On Measuring Smartphones' Software Energy Requirements

Hagen Höpfner[1], Maximilian Schirmer[1] and Christian Bunse[2]
[1]*Mobile Media Group, Bauhaus-Universität Weimar, Bauhausstraße 11, Weimar, Germany*
[2]*Software-Systeme, University of Applied Sciences Sciences Stralsund, Stralsund, Germany*

Keywords: Software Engineering, Data Processing, Energy Awareness.

Abstract: The continuous technological evolution of smartphones regarding their performance, networking facilities, and memory capacity, as well as various sensors, leads to a significant increase of a device's energy requirements. Hence, energy demand is one of the most limiting factors of battery-driven, mobile devices. Improving energy demand by software optimisation often relies on simulated energy demand data. This paper evaluates two approaches for actually measuring real energy data with the goal to build an efficient and cost-effective basis for future research. The main question underlying this paper is: Is the preciseness of energy data provided by a smartphone's operating system reliably close to the preciseness of data obtained via classic (i.e., hardware-based) measurement approaches. We defined a case study to evaluate this question. Our evaluation results show that software-based energy measures have an acceptable preciseness for comparative measures and are thus sufficient for research that warrants or requires total values.

1 INTRODUCTION AND MOTIVATION

Energy is one of the most limiting factors for the growth of information and communication technologies, especially when it comes to mobile devices such as smartphones. In most application scenarios, mobile devices do *not* have a permanent power supply but use batteries. Due to increases in hardware performance, display qualities, and the integration of additional hardware, like global positioning system (GPS) receiver, accelerometer, and other sensors into smartphones, energy requirements are constantly growing. However, software utilizes hardware and therefore, directly affects the energy requirements of the entire system. Energy-aware software development, energy-aware algorithms and energy-aware sensor substitution are only three examples for recently initiated research areas that try to reduce energy requirements by optimising the software rather than the hardware (Höpfner and Bunse, 2011). A common problem is the lack of guidelines and approaches to evaluate results. In this paper, we address the question on how to systematically measure energy requirements on smartphones. We examine two different approaches: First, a physical experiment setup for measuring energy requirements of dedicated hardware components. Second, a software-based approach using energy measurement facilities of the operating system. While it is assumed that the hardware approach provides precise data the second can easily be used with different devices without the need for manipulating the device.

We compare both strategies, examine differences regarding preciseness and reasons thereof, and address their use in the context of the following measurement scenarios: *Comparative measures* allow to compare different implementations, e. g. of algorithms, regarding energy requirements. They are used to specify that an algorithm requires less energy than another one. *Total demand measures* allow to quantify the amount of energy required in total to perform a certain task, e. g., to sort a list of integer values. *Quantitative measures* allow to quantify the amount of energy required per hardware component to perform a certain task; e. g., how much energy does a GPS receiver require per request.

The paper is structured as follows: Section 2 discusses related work. Section 3 introduces our hardware-based reference setup. Section 4 presents software-based techniques to measure energy. Section 5 discusses their appropriateness. Section 6 concludes the paper and shows future research options.

Höpfner H., Schirmer M. and Bunse C..
On Measuring Smartphones' Software Energy Requirements.
DOI: 10.5220/0004067601650171
In *Proceedings of the 7th International Conference on Software Paradigm Trends* (ICSOFT-2012), pages 165-171
ISBN: 978-989-8565-19-8

