Mobile Application Power Profiles CS6204 Final Report

Ethan Holder
Department of Computer Science
Virginia Tech
eholder0@vt.edu
Due Thursday, December 11th at 11:59pm

***Abstract*—Mobile devices play an ever increasing role in today's always connected world, from handheld devices to the next potential super computer. However, numerous challenges plague developers wishing to capitalize on the mobile application (app) marketplace. Chiefly among these is how to effectively create an app that will generate sustained interest across a plurality of platforms. Software engineering has given us several development patterns that fall along a spectrum of tradeoffs, with ease of development and maintenance on one end and broadly defined user appeal on the other, such as writing applications utilizing PhoneGap [2, 12] or with the purely native Android platform [3]. However, comparisons of these patterns chiefly examine performance (speed, latency) and aesthetics as critical measures that affect which alternative should be utilized in a given scenario. In this paper I will consider a tradeoff that has received less attention lately from the application development perspective: energy efficiency. By profiling energy consumption of these classes of apps at a fine-grained (component-by-component) level, future developers and researchers can better understand where energy is being lost and thus implement improvements.**

I. Introduction

Energy efficiency and profiling have previously been studied in numerous published works, including [6, 11] within this research group. The following subsections outline the purpose of this study. First, I present the motivation for this work in I-A. Next, I outline the research questions/goals of this project in I-B. Finally, I briefly describe the layout of the remainder of the paper in I-C.

A. Motivation

Energy efficiency has long been a staple in computer systems and has recently emerged as a hot topic in distributed systems, where energy is at a premium. The mobile space is no different; in many ways mobile devices have an even deeper reliance on energy, given battery concerns on each system. In order to improve existing development patterns, we must first understand what the current tradeoffs are along this dimension. However, understanding this tradeoff is very difficult because of the layers of abstraction between the components consuming energy and the source code triggering those components. Previous work in profiling energy consumption at a fine-grained (component-by-component in correlation with source code) level has shown exactly how difficult yet impactful this analysis can be in [6, 10, 11, 19, 20, 24]. The purpose of this project is thus to expand on previous work in profiling the energy consumption of varying classes of apps by performing an analysis on a modern tablet, demonstrating the current levels of energy consumption and what improvements can be made to existing systems.

Dr. Cameron's research group (whom this project collaborates with) has already implemented similar profiling approaches on traditional computer systems and ported their approach to mobile devices. Past work on a Qualcomm DragonBoard [25] thus serves as the basis and the reason why this project will succeed. This project will extrapolate from that work on the DragonBoard to profile a similar Android system in the Asus Transformer Pad (TF300T) [5]. By instrumenting analysis on this device and more in the future, I hope to form a consensus that can generalize to further devices.

B. Research Questions/Goals

To conduct a successful project, some research questions must be answered or goals be met in order to guide the work. This one aims to address a number of questions, all of which are unified by a single overarching goal. The key research questions are listed below:

- **A)** Firstly, how can PowerPack be extended to a consumer device, such as the Asus TF300T, utilizing the power rails coming from the internal battery?
- **B)** What is the basic component-level power profile of that device?
- **C)** How do apps developed for a translation platform compare to those developed for other platforms in terms of energy consumption?

The larger goal is therefore **to analyze current mobile energy consumption across various development schemes**. The implementation section III will address exactly how this system was created. Based on that, the experimental setup IV and results V sections will then address the last two research questions, while the goal should be considered throughout.

C. Layout

The remainder of this paper is structured as follows. First, related work and some background information is highlighted in II. Next, I explain the implementation of energy profiling on the Asus tablet in III. This leads into the experimental setup section which shows how I collected data on the system in IV. The results in V then show exactly what I uncovered through the implementation and experiment. The analysis follows after the results and discusses what the results mean in VI. Finally, I present my conclusions for this project in VII.

