On the Impact of Sampling Frequency on Software Energy Measurements

Rubén Saborido[1], Venera Arnaoudova[4], Giovanni Beltrame[2], Foutse Khomh[3], Giuliano Antoniol[1]
SOCCER[1]–MIST[2]–SWAT[3] [1] Labs., DGIGL, Polytechnique Montréal, Canada
[4] Dep. of Computer Science, The University of Texas at Dallas

***Abstract*—Energy consumption is a major concern when developing and evolving mobile applications. The user wishes to access fast and powerful mobile applications, which is usually in contrast to optimized battery life and heat generation. The software engineering community have acknowledged the relevance of the problem and researchers are investigating ways to reduce energy consumption, for example by examining which library, device configuration, and applications parameters should be used to promote long battery life. We conjecture that these studies are at the border between hardware and software and we must be careful on how the energy consumption is measured and how the energy consumption is attributed to methods and libraries.**

To the best of our knowledge, no previous work investigates how much energy and power consumption is due to high frequency events missed when sampling at low frequencies such as 10 kHz and verified the error at the precision of method level. Low frequency sampling is a rough approximation that hinders the understanding of fine grain details: the real picture of energy consumption as well as the root causes are missed. This has profound implications on the choice of methods to evolve or components to replace. In this paper, we propose an approach for accurate measurements of the energy consumption of mobile applications. We apply the proposed approach to assess the energy consumption of 21 mobile, closed source, applications and four open source Android applications.

We show that by sampling at 10 kHz one may expect a median error of 8%, however, such error may be as high as 50% for short fast executing methods. Finally, we revisit a previous approach that estimates the energy consumption of methods based on execution time and found that it can miss as much as 84% of the energy, with a median of 30%.

***Index Terms*—Software Energy Consumption, Performance, Android, Monitoring.**

I. Introduction

With the current trend of pervasive mobile devices, which will eventually lead to the Internet-of-Things, there is an increasing interest in reducing the energy consumption of mobile applications, and therefore prolonging the time between battery recharges. Pinto *et al.* [17] analysed more than 300 questions and 550 answers on the popular StackOverflow question-and-answer site for developers and found that the number of questions on energy consumption increased by 183% from the first trimester of 2012 to 2013. The majority of those questions were related to software design, showing that developers need guidance for designing, maintaining, evolving green applications. Another clear sign of the relevance of this problem is the Google Volta Project[1], which aims to improve the battery life of Android devices by creating a new more energy aware Android version.

Battery usage has complex dependencies on the hardware platform, and multiple software layers. The hardware, its firmware, the operating system, and the various software components used by an application, all contribute to determining its energy footprint. Evolving, updating software components or changing component configuration may have a profound impact on battery life. The classical approach to fine-grained software power modelling is to collect measures for single instructions [21]. These values are then back-annotated to higher level languages. However, such an approach produces only approximated results, plus it is cumbersome as modern processors implement pipelines and superscalar and multi-processor architectures that often execute more than one instruction per cycle.

Researchers have recently relied on platforms like Atom-LEAP [20] or Monsoon power monitor [22] to acquire power measurements. However, for both devices the sampling frequency is at maximum 10 kHz. These are important steps forward in understanding the impact of software on energy consumption.

We claim that recent investigations in software engineering related to mobile energy consumption are inaccurate as they are biased by an excessively low sampling frequency, plus the adopted methodology prevents the collection of accurate and precise measures at method level. To support our claim, we design an approach that enables accurate measuring of energy consumption at higher sampling frequencies (*i.e.,* above 10 kHz) up to the method level. Using this approach, we compare energy consumption measurements of 21 (closed source) Android applications at sampling rates of 60 Hz, 5 kHz, 10 kHz, 125 kHz and 500 kHz. Our findings show that an important fraction of the power is consumed at high frequencies and thus missed by current approaches, and the error can be as high as 50%. We also observed that only 1% of energy is consumed above 125 kHz, hence we claim that 125 kHz is sufficient to measure the power consumption of mobile applications. We further show how accurate measurements can be obtained at method level and we assess the error made by current work that attributed energy to methods proportionally to their execution times. We found that these works may have underestimated as much as 84% of the energy, with a median of 30%.

The contributions of this paper are:

[1]http://goo.gl/7QDTgk

1) Empirical evidence that a higher sampling rate is needed;
2) A methodology for more accurate measurements of energy consumption at application and method levels;
3) Evidences that energy should not be attribute to methods simply by considering their execution times.

Paper organization. The remainder of this paper is organized as follows. Section II discusses related literature on energy consumption monitoring. Section III describes our proposed methodology for more accurate measurements of energy consumption, while Section IV presents our case study, and Section V the results obtained showing the limitation of current approaches. Finally, Section VI comments threats to validity, and Section VII draws our conclusions and lays out directions for future work.

II. Related Work

There is a rich literature on energy consumption monitoring, especially in the area of embedded devices, as such devices only have limited battery power. The earliest work on energy consumption focused on modeling, monitoring, and improving energy consumption at the hardware level (*e.g.,* [2], a detailed survey of these works is presented in Hindle [9], [10]). Later on, focus shifted to techniques to model, monitor, and improve energy consumption at the level of the operating system and applications. For example, Bohrer *et al.* monitor the power consumption in web servers [1] using a sense resistor in series with the systems to be measured. The signals from the sense resistors are filtered with a 10 kHz low pass filter and passed to a PCI-6071E A-to-D board from National Instruments. Hindle [8], [10] measures the power using an external power monitor called the *Watts Up? Pro*, operating at a frequency of 60 Hz, and developed a test-bed that can be used to assess the energy consumption of different revisions of a mobile application. Vásquez *et al.* [22] mine energy-greedy API usage patterns in Android applications using a Monsoon [15] power monitor capable of sampling power at 5 kHz frequency. Li *et al.* [11] also use a Monsoon monitor to investigate best energy-saving programming practices at 5 kHz sampling rate. Hao *et al.* [7] estimate the power consumption of Android applications at a fine-grained level (per-instruction). They use a Low Power Energy Aware Processing (LEAP) power measurement device (Atom-LEAP) [20] operating at 10 kHz. Using the same measurement device, Li *et al.* measured the energy consumption of source code lines [13] and studied the API level energy consumption patterns of 405 mobile applications [12]. They found that networking components consume more energy than other components and also that half of the energy consumption is spent on idle state. Li *et al.* also proposed a test minimization technique that prioritizes the test suites with higher energy consumption [14], using Atom-LEAP. Similarly using Atom-LEAP, Manotas *et al.* [6] provide recommendation systems to support developers in coding more energy aware applications All of these previous works suffer from one main limitation, *i.e.,* the frequency of energy measurement reaches 10 kHz at best.

