EMSE manuscript No.
(will be inserted by the editor)

An Exploratory Study on Assessing the Energy Impact of Logging on Android Applications

Shaiful Chowdhury · Silvia Di Nardo · Abram Hindle · Zhen Ming (Jack) Jiang

Received: date / Accepted: date

Abstract BACKGROUND: Execution logs are debug statements that developers insert into their code. Execution logs are used widely to monitor and diagnose the health of software applications. However, logging comes with costs, as it uses computing resources and can have an impact on an application's performance. Compared with desktop applications, one additional critical computing resource for mobile applications is battery power. Mobile application developers want to deploy energy efficient applications to end users while still maintaining the ability to monitor. Unfortunately, there is no previous work that study the energy impact of logging within mobile applications.

OBJECTIVE: This exploratory study investigates the energy cost of logging in Android applications using GreenMiner, an automated energy test-bed for mobile applications.

METHOD: Around 1000 versions from 24 Android applications (e.g., Calculator, FeedEx, Firefox, and VLC) were tested with logging enabled and disabled. To further investigate the energy impacting factors for logging,

Shaiful Chowdhury
Department of Computing Science
University of Alberta, Edmonton, AB, Canada
E-mail: shaiful@ualberta.ca

Silvia Di Nardo
Braindrain solutions Ltd.
London, UK
E-mail: silvia@braindrainltd.com

Abram Hindle
Department of Computing Science
University of Alberta, Edmonton, AB, Canada
E-mail: {shaiful,abram.hindle}@ualberta.ca

Zhen Ming (Jack) Jiang
Software Construction, AnaLytics and Evaluation (SCALE) lab
York University, Toronto, ON, Canada
E-mail: zmjiang@cse.yorku.ca

controlled experiments on a synthetic application were performed. Each test was conducted multiple times to ensure rigorous measurement.

RESULTS: Our study found that although there is little to no energy impact when logging is enabled for most versions of the studied applications, about 79% (19/24) of the studied applications have at least one version that exhibit medium to large effect sizes in energy consumption when enabling and disabling logging. To further assess the energy impact of logging, we have conducted a controlled experiment with a synthetic application. We found that the rate of logging and the number of disk flushes are significant factors of energy consumption attributable to logging. Finally, we have examined the relation between the generated OS level execution logs and mobile energy consumption. In addition to the common cross-application log events relevant to garbage collection and graphics systems, some mobile applications also have workload-specific log events that are highly correlated with energy consumption. The regression models built with common log events show mixed performance.

CONCLUSIONS: Mobile application developers do not need to worry about conservative logging (e.g., logs generated at rates of ≤ 1 message per second), as they are not likely to impact energy consumption. Logging has a negligible effect on energy consumption for most of the mobile applications tested. Although logs have been used effectively to diagnose and debug functional problems, it is still an open problem on how to leverage software instrumentation to debug energy problems. [1]

1 Introduction

Execution logs are generated by output statements (e.g., `System.out.println` or `printf`) that developers insert into their source code. Execution logs record the run-time behaviour of the application ranging from scenario executions (e.g., "Browsing scenario purchase for user Tom") to error messages (e.g., "Database deadlock encountered") and resource utilization (e.g., "20 out 150 worker threads idle"). Software developers, testers and operators leverage logs extensively to monitor the health of their applications [70], to verify the correctness of their tests [36] and to debug execution failures [72,74]. To cope with these tasks, there are many open source and commercial log analysis and monitoring frameworks available for large-scale server applications (e.g., Chukwa [1], Splunk [4], and logstash [2]).

Excessive logging could cause additional overhead inducing higher resource utilization or worse run-time performance [23]. For example, Google has shown that turning on the full logging would slow down their systems' run-time by 16.7% [65]. Developers, testers, and system administrators are concerned about the impact of logging on their applications [19]. This is also the case for mobile application developers [37,33,34]. Compared with desktop applications, one of the additional critical computing resources for mobile applications is

[1] EMSE-D-16-00048R3 http://rdcu.be/v4DT

battery power. Mobile application developers (short for *app developers*) want to deploy energy efficient applications to end users while still maintaining the ability to monitor and debug their applications using logs. However, the energy impact of logging on mobile applications is not clear to the developers. When one app developer asked whether logging would drain the battery for Android phones, on the Stack Overflow forum [59], he received three conflicting responses: "yes", "no", and "it depends". This lack of definitive response is similar to the disagreement between practitioners on energy consumption questions observed by Pinto *et al.* [53].

In this paper, we have studied the energy impact of logging on Android applications using the GreenMiner [29]. The GreenMiner is an automated test-bed for studying the energy consumption of mobile applications. It automatically tests the mobile applications while physically measuring the energy consumption of mobile devices (Android phones). The measurements are automatically reported back to developers and researchers. Using the GreenMiner the following three research questions are studied to assess the energy impact of logging on mobile applications:

- **RQ1: What is the difference in energy consumption for Android applications with and without logging?**
  This research question investigates whether the energy consumption of an Android application would be different when enabling and disabling logging. Around 1000 versions from 24 real-world Android applications, including `Calculator`, `FeedEx`, `Firefox`, and `VLC`, were studied.
- **RQ2: What are the factors impacting the energy consumption of logging on Android applications?**
  This research question aims to identify the important factors in logging that impact software energy consumption. Controlled experiments were conducted to investigate two factors of logging energy consumption: log message rate and log message size. In addition, the relationship between energy consumption and the number of disk flushes was analyzed.
- **RQ3: Is there any relationship between the logging events and the energy consumption of mobile applications?**
  This research question explores the relationship between log events and software energy consumption to see if some log events are more correlated with energy consumption than others. Data analyses, like correlation and multiple linear regression, were carried out to study the relationship between events recorded in logs and mobile energy consumption.

The contributions of this paper are summarized as follows.

1. To the best of our knowledge, this is the first work that proposes a systematic approach to study the energy impact of logging on mobile applications.
2. The findings of this paper were based on an extensive set of measurements/experiments (approximately 70 days of testing time), which includes a wide variety of Android applications with logging enabled and disabled, and a controlled experiment with varying logging rates and message sizes.

Each experiment was repeated multiple times to avoid measurement bias and errors.
3. We provide evidence for developers that they need not worry about impacting energy consumption of their mobile applications if they conservatively employ logging.
4. To encourage replication and further study on this important topic, we have disclosed our dataset and source code for our analysis in our replication package. We believe such data can be very useful for software engineering researchers and app developers [3].

The rest of this paper is organized as follows: Section 2 provides some background information on logging and the GreenMiner. Sections 3, 4, and 5 discuss the research questions RQ1, RQ2, and RQ3, respectively. Section 6 discusses the threats to validity. Section 7 explains the prior works in the area of mobile energy analysis and execution logs. Section 8 concludes this paper.

2 Background

This section provides the background information on software logging. It is broken down into three parts. First, we discuss the general approaches for software instrumentation. Then, we explain how logging is realized in Android applications. Finally, a brief description of our automated energy test-bed for mobile applications, GreenMiner, is provided.

2.1 General Approaches for Software Instrumentation

Execution logs are generated by the instrumentation code that developers insert into the source code. Execution logs are widely available for software systems to support remote issue resolution and to cope with legal compliance [5]. There are three types of instrumentation approaches [69]:

- *Ad-hoc logging:* Developers insert logging in an ad-hoc manner using output methods such as print statements like `System.err.println`. Although this is the easiest approach to instrument, unexpected side effects might happen if one is not careful. For example, logs lines can be garbled if there are multiple threads trying to output log lines to the same file using I/O methods that are not thread-safe.
- *Systematic logging:* The general purpose instrumentation frameworks (e.g., Log4j [9]) address the limitations of the ad-hoc logging approach, as the frameworks support thread-safe logging. In addition, these frameworks provide better control of the types of information outputted. For example, the Log4J framework provides multiple verbosity levels: ERROR, WARN, INFO, DEBUG, and VERBOSE. Each of these verbosity levels can be used for different software development activities. For example, implementation level details can be logged using DEBUG or VERBOSE level, whereas

critical errors should be logged under the ERROR level. When deploying a particular application, a verbosity level should be set. For example, if the verbosity level is set to be DEBUG, all the logs instrumented with DEBUG and higher (a.k.a., ERROR, WARN, INFO) are printed whereas lower level logs (a.k.a., VERBOSE) are discarded.
– *Specialized logging:* There are also instrumentation frameworks available to facilitate special purpose logging. For example, it is easier to instrument the system using the ARM (Application Response Measurement) framework [24] to gather performance information from the running application, than to manually instrument the system.

2.2 Android Logging

Android handles application logs similar to how UNIX handles `syslog` logs. Calls to the Android logging API [9] write logs to a circular buffer in memory. This buffer can be ignored or dumped to disk. Periodic writing of logs can lead to log rotation where old logs are renamed and kept until too many logs are allocated. While running the Android applications, the circular buffer could be filled causing it to periodically dump the logging data to the disk, these dumps or writes are referred to as *disk flushes*. Different mobile phones can have different buffer sizes (e.g., 256 KB or 512 KB). To collect and filter logs for Android applications, there is a utility called `logcat` [10]. Similar to Log4j, the Android API for logging [9] provides multiple verbosity levels. Table 1 shows a set of sample log lines from the Android Calculator application. At the beginning of each log line, there is a letter (e.g., "I/" or "D/"). These letters correspond to different verbosity levels. "I/" corresponds to the INFO level logs and "D/" to the DEBUG level logs. The words after the verbosity level show the components where the logs are generated. For example, the first log line is generated by the `ActivityManager` component from the calculator application. The second log line is generated by the `dalvikvm` component, which is the Java Virtual Machine used by the Android operating system.

When released, ERROR, WARN, and INFO level logs are printed for Android applications. Logging for Android applications can also be completely disabled. The following are mechanisms to enable and disable logging for Android applications:

– **Logging Enabled:** First, the log buffer is cleared with the command `logcat -c` [10]. Then, the following command is invoked to redirect the log output of a particular application to a log file (`logcat.txt`) on the SD card.

```
logcat -d | grep -e $PID -e \
net.fred.feedex > /sdcard/logcat.txt
```

– **Logging Disabled:** The configuration shown below was added to the `build.prop` file in the `/system` folder of the smartphones. The `/dev/log` folder was removed, along with all its contents.