II. Related Work

Previous work in Dr. Cameron's SCAPE lab pioneered the PowerPack approach [11] for energy profiling, although not for mobile device profiling specifically. PowerPack examines the amount of energy flowing through the power lines of the system (cables from the power supply on desktop and battery rails on mobile devices) during interaction by externally tracking software calls throughout [1]. By associating the difference in energy consumption from one unit of time to another with the corresponding software calls that executed in that span, the authors are able to show exactly which calls trigger increases or decreases in power consumption and which components are consuming it. This approach and its extensions [27, 17] form the basis for my project.

Other work by Dr. Cameron's group showed some basic profiling of Android devices already and provided a hierarchy of suggestions for future tooling to minimize energy consumption [6]. This work was a basic proof of concept in this lab for carry out profiling on an Android device with significant results. Based on their information, further software tuning can be performed at the application or operating system level to improve scheduling and thus increase idle time.

Several outside works have performed energy profiling on mobile devices in some sense already. Both Pathak et al. works [20, 19] demonstrate the *eprof* system of energy profiling. In utilizing *eprof* along with their energy bundles mechanism, the authors are able to refactor the source code of well-known target applications to save up to 65% in energy consumption. Both of these works were performed under Dr. Y. Charlie Hu at Purdue University.

Tangential works by Dr. Hu's group have leveraged energy profiling approaches in order to detect energy-based bugs in smartphones, such as the "no sleep" bug that can keep a device's hardware components awake indefinitely thus draining energy [21, 15]. Other research [29] has also attacked the "no sleep" bug by profiling and examining the Android WakeLock API. Vekris et al. was able to verify 145 out of 328 apps did not have the bug, while the remainder were either confirmed to have it or present a warning.

Beyond the SCAPE lab and Dr. Hu's group, the Qian et al. paper (under Dr. Oliver Spatscheck) [24] demonstrates another means of profiling energy consumption, this time based on layers of interaction rather than calls down to hardware layers. The authors consider hardware components, low-level systems calls, and high-level source code calls in profiling where energy is consumed in a given app. The authors profiled several well known applications, such as Pandora, Fox News, and BBC news among others, and were able to identify problem areas then infer a reasoning for what causes those areas to exhibit such high energy consumption.

III. Implementation

In order to measure the energy consumption and thus profile an Android tablet, we have to tap into the power flowing through the device [2]. The simplest, most portable method our group could find to measure the energy while it is flowing is the power coming out of the battery. This avoids modifying and potentially damaging the motherboard. In our test system, an Asus TF300T [5], the battery has an 8-line ribbon cable attaching itself to the motherboard as shown in figure 1. Since we cannot obtain proprietary information from Asus to identify what each line represents, it is our assumption that each line (except the ground) provides power to some subset of the components on the motherboard. This could be as simple as 1 line for each of 6 major components (such as CPU, GPU, RAM, storage, screen, and wireless) and 1 for the rest of the component, or the lines could represent a significantly more complex structure. For now, we assume the simple version and will instrument across each line to determine an answer.

Our original intent was to purchase or fabricate a ribbon cable that could adapt between the connectors on

[1] More information about iterations of PowerPack can be found at the SCAPE lab page: http://scape.cs.vt.edu/software/powerpack-3-0/

[2] The implementation of this system was performed in conjunction with members of the SCAPE lab who are not part of this publication. Therefore, "we" is used throughout to reflect the group's effort.

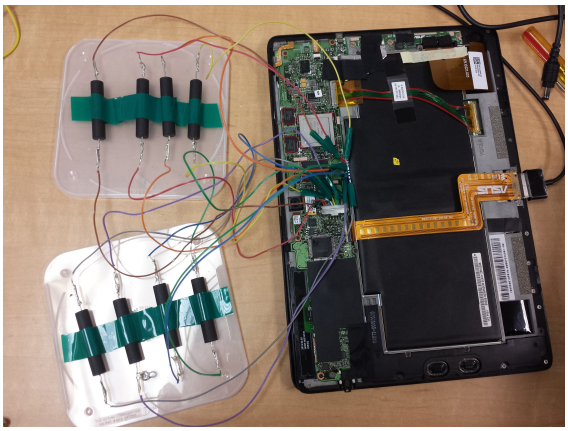

Fig. 1: Soldered wires and resistors replacing 8 line battery ribbon cable.