III. Methodology

This section is organized following the logic of a primer on energy measurement, signal acquisition and accurate energy measurement. Due to the limited space, the description is necessarily very succinct and missing background on signal theory, Fourier and spectral analysis, time domain analysis, as well as other non-essential details. We refer the interested reader to the classic signal processing books [3], [16].

A. A Primer on Energy Measurement

Given a device with input voltage V and current I, the input power can be computed at the device power supply via Ohm's law *i.e.,* $P = V \cdot I$. In general, both voltage and current are functions of time. Thus, at a given instant t, the absorbed power is the product of the voltage, $V(t)$, and the current $I(t)$: $P(t) = V(t) \cdot I(t)$. The energy consumed in the interval $[0, T]$, is then computed as the integral over time of the power $P(t)$:

$$E_T = \int_0^T P(t)dt \tag{1}$$

In other words, the energy is the area under the power curve.

If we assume that measures are taken at discrete intervals $\Delta\tau$, *i.e.,* they are sampled with a sampling frequency $F_c = 1/\Delta\tau$, two samples are $\Delta\tau$ seconds apart [16], the energy is approximated assuming a constant power between two measurements and the integration is replaced by a summation:

$$E_T \simeq \sum_k P(k\Delta\tau)\Delta\tau \tag{2}$$

B. Signal Acquisition

The choice of the sampling frequency $F_c = 1/\Delta\tau$ (measured in hertz Hz, events per second), is critical and impacts the accuracy of energy consumption estimate. A too low frequency can be misleading: all events (power peaks) between two samples are lost and averaged away. Even worse, a method execution lasting less than $\Delta\tau$ seconds will be completely lost. Indeed, the maximum frequency that is observable in a signal sampled with frequency F_c is $F_c/2$, known as the Nyquist frequency F_N [16]. We surmise that all recent investigations in software engineering related to mobile energy consumption are biased by an insufficient sampling frequency.

The time domain analysis has its counterpart in the frequency domain. Time varying signals can be represented via Fourier transform in the frequency domain by the signal spectra. For discrete signals, the passage between time and frequency domains is performed via discrete Fourier transform.

The Fourier transform is an elegant way to obtain the signal representation as sum of sine waves of various amplitudes (Fourier series). Actually, the k-th line of the discrete Fourier transform, *i.e.,* line of the signal spectrum, corresponds to the amplitude of the k-th sine wave and the zero line is the continuous component. Thus, to obtain the power present in the signal between two given frequencies, say between 1 Hz and 5 kHz, it is sufficient to sum the square of the spectral

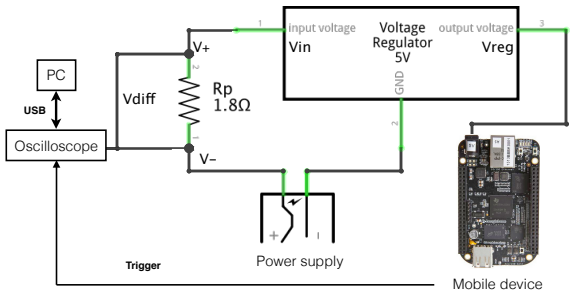

Fig. 1. Measurement setup.

lines in that frequency range. Notice that in any case we are not interested in the continuous power (spectral line at zero); this non-dynamic part is due to hardware and is present even if nothing is executing. In fact, the non-dynamic contribution are bound to be less and less important as hardware architectures become more efficient: active power management systems can greatly reduce the baseline power of a device, that is the power when the system is not or only partially active.

The signal (power in our context) sampling frequency fixes the maximum frequency that one will observe (with no distortion) in the signal spectra. More precisely, if a signal with a maximum frequency F_M is sampled with a frequency lower than $2 * F_M$, his spectral representation will be distorted by the aliasing effect [16]. Indeed, for a maximum frequency of F_M the Nyquist frequency will be F_M and thus the sampling frequency needed for an exact signal reconstruction (and no spectral distortion) is $F_c = 2 * F_M$.

The number of bits used in the signal acquisition has an impact on the quality of the obtained signal. The signal input amplitude is quantized into a set of discrete values; assuming an uniform quantization, the smallest observable change in signal amplitude is the quantization delta, *i.e.,* roughly the number of possible bits configuration. Modern analog to digital converters typically use 12, 16 or 24 bits (with 16 bits deemed sufficient for hi-fi quality).

Incidentally, we adopted an approach inspired by the hi-fi signal acquisition sampling at high frequency and the down-sampling at the final sampling rate. This process ensures a reduction of noise, while keeping the signal as close as possible to the original signal once reconstructed. Our measurement apparatus uses a proprietary adaptive quantization which ensures a sensitivity of 5 mV.

C. Accurate Energy Measurement

There are three approachable ways to measure and compute $P(t)$ and thus E_T. The first and most intuitive way, would be to measure the voltage and the current at the same time on different channels of some device. For example, an oscilloscope or a dedicated analog to digital board plus a PC to acquire and store the sampled signal, then calculate the power $V(k\Delta)I(k\Delta)$ for each measurement, summing up to obtain E_T. However, accurately synchronizing the voltage and the current measures is far from obvious, and the smallest glitch or difference in measuring cables properties (*e.g.,* length) would result in a temporal shift and thus in imprecise power calculation. Furthermore, multi-channel fast and precise acquisition boards are rather expensive. A more accurate way, with less risk of miscalculation, is to use a power analyzer or a clamp meter that allows to measure both the voltage and the current at the same time. For accurate measures with low current/voltage and high frequency, such device can cost a few thousands of dollars at the time of writing.

