Progress in IS

Rüdiger Schaldach
Karl-Heinz Simon
Jens Weismüller
Volker Wohlgemuth *Editors*

Advances and New Trends in Environmental Informatics

ICT for Sustainable Solutions

Springer

Progress in IS

"PROGRESS in IS" encompasses the various areas of Information Systems in theory and practice, presenting cutting-edge advances in the field. It is aimed especially at researchers, doctoral students, and advanced practitioners. The series features both research monographs that make substantial contributions to our state of knowledge and handbooks and other edited volumes, in which a team of experts is organized by one or more leading authorities to write individual chapters on various aspects of the topic. "PROGRESS in IS" is edited by a global team of leading IS experts. The editorial board expressly welcomes new members to this group. Individual volumes in this series are supported by a minimum of two members of the editorial board, and a code of conduct mandatory for all members of the board ensures the quality and cutting-edge nature of the titles published under this series.

More information about this series at http://www.springer.com/series/10440

Rüdiger Schaldach · Karl-Heinz Simon ·
Jens Weismüller · Volker Wohlgemuth
Editors

Advances and New Trends in Environmental Informatics

ICT for Sustainable Solutions

Springer

Editors
Rüdiger Schaldach
CESR
University of Kassel
Kassel, Germany

Karl-Heinz Simon
CESR
University of Kassel
Kassel, Germany

Jens Weismüller
Leibniz Supercomputing Centre
Bavarian Academy of Sciences
and Humanities
Munich, Germany

Volker Wohlgemuth
Department of Engineering - Technology
and Life
HTW Berlin - University of Applied
Sciences
Berlin, Germany

ISSN 2196-8705 ISSN 2196-8713 (electronic)
Progress in IS
ISBN 978-3-030-30861-2 ISBN 978-3-030-30862-9 (eBook)
https://doi.org/10.1007/978-3-030-30862-9

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

This book presents the main research results of the 33rd edition of the long-standing and established international and interdisciplinary conference series on environmental information and communication technologies (Envirolnfo 2019).

The conference was held from 23 to 26 September 2019 at the University of Kassel. It was organized by the Center for Environmental Systems Research (CESR), under the patronage of the Technical Committee on Environmental Informatics of the Gesellschaft für Informatik e.V. (German Informatics Society—GI).

Combining and shaping national and international activities in the field of applied informatics and environmental informatics, the EnviroInfo conference series aims at presenting and discussing the latest state-of-the-art development on information and communication technology (ICT) and environmental-related fields. A special focus of the conference was on potential contributions of ICT technologies and tool to achieve the sustainability goals (SDGs) of the United Nations in context of the Agenda 2030 and to support societal transformation processes.

Accordingly, the articles in this book not only present innovative approaches and ICT solutions related to a wide range of SDG-relevant topics such as sustainable mobility, human health and circular economy but also to other questions that are central for environmental informatics research, including advanced methods of environmental modelling and machine learning.

The editors would like to thank all the contributors to the conference and these conference proceedings. Special thanks also go to the members of the programme and organizing committees. In particular, we like to thank the organizers of the GI Informatik 2019 conference that took place as a parallel event, for their support in providing local logistics. Last but not least a warm thank you to our sponsors who supported the conference.

Kassel, Germany — Rüdiger Schaldach
Kassel, Germany — Karl-Heinz Simon
Garching, Germany — Jens Weismüller
Berlin, Germany — Volker Wohlgemuth
July 2019

Contents

Assessing the Sustainability of Software Products—A Method Comparison

Javier Mancebo, Achim Guldner, Eva Kern, Philipp Kesseler, Sandro Kreten, Felix Garcia, Coral Calero and Stefan Naumann

Abstract As part of Green IT, the field of green software engineering has seen a rise in interest over the past years. Several methods for assessing the energy efficiency of software were devised, which are partially based upon rather different approaches and partially come to similar conclusions. In this paper, we take an in-depth look at two methods for assessing the resource consumption that is induced by software. We describe the methods along a case study, where we measured five sorting algorithms and compared them in terms of similarities, differences and synergies. We show

J. Mancebo · F. Garcia · C. Calero
Institute of Technology and Information Systems, University of Castilla-La Mancha Ciudad Real, Ciudad Real, Spain
e-mail: javier.mancebo@uclm.es