Table 1 Sample log events from the CALCULATOR application

#	LOG LINES
1	I/ActivityManager(387): START u0 flg=0x10000000 cmp=com.android2.calculator3/.Calculator from pid 21740
2	D/dalvikvm(21750): GC CONCURRENT freed 177K, 3% free 8922K/9128K, paused 2ms+3ms, total 18ms
3	D/libEGL (21750): loaded /vendor/lib/egl/libEGL-POWERVR-SGX540-120.so
4	D/OpenGLRenderer(21750): Enabling debug mode 0
5	I/ActivityManager(387): Displayed com.android2.calculator3/.Calculator: +643ms
6	I/WindowState(387): WIN DEATH: Window(41e6d7c8 u0 com.android2.calculator3/.Calculator)
7	I/ActivityManager(387): Force stopping package com.android2.calculator3 appid=10062 user=0

```
logcat.live = disable
```

2.3 GreenMiner

To measure the energy consumption of the selected Android applications, we used the GreenMiner framework [29]. The GreenMiner has been widely used and accepted in the software energy research community. There are many published works on energy research that used the GreenMiner as their energy measurement tool [27,14,16,29,56,7,6].

The GreenMiner is a hardware-based energy measurement system that operates 4 Android Galaxy Nexus phones in parallel. Table 2 shows the detailed hardware and software specifications for these phones. These phones are used as the systems under test and are controlled by 4 different Raspberry Pi model B computers. Each Pi acts as a test manager for one single phone. It deploys and runs tests, collects energy measurements, and uploads the results to a central server. When a batch of tests are submitted to the GreenMiner, one of the four phones are selected randomly to execute a test. In this way four tests can be executed in parallel, enabling the expedient evaluation of experiments, reducing data collection time significantly. It is important to note that after completing a test for an app, the GreenMiner uninstalls the app and deletes app related data. This is to make sure that each test run is independent and is not affected by any of the previous test runs.

A constant voltage of 4.1 V, from a YiHua YH-305D power supply, was first passed through an Adafruit INA219 breakout board, and then to an attached phone. The pins, where the phone's battery is usually attached, were wired to receive energy from the power supply. This voltage and amperage were reported to an Arduino Uno by the INA219. The INA219 [**?**] relies on a shunt resistor to measure changes in amperage. The Arduino Uno then delivers the readings to a Pi through a serial USB connection. Figure 1 depicts the innards

Table 2 Specs of the Samsung Galaxy Nexus phones used for the experiments.

Component	Specs
OS	Ice Cream Sandwich, 4.4.2
CPU	Dual-core 1.2 GHz Cortex-A9
GPU	PowerVR SGX540
Memory	16 GB, 1 GB RAM
Display	AMOLED, 4.65 inches
WLAN	Wi-Fi 802.11 a/b/g/n

of the GreenMiner (one out of the four identical settings). For a more detailed GreenMiner methodology and architecture, please refer to [29,54].

As energy measurements can vary slightly between different runs for the same tests, it has been a common practice in software energy research to run each test at least 10 times, to achieve acceptable statistical power, and to report average measurements [16,14,7,27].

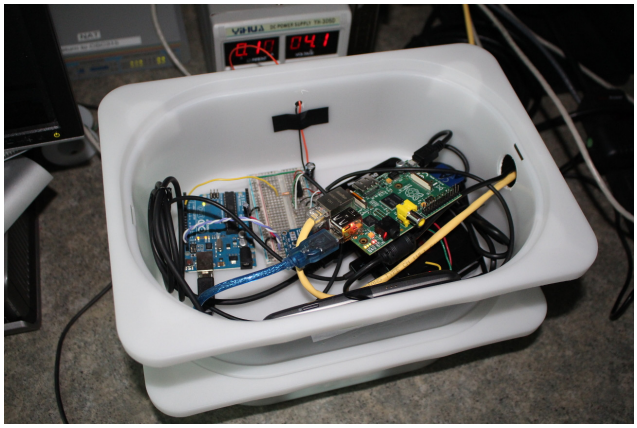

Fig. 1 GreenMiner consists of an Arduino, a breadboard with INA219 chip, a Raspberry Pi, a USB hub, and a Galaxy Nexus phone connected to a Power Supply. Photo used with permission from the Green Miner paper [29].

We measure energy consumption by the integration of power (watts) over time. This energy measurement is called joules (J). Joules are typically stored within a mobile device battery when it charges, and are expended for computation, communication, and peripherals while the device is in operation. 1 joule is 1 watt-second. The phones we use typically consume 0.7 J per second while idle with the screen on, and 1.5 J to 3 J per second when very busy with the screen on. Thus the difference of 10 J between 2 test runs could be due to 14 seconds of runtime or a few seconds of high CPU workload difference. All the measurements of energy consumption in this paper are in joules.

3 RQ1: What is the difference in energy consumption for Android applications with and without logging?

3.1 Motivation

On one hand, execution logs can bring insights about the run-time behaviour of mobile applications. On the other hand, should app developers be concerned about the potential energy overhead of logging on their applications? The energy impact of execution logs on 24 real-world Android applications is examined in this section.

3.2 Experiments

In order to draw a reliable conclusion on how logging impacts energy consumption of existing apps, we experimented with 24 Android applications from different domains (e.g., Games, Entertainment, Communication, News, and Utility). To capture the general behaviour of each studied application, multiple versions for each application are studied. One version in this paper refers to one binary compiled from one distinct commit from a source code repository, or one compiled binary released by the project. For example, we have studied 46 code commit versions (a.k.a., 46 versions) for the VLC app. The source code for the multiple versions of these 24 applications are part of the *GreenOracle* dataset collected by Chowdhury *et al.* [16].

Software changes over time. The logging changes of some versions of an app may consume much more energy than the other versions. Hence, it is worthwhile to study a number of versions for the same application. Another important aspect of studying the energy impact of logging is that writing test cases manually is difficult. But if we use multiple versions of the same app (versions with identical user interface) then writing a single test script is enough for all the versions. This can enlarge the size of our measurements and thus allows more reliable analysis. Out of the 24 Android applications in *GreenOracle* dataset, we had to exclude YELP from our analysis, as this particular application disables logging internally. We included one more application (FEEDEX with 35 versions) in our dataset as a compensation. Table 3 shows the details of the studied Android applications.

Figure 2 illustrates the process for this study. For each version of the selected applications, we measured the energy consumption with logging enabled and disabled with one realistic test case per app. A test scenario for an application, which is automated by a test script written in Android `adb shell`, simulates how an average user would use the application. For example, the FEEDEX first adds RSS feeds from Google News. Then it emulates a normal user opening and reading the first two RSS feeds. The CALCULATOR application test converts miles to kilo-meters, calculates tax amounts, and solves an equation using the quadratic formula.

Table 3 The applications under test, selected from the *GreenOracle* [16] dataset.

Categories	App Names	App Descriptions	# of Versions and Committed time	Repo
Games	2048	Puzzle game	44 (03/2014 - 08/2015)	GitHub
	24game	Arithmetic game	1 (01/2015 - 01/2015)	F-Droid
	Agram	Anagrams	3 (03/2015 - 10/2015)	F-Droid
	Blockinger	Tetris game	74 (04/2013 - 08/2013)	GitHub
	Bomber	Bombing game	79 (05/2012 - 11/2012)	GitHub
	Vector Pinball	Pinball game	54 (06/2011 - 03/2015)	GitHub
News	FeedEx	Reading news feeds	35 (05/2013 - 04/2014)	GitHub
	Exodus	Browse 8chan	3 (01/2010 - 04/2015)	GitHub
	Eye in the Sky	Weather app	1 (09/2015 - 09/2015)	Google Play
Entertainment	Acrylic Paint	Finger painting	40 (03/2012 - 09/2015)	GitHub
	Memopad	Free-hand Drawing	52 (10/2011 - 02/2012)	GitHub
	Paint Electric Sheep	Drawing app	1 (09/2015 - 09/2015)	Google Play
	VLC	Video player	46 (04/2014 - 06/2014)	GitHub
References	AndQuote	Reading quotes	21 (07/2012 - 06/2013)	GitHub
	Wikimedia	Wikipedia mobile	58 (08/2015 - 09/2015)	GitHub
Communication	ChromeShell	Web Browser	50 (03/2015 - 03/2015)	APK repository
	Face Slim	Connect to Facebook	1 (11/2015 - 11/2015)	F-Droid
	Firefox	Web browser	156 (08/2011 - 08/2013)	APK repository
Business	Budget	Manage income/expenses	59 (08/2013 - 08/2014)	GitHub
	Calculator	Calculations	97 (01/2013 - 05/2013)	GitHub
	GnuCash	Money Management	16 (05/2014 - 08/2015)	GitHub
	Temaki	To do list	66 (09/2013 - 07/2014)	GitHub
System Utilities	DalvikExplorer	System information	13 (06/2012 - 01/2014)	code.google
	Sensor Readout	Read sensor data	37 (03/2012 - 03/2014)	GitHub

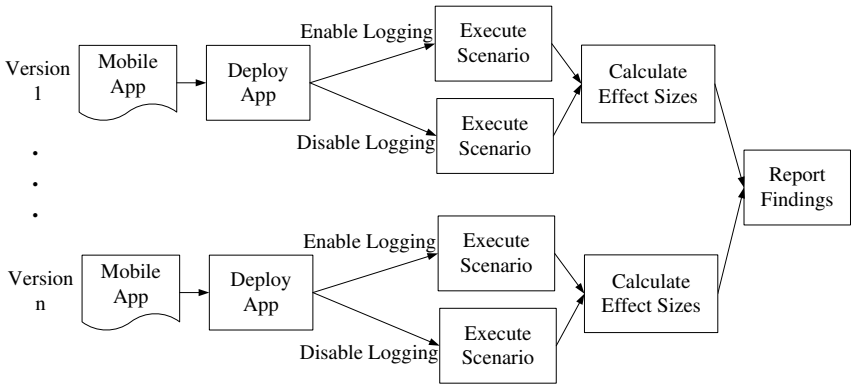

Fig. 2 Process to Investigate the Energy Impact of Logging (RQ1)

Table 4 shows the list of test scenarios for all the applications. The test scenarios and the test duration are the same across different versions of the same application. For example, a single FeedEx test (with or without logging), lasts for 100 seconds. Table 4 also shows the average number of log lines per test run, the average test duration in seconds, the average logging rate (events per second), and the joules consumed with logcat enabled (logging enabled) and logcat disabled (logging disabled).

For both logging conditions (logging enabled and disabled), the experiments were repeated 10 times (for each version) to address measurement error and random noise [22,38]. For an approximate average test duration of five minutes (including uploading data to a server after each test), it took around 70 days to run and collect all the measurements from GreenMiner. All of these measurements were then used to compare the energy consumption between logging enabled and disabled.

3.3 Analysis

For the logging enabled tests, the log files have on average 142 log lines, ranging from 12 messages to 1080 messages per test run. The average test duration can last from 52 seconds (AndQuote) to 210 seconds (FireFox). We will perform a two-step analysis on the energy measurement data. First, we will perform a hypothesis testing to examine whether the energy consumption with and without logging for each version of the app is different. Then we will study the magnitude of the differences (a.k.a., effect sizes) to help quantify the size of the differences.