In this paper, we propose a third way, which is to use a medium/high end digital oscilloscope (triggered by the mobile device/application) and measure only the current via a resistor as shown in Fig. 1 . We need three components: (i) a stabilized power supply that provides a higher voltage than what the device requires, (ii) a precise power regulator that will bring the voltage down to the device voltage, say 5 V, *i.e.,* the mobile device input voltage, and (iii) a high precision (1% metal film) resistor placed before the input of the power regulator.

Fig. 1 shows the setup we used for the measurement. The regulator is used to stabilize the voltage, and the known resistor R_p is used to measure the current flow. With such setup the power consumed by a device, at a given instant t, is computed by the product of the voltage, $V_{reg}(t)$, and the current $I(t)$, which is easily obtained: $I = \frac{V_{diff}}{R_p}$, where V_{diff} is the drop of voltage on the extremities of the resistor R_p. It is worth commenting on the position of the resistor R_p. At a first look it may seem wrong as it is on the input circuit. However, this ensures that $V_{reg}(t)$ is exactly clamped at the nominal value. Once the regulator is warmed up, its energy consumption can be considered constant thus $V_{diff}(t)$ is an accurate estimate (minus a constant factor) of $P(t)$ *i.e.,* of sampled value $P(k\Delta T)$. If the resistor was placed between the source $V_{reg}(t)$ and the device, it would be much more complicated to ensure that the actual voltage input of the device remains fixed. Due to the time varying drop $R_pI(t)$ the input voltage would then be $V_{reg}(t) - R_pI(t)$. In such configuration, either a more complex setup is needed or both voltage and current must be measured, forcing the use of two measurement channels.

A key element of our measurement approach is the presence of a trigger signal, a signal raised by the mobile application/device activating the signal acquisition. This is crucial to ensure 1) the synchronization between code execution and sampled input values and 2) that only what is really needed is measured. This setting greatly simplifies the separation between, startup, environment setup and configuration, real measurement phase and final tear down step.

The macro phases of signal acquisitions are illustrated in the pseudo-code of Algorithm 1.First and foremost one must make sure that the cabling is correct, the right sampling frequency is selected, and the proper probe impedance is set up. Then the mobile device is activated, the execution environment set up and configured, and the application under study loaded.

The application under study must be instrumented properly to ensure data collection. Typically, there will be some code section devoted to initialization, followed by method tracing starts. At this point the application has to call a specific method

Algorithm 1 Accurate Energy Measurement Setup and Tear Down Pseudo Code.

```
Require: Setup Oscilloscope frequency, sampling probe and trigger probe connection.
Ensure: Mobile Device is Running
Ensure: Mobile Application is Loaded
  Environment Setup and Configuration
  Start Monitored Application
  if Collecting Method Trace then
      Start Debugger
  end if
  SignalAcquisition(True)
  Execute Code under Study
  SignalAcquisition(False)
  if Collecting Method Trace then
      Stop Debugger
  end if
  Stop Monitored Application
  Environment Tear Down
```

to raise the trigger signal (and therefore one needs access to the source code). This method is device dependent and is used to raise a tension value on a device pin, led, or output port. Once the code section under measure has been executed, the trigger signal is lowered to stop signal acquisition. Then the tracing is stopped, the collected data saved and the environment cleaned up. Notice that, the two method calls to start and stop the signal acquisition are of course included in the trace but, since the method names are known, they are easily removed. Finally, notice that our approach can be used to collect information at various granularities. However, the limitation is in the data acquisition sampling frequency: the finer the granularity, the higher the sampling frequency should be.

IV. Case Study

The overarching *goal* of this paper is to provide a guidance to developers helping them to better gauge and understand the energy consumption role of various components. In our case study we focus on Android applications. Android applications live in an ecosystem of Android provided components and third party applications. Even if a given application does not change, when Android evolves, or the user installs new components, the unchanged application can be positively or negatively impacted by these changes in the ecosystem.

Our experiments and associated analysis have three specific objectives: (1) show that our measurement setup has sufficiently low noise for measuring fine-grained changes in power consumption at the method level; (2) show that previous works do not take into account a portion, sometime quite large, of the power consumption spectrum (namely the high frequencies)

The *quality focus* of our work is to improve the accuracy of the energy consumption diagnostic of Android applications. We surmise that precise energy measures are key to identify which parts of an application have to be evolved or which application's component is responsible for unexpected energy consumption. The *perspective* is that of researchers interested in developing accurate energy consumption measurements techniques and developers interested in estimating the energy consumption of their components and-or methods. It is important to understand that a certain level of imprecision may be considered physiological and tolerable. As in any engineering discipline there is always a tension between cost and accuracy, in this work, we aim to provide a guidance and some error bounds letting developers decide if the accuracy they may expect is "just good enough" or if a better setup is needed.

The *context* of our study consists of 21 closed source Android applications belonging to nine different domains (see Table I), four open source applications and a hardware setup presented in Fig. 1. The 21 closed source applications are a subset of applications previously used works aiming at quantifying energy consumption, see for example [22]. The four open source applications have been selected to belong to different domain but being somehow related to the closed source applications, thus for example, we selected Tomdroid, an open source note-taking application conceptually similar in functionality to Droid notepad.

TABLE I
Distribution of 21 closed source applications across categories.

Category	Applications (%)
Books and reference	1 (4.76)
Business	1 (4.76)
Entertainment	3 (14.29)
Health and fitness	2 (9.52)
Lifestyle	1 (4.76)
Music and audio	3 (14.29)
News and magazines	2 (9.52)
Productivity	2 (9.52)
Tools	6 (28.58)

A detailed description of the 21 Android applications is shown in Table II. We choose these applications because they can be downloaded freely from the Android market, making our results fully reproducible.