2 RELATED WORK

The research presented in this paper is rooted in the research fields of energy-aware computing in general, and on energy requirement ascertainment techniques in specific. Most energy-aware computing approaches either try to reduce energy requirements by substituting hardware resources (Bunse and Höpfner, 2008), or by balancing energy requirements and information quality (Sousa et al., 2008). The authors of (Bunse et al., 2011a) illustrate that even a simple substitution of the resources central processing unit (CPU) and memory helps to reduce the amount of energy required for sorting data. The reason is the fact that CPU usage needs comparatively less energy than memory storage (Marwedel, 2007). The authors of (Veijalainen et al., 2004) found out that file compression (more CPU) reduces the energy requirements of wireless data transmissions (less network traffic). According to (Kansal and Zhao, 2008), a comparable effect appears for hard-drive operations. Compression does not reduce the quality of processed information but reduces sizes and therefore energy needs. The authors of (Schirmer and Höpfner, 2011) presented an approach to reduce energy requirements of location determination by reducing the amount of GPS requests. Hence, they also reduced the accuracy in order save energy (i.e., graceful degradation). The authors of (Höpfner and Bunse, 2010) show that processing less precise data requires less energy, and also present an experimental setup for measuring the energy requirements of core and memory of a micro controller based system, running standard sorting algorithms. We adapted this setup (cf. Section 3). Nokia provides a tool called Nokia Energy Profiler[1] and an API[2] that enables developers to monitor power consumption, as well as battery voltage and current on Nokia S60 devices. To the best of our knowledge Nokia did not publish any information about data correctness. However, their approach falls into the software-based techniques discussed in Section 4 and was used, e. g., by the authors of (Kjærgaard et al., 2009). They examined energy-efficient position tracking techniques. In contrast to the Nokia tool, the authors of (Zhang et al., 2010) evaluated their software based energy measurement approach called PowerTutor using hardware-based reference measurements.

[1]http://www.developer.nokia.com/Resources/Tools_and_downloads/Other/Nokia_Energy_Profiler/

[2]http://www.developer.nokia.com/Resources/Tools_and_downloads/Other/Nokia_Energy_Profiler/External_APIs.xhtml

3 HARDWARE-BASED MEASURE

Based on (Höpfner and Bunse, 2010), energy requirements can be measured via examining voltage drops at sense resistors captured during the execution of a service or application. Energy can then be calculated, following Ohm's and Kirchhoff's law, by evaluating the integral of the curve defined by the data. Using resistors for measurement purposes requires the use of specifically designed sense resistors that have an error limit $g \leq 10^{-3}$. In the context of this study we used a precision measurement sensor that has a resistance tolerance of $0.005\,\%$ and a temperature coefficient $\pm 0.05 \times 10^{-6}\,\mathrm{K}^{-1}$.

In contrast to performance or execution time that can be measured at specific (local) points, energy is a "delocalized" property. The energy required by or for a service is the sum of the energy required by all involved components (CPU, memory, etc.). Exact measures require multiple measurement points resulting in massive data sizes given that each component provides a measurement opportunity. A typical solution regarding this problem is to examine the energy demanded by a component in isolation in order to either provide a fixed value (e. g., line losses) or by defining a function (e. g., correlation to load). Energy may then be measured at a central interface. Unfortunately, it is not trivial to insert a sense resistor between, e. g., a CPU and the power supply since modern CPUs have multiple power lines. To avoid such problems we decided to use a board that offers dedicated measurement points for most components. Modern CPUs using multiple cores embedded in one chip cause another problem. A parallel execution of processes would lead to errors if energy is measured at one core only. Our first solution was to configure the system in a way that only one core is used, but we are revising our approach to address this problem.

As a side note, the sketched path to energy measurement has a limiting factor. The quality of results directly depends on the sampling rate. The higher the sampling rate the better the calculation result becomes. Our experiments have shown that reliable results require a sample rate of at least $1\,\mu\mathrm{s}$. So, every second we do get 10^6 data points. To mitigate size problems data should therefore only be collected for the time needed for fulfilling a single request and *not* for the lifetime of a service.

We used our own measurement framework that already has been successfully used in the context of several experiments (Bunse et al., 2011b). The central element is the smartphone whose energy demand is measured in relation to the execution of software artefacts. We chose a Google Nexus One out of the

following reasons: (1) It is a developer phone that allows, due to an open boot loader, to install modified Android versions without the need for jailbreaking. (2) It makes use of a single-core Qualcomm processor; problems that may arise when measuring energy requirements during parallel execution are avoided. (3) It can easily be disassembled in order to get access to relevant hardware components. (4) It can be altered in order to attach measurement lines.

We attached sense resistors to the (direct) voltage supplies of core, power supplies, etc. A digital oscilloscope continuously measures the voltage drops at theses resistors. Data is sent to and stored/processed by a a small industrial personal computer that preprocesses the raw data and calculates energy values (in Joule) for a predefined time period.

