Improving Energy Consumption in Android Apps

Carlos Bernal-Cárdenas
The College of William and Mary, Williamsburg, VA, USA
cebernal@cs.wm.edu

ABSTRACT

Mobile applications sometimes exhibit behaviors that can be attributed to energy bugs depending on developer implementation decisions. In other words, certain design decisions that are technically "correct" might affect the energy performance of applications. Such choices include selection of color palettes, libraries used, API usage and task scheduling order. We study the energy consumption of Android apps using a power model based on a multi-objective approach that minimizes the energy consumption, maximizes the contrast, and minimizes the distance between the chosen colors by comparing the new options to the original palette. In addition, the usage of unnecessary resources can also be a cause of energy bugs depending on whether or not these are implemented correctly. We present an opportunity for continuous investigation of energy bugs by analyzing components in the background during execution on Android applications. This includes a potential new taxonomy type that is not covered by state-of-the-art approaches.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms

User interfaces

Keywords

Energy consumption, mobile applications, empirical study

1. MOTIVATION

Due to the large volume of mobile applications (apps), now numbering more than 1.5 million according to Appbrain [2], and the resource constrained environment of modern smart devices energy consumption is a popular research topic. Apps address users' needs through the implementation of numerous features aimed at making life easier. However, these features can sometimes implement more functionality than they should. This leads to apps that drain battery unnecessarily due to the overuse of device resources, and the incorrect implementation and usage of components. In general, analyzing different aspects about implementations of Android apps can lead to a better understanding of potential energy bugs. Our approach focuses on the analysis of GUI components and their usages, as well as hidden behaviors that are not triggered by users.

ESEC/FSE'15, August 30 – September 4, 2015, Bergamo, Italy
ACM. 978-1-4503-3675-8/15/08
http://dx.doi.org/10.1145/2786805.2807558

2. RELATED WORK

Previous work has focused on detecting energy bugs in mobile applications using several different strategies:

API Calls: Vekris *et al.* [12] presents an approach focused on detecting the usage of the WakeLock API that forces components to no-sleep (e.g. keeping the screen awake) by performing an inter-procedural data flow analysis. Zhang *et al.* [13] compares the similarities of energy bugs for Android, iOS, and Windows platforms in order to check whether the approach for handling energy bugs in one platform works for the other. Finally Linares-Vasquez *et al.* [6] investigate patterns of energy-greedy API calls that consume unnessecary energy in Android applications.

Bytecode: Liu *et al.* [8] presents a work that analyzes the bytecode of Android apps in order to check whether a listener for a sensor usage does not unregister it. This work systematically diagnoses inefficient usages of sensor data.

Data flow: Pathak *et al.* [10] perform a data flow analysis to look for no-sleep code paths detecting possible energy bugs. In their approach, detecting how wakelocks are used helps to mitigate unnecessary resource consumption.

Objects: Zhang *et al.* [14] present an approach called ADEL that automatically detects energy leaks using dynamic taint-tracking analysis. This approach focuses on unnecessary network communication.

Third Party Libraries: Rasmussen *et al.* [11] analyze energy consumption of ads on mobile apps. It uses ad-blocking software to investigate the gain that these types of programs introduce in terms of energy. In addition they compared different types of ad-blocking programs resulting in many cases that lead to higher energy consumption.

3. APPROACH

This section presents the work of improving energy consumption in Android apps by performing two different types

of analysis: First, on GUI components that are instantiated on the screen minimizing the energy consumed by SUPER AMOLED screens. Second, an ongoing work of analyzing components in the background during execution of Android apps.

3.1 GUI ANALYSIS

Our approach is called GEMMA (Gui Energy Multi-objective optiMization for Android apps) and generates color composition solutions, which optimizes the energy consumption and contrast while staying consistent with the original palette using multi-objective optimization [7]. The GEMMA approach is divided into multiple phases:

Estimating Power Consumption: This phase builds a power model for a SUPER AMOLED screen (1080×1920 pixels) of a Samsung Galaxy S4. We collected energy consumption using a Power Monitor [3] on the idle state of the screen for primary and black colors changing the RGB component level from 1 to 255 for 30 seconds at each. Using these measurements we are able to build a linear RGB (i.e. read, green, blue) function that simplifies the construction of the model [5]. To compute the total energy consumed for a specific screen we need to sum the power consumption values for each GUI component according to pixel's position (e.g. (x, y)), linear RGB values.

Extracting Color Composition: The next phase dynamically extracts the color composition of GUI elements on the screen. This uses a tool provided in the Android SDK called UIAutomator [4] in order to detect the position of the components and assign the pixels that correspond to each GUI element. We use two structures to achieve this; bag-of-color-pixels (BOCP) and bag-of-color-components (BOCC). BOCP is a set of pixels with the same color and BOCC is a set of components with the same GUI color. We automatically take a screen shot, then extract the tree representation from the xml using UIAutomator tool which includes dimensions, properties and location. Finally we traverse the tree using a bottom-up approach to assign the pixels. This means starting from the biggest component we start assigning pixels to its children and then repeat the process with each child.