In addition, we analyse four open source Android applications. A description of these open source applications is shown in Table III.

TABLE III
List of Android open source applications analysed.

Application name	Version	Description
Cool Switch	1.0.0	A custom view for Android with an awesome reveal animation.
F-Droid	0.83	An installable catalogue of FOSS (Free and Open Source Software) applications for the Android platform.
Ringdroid	2.4	An Android application for editing and creating your own ringtones, alarms, and notification sounds.
Tomdroid note	0.7.5	A desktop note-taking app using a wiki approach and a simple user interface.

The remainder of this section introduce our research questions, describe our measurement setup, and data analysis approaches.

(RQ_1) Can the proposed approach measure fine-grained changes in power consumption without noise or with a very low noise level?

Electronic devices used to measure power consumption produce noise. For example, thermal noises generated by the random thermal motion of charge carriers inside an electrical conductor are unavoidable at a non zero temperature. Other noises can come from the environment

TABLE II
LIST OF ANDROID CLOSED SOURCE APPLICATIONS ANALYSED.

Application name	Version	Description from Google Play
25000 Best Quotes	1.0.7	Best quotes from various authors and categories.
8500+ Drink Recipes	1.0.6	Over 8500 Drink & Cocktail recipes at the tap of your fingertips.
Android Antivirus	2.0.1	Protects your Android mobile phone or tablet against viruses, malware and spyware.
AnEq Equalizer Free	1.0.9	Improve the sound of your device with a 5-band equalizer.
Anti dog mosquito whistle	1.3	This app let you to scary some annoying barking dogs or neighbour cat.
Anti Mosquito Sonic Repellent	1.0.0	Repellent against mosquitoes.
Antivirus Security Free	4.1.4.204288	Protects you from harmful viruses, malware, spyware and text messages.
aTimer	1.3	A super readable timer. Big bold numbers for visibility.
AudioPlayer	1.2.0	A compact audio player for Android Smartphone/tablet.
Battery Info	1.6	Show battery information, include capacity, temperature and voltage.
Battery Info Always	1.2.0	Always display battery level on screen.
Better Notepad	0.0.5	An another simple, elegant, easy to use notes app.
Botanica	1.0.0	It helps you research the best plants for your climate and location, keeps track of plant growth.
Classical Music Radio Lite	1.0.3	Listen to classical music shows streaming live on internet radio.
Droid Notepad	1.11	Is a note taking app for Android. It allows you to take notes quickly any time.
Inspiring Quotes	1.2.0	A collection of inspirational quotes, drawn from experiences of people.
news|swipe	1.0.0	A modern, simple to use, simple to configure and fast offline RSS reader.
Simple Weather	1.1.3	A very basic weather app that allows you to get the most important details you need.
Sleep Sound Aid	20121007	Soothing sounds such as spring creeks, raging waterfalls, soft rainfalls, tropical birds.
Write Now Notepad	1.1.5	An easy and fast way to take notes without leaving your current app.
YouTube	1.0.5.4	See what the world is watching, from music videos to what's trending in gaming, news, and more.

in which measurements are performed; *e.g.,* vibrations, variations of temperatures, or variations of humidity can introduce noises in the measurements. It is therefore important to control for environmental factors when performing power measurements, and apply effective noise reduction techniques to reduce and possibly remove all the noise. This research question investigates the effectiveness of our noise reduction techniques.

(RQ_2) Do applications consume power at frequencies higher than 10 kHz?
Previous work have investigated the energy consumption of applications, using sampling frequencies lower or equal to 10 kHz. For example, Li *et al.* [13] examined the energy consumption of 405 mobile applications, using a sampling frequency of 10 kHz and reported that half of the energy consumption of most applications is spent on idle state. We claim that these measurements are biased by their low sampling frequencies, and such error in measurement may lead to inaccurate conclusions. More in details we claim that there are two components of the measure: the overall trend (*i.e.,* the average power value) and the dynamic part. Sampling at low frequency may not affect the continuous power consumption but will affect the dynamic part. In other words, fast methods or events may be lost or averaged out. In Section III we describe an approach that can measure energy consumption at frequencies higher than 10 kHz. In this research question, we examine the amount of power consumed by Android components and methods, at frequencies higher than 10 kHz and estimate the errors incurred by current approaches.

(RQ_3) Which error do we make when measuring the energy consumption of methods at a low sampling frequency?
Li *et al.* [13] measured the energy consumption of methods in Android applications using a sampling frequency of 10 kHz and reported that they can achieve an accuracy within 10% of the ground true. We revisit this claim in this research question by investigating the error bounds induced by sampling frequencies lower or equal to 10 kHz.

A. Measurement Setup

Device. The experiment was run on a BeagleBone Black[2] on which we installed Android 4.2.2 Jelly Bean[3].

Circuit. We use a DC power supply (Extech Instruments 382270) as input to the circuit of Fig. 1 and set the voltage to 10V. We connect the device (*i.e.,* BeagleBone Black) to the output of the regulator (7805C, $V_{reg} = 5$V). We measure the drop of voltage on the extremities of the resistance ($R_p = 1.8\Omega$, rated for $12W$) on the oscilloscope (Tektronix MSO3012). We connect the oscilloscope via USB to a laptop (3G RAM, Windows 7), to measure and process the data. The BeagleBone platform is an attractive platform for several reasons. The ARM processor is a processor also used in mobile device applications; it is reasonably cheap and it runs a fully flagged Android configuration including almost any application available on the Android market. As a plus, it has ports that can be easily driven to trigger data acquisition. On the negative side, it does not have a WI-FI or bluetooth chipset. This may be regarded as a major limitation; however, such limitation may be easily circumvent by using an usb dongle. Moreover, our goal is to provide a general framework easy to replicate and reproduce by other researchers at a reasonable cost much more than measuring power consumption tied to any specific chipset.