F. Garcia
e-mail: felix.garcia@uclm.es

C. Calero
e-mail: coral.calero@uclm.es

A. Guldner (✉) · E. Kern · P. Kesseler · S. Kreten · S. Naumann
University of Applied Sciences Trier, Environmental Campus Birkenfeld, Birkenfeld, Germany
e-mail: a.guldner@umwelt-campus.de
URL: http://green-software-engineering.de/en

P. Kesseler
e-mail: s.kreten@umwelt-campus.de
URL: http://green-software-engineering.de/en

S. Kreten
e-mail: s.kreten@umwelt-campus.de
URL: http://green-software-engineering.de/en

S. Naumann
e-mail: s.naumann@umwelt-campus.de
URL: http://green-software-engineering.de/en

E. Kern
Leuphana University, Lueneburg, Germany
e-mail: s14b75@umwelt-campus.de; mail@nachhaltige-medien.de
URL: http://green-software-engineering.de/en

R. Schaldach et al. (eds.), *Advances and New Trends in Environmental Informatics*,
Progress in IS, https://doi.org/10.1007/978-3-030-30862-9_1

that even though the methods use different measurement approaches (intrusive vs. non-intrusive), the results are indeed comparable and combining the methods can improve the findings.

Keywords Green software · Sustainable software · Software energy consumption · Energy measurements

1 Introduction

In a study by Huawei Technologies on the total energy consumption of all consumers in 2017, it was predicted that the entire information and communications technology (ICT) will consume around 2,800 TWh of energy in 2025 at best. In the worst case, the consumption could be more than double. By 2018, the energy consumption of ICT had already risen to 1,895 TWh, or around 9% of total global energy consumption [3]. In order to counteract this development, it is reasonable to review the energy and resource consumption of a wide variety of software to provide ICT users and developers with recommendations regarding sustainable software applications.

There are different approaches regarding energy measurements, which differ in several aspects. Thus, in this paper, we describe and compare two methods on how to measure the resource consumption of software.

2 Related Work

It is possible to identify different tools and techniques for measuring or estimating software energy consumption. They can be classified into two approaches. *(i)* Software-based approaches, which are easy to adopt and use. However, they give only a vague and global estimation of the power consumption of different components [12]. In contrast, *(ii)* Hardware-based approaches are more difficult to implement, but yield more accurate results. This is because they use physical energy meters, connected directly to the hardware [13].

2.1 Software-Based Approaches

This type of method uses mathematical formulas to estimate the energy consumption of the component under test. One of the best-known tools for estimating consumption is Microsoft's Joulemeter.[1] It is a software tool for estimating the energy

[1] cf. https://www.microsoft.com/en-us/research/project/joulemeter-computational-energy-measurement-and-optimization/ [2019-04-25].

consumption of the hardware resources used to execute a software application on a given PC. Joulemeter uses mathematical models to estimate the consumption on the basis of the information obtained from resources such as CPU usage, monitor brightness, etc. [10, 12].

Another tool that works in a similar way, estimating the energy consumption of all CPU-intensive processes is Intel Power Gadget (IPG) 3.0 [16]. In [1], the authors propose a tool called TEEC (Tool to Estimate Energy Consumption), to estimate the power consumption of a given software at run time by taking into account CPU, memory, and hard disk power consumption. Green Tracker is a similar tool [2], which calculates the energy consumption of the software by estimating its CPU usage. Unlike the previously presented tools, Green Tracker requires an external device to register the energy consumption and CPU usage. Jalen [18] uses mathematical models to estimate the software consumption, dividing this amount by the hardware resources used. It was created to control the energy consumption of software applications, taking into account the granularity of the code.

In an earlier work that originated at the Environmental Campus Birkenfeld, powerstat[2] was used to estimate the energy consumption of a system and the results were compared to the results from a hardware-based approach [4].