Comparing the Differences Between Two Groups: Some of the measured energy distributions are not always normally distributed, according to the Shapiro-Wilk normality test. Hence, we will use non-parametric tests throughout this paper. Different from parametric tests, non-parametric tests do not have any underlying assumptions of the distribution of the data be-

Table 4 Test scenarios and test results for the selected Android applications.

App Names	Test Scenarios	Avg # of log lines	Avg Test Duration	Avg Rate of Logging	Avg Energy(J) with	
					logcat Enabled	logcat Disabled
2048	Makes some random moves	15.737	60.008	0.262	58.369	59.057
24game	Randomly tries different numbers	110.000	80.014	1.375	85.816	84.407
Acrylic Paint	Draws a hexagon with legs	24.621	95.011	0.259	82.838	83.998
Agram	Generates anagrams (single and multiple)	46.447	77.006	0.603	75.299	74.985
AndQuote	Reads a series of famous quotes	24.265	52.003	0.467	44.671	44.473
Blockinger	Repositions/rotates blocks randomly	58.715	150.002	0.391	197.315	197.984
Bomber	Drops bombs at fixed intervals	194.091	130.008	1.493	170.483	170.826
Budget	Inserts and calculates expenses	101.684	125.010	0.813	113.017	113.007
Calculator	Converts units, calculates taxes, and solves equations	24.413	125.008	0.195	107.781	107.062
ChromeShell	Opens a webpage and scrolls	153.224	100.010	1.532	106.621	107.049
DalvikExplorer	Reads the system's information	20.799	80.004	0.260	65.750	65.591
Exodus	Reads threads from different topics	239.297	84.012	2.848	96.981	96.001
Eye in the Sky	Looks for the current temperature in Edmonton	182.583	130.008	1.404	116.768	119.310
Face Slim	Connects to Facebook homepage and access the help page	24.500	60.009	0.408	65.935	66.571
FeedEx	Adds and reads feeds from Google News	94.451	100.000	0.945	95.430	92.956
Firefox	Opens a webpage and scrolls	75.325	210.004	0.359	213.679	211.544
GnuCash	Creates an account and saves transactions	90.810	75.012	1.211	76.150	76.713
Memopad	Draws a hexagon with legs	17.966	95.011	0.189	79.649	79.908
Paint Electric Sheep	Draws a hexagon with legs	30.000	60.007	0.500	56.296	57.601
Sensor Readout	Shows graphs for different sensor reads	166.220	182.998	0.908	176.278	177.226
Temaki	Makes a TODO list, updates and deletes the list	12.244	75.010	0.163	72.373	72.189
Vector Pinball	Throws the ball several times and tires to hit the ball randomly	17.340	120.009	0.144	116.359	116.700
VLC	Plays a fireworks `.3gp` video	1,079.676	110.010	9.814	116.464	117.460
Wikimedia	Searches for the Bangladesh page and scrolls	169.783	120.011	1.415	160.015	160.171

ing analyzed. For each version of the mobile applications, the Wilcoxon Rank Sum test is performed to check whether the differences in energy consumption between the cases of logging enabled and disabled are statistically significant. Table 5 shows the results of these tests.

Table 5 Wilcoxon Rank Sum Tests ($\alpha = 0.05$) comparing energy consumption between logging enabled versus disabled per version. $p \leq 0.05$ means that there is a statistically significant difference in the energy consumption between logging enabled and disabled, whereas $p > 0.05$ means otherwise. Cliff's δ magnitude across applications versions is from [30].

App Names	% versions with $p \leq 0.05$	Mean Cliff's δ	Effect Sizes			
			% Negligible	% Small	% Medium	% Large
2048	0.000	−0.165	40.909	31.818	20.455	6.818
24game	0.000	−0.077	100.000	0.000	0.000	0.000
Acrylic Paint	7.500	−0.439	5.000	17.500	35.000	42.500
Agram	0.000	0.038	33.333	66.667	0.000	0.000
AndQuote	0.000	0.127	47.619	23.810	19.048	9.524
Blockinger	0.000	−0.085	41.892	37.838	14.865	5.405
Bomber	0.000	−0.120	34.177	46.835	13.924	5.063
Budget	0.000	−0.080	40.678	44.068	8.475	6.780
Calculator	3.093	0.352	21.649	22.680	22.680	32.990
ChromeShell	0.000	−0.159	32.000	42.000	22.000	4.000
DalvikExplorer	0.000	−0.073	53.846	38.462	0.000	7.692
Exodus	0.000	0.340	33.333	0.000	33.333	33.333
Eye in the Sky	0.000	−0.375	0.000	0.000	100.000	0.000
Face Slim	0.000	−0.319	0.000	100.000	0.000	0.000
FeedEx	54.286	0.612	8.571	5.714	14.286	71.429
Firefox	0.000	0.152	34.615	37.179	19.872	8.333
GnuCash	0.000	−0.147	37.500	18.750	37.500	6.250
Memopad	0.000	−0.187	34.615	36.538	17.308	11.538
Paint Electric Sheep	0.000	−0.597	0.000	0.000	0.000	100.000
Sensor Readout	0.000	−0.085	43.243	37.838	16.216	2.703
Temaki	0.000	−0.028	43.939	39.394	12.121	4.545
Vector Pinball	0.000	−0.047	40.741	35.185	20.370	3.704
VLC	4.348	−0.294	15.217	45.652	13.043	26.087
Wikimedia	0.000	−0.123	43.103	36.207	12.069	8.621
Overall (paired)	$p = 0.3748$		0.0139 (Negligible)			

Given $\alpha = 0.05$, $p \leq 0.05$ means that there is a statistical difference in the energy consumption between the logging enabled and logging disabled tests, whereas $p > 0.05$ means otherwise. We also correct for multiple comparisons/hypotheses using the Benjamini and Hochberg method [12] which attempts to control the false discovery rate. Most applications (e.g., 2048 and AndQuote) do not have statistical differences in terms of energy consumption in any of their versions between the logging enabled and disabled tests. However, for some other applications (e.g., FeedEx and Acrylic Paint), many of their versions exhibit statistical differences between the logging enabled and disabled tests. Figure 3 depicts the p-values per application per version comparing logging enabled (logcat enabled) and logging disabled tests. Only 4 out of 24 applications exhibit cases where their energy consumption in the

logging enabled and disabled tests are statistically significantly different after correction for multiple hypotheses.

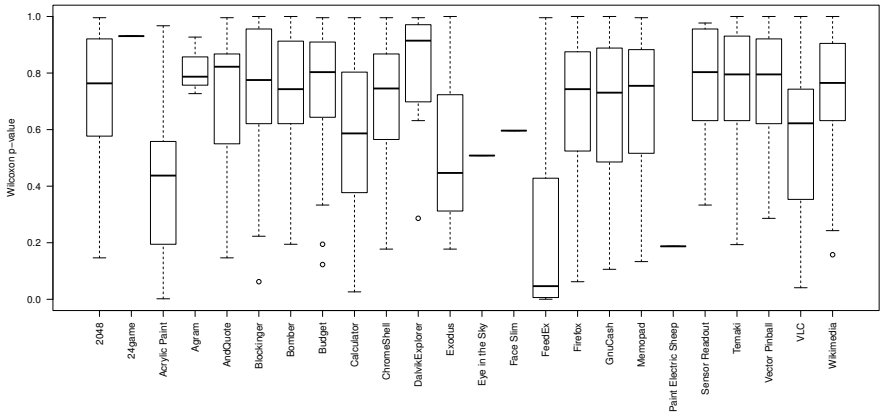

Fig. 3 Wilcoxon Rank Sum p-values per application of energy consumed with logging and without logging. p-values less than 0.05 indicate that logging enabled and logging disabled consumed different amounts of energy. p-values were corrected for multiple hypotheses using Benjamini and Hochberg correction [12]

Effect Sizes: Although the Wilcoxon rank sum test can examine whether there is a statistical difference in terms of energy consumption between the logging enabled and disabled tests, it cannot quantify the magnitude of the differences. Hence, Cliff's δ (Cliff's delta) is used to calculate the differences of energy consumption for logging enabled and disabled tests. Cliff's δ is a non-parametric effect size measure that quantifies the proportional difference (or dominance) between two sets of data [55]. Cliff's δ has four categories: negligible effect, small effect, medium effect, and large effect. Effect sizes, which can be applied regardless of significance of a T-test or a Wilcoxon rank sum test, is used to characterized the observed differences of the effect in the measurements [66]. Reporting effect size is recommended in cases where there is not enough statistical power [46]. For instance given the number of repeated tests and given the number of hypotheses—how many times we repeated a statistical test—the critical value will be low. This means in order to be conservative enough to reduce false positive rates the p-value correction will make the multiple Wilcoxon rank sum tests quite conservative. This effectively turns the Wilcoxon rank sum test into a measure of sample size, but the effect still remains. Thus we report effect sizes to give the reader an idea of the differences between logging and not logging within the data, regardless of statistical significance.

Table 5 tabulates the effect size values, Cliff's δ, for different versions of the Android applications. For example, in Table 5, the values of Cliff's δ show that 21% of the versions of the CALCULATOR application have negligible effect, 23%

of versions with small effect, another 23% versions with medium effect, and the remaining 33% of versions with large effect. In addition to the CALCULATOR application, there are five other applications that have more than half of their versions exhibit medium to large effect sizes.

Thus based on these observations, we want to statistically verify the effect of logging on applications. Between applications, aggregated by averaging joules across versions, we find that with the paired Wilcoxon signed rank test there is no statistically significant difference between enabling and disabling logcat across these applications ($p = 0.3748$ and $p > \alpha$). The effect size, over all applications, according to Cliff's δ is negligible (0.0139). The paired Wilcoxon signed rank test is used because the samples are related and paired (e.g., mean joules of FIREFOX with logging, and mean joules of FIREFOX without logging).

The results show that the differences in energy consumption are not statistically significant for most versions of the studied 24 applications. Furthermore, within the same applications we find that the effect of enabling or disabling logging is typically statistically insignificant and of negligible to small magnitude. However, 79% (19/24) of the studied applications have at least one version with medium to large effect sizes in terms of the differences of energy consumption when enabling and disabling logs.

In order to have more insight into the impact of logging on energy consumption, we select FEEDEX, an application in our dataset, which shows not only statistically different energy measurements between logging enabled and disabled tests (in 54% versions), but also have 71% versions with large effect size. For instance, there is a big difference (≈ 10 joules) in terms of energy consumption between logging enabled and disabled tests for the version 1.6.0. Compared to the previous versions, there were 178 more Dalvikvm *WAIT_FOR-_CONCURRENT_GC* log lines and 224 more Dalvikvm *GC_CONCURRENT* log lines. These logs are related to the memory management of the applications. This version of the FEEDEX app, seemed to suffer from memory bloat issues and produces a larger log file than its predecessor.

