [1]. Note that to control the experimental variance, we follow the same steps but without removing ad invocations to generate the with-ads version. Hence, after the instrumentation, each app has two versions: with-ads and no-ads. We used the */proc* files to monitor the CPU time at different frequencies, the *top* command to record the memory usage, and the *tcpdump* tool to monitor the network usage. For the measurement of the display energy, our approach modified the *Ashot* [3] tool to capture screenshots of both the with-ads and no-ads version of the app. *Ashot* took the screenshot whenever there was a change on the user interface of the screen. For each screenshot s_i, our approach also attached the time spent on the screenshot, T_i, with it.

To measure the energy and provide the ground truth for calculating the accuracy, we ran all of our applications on a Samsung Galaxy SII smartphone and measured the energy consumption through the Monsoon Power Meter [5]. Finally, we performed the robust linear regression analysis using the *rlm* function from the R toolkit [4], and the cross validation for RQ2 using Weka [14].

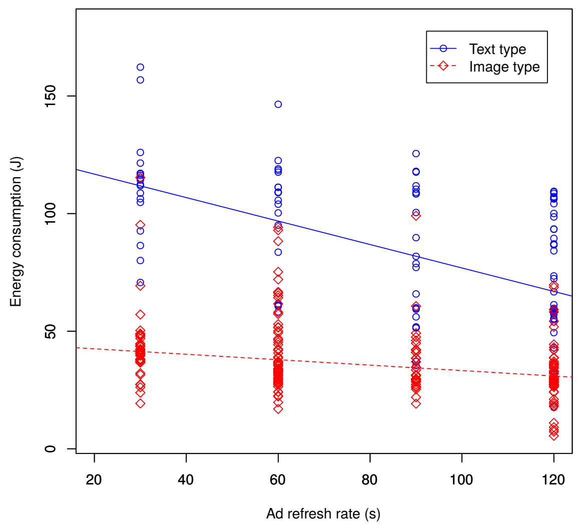

Figure 2: Energy of the static model versus ad refresh rates.

5.3 RQ 1: Accuracy of the Static Model

To evaluate the accuracy of the static model, we instrumented the market apps to replace the original *AD_UNIT_ID* with one under our control, which enabled us to modify the TYPE and RRATE of the ads. This was done through reverse engineering and bytecode rewriting techniques. We varied the RRATE over the four standard refresh values: 30, 60, 90, and 120 seconds. In the meanwhile, we fixed the TYPE and SIZE to control variables for the model generation. We repeated this experiment for each possible value of TYPE (i.e., image and text). We did not vary SIZE, as this would move UI elements on the page due to the change of ad size, making it difficult to automatically rerun the recorded workloads.

To measure the predictive energy of the model, we built the ground truth as follows: first, we randomly chose a UI that contained an ad; second, we set a specific configuration for the ads, this configuration included SIZE, TYPE, and RRATE; third, we displayed both the with-ads and no-ads version of the selected UI for 250 seconds and measured their energy consumption; fourth, we subtracted the energy of the with-ads version from that of the no-ads version, and used this value as the ground truth of the energy consumed by ads in the static model generation.

Based on the SIZE, RRATE, and the energy consumption collected in the above measurement, we built a regression model for these collected data points. The result for the impact of RRATE on the energy consumed by ads is shown in Figure 2, where the x-axis is the value of RRATE and the y-axis is the corresponding energy. Each data point in Figure 2 is the energy measured for the apps at different RRATE values. After applying the robust linear regression analysis, there are two fitted lines, which correspond to two different TYPE values. The residual standard error for the image based ads was 0.03591 and 0.1198 for the text based ads. Our approach is able to estimate the energy consumption to within 29% and 33% of the ground truth for the text and image type, respectively. On average, the text-type ads consume more energy than the image-type ads at the same refresh rate. One possible reason is that the text ads in general have HTML/CSS/JavaScript code embedded, which is used to monitor the real-time event handler and consumes more energy, while the image ads have static images with corresponding URLs linked, which require less monitoring resources. We further investigated the average energy consumption at each RRATE value, and found that for each ad type, the average energy had a strong linear relationship. The error rates in the fitted results were 1.8% and 3.1% for the image and text type, respectively.

