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
<first name>.<last name>@polymtl.ca

ABSTRACT

Energy consumption is a major concern when developing and evolving mobile applications and researchers are investigating ways to reduce energy consumption.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. 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.

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%.

Keywords

Software Energy Consumption, Performance, Android.

1. 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.

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.

What is the new idea?

Researchers have recently relied on platforms like AtomLEAP [10] or Monsoon power monitor [11] 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 dynamic behaviour exceeds 10 kHz and thus the accuracy of recent investigations in software engineering related to mobile energy consumption can be improved by increasing energy sampling frequency and more precisely identify application consumed energy. 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 inaccuracy 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.

Why is it new?

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. Considering this fact, the contributions of the idea proposed in this paper are the following:

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;

What are the most related papers?

There is a rich literature on energy consumption monitoring, especially in the area of embedded and mobile devices, as such devices only have limited battery power.

Hindle [3, 4] 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.* [11] mine energy-greedy API usage patterns in Android applications using a Monsoon power monitor capable of sampling power at 5 kHz frequency. Li *et al.* [5] also use a Monsoon monitor to investigate best energy-saving programming practices at 5 kHz sampling rate. Hao

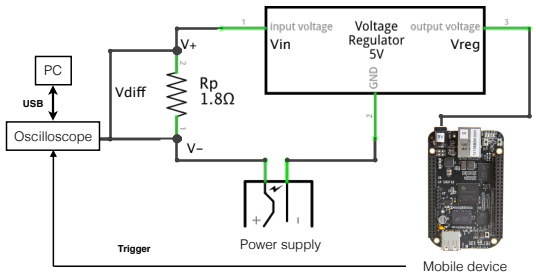

Figure 1: Measurement setup.

et al. [2] 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) [10] operating at 10 kHz. Using the same measurement device, Li *et al.* measured the energy consumption of source code lines [7] and studied the API level energy consumption patterns of 405 mobile applications [6]. All of these previous works suffer from one main limitation, *i.e.,* the frequency of energy measurement reaches 10 kHz at best.

2. 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 the interested reader may wish to refer to the classic signal processing books [1, 8].

2.1 Energy Measurement

Given a device with input voltage $V(t)$ and current $I(t)$, the input power at time t can be computed at the device power supply via Ohm's law *i.e.,* $P(t) = V(t) \cdot I(t)$. The energy consumed in the interval $[0, T]$, is then computed as the integral over time of the power $P(t)$. 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 [8], the absorbed energy is computed assuming a constant power between two measurements:

$$E_T = \int_0^T P(t)dt \simeq \sum_k P(k\Delta\tau)\Delta\tau \qquad (1)$$

2.2 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. 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 [8].

2.3 Accurate Energy Measurement

There are different ways to measure and compute $P(t)$ and thus E_T. One may resort on a special dedicated instrument such as the Monsoon power meter or use the processor energy estimation features or assume some kind of relation between execution time and the energy used. In this paper we propose to use a medium/high end digital oscilloscope or similar device; the oscilloscope is 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, (ii) a precise power regulator that will bring the voltage down to the device 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.

A key element of our measurement approach is the presence of a trigger signal, a signal raised by the mobile application/device activating/stopping 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. The application under study is instrumented (decompiled and re-packaged if needed) with specific, device dependent, trigger code used to rise and lower the trigger signal before(after) the code region (*e.g.,* method) to measure. Obviously, on a standard Android phone one need to add an interface between the USB port and an external GPIO or to hack the kernel and, for example, remap the volume button into a generic GPIO to use it like a trigger.

3. 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 Android applications.

Our experiments have the specific objective to examine the amount of power consumed by Android components at frequencies higher than 10 kHz and estimate the errors incurred by current approaches. The *context* of our study consists of 21 closed source Android applications presented in Table 1, four open source applications shown in Table 2) and a hardware setup described in Fig. 1.

Table 1: Android closed source applications used.

Application name	Version
25000 Best Quotes	1.0.7
8500+ Drink Recipes	1.0.6
Android Antivirus	2.0.1
AnEq Equalizer Free	1.0.9
Anti dog mosquito whistle	1.3
Anti Mosquito Sonic Repellent	1.0.0
Antivirus Security Free	4.1.4.204288
aTimer	1.3
AudioPlayer	1.2.0
Battery Info	1.6
Battery Info Always	1.2.0
Better Notepad	0.0.5
Botanica	1.0.0
Classical Music Radio Lite	1.0.3
Droid Notepad	1.11
Inspiring Quotes	1.2.0
news|swipe	1.0.0
Simple Weather	1.1.3
Sleep Sound Aid	20121007
Write Now Notepad	1.1.5
YouTube	1.0.5.4

Table 2: Android open source applications used.

Application name	Version
Cool Switch	1.0.0
F-Droid	0.83
Ringdroid	2.4
Tomdroid note	0.7.5

The 21 closed source applications are a subset of applications previously used works (*e.g.,* [11]) aiming at quantifying energy consumption.

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

RQ: 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. We claim that these measurements, probably, are biased by their low sampling frequencies, and such error in measurement may lead to inaccurate conclusions. 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 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.

3.1 Measurement Setup

Device. The experiment was run on a BeagleBone Black[1] on which we installed Android 4.2.2 Jelly Bean[2]. This platform is attractive 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.