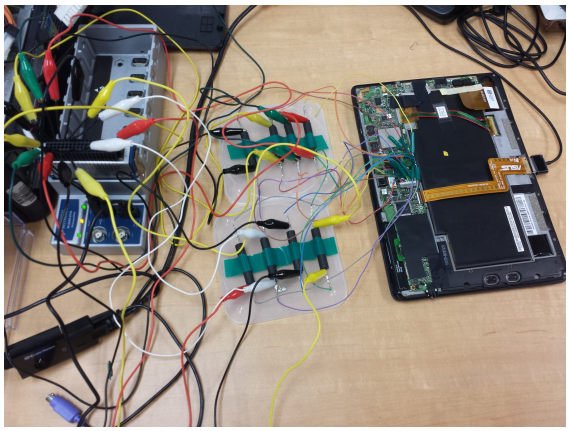

Fig. 2: Instrumentation attached to PowerPack measuring each battery line.

the motherboard and the battery. By doing this, we could instrument PowerPack on our own replaceable cable without modifying the original system. Unfortunately, due to the size and shape of the connectors on the battery and motherboard ends, the proprietary nature of the hardware, and most importantly, the time and monetary constraints of this semester, we were unable to purchase an equivalent ribbon cable or even suitable pins and connectors to fabricate our own. However, we found out that replacement batteries for the TF300T include their own ribbon cable to connect to the motherboard. Therefore, we determined that we could safely deconstruct the cable on our existing battery to instrument across those wires without adversely affecting the rest of the system. This solution is not ideal and is less portable than we originally envisioned, but it is sufficient for the purposes of this project. We leave further investigation of our original solution to future work.

Figure 1 shows how we were able to instrument across the battery's ribbon cable. We cut and split the existing cable before soldering separate extension wires onto each end. These extension wires gave us much more room for instrumentation. We then soldered the loose ends of each extension wire onto a $<0.001\Omega$ resistor, matching the ends so as to preserve the original pinout from the battery. The resistor gives us something solid to attach PowerPack to while limiting the impact to the existing system. Figure 2 shows how we attach the PowerPack instrumentation to the expanded ribbon cable. The alligator clips are attached to capture software that independently monitors each matched in and out side from the resistors, determining the voltage.

IV. Experimental Setup

As mentioned above, the system instrumented on was an Asus TF300T [5] which was running Android [4]. Table I presents more information on its configuration. In order to compare applications running on the TF300T, I first must form a basic power profile for its execution based on each component. To do so, I captured the voltage of every battery rail (the wires of the ribbon cable instrumented across) while the tablet is idle (unlocked and awake but not accessing any application) as well as during a benchmark. The idle run gives a baseline of what voltage is being drawn while the tablet is not under stress. The conjunction of runs presents a basic outlook of which rail corresponds to which hardware component(s) based on what components the benchmark stresses at each time versus what isn't stressed during idle time.

Manufacturer	Asus
Friendly Name	Transformer Pad
Product Name	TF300T
Storage	32GB
OS Version	Android 4.2.1
OS Name	Jelly Bean
Benchmark Name	AnTuTu
Benchmark Version	5.3

TABLE I: Tablet and benchmark configurations.

The idle run shows sample data over 15 seconds. PowerPack currently collects 50 (± 1) measurements per second, so even this short time presents over 700 samples. The benchmark however takes around 5 minutes to fully execute, so 7 minutes (420 seconds) of sampling is performed to sufficiently capture it all. For completion's sake, I present both the idle and benchmark runs each

while using (approximately full) battery power as well as while plugged in and charged in case there is a difference.