Measurements. We recorded the output of the oscilloscope on a laptop using LabView Signal Express[4] 2012. We measured the signal of the oscilloscope (V_{diff} of Fig. 1) in two different configurations. First, we measured the energy consumption for a fix duration (up to 60 seconds) after which we played back

[2]http://beagleboard.org/BLACK
[3]http://elinux.org/Beagleboard:Android
[4]http://goo.gl/nokSgE

the recording and exported the measurements in an ASCII file. This was done to collect data in a way similar or identical (but for the equipment and the sampling frequency) to some previous works such as [8], [22]. The goal was primarily to obtain data consistent in the acquisition methodology with previous works, sampled at high frequency and thus being able provide bounds on energy estimation error as a function of the sampling.

Following this first set of measurements, we selected four open source applications, and performed method level energy measurements for a set of methods sampled from these applications. We could not perform method level measurements on applications used in previous works because they are closed source and a reliable measurement of the energy at method level is impossible on a close source application.

In both sets of measurements (*i.e.,* at application and method levels), we set the oscilloscope in high-resolution mode: this makes the MSO3012 oversample the signal at the maximum frequency and average the values of each sampling interval to generate one data point at the set sampling rate. This measurement mode acts as a smoothing filter, and removes all high-frequency noise.

We set the data collection frequency sampling first at 500 kHz and then at 125 kHz, which is in any case one order of magnitude higher than what is commonly used in the literature [11], [22], *i.e.,* we take a measure every eight microseconds. We use the 500 kHz frequency to ascertain if indeed the frequency of 125 kHz was accurate enough to keep the energy error estimation below 1%. The 1% threshold is an arbitrary threshold deemed sufficient for the purposes of getting precise measures at a reasonable cost. In theory one could have used a frequency higher than 500 kHz, however, the MSO3012 model available to us has an internal buffer of limited size (about 20 seconds at 500 kHz), thus for practical reasons 500 kHz was our upper bound.

It is important to underline that when sampling signals with possibly small variation at frequencies as high as 500 kHz, one risks to acquire noise. Cabling, impedance and radio frequency noise sources must be carefully controlled. As this work is the first to use such high frequencies, we were forced to first quantify the amount of noise induced by our setup before proceeding with measurements.

We measured the noise in open loop (detaching the probes), with no load (board attached, but unplugged), and with load (board plugged in, but not powered up). Results show that the noise is in the range of 10mV (see Fig. 2), a value that is very close to the sensitivity of our instrument. We also measured the power consumption of the BeagleBone while Android is idling, *i.e.,* without any application running. Once we verified that noise was one order of magnitude smaller than signal, we moved to the real measurement phase and measure the power consumption associated to all applications described in Table II. We also measured the energy consumption of the methods sampled from our four studied open source applications.

B. Analysis Method

To answer our research questions $\mathbf{RQ}_1$ and $\mathbf{RQ}_2$, and prove that recent contributions inaccurately computed the dynamic part of the power, we applied frequency domain analysis (via MATLAB[5]) and time domain analysis via custom scripts developed to perform re-sampling and energy computation.
Let PF_c be the fastest sampling frequency used in previous studies from the literature. To the best of our knowledge, the highest value of PF_c is 10 kHz and thus $PF_c/2$ (*i.e.,* the Nyqvist frequency) is 5 kHz.
We sampled the energy consumption as follows: First, as stated in the Section IV-A, we sampled the voltage V_{diff} at 500 kHz and inspect the frequency spectra to ascertain the presence of high frequency components.

Next, we set the sampling frequency F_c at one order of magnitude higher than the fastest sampling frequency used in previous studies PF_c, *i.e.,* $F_c = 10PF_c$, we selected 125 kHz. We chose 125 kHz because it is a sub-multiple of 500 kHz, which allowed us to compare the signal acquired at 125 kHz with the down-sample (one sample every four) version of the signal acquired at 500 kHz. This was needed in order to verify the quality of the signal measurement chain, which is essential for the following steps. After this step, we sampled the signal at 500 kHz and computed the spectra. Using MATLAB we removed the spectral line at zero frequency, compared the energy of the signal sampled at 500 kHz with the energy in the bands $(0, 30]$, $(0, 2500]$, $(0, 5000]$, $(0, 62500]$, and compute the percentage error. These energy bands corresponds respectively to the sampling frequencies of 60 Hz, 5 kHz, 10 kHz and 125 kHz. We expect that the noise spectra is flat and very low in value; we also expect that the signal spectra has non zero frequencies above $PF_c/2$. The measure just considering only frequencies below $PF_c/2$ should have a low power with respect to the signal sampled at F_c. All the computations are performed using the Scientific Python and Numeric Python toolkits.

To answer $\mathbf{RQ}_3$, we used the Android debugging mechanism to collect method execution traces. These binary traces are very detailed and report various timing events such as the real, inclusive, or exclusive execution time of methods as well as the number of calls, and recursion calls to the methods in the execution thread, and the overall execution time (all time values are in microseconds).

To compute the error that one makes when measuring the energy of methods at 5 kHz or 10 kHz, we collected the energy used by each of the sampled methods listed in Table V, instrumenting properly the source code as we explained in Section III-C. Let $E_{f_c}(m_i)$ be the energy for method m_i, measured when sampling at frequency f_c, this measure is taken with the measure setup of Fig. 1 and, given the features of the measurement apparatus, it is accurate to the nanosecond time precision and 5mV voltage accuracy. We measured the error incurred by sampling at f_c versus 125 kHz using the following operation : $100\frac{E_{f_c}(m_i)-E_{125}(m_i)}{E_{125}(m_i)}$.

[5]http://www.mathworks.com/products/matlab/

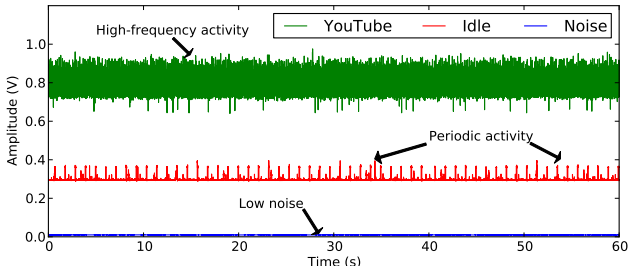

Fig. 2. Consumption amplitude.