Figure 4 shows the energy consumption of FEEDEX over time, for both logging enabled and disabled tests. It is clear that the later versions are more energy inefficient than their predecessors. Figure 5 shows the energy consumption for the FEEDEX versions against the number of log lines. With few exceptions, we observe a monotonous increase in energy consumption with the increase in logging. This suggests that with more information in log files, one could investigate what types of log events can impact the energy consumption, and thus motivated us for RQ3. However, the energy differences between logging and no logging do not show any consistent pattern with the increase in log messages. With randomly selected real-world applications, there can be many factors that can significantly impact the energy consumption [16,51,68] of Android applications. Such uncontrolled tests can indicate if logging matters or not, but cannot offer an accurate estimation of the impact of logging on energy consumption.

These results indicate the need for a more controlled experiment—to show how much logging can be harmful in terms of energy consumption. We did

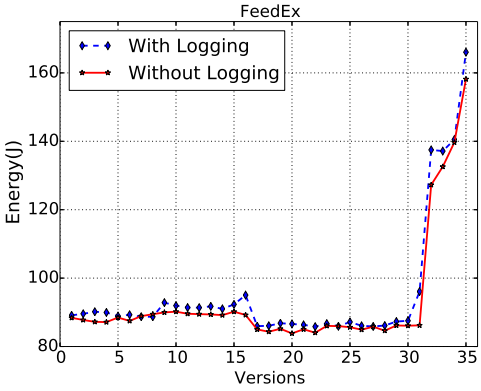

Fig. 4 FeedEx Energy consumption over time. Versions 32 to 35 exhibit very different energy profiles compared to the previous versions.

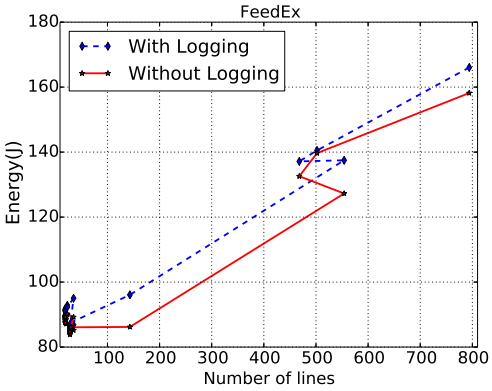

Fig. 5 Energy consumption against the number of log lines across different FeedEx versions. The graph depicts 2 measurements and the lines connects between adjacent versions. The line depicts how the FeedEx versions move through the space of log length and energy consumption. Essentially consecutive FeedEx versions use more and more energy.

not have control over the development of these applications and their use of logging. Furthermore, high logging rates (a.k.a., consistent logging rates faster than 20 msg/sec) were not observed from these applications and tests. Hence, in the next RQ (Section 4), we will study the factors impacting the energy consumption of logging on Android applications using controlled experiments and with various logging rates.

3.4 Summary

Findings: The energy consumption between logging enabled and disabled tests are not statistically significant for most versions of the studied mobile applications. However, approximately 79% of the studied applications have at least one versions with effect sizes larger than or equal to medium. Internal factors such as memory management issues are the causes behind the energy increases correlated with logging.
Implications: Logging usually does not have a noticeable impact on the energy consumption of Android applications, although in some cases it can. Developers should be careful when adding additional instrumentation code, yet still leveraging this valuable debugging tool. Characterizing the best practices on making energy-efficient logging decisions in mobile applications is still an open research problem.

4 RQ2: What are the factors impacting the energy consumption of logging on Android applications?

4.1 Motivation

Currently, there are few guidelines regarding logging on mobile devices and logging's energy impact for mobile developers to follow. It is not clear to mobile developers how much they can log and how often. This section seeks to provide some insights into this aspect of logging and energy consumption.

4.2 Experiments

There are three orthogonal factors that can potentially impact the energy consumption of logging: (1) the rate of logging, (2) the size of log messages, and (3) the number of disk flushes. The rate of logging and the size of log messages can be controlled by the individual applications, but not the number of disk flushes. Depending on the volume of the logs and the buffer size, the number of disk flushes can vary. The volume of the logs (a.k.a., the size of the log file) depends both on the rate of logging and the size of log messages. Bigger log messages and more frequent logging lead to higher volumes of logs. The buffer size varies depending on the mobile phones. Our test-bed uses Android phones with 256 KB circular buffer sizes. The disk flush happens when the buffer gets filled up.

It is not easy to investigate the energy impacting logging factors with real-world applications. First, isolating the pure energy costs for logging is difficult; these applications interact with other components that also consume energy (e.g., radio and screen). Second, unstable logging rates and log size with real-world applications hinder controlled experiments.

Table 6 Controlled experiments with varying logging rates and message sizes.

Logging Rate		Message Size	
Rate (msg/sec)	Rationale	Size (bytes)	Rationale
0.01	infrequent logging	64	a single line of text
0.10	browsing UI level logging	512	a medium sized line of text
1.00	UI event level logging	1024	a URL sized line of text
10.00	network traffic level logging	1536	maximum ethernet data frame size
100.00	printf debugging logging	2048	a large log message
1000.00	very frequent logging	8192	an exceptionally large log message

Hence, we have built a test Android application to assess the energy-impacting factors for logging. Our test application, which consists of only the `MainActivity` and a JUnit test case, performs only one task: generating logs at different rates and with different message sizes. For each test, the operations are the same: the `MainActivity` is launched. Then the application starts to generate log messages of a specific size at a specific rate for 120 seconds and stops. The duration of 120 seconds is chosen to help stabilize measurements against unexpected CPU frequency switches. By the first 60 seconds of the test, the CPU frequency should be appropriately set for the logging workload. Table 6 lists the set of different message rates and message sizes that were run. For each of the specified message rates and sizes, the rationale is also included. For example, app developers might be interested in printing and storing stack traces or packet dumps in a log file. A typical stack trace is around 8192 bytes (8 KB) and a typical Ethernet packet is 1536 bytes (1.5 KB). If one logged UI level events, the UI events are usually generated at a rate of 1 to 5 events per second.

Much like RQ1, each experiment was repeated multiple times (40 times) to avoid measurement errors and random noise [22,38]. It took 70 hours in total to run these tests. The testing results are gathered for further analysis.

4.3 Analysis

The average energy consumption of each test is calculated. The data is grouped according to the rate of the logging (a.k.a, msg/sec). The average energy consumed for the idle tests (a.k.a., generating zero msg/sec) is used as the baseline in order to track the percentage increase in terms of energy consumption for the log generating tests. Table 7 shows the results. For example, there is a 0.2% energy increase under 0.01 msg/sec with 64 bytes as the message size.

In fact, for 10 msg/sec there is a very small energy increase ($\leq$ 2.46%), with the largest message size. However, the impact is significant with larger message rates. As the logging rate increases to 100 msg/sec, the increase in the energy consumption ranges from 8.26% to 14.20% for different message sizes. For 1000 msg/sec, the increase of the energy consumption can go up

Table 7 Percentage growth rates of energy consumption (joules) for the log generating tests. All the calculations below used the energy consumption of the idle tests as the baseline.

MSG/SEC	64 BYTES	512 BYTES	1 KB	1.5 KB	2 KB	8 KB
0.01	0.20%	0.21%	0.18%	0.51%	0.28%	0.37%
0.10	0.27%	0.31%	0.38%	0.73%	0.30%	0.61%
1.00	0.65%	0.64%	0.70%	0.99%	0.64%	0.90%
10.00	1.38%	1.59%	1.71%	2.16%	1.91%	2.46%
100.00	8.26%	8.48%	9.23%	10.33%	10.55%	14.20%
1000.00	27.88%	30.66%	36.50%	45.14%	48.26%	75.47%

to 75.47%. Another interesting observation is that the energy increase for 10 msg/sec and 8 KB message size is much smaller than 100 msg/sec and 64 byte message size (2.46% vs. 8.26%), even though the unit volume of the generated logs are much higher (80 KB/sec vs. 6.25 KB/sec). *Evidently, logging rate is a more dominant factor than message size for energy consumption*

For further verification, we apply factor analysis to verify the importance of message size and log rate. Kruskal-Wallis (Kruskal-Wallis X^2) tests are performed to check whether the factors of logging rates and the message sizes have statistically significant impacts on the energy consumption. Kruskal-Wallis test is a non-parametric statistical test for checking whether the measurements of 3 or more groups, under different kinds of treatments, come from the same distribution. We test 2 factors independently: logging rates and logging message sizes. We correct for multiple/hypotheses with the Benjamini and Hochberg method [12]. Logging rate was a significant factor ($p < 2.2e - 16$) for energy consumption. Although the message size is also statistically significant factor, the p-value ($p < 0.0471$) is very close to our α ($\alpha = 0.05$). In fact, when we test with the pairwise Wilcoxon rank sum test between the *message sizes*, corrected using Benjamini and Hochberg, we find no statistically significant differences in energy consumption between distributions of different message sizes ($p > \alpha$). Yet a pairwise Wilcoxon rank rum test shows there are statistically significant differences ($p \leq \alpha$) for all log rate comparisons except for 2 comparisons of log rates of 0.01 to 0.1 and log rates of 0.1 to 1.0. The similar distributions of energy consumption at low frequency log rates also helps to explain the mild inconsistency in trend observed in energy consumption with the message sizes.

In order to better understand the relationship between the message size and the energy consumption, we calculate the Pearson correlation coefficient between them—an indicator of a linear relationship between two random variables. The correlation coefficient is low (only 0.17 with $p \approx 0$) when the message rate is not fixed. However, with fixed and high message rate (e.g., 100 msg/sec), the correlation is high (0.72 with $p \approx 0$). This is also consistent when the message rate is fixed at 1000 msg/sec. These observations corroborate the results in RQ1 whereby most differences were not statistically significant as the message sizes and logging rates in RQ1 were often low.

We observe that with the increase in logging rate energy consumption also increases; but the same does not apply for message size. However, in case of heavy logging both the rate and the size become significant factors towards energy consumption. This also explains the observed inconsistencies in Table 7. One would expect that with the increase in message size, the energy consumption would also increase. However, no such trend is observed from Table 7 when the message rate is low. For instance, the energy increase in joules for 10 msg/sec with 1.5 KB message size is 2.16%, which is higher than 2 KB message size with the same rate (1.91% increase). This led us to evaluate if the differences in energy consumption with this two settings are really different, because Table 7 only shows the increase in percentage considering the average of the 40 measurements for each scenario.