Using the basic energy profile obtained from the idle and benchmark runs, I can then compare the performance of classes of apps to show which class is more energy efficient. To determine which apps to compare, I first selected 5 free apps from the "Top Free Apps Designed for Tablets" category on the Google Play store [13]. These 5 included two games, a social media site application, online radio, and an LED flashlight, and represent apps developed normally. Although I cannot conclusively prove that each was developed for the native platform, I know from examination of PhoneGap's site that they were not developed for PhoneGap but are obviously very popular, so they will serve as well-performing normal apps which could be developed via a variety of methods. To approximately match those, I selected 5 apps developed on PhoneGap for Android [22]. Of these 5, 2 were featured Android apps (to reflect popularity), and the rest were chosen to mimic specific behavior of the normal set (a game, online radio, and an LED flashlight). All applications are listed in table II.

Type	Normal	PhoneGap
Popular1	Candy Crush Soda Saga [16]	World Heritage Calendar [7]
Popular2	Facebook [9]	TripCase Travel Alerts [26]
Game	Trivia Crack [8]	ISIER IQ Test [1]
Online Radio	Pandora [18]	Phonostar Radio-app [23]
Flashlight	Super-Bright LED Flashlight [14]	Brightest LED Flashlight Torch [28]

TABLE II: Applications for testing.

For sampling each application, only trials of natural runtime are presented so as to eliminate potential noise from the OS. Each application was started and allowed to hit a non-loading screen before beginning the sampling. The sampling lasts 3 minutes (180 seconds) in each application, so there are roughly 9000 sample points for each battery rail [3].

V. Results

From the first experiment performed, figure 3 shows the baseline power during an idle run while on battery and outlet power, respectively. It may appear that only two lines each for battery and outlet rails are shown, but actually only those two rails produce any readings under sample. The remaining rails all produced samples so small (<0.1 millivolts) that PowerPack could not detect them and output 0. They are thus overlapping on the graph shown. This reading was verified manually with a simple multimeter to ensure correctness. Because of this, further results omit the remaining 6 rails for aesthetic purposes. In all cases, those rails still produce <0.1 millivolts. On battery power, the first 2 rails registered an average of 1183.91 and 1163.98 millivolts, respectively, over the 15 seconds. Similarly, on outlet power the first 2 rails registered an average of 498.01 and 470.31 millivolts, again respectively.

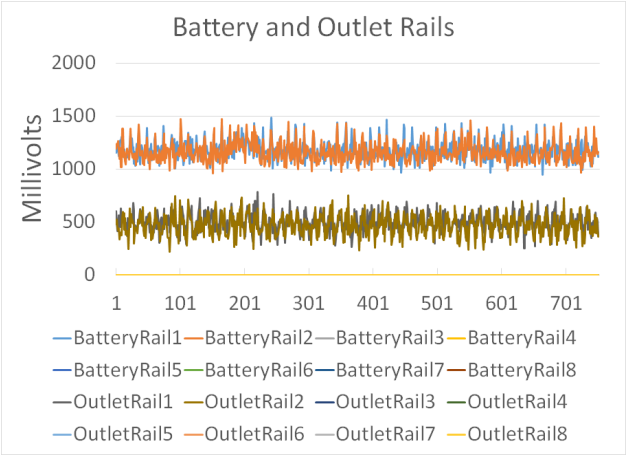

Fig. 3: Millivolts for all 8 rails under battery and outlet power.

The next experiment was to run the AnTuTu benchmark while profiling the power consumed. Figures 4 and 5 show the power in millivolts during the benchmark while on battery and outlet power, respectively. The flat line at the end is when the benchmark completed after roughly 4.5 minutes. Table 6 shows the corresponding scores of AnTuTu under battery and outlet power, respectively. Unfortunately, since only two rails register any voltage at all and those two rails closely mirror each other, it is impossible to tell much if anything about the component breakdown of power here.