Typically, Android systems run several processes, services, or threads in parallel. It is not unusual that after some hours of operations more than 70 threads or processes are active and thus, are increasing the system load. Unfortunately, to satisfy all requests, the system randomly switches between processes (based on process priorities). In order to exactly measure the energy demand that is related to a specific software artifact, switching might introduce measurement errors. During our studies we had to learn that, although we are able to identify process switching via triggering, we cannot guarantee for loss-free measurements. So, we stripped our custom Android ROM and switched off as many services and processes as possible.

Most Android apps are written in Java. The Dalvik VM executes the Java byte code by the line. Every app is associated to its own VM instance in a separate GUI thread and may spawn separate threads and processes if needed. The garbage collector that manages the memory is an additional error source regarding energy measures. Furthermore, the use of kernel functions (i.e., triggering) is only possible using JNI that increases complexity and implies limitations of the stripped Android version. So, we used the Android NDK, aimed for performance critical apps. It allowed us to execute software directly on the Linux kernel without using the VM.

4 SOFTWARE-BASED MEASURE

Software-based measurement of power consumption and energy demands is an interesting alternative to the previously discussed hardware-based approach. The advantage being that no additional hardware components are required. This allows measurements in the field and with a short preparation phase. Power and energy characteristics are measured through the device's internal power management software components. Technically, these components are used to calculate the remaining uptime of the device.

Most smartphone SDKs provide high-level access to interpreted power management values in the form of percentage of full charge, or remaining battery capacity in mAh. While this information is sufficient to react to emergency situations (nearly empty battery), it is not sufficient to make clear statements about the energy demands of software or software components. Regarding the source of the power and energy data that is gathered with a software-based measurement, two approaches can be distinguished:

In the **model-based approach**, a cost model regarding the power consumption and energy demands for the hardware of a mobile device is required. For each component, different operation states have to be respected (e. g., the Wi-Fi chipset may be in a low-power sleeping state, or at its highest power setting while transmitting at high speeds on a weak channel). Hence, a high level of granularity and accuracy is required for every feature in the model. The *PowerTutor* project (PowerTutor, 2011), e. g., used high-accuracy hardware-based measurements to determine a device's power and energy properties. Given the model, statements regarding the power consumption of a given software can be made by measuring the intensity of use of the involved hardware components (e. g., *How long was the display turned on?*) and their state (e. g., *Which brightness setting was it on?*). Integrating the cumulated power consumption of all components provides a measure for their energy demand.

Compared to hardware-based measures, this approach clearly cannot provide the same level of accuracy as it bases on constant values for the power consumption of individual components. The *PowerTutor* system can reach $99.2\,\%$ of accuracy for short measurement periods ($10\,\mathrm{s}$ intervals), compared to an actual hardware measurement (Zhang et al., 2010).

The applicability of the model-based approach is also limited to the devices included in the model. For unknown devices, no assumptions can be made as used hardware components and their characteristics vary greatly between different devices.

The **data-based approach** relies on live data. Some smartphones include an electrical current sensor, which provides in combination with up-to-date voltage data from the device's battery all required data to calculate electrical power (cf. Figure 1).

There are, however, two major drawbacks to this approach: *resolution* and *availability*. In our experience, the data provided by the smartphones is updated only at a very low frequency. Values tend to remain stable for a few seconds, even when there is actually a

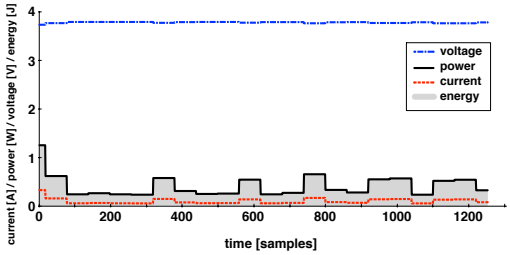

Figure 1: Data-based software measurement example.

