The Impact of User Choice on Energy Consumption

Chenlei Zhang, Abram Hindle
Department of Computing Science
University of Alberta
Edmonton, Canada
{chenlei.zhang, abram.hindle}@ualberta.ca

Daniel M. German
Department of Computer Science
University of Victoria
Victoria, Canada
dmg@uvic.ca

***Abstract*—Hardware and software engineers are instrumental in developing energy efficient mobile systems. Unfortunately the last mile of energy efficiency comes from the choices and requirements of end-users. Imagine an end-user who has no power outlet access and must remain productive on the laptop's battery life. How does this user maximize their laptop's battery life, yet remain productive? What does the user have to give up to keep on working? We highlight the perils that users face and the ultimate responsibility users have for the battery life and energy consumption of their mobile devices; using multiple scenarios we show that executing a task can consume more or less energy depending on the requirements and software choices of users. We investigate multiple scenarios demonstrating that applications can consume energy differently for the same task thus illustrating the tradeoffs that end-users can make for the sake of energy consumption.**

***Index Terms*—energy consumption; energy efficiency; user choice**

I. Introduction

Classically the energy consumption and battery life of mobile devices has relied on the computer, electrical and software engineers who built and configured the system. In the realm of software, energy efficiency has been primarily the concern of operating systems (OS). There is little research in the area of energy efficient end-user applications, and it has primarily focused on recommendations on how to increase energy efficiency.

From the perspective of energy efficiency, software can be seen as a service. Therefore, we define energy efficiency of a software application as the amount of energy required per "unit of service" it provides. In general, the purpose of measuring the energy efficiency of a device or service is to use it as a comparison: if we can normalize two devices to provide the same amount of service (say, two refrigerators of the same size, or two laptops with comparable displays playing the same movie), their energy efficiency ratings can be used to identify the more energy efficient device or service.

Within the software realm, a "unit of service" is a malleable unit that must be defined from the point of view of the user. This can be: typing an average page, listening to 1 hr of music, checking and downloading email every 5 minutes, etc. A unit of service can be seen as a metric of quality-of-service, and in many cases, it does not make any difference if it can be executed any faster: the user will be satisfied as long as she can run the services she wants with the desired quality of service. For example, when listening to audio it does not make sense to play it faster; similarly, the user might not want to be interrupted more frequently than every five minutes by new email, and therefore, there is no need to check for it any faster.

The user might have different applications to satisfy her needs. In general, users will be confronted with many applications to choose from. For example, there are a very large number of text editors, music players, and email clients available. The decision on which to use lies on more subjective requirements such as usability, features, cost, etc.

One requirement that is rarely considered by users is energy efficiency. Given two software applications that can provide the same service, at the expected quality, is one more energy efficient than the other? This is particularly important if the goal is to maximize battery life. For example, assume you are on a transoceanic flight without power outlet access, and you want to play music in your laptop as you type a letter. You can choose between multiple applications to do it. How much does the choice of application impact the battery life? How much battery life would you lose if you decide to play the most efficient one versus not playing any music at all?

In this paper we demonstrate that users, by selecting the application to perform a task, play an important role in the energy consumption and battery life of their mobile devices (and by extension in the energy requirements of society).

The contributions of this paper are: we define the concept of energy efficiency of software applications; we propose and perform some user-oriented energy efficiency benchmarks. We show that applications running within the same domain can have wildly different power use profiles. Finally, we demonstrate that users can have a positive impact in reducing the energy consumption of the devices they use by choosing the energy efficient applications.

II. Previous Work

Energy consumption induced by software and peripherals is of great industrial interest to GreenIT and other companies [1]–[7]. Mobile manufacturers such as Apple, Microsoft and Intel [2], [3], [5], [6] have dedicated many resources in terms of developer education and tools.

Modeling is an important part of energy consumption since power regression tests are so difficult to make. Gurumurthi et al. [8] utilized simulation in their approach to model and estimate the power use of running systems. Within the mobile

space, Dong et al. [1] created an Android power monitor to show which running software is responsible for power use. Pathak et al. [9] tested applications like Angry Birds, with and without advertisements, and found that advertisements in free Android apps often led to more energy consumption. Many of these models are not relevant to user-oriented energy consumption due to heavy reliance on non-idle behaviour.