Furthermore, the battery and outlet power do not present significantly different results, only outlet power is adjusted lower. I suspect the cause of this is one of the remaining rails serving as communication between the motherboard and battery. In that case, outlet power bypasses the battery and directly supplies the motherboard while the battery only emits enough remaining power to make up for any discrepancy. For brevity, the remaining results thus omit results from outlet power trials and only display battery power trials. Future work may examine the differences shown and investigate alternative causes.

[3]Occasionally PowerPack adds or omits one random sample per second, but the omission was infrequent enough that the total samples were never below 8995 after 3 minutes for each application.

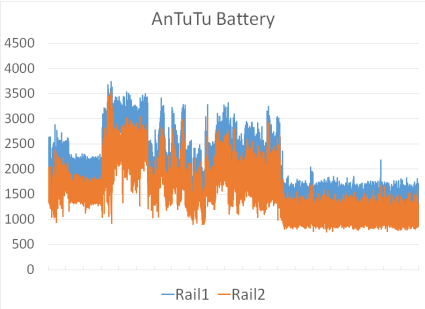

Fig. 4: Millivolts for both rails during AnTuTu under battery power.

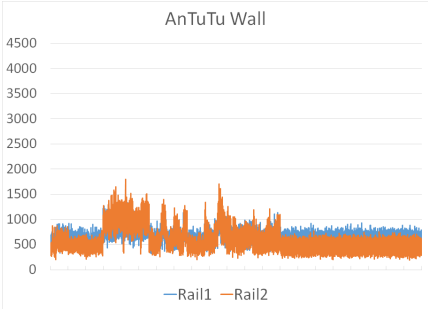

Fig. 5: Millivolts for both rails during AnTuTu under outlet power.

Test	Battery	Wall
Overall	15983	15909
Multitask	2822	2530
Runtime	996	993
RAM Operation	1524	1527
RAM Speed	628	650
4x CPU Integer	1385	1384
4x CPU Float	1554	1546
1x CPU Integer	883	883
1x CPU Float	976	983
2D Graphics	682	716
3D Graphics	3716	3819
Storage IO	407	443
Database IO	410	435

Fig. 6: AnTuTu score for both rails during AnTuTu under battery and wall power.

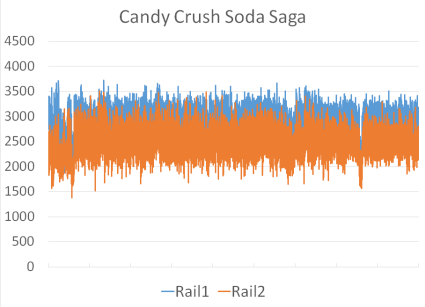

Fig. 7: Results from sampling Candy Crush Soda Saga for 3 minutes. Average: rail1=2892.74, rail2=2492.79.

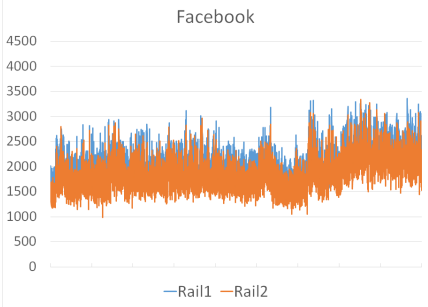

Fig. 8: Results from sampling Facebook for 3 minutes. Average: rail1=2087.20, rail2=1840.28.

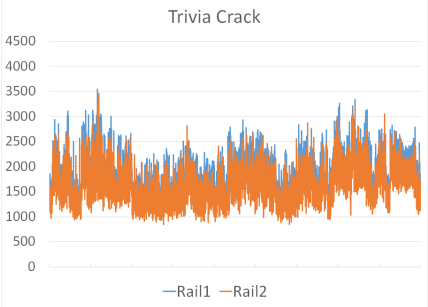

Fig. 9: Results from sampling Trivia Crack for 3 minutes. Average: rail1=1846.09, rail2=1624.91.