high-frequency oscillation going on. Compared to the hardware-based approach, where sampling rates in the range of a few kHz can be achieved, it is possible to miss short-time fluctuations. Figure 1 illustrates the low *temporal resolution* with an exemplary measurement from a HTC Sensation. Samples were logged at a frequency of 1 Hz. The diagram shows the progression of electrical current, voltage, and derived from this data, electrical power. Furthermore, we included the electrical energy as the area below the electrical power curve. It is easy to see that peaks (in this case caused by GPS requests) can be detected, but changes are abrupt and it is hard to detect short-time fluctuations. One must also note that data is valid only, when the device is currently not charging. Otherwise, the current sensor reports the charging current that is not directly related to the power consumption of the device. Some devices distinguish charge and discharge current by signing the value, others provide separate values, or just one of them. This is directly related to the second drawback, *availability*. Electrical current sensors in smartphone devices are common, but not guaranteed to be included in every device, and while some devices include the sensor, they do not expose its values through the SDK. Despite these drawbacks, the software-based energy measurement provides an important opportunity to collect energy-related data during real-world use of mobile devices in the field. The approach requires only little preparation, generated data can easily be analysed and processed, and it is our theory that the low temporal resolution is still sufficient for comparative or total demand measures.

5 EVALUATION

As introduced before, the main question underlying this study is: *Is the precision and correctness of software-based measurement sufficient in order to make statements regarding a software's energy requirements?* We defined and performed a case-study in order to answer it. Our comparison-based evaluation compares software-based measurement results against a traditional, hardware-based treatment that acts as baseline. Data is collected for a set of algorithms of a varying runtime and space complexity. In detail, we defined simple algorithms for each combination of the four major complexity classes (i. e., linear, polynomial, logarithmic, exponential). Each algorithm was executed several times with an ascending input data size. Hence, we simulated the CPU load and memory requirement of a broad range of data processing applications. To minimise the impact of other energy demanders we switched off the display and all wireless communication interfaces and avoided any user interaction during the measurement phase of the experiments. Furthermore, we used internal memory only and removed external storage media.

Hypotheses and Research Questions.

We defined a couple of assumptions prior to the actual measurements, based on our experiences with software- and hardware-based energy measurement experiments:

1. Software-based measurements are sufficient for **comparative** and **total demand** measures.
2. Software-based measurements are not sufficient for **quantitative** measures.
3. Software-based measurements require longer **sampling periods** ($> 10\,\mathrm{s}$), compared to the hardware-based approach. Sampling period and accuracy of software-based measurements are correlated.

Discussion of Results.

In order to assess the quality of the software-based measurement results, we compared the average and total deviation between all hardware and software measurements, and the total amount of measured energy for both. The first two metrics serve as a measure for the accuracy of the software-based method.

Average Deviation (ε_{avg}) is the average of the percentaged difference between each measurement with varying input size n for a algorithm combination. In our experiment, this measure overvalued the short measurements as they are overrepresented in our range of n. Table 1 illustrates the distribution of average deviation across our experiment runs. The results indicate that a low average deviation is given for long-running tasks with an even distribution of the task times. It severely increases when short-running tasks are overvalued, like in the poly-poly case. In every experiment run with polynomial runtime complexity, we capped n at 750 to avoid extreme task times. As shown in Table 2, 17 of the total 20 experiment runs in

Table 1: Compressed comparison of results. n_{max} represents the maximum n that was used for this algorithm combination. t represents the total runtime for this combination.

Space complexity		Runtime complexity: log	lin	poly	exp
log					
	ε_{avg}	6.68%	7.27%	21.22%	243.5%
	ε_{total}	5.52%	5.28%	13.66%	62.42%
	n_{max}	2.5M	2.5M	750	15
	t	5.5 s	5.0 s	2.4 s	1.9 s
lin					
	ε_{avg}	4.23%	5.76%	19.86%	19.85%
	ε_{total}	4.46%	5.02%	12.58%	13.85%
	n_{max}	2.5M	2.5M	750	15
	t	149.2 s	148.9 s	2.7 s	2.0 s
poly					
	ε_{avg}	4.43%	5.02%	84.82%	42.36%
	ε_{total}	0.46%	0.41%	6.62%	25.62%
	n_{max}	750	750	750	15
	t	31.4 s	30.5 s	30.6 s	2.1 s
exp					
	ε_{avg}	4.17%	4.91%	74.59%	6.44%
	ε_{total}	0.08%	0.08%	0.01%	0.10%
	n_{max}	15	15	15	15
	t	163.7 s	164.9 s	165.1 s	165.5 s

the poly-poly condition had a task time below 1 s. Because of the low temporal resolution of the software-based approach, the peak energy levels during these short intervals were missed. However, the average deviation immediately dropped to 5 % and below with an increased task time for the 3 runs with larger n.