The linear relationship shown in Figure 2 demonstrates reasonable accuracy of the static model. It provides high level information about ads, and allows developers to get early feedback about the potential energy cost of ads, even without a working implementation.

5.4 RQ 2: Accuracy of the Runtime Metric Based Model

To measure the accuracy of the runtime metric based model, we used Error Estimation Rate (EER), which is defined as the percent difference between the estimated values and the ground truth. In the experiments, we collected a total of 77 execution traces of mobile ads. We then applied linear regression analysis with 10-fold cross validation. The correlation coefficient for the output was 0.964, which shows a strong linear relationship in the model. After applying the generated model to estimate the energy cost of all data points, the overall average error rate was 14%. This value is roughly comparable to the accuracy results (e.g., 10% in [15, 19]) obtained by other energy estimation techniques. However, note that these techniques do not calculate ad related energy information for the reasons described in Section 2. In the fitted full model, CPU consumed the most amount of energy in ads, while other components consumed relatively little. The results indicate that we can obtain accurate energy information of ads efficiently without requiring external equipment or extensive developer effort.

6. RELATED WORK

Our previous work [13] conducted an empirical study of five hidden costs associated with mobile ads. We found that the cost of ads in terms of performance, energy, bandwidth, and maintenance were substantial. Moreover, these hidden costs could impact the app ratings. Wei and colleagues [31] quantified the network usage and system calls related to mobile ads. Vallina-Rodriguez and colleagues [29] analyzed the characteristics of mobile ads from traffic traces of a major European mobile carrier with over three million subscribers. However, these approaches did not address how to build models to automatically predict ad energy cost.

Estimating and improving the energy consumption of mobile apps has been extensively studied in related work. *Bouquet* [20] used static program analysis techniques to reduce the energy of mobile apps by automatically bundling small HTTP requests. Other techniques [21, 30] detect and improve the display energy consumed by apps. Some other work [18, 25, 33] also proposed techniques to predict the energy consumption of mobile apps. The above work focused on models to predict the energy usage of mobile apps as a whole and cannot handle the specific issues related to ads.

Some other research has been done concerning mobile ads. One important topic is to optimize the ad cost [17, 23, 29]. Rasmussen and colleagues [27] analyzed the effects of advertisement blocking methods on energy consumption. Another significant topic is detecting security problems in mobile ads. Pearce and colleagues [26] presented *AdDroid* to separate privileged advertising functionality from host applications. *DECAF* [22] detected violations of ad placement and content policies. *PUMA* [16] detected ad fraud by comparing the `WebView` size with the minimum allowed ad size in mobile apps. Mobile ads' impact on the host app is a main concern for developers. Ruiz and colleagues [24, 28] studied ad libraries and their impact on app ratings. Xu and colleagues [32] conducted surveys to identify factors that influenced consumers' response to mobile ads.

7. CONCLUSIONS

In this paper, we presented new techniques to predict the energy cost of mobile ads. The approach proposes two models to help developers understand the energy cost of mobile ads. In our evaluation, we showed that the static model is accurate to estimate the energy cost of ads to within 31% of the ground truth, and the runtime model is accurate to estimate the runtime energy cost of ads to within 14% of the ground truth. Overall, the results are promising, and show that our techniques could potentially help guide developers in how to use mobile ads in their apps in a way that reduces their energy consumption.

In future work, we plan to improve the accuracy of ad energy estimation by incorporating more ad relevant metrics into both models, and developing new models which focus on ads from a different perspective (e.g., monitoring ad threads and API execution). More broadly, our work enables the research community to predict ad energy consumption, and develop techniques that leverage this capability to optimize ad usage.

Acknowledgment

This work was supported by National Science Foundation grant CCF-1321141.