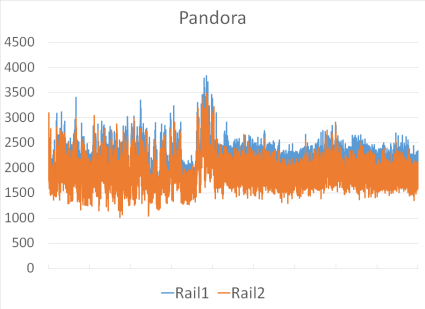

Fig. 10: Results from sampling Pandora for 3 minutes. Average: rail1=2169.85, rail2=1911.29.

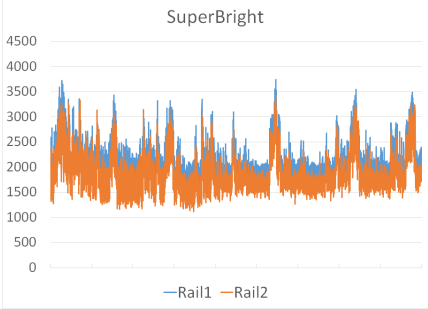

Fig. 11: Results from sampling Super-Bright LED Flashlight for 3 minutes. Average: rail1=2107.72, rail2=1857.41.

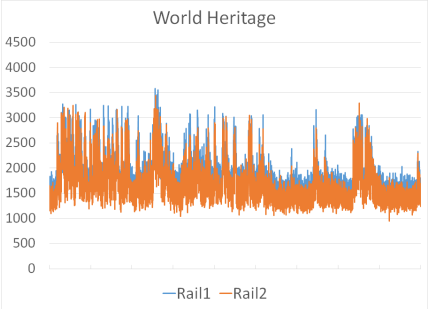

Fig. 12: Results from sampling World Heritage Calendar for 3 minutes. Average: rail1=1865.56, rail2=1661.79.

Had more than two rails showed any voltage, or those rails followed substantially different patterns, I could estimate something about the distribution of component voltage between them. Without this key insight it remains very difficult to provide a true profile of the energy consumption. Therefore, I shall simply consider the aggregate power of each rail in the app comparisons to follow.

Figures 7 through 16 each represent the results from the last experiment of running each app. They show the samples in millivolts from their respective app for 3 minutes of normal use, and each caption shows the rails' averages. Analysis of each is presented in the following section.

VI. Analysis

Based on the average sample from each application, Candy Crush Soda Saga uses the most energy of all applications tested with 2892.74 and 2492.79 millivolts for each respective rail. Phonostar used the least with 1745.66 and 1537.20 millivolts average for each respective rail. By adding up the rails for each app and summing those together based on type, in total PhoneGap applications seem to use less energy than their counterparts (19109 total of average between PhoneGap apps vs 20830 total of average between normal apps). This shows however that they are only narrowly better, $<9\%$ in fact. For a more direct comparison, the online radio and flashlight apps for PhoneGap were more efficient than their counterparts. The game app did favor normal apps (if I consider Candy Crush in the popular category instead of a game), but the two popular apps for PhoneGap combined were more efficient than the two normal popular apps.

As mentioned in the results, it is difficult to shed more light on where energy is actually being consumed since 6 of the 8 battery rails show so little voltage (including the ground). Therefore, forming a more concrete reason for the discrepancies between applications is near impossible. Generalizing this work to the broader population of apps may thus not be feasible based on these limited results. However, this presents an interesting first step towards measuring the differences between development platforms. Future work will aim to address this concern while expanding PowerPack so better results can be obtained.

VII. Conclusions

This project set out to address 3 research questions from the start. The implementation presented addressed how PowerPack can be extended to the Asus TF300T as a model for future devices. However, we saw that

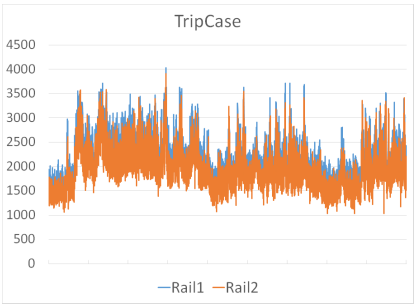

Fig. 13: Results from sampling TripCase for 3 minutes. Average: rail1=2189.34, rail2=1949.72.