Whenever pertinent we apply the Wilcoxon rank sum test [19] for example to compare error rates for different sampling frequencies, using a 95% confidence level (*i.e.,* p-value<0.05). For any comparison exhibiting a statistically significant difference, we further compute the Cliff's δ effect size [18] to quantify the importance of the difference because Cliff's δ is reported to be more robust and reliable than Cohen's d [5].

The Wilcoxon rank sum test is a non-parametric statistical test to assess whether two independent distributions have equally large values. Non-parametric statistical tests make no assumptions about the distributions of assessed variables. Cliff's δ is a non-parametric effect sizes measure (*i.e.,* it makes no assumptions of a particular distribution) which represents the degree of overlap between two sample distributions [18]. It ranges from -1 (if all selected values in the first group are larger than the second group) to +1 (if all selected values in the first group are smaller than the second group). It is zero when two sample distributions are identical [4].

V. Results

This section presents and discusses the results of our three research questions.

RQ1: Can the proposed approach measure fine-grained changes in power consumption without noise or with a very low noise level?

Fig. 2 shows the consumption amplitude for 60 seconds of the noise, Android idling and Android playing YouTube time behaviour. The noise is negligible; the idle and YouTube measurements show a higher dynamics. In particular, YouTube has an evident high frequency component. This is confirmed by the spectra shown in Fig. 3. A big amount of power is present above $PF_c/2$. The power in the grey area (*i.e.,* frequencies below $PF_c/2$) clearly does not account for all the power and thus all the consumed energy. We can also observe power peaks in the regions between 5 kHz and 20 kHz as well as a surge of power around 48 kHz.

> *The analysis in the time domain of the reconstructed signals (with all spectrum and the frequency below $PF_c/2$, but the zero line) shows that the noise just accounts for about 0.2% of the entire estimated energy.*

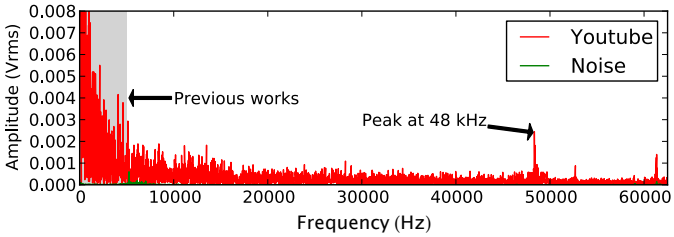

Fig. 3. Spectrum analysis.

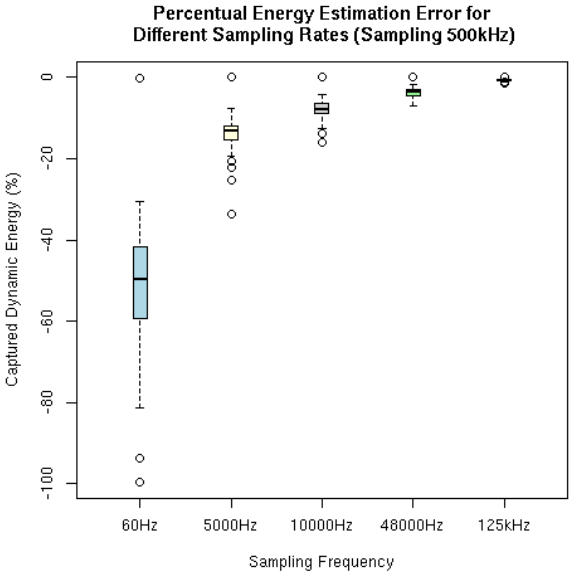

Fig. 4. Power measurement error rate at various sampling rates for a sample of 21 Android applications.

RQ2: Do applications consume power at frequencies higher than 10 kHz?

Fig. 4 reports the boxplot of the energy estimation errors for various sampling frequencies. As described above, we sampled at 500 kHz one execution of each application listed in Table II and then computed the percentage error of the dynamic component (*i.e.,* we removed the spectral line at zero frequency) of the signal. The reference value of Fig. 4 is the total energy between $(0, 250000]$. It is clear that when sampling at very low frequencies one can miss up to 50% (sampling at 60 Hz) of the signal dynamic. Table IV reports summary statistics of V_{diff} for the same energy traces used in the boxplot of Fig. 4. The table reports the min, max and median percentage errors plus the results of the Wilcoxon paired test between the error at 125 kHz (median -0.71%) and the error at a given sampling frequency. The table figures confirm the intuition that a sampling frequency of 125 kHz is sufficient with a worst case error below 1.5% and a median of 0.7%. We can also observe that sampling at 10 kHz is likely sufficient for many application as the median error is of about 8%, but with the risk of a maximum error close to 16%. This means that the dynamic part of the energy may be underestimated by any value between zero and 16% with a median of 8%. It is also clear that sampling frequencies below

10 kHz may severely underestimate energy components in the high frequency bands.

TABLE IV
PERCENTUAL ERROR SUMMARY STATISTICS FOR VARIOUS SAMPLING FREQUENCIES.