Table 2: Detailed data for the poly-poly case.

n	t[s]	$\mathbf{E_{SW}}$[J]	$\mathbf{E_{HW}}$[J]	ε[%]
1	0.04	0.0295	0.0998	238.12
2	0.12	0.0810	0.1531	89.07
3	0.08	0.0535	0.1325	147.44
4	0.13	0.0892	0.1674	87.59
5	0.10	0.0686	0.1418	106.57
6	0.06	0.0419	0.1120	167.55
		...		
15	0.17	0.1361	0.2077	52.58
50	0.32	0.2178	0.2940	35.03
100	0.56	0.3830	0.4592	19.88
250	2.19	1.5119	1.5896	5.14
500	7.98	5.2660	5.3407	1.42
750	17.9	12.4818	12.5578	0.61

Total Deviation (ε_{total}) is the percentaged difference between the cumulated amount of energy of the hardware measurement and the software measurement, for an algorithm combination. It helps to overcome the unbalanced expressiveness of the average deviation as it is more dependent on the total amount of energy that results from the measurements. Interpreting the results confirms our assumption that the extreme deviations during the experiment runs with short task times overshadowed the rather low deviations of the longer running tasks with an increased n. In the poly-poly case, the difference between average (84.82 %) and total (6.62 %) deviation is immense and reflects the shortcomings of the software-based measurement in the cases with short task times.

Total Demand The energy demand over all algorithm combinations and all n was 740.97 J for the software measure, and 746.98 J for the hardware measure. So, the difference was 0.81 % only.

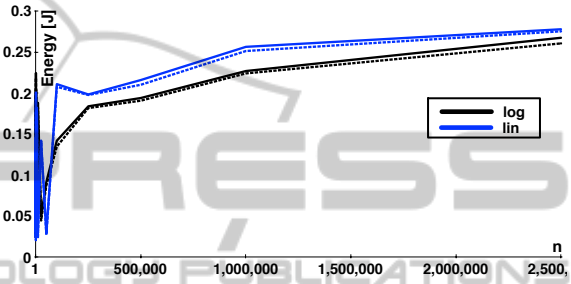

Figure 2: Logarithmic space complexity.

Figure 2 illustrates the energy demand for the algorithms with logarithmic space complexity and logarithmic and linear runtime complexity. Even at this small scale (< 0.3 J), the software results (dotted) are very close to the hardware measurements.

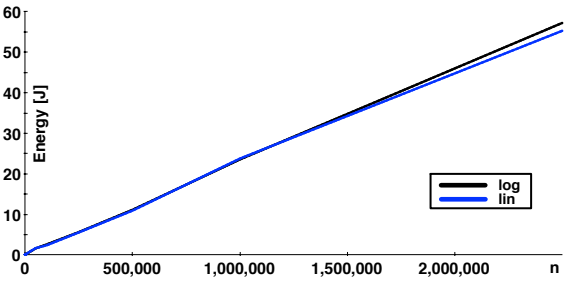

Figure 3: Linear space complexity.

Figure 3 similarly illustrates the energy demand for the algorithms with linear space complexity and logarithmic and linear runtime complexity $n_{max} = 2.5\text{M}$.

The Figures 4 finally illustrate all algorithm combinations for linear space complexity with n_{max} capped at 15. It highlights the small differences especially for experiment runs with low n.

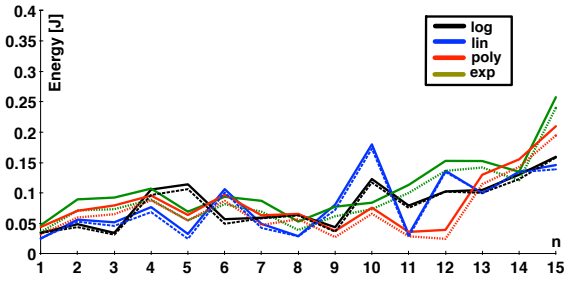

Figure 4: Linear space complexity.

6 CONCLUSIONS AND OUTLOOK

In this paper, we presented a case study to systematically examine and compare two approaches for measuring smartphones' software energy requirements. First, we created a hardware-based reference setup. Second, we introduced a software-based approach that collects and computes energy data provided by built-in features of the Android OS. Both setups were used to measure the energy requirements of a representative set of algorithms.