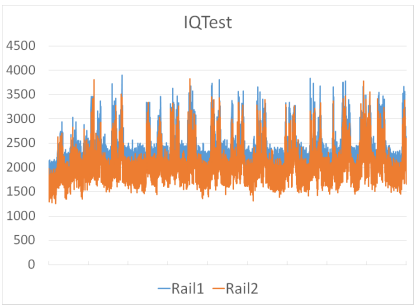

Fig. 14: Results from sampling ISIER IQ Test for 3 minutes. Average: rail1=2297.98, rail2=2038.87.

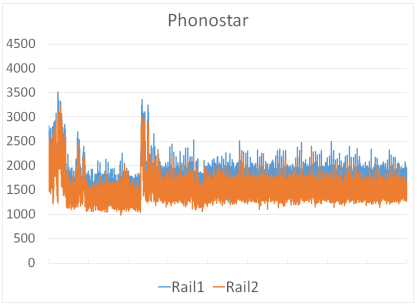

Fig. 15: Results from sampling Phonostar Radio App for 3 minutes. Average: rail1=1745.66, rail2=1537.20.

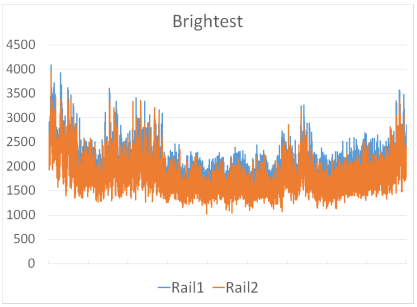

Fig. 16: Results from sampling Brightest LED Flashlight for 3 minutes. Average: rail1=2027.65, rail2=1795.35.

by utilizing that approach to measuring the voltage on battery rails, it was practically impossible to formulate a power profile at the component level. It is unknown at this point whether this result is the norm or simply a unique case for tablets. All that could be shown was the aggregate power of two rails on battery power, which averaged 1183.91 and 1163.98 millivolts, respectively, while idle and 1956.94 and 1552.43 millivolts, respectively, during benchmark. Nevertheless, by examining the voltage registered by the two rails measured, an initial comparison of development platforms was performed. The data showed that PhoneGap apps actually required less energy in most cases than their popular counterparts developed elsewhere. This analysis still requires more fine-grained profiling and more apps to generalize conclusively beyond this subset, but it still presents a novel first step. Future expansion of PowerPack or collaboration with outside groups using other methods could reveal more impactful results.

ACKNOWLEDGMENTS

I would like to thank my instructor, Dr. Kirk Cameron, for his support to undertake and feedback to complete this project. I must also thank my fellow researcher, Michael Vandeberg, for his help and collaboration throughout. The entire SCAPE lab was also crucial in getting these systems up and running. I will especially call out Bo Li for his unwavering assistance with all phases of the implementation and evaluation. I would be remiss if I did not thank my advisor, Dr. Eli Tilevich, for his continued guidance throughout all my work.

REFERENCES

[1] D. K. 3. Isier iq test. https://play.google.com/store/apps/details?id=com.digikm.isier1, 11 2014. Accessed: 2014-12-10.

[2] S. Allen, V. Graupera, and L. Lundrigan. Phonegap. In *Pro Smartphone Cross-Platform Development*, pages 131–152. Apress, 2010.

[3] Android. Introduction to android. http://developer.android.com/guide/index.html, 12 2014. Accessed: 2014-12-10.

[4] Android. Jelly bean. http://developer.android.com/about/versions/jelly-bean.html, 12 2014. Accessed: 2014-12-10.

[5] Asus. Asus transformer pad (tf300t). http://www.asus.com/us/Tablets_Mobile/ASUS_Transformer_Pad_TF300T/, 12 2014. Accessed: 2014-12-10.