To investigate the difference across both factors at once, we run a handful of tests to investigate trends depicted in Table 7. We apply the Wilcoxon rank sum test (a non-parametric test), and found that the energy consumption between the above mentioned two settings (10 msg/sec with 1.5 KB log message size versus 10 msg/sec with 2 KB log message size) are not statistically different (p-value > 0.05). The difference is not statistically significant either (p-value > 0.05) for 0.01 msg/sec with 1.5 KB log message size versus 0.01 msg/sec with 2 KB log message size. However, when the message rate is high, the difference in energy consumption with different message sizes are significant. For example, the energy consumption differences are statistically significant (p-value < 0.05) for 1000 msg/sec with 1.5 KB log message size vs. 1000 msg/sec with 2 KB log message size. This is another confirmation that message size only has a noticeable effect with high message rates.

We also build a linear regression model to estimate the energy consumption using the message sizes, the logging rates, and the number of disk flushes. This further clarifies the impact of these factors on energy consumption, as we executed the same application that does nothing than writing log messages. The logging rates and the message sizes are obtained from each test configuration (Table 7). The number of disk flushes can be calculated by dividing the estimated file size with the buffer size. For example, after 120 seconds of testing, the size of the log file from the 1000 msg/sec and 1536 bytes test would be 180,000 KB. Hence, with 256 KB buffer size, the estimated number of disk flushes would be 703. The resulting regression model is shown below as Equation 1 and has an adjusted R-squared value of 0.87.

$$\begin{aligned} \text{joules} = & 0.03370 \times \text{message rate} + \\ & 0.00006 \times \text{message size} + \\ & 0.01328 \times \text{number of disk flushes} + \\ & 112.10958 \end{aligned} \tag{1}$$

This model also confirms that message rate, and subsequently the number of disk flushes are more significant factors for energy consumption than message size. The rationale is that, as we have already shown, with very low

message rate message size does not impact the number of disk flushes significantly.

In summary, our results suggest that mobile application developers do not need to prematurely optimize and trade-off energy consumption for logging. Infrequent logging has limited impact on the overall energy consumption. However, if there is a need to generate large amount of logging content, to conserve energy, it is preferable to log infrequently with larger message sizes rather than logging frequently with smaller message sizes. This is similar to the earlier findings [42,51,32] that bundling smaller packets together reduces significant energy consumption in data communication.

4.4 Summary

Findings: Small amounts of logging ($\leq$ 10 log messages per second) have little or no energy impact on the mobile applications. In fact, message size does not have any significant impact on energy when the logging rate is very low. On the other hand, both the message rate and size are significant factors towards draining energy under heavy logging. Under heavy logging, logging large amounts of data infrequently consumes much less energy than frequently logging smaller amounts of data.
Implications: To conserve energy, developers should strategically instrument their code. The preferred logging points can contain more contextual information but are less frequently executed (e.g., avoid logging within loops or commonly called library functions). When heavy logging is needed, developers should group small log messages and write them together to conserve energy.

5 RQ3: Is there any relationship between the logging events and the energy consumption of mobile applications?

5.1 Motivation

Are the causes of energy consumption, events correlated with energy consumption, apparent in the log? Measuring energy consumption directly often requires both hardware instrumentation and software instrumentation. It is a time-consuming process as hardware test-beds instrumented with a power monitor must run tests multiple times to get a statistically reliable estimate of power use [28,56]. Moreover, such a test-bed might be expensive for many app developers.

The execution logs are debug statements that developers inserted into their code. These instrumentation locations are strategically selected to debug and monitor the functionalities of the applications. The key steps (e.g., displaying the hand-drawn objects or performing email reconciliations) during the executions are often logged and can provide us hints on the energy consumption

patterns of the applications. It would be cheaper and faster for developers to diagnose their mobile application energy regression problems by analyzing their log files.

This RQ investigates the feasibility in terms of using the readily available execution logs to understand the energy consumption of mobile applications. In particular, we want to check if there is any relationship between the logging events and the energy consumption of the mobile applications. The question is what events, that get recorded in the log, induce energy consumption. Thus, we are not seeking to truly estimate energy but we seek to investigate the relationship between common events that get recorded in logs, and energy consumption.

5.2 Experiments

We do not perform additional performance testing in this RQ. Rather, we reuse the log files and the energy measurements from RQ1. In particular, we reuse the measurements of the log-enabled tests of the 24 Android applications, including all of the versions used.

5.3 Analysis

There are three steps involved in this analysis. First, the free-form log messages are abstracted into log events. Second, correlations are calculated between individual log events and energy consumption. Third, we look into the combination of variables using multiple regression—by exhaustive model building we hope to better understand what log events work together to consume energy.

Step 1 - Log Abstraction

Execution logs typically do not follow a strict format. Each log line contains a mixture of static and dynamic information. The static information is the descriptions of the execution events, whereas the dynamic values indicate the corresponding context of these events. For example, the last log line in Table 1 contains static information like "I/ActivityManager", "Force stopping package com.android2.calculator3", "appid" and "user". The numbers "387", "10062" and "0" are likely generated during run-time.

Such free-formed log messages need to be abstracted into events so that they can be used in automated statistical or data mining analysis. We apply the log abstraction technique proposed by Jiang *et al.* [35] to automatically map log messages to log events.

Since the same test scenario was executed for the same version of an application, the generated log events should be similar or even identical. Hence, test runs on the same version are combined into a single log file, by averaging the count of each log event obtained during the repeated test.

Table 8 summarizes the number of unique log events and log length per application across all of the different test runs of their multiple versions.

Table 8 Summary of unique log events per application across all the versions.

App Names	Total Versions	# of Unique Events					Standard Deviations
		total	minimum	median	average	maximum	
2048	44	15	12	13.000	13.091	14	0.603
24game	1	53	53	53.000	53.000	53	
Acrylic Paint	40	19	11	14.000	13.800	17	1.324
Agram	3	15	15	15.000	15.000	15	0.000
AndQuote	21	11	11	11.000	11.000	11	0.000
Blockinger	74	168	23	27.000	27.392	35	1.560
Bomber	79	24	17	20.000	20.152	22	1.292
Budget	59	74	13	37.000	31.525	41	9.482
Calculator	97	53	9	11.000	14.526	23	5.354
ChromeShell	50	105	92	101.000	100.680	104	2.860
DalvikExplorer	13	17	16	16.000	16.308	17	0.480
Exodus	3	88	52	54.000	54.000	56	2.000
Eye in the Sky	1	58	58	58.000	58.000	58	
Face Slim	1	18	18	18.000	18.000	18	
FeedEx	35	39	14	17.000	17.486	28	4.231
Firefox	156	138	29	44.000	52.340	80	22.350
GnuCash	16	399	39	54.500	54.688	62	5.449
Memopad	52	13	12	13.000	12.712	13	0.457
Paint Electric Sheep	1	21	21	21.000	21.000	21	
Sensor Readout	37	12	11	11.000	11.189	12	0.397
Temaki	66	15	8	9.000	9.485	12	1.231
Vector Pinball	54	56	14	16.000	16.463	21	1.342
VLC	46	492	385	390.000	390.022	396	2.679
Wikimedia	58	214	71	96.000	94.000	116	8.840
Average	42	88	42	46.646	46.911	52	3.597

The number of unique log events for each application ranged from 11 unique log events for AndQuote to 492 unique log events for VLC. The mean number of unique log events per application is 88, while the median was 46.646.

Now we look into how the prevalence of unique log events varies within log files from different runs and versions. The total number of unique log events can change across different versions of an application. The average standard deviation of total unique log events per application across versions was 3.597 events with a minimum standard deviation of total log events was 0.000 unique log events (no change) for AndQuote and Agram and a maximum standard deviation of total unique log events 22.350 for Firefox. The statistics about each application are described in Table 8.

Step 2 - Correlation between Log Events and Energy Consumption

Table 9 depicts the Spearman correlation coefficients between each log event and the energy measurements for all the 24 applications.

Table 9 Spearman's ρ correlation coefficient distribution between log event types and joules per application. Each column shows how many log events correlated with the correlation scale proposed by Hopkins *et al.* [31]

App Names	# of Unique Log Events	% Trivial [0.0, 0.1)	% Small [0.1, 0.3)	% Moderate [0.3, 0.5)	% Large [0.5, 0.7)	% Very Large [0.7, 0.9)	% Near Perfect [0.9, 1.0]
2048	15	80.000	13.333	6.667	0.000	0.000	0.000
24game	53	100.000	0.000	0.000	0.000	0.000	0.000
Acrylic Paint	19	36.842	15.789	26.316	21.053	0.000	0.000
Agram	15	80.000	0.000	0.000	6.667	0.000	13.333
AndQuote	11	72.727	27.273	0.000	0.000	0.000	0.000
Blockinger	168	62.500	36.905	0.595	0.000	0.000	0.000
Bomber	24	33.333	25.000	41.667	0.000	0.000	0.000
Budget	74	39.189	22.973	13.514	5.405	18.919	0.000
Calculator	53	18.868	47.170	24.528	9.434	0.000	0.000
ChromeShell	105	22.857	70.476	6.667	0.000	0.000	0.000
DalvikExplorer	17	94.118	0.000	0.000	5.882	0.000	0.000
Exodus	88	21.591	0.000	0.000	21.591	39.773	17.045
Eye in the Sky	58	100.000	0.000	0.000	0.000	0.000	0.000
Face Slim	18	100.000	0.000	0.000	0.000	0.000	0.000
FeedEx	39	38.462	17.949	0.000	35.897	7.692	0.000
Firefox	138	20.290	78.261	1.449	0.000	0.000	0.000
GnuCash	399	21.554	30.576	39.850	7.769	0.251	0.000
Memopad	13	69.231	0.000	0.000	7.692	23.077	0.000
Paint Electric Sheep	21	100.000	0.000	0.000	0.000	0.000	0.000
Sensor Readout	12	91.667	0.000	8.333	0.000	0.000	0.000
Temaki	15	46.667	26.667	26.667	0.000	0.000	0.000
Vector Pinball	56	16.071	23.214	26.786	3.571	30.357	0.000
VLC	492	41.260	13.415	2.236	1.423	41.667	0.000
Wikimedia	214	22.897	1.402	53.738	21.963	0.000	0.000

Spearman's ρ correlation is a non-parametric test that assesses the relationship between two variables. The characterization of the strength of the correlation (trivial, small, medium, large, very large and near perfect) is proposed by Hopkins *et al.* [31]. For example, 205 out of the 492 unique events in VLC exhibit very large correlations with the energy consumption; whereas all the log events in Face Slim have little or no correlation. The correlation values in the table show that 79% of the studied applications have at least one log events which exhibit medium to near perfect correlation values with the energy consumption of the mobile applications. If only large to near perfect correlations are considered, there are still 50% of the studied applications that have some log events strongly correlated with energy consumption.