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 to the output of the regulator (7805C, $V_{reg} = 5$V) and 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.

Measurements. We measured the signal of the oscilloscope (V_{diff} of Fig. 1) but, before, we verified the noise of the measurement circuit, the power supply and the overall measurement apparatus by acquiring two channels (one for V+ and one for V- versus the common ground – common mode rejection setup). We observed a very low and acceptable level of noise. Furthermore, we observed that even a single channel measurement gave acceptable noise level. We therefore used one channel for the energy measurement and one channel for the trigger events. Finally, we sampled the energy consumption as follows: First, 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.,* $F_c = 10PF_c$.

[1] `http://beagleboard.org/BLACK`
[2] `http://elinux.org/Beagleboard:Android`

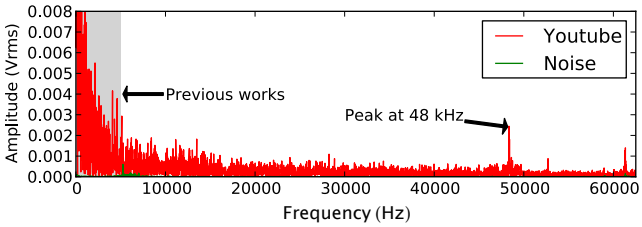

Figure 2: Spectrum analysis.

We chosen 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. After this step, we sampled the signal at 500 kHz and computed the spectra. 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.

4. RESULTS

Fig. 2 shows the consumption amplitude for 60 seconds of the noise, Android idling and Android playing YouTube. The noise is negligible (low green color peaks close to 10000 Hz). In particular, we observe that YouTube has an evident high frequency components above 10 kHz. The power in the grey area (*i.e.,* frequencies below 5 kHz) 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.

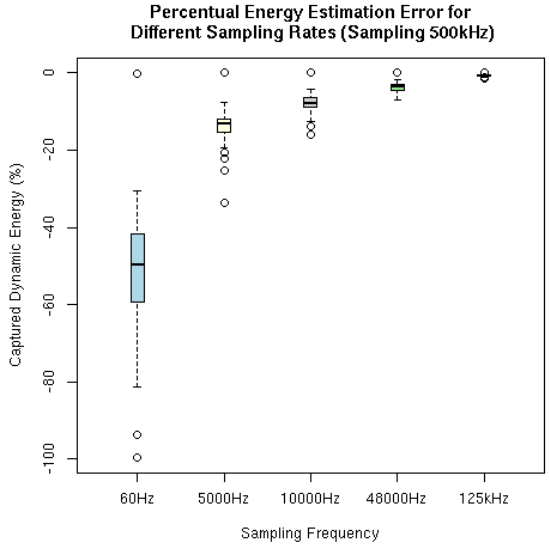

Figure 3: Power measurement error rate at various sampling rates for a sample of 21 Android applications.

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

Fig. 3 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 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. 3 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 3 reports summary statistics of V_{diff} for the same energy traces used in the boxplot of Fig. 3. The table reports the min, max and median percentage errors plus the results of the Wilcoxon paired test [9] (using a 95% confidence level) between the error at 125 kHz (median -0.71%) and the error at a given sampling frequency. For any comparison exhibiting a statistically significant difference, we further compute the Cliff's δ effect size [9]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 3: 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		

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.

5. CONCLUSIONS

We have measured on an Android platform the energy consumption of several applications (21 close source and four open source) using different sampling frequencies between 60 Hz and 500 kHz. In our setup and for the applications used in the study we found that (1) above 100 kHz there is no real energy (we measured only 0.2% of the total power in that band); (2) 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.
It is worth underlying two points. First and foremost it is unclear the origin of the high frequency energy. As suggested by [6] perhaps tight loops can be the major source or perhaps it is due to very short and fast methods. Second, it is again not obvious how this high frequency energy has to be attributed to different code area. We believe it is essential to replicate the measurement presented in [6] to fully assess the impact of our findings on a larger code base and precisely ascertain where the high frequency energy originates.

Acknowledgment

The authors would like to thank Jérôme Collin and Guillaume Rivest for their valuable help as well as the Electrical Engineering department of Polytechnique Montreal for sharing their resources.

6. REFERENCES

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

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

[3] Hindle, A. 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.

[4] Hindle, A., Wilson, A., Rasmussen, K., Barlow, E. J., Campbell, J. C., and Romansky, S. Greenminer: A hardware based mining software repositories software energy consumption framework. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (New York, NY, USA, 2014), MSR 2014, ACM, pp. 12–21.

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

[6] Li, D., Hao, S., Gui, J., and Halfond, W. 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.

[7] Li, D., Hao, S., Halfond, W. G. J., and Govindan, R. Calculating source line level energy information for android applications. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis* (New York, NY, USA, 2013), ISSTA 2013, ACM, pp. 78–89.

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

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

[10] Singh, D., Peterson, P. A. H., Reiher, P. L., and Kaiser, W. J. 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.

[11] Vásquez, M. L., Bavota, G., Bernal-Cárdenas, C., Oliveto, R., Penta, M. D., and Poshyvanyk, D. Mining energy-greedy api usage patterns in android apps: an empirical study. In *MSR* (2014), pp. 2–11.