[6] H.-C. Chang, A. Agrawal, and K. Cameron. Energy-aware computing for android platforms. In *Energy Aware Computing (ICEAC), 2011 International Conference on*, pages 1–4, Nov 2011.

[7] P. Corporation. World heritage calendar. https://play.google.com/store/apps/details?id=com.panasonic.jp.worldheritage&hl=en, 3 2014. Accessed: 2014-12-10.

[8] Etermax. Trivia crack. https://play.google.com/store/apps/details?id=com.etermax.preguntados.lite&hl=en, 12 2014.

[9] Facebook. Facebook. https://play.google.com/store/apps/details?id=com.facebook.katana&hl=en, 12 2014.

[10] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, WMCSA '99, pages 2–, Washington, DC, USA, 1999. IEEE Computer Society.

[11] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *Parallel and Distributed Systems, IEEE Transactions on*, 21(5):658–671, May 2010.

[12] R. Ghatol and Y. Patel. *Beginning PhoneGap: Mobile Web Framework for JavaScript and HTML5*. Apress, Berkely, CA, USA, 1st edition, 2012.

[13] Google. Google play store. https://play.google.com/store?hl=en, 12 2014. Accessed: 2014-12-10.

[14] S. T. Inc. Super-bright led flashlight. https://play.google.com/store/apps/details?id=com.surpax.ledflashlight.panel&hl=en, 10 2014. Accessed: 2014-12-10.

[15] A. Jindal, A. Pathak, Y. C. Hu, and S. Midkiff. On death, taxes, and sleep disorder bugs in smartphones. In *Proceedings of the Workshop on Power-Aware Computing and Systems*, HotPower '13, pages 1:1–1:5, New York, NY, USA, 2013. ACM.

[16] King. Candy crush soda saga. https://play.google.com/store/apps/details?id=com.king.candycrushsodasaga&hl=en, 12 2014.

[17] B. Li, H.-C. Chang, S. Song, C.-Y. Su, T. Meyer, J. Mooring, and K. W. Cameron. The power-performance tradeoffs of the intel xeon phi on hpc applications. In *Proceedings of the 2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, IPDPSW '14, pages 1448–1456, Washington, DC, USA, 2014. IEEE Computer Society.

[18] Pandora. Pandora internet radio. https://play.google.com/store/apps/details?id=com.pandora.android&hl=en, 11 2014.

[19] 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*, EuroSys '12, pages 29–42, New York, NY, USA, 2012. ACM.

[20] 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*, EuroSys '11, pages 153–168, New York, NY, USA, 2011. ACM.

[21] 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*, MobiSys '12, pages 267–280, New York, NY, USA, 2012. ACM.

[22] PhoneGap. Apps created with phonegap. http://phonegap.com/app/android/, 12 2014. Accessed: 2014-12-10.

[23] phonostar GmbH. phonostar radio-app. https://play.google.com/store/apps/details?id=de.phonostar.player, 10 2014.

[24] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Profiling resource usage for mobile applications: A cross-layer approach. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 321–334, New York, NY, USA, 2011. ACM.

[25] I. Qualcomm Technologies. Dragonboard. https://developer.qualcomm.com/mobile-development/development-devices/dragonboard, 12 2014. Accessed: 2014-12-10.

[26] S. T. Solutions. Tripcase travel alerts. https://play.google.com/store/apps/details?id=com.sabre.tripcase.android, 12 2014.

[27] S. Song, R. Ge, X. Feng, and K. W. Cameron. Energy profiling and analysis of the hpc challenge benchmarks. *Int. J. High Perform. Comput. Appl.*, 23(3):265–276, Aug. 2009.

[28] S. Sugar. Flashlight led torch free. https://play.google.com/store/apps/details?id=com.sweetsugar.flashlight, 7 2014.

[29] P. Vekris, R. Jhala, S. Lerner, and Y. Agarwal. Towards verifying android apps for the absence of no-sleep energy bugs. In *Presented as part of the 2012 Workshop on Power-Aware Computing and Systems*, Berkeley, CA, 2012. USENIX.