Across most applications (e.g., Firefox, Agram, Acrylic Paint, and Memopad), the log events with the highest correlation with energy consumption are often related to the Dalvik Virtual Machine, DalvikVM. The DalvikVM is the Java Virtual Machine used by the Android operating system to run mobile applications. The lists of events that are highly correlated with energy consumption are shown below. Some of them are related to identified "energy greedy APIs" [67].

- `Dalvikvm: GC CONCURRENT (X1)`: this event is triggered when the heap starts to fill up;
- `Dalvikvm: GC FOR ALLOC (X5)`: this event occurs when there is not enough memory left on the heap to perform an allocation;
- `Dalvikvm: GROW HEAP (X6)`: in order to save memory, Android does not allocate maximum amount of requested memory to every application automatically. Instead, the OS waits until the application requests more memory. Then this event is triggered to give more heap space until the maximum amount of memory is reached.

For these 3 events, the median magnitude of Spearman's ρ correlation (absolute value) over all applications with more than 1 version is 0.333 for `Dalvikvm: GC CONCURRENT`, 0.308 for `Dalvikvm: GC FOR ALLOC`, and 0.219 for `Dalvikvm: GROW HEAP`. This shows a small to medium relationship between Dalvikvm memory management and energy consumption.

Not all log events that are correlated with energy consumption are common across applications. Some of the highly correlated log events are workload specific for a particular application. For example, in GnuCash, workload-specific log events regarding the *onCreateView* method for the *DatePickerDialog* class, and a log event about replacing account entries in the database exhibited large positive ($\rho = 0.6873$) and very large negative correlations with energy consumption ($\rho = -0.7515$), respectively. For Vector Pinball, trying to load the JNI library for Box2D, a 2D physics library, (`DalvikVM trying to load lib.data.app.lib.com.dozingcatsoftware.bouncy libgdx.box2d.so`) has a very large negative correlation ($\rho = -0.7923$) with joules. VLC has very large positive correlation ($\rho = 0.8377$) with input controls (`VLC core input control stopping input`).

Step 3 - Building Energy Consumption Models Using Logs

In this step, the relationship between these log events and energy consumption is further studied through multiple regression analysis. If an independent variable (i.e., a log event) reoccurs in numerous models, then we argue that that variable demonstrates a strong relationship with software energy consumption for different mobile applications. The intent of this section is not necessarily to build reusable predictors, but to further study the relationship between energy consumption and common log events. We use multiple linear regression to study the relationship between different log events (as independent variables) and the energy consumption (as the dependent variable).

There were 122 log events that commonly occurred across 3 or more applications. When considering common log events shared by four or more applications, there are only a total of 17 log events. There are 10 log events commonly shared by 6 or more applications. Hence, in this step, we pick the common log events that are shared by at least four applications, as we want to derive more general prediction models in order to study the effect of common log events on software energy consumption. Table 10 shows the 17 selected log events.

For the sake of brevity, a short log event name is shown instead of the fully abstracted log events.

Table 10 OS Level Log events shared by all the applications

#	# applications	Event Name
X1	25	dalvikvm GC_CONCURRENT
X2	24	dalvikvm WAIT_FOR_CONCURRENT_GC
X3	21	libEGL loaded vendor lib egl libGLESv2_POWERVR_SGX540_120
X4	20	OpenGLRenderer Enabling debug mode
X5	19	dalvikvm GC_FOR_ALLOC
X6	17	dalvikvm heap Grow
X7	14	dalvikvm Late enabling CheckJNI
X8	8	dalvikvm Turning on JNI app bug workarounds for target SDK version
X9	6	TilesManager Starting TG
X10	6	Choreographer Skipped frames The application may be doing too much work on its main thread
X11	4	dalvikvm Jit resizing JitTable
X12	4	webviewglue nativeDestroy view
X13	4	GLWebViewState Reinit transferQueue
X14	4	dalvikvm null clazz in OP_INSTANCE_OF single stepping
X15	4	InputMethodManagerService Focus gain on non focused
X16	4	dalvikvm VFY replacing opcode
X17	4	GLWebViewState Reinit shader

Many log event counts are highly correlated with other event counts. We have exhaustively tried all subsets of variables in the model and ignored models that included co-linearity whereby any two independent variables had a Pearson correlation greater than 0.3 or less than −0.3. The models are kept if all the independent variables are reported as statistically significant to the model ($p \leq 0.05$). Models that do not produce a significant F-statistic ($p \leq 0.05$) are not kept.

The final models are the ones with the largest number of significant events. Several regression models with more than two log events are found. These models are for both the individual applications and all the applications at once. Table 11 shows the models extracted and their adjusted R-squared values.

Some applications do not have enough versions, or their common log events do not correlate well enough to produce a significant linear model (e.g., 24Game, 2048, and Blockinger). Each model has the following form:

$$\text{joules} \backsim c_0 + c_1 \times \text{event}_1 + c_2 \times \text{event}_2 + ... + c_n \times \text{event}_n \quad (2)$$

In general, different models from the same application share similar prediction performance. The three models from FeedEx show that the top common log events can predict the energy consumption of FeedEx very well (with Adj-$R^2 \geq 0.92$) . However, the models from the Calculator and the Firefox applications show moderate prediction performance with Adj-$R^2 \geq 0.39$

Table 11 Linear models of energy consumption based on log events across numerous Android applications. Top three models are shown only if they are significant ($p \leq 0.05$).

	Events#	Adj-R^2	p-value
All applications/Versions	X4 + X6 + X17	0.4755	1.1222e-140
	X1 + X4 + X6	0.4965	1.4197e-149
	X1 + X4 + X6 + X9	0.5233	2.3270e-160
	X1 + X4 + X6 + X13	0.5328	9.2390e-165
	X1 + X4 + X6 + X17	0.5328	9.2390e-165
Acrylic Paint	X3 + X5	0.4870	1.6392e-06
	X4 + X5	0.4870	1.6392e-06
	X2 + X5	0.5306	3.1700e-07
Bomber	X6 + X14	0.2375	3.1351e-06
	X4 + X6	0.2375	3.1351e-06
	X6 + X13	0.2375	3.1351e-06
Calculator	X3 + X5	0.3991	2.3918e-12
	X5 + X16	0.3991	2.3918e-12
	X5 + X17	0.3991	2.3918e-12
ChromeShell	X1 + X15 + X16	0.1076	2.5888e-02
	X1 + X13 + X16	0.1076	2.5888e-02
	X1 + X2 + X16	0.1112	3.8161e-02
Firefox	X2 + X3 + X11	0.2242	1.3689e-09
	X2 + X3 + X13	0.2242	1.3689e-09
	X2 + X3 + X10	0.2242	1.3689e-09
GnuCash	X1 + X5 + X6	0.5914	3.0130e-04
	X1 + X4 + X5	0.5914	3.0130e-04
	X1 + X3 + X4	0.5914	3.0130e-04
Memopad	X1 + X8 + X12	0.6377	5.9170e-12
	X1 + X8 + X13	0.6377	5.9170e-12
	X1 + X8 + X14	0.6377	5.9170e-12
Sensor Readout	X1 + X2	0.4276	2.8786e-05
	X1	0.4427	4.2210e-06
Budget	X2 + X15	0.8176	7.6227e-22
	X1 + X5	0.8262	1.9788e-22
	X1 + X6	0.8330	6.4611e-23
Vector Pinball	X1 + X5 + X7 + X8	0.9456	3.1336e-32
	X1 + X5 + X8 + X16	0.9456	3.1336e-32
	X1 + X8	0.9465	1.4174e-33
Temaki	X1 + X7 + X15	0.4848	1.2541e-09
	X1 + X3 + X7 + X15	0.4848	1.2541e-09
	X1 + X7 + X15 + X17	0.4848	1.2541e-09
VLC	X1 + X5 + X16	0.6134	5.0341e-10
	X1 + X3 + X5	0.6134	5.0341e-10
	X1 + X5 + X17	0.6134	5.0341e-10
Wikimedia	X3 + X5	0.6396	3.1125e-14
	X1 + X5	0.6666	2.8477e-14
	X2 + X5	0.7289	9.6649e-17
FeedEx	X17	0.9297	8.2694e-21
	X16	0.9297	8.2694e-21
	X13	0.9297	8.2694e-21

and Adj-$R^2 \geq 0.22$, respectively. The CHROMESHELL app is not modeled well by the common log events with an Adj-$R^2 \geq 0.11$, while the VECTOR PINBALL app performs the best for predictability with models with an Adj-$R^2 \geq 0.94$.

Across all the studied version of Android applications, typically events X1, X5, and X6 are part of the successful models. These events are related to the Dalvik Virtual Machine. The listed events are related to memory management operations such as garbage collection and memory allocations. In the models utilizing all applications and versions, X17 and X4 were also quite significant, as well as X1 and X6. X4 and X17 are OpenGL and graphics relevant.

5.4 Summary

Findings: Around 80% of the applications have at least one log event whose correlation with the energy consumption are medium or stronger. Memory management and graphics-related (OpenGL) log events are the most correlated log events related to mobile software energy consumption. For some applications, there are also some workload-specific log events which exhibit high correlation with the energy consumption. Models trained on top common log events demonstrate a clear relationship between those log events and the energy consumption for some but not all mobile applications.
Implications: App developers should watch out for log events related to garbage collection and graphics if they are concerned with the energy consumption of their applications — especially changes in the number of these log events. Furthermore, although logs have been used effectively to debug and troubleshoot functional problems, there is still no clear relation between the logging contents and the energy consumption for some applications. Researchers should investigate into innovative logging approaches which can help debug both energy and other performance-related problems.

6 Threats to Validity

In this section, we discuss the threats to validity.

6.1 Construct Validity

Reliability of the Energy Measurement

It is important to ensure reliable performance measurement, as performance measurement is subject to measurement error and random noise [22, 45, 38]. In this paper, we have used two strategies to mitigate this threat: (1) around 1000 versions from 24 Android applications were studied with both logging

enabled and disabled; and (2) each version of the same application was repeatedly tested and measured to ensure measurement accuracy. We did not have clear control over laboratory temperature, but according to the INA219's specification [**?**], measurements do not deviate much over the range of temperatures expected while running the tests. Energy measures can suffer from sampling, but the INA219 does sample at a high rate and output aggregated measurements at a lower rate. This aggregation can induce error but given the high rate of sampling by the INA219 it is unlikely to have meaningful effect on test runs. Most importantly energy consumption is a physical process thus one must measure multiple times as we do in this paper.

6.2 Internal Validity

Controlling Confounding Factors While Assessing the Energy Impact of Logging on Real-world Android Applications

There are many Android applications that try to log data in a real-world setting. Hence, while executing our performance tests on real-world Android applications, we make sure all the applications under test are running in the same Android running environment. In addition, we also make sure the application under test is the only running user application during the tests. For each application under test, we have executed two types of performance tests: logging enabled and logging disabled. All the test configurations are the same for these two types of tests, except for enabling and disabling logs. We ran the experiments one after the other. Each experiment takes less than five minutes. Hence, for some applications that access outside resources (e.g., FIREFOX requesting data from Wikipedia), the chances of a resource changing during a test (e.g., content updates in the Wikipedia webpage) exists but would be very low.