F_c (Hz)	Min	Max	Median	p-value	Cliff Delta
60	-0.07	-99.64	-49.69	<0.00001	-0.91 (Large)
5000	-0.001	-33.47	13.27	<0.00001	-0.91 (Large)
10000	-0.001	-16.08	-7.85	<0.00001	-0.91 (Large)
48000	-0.0002	-6.94	-3.64	<0.00001	-0.91 (Large)
125000	-0.00008	-1.32	-0.71		

It is important to understand the difference between total power and dynamic power. The errors of Fig. 4 and the bounds of Table IV pertain to the dynamic part of the signal when the continuous component is removed. If one consider the overall power (continuous plus dynamic) and it measures over a sizeable time interval the power peaks and valley average out and the error is negligible or non existent. For the same applications, we observed an error (for the entire energy trace and overall power) below 1%. Thus the developer has to clearly define what is his/her goal; the two goals: how much energy used my application and how much energy uses this method are not the same. For the first goal a low sampling frequency may be fine but for the second it may not. In fact, a low sampling frequency can make it very hard to assess the energy consumption of any given method. Consider, for a moment, the *8500+ Drink Recipes* application. The method *onCreate*, has an execution time (inclusive of called methods) of about 190 milliseconds, thus sampling at 125 kHz or 10 kHz does not make a big difference as enough data points will be collected. However, if we consider, for the same application, the method *DataBaseWrapper.setup*, this method execution lasts only 792 microseconds. At 10 kHz this method will be captured by at most seven samples. In essence, if a method execution requires a lot of CPU time such as in *onCreate*, the errors will generally averaged out; making the energy estimation error negligible or quite low. However, for a very short method, in the order of few milliseconds, the error may be much higher.

The analysis of the energy in different bands shows that a 10 kHz sampling may underestimate high frequency energy components of about 8% while sampling as 125 kHz just accounts for about 0.7% underestimation error.

RQ3: Which error do we make when measuring the energy consumption of methods at a low sampling frequency?

When measuring the energy consumption of methods at 5 kHz or 10 kHz (versus 125 kHz), one should expect a minimum error of about -5%, a maximum error of 53.13% with a median of 0.54%. Indeed, the distribution is concentrated around zero and a Wilcoxon paired test does not reject the null hypothesis that the data are drawn from the same population. Unfortunately, one has no way to know a-priori if the error will be positive, negative, or close to zero.

TABLE V
INSTRUMENTED METHODS FOR THE ANDROID OPEN SOURCE APPLICATIONS ANALYSED.

Application	Class	Method
Cool Switch	CoolSwitch	onDraw
	CoolSwitch	setAnimationProgress
	CoolSwitchRevealAnimation	startRevealAnimation
F-Droid	AppListAdapter	newView
	AppProvider	query
	AvailableAppsFragment	onCreateView
Ringdroid	CheapMP3	ReadFile
	RingdroidEditActivity	loadGui
	RingdroidSelectActivity	createCursor
Tomdroid Note	EditNote	onCreate
	EditNote	saveNote
	Tomdroid	newNote

If the energy is calculated proportionally to the inclusive execution time of the method over the total application's execution time, *i.e.*, $\widehat{E_{p,f_c}}(m_i) = E_{T,f_c} * T(m_i)/(calls(m_i) * T_{app})$ where E_{T,f_c} is the application total dynamic energy measured at f_c; $calls(m_i)$ is the number of calls of m_i; and $T(m_i)$, T_{app} are the method and application total CPU execution times respectively. The error estimation will be much higher. One should expect between a minimum of -84% and a maximum overestimate of 64% of error with a median value of -30%. Such a high variability show that indeed one must be extremely careful to assume a proportional relation between method execution time and energy used by the method.

There are many other factors to account for that a simple ratio is not able to capture. Previous approaches that make similar assumption (*e.g.,* [13], [22]) are likely to suffered from the sampling frequency error detailed above (*i.e.,* an error between -5% and +53%) plus the error due to the delay to align the method (debugger) execution trace with the energy signal (since they applied this heuristic on different sampling time partitions), plus an error related to the proportionality assumed between the method execution time and its energy.

Overall, we summarize our findings as:

The analysis of the method energy shows that if one samples at a frequency lower or equal to 10 kHz, the error can be as high as 53.13% (and not 10% as claimed by Li et al. [13]). The traditional way to compute method energy by summing power measurements taken during the time that the method was executed, (proportionally to the execution time of the method over total execution time), may underestimate as much as 84% of the the energy with a median of 30%.

VI. THREATS TO VALIDITY

There are several threats to validity possibly impacting this work. We are exploring an uncharted territory, a land between software and hardware. On one hand we have all the risk of any software maintenance or evolution study. On the other hand we also need to collect accurate voltage measures, which is not a typical software engineering task. In a way, this paper is closer to embedded system software change and evolution,

although it stands out for the goals, environment, languages and the kind of application under study.

Threats to *construct validity* concern relationship between theory and observation, and are related to inaccuracies in our measurements when collecting the trace and measuring the voltage V_{diff}. We do not have issues concerning the trace collection, as the traces have been collected with standard Android tools. We cannot however exclude a bias in the way Android traces, stores and represents data, or the Android time difference computation (*i.e.,* method start and end times). We did our best to limit this threat by using 1) standard tooling 2) by reusing as much as possible application used in previous studies; 3) by repeating the same measure several times (three or more) and 4) by making measures available to other researchers[6]. As far as the V_{diff} measure is concerned, we used state of the art tools (including the MATLAB and LabView Signal Express), and using a trigger mechanism driven by the board. We believe that overall, the construct validity threads are negligible but for a final detail. We, as previous works, cannot claim a direct causal relation between a method execution, the energy used by the method execution, and the overall energy consumed due to the given method execution. Consider a method switching on the WI-FI: it may take a certain amount of energy to switch the bits responsible to activate the WI-FI peripherals, but the cost of having the WI-FI activated is not assigned to the method. We believe this has to be modelled as different scenarios, and it is one of our future works.

Threats to *internal validity* concern factors, internal to our study, that could have influenced the results. When we study the energy distribution in different bands of frequency we used the highest possible sampling frequency (*i.e.,* 500 kHz) given our apparatus. We assumed no sizable portion of energy was in the bands higher than 250 kHz. Given what we observed this seems a reasonable assumption, but we cannot exclude that some components will be missing. When sampling at 125 kHz, we know we have (with respect to 500 kHz) an error of about 1% and we know that the input voltage of our setup is stable (about 1% error) and in the worst case we will have some frequency aliasing effect, but clearly any method having an execution time shorter than eight microseconds will be missed. There may be also a small time delay between the trigger activation, the data collection and method trace collection. We preferred to keep the setup as simple as possible, and we did not use, for example, a trigger hold. Overall, we believe that it is not an issue since we are not interested in estimating the energy consumption of the method responsible for activating/deactivating the trigger. Therefore, even if one or two samples are lost, they will be attributed to the trigger activation method. When comparing error rates for different sampling frequencies, we used non-parametric tests that do not require making assumptions about the data set distribution.