In terms of user-centric evaluation, Amsel et al. [10] have tested benchmarks on web browsers such as Chromium and Firefox to determine which one was more power efficient. This was an example of one benchmark, while we take a much broader and task oriented approach in this paper. Procaccianti et al. [11] profiled the energy consumption of different applications on a desktop computer to show the energy consumption impact of software. However, we sort applications into different categories and contrast the impact of user choice on energy consumption.

III. Energy efficiency of software

To this day, power benchmarking has been primarily concerned with the energy consumption of different components or the entire system. Nonetheless, the energy consumption of a computer also depends on the applications that it runs at that moment.

Johann et al. [12] have studied the energy efficiency of software by measuring energy consumption of sorting programs. We argue that energy efficiency benchmarks should be based on real applications and be user-centric. The user expects to complete certain amount of work within a given time; hence she determines the tasks that the software is expected to accomplish. In some cases, the software, such as an email client, is expected to run continuously. For others, the user chooses when the software runs and stops. To properly benchmark the energy consumption of an application is necessary to determine the typical amount of work expected from the application. This is highly dependent on the domain of the application. In some cases this unit of work might be measured by work completed per unit of time (throughput– such as running an email client continuously), in others, simply in terms of units of work completed (such as compressing a file). Any of the benchmarked applications should be capable of completing this amount of work. We call this the expected quality-of-service. We will refer to the energy required to complete the expected quality of service as the *energy efficiency of an application* (and measure it in watts per unit-of-work). The energy efficiency of an application will be the difference between what energy a computer will consume while running an application to complete that unit of work compared to not running the application (all other things equal).

Any computer consumes energy whether it is idle or not. In the same manner, running applications use energy whether they are being used or not. Benchmarks should also be created to measure the energy consumption of idle applications. We will name this the *ghost energy consumption* of an application. As its appliance's counterpart, the ghost energy consumption is particularly worrisome because it might be more expensive than stopping and restarting the application.

TABLE I
Applications Tested

Test	Application	Version
Text Editing	`gedit`	3.4.1
	LibreOffice Writer	3.5.7.2
	Google Docs	December 2012
Email Receiving	Mozilla Thunderbird	16.0.2
	Gmail on Mozilla Firefox	December 2012 on 16.0.2
Music Playing	`mpg123`	1.12.1
	Banshee	2.4.1
	Rhythmbox	2.96

IV. Methodology

In this section we present the methodology for measuring and comparing the energy consumption among applications with equivalent functionality for completing the same task. The general process is derived from our previous work on Green Mining [13] and is as follows:

1) Choose a software product.
2) Decide on the level of instrumentation.
3) Develop a scenario and simulate users to apply each application.
4) Build up the testbed to measure the energy consumption.
5) Run the tests and analyze results.

A. Choosing Software Products

In this study, we chose to develop tests for text editing applications, email clients and music players. The word processing applications include the text editor `gedit` on GNOME desktop, the word processor LibreOffice Writer (we will simply refer to it as LibreOffice from now on) and Google Docs. In terms of the email clients, Mozilla Thunderbird and Gmail were chosen. When testing Google Docs and Gmail, we used Mozilla Firefox as the web browser. Three music players were tested and they are `mpg123`, Banshee and Rhythmbox. The tested versions of these applications are shown in Table I.

B. Deciding on the Level of Instrumentation

The device we used to measure the energy consumption of the testing machine is the AC power monitor *Watts Up? Pro* [13]. This meter monitors real-time electricity usage with an accuracy of $\pm 3\%$ and collect a variety of data, including power use in watts, and transmit this result over a USB-serial connection.

C. Developing Use-Case Test Cases

In this paper, we sought to imitate real world users using these applications and developed three scenarios to test the energy consumption for each application in each scenario.

1) Idling Consumption: To properly measure the energy consumption of the application on the machine where the tests were run, we had to measure its idling energy consumption, i.e. the energy it consumes when no application is explicitly run. We left the Ubuntu Unity desktop running idle for a period of 5 minutes and in the meanwhile measured its energy consumption.