Controlling Various Logging Factors While Investigating the Energy Consumption of Logging on Android Applications

There are many factors impacting the energy consumption of logging on Android applications. Factors such as the logging rate and the log message sizes cannot be easily controlled on real-world use cases and applications. In addition, real-world applications also perform other tasks (e.g., networking and video playing), which makes it difficult to isolate the energy impact of logging. Hence, to control the various confounding factors, we have developed a testing Android application which is dedicated only to log messages at different rate and size. The values of the logging rates and message sizes were derived based on actual scenarios in practice (e.g., the size of a network packet and the size of a typical stack trace). Since the logging rates and the message sizes combined could have an impact on the size of the log files, an additional factor, the number of disk flushes, is introduced to assess the combined impact of

logging rate and log message size. Yet the Android operating system is a complex piece of software, thus state will slowly change while the operating system running, for instance the file system state will change between runs. Future tests could replace the file system each and every time in order to control the non-determinism in the file system.

6.3 External Validity

Generalizing the Energy Impact of Logging on Real-world Android Applications

To ensure our findings on the energy impact of logging on real-world Android applications are generalizable, we have selected 24 Android applications from different application domains. In addition, many of the applications in our dataset have many versions. These versions correspond to a range of different software development activities (e.g., new features and bug fixes). Increasing both the number of applications and versions covered would provide better generality. However, our findings might not be able to generalize to other mobile application platforms (e.g., BlackBerry, iOS or Windows phones) and other Android phones.

In addition, although we have designed our test cases to closely mimic the realistic user usage of mobile applications, the resulting test cases may not cover all the possible uses for the studied applications.

7 Related Work

In this section, we will discuss three areas of prior research that are related to this paper: (1) energy testing and modeling for mobile applications, (2) empirical studies on energy-efficient mobile development, and (3) execution logs.

7.1 Energy Testing and Modeling for Mobile Applications

Hindle *et al.* developed the GreenMiner, an automated test-bed to assess the energy consumption for each revision of a given mobile application [28,29]. Since running performance tests on each revision is time consuming, Romansky *et al.* [56] proposed a search-based test approximation technique to reduce the testing efforts in GreenMiner. Li *et al.* [41] proposed a test minimization technique that prioritizes the test suites with higher energy consumption. This paper leverages the GreenMiner [28,29] to perform energy testing on different versions of the mobile applications with and without logging enabled.

There have been many studies dedicated to modeling energy consumption for mobile applications. In general, there are three approaches, which use three different datasets, collected by different monitoring and profiling tools, to model mobile energy consumption: (1) hardware-based counters [13,

64,25,20,75,18]; (2) program instructions from the applications [60,26,40]; and (3) system calls [7,16,51]. Different mobile monitoring and profiling tools can bring different insights into the mobile applications' dynamic behavior. However, they all have some runtime overhead. Different from the above three approaches, this paper builds the energy consumption models to explore factors of energy consumption prevalent in execution logs.

7.2 Empirical Studies on Energy-efficient Mobile Development

We further divide the empirical studies on energy-efficient mobile development into the following three sub-areas:

- **App Developers:** Pinto *et al.* [53] investigated questions on StackOverflow that programmers had about energy. They found that programmers lacked the resources to answer questions about software energy consumption. Similar findings were confirmed by other studies that surveyed programmers about their understanding on software energy consumptions [44, 48]. Chowdhury *et al.* [15] compared energy-aware software projects with projects that did not consider energy-efficiency as one of the non-functional requirements. They found that energy-aware software projects are more popular in terms of number of forks, and contributors.
- **Code Obfuscation:** Sahin *et al.* studied the impact of code obfuscation [58] and refactoring [57] on energy consumption of several Android applications. They found that code obfuscation does impact energy consumption but the differences could be too small for users to notice, whereas the impact of code refactoring could be mixed (a.k.a., either increases or decreases in energy consumption).
- **Energy Greedy APIs, Frameworks, and Platforms:** Li *et al.* [39] leveraged their technique of estimating energy consumption for source lines in [40] and studied the API level energy consumption patterns of different mobile applications. They found that the networking component consumes the most energy and more than half of the energy consumption is spent on idle state. This observation indicates that reducing the number of idle states can optimize energy consumption for mobile applications. Linares *et al.* [67] identified energy greedy Android APIs that can be helpful for the developers to write energy efficient code. Chowdhury *et al.* [14] found that employing HTTP/2 server can save energy for the mobile clients. Pathak *et al.* suggested that around 70% of mobile software energy bugs are the direct result of problems linked to wake locks [49]. Hence, many studies have focused on understanding and optimizing wake lock in mobile applications [50,8,43,11,68,52]. Tail energy leaks, the energy cost of powering up and eventually powering down peripherals, have been studied as a source of energy consumption in mobile applications [51,14,42]. Tail energy leaks can be optimized by bundling I/O operations together [14,42]. Hasan *et al.* [27] studied the energy profiles of frequently used Java collection classes

and suggested that using the most energy efficient collection classes can save up to 300% software energy.

Logs are widely used in software development for various purposes like debugging, monitoring and user behavior tracking. However, there are no prior studies focused on the energy impact of logging for mobile applications. Hence, in this paper, we performed an empirical study on another aspect of energy-efficient mobile development: the energy consumption of software logging.

7.3 Execution Logs

We will discuss two areas of related research on execution logs:

- **Empirical Studies on Execution Logs:** There have been a few empirical studies conducted to investigate the logging activities in practice. Shang *et al.* [61] analyzed how log events evolve over time by executing the same scenarios across different versions of the same applications. They found that logs related to domain level events (e.g., workload) are less likely to change compared to logs related to feature implementations (e.g., opening a database connection). Yuan *et al.* [73] analyzed the source code revision history for 4 C-based open source software systems. They found that log events are often added as "after-thoughts" (a.k.a., after failure happens). They also developed a verbosity-level checker to automatically detect anomalous log levels (e.g., DEBUG vs. FATAL) using clone analysis. Fu *et al.* [21] performed a similar log characteristic study but on the source code of two large industry systems at Microsoft. Shang *et al.* [63] studied the release history of two open source applications (Hadoop and JBoss) and found that files with many logging statements have higher post-release defect densities than those without. Unfortunately, all of the prior empirical studies on execution logs focused on desktop and server-based applications. This paper is the first research work focused on studying the execution logs on mobile applications.
- **Analyzing Execution Logs** Execution logs have been used extensively by developers, testers and system operators to monitor and diagnose problems for large-scale software systems [47, 72]. Execution logs have a loosely-defined structure and a large non-standardized vocabulary. Due to its sheer volume of size (hundred megabytes or even terabytes of data), it is usually not feasible to analyze the logs manually. Techniques have been proposed to automatically abstract the loosely structured execution log events into regularized log events [35, 71]. Then automated statistical and AI techniques can be applied on these regularized log events to analyze the results of load tests [36], and to monitor, detect and diagnose problems in big data applications [62, 71, 70]. In addition to leveraging the existing logs, Yuan *et al.* proposed a technique to automatically suggest logging points to aid the debugging activities using program analysis [74]. Finally, Ding *et al.* [17] proposed a cost-aware logging mechanism so that informative logs can be

generated while still ensuring the performance overhead is within the specified budget. Their performance overhead is defined in terms of resource usage (e.g., CPU, memory and disk I/O) and their target applications are large-scale server applications. In this paper, we used the log abstraction technique proposed by Jiang *et al.* [35] to build our energy consumption model.

8 Conclusion

Software developers use execution logs to debug and monitor the health of mobile applications. This paper investigates the energy impact of execution logs on Android applications. Around 1000 versions of 24 Android applications were tested and measured under logging enabled and disabled. In addition, a controlled experiment with varying rates of logging and sizes of the log messages was carried out.

Our experiments show that limited logging (e.g. $\leq$ 1 msg/sec) has little to no impact on the energy consumption of mobile applications. Although there is little to no impact on the energy consumption of logging for most of the versions, there are still many versions with medium to large effect sizes when comparing the energy consumption between when logging is enabled and logging is disabled. The rate of logging, the size of log messages, and the number of disk flushes are three statistically significant factors that impact the energy consumption of logging. Log events can be used in energy consumption debugging as some events common across applications, that are logged as log events, are highly correlated with energy consumption—especially those regarding garbage collection or graphics. Depending on the application, some workload-specific log messages are also correlated with energy consumption. However, building energy consumption models with log events yield mixed performance. It would be an interesting future work to leverage event logs, as a proxy to predict the energy consumption of applications.

In conclusion we have presented evidence that logging under relatively liberal conditions of less than 1 log message per second does not have a significant effect on energy performance. Furthermore we have shown with numerous existing Android applications that logging typically has a negligible effect on energy consumption. Although there are some log events recorded in logs which are highly correlated to the energy consumption of the mobile applications, it is still an open research question on how one can leverage software logging to debug energy problems.

Replication Package

To aide replicability, we freely disclose and share our dataset and source code for our analysis in our replication package [3]. The GreenOracle tests that were run on the GreenMiner are located at https://github.com/shaifulcse/-GreenOracle-Data/tree/master/Tests.

Acknowledgements

Shaiful Chowdhury is grateful to the Alberta Innovates - Technology Futures (AITF) to support his PhD research. Both Abram Hindle and Zhen Ming (Jack) Jiang are supported by an NSERC Discovery Grant.