2.2 Hardware-Based Approaches

Hardware-based approaches use a hardware device to measure the power consumption of a specific component or overall system. There are different hardware tools that provide accurate measurements of energy consumption, such as the WattsUp-Pro power meter.[3] In [21], the authors indicate that this device makes it possible to assess the overall consumption of a computer, providing only a measurement error rate of 1.5%. Another tool is PowerPack [9], which consists of several hardware components such as sensors or meters that allow direct measurement of power. It also includes software components that control the creation of power profiles and the synchronization of codes. GreenHPC [20] is a framework for measuring energy consumption on High Performance Computing (HPC) environments. It has three components: a sensor board which is responsible for current sensing; a data acquisition board which collects the sensor board data and the voltage from the power source; and a virtual instrument which is responsible for data processing, visualization, and distributed clock synchronization. Rasid et al. [19] used a Raspberry Pi in their measurement approach accessing the energy impact of algorithm execution using different programming languages (ARM assembly, C, Java). Within their Software Energy Footprint Lab (SEFLab) [7], Hankel et al. [11] present a hardware based approach which allows measuring the energy consumption of components of the motherboard. They refer to other hardware-based approaches for analyzing

[2] cf. https://github.com/ColinIanKing/powerstat [2019-04-19].

[3] cf. https://www.wattsupmeters.com/secure/index.php [2019-04-24].

the power consumption of software [11], e.g. the PowerScope architecture by [8], addressing the energy consumption of mobile applications.

3 Methods

In the following, we describe the two methods that we compared. Method A was developed at the Institute of Technology and Information Systems at the University of Castilla-La Mancha, and Method B was developed at the Institute for Software Systems at the University of Applied Sciences Trier, Environmental Campus Birkenfeld. These methods have been selected for comparison as they both follow a hardware-based approach. Therefore, Method A as well as Method B allows us to obtain accurate measurements of the energy consumed by a software when it is running. In addition to these similarities, there are also some differences between the two methods. The possibility of combining the results obtained by both measurement methods provides us with the best information on the energy consumption.

3.1 Method A

The **FEETINGS** [17] (Framework for Energy Efficiency Testing to Improve eNviromental Goals of the Software) is used to measure the energy consumed by a software product running on a PC, to collect consumption data, and its subsequent visualization and interpretation of the information. This framework is divided into two main components, as shown in Fig. 1.

- The **Energy Efficient Tester (EET)** is the device that has been developed to measure the energy consumption of a set of hardware components used by a software product during its execution. EET is composed of different sensors that support the measurement of three different hardware elements: processor, hard disk, and graphics card. It also includes two sensors that provide both the total power consumption of the PC and the power consumption of the monitor connected to the equipment on which the software being evaluated is running. Once the measurements are completed, they are stored in a removable memory, so that they can be used for analysis.
- The **Software Energy Assessment (SEA)** is responsible for processing the data, the analysis and generation of an appropriate visualization of the data collected by EET. This part is currently under development.

In order to carry out the measurements, it is necessary to configure the computer (Device Under Test, DUT) to which EET is connected. The DUT should only include the applications necessary for its operation, such as the operating system, so that there are no other applications running in the background that affect consumption

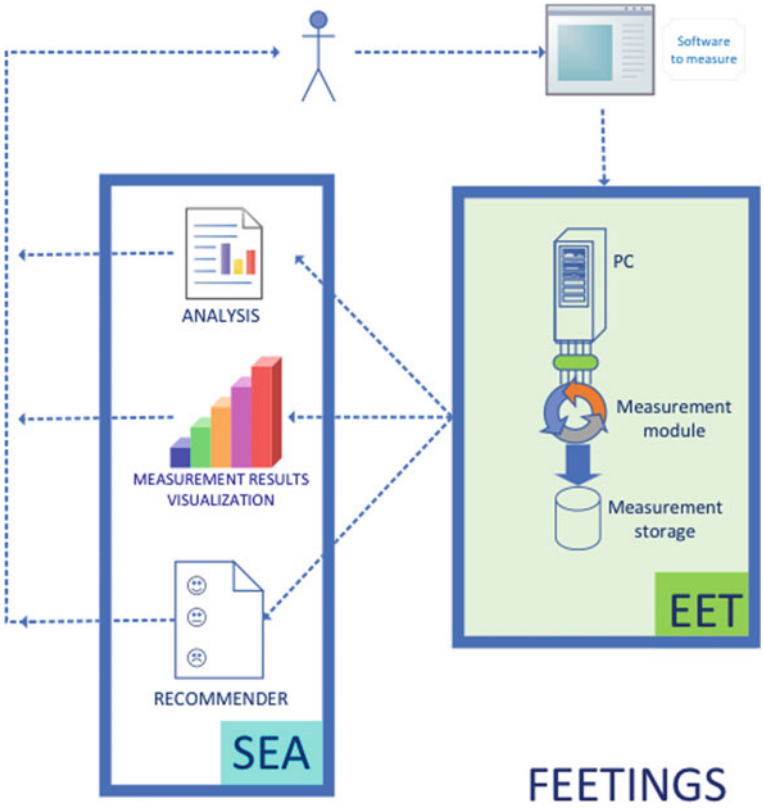

Fig. 1 FEETINGS overview

measurements. In addition, to ensure the consistency of the energy consumption measurements, the measurements will be repeated 30 times for each of the test cases that have been defined, so that the distribution in the sample will tend to be normal.

3.2 Method B

This assessment method was mainly developed in the course of two research projects:

– The measurement method was first developed by Dick et al. [5], in order to gain insight into the hardware- and energy efficiency of software products. It was devised, based upon ISO/IEC 14756, as described in [6].

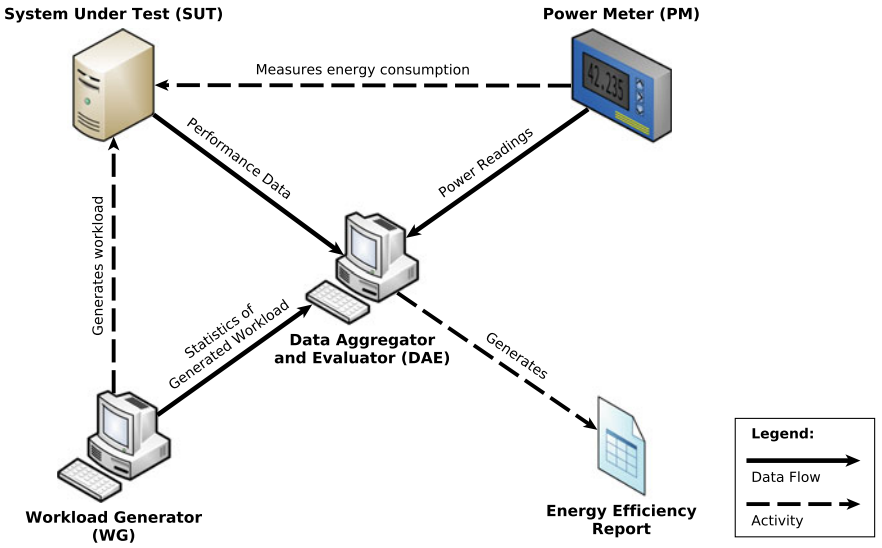

Fig. 2 Setup for measuring the energy consumption of software (cf. [5])

– The method was updated in [15], which focused on the development of a criteria catalog[4] to assess the sustainability of software on a broad level.

Figure 2 depicts the implemented measurement setup. It mainly consists of four parts:

– The System Under Test (SUT), which executes the software that is to be assessed and measures its own hardware usage,
– a Workload Generator (WG), which generates the load on the SUT by executing a scenario on it and logging the measurement timestamps,
– a Power Meter (PM), which measures the consumed energy of the SUT,[5] and
– a central Data Aggregator and Evaluator (DAE), which is used to analyze the results of the measurements.

To compare the energy and resource consumption of two software products (e.g. word processors, browsers, etc.), we first devise a usage scenario with a duration of approximately 10 min that produces the same result for each software product from the same product group. Depending on the software to be assessed, we use an automation software (e.g. WinAutomation,[6] a bash-script or benchmark program) to ensure the scenario is executed in a reproducible and automated manner. We currently also develop an Arduino-based mouse- and keyboard emulator to execute inputs directly through USB on the SUT.

[4]The resulting catalog can be accessed at http://green-software-engineering.de/en/kriterienkatalog [2019-04-16].

[5]Currently, we use a Janitza UMG 604.

[6]cf. https://www.winautomation.com/ [2019-04-16].

The SUT is then set up with all the required software installed for the execution of the software product. This includes the operating system, frameworks, packages, etc. Afterwards, we perform a baseline measurement without the software product. Then, the products are successively installed on the SUT and we measure the devised scenarios, at least 30 times to receive a normally distributed sample.

To analyze the results with the DAE, we implemented an open source consumption analysis calculator (OSCAR).[7] So far, it is only available in German, which is why, for the analysis of the case study presented in Sect. 4 of this paper, we created a separate analysis script in R Markdown for the purpose of this paper. It can be found, together with the measurement results for the algorithms, in the replication package for this paper.[8]

4 Case Study: Sorting Algorithms

To compare both systems, we measured the execution of a Java Program with Oracle Java 8 on a Windows 10 System. Table 1 shows the specifications of both systems.

The code was set up to run a loop for 30 times (test runs). Each loop sorted an item array with 50,000 random numbers from 1 to 1,000, using five sorting algorithms, namely bubble sort, cocktail sort, insertion sort, quicksort and mergesort. To increase the quality of the recorded data, the test runs should take between 5 and 10 min. Thus, the sorting algorithms were executed multiple times, so that the execution of every sorting algorithm takes approximately 2 min. Therefore,

Table 1 System specifications

Component	Research group 1 (Spain)	Research group 2 (Germany)
Processor	AMD Athlon 64 X2 Dual Core 5600+ 2,81 GHz	Intel Core 2 Duo E6750 2,66 GHz
Memory	4 × 1 GB DDR2	4 × 1 GB DDR2
Hard disk	Seagate barracuda 7200 500 Gb	320 GB WD 3200YS-01PBG0
Mainboard	Asus M2N-SLI Deluxe	Intel Desktop Board DG33BU
Graphics card	Nvidia XfX 8600 GTS	Nvidia GeForce 8600 GT
Power supply	350 W AopenZ350-08Fc	430 W Antec EarthWatts EA-430D
Operating system	Windows 10 Enterprise	Windows 10 Pro
Java version	Oracle Java 8u201	Oracle Java 8u201

[7] cf. https://www.oscar.umwelt-campus.de (German only) [2019-04-18], source code available at https://gitlab.umwelt-campus.de/y.becker/oscar-public [2019-04-18].

[8] cf. https://doi.org/10.5281/zenodo.3257517.

Table 2 Measurement results of the test runs

Result	Method A	Method B
Average power per second	104.565 W	109.610 W
Average scenario duration	779.347 s	603.846 s
Average energy consumption	22.631 Wh	18.386 Wh
Efficiency factor (sorted items per Joule)	18,612.72	22,910.65

– Bubble Sort was executed 18 times (900,000 items sorted),
– Cocktail Sort was executed 30 times (1,5 million items sorted),
– Insertion Sort was executed 280 times (14 million items sorted),
– Quicksort was executed 20,000 times (1 billion items sorted) and
– Mergesort was executed 10,000 times (500 million items sorted).

Between every sorting algorithm there was a break of 10 s and after every loop run there was a break of 60 s. The pauses between the execution of the algorithms was added to allow the SUT to return to its idle state, before starting the next task, in order to capture irregular patterns in the consumption, like CPU ramp-up and RAM allocation. While the loop was running, two log files were generated for further analysis with the power consumption data. In those log files, the starting and ending timestamps of every test run, and every sorting algorithm loop were recorded.

4.1 Results

This section presents the results obtained from consumption measurements for the methods described in Sect. 3. Table 2 shows an overview of the results measured with both methods.

The **efficiency factor** is calculated, based upon the metrics proposed in [14], where, in this case, we use the number of sorted items (1.5156×10^9) as the "useful work done":

$$\text{Energy efficiency} = \frac{\text{Useful work done}}{\text{Used energy}}$$

As can be seen from Table 1, the execution of the algorithms was carried out on computers with different hardware specifications for Methods A and B. To be able to compare both measurements, it is necessary to check the consumption of the computers when they are in idle mode, running only the operating system (consumption baseline). Table 3 shows the results of the baseline measurements.

The **mean adjusted baseline energy consumption** is calculated by dividing the average energy of the baseline by the average duration of the baseline and multiplying it by the average duration of the measurement:

Table 3 Measurement results of baseline

Result	Method A	Method B
Average power per second	73.395 W	78.785 W
Average scenario duration	302.910 s	599.999 s
Average energy consumption	6.176 Wh	13.133 Wh
Mean adjusted baseline energy consumption	15.888 Wh	13.217 Wh

Table 4 Mean measurement results for the individual actions from Method A

Action	Mean power (W)	Energy cosumed (Wh)	Efficiency factor ($\frac{\text{items}}{\text{Joule}}$)	Software induced energy consumption (Wh)
Bubble sort	96.743	3.427	72.95	0.826
Cocktail sort	97.497	3.425	121.64	0.847
Insertion sort	98.679	5.903	658.76	1.512
Quick sort	97.692	2.374	116.987	0.612
Merge sort	114.075	3.299	42.106	1.176

Table 5 Mean measurement results for the individual actions from Method B

Action	Mean power (W)	Energy consumed (Wh)	Efficiency factor ($\frac{\text{items}}{\text{Joule}}$)	Software induced energy consumption (Wh)
Bubble sort	109.689	3.357	74.47	0.946
Cocktail sort	110.322	3.473	119.97	0.992
Insertion sort	116.709	4.554	853.95	1.484
Quick sort	108.469	2.419	114.834	0.661
Merge sort	111.860	3.404	40.802	1.005

$$\text{Mean adjusted baseline energy consumption} = \frac{\overline{E}_{\text{Baseline}}}{\bar{t}_{\text{Baseline}}} * \bar{t}_{\text{Testrun}}$$

Finally, the adjusted mean reference energy is subtracted from the mean energy of the scenario measurements (the execution of the sorting algorithms), resulting in the additional energy needed to operate the software. Hence, the **energy consumption induced** by the execution of the sorting algorithms in **Method A** is **6.742** Wh, and for **Method B** is **5.169** Wh.

In addition, both methods allow for subdividing the scenario to analyze specific parts. In this case, we divided the scenario according to the run time of the sorting algorithms to calculate their efficiency factor. Tables 4 and 5 show the results.

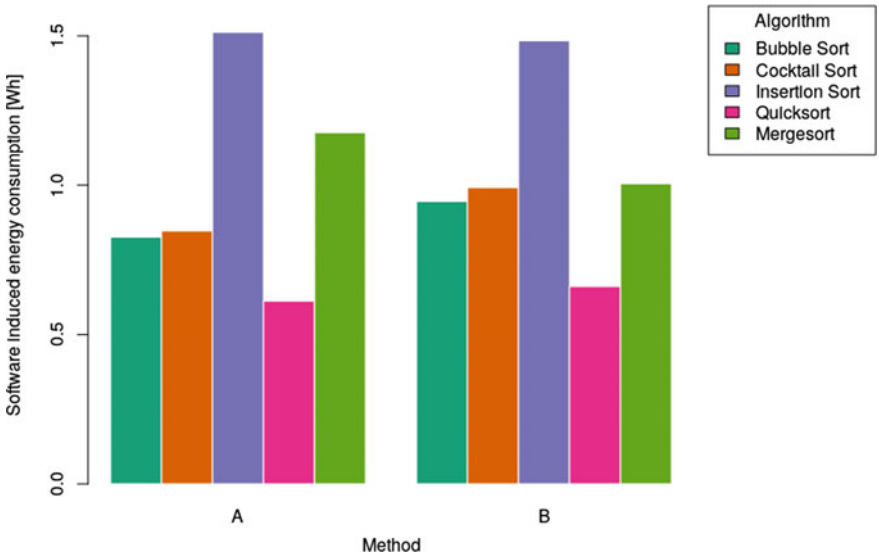

Fig. 3 Comparison of the software induced energy consumption for both methods

Because of the differences in the hardware used in the approaches, the results cannot be compared directly. However, there is a correlation between the software induced energy consumption of both methods for each algorithm, as can be seen in Fig. 3. Furthermore, the data can also be interpreted. As shown in Table 4, the mean consumption of insertion sort is 98.679 W, with a software induced consumption of 1.512 Wh. If we compare this with the corresponding values of insertion sort from Table 5, it can be seen, that the mean consumption there is higher with 116.709 W, whereas the software induced consumption is significantly lower than in Table 4 with 1.484 Wh. So there is a difference of $+18.03$ W between the systems A and B considering the mean power and a difference of -0.028 Wh between the software induced consumptions of A and B. If we assume a continuous and permanent use of both systems, there is a difference of 18.03 Wh between the two systems which can be compensated by the difference of the software-induced power consumption. If we now consider the multiple successive execution of insertion sort on both systems, it is noticeable that the difference in the mean power consumption between systems A and B would be consolidated after 644 test runs. For this reason, it can be assumed that even with less resource-efficient hardware, software induced power consumption can be reduced over time. This would require an amortized analysis for each system and algorithm.