Threats to *conclusion validity* concern the relationship between experimentation and outcome. While part of the analyses of $\mathbf{RQ}_1$ and $\mathbf{RQ}_2$ are supported by appropriate statistical procedures, other findings of $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$ may require much more trace and method collection to apply appropriate statistical tests.

Threats to *external validity* concern the generalization of our findings. Admittedly, the study is limited to 21 closed source applications and only four open source applications. Although we are aware that further studies are needed to support our findings, our investigation was intentionally, relatively limited in size to allow us to present a complete methodology correlated with all the steps, data and findings. We share our data and scripts on the Web[6]. Further studies with different applications and hardware devices are required to verify our results and make our findings more generic.

[6]http://ser.soccerlab.polymtl.ca/ser-repos/public/tr-data/energy.tar.gz

VII. Conclusions

Estimating the energy consumption on a mobile device due to software components is not an easy task and to obtain accurate measures there is the need of an adequate setup including medium/high end measuring devices. The most critical element is the power sampling frequency: a too low sampling frequency may be misleading.

We have measured on an Android platform the energy consumption of several applications (21 close source and four open source) and 12 methods (out of the four open source applications) using different sampling frequencies between 60 Hz and 500 kHz. Our setup allows us to reliably measure energy consumption at frequencies up to 100 MHz. However very high sampling frequencies are problematic too; typically, measures will be affected by noise due to WI-FI, non shielded cables and so on. In this proof of concept setup we verified spectrum up to 100 MHz and to remove the electromagnetic noise we performed a downsampling at 125 kHz. In our setup we found that (1) above 100 kHz there is only noise (we measured only 0.2% of the power in that band); (2) noise reduction is a major issue and (3) there is non negligible energy in the frequency band between 5 kHz and 50 kHz. In a nutshell, a sampling frequency of 10 kHz, in our setup, can miss up to 50% of the used method dynamic energy due to missed high frequencies components. Overall, our results cast serious doubts on previous studies as we prove that a proper methodology is needed and that an adequate set up is a must.

We know that different applications and libraries use different energy [22] but we do not know what is the energy estimation error for different libraries and components. Therefore, future works will aim to verify, for different configurations and applications, the real energy consumption. We surmise that different applications have different power spectra; that applications power spectra shapes depend on used libraries and component; and that if the power spectra is not properly computed one may falsely believe to have optimized the application energy consumption while she has actually not.

Acknowledgment

The authors would like to thank Jérôme Collin and Guillaume Rivest for their valuable help. We also thank the

Electrical Engineering department of Polytechnique Montreal for sharing their resources.

REFERENCES

[1] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, *Power aware computing*. Kluwer Academic Publishers, 2002, ch. The case for power management in web servers, pp. 261–289.

[2] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, June 2000, pp. 83–94.

[3] C. Chatfield, *The Analysis of Time Series: An Introduction*, 6th ed. Chapman and Hall/CRC, 2003.

[4] N. Cliff, "Dominance Statistics: Ordinal Analyses To Answer Ordinal Questions," *Psychological Bulletin*, vol. 114, no. 3, pp. 494–509, November 1993.

[5] J. Cohen, *Statistical Power Analysis For The Behavioral Sciences*, 2nd ed. Lawrence Erlbaum, January 1988.

[6] I. L. M. Gutiérrez, L. L. Pollock, and J. Clause, "Seeds: a software engineer's energy-optimization decision support framework," in *ICSE*, 2014, pp. 503–514.

[7] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating mobile application energy consumption using program analysis," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 92–101.

[8] A. Hindle, "Green mining: A methodology of relating software change to power consumption," in *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, June 2012, pp. 78–87.

[9] ——, "Green mining: A methodology of relating software change to power consumption," in *Proc. of the 9th Working Conf. on Mining Software Repositories (MSR)*, 2012, pp. 78–87.

[10] 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*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 12–21.

[11] D. Li and W. G. J. Halfond, "An investigation into energy-saving programming practices for android smartphone app development," in *Proceedings of the 3rd International Workshop on Green and Sustainable Software*, ser. GREENS 2014. New York, NY, USA: ACM, 2014, pp. 46–53.

[12] D. Li, S. Hao, J. Gui, and W. Halfond, "An empirical study of the energy consumption of android applications," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, Sept 2014, pp. 121–130.

[13] 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*, ser. ISSTA 2013. New York, NY, USA: ACM, 2013, pp. 78–89. [Online]. Available: http://doi.acm.org/10.1145/2483760.2483780

[14] 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*, ser. ISSTA 2014. New York, NY, USA: ACM, 2014, pp. 339–350. [Online]. Available: http://doi.acm.org/10.1145/2610384.2610414

[15] Monsoon Solutions Inc., https://www.msoon.com/LabEquipment/PowerMonitor/, last viewed: 20-Nov-2014.

[16] A. V. Oppenheim and R. W. Schafer, *Digital signal processing*. Prentice-Hall, 1975.

[17] G. Pinto, F. Castor, and Y. D. Liu, "Mining questions about software energy consumption," in *MSR*, 2014, pp. 22–31.

[18] 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, pp. 1–33.

[19] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures (fourth edition)*. Chapman & All, 2007.

[20] D. Singh, P. A. H. Peterson, P. L. Reiher, and W. J. Kaiser, "The Atom LEAP platform for energy-efficient embedded computing: Architecture, operation, and system implementation," http://lasr.cs.ucla.edu/leap/FrontPage?action=AttachFile&do=get&target=leapwhitepaper.pdf, last viewed: 20-Nov-2014.

[21] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 2, no. 4, pp. 437–445, Dec 1994.

[22] 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 *MSR*, 2014, pp. 2–11.