References

1. Chukwa - hadoop wiki. http://wiki.apache.org/hadoop/Chukwa. Last accessed 04/18/2015.
2. logstash - open source log management. http://logstash.net/. Last accessed 04/18/2015.
3. Replication Package Android Logcat Energy Study. https://archive.org/details/ReplicationPackageAndroidLogcatEnergyStudy. Last accessed 05/24/2017.
4. Splunk. http://www.splunk.com/. Last accessed 04/18/2015.
5. Summary of Sarbanes-Oxley Act of 2002. http://www.soxlaw.com/.
6. K. Aggarwal, A. Hindle, and E. Stroulia. Greenadvisor: A tool for analyzing the impact of software evolution on energy consumption. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 311–320, Sept 2015.
7. K. Aggarwal, C. Zhang, J. C. Campbell, A. Hindle, and E. Stroulia. The Power of System Call Traces: Predicting the Software Energy Consumption Impact of Changes. *Proceedings of the 2014 Conference of the Center for Advanced Studies on Collaborative Research (CASCON)*, 2014.
8. F. Alam, P. R. Panda, N. Tripathi, N. Sharma, and S. Narayan. Energy optimization in android applications through wakelock placement. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–4, March 2014.
9. Android Open Source Project. Andriod API for Logging. http://developer.android.com/reference/android/util/Log.html. Last accessed 02/10/2015.
10. Android Open Source Project. logcat. http://developer.android.com/tools/help/logcat.html. Last accessed 02/10/2015.
11. Banerjee, Abhijeet and Chong, Lee Kee and Chattopadhyay, Sudipta and Roychoudhury, Abhik. Detecting Energy Bugs and Hotspots in Mobile Apps. In *FSE 2014*, pages 588–598, Hong Kong, China, Novemeber 2014.
12. Y. Benjamini and Y. Hochberg. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(1):289–300, 1995.
13. A. Carroll and G. Heiser. An Analysis of Power Consumption in a Smartphone. In *Proceedings of the USENIXATC'10*, 2010.
14. S. Chowdhury, S. Varun, and A. Hindle. Client-side Energy Efficiency of HTTP/2 for Web and Mobile App Developers. In *SANER '16*, Osaka, Japan, March 2016.
15. S. A. Chowdhury and A. Hindle. Characterizing energy-aware software projects: Are they different? In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 508–511, New York, NY, USA, 2016. ACM.
16. S. A. Chowdhury and A. Hindle. Greenoracle: Estimating software energy consumption with energy measurement corpora. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR)*, 2016.
17. R. Ding, H. Zhou, J.-G. Lou, H. Zhang, Q. Lin, Q. Fu, D. Zhang, and T. Xie. Log2: A cost-aware logging mechanism for performance diagnosis. In *2015 USENIX Annual Technical Conference (USENIX ATC)*, 2015.
18. M. Dong and L. Zhong. Self-constructive High-rate System Energy Modeling for Battery-powered Mobile Systems. In *Proceedings of the MobiSys '11*, pages 335–348, June 2011.
19. A. Fedotyev. The Real Cost of Logging. http://blog.appdynamics.com/net/the-real-cost-of-logging/, April 2014. Last accessed 04/18/2015.

20. J. Flinn and M. Satyanarayanan. PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications. In *WMCSA '99*, pages 2–10, New Orleans, Louisiana, USA, February 1999.
21. Q. Fu, J. Zhu, W. Hu, J.-G. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie. Where do developers log? an empirical study on logging practices in industry. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, 2014.
22. A. Georges, D. Buytaert, and L. Eeckhout. Statistically rigorous java performance evaluation. In *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented Programming Systems and Applications (OOPSLA)*, 2007.
23. A. Grabner. Top Performance Mistakes when moving from Test to Production: Excessive Logging. http://tinyurl.com/q9odfsm, August 2012. Last accessed 04/18/2015.
24. T. O. Group. Application Response Measurement - ARM. https://collaboration.opengroup.org/tech/management/arm/, visited 2014-11-24.
25. S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L. K. John. Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, HPCA '02, pages 141–150, 2002.
26. S. Hao, D. Li, W. G. J. Halfond, and R. Govindan. Estimating Mobile Application Energy Consumption Using Program Analysis. In *ICSE '13*, pages 92–101, 2013.
27. S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, and A. Hindle. Energy profiles of java collections classes. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 225–236, New York, NY, USA, 2016. ACM.
28. A. Hindle. Green mining: a methodology of relating software change and configuration to power consumption. *Empirical Software Engineering*, 2013.
29. A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky. GreenMiner: A Hardware Based Mining Software Repositories Software Energy Consumption Framework. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*, 2014.
30. M. Hlavac. *stargazer: Well-Formatted Regression and Summary Statistics Tables*. Harvard University, Cambridge, USA, 2015. R package version 5.2.
31. W. G. Hopkins. A new view of statistics. [Online accessed 2017-01-15] http://www.sportsci.org/resource/stats/index.html, 2016.
32. R. Jabbarvand, A. Sadeghi, H. Bagheri, and S. Malek. Energy-aware test-suite minimization for android apps. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ISSTA 2016, pages 425–436, 2016.
33. A. Jay. Do unused logging statements affect performance in Android apps? Stack Overflow http://tinyurl.com/ncf2nl9.
34. A. Jay. Log.d and impact on performance. Stack Overflow http://stackoverflow.com/questions/3773252/log-d-and-impact-on-performance. Last accessed 04/18/2015.
35. Z. M. Jiang, A. E. Hassa, G. Hamann, and P. Flora. An automated approach for abstracting execution logs to execution events. *Journal Software Maintenance Evolution*, 20:249–267, July 2008.
36. Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora. Automatic Identification of Load Testing Problems. In *Proceedings of the 24th IEEE International Conference on Software Maintenance (ICSM)*, 2008.
37. JoJo. Does logging slow down a production Android app? Stack Overflow http://stackoverflow.com/questions/6445153/does-logging-slow-down-a-production-android-app. Last accessed 04/18/2015.
38. T. Kalibera and R. Jones. Rigorous Benchmarking in Reasonable Time. In *Proceedings of the 2013 International Symposium on Memory Management (ISMM)*, 2013.
39. D. Li, S. Hao, J. Gui, and H. William. An Empirical Study of the Energy Consumption of Android Applications. In *Proceedings of the 30th International Conference on Software Maintenance and Evolution (ICSME)*, 2014.
40. D. Li, S. Hao, W. G. J. Halfond, and R. Govindan. Calculating Source Line Level Energy Information for Android Applications. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis (ISSTA)*, 2013.

41. D. Li, Y. Jin, C. Sahin, J. Clause, and W. G. J. Halfond. Integrated Energy-directed Test Suite Optimization. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA)*, 2014.
42. D. Li, Y. Lyu, J. Gui, and W. G. J. Halfond. Automated energy optimization of http requests for mobile applications. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 249–260, New York, NY, USA, 2016. ACM.
43. Y. Liu, C. Xu, S. Cheung, and V. Terragni. Understanding and detecting wake lock misuses for android applications. In *FSE 2014*, Seattle, WA, USA, Nov 2016.
44. I. Manotas, C. Bird, R. Zhang, D. Shepherd, C. Jaspan, C. Sadowski, L. Pollock, and J. Clause. An empirical study of practitioners' perspectives on green software engineering. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 237–248, New York, NY, USA, 2016. ACM.
45. T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney. Producing wrong data without doing anything obviously wrong! In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2009.
46. J. Neill. Why use effect sizes instead of significance testing in program evaluation, 2008.
47. A. Oliner, A. Ganapathi, and W. Xu. Advances and Challenges in Log Analysis. *Commun. ACM*, 55(2):55–61, Feb 2012.
48. C. Pang, A. Hindle, B. Adams, and A. E. Hassan. What do programmers know about the energy consumption of software? *IEEE Software*, pages 83–89, 2015.
49. 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*, HotNets-X, pages 5:1–5:6, 2011.
50. 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 *EuroSys '12*, pages 29–42, Bern, Switzerland, April 2012.
51. A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-grained Power Modeling for Smartphones Using System Call Tracing. In *EuroSys '11*, pages 153–168, Salzburg, Austria, April 2011.
52. P. S. Patil, J. Doshi, and D. Ambawade. Reducing power consumption of smart device by proper management of wakelocks. In *Advance Computing Conference (IACC), 2015 IEEE International*, pages 883–887, June 2015.
53. G. Pinto, F. Castor, and Y. D. Liu. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*, 2014.
54. 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*, GREENS 2014, pages 38–45, 2014.
55. J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen's d for evaluating group differences on the NSSE and other surveys? In *Annual Meeting of the Florida Association of Institutional Research*, February 2006.
56. S. Romansky and A. Hindle. On Improving Green Mining For Energy-Aware Software Analysis. *Proceedings of the 2014 Conference of the Center for Advanced Studies on Collaborative Research (CASCON)*, 2014.
57. C. Sahin, L. Pollock, and J. Clause. How do code refactorings affect energy usage? In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2014.
58. C. Sahin, P. Tornquist, R. McKenna, Z. Pearson, and J. Clause. How Does Code Obfuscation Impact Energy Usage? In *Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2014.
59. sandalone. Android debugging -¿ battery drains. Stack Overflow http://stackoverflow.com/questions/4958543/android-debugging-battery-drains. Last accessed 04/18/2015.
60. C. Seo, S. Malek, and N. Medvidovic. Component-level energy consumption estimation for distributed java-based software systems. volume 5282 of *Lecture Notes in Computer Science*, pages 97–113. Springer Berlin Heidelberg, 2008.

61. W. Shang, Z. M. Jiang, B. Adams, A. E. Hassan, M. W. Godfrey, M. Nasser, and P. Flora. An Exploratory Study of the Evolution of Communicated Information About the Execution of Large Software Systems. In *Proceedings of the 18th Working Conference on Reverse Engineering (WCRE)*, 2011.
62. W. Shang, Z. M. Jiang, H. Hemmati, B. Adams, A. E. Hassan, and P. Martin. Assisting Developers of Big Data Analytics Applications when Deploying on Hadoop Clouds. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, 2013.
63. W. Shang, M. Nagappan, and A. E. Hassan. Studying the relationship between logging characteristics and the code quality of platform software. *Empirical Software Engineering*, 20(1), 2015.
64. A. Shye, B. Scholbrock, and G. Memik. Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures. In *IEEE/ACM MICRO 42*, pages 168–178, New York, NY, USA, December 2009.
65. B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag. Dapper, a Large-Scale Distributed Systems Tracing Infrastructure. Technical report, Google, Inc., 2010.
66. B. Thompson. A suggested revision to the forthcoming 5th edition of the apa publication manual, 2000.
67. M. L. Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. D. 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 (MSR)*, 2014.
68. X. Wang, X. Li, and W. Wen. Wlcleaner: Reducing energy waste caused by wakelock bugs at runtime. In *Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on*, pages 429–434, Aug 2014.
69. M. Woodside, G. Franks, and D. C. Petriu. The future of software performance engineering. In *Proceedings of the Future of Software Engineering (FOSE) track, International Conference on Software Engineering (ICSE)*, 2007.
70. W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan. Online System Problem Detection by Mining Patterns of Console Logs. In *Proceedings of the Ninth IEEE International Conference on Data Mining (ICDM)*, 2009.
71. W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan. Detecting Large-scale System Problems by Mining Console Logs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP)*, 2009.
72. D. Yuan, H. Mai, W. Xiong, L. Tan, Y. Zhou, and S. Pasupathy. SherLog: Error Diagnosis by Connecting Clues from Run-time Logs. In *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2010.
73. D. Yuan, S. Park, and Y. Zhou. Characterising Logging Practices in Open-Source Software. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, 2012.
74. D. Yuan, J. Zheng, S. Park, Y. Zhou, and S. Savage. Improving Software Diagnosability via Log Enhancement. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011.
75. 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 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, 2010.