The interpretation of the evaluation results strongly support our hypotheses. The energy values for the individual n in the algorithm combinations allow to reconstruct the typical behavior, as one would expect from these algorithms. This means, software measurements are indeed sufficient for *comparative measures* (e. g. to test the difference between an algorithm with linear space complexity and an algorithm with polynomial space complexity at $n = 15$). Software measurement results are also sufficient for *total demand measures*. In our experiment, the deviation between the total energy demand, as determined with the software method, and with the hardware method, was as low as $0.81\,\%$. When applied to a real-world evaluation with a typically longer experiment runtime, the software method provides valuable insights into the total energy demand. The nature of the used data-based approach clearly impedes *quantitative measures*. As we were only able to gather data about the system's overall power consumption and energy demand, it is not possible to make clear statements regarding the energy demand for individual components. This strongly supports our hypothesis that software-based measures are indeed not sufficient for quantitative measures. The deviation between the hardware and software measurements strongly *correlated with the analysed time frame*. Due to the low temporal resolution of the software measurement method, evaluations with a short runtime ($< 2\,\mathrm{s}$) are error-prone and resulted in a drastic deviation. In fact, we expected this threshold to be even greater.

During our study, we were able to support our hypotheses but, in turn, also identified several open issues that warrant further research. First of all, we have to validate the robustness of our approach regarding different hardware platforms. Here, we also need to support hardware measurements of multithreaded systems. Furthermore, it is interesting to take a deeper look into the characteristics of the energy requirements of algorithms. Our as well as the cited results support the impression that algorithms have an energy signature that would help to define an energy complexity classification similar to space and runtime complexity. Finally, we have to take a look onto systems that provide only a subset of the data used in this paper for software base energy calculations.

REFERENCES

Bunse, C. and Höpfner, H. (2008). Resource substitution with components — optimizing energy consumption. In *ICSOFT '08 Proc.*, pages 28–35. INSTICC.

Bunse, C., Höpfner, H., Roychoudhury, S., and Mansour, E. (2011a). Energy efficient data sorting using standard sorting algorithms. In *Software and Data Technologies*, pages 247–260. Springer.

Bunse, C., Klingert, S., and Schulze, T. (2011b). GreenSLAs for the Energy-efficient Management of Data Centres. In *E-Energy '11 Proc.*

Höpfner, H. and Bunse, C. (2010). Energy Aware Data Management on AVR Micro Controller Based Systems. *ACM SIGSOFT SEN*, 35(3).

Höpfner, H. and Bunse, C. (2011). Energy Awareness Needs a Rethinking in Software Development. In *ICSOFT '11 Proc.*, volume 2, pages 294–297. SciTePress.

Kansal, A. and Zhao, F. (2008). Fine-grained energy profiling for power-aware application design. *ACM SIGMETRICS PER*, 36(2):26–31.

Kjærgaard, M. B., Langdal, J., Godsk, T., and Toftkjær, T. (2009). EnTracked: energy-efficient robust position tracking for mobile devices. In *MobiSys '09 Proc.*, pages 221–234. ACM Press.

Marwedel, P. (2007). *Embedded System Design*. Springer.

PowerTutor (2011). A power monitor for android-based mobile platforms. http://ziyang.eecs.umich.edu/projects/powertutor/index.html.

Schirmer, M. and Höpfner, H. (2011). SenST: Approaches for Reducing the Energy Consumption of Smartphone-Based Context Recognition. In *CONTEXT '11 Proc.*, volume 6967 of *LNCS*, pages 250–263. Springer.

Sousa, J. P., Balan, R. K., Poladian, V., Garlan, D., and Satyanarayanan, M. (2008). User Guidance of

Resource-Adaptive Systems. In *ICSOFT '08 Proc.*, pages 36–44. INSTICC.

Veijalainen, J., Ojanen, E., Haq, M. A., Vahteala, V.-P., and Matsumoto, M. (2004). Energy Consumption Tradeoffs for Compressed Wireless Data at a Mobile Terminal. *IEICE ToC*, E87-B(5):1123–1130.

Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R. P., Mao, Z. M., and Yang, L. (2010). Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In *CODES/ISSS '10 Proc.*, pages 105–114. ACM.