Then we extract the histogram of each component with the top colors avoiding gradients, fonts, and shadows. Finally using the centroids of the histogram we apply quantization by assigning the colors to the closest quantized color to the original one. As a result we obtain an association of pixels for each component(i.e. BOCP) and the BOCC which is the mapping between quantized color and components associated to pixels with this color.

Multi-objective optimization: This phase is based on a genetic algorithm technique called Nondominated Sorting Genetic Algorithm (NSGA)-II. It produces a pareto-optimal set of solutions that minimizes the energy consumption, maximizes the contrast, and minimizes the distance between the chosen colors by comparing the original palette. In addition to avoid trivial solutions (i.e. low contrast) we added a constraint that influences the tournament selection of the NSGA-II.

3.2 BACKGROUND ANALYSIS

In addition we want to analyze the behavior of apps that execute components in the background in order to identify possible causes of energy bugs when implementing features on Android apps. According to the official Android API documentation [1], there are different components that are able to run in the background:

Services: This is an application component that can assume a longer-running time when the user is not interacting with it. This type of processes keeps running while the device has enough memory to execute it. Services usually run actions that users care about like playing music.

Broadcast Receivers: This type of process exists at any time during application execution. This receives intents and transforms them into actions that usually notify the user when a specific event occurs.

Threads: This type of process is used when a developer needs to execute actions in the background. However, they should terminate instantaneously in order to avoid blocking the main UI thread.

Workers: This is a type of thread that is used in the case a developer needs to perform operations that take a long period of time without blocking the main UI thread. In addition, when using multiple threads, handlers should be used to send a message to communicate between threads.

AsyncTask: This allows worker threads to enable the proper manipulation of the main UI thread. Moreover, it helps to perform background updates to the GUI without manipulating handlers or threads.

These are the different ways for implementing components that execute actions in the background that could lead to energy bugs if the implementation is not appropriate. According to Pathak *et al.* [9, 10] taxonomy there is no classification that fits on this types of energy bugs. Pathak *et al.* focused on analyzing wakelocks however this is not the only case of apps consuming unnecessary resources.

4. RESULTS AND CONTRIBUTIONS

We have used GEMMA in generating solution palettes for 25 Android apps that reduces the energy consumption. We surveyed 85 mobile user developers and three companies, one manager for each company. As a result, solutions with highest energy consumption are not preferred by the participants, because these solutions involve palettes that contains only dark colors in most of the cases. However, GEMMA generates solutions acceptable by participants that also consumes 42% less energy on average than the original composition. Managers of the three different companies agree that this can be used in future releases of its own products, confirming that GEMMA should be considered for developers in companies for commercial products. Using a multi-objective approach GEMMA is able to balance the energy consumption, contrast, and closeness to the original composition generating solutions palettes that are energy-friendly. The lack of analysis in the background execution of actions on Android apps gives an opportunity to investigate the causes that can help to detect unnecessary usage of resources. Performing such an analysis can potentially help to locate the origins of energy bugs other than those previously explored namely: analyzing usage patters, wakeLocks, data flow, bytecode, and objects.

5. REFERENCES

[1] Android documentation. http://developer.android.com/guide/components/processes-and-threads.html.

[2] Appbrain. http://www.appbrain.com/stats.

[3] Monsoon-solutions. power monitor. http://www.msoon.com/LabEquipment/PowerMonitor/.

[4] UIAutomator. https://developer.android.com/tools/testing-support-library/index.html#UIAutomator.

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

[6] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk. Mining energy-greedy api usage patterns in android apps: an empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 2–11. ACM, 2014.

[7] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk. Optimizing energy consumption of guis in android apps: A multi-objective approach. In *Proceedings of 10th Joint Meeting of the European Software Engineering Conference and the 23rd ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2015.

[8] Y. Liu, C. Xu, and S. Cheung. Where has my battery gone? finding sensor related energy black holes in smartphone applications. In *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on*, pages 2–10. IEEE, 2013.

[9] A. Pathak, Y. C. Hu, and M. Zhang. Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, page 5. ACM, 2011.

[10] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff. What is keeping my phone awake?: characterizing and detecting no-sleep energy bugs in smartphone apps. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 267–280. ACM, 2012.

[11] 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*, pages 38–45. ACM, 2014.

[12] P. Vekris, R. Jhala, S. Lerner, and Y. Agarwal. Towards verifying android apps for the absence of no-sleep energy bugs. In *Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems*, pages 3–3. USENIX Association, 2012.

[13] J. Zhang, A. Musa, and W. Le. A comparison of energy bugs for smartphone platforms. In *Engineering of Mobile-Enabled Systems (MOBS), 2013 1st International Workshop on the*, pages 25–30. IEEE, 2013.

[14] L. Zhang, M. S. Gordon, R. P. Dick, Z. M. Mao, P. Dinda, and L. Yang. Adel: An automatic detector of energy leaks for smartphone applications. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 363–372. ACM, 2012.