Energy Performance Testing of Smartphones: A First Look at Energy Bugs in Mobile Devices

by

Yasir Ali

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2014

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Smartphones have revolutionized the way people live their daily lives, the way they communicate with each other and the way they access information on-line. A decade ago, desktop computers and laptops were the primary source to use internet and access on-line information. But with all the technological advancements, smartphones and tablets have taken over. An important factor that aided to the popularity of smartphones is different applications available on smartphones. Whether a user wants to play games, watch videos, read books, access on-line information or check his/her email, there are applications for each and every one of them. These applications have greatly enhanced the user experience on smartphones.

According to an old saying, *everything comes at a price*. The same is the case with these smartphone applications. In addition to enhancing user experience and providing easy accessibility, they affect the smartphone battery consumption. They utilize the hardware resources and in turn consume the battery's energy. In comparison to the advancements in hardware and software industry, the development in battery technology is significantly slow. Even the battery energy density has little effect on the battery life with inefficient applications. Therefore there is a need: (a) for applications that efficiently utilize the smartphone battery, (b) to investigate the energy issues (energy bugs) in smartphones. For applications to be energy efficient; we need to have some testing methodologies so that the developers are aware of the energy consumption of their applications and can take appropriate measures while the applications are still in the development phase. Bugs are usually defined as an error in the system and energy bugs in smartphones are responsible for the unexpected and substantial battery drain. In order to research the energy bugs in smartphones, we need to have a comprehensive definition in context of software testing so that the developers can use it as a reference while testing their applications and improve the functionality of their applications.

With the above objectives in mind, in this thesis we have proposed and implemented a methodology to efficiently reduce the configuration parameters of smartphone applications that will help in reduction of test cases and will efficiently reduce the testing time. We also validated our methodology by measurements and experiments on four different smartphones. We have investigated the energy issues in smartphones and have defined energy bug. We also validated our definition with measurements and experiments.

Acknowledgements

First and foremost, I am grateful to Almighty Allah for His countless blessings and for giving me the knowledge and strength to accomplish my research work.

I would like to express my deepest and sincere gratitude to my supervisor Dr. Kshirasagar Naik for his support, guidance and encouragement throughout my research. I would also like to thank all of my research colleagues, former members of my research group and friends for their help and support.

I am also thankful to my committee members, Professor Pin Han Ho and Professor Andrew Morton, for their constructive feedback and comments on this thesis.

Last and by far not least, I am indebted to my parents and family for their unconditional love, continuous support and prayers.

Dedication

This thesis is dedicated to my parents.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Ever since the inception of the *telephone* by "Alexander Graham Bell", there has been a continuous development to improve the means of communication between people. And this development has led to new inventions, new technologies and enhanced communication and networking protocols. The technological world is changing rapidly and the communication services providers as well as the mobile device manufacturers are investing in research and development to enhance their products, services and to provide users with easily accessible state of the art technology.

1.1 Background

Mobile phones were introduced in the early 1970's [41]. They are the most popular communication devices ever in human history. Only second to them are the *'Smartphones'*, introduced by Research-In-Motion (now BlackBerry) in 2005. There has been an exponential increase in the number of mobile phone users in the past two decades and now that trend is shifting towards smartphones.

According to a report published by International Telecommunication Union (ITU), there are nearly 7 billion mobile subscriptions worldwide [26]. In the beginning of 2014, there were 14 countries with over 100 million mobile subscribers each. China alone has over 1.2 billion mobile subscribers [26]. Smartphones are also becoming a popular means to access the internet. According to the International Telecommunication Union (ITU), by the end of 2014 the mobile broadband subscription will reach 2.3 billion users [44]. The

reason that these smartphones are becoming so popular is that they not only fulfil the basic telephony needs of users but also act as a computing device equipped with latest technology. On the plus side, with recent developments in the chip-set technology these smartphones are becoming slim and small in size each year.

One other reason for this rapid and continual increase in smartphone users is thousands of applications (commonly referred as Apps) available on *App Stores*. As of June 2014, there are more than 1.2 million apps available in both *Apple's App Store* and *Google's Play Store* [27], [23]. These applications with their simplicity and easy to use features are available on the go and users do not have to always rely on their PC machines any more. These applications range from education to business, from travel to entertainment and from games to technology.

Smartphones with all new exciting features, latest technology and these amazing applications are favourite means of communication. Their functionalities and development however, do have some constraints like processing capabilities, battery power, data privacy and bandwidth; where *battery power* is the most important one.

1.2 Motivation

Following factors were the motivation behind this research:

1. *The growing popularity of smartphones.*

   The smartphone market is growing day-by-day and there is no parallel to its popularity. Recent statistics (Figure 1.1) have shown that users prefer to use smartphones over computers and laptops [24].

2. *The tremendous increase in smartphone applications.*

   Huge development of smartphone applications also aids the popularity of smartphones. Most of these applications use smartphone's limited resources and consume a large portion of limited battery energy. The number of downloads for smartphone applications is expected to grow from 10.9 billion in 2010 to 76.9 billion in 2014 [11] .

3. *"Energy Bugs" in smartphones.*

   Bug is defined as an error in the system. The term energy bug, as the name suggests, attributes that error with the energy of the system. In smartphones energy bugs are responsible for the loss of the battery's energy.

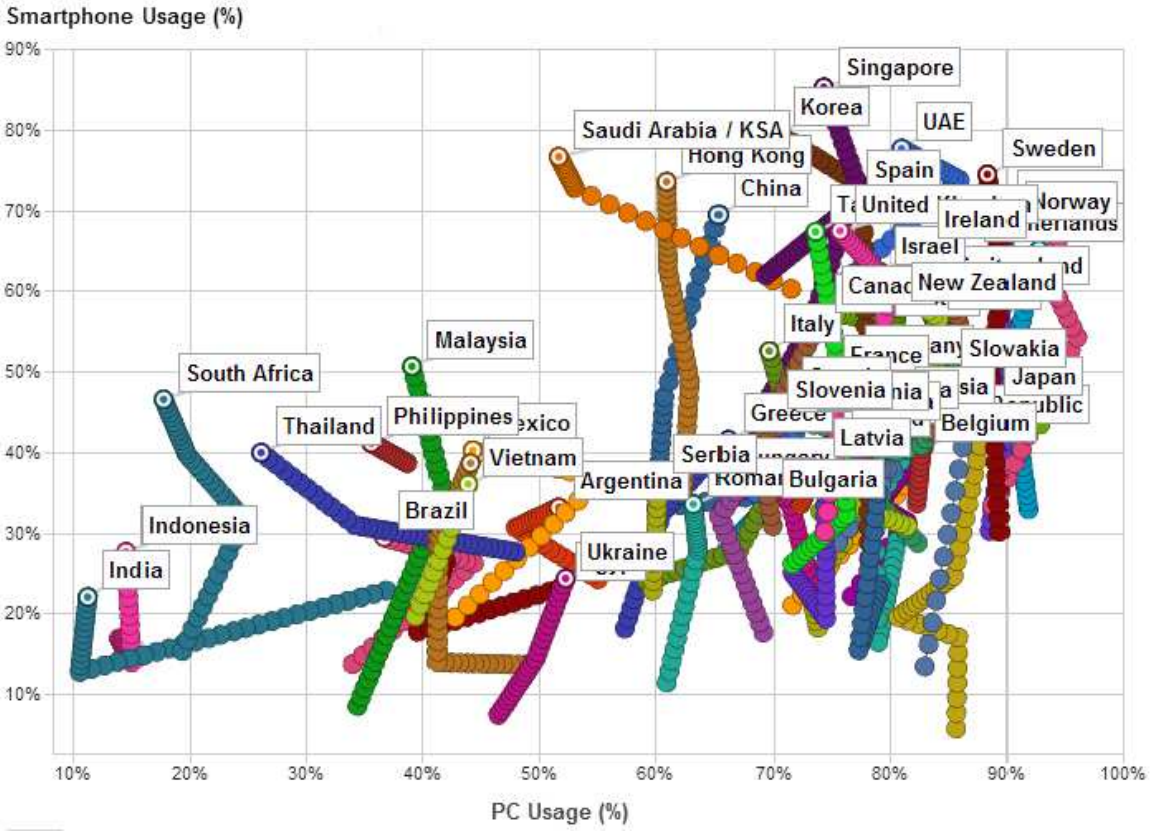


Figure 1.1: Shift of Internet usage trend from PC to Smartphone [24].

So with the above factors in mind, we first studied the power consumption by smartphone applications. After studying the impact of smartphone applications on battery, we focussed on methodologies for energy performance testing of smartphones. We proposed a methodology to select test configurations for energy cost evaluation of smartphone applications. We also defined energy bug and with the help of measurements and experiments showed that the energy issues in smartphones are crucial for battery performance.

1.3 Problem Statement

The number of smartphone applications is increasing every day. There are already more then million applications (apps) available in App Stores. The reason for the popularity of these apps is the fact that they provide users with easy access to their needs. Whether you need to check your email, watch videos, play games, read newspapers, access information on-line or access social media there are apps available with these features. As shown in figure 1.1, the trend in usage of smartphones to access internet is continuously increasing. In some countries almost 80% of the users access internet through their smartphones [24].

Many of these applications have parameters that use hardware resources and can be configured by the user. These hardware resources ultimately consume power from smartphone battery. The growth in battery technology is not keeping pace with the rapid developments in the other subsystems of a smartphone namely, central processing unit, memory, display, storage, and radio interfaces [30]. Therefore, there is a need to make these applications energy efficient and there is a need to develop testing methodologies to evaluate the energy efficiency of smartphones [40], [59].

Energy performance testing is different from functional testing (as discussed in section 3.1) and requires consideration of all the parameters and their configurations. With all the latest developments in 'app industry', there are an enormous number of application parameters and configurations in smartphones. As a result the number of test cases for energy performance testing is huge. Therefore we have proposed a methodology in this work that not only reduces the configuration parameters and test cases that a user has to consider to perform energy performance testing but also give accurate results.

Sometimes smartphone applications start to consume more power which eventually leads to drainage of the battery. The reason for this scenario is that there might be an *energy bug* in the system/application. In this case the fault is either at the developers end (the way the application was developed) or something might occur during execution that leads the application to consume more power.

There is a difference between traditional software bugs (mostly due to programming errors) and energy bugs. The energy bugs do not lead to an application crash or the system being in a halt state but instead let the system/application to continue its operation. The only problem is the excessive consumption of battery power [61].

The core issues being discussed in this thesis can be best summarized as:

- From energy performance testing perspective, how can we optimize the testing process while maintaining accurate results?
- From software testing perspective, how can we define the "energy bug" phenomenon in smartphones?

1.4 Solution Strategy

The following points outline the strategy to achieve the goals of this thesis. The details of these strategies are discussed in Chapter 3 and chapter 4.

1. *Categorizing Configuration Parameters of Smartphones.*

   Smartphone applications have numerous parameters that a user can configure. For energy performance testing of smartphones, it is necessary that these configurations should be reduced to a small set that would not only reduce the time for exhaustive testing but will also give almost the same results as if we would have considered all the configurations. To achieve this goal we proposed a methodology. We first selected the smartphone parameters and measured power consumed by each individual parameter and then applied our methodology to reduce the parameters. We used four state of the art smartphones and compared the results to verify our methodology [56].