2) Editing Text: For text editing applications, the testing scenario is to emulate a user creating a new document, followed by typing text into it and finally saving it. We built a X11::GUITest UI driver to emulate the mouse actions and typing actions that we pre-recorded based on the first author typing in preamble in the GNU GPL (560 words) in 6 minutes.

More specifically, for `gedit` and LibreOffice, the procedure is 1) start the application, which opens a new document; 2) type the GNU GPL Preamble; 3) save the file; and, 4) close the application. For Google Docs, 1) start Firefox; 2) go to the Google Docs web page; 3) start a new document, 4) type the document; 5) save it; and 6) close Firefox.

3) Receiving Email: Our scenario to test email clients was meant to simulate a user idly receiving emails using either Thunderbird or Gmail in the foreground without user interaction. In order to implement this, a separate test computer sent plain text emails, 1 per minute, to a single Gmail account. Each email client was instructed to monitor the receiving email account.

To test Thunderbird, 1) we started Thunderbird; 2) and monitored Thunderbird for 10 minutes watching the 10 emails appear; and 3) we closed Thunderbird.

To test Gmail, 1) Firefox was started with the Gmail cookie already set; 2) the client would be instructed to type in the Gmail URL to go to the Gmail web page; 3) monitor Firefox and Gmail for 10 minutes as it received the 10 emails; and 4) close Firefox.

4) Playing Music: Our third scenario was to listen to music. For this we needed to test music players playing a three-minute long song.

The `mpg123` player was tested within a GNOME Terminal since it is a command-line based player. It was started with the song as a command-line parameter and would play the song when it started; it terminated once the song finished playing.

Banshee and Rhythmbox have GUIs, making their testing more complicated and requiring a GUI driver. They also maintain a database of the music of the users. For this reason, before we tested each application, we added the test song to their databases (it was the only song in them). Both Banshee and Rhythmbox were started by clicking on their respective icons on the Ubuntu 12.04's Unity panel. Once each application was opened we clicked the play icon which played the song, as it was the only song in the library. Once the song was finished playing, our GUI driver clicked the close icon and shutdown the music player.

5) Idling Applications: In order to understand the difference between ghost energy consumption of the idle applications and compare them to the energy consumption of the testing scenarios above, we tested the energy consumption of all the applications without handling any workload but staying in the background. To be specific, we started each of the text editing, email and music applications (except for `mpg123` because it is command line based and it is not intended to run idle), and immediately iconized them, without doing any work (except for the email clients, who continued to check for new email, but did not receive any).

D. Configuring the Testbed

We implemented our tests on a laptop, Lenovo ThinkPad X31, running 32-bit Ubuntu 12.04. To minimize the measurement noise during the tests, we removed its battery and turned off any services and automatic updates performed by the OS. We also disabled the screen saver and left the screen on at maximum brightness during the tests. Our initial setup had to balance the independence of the measurements versus the real-world relevance to the end-user. Headphones were plugged-in for the music tests.

We plugged the X31 into a *Watts Up? Pro* power meter and recorded its reading in the associated scenarios of each application by our application GreenLogger [13]. The *Watts Up? Pro* lets us measure the total system power utilization, without relying on a battery or recharging a battery between tests.

E. Running the Tests and Analyzing Results

We rebooted the testbed between different application tests. The first test after reboot was discarded in order to ensure uniformity of having a hot disk-cache in each subsequent test. We ran each test 40 times, ensuring that even in the presence of skew, we could statistically evaluate difference between the distributions of different applications using *t*-test. For each test, we calculated the mean consumption (in watts) during the test.

V. Results

In this section, we present the results and investigate the energy consumption of all the tested applications. Note that according to the t-test that all the comparisons in this section are statistically ($p < 10^{-10}$) significant even after correction for multiple hypotheses.

A. Idling on the Testbed and Applications

These tests could help us benchmark the testbed's idling energy consumption and applications' ghost energy consumption. The idling on the computer used 19.5 watts on average and we set it to be the system baseline, which is zero in Figure 1 (the distributions of each test are statistically different from any other, except for the idle and busy measurement of Gmail, which is a statistical tie). As shown in our results, the ghost energy consumption (while idle) of text editing applications, `gedit`, LibreOffice and Google Docs are 0.3, 0.5 and 1.1 watts correspondingly. The ghost energy consumption of music playing applications, Banshee and Rhythmbox are 0.9 and 1.0 watts.

The ghost energy consumption of most of the tested applications shows that it would cost a significant amount of power to just run the applications in the background doing nothing.

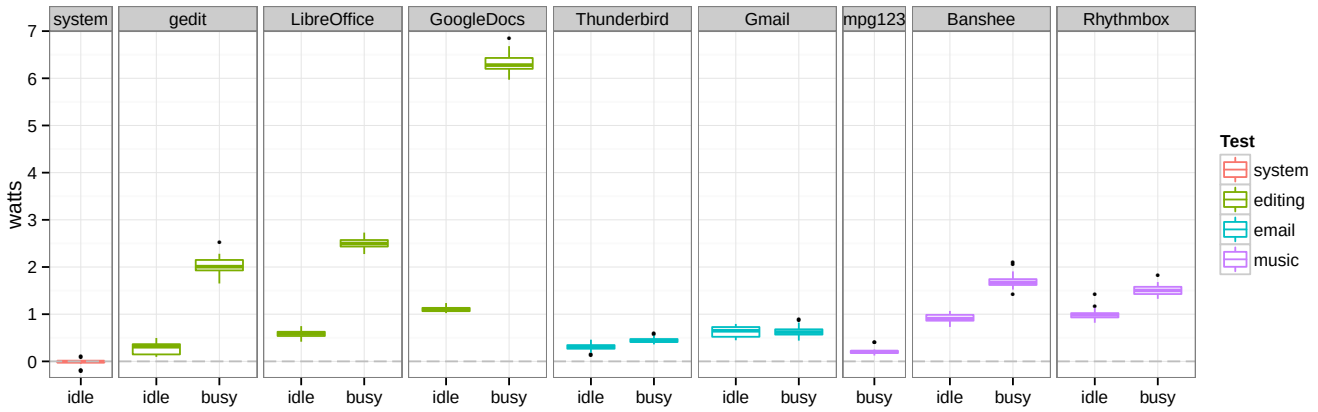

Fig. 1. Distributions of the mean watts consumed per test: idling on testbed, text editing, email receiving, and music playing tests. 40 tests total each.

B. Editing Text

In this part, we discuss the energy consumption of three applications, `gedit`, LibreOffice and Google Docs executing our text processing scenario.

`gedit` has limited functionality compared to LibreOffice and Google Docs which both focus more on word processing than text editing. Thus few layout features and formatting attributes are included in `gedit`. Since `gedit` is a lightweight text editor, it is unsurprising that `gedit` has the lowest energy consumption of the three. As shown in Figure 1, the average consumption of `gedit` is about 2.0 watts higher than the baseline.

LibreOffice provides more features than `gedit`. It includes some layout features and formatting attributes like centering and making bold titles. LibreOffice also has automatic spell check that can automatically highlight and correct misspelled words based on a dictionary. LibreOffice had 0.5 watts higher energy consumption than `gedit` did, as shown in Figure 1.

Google Docs is similar to LibreOffice in terms of its typesetting and layout features. Google Docs includes spell checking too. Unlike `gedit` and LibreOffice, Google Docs automatically synchronizes and saves text that is typed. This auto-saving feature causes Google Docs to synchronize with Google's servers frequently. Thus, as we can observe from Figure 1, the average consumption of Google Docs is 6.3, 4.3 and 3.8 watts higher than that of the system baseline, `gedit`, and LibreOffice respectively.

C. Receiving Email

As shown in Figure 1, the energy consumption of Thunderbird and Gmail are relatively similar: 0.4 and 0.6 watts above the baseline on average respectively.

Using different protocols to retrieve emails, Thunderbird and Gmail didn't behave the same in terms of energy consumption. Gmail had 0.2 watts higher energy usage than that of Thunderbird on average.

D. Playing Music

`mpg123` is a command-line music player, whereas Banshee and Rhythmbox both have GUIs. We used the default settings such as volume and EQ in these three application when playing the song. As shown in Figure 1, on average, `mpg123` has the lowest energy consumption, 0.2 watts above the baseline, compared to Banshee, 1.7 watts and Rhythmbox, 1.5 watts higher than the baseline.

For these tests, it is clear that GUIs and the music library management features affect the energy consumption.

The non-GUI music player is approximately five to six times more energy efficient than the ones with a GUI. It is also remarkable that ghost energy consumption of the GUI players is 0.9 and 1.0 watts.

Although the changes in power among applications are small, the impact of the applications on the battery-time could be dramatic. For example, using `gedit` would extend the battery-time by 20% compared with Google Docs if one used the testbed's 71 watt-hours battery.

VI. Discussion

A. Causes of Energy Consumption

In the tests we conducted, the main causes of energy consumption we observed from our results were:

- Synchronization with the cloud: Google Docs suffered greatly from constantly synchronizing the document in the cloud.
- Web-based applications are less efficient than their standalone counterparts likely due to network and browser overhead.
- Heavy startup: Banshee and Rhythmbox perform many tasks before they are ready to perform the tasks required.
- Continuous events are expensive: Spellcheckers hurt both Google Docs and LibreOffice as they increased the number of events per keystroke.
- UI updates are expensive: UI interfaces that are continuously updated consume a significant amount of energy.

B. Functionality Versus Consumption

If maximizing battery life is a major concern for users, they would choose their application based on its energy consumption. With the knowledge of different energy consumption

behaviours among applications in the same category, users are likely to get the application with the least energy consumption which also provides the expected quality of service. For example, start by typing the document in `gedit`, then spell-checking it in LibreOffice and finally uploading it to Google Docs (one could take advantage of the features of these products without incurring much of overhead of running them a long time).

It is not surprising that more functionality often leads towards an increase in energy consumption. Due to their simplicity, both `gedit` and `mpg123` are expected to perform better than their counterparts. In the future, applications might document the impact in energy consumption for each of their features, and include different options for different levels of energy consumption similar to the way OSes do it; e.g., a menu option for low power or normal energy consumption where some less important features—such as GUI animations—are disabled. Users who want to optimize energy consumption will be capable of making informed decisions in terms of what to run and when. Similarly, the interface is probably less important for the users than the ability to listen to music.

As shown in the previous section, many applications running in the background continue to consume energy. The ghost energy consumption of Banshee and Rhythmbox is even higher than playing music with `mpg123`. So application developers should consider reducing the number of events that occur during idle time, since for certain applications idle-time dominates (such as our email tests). Publicizing results of the benchmarking ghost energy consumption would pressure developers to focus on the energy efficiency of their idle applications. We expect that, as hardware and operating systems become more efficient, the focus will now turn towards the energy consumption of applications. Developers could use existing power estimating tools like PowerTOP [5] to start.

C. Software Application Energy Consumption Ratings

In general, users are unlikely to have the equipment, the expertise or the time to measure the energy consumption of applications. We propose the creation of *Software Application Energy Consumption Ratings* (SAECR). A SAECR has three main goals: 1) to define a framework and a methodology for consistent measuring of consumption of applications; 2) to create guidelines for the creations of benchmarks that represent typical user needs; and 3) to simplify the reporting of results in a manner that is easy-to-understand by typical users.

We believe that a SAECR can start with two benchmarks. The first, measuring the ghost energy consumption of applications. The second, relevant to applications that are expected to continuously perform the same task without user interaction (such as playing a movie, playing sound, a desktop widget, etc). In both cases the benchmark is straightforward (run the application and measure its consumption). These tests will allow users to compare the energy efficiency of similar applications. As mentioned before, this comparison will pressure developers to improve the energy efficiency of applications they develop and will allow users to determine what applications they should use (or stop using) when they are concerned with energy consumption (e.g. running on battery power). For SAECR to be effective, policies could be proposed to ensure SAECR compliance or competitiveness.

VII. Conclusion

Users have a choice of what application to use to complete a task. In this paper we have proposed a user-centric method to measure the energy consumption of applications based upon the scenarios that correspond to tasks that users are expected to complete. We demonstrate its effectiveness by defining three such scenarios, and an implementation of benchmarks to measure the consumption during each of them.

The results indicated that different applications can have dramatically different energy consumptions when performing the same task (e.g. a command line music player uses more than six times less energy than a GUI one). We also found that web-based applications tend to consume more energy than non-web based, and that idle applications can incur a significant amount of ghost energy consumption.

Unfortunately it is not trivial for users to know which applications are more energy efficient. We expect future work to be directed towards the creation of benchmarks and reporting mechanisms (similar to Energy Star [1]) that inform developers and users of the energy efficiency of their applications. This will likely generate pressure on the developers to improve the energy efficiency of the applications they develop.

Users, by making a conscious decision to use applications that are optimized towards energy consumption, can improve the battery life of their mobile devices, and perhaps more importantly, contribute towards helping the environment.

Acknowledgment

We thank NSERC for its financial support.

References

[1] M. Dong and L. Zhong, "Self-Constructive High-Rate Energy Modeling for Battery-Powered Mobile Systems," in *MobiSys '11*, 2011, pp. 335–348.

[2] Apple Inc., "iOS Application Programming Guide: Tuning for Performance and Responsiveness," http://ur1.ca/696vh, 2010.

[3] C. Gray, "Performance Considerations for Windows Phone 7," http://create.msdn.com/downloads/?id=636, 2010.

[4] A. Gupta, T. Zimmermann, C. Bird, N. Naggapan, T. Bhat, and S. Emran, "Detecting Energy Patterns in Software Development," Microsoft Research, Tech. Rep. MSR-TR-2011-106, 2011.

[5] Intel, "Saving Power with Linux on Intel Platforms," 2011. [Online]. Available: https://lesswatts.org/projects/powertop/

[6] Intel Corporation, "Developing Green Software," http://software.intel.com/en-us/articles/developing-green-software/, June 2011.

[7] S. Murugesan, "Harnessing Green IT: Principles and Practices," *IT Professional*, vol. 10, no. 1, pp. 24–33, 2008.

[8] S. Gurumurthi, A. Sivasubramaniam, M. Irwin, N. Vijaykrishnan, and M. Kandemir, "Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach," in *HPCA '02*, 2002, pp. 141 – 150.

[9] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the Energy Spent inside My App?: Fine Grained Energy Accounting on Smartphones with Eprof," in *EuroSys '12*, 2012, pp. 29–42.

[1] http://www.energystar.gov/

[10] N. Amsel and B. Tomlinson, "Green Tracker: A Tool for Estimating the Energy Consumption of Software," in *Proceedings, CHI EA*, 2010, pp. 3337–3342.
[11] G. Procaccianti, A. Vetro', L. Ardito, and M. Morisio, "Profiling Power Consumption on Desktop Computer Systems," in *ICT-GLOW'11*. Springer-Verlag, 2011, pp. 110–123.
[12] T. Johann, M. Dick, S. Naumann, and E. Kern, "How to Measure Energy-Efficiency of Software: Metrics and Measurement Results," in *GREENS*, 2012, pp. 51–54.
[13] A. Hindle, "Green Mining: A Methodology of Relating Software Change to Power Consumption," in *MSR*, 2012, pp. 78–87.

Chenlei Zhang will be joining Microsoft as a software development engineer. He has received his masters from University of Alberta. His research interests focus on mining software repositories and the impact of software change on energy consumption.

Abram Hindle is an assistant professor of computing science at the University of Alberta. Abram received his PhD from the University of Waterloo and his masters and bachelors from the University of Victoria. He works on problems relating to mining software repositories, improving software engineering-oriented information retrieval with contextual information, and the impact of software maintenance on software power use and software energy consumption. Abram is also prone to engage in musical performances where the audiences collaborate with the performance via technology.

Daniel German is professor of Computer Science at the University of Victoria. He completed his PhD at the University of Waterloo in 2000. His work spans the areas of mining software repositories, software evolution, open source, and intellectual property.