8. REFERENCES

[1] http://asm.ow2.org/.
[2] http://en.wikipedia.org/wiki/Robust_regression/.
[3] http://sourceforge.net/projects/ashot/.
[4] http://www.r-project.org/.
[5] Monsoon Solutions. Power monitor. https://www.msoon.com/LabEquipment/PowerMonitor/.
[6] UI/Application Exerciser Monkey. http://developer.android.com/tools/help/monkey.html/.

[7] Driven by Facebook and Google, Mobile Ad Market Soars 105% in 2013, March 2014.

[8] J. P. M. Consulting. IAB Interactive Advertising Outlook 2014, January 2014.

[9] M. Dong and L. Zhong. Chameleon: A Color-adaptive Web Browser for Mobile OLED Displays. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, 2011.

[10] M. Dong and L. Zhong. Power Modeling and Optimization for OLED Displays. *Mobile Computing, IEEE Transactions on*, 2012.

[11] I. Gartner. Gartner Says Mobile Advertising Spending Will Reach 18 Billion in 2014, January 2014.

[12] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein. RERAN: Timing- and Touch-Sensitive Record and Replay for Android. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, 2013.

[13] J. Gui, S. Mcilroy, M. Nagappan, and W. G. J. Halfond. Truth in Advertising: The Hidden Cost of Mobile Ads for Software Developers. In *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, May 2015.

[14] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[15] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan. Estimating Mobile Application Energy Consumption using Program Analysis. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, May 2013.

[16] S. Hao, B. Liu, S. Nath, W. G. Halfond, and R. Govindan. PUMA: Programmable UI-Automation for Large Scale Dynamic Analysis of Mobile Apps. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 204–217. ACM, 2014.

[17] A. J. Khan, V. Subbaraju, A. Misra, and S. Seshan. Mitigating the True Cost of Advertisement-Supported "Free" Mobile Applications. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*, 2012.

[18] D. Li, S. Hao, J. Gui, and W. G. Halfond. An Empirical Study of the Energy Consumption of Android Applications. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, September 2014.

[19] D. Li, S. Hao, W. G. Halfond, and R. Govindan. Calculating Source Line Level Energy Information for Android Applications. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, July 2013.

[20] D. Li, Y. Lyu, J. Gui, and W. G. Halfond. Automated Energy Optimization of HTTP Requests for Mobile Applications. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, May 2016.

[21] 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 (ICSE)*, pages 527–538. ACM, 2014.

[22] B. Liu, S. Nath, R. Govindan, and J. Liu. DECAF: Detecting and Characterizing Ad Fraud in Mobile Apps. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014.

[23] P. Mohan, S. Nath, and O. Riva. Prefetching mobile ads: Can advertising systems afford it? In *Proceedings of the 8th ACM European Conference on Computer Systems*, 2013.

[24] I. Mojica Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. Hassan. Impact of Ad Libraries on Ratings of Android Mobile Apps. 2014.

[25] A. Pathak, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app? Fine Grained Energy Accounting on Smartphones with Eprof. In *Proceedings of the 7th ACM european conference on Computer Systems*, 2012.

[26] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner. Addroid: Privilege separation for applications and advertisers in android. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, 2012.

[27] K. Rasmussen, A. Wilson, and A. Hindle. Green Mining: Energy Consumption of Advertisement Blocking Methods. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software*, 2014.

[28] I. J. M. Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. E. Hassan. On Ad Library Updates in Android Apps.

[29] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, K. Papagiannaki, H. Haddadi, and J. Crowcroft. Breaking for Commercials: Characterizing Mobile Advertising. In *Proceedings of the 2012 ACM conference on Internet measurement conference*, 2012.

[30] M. Wan, Y. Jin, D. Li, and W. G. J. Halfond. Detecting Display Energy Hotspots in Android Apps. In *Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, April 2015.

[31] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. ProfileDroid: Multi-layer Profiling of Android Applications. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, 2012.

[32] D. J. Xu. The influence of personalization in affecting consumer attitude toward mobile advertising in China. *Journal of Computer Information Systems*, 2006.

[33] 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*, 2012.

[34] 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*, 2010.