2. *Defining "Energy Bugs" in Smartphones.*

   The issue of Energy Bug in smartphones is an important one and a few definitions can be found in literature but all of them are either generic or specific to one aspect of smartphone. In this work, we have tried to give a comprehensive definition to this phenomenon (in context of software testing) and have supported our claims with analysis, measurements and experiments.

1.5 Contributions

The work in this thesis makes the following contributions:

- generalising the energy performance testing framework in [59] and proposing a new methodology.
- evaluating the energy cost of some basic smartphone applications and showing how our methodology reduces the configuration parameters for energy performance testing.
- explaining and defining the phenomenon of energy bug.
- investigate the impact of smartphone applications on energy consumption and perform experiments and measurements to support the energy bug definition.

1.6 Organisation

The rest of the thesis is organized as follows. Chapter 2 discusses related research and literature for energy performance testing and energy bugs. Chapter 3 describes the energy performance testing framework, testing methodology used and the experiments carried out to validate our methodology. Chapter 4 contains the generalized definition of energy bug and the validation results. Finally, Chapter 5 concludes this work and present some recommendations for future work.

Chapter 2

Literature Review

In this chapter, we have discussed the energy issues in smartphones (Section 2.2). In order to have a good understanding of the research work presented in this thesis (Chapter 4), it is imperative to have knowledge of different bugs in smartphones. Therefore, we have presented reasons for the occurrence of energy bugs in section 2.3 and then a taxonomy of smartphone energy bugs in section 2.4. In section 2.5, we have presented a summary of related work done in the domain of smartphone bugs and energy bugs.

The goal of this chapter is to outline the concept of bugs in smartphones, provide a classification for these bugs and discuss existing research in this domain. All of this will lay a foundation to better understand the concept of 'Energy Bug' (discussed in Chapter 4).

2.1 Introduction

Smartphone market is estimated to be a $320 billion market by 2016 [17]. Smartphones are already a leader in the technological arena and gadget industry. The desktop computer sales around the world were outpaced by smartphone sales in 2011, making them the most prevalent computing platform [16]. Smartphone sales are projected to grow at nearly 30% compound annual growth rate over the next five years, exceeding 1.5 billion units per year by 2016 [17]. Despite their immense success in the market, smartphones are constrained to their battery life. Unfortunately the development in the battery technology is not at par with the developments in software industry and the hardware components in smartphones [65].

If we look at the battery energy density in smartphones, it has improved over the years but the rate of improvement and rate of increase in battery density is significantly slow. In a recent study [19], the authors took the watt hour ratings on each smartphone battery divided by the battery volume in square millimetres. This gives a normalized rating of battery efficiency as shown in figure 2.1

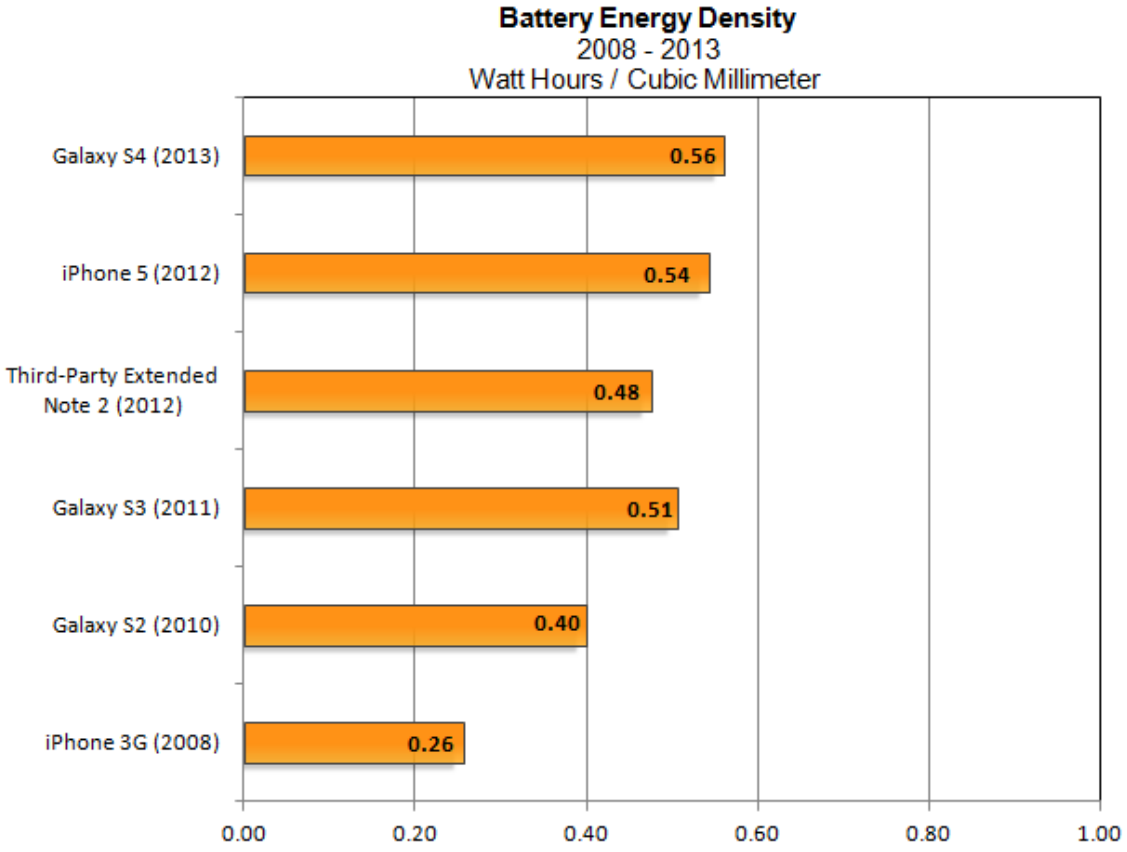


Figure 2.1: Battery Energy Density in Smartphones [19].

The rate of increase is substantially slow. As figure 2.1 shows that Galaxy S2 is 53% more dense then the iPhone 3G and the Galaxy S3 is 27% more dense then the Galaxy S2. The Galaxy S4 is approximately 10% better in terms of battery energy density then Galaxy S3. The iPhone 5 and Galaxy S4 batteries carry roughly the same amount of energy in their respective volumes [19].

Historically, the performance of 'lithium ion' batteries increases at about 10% a year [19]. As mentioned earlier, this improvement is not significant considering the technological advancements in the software and hardware industry. Therefore there is a need for energy efficient applications and energy performance testing methodologies so developers are aware of the energy consumption of their applications.

2.2 Energy Problem in Smartphones

Modern smartphones are equipped with state of the art technology but are constrained to their battery life. One possible reason for this fact is that modern smartphones are not only equipped with traditional components such as CPU, WiFi, NIC, radio, memory, screen and storage (found in desktop or laptop machines) but also other accessories like GPS, camera, accelerometer and other various sensors. And the power consumed by these individual I/O components is often comparable to, or higher than, the power consumed by the CPU in desktop or laptop machines [60].

Another important aspect of this energy problem is the smartphone applications. In the past few years the smartphones have transitioned from being a closed platform containing only pre-installed applications to open platforms hosting a variety of third party applications. This change has led to different energy drain problem in smartphones [50]. Free applications are also one of the reason for smartphone energy problem as they perform several different operations along with their intended function and increase battery consumption. According to [62], free apps like Free Chess and Angry Birds spend under 25-35% of their energy on gameplay, but over 65-75% on user tracking, uploading user information and downloading ads. These extra operations are a major reason for substantial battery consumption and battery depletion in smartphones.

If we analyse this issue in context of producer and consumer, producer being the smartphone battery and consumer being the actual hardware components used by different applications, we will find a mismatch between the two and this indicates that this issue should be dealt with more importance [60]. The trends in battery technology do not show a significant growth and development [65] and it all points to the important problem of energy being faced by smartphones today.

This energy issue is of such importance that almost 70% of phones returned to Motorola in 2011 were related to energy problems [61]. While another study [5] shows that the percentage of all technical support calls in context of faulty hardware are 14% for Android, 9% for Windows Phone 7 (WP7), 8% for Apple's iPhone and 3.7% for BlackBerry devices.

Therefore the smartphone applications have to be energy aware and energy efficient, there should be mechanisms and methodologies for developers to perform energy performance testing and there should be a definition of energy bugs in smartphones which can be used as a reference for software testing (all of these points are the goals and highlights of this thesis).

2.3 Bugs in Smartphones

According to [60], energy bugs in smartphones can arise due to a variety of different reasons:

- **Applications**:

  Wrong use of APIs and poorly managed and programmed applications can lead to battery drain.

- **Configurations**:

  Different configurations of applications can affect the power consumption of battery in different ways. And if not configured properly, can lead to massive battery drain.

- **Middle-ware**:

  Frameworks such as Android can be poorly programmed and can lead to severe battery drain.

- **Operating System**:

  Improper settings of operating system can drain considerable amount of battery.

- **Firmware**:

  Poorly programmed device drivers can lead to significant battery drain.

- **Hardware**:

  A faulty hardware like an old battery, an old and scratched SIM card or a faulty audio amplifier or a faulty LCD can drain substantial amount of energy.

- **Network**:

  Poor network conditions can lead to severe battery problems.

2.4 Energy Bugs: A Taxonomy

In the following we will classify the energy bugs (taxonomy) in terms of software issues, hardware issues or due to various other reasons.

2.4.1 Hardware Energy Bugs

- **Battery**:

  One of the many reasons for existence of energy bugs is the battery malfunction. If a battery is not working properly, it will not only increase the frustration of user but will also make him/her believe that there is a problem somewhere. In [61], authors studied various internet forums and recorded several issues reported by users of different devices on different platforms. The authors then classified those issues into different categories. According to their classification about 15.71% of the posts (a large number of users) reported that the energy depletion was due to faulty battery. Also if a battery is faulty, it never gets fully charged. It keeps losing the charge internally and results in heating the battery.

- **SIM Card**:

  The SIM cards in smartphones can also be the reason for battery drainage and energy bugs. In [61], authors found that about 0.43% of the posts on on-line forums mentioned SIM card being the reason for unexpected battery depletion. The reason could be that the SIM card is old and has scratches on it or that it is damaged and is shorting the pins. Also sometimes users themselves try to cut their SIM cards (regular SIM to micro SIM), that could also lead to damage and malfunction of SIM card.

- **SD Card**:

  Another important cause for battery drain is SD card in smartphones. If a SD card is faulty or corrupted it could either put the buggy apps in a looping state, where they try to access the hardware continuously [3], or in a hanging state, where they continue to drain battery [61].

- **Exterior Hardware Damage**:

  This could also be the reason for battery depletion. Hardware damage can act as a trigger for other parts of the phone (software) to drain battery. For example a 'home button' of phone could be damage and result in random unlocking of the phone, turning on the back light and consuming more power [61].

2.4.2 Software Energy Bugs

- **Operating System**:

  According to [61], OS updates either by user or Over-the-Air (OTA) represent a large number of complains for battery depletion. Among all the post analysed in [61], 19.54% posts were regarding energy bugs due to OS updates. IOS updates (4.0 - 4.3.3) resulted in substantial battery depletion among Apple devices including iPhone, iPads and iPods(4.74% of the posts). Android OS updates also caused battery depletion (2.39% of the posts).

  It is hard to identify the root cause for an energy bug. In case of an OS update, the energy bug could either be in the updated OS itself or in any of the accompanying applications or it might be due to the new OS configurations [61].

- **Configurations**:

  The OS configurations and even the configurations for different applications affect the power consumption and can be a reason for an energy bug. A change in OS configurations of device might lead to battery depletion [4], [66].

- **Applications and Frameworks**:

  Another important cause for energy bugs is the way applications are designed and the way different APIs and frameworks are used by the designers. In [61] energy bugs due to wrong use of APIs and frameworks have been classified as follow:

  - No Sleep Bug

    A no-sleep bug is a situation where an app acquires a wake lock for a component which wakes the component up, but does not release the lock even after the job is completed.

  - Loop Bug

    A loop bug happens where a part of an application enters a looping state performing periodic but unnecessary tasks, draining significant battery. The periodic task could be either unnecessary network polling, processing, or use of any other component in a loop.

  - Immortality Bug

    An immortality bug is a situation where a buggy application that drains battery, upon being explicitly killed by the user, respawns, enters the same buggy state, and continues to drain battery.

2.4.3 Energy Bugs due to External Conditions

- **Wireless Signal Strength**:

  It is also an important factor that triggers battery drain in some cases. If the wireless signal strength is weak, it will cause the Network Interface Card driver to increase the Tx/Rx power [63], which can increase the battery consumption. According to [61], about 11.11% posts indicated weak wireless signal strength to be the reason for battery depletion.

- **Wireless Handovers**:

  Repeated network handovers could also be the cause of battery drain. About 0.77% posts in [61] indicates that unexpected battery drains were observed with frequent wireless handovers.

2.5 Review of Existing Research

Pathak et al. [61] presented a taxonomy of energy bugs in smartphones and also discussed the reasons for these energy bugs. Although it is difficult to identify the root cause for energy bugs but possible reasons could either be programming errors, inappropriate API usage, flaw in design of applications or changing external conditions. It also presented a roadmap for developing a diagnostic framework to debug energy bugs.

Pathak et al. [64] discussed wakelock related energy bugs, which prohibit smartphones from going into a sleep mode. They then proposed a dataflow approach to detect these energy bugs.

Pathak et al. [62] presented design and implementation of *eprof*, an energy profiler for smartphone applications. They evaluated six different applications and proposed bundle presentation that helps in reducing energy consumption of applications.

Zhang et al. [69] discussed different energy models for different platforms. An incorrect implementation of these models can result in excessive battery use (drain). It explained how applications switch between different power models and also presented the similarities and differences between the different platforms.

Zhang et al. [70] developed ADEL, an Automatic Detector of Energy Leaks. It helps to detect and isolate the energy leaks due to unwanted network communication in mobile applications. ADEL determines whether the data received by an application is important to the user and also traces the direct and indirect use of this received data to determine its effects.

Jindal et al. [45] studied a class of energy bugs and refereed it as *sleep conflicts*, which can happen in smartphone device drivers when a particular component in a high power state is unable to transition back to the base power state. They presented the root causes of these sleep conflicts and classified them as well. They also presented a deployment scheme that successfully performed sleep conflict avoidance and showed its implementation on two Android smartphones.

Jindal et al. [46] discussed the sleep disorder bugs and presented a taxonomy of these bugs. They categorized the time critical sections, which are the root cause of these sleep disorders. The also presented a system to detect these sleep disorder bugs.

Muccini et al. [54] analysed the mobile application development and testing techniques and investigated that whether mobile applications are different than traditional software applications and if they require special testing techniques? They also debated on the challenges of mobile application testing strategies.

Wilke et al. [68] analysed the energy consumption issue in smartphone applications according to user feedback (users comments on Google Play market). It also identified major causes for inefficiency in smartphone applications.

Abdelmotalib and Wu [29] discussed energy management techniques in smartphones operating systems. They provided a summary of those techniques that can reduce the power consumption in mobile devices. They analysed and compared the general solutions and showed that efficient operating system techniques can sufficiently reduce power consumption in mobile devices.

Hao et al. [42] proposed a new approach, *eLens*, that provides fine grained estimate of energy consumption at the code level. eLens combines two ideas: (a) program analysis to determine the paths traversed and track energy information during execution, (b) per instruction energy modelling. For evaluation purpose it uses energy models provided by a software environment energy profile (SEEP), which enables the per instruction energy modelling.

Li et al. [48] designed and implemented the *Bugu* service that targets the applications running on mobile devices, analyses event power relationship and provides user with an overview of the power behaviour of applications. It consists of two parts: (a) Bugu server,

that collects the power information of applications and, (b) Bugu client, which uses the data from server and monitor the power consumption of applications.

Ma et al. [50] discussed the abnormal battery drain (ABD) issues in smartphones. It also presented the design and implementation of *eDoctor*, a practical tool that helps user to troubleshoot battery issues in smartphones. This tool captures the time varying behaviour of applications and then based on a diagnosis, it suggests user to take appropriate repair solutions. The paper also showed the accuracy and effectiveness of the tool by different experiments.

Oliner e al. [58] described the design and implementation of *Carat*, a tool for performing energy diagnosis on mobile devices. The Carat architecture includes (a) an application, (b) a central server and (c) an analysis running in the cloud. The application sends information and measurements to the server which identifies the energy issues with the users. The paper also shows an execution and evaluation of the tool.

S, Nakajima [57] discussed that energy bugs should be identified at early stages of development and not at the execution time because they are the design faults. This will give designers a way to improve their applications. The paper also proposed a model, power consumption automation, to account for power consumption. It also discussed how the tool assisted analyses are conducted.

Liu et al. [49] investigated the smartphone applications for energy inefficiency issues. Thy focused on sensor related energy issues in smartphone applications and advocated that mostly energy issues are due to ineffective use of sensors and their data by the applications. They presented an application model to simulate applications runtime behaviour. Their approach can analyse the sensory data utilization by the application at different states. They also presented a tool, *GreenDroid* on top of Java PathFinder and evaluated it using Android applications.

Balasubramaniam et al. [36] presented a measurement study of the energy consumption of the network technologies (GSM, 3G and WiFi) in smartphones. It also presented a model for energy consumption by each technology and using those models presented some protocols that reduced the energy consumption of different applications.

Vallina-Rodrihuez and Crowcroft [67] presented an extensive survey of the software solutions to achieve energy efficiency in smartphones. The survey covered different research areas namely: (a) energy-aware operating systems, (b) efficient resource management, (c) the impact of user interaction patterns with mobile devices and applications, (d) wireless interfaces and sensors management, and (e) benefits of integrating mobile devices with cloud computing services.

Chapter 3

Energy Performance Testing

In this chapter, we have presented the techniques and methodologies used to perform energy performance testing of smartphones. In section 3.1 we have presented an introduction to Energy Performance Testing and its difference from the Functional Testing. In section 3.2 we have shown a classification of configuration parameters of smartphone applications and their importance. In section 3.3 we have presented a summary of related work done in this domain. In section 3.4 we have described the test bench used to perform all the experiments. And in section 3.5 we have proposed a new technique to reduce the configuration parameters and then its validation across different smartphones.

3.1 Introduction

Applications that run on smartphones provide users with an easy way to access information on internet or act as a source of entertainment (gaming apps, video players etc.). These applications in turn consume the smartphone's limited resources like processor, memory, communication interface, memory and storage to perform the required tasks. One may argue that these hardware components practically consume the battery energy but the fact is that these software applications are the real consumer in a sense that they utilize the resources [31]. Figure 3.1 illustrates this concept and gives a high level over view of how software applications utilize the resources through middle-ware and operating system. And if these applications utilize the resources in an efficient way, the battery performance could be greatly improved and enhanced. Therefore applications have to be energy efficient to save and improve energy consumption in smartphones. On the other hand, the developers

should perform *Energy Performance Testing* along with *Functional Testing* to ensure that there applications are energy efficient and there are no energy related bugs in them.

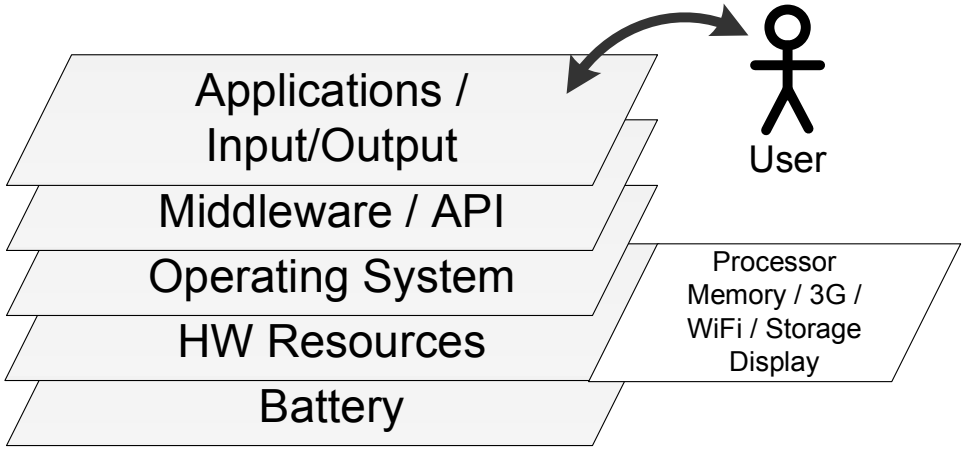
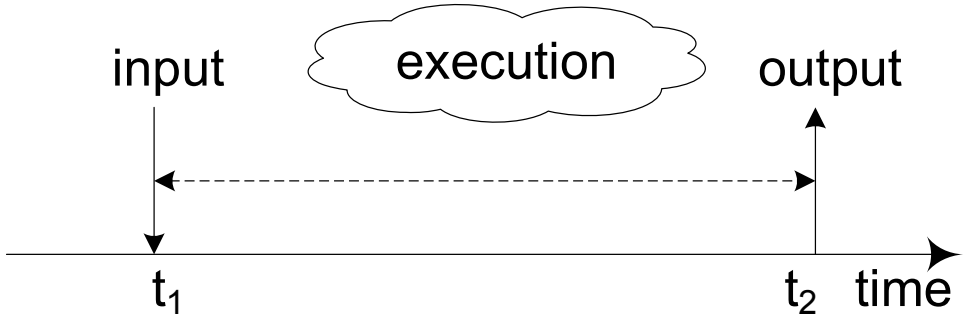


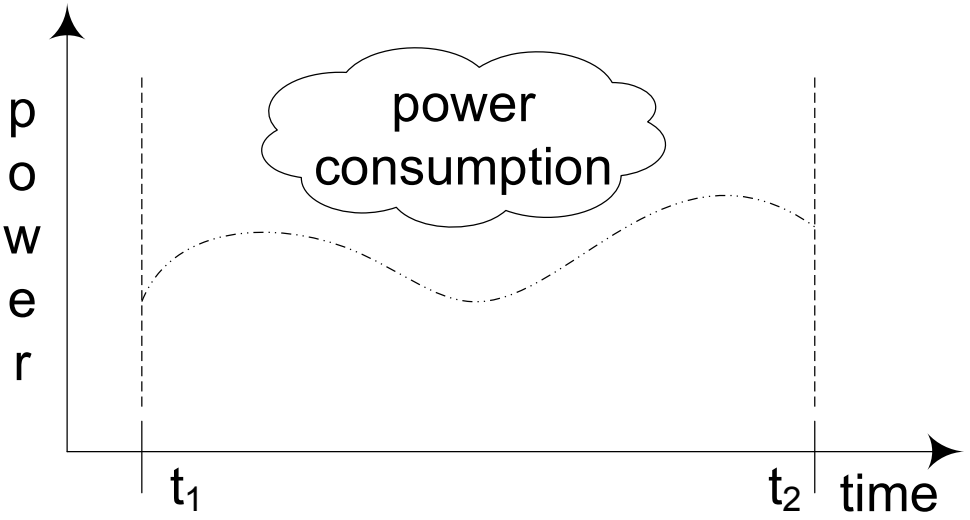
Figure 3.1: Smartphone environment in terms of energy consumption hierarchy.

Although there are some significant differences between functional testing and energy performance testing but experiments corresponding to a functional requirement and energy performance testing can be accomplished at the same time. As shown in figure 3.2(a), a user gives input to an application at time t_1, the device processes the input and provides output at time t_2. During the time interval $[t_1,t_2]$, the power consumption of the device can be monitored, and thus after time t_2, the output of the application can be verified as part of functional testing and the consumed energy can be calculated as a part of energy performance testing figure 3.2(b). This idea is also applicable to the testing of a component of an application.

Energy performance testing along with functional testing should be prerequisite for application development so that applications can be energy efficient and smartphone's battery consumption can be improved. Figure 3.3 depicts an energy-aware smartphone application development process model. Following this model, in the design phase; developers keep energy efficiency in mind and conduct both functional and energy performance testing after initial implementation of an application. The feedback from functional and energy performance testing helps them make changes in the design and implementation. Knowledge about the impact of the design decisions on energy consumption is very useful at the development stage as changes made in the final stage of an application are more expensive [47].



(a) Functionality testing.



(b) Energy performance testing.

Figure 3.2: Schematic diagrams of functionality and energy performance testing.

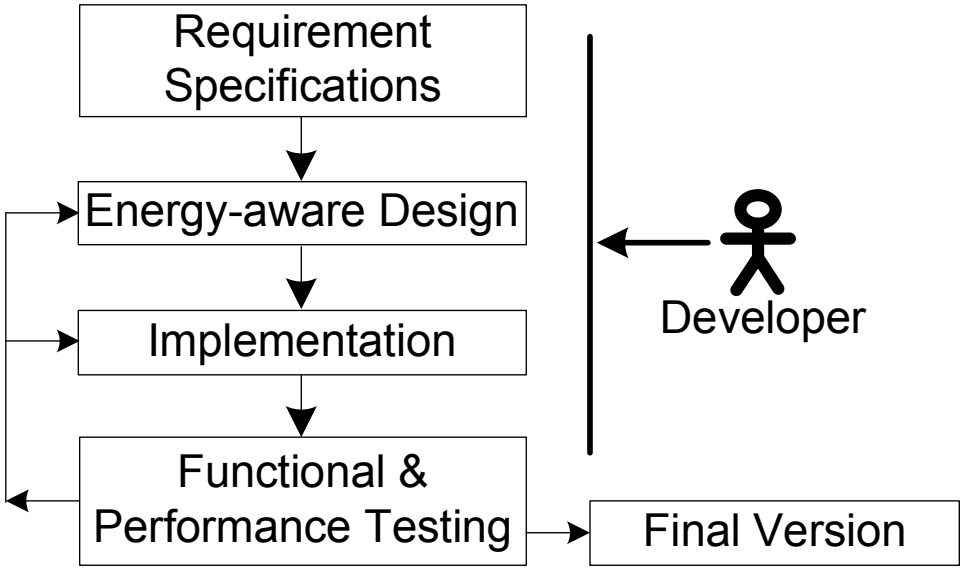


Figure 3.3: Smartphone application development process.

Smartphone applications consume battery energy based on the hardware resources that they utilize. This energy consumption by the smartphones can be measured by monitoring the power discharge rate from the battery [31]. In a similar way the energy consumed by the smartphone application can be calculated by taking the difference between the energy consumption of smartphone device *with* and *without* that specific application. The operating system in smartphones; while in its idle state, still consumes a certain amount of energy to keep the device running and ready for the applications to execute. In each of the above scenario; whether the smartphone device is in its idle state or it is executing an application, the *configuration* of smartphone *parameters* play an important role [31].

3.2 Configuration Parameters

As mentioned earlier, there are significant differences in energy performance testing and functional testing with both having different requirements. In functional testing the input (parameters) to the system is treated equally whereas, in energy performance testing more significance is given to the configuration parameters that eventually impact the consumption of battery energy [30]. Due to a large number of configuration of smartphone applications, there are enormous number of test cases [33]. Therefore there is a need for a *selection technique* to reduce this enormous number of test cases. In order to evaluate

the energy cost of smartphone applications, [59] proposes a mechanism for user level test cases. In [30], the author outlines the concept of *parameters* and *configurations*.

Intuitively, categorization of these parameters according to their impacts on the energy consumptions is useful in energy performance testing. As illustrated in figure 3.4, the parameters are categorized into three groups: *basic*, *active*, and *passive* parameters. The active parameters are further partitioned into two groups: *primary* and *standalone* parameters. All those groups are explained in the following [56]:

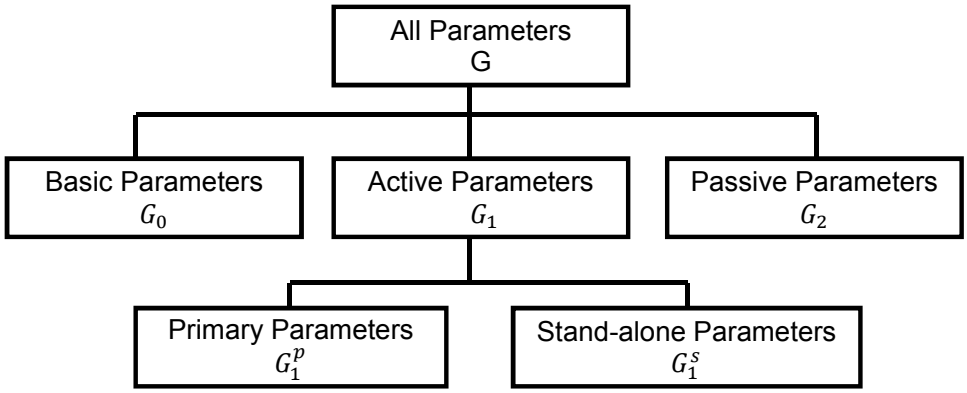


Figure 3.4: Categorization of smartphone parameters.

- *Basic Parameters* (G_0):

  This group contains all the parameters whose values are fixed. Table 3.1 shows examples of basic parameters of the four smartphones. These parameters affect the energy consumptions of a device, and their impacts just remain the same throughout the testing process.

- *Active Parameters* (G_1):

  This group contains the parameters whose values are configurable and they particularly control the hardware components. Table 3.2 gives some active parameters for the four smartphones. The active parameters are further divided into following two groups:

– *Primary Parameters* (G_1^p)

– *Standalone Parameters* (G_1^s)

The primary parameters cause the device to consume a significant proportion of the total power and their variations yield informative test results. A technique to further reduce the primary parameters of smartphone is discussed in Section 3.5. The remaining active parameters are called standalone parameters; these are considered for performing standalone configurations.

- *Passive Parameters* (G_2):

  This group contains the rest of the parameters whose values are also settable by the users. They do not control the hardware components but they use hardware components so they affect the test outcomes. Thus, we turn off or fix certain values for these parameters. Examples of passive parameters have been shown in Table 3.3.

Since the operational behaviour of a smartphone is largely determined by its configuration parameters, it is important to identify and classify those parameters. We considered four smartphones, namely, *BlackBerry 9700*, *BlackBerry Z10*, *Apple iPhone 3GS*, and *Samsung Galaxy Nexus* and identified their configuration parameters. Tables 3.1, 3.2, and 3.3 show the *basic*, *active (primary)*, and *passive* parameters, respectively.

B_i	Parameter	Description	Samsung Galaxy Nexus	BlackBerry Z10	Apple Iphone 3GS	BlackBerry Bold 9700
01	Display	Size of display	1280 x 720 pixels, 4.65”	S1280 x 768 pixels, 4.2”	320 x 480 pixels, 3.5”	480 x 360 pixels, 2.4”
02	Operating System (OS)	Name of the OS	Android	BlackBerry OS 10	Iphone OS 3	BlackBerry OS
03	Battery Capacity	Type and capacity	Li-ion 1,750 mAh	Li-ion 1800 mAH	Li-ion 1500 mAh	Li-ion 1250 mAh

Table 3.1: Examples of Basic Parameters (G_0)

B_i	Parameter	Description	Samsung Galaxy Nexus	BlackBerry Z10	Apple Iphone 3GS	BlackBerry Bold 9700
31	Volume	Allows user to adjust volume level	Yes (0-8)	Yes (0-16)	Yes (0-16)	Yes (0-10)
32	Brightness	Allows user to adjust brightness level	Yes (0-100%) Continuous	Yes (0-100%) Continuous	Yes (0-100%) Continuous	Yes (0,10.. ..100)
33	Bluetooth	Allows user to turn on/off bluetooth	Yes	Yes	Yes	Yes
34	Camera	Allows user to turn on/off camera	Yes	Yes	Yes	Yes
35	Network Access Mode (NAM)	Allows user to turn on/off network mode and select from WiFi/3G	Yes	Yes	Yes	Yes
36	GPS	Allows user to turn on/off GPS	Yes	Yes	Yes	Yes
37	NFC (Near-Field-Comm.)	Allows user to turn on/off NFC	Yes	Yes	No	No

Table 3.2: Examples of Active Parameters (G_1); "Yes" means the parameter is available; "No" means the parameter is not available.

B_i	Parameter	Description	Samsung Galaxy Nexus	BlackBerry Z10	Apple Iphone 3GS	BlackBerry Bold 9700
61	Network selection mode	This option lets the mobile device to select the network manually or automatically	Yes (Auto/ Manual)	Yes (Auto/ Manual)	Alternative (Auto - Select/ Deselect)	Yes (Auto/ Manual)
62	WiFi settings - Network Notification	This option prompts user when a WiFi network is available	Yes (On/Off)	Yes (On/Off)	Yes (On/Off)	Alternative (Prompt when manual connection or login is required)
63	Portable WiFi Hotspot	This option allows the device to act as a WiFi hotspot	Yes	Yes	Alternative (Setup Internet Tethering)	No
64	Screen Timeout	This option allows the user to set the display timeout for screen	Yes (15 sce -10 min)	Yes (10 sec -5 min)	Yes (1 - 5 min/never)	Yes (10 sec -2 min)

Table 3.3: Examples of Passive Parameters (G_2); "Yes" means the parameter is available; "No" means the parameter is not available; "Alternative" means a similar parameter is available.

3.3 Related Work

Accurate energy performance testing of smartphones is a challenging task, because there are millions of system configurations [59] and it is extremely difficult to control the operational environment of a smartphone [59], [53]. To reduce the number of test configurations for conducting energy performance testing; in our previous work [59] and [30], we:

- studied the parameters of five different state-of-the-art smartphones, and categorized them into three groups: *basic*, *active* and *passive* (as discussed in section 3.2 in detail);
- showed that there is no need to consider all possible combinations of all the different values of all the configuration parameters;
- partitioned the active parameters into *primary* and *standalone* parameters;
- introduced the concepts of *primary* and *standalone* test configurations;
- gave a proceduce to apply the methodology to derive the reduced number of test configurations.

Although exhaustive testing; considering all possible combinations of inputs is desirable [37], time and resource limitations dictate that only a subset of all test cases be considered as a practical test space [51]. Therefore, reduction of test space is a continued topic of interest. Pairwise testing is a well-known technique that covers all possible pairs of parameter states. For each and every pair of states that belong to different parameters, there is at least one test case that contains both of those states [39]. This idea can easily be generalized by covering t-wise combinations of states rather than pairs [55].

As an example of pairwise testing, consider a system H with three input parameters x, y and z, where their domains are: $D(x) = \{Yes, No\}$, $D(y) = \{1, 10\}$ and $D(z) = \{True, False\}$. The number of all possible test cases for this system is $2 \times 2 \times 2 = 8$; however, all pairwise combinations can be covered by only 4 cases, as shown in Table 3.4.

Test ID	x	y	z
TC_1	Yes	1	$True$
TC_2	Yes	10	$False$
TC_3	No	1	$False$
TC_4	No	10	$True$

Table 3.4: Pairwise test cases for system H.

Unlike pairwise testing, where each possible pair of parameter states is considered in at least one of the test cases, the strategy proposed by Palit *et al.* [59] and Abogharaf *et al.* [30] consider all possible combinations between primary parameter states and other active parameter states. The latter technique puts more emphasis on parameters that have more significant impacts on the energy performance metric.

Selection of a reduced number of test configurations for energy performance testing of smartphones was originally proposed in [59] and [30]. First, we assumed that the primary parameter set comprised of just one parameter [59], whereas the restriction was completely eliminated in [30]. Second, in [30] we proposed a flowchart to derive test configurations for a smartphone by applying the concepts developed in [59] and [30]. Since the current work extends the concepts developed in reference [30], we briefly explain those concepts in the following.

Assume that there are m active parameters in a smartphone d grouped in the set $G_1^d = \{B_1, B_2, \cdots, B_m\}$. A specific state of a parameter B_j is $b_j^{(l)}$, and it has $\eta(B_j)$ states. For example, the Bluetooth parameter B_{33} in Table 3.2 has a state $b_{33}^1 =$ 'ON', and $\eta(B_{33}) = 2$ (i.e. ON and OFF).

A configuration β_i represents an instance of all settable parameters of a smartphone. An app A_i processes a certain types of contents denoted by C_i, and the number of contents for app A_i is denoted by $\eta(C_i)$. For example, a video player application A_1 supports only two file formats: $c_1^1 = MPEG$ and $c_1^2 = Flash$. Thus, $\eta(C_1) = 2$.

Assuming that one test case is used to cover one test configuration, the total number of test cases N_c is as follows:

$$N_c = S_d \times \sum_{i=1}^{M} X_d(A_i) \times \eta(C_i), \tag{3.1}$$

where S_d is the total number of device configurations for a smartphone d, M is the number of chosen applications, and $X_d(A_i)$ indicates whether or not application A_i is available on d. The number of all possible configurations on a smartphone d can be expressed as:

$$S_d = \prod_{j=1}^{m} \eta(B_j) \tag{3.2}$$

S_d in Eq. 3.2 is extremely large (i.e. billions) for modern smartphones [59]. To reduce the value of S_d to a reasonably small value (e.g. tens), we: (i) partitioned the active parameter set into two subsets, namely, primary parameters and standalone parameters; and (ii) design primary configurations and standalone configurations. The primary configurations are obtained by varying only the primary parameters, whereas standalone configurations are obtained by varying standalone parameters one by one for all states of the primary parameters. Now, N_c is expressed as:

$$N_c = (S_d^P + S_d^S) \times \sum_{i=1}^{M} X_d(A_i) \times \eta(C_i), \tag{3.3}$$

where S_d^P is the number of primary configurations, and S_d^S is the number of standalone configurations.

The dependency between parameters is a key idea in our methodology. Suppose the parameters $B_i = \{b_i^1, b_i^2 \cdots, b_i^I\}$ and $B_j = \{b_j^1, b_j^2 \cdots, b_j^J\}$ are primary parameters, and $\theta(b_i^u, b_j^q)$ is the energy cost of a test case involving configuration parameters $b_i^u \in B_i$ and $b_j^q \in B_j$. The parameters B_i and B_j are independent of each other, denoted by $B_i \perp B_j$, if their energy costs are additive; that is:

$$B_i \perp B_j \iff$$

$$\left[\left(\theta\left(b_i^u, b_j^q\right) - \theta\left(b_i^v, b_j^q\right)\right) \simeq \left(\theta\left(b_i^u, b_j^r\right) - \theta\left(b_i^v, b_j^r\right)\right)\right]_{u \neq v, q \neq r}$$

Suppose a set G_1^p, s.t. $G_1^p \subseteq G_1$, represents the set of primary parameters for smartphone d. In order to obtain general expressions for number of configurations, we make the following assumptions:

- G_1^p is said to be independent if all members of G_1^p are independent of each other. The notation $Ind(G_1^p)$ means that the set G_1^p is an independent set.
- A parameter B_j is independent of the set G_1^p if B_j is independent of all members of G_1^p. The notation $Ind(B_j\,,\,G_1^p)$ signifies that B_j is independent of all members of G_1^p.

The number of the primary configurations S_d^p is expressed as follows [30]:

$$S_d^P = \begin{cases} 1 + \sum_{i=1}^{k} \left[\eta\left(B_i\right) - 1\right] + \frac{k(k-1)}{2} & , \; Ind(G_1^p) \\ \prod_{i=1}^{k} \eta\left(B_i\right) & , \text{ Otherwise} \end{cases} \tag{3.4}$$

Looking at the upper expression of Eq. 3.4, which is applied when G_1^p is independent (i.e. $Ind(G_1^p)$ holds.), the first term represents the first experiment needed to be conducted which could have any parameter settings. Since the primary set is independent, the second term represents the number of test cases required to be conducted across each parameter in the primary set but subtracting one state, which was used in the first experiment, from each parameter. The third term stands for the minimum number of test cases required to examine the independency among the parameters. The lower expression basically represents all possible combinations of the parameters in the primary set since they are dependent parameters.

The number of standalone configurations corresponding to a parameter B_j is Q_j, as expressed in what follows:

$$Q_j = \begin{cases} \eta(B_j) - 1 + k & , \; Ind(B_j\,,\,G_1^p) \\ \left[\eta\left(B_j\right) - 1\right]\left[1 + \sum_{i=1}^{k}\left(\eta\left(B_i\right) - 1\right)\right] & , \; Ind(G_1^p) \\ \left[\eta\left(B_j\right) - 1\right]\prod_{i=1}^{k}\eta\left(B_i\right) & , \text{ Otherwise} \end{cases} \tag{3.5}$$

In the case where B_j is independent of G_1^p, the first and the second terms represent the number of experiments required to be conducted across B_j where one of the states of B_j was already conducted in the primary stage. The third term represents the minimum number of experiments required to verify the independency of B_j among all parameters in G_1^p.

In the case where G_1^p is independent, the same number of configurations required in the primary configurations in the case where G_1^p is independent is needed to be conducted for the rest of the states of B_j; in here we do not need to include the term used to examine the independency because we would already know, from the primary stage, that G_1^p is independent. When G_1^p is a dependent set, all possible combinations are required to be considered while subtracting the configurations which were conducted in the primary stage. The number of all standalone configurations S_d^S can now be expressed as:

$$S_d^S = \sum_{\forall B_j \notin G_1^p}^{m} Q_j \tag{3.6}$$

The flowchart in [30] obtains a reduced number of test configurations as explained above.

3.4 Experimental Setup

The test bench used to conduct experiments for validation of our proposed technique is shown in figure 3.5 [30]. A smartphone under test is powered by an external power supply with a high precision current measurement unit. A desktop or a laptop computer monitors the power supply unit and gathers the measurements from the power supply throughout the duration of an experiment.

A special battery connection setup is made to power the smartphone from external power supply; where we disconnect the battery's power interface but keep the communication interface connected to the smartphone. In order to have accurate measurements and to avoid the power saving mode, the smartphone's battery has to be fully charged during all the experiments.

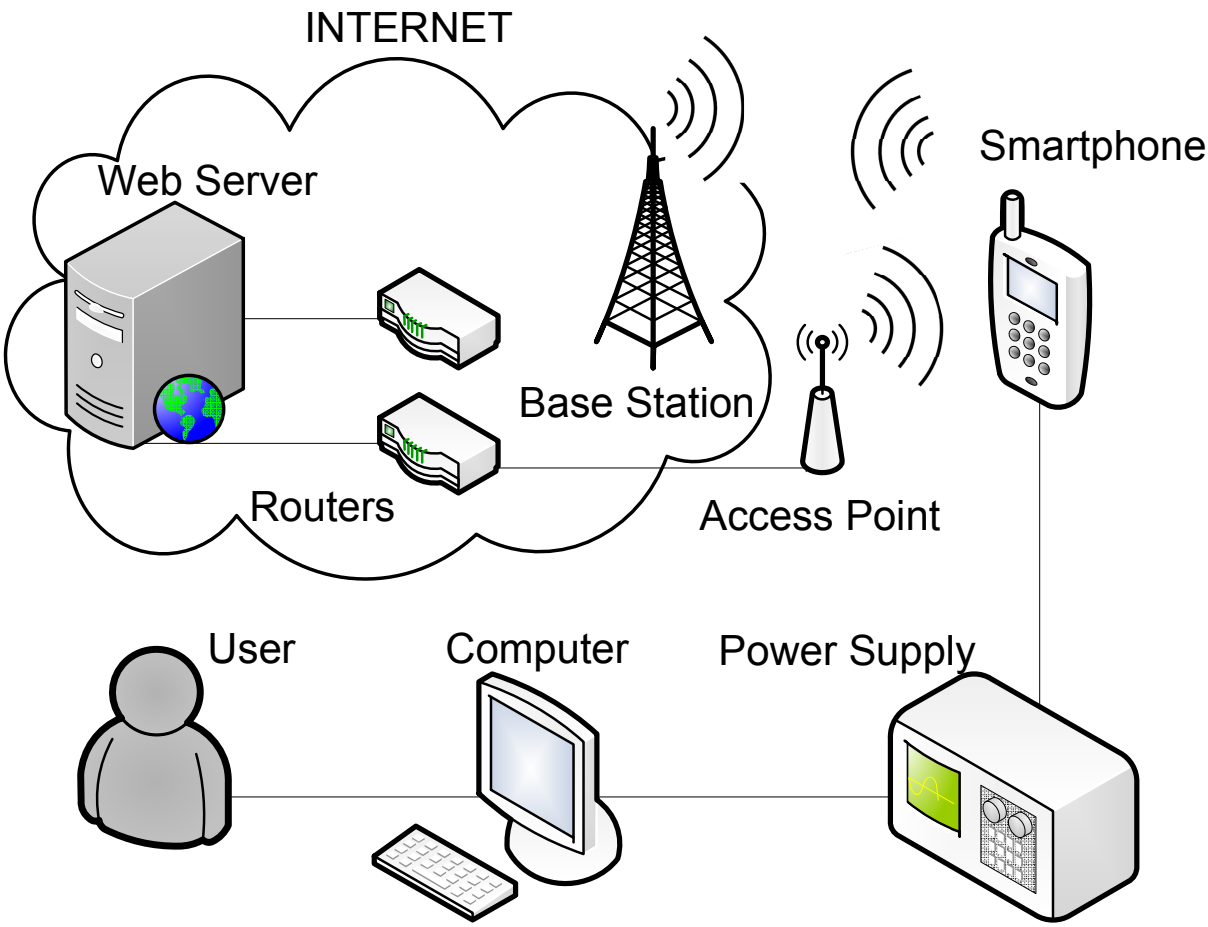


Figure 3.5: Experimental Setup.

3.5 Proposed Methodology

We propose a methodology to select the primary parameters from the active parameters. Intuitively, a larger number of primary parameters means more test configurations to be used, thereby giving better accuracy in testing at the cost of more testing. A key attribute of our technique is that testers can fine tune their needs by means of a control variable (discussed in section 3.5.1).

3.5.1 Maximum Differential Power

Intuitively, an active parameter that consumes a significant amount of power is a primary parameter, and an active parameter that consumes a very small amount of power is a standalone parameter. Therefore, the subset of active parameters which account for most of the power cost is viewed to be the primary set, and the remaining active parameters are put in the standalone set.

Now we introduce the concept of *maximum differential power* of an active parameter B_i, denoted as $maxDP(B_i)$. For B_i, $maxDP(B_i)$ is calculated as the maximum power difference among all pairs of states of B_i. Alternatively, since the off state of B_i consumes the least power, $maxDP(B_i)$ is the difference between the maximum power consumed among all states of B_i and the minimum power consumed in its off state. In the following, we give a procedure, namely, ComputePrimary() to compute G_1^p by using a threshold value called τ.

Begin Procedure ComputePrimary()

Step 1: Find the power consumption of all active parameters in all states.

Step 2: Sort the parameters of G_1^p decreasingly according to their maximum differential power consumption.

Step 3: Let $SDP = \sum maxDP(B_i)$, $\forall B_i$, $B_i \in G_1$. Identify G_1^p to be the minimal subset of G_1 such that the sum of the differential power of all the members of G_1^p is at least $\tau\%$ of SDP.

End

That is,

$$G_1^p = \left\{ B_i \left| \sum_{i=1}^{k} maxDP\left(B_i\right) \geq \tau\% \;\; of \;\; SDP, B_i \in G_1 \right. \right\} \tag{3.7}$$

For example, the Network Access Mode (NAM) parameter can be in one of 3 states (Table 3.5), namely, Off, WiFi, and 3G. The Network Access Mode (NAM) parameter in figure 3.6 for the *Samsung Galaxy Nexus* phone consumes the least power in its Off state and the maximum power in its 3G state. Therefore, $maxDP(NAM)$ for the *Samsung Galaxy Nexus* is the power difference between its 3G state and its Off state.

3.5.2 Validation

To validate the methodology, we conducted tests on four different devices:

- Samsung Galaxy Nexus (Android)
- BlackBerry Z10 (BB 10 OS)
- BlackBerry Bold 9700 (BB 7 OS)
- iPhone 3GS (iOS).

We used the test bench outlined in Section 3.4 [30]. For all the experiments on above mentioned devices, we considered the active parameters shown in Table 3.2. Table 3.5 shows the number of states of these active parameters used in experiments.

Out of four devices considered for testing purposes, two of them (*Samsung Galaxy Nexus* and *BlackBerry Z10*) are new, whereas the other two (*BlackBerry Bold 9700* and *Apple iPhone 3GS*) are relatively old. For the older smartphones we considered 6 active parameters, and for the newer smartphones we considered 7 active parameters. The reason for this is the fact that the newer smartphones have more parameters then the older generation of smartphones. In the following experiments, we set $\tau = 85\%$.

The parameters and their states are selected so that there is consistency in the experiments. For example, the active parameter 'Brightness' has discrete values (0 - 100) for *BlackBerry Bold 9700*, but it is continuous for the other three devices; for consistency, we

	Volume	Brightness	Bluetooth	NFC	GPS	NAM	Camera
Samsung Galaxy Nexus	9	2	2	2	3	3	2
BlackBerry Z10	17	2	2	2	3	3	2
Apple iPhone 3GS	17	2	2	NA	3	3	2
BlackBerry Bold 9700	11	2	2	NA	3	3	2

Table 3.5: Number of states for active parameters of Smartphones (NFC: Near Field Communication; GPS: Global Positioning System; NAM: Network Access Mode; NA: Not Available).

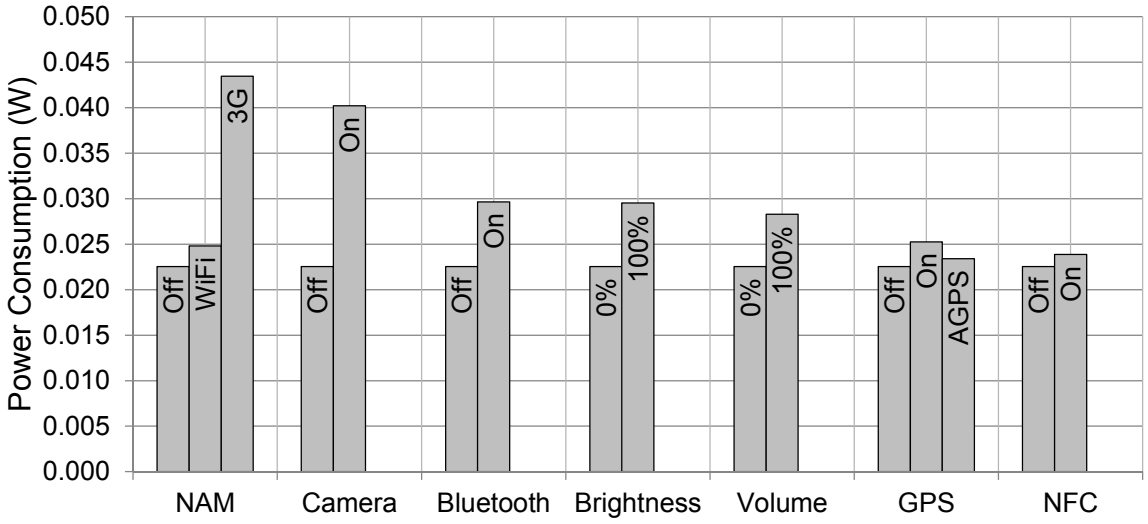


Figure 3.6: Power consumption of all the states for active parameters of *Samsung Galaxy Nexus*

considered only two states for 'Brightness': 0% and 100%. Similarly the active parameter 'Volume' has different states for all the four devices so we considered the minimum and maximum volume for all the devices.

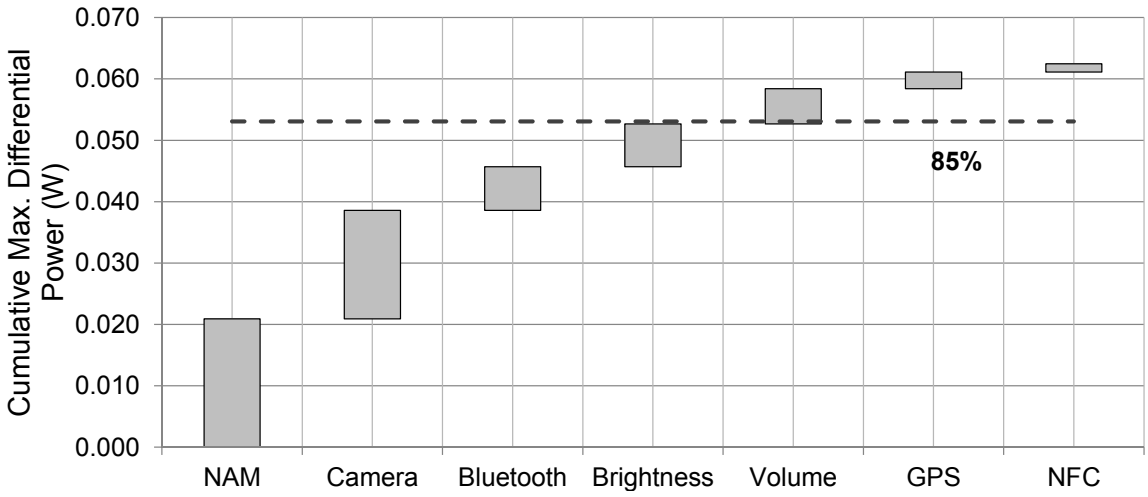


Figure 3.7: Obtaining the primary parameters for *Samsung Galaxy Nexus*.

In figure 3.6, we have shown power consumption of active parameters of *Samsung Galaxy Nexus* smartphone. Figure 3.7 shows the cumulative maximum differential power costs of all the active parameters in decreasing order. For this smartphone, with $\tau = 85\%$; which is represented by the dotted horizontal line, the four parameters in the primary set that strongly impact the power consumption are *Network Access Mode (NAM), Camera, Bluetooth* and *Brightness*.

Next, we consider *BlackBerry Z10* and plot the power costs of its active parameters and their cumulative maximum differential power cost in figure 3.8 and figure 3.9, respectively. For *BlackBerry Z10*, with $\tau = 85\%$; there are three parameters in the primary set that strongly impact the power consumption: *Network Access Mode (NAM), Brightness* and *Volume*.

Figures 3.7 and 3.9 show that the consumption of power by the same parameters are different on different devices. Our technique yields four primary parameters for *Samsung Galaxy Nexus* and three primary parameters for *BlackBerry Z10*. However, Network Access Mode (NAM) and Brightness are the common parameters in both of the primary sets.

Next, we consider an *iPhone 3GS* and plot the power costs of its active parameters and their cumulative maximum differential power cost in figure 3.10 and figure 3.11, respectively. For this smartphone, we have considered 6 active parameters because it belongs to

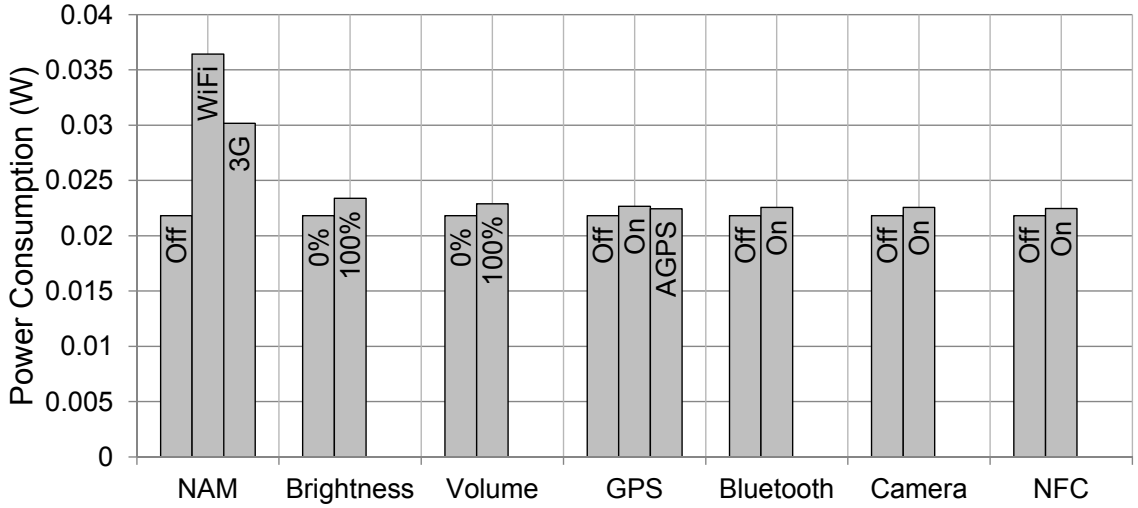


Figure 3.8: Power consumption of all the states of the active parameters of *BlackBerry Z10*.

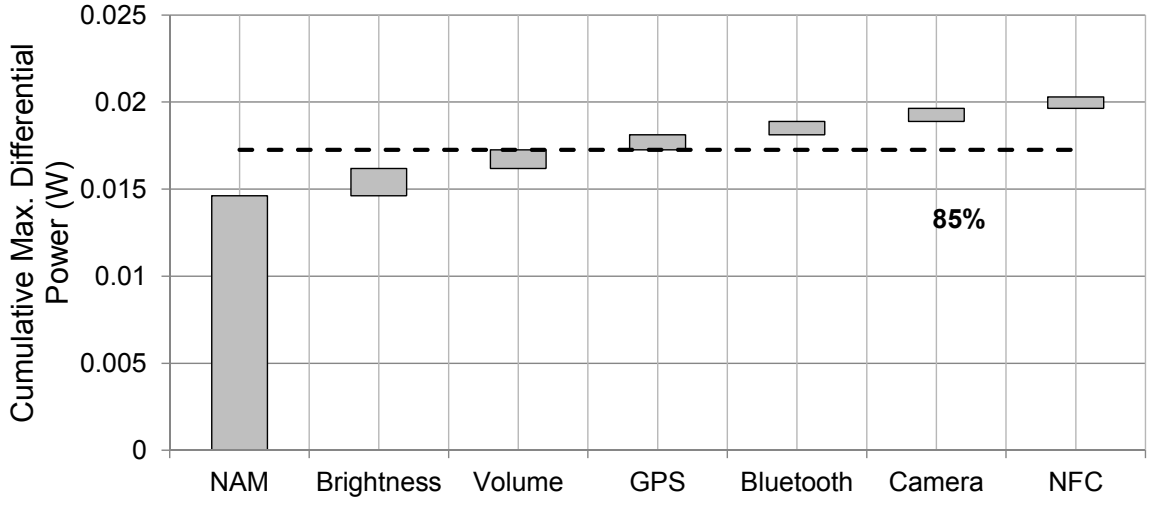


Figure 3.9: Obtaining the primary parameters for *BlackBerry Z10*.

a relatively old generation of smartphones. Considering $\tau = 85\%$ for *iPhone 3GS*, there are three parameters in the primary set: *Network Access Mode (NAM), Camera* and *Bluetooth* that strongly impact the power consumption.

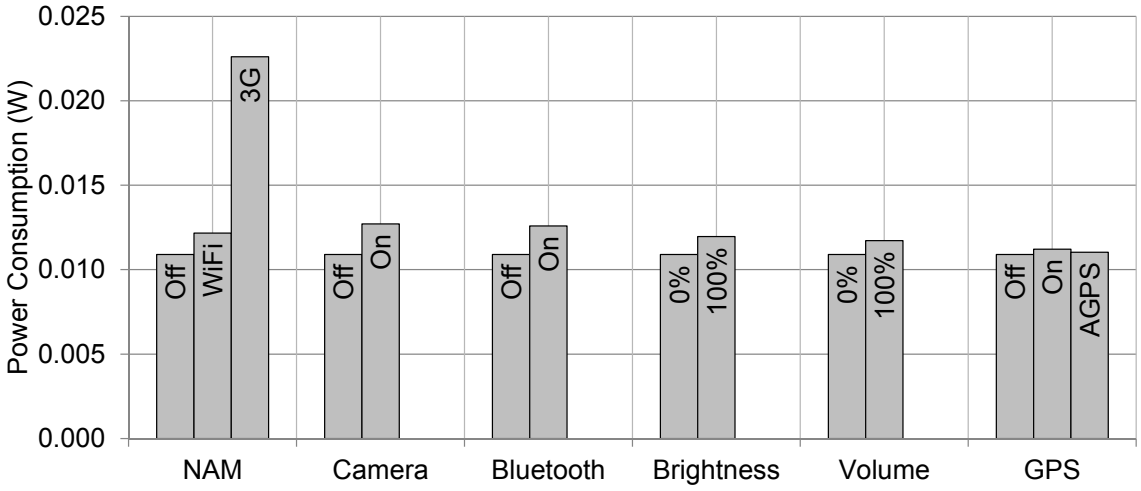


Figure 3.10: Power consumption of all the states of the active parameters of *iPhone 3GS*.

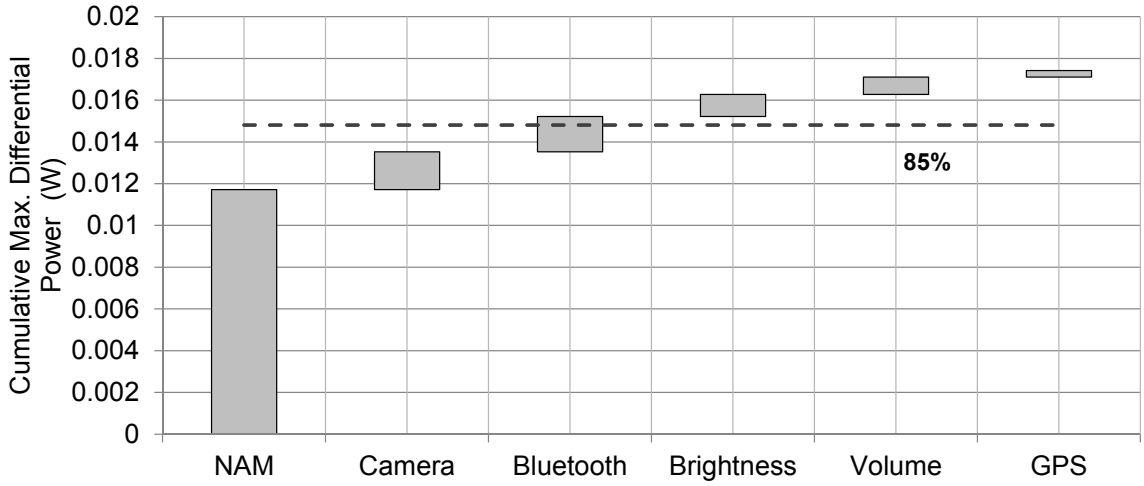


Figure 3.11: Obtaining the primary parameters for *iPhone 3GS*.

Next, we consider a *BlackBerry Bold 9700* and plot the power costs of its active parameters and their cumulative maximum differential power cost in figure 3.12 and figure 3.13, respectively. With the same $\tau = 85\%$ threshold, there are two parameters in the primary set: *Network Access Mode (NAM)* and *Bluetooth* that strongly impact the power consumption.

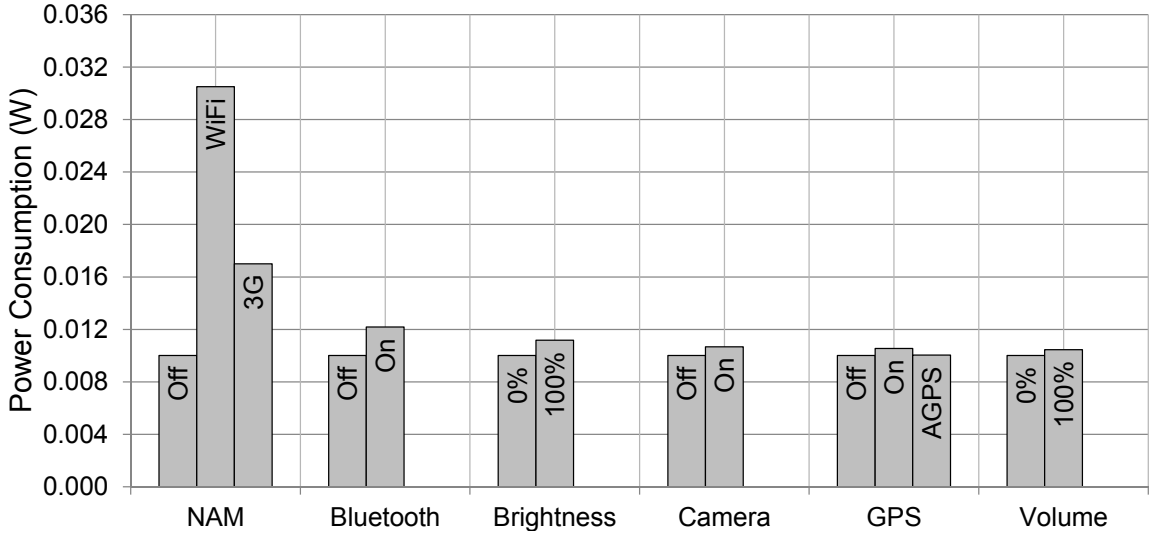


Figure 3.12: Power consumption of all the states of the active parameters of *BlackBerry Bold 9700*.

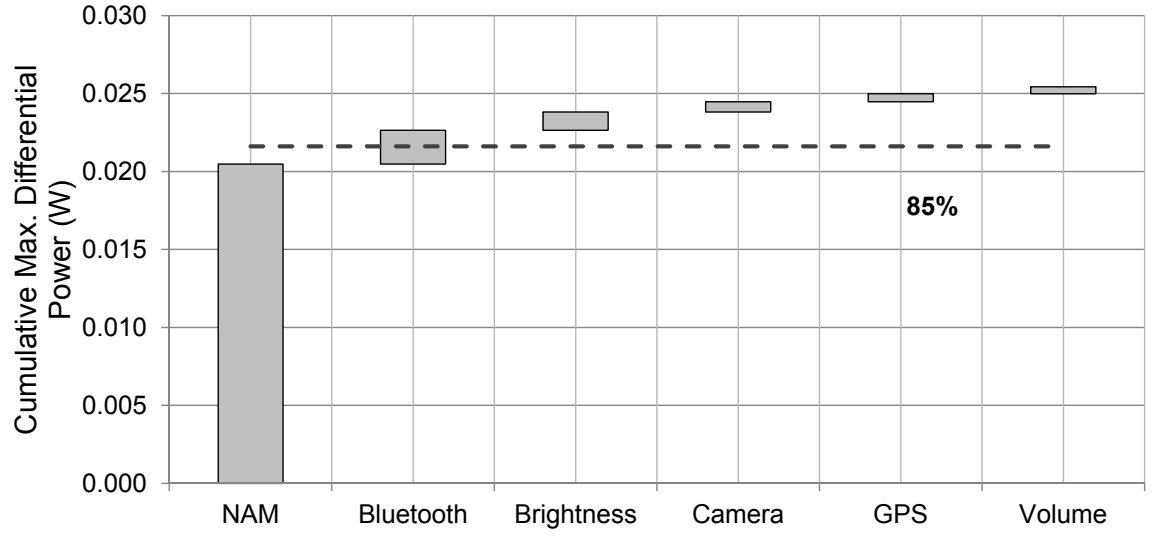


Figure 3.13: Obtaining the primary parameters for *BlackBerry Bold 9700*.

Comparing figure 3.11 and figure 3.13, we notice that the technique yields three primary parameters for *iPhone 3GS* and two primary parameters for *BlackBerry Bold 9700*. However, Network Access Mode (NAM) and Bluetooth are the common parameters in the two primary sets for both the devices.

Our experiments on four different smartphones from three makers and with four different operating systems reveal the following interesting observations:

- Conforming to intuition, the technique selects the primary parameters which consume more power. As a consequence, the technique does not select those active parameters which do not incur much power cost.
- The technique selects very similar primary parameters for all the devices.
- The number of primary parameters for a device is a characteristic of the device itself, which is intuitive.
- Obviously, a larger τ will produce a larger number of primary parameters, thereby increasing the number of test configurations.
- Test designers have the option to choose an appropriate τ to strike a balance between time to test and exhaustiveness of testing.

Chapter 4

Energy Bug

Energy bugs in smartphones have been a concern for the manufacturers, the users and the application developers. In comparison to traditional software bugs, energy bugs are not only hard to detect but it is also difficult to find their root causes [61]. The reason for this is that in most of the cases energy bugs do not only affect the system performance and its operation but also consume large amount of energy. From software testing perspective, it is imperative to have a definition in literature which can be used as a reference. None of the studies surveyed for this thesis discuss energy bug from the software testing point of view. In this chapter, we have defined energy bug relative to software testing domain (section 4.2). In section 4.3 and section 4.4 we have discussed a couple of scenarios and have verified our definition with measurements and experiments.

4.1 Introduction

Energy bug is usually credited as an error in the system but it is not always the case. The presence of energy bug might or might not affect the system output. There are many aspects of battery and smartphone that have to be taken into consideration before defining *energy bug*. In the following work we have researched the literature, on-line resources and have used experimentation to define the energy bug phenomenon.

4.2 Energy Bug: A Comprehensive Definition

We define *energy bug* as follow:

Bug that affects the smartphone's (system) battery consumption and in some cases leaves the smartphone in a different power consuming state. Energy bugs might or might not affect the system's output.

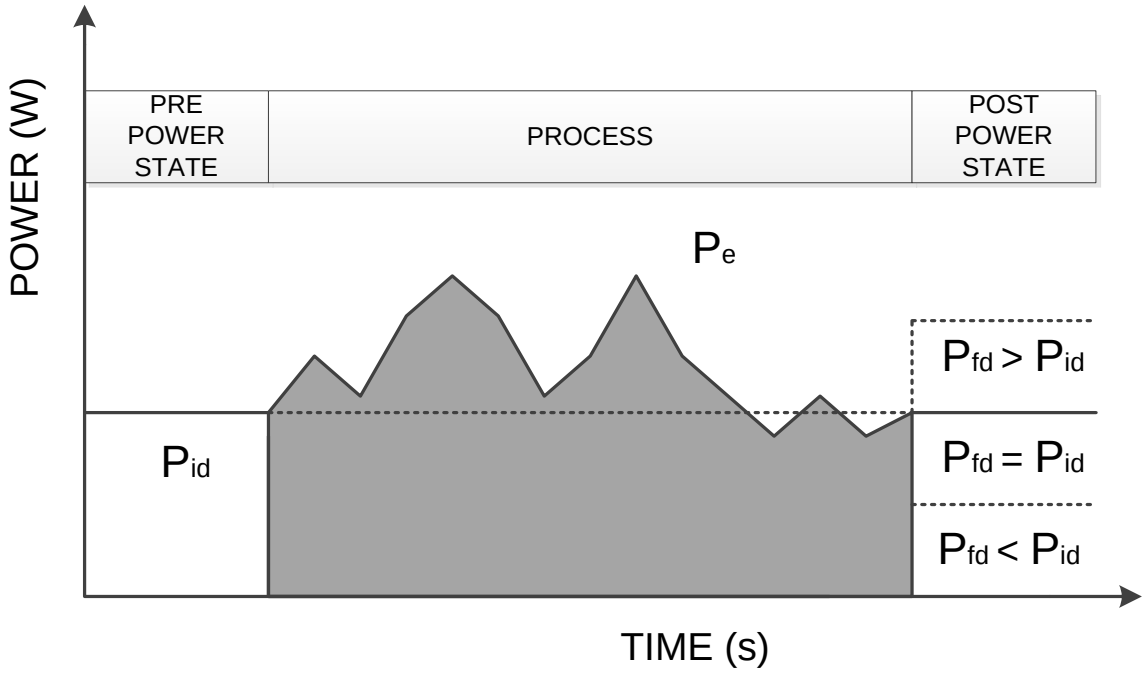


Figure 4.1: Complete execution cycle of an application process

Figure 4.1 shows the execution cycle of a complete application process for power consumption pattern in smartphones. Let us suppose there are three states with respect to power consumption for a process to complete its task: *pre power state*, *post power state* and the state where actual process takes place. Now suppose that in the pre power state the device was idle consuming P_{id} power before the execution of the process started (where "id" stands for initial idle state). Then during the execution of the process, the device consumes P_e power. After the execution time t_e seconds, the device goes back to the idle

state. Suppose that in this final state, the device being idle consumes P_{fd} power (where "fd" stands for final idle state) and that the device remains in idle state for t_d seconds before and after the execution of process. Here we have assumed that the idle state of the device is when the device is not in sleep mode but is actually running with basic applications. And whenever a device has to execute a process, it is an additional application that requires the device to perform additional tasks consequently consuming more power from the battery.

According to the definition, a device is in an ideal working condition without the presence of an energy bug if the post-condition of the device after execution of the process is same as the pre-condition of the device before the execution of process. This energy cost relationship is given in Eq. 4.1:

$$P_{fd} = P_{id} \tag{4.1}$$

$$\rightarrow \text{Post Power State} = \text{Pre Power State}$$

$$\rightarrow \text{Actual Output} = \text{Desired Output}$$

Now if there is an energy bug in the device (system), the desired output will not be achieved and it may lead the system to either of the following two states:

1. *High power consuming state:*

   In this state the device will consume more power than it was consuming before the execution of the process started. It will not only affect the output of the process but will also lead to drainage of the device battery. Following equation explains this state:

$$P_{fd} >> P_{id} \tag{4.2}$$

$$\rightarrow \text{Post Power State} \neq \text{Pre Power State}$$

$$\rightarrow \text{Actual Output} \neq \text{Desired Output}$$

2. *Low power consuming state:*

   In this state the device will consume less power than it was consuming before the execution of the process started. It will not only affect the output of the process

but will also affect the operation and performance of the device. Following equation explains this state:

$$P_{fd} << P_{id} \tag{4.3}$$

$$\rightarrow \text{Post Power State} \neq \text{Pre Power State}$$

$$\rightarrow \text{Actual Output} \neq \text{Desired Output}$$

The important thing to realize here is that an energy bug in the system can lead to either of the high or low power consuming states of the device which in turn might lead to the failure of the process. Other possibility could be the fact that the requirements of the execution time for the process are not followed precisely. In the latter case where requirements of execution time are not fulfilled, there may or may not be an energy bug in the system.

The new generation of smartphones are equipped with a large number of sensors and their presence not only enhances the functionality of the device but also allows the users to perform a variety of different tasks and access information in a convenient way. In order for a device to function properly in an ideal way, these sensors have to work in a perfect way as they are supposed to. The reason for this is that any malfunction in a sensor could possibly be due to presence of an energy bug, as it directly affects the power consumption of a device.

Proximity sensor is one of the numerous sensors present in new generation of smartphones. A proximity sensor is a sensor, able to detect the presence of nearby objects without any physical contact. A proximity sensor often emits an electromagnetic field or a beam of electromagnetic radiation (infra-red, for instance) and looks for changes in the field or return signal. The object being sensed is often referred to as the proximity sensor's target [1]. Proximity sensors are commonly used in smartphones to detect (and skip) accidental touch screen taps when held to the ear during a call [2].

In the following, we will give two real life examples of the proximity sensor malfunction and presence of an energy bug. Where in first case an energy bug in the system leads to massive battery drain leaving the device in a high power consuming state. In the second case an energy bug in the system affects the basic operation and performance of the device and leaves the device in a low power consuming state then the pre-process power consuming state.

4.2.1 High Power Consuming State

As explained earlier, this state of a device not only affects the output of the process but also leads to excessive battery consumption and in some cases completely drains the battery. Although the execution time of a process is an important factor that might lead the device to this high current consuming state but this is not a factor majority of the time. That leaves us with the important problem of an "Energy Bug".

An energy bug that caused the device to consume massive battery power and eventually led to battery drainage was detected on iPhone 4S running iOS 5.0.1 and 6.1.1 [6], [13], [18], [12], [21]. Apple's iPhone 4S was introduced with a new feature called *Siri*, an intelligent assistant that helps user get things done just by asking [7]. Although Siri was a new and exciting feature but its initial release had a serious energy bug related to it that was causing massive battery drain leading the device to a high power consuming state.

In order to activate the Siri, users had to long press the home button. Siri however, had another option to activate it from the settings called *"Raise to Speak"*. This option if activated, allowed the users to use Siri by just raising the phone to their ear. With this option turned on, the iPhones proximity sensor would remain in an active state (consuming more power) for the duration of time till this option was on (as normally the proximity sensor only comes to an active state when a user receives a phone call or when a phone call is made by the user).

The energy bug that was leading the device to a high power consuming state was that in iOS 5.0.1, this "Raise to Speak" option had no effect to control the functionality of proximity sensor [20], [15], [9]. It is a known fact that with the proximity sensor being in an active state (Raise to Speak option) to activate Siri, the device will consume more battery power. But the desired behaviour is that when this option is turned off, the device should go back to its normal (power consuming) state. That was not the case in iPhone 4S running iOS 5.0.1 where whether the Raise to Speak option was turned on or off, the proximity sensor was always in an active state and as a result the device was constantly in a high power consuming state, draining the battery [10], [14], [8].

4.2.2 Low Power Consuming State

As the name suggests, this state of the device affects the performance and functionality of the device and leaves the device in a state where it consumes less power than it was consuming before a particular process started. An energy bug that led the device to this state was detected on Samsung Galaxy Note 3 running Jelly Bean (Android OS v4.3). The issue was again with the malfunctioning of proximity sensor. But unlike an iPhone, in this case the proximity sensor malfunction was leading the device to a low power consuming state.

The normal operation of proximity sensor is to turn off the screen when a user answers a phone call and then turn on the screen as soon as the call ends. The energy bug was that whenever user received a phone call, the screen would turn off but then would not turn on unless the power button was pressed. Due to presence of this energy bug, the smartphone's screen would remain off after a phone call consequently affecting the functionality of device and leaving the device in a low power consuming state [25], [28], [22].

4.3 Energy Bug Across Different OS Versions

Energy bug is an important phenomenon and it requires extensive research in this domain to make the new generation of smartphones energy efficient. Motivated by our findings (section 4.1), we researched energy bugs across different *OS versions*. It is a known fact that with each OS update either there are new features being added to the operating system or there are bug fixes or it is combination of both. But the important question here is:

what happens to the power consumption of the device?

In order to investigate this, we used two identical new smartphones and made sure that two different OS versions were available (one with a new OS version and one with a relatively old OS version). The next step was to devise an experiment. In the following, we will outline the steps for the experiment ComputeEnergy():

Begin Experiment ComputeEnergy()

Step 1: Open the built-in browser of the smartphone.

Step 2: Goto the following URL: `http://youtube.com` and select the full website option.

Step 3: Play a video for 6 minutes and then close the browser.

End

	Old OS Version - Energy (J)	New OS Version - Energy (J)	Difference
Experiment 1	569.99	473.88	18.41 %
Experiment 2	564.02	478.91	16.32 %
Experiment 3	564.63	470.18	18.25 %
Experiment 4	568.05	478.04	17.1 %

Table 4.1: Energy consumption over different OS versions.

To have a consistency in our experiments, we performed the same experiment on two same devices with same configurations but different *OS versions*. We performed the same experiment multiple times to verify our results. We also used the same video for all the experiments. In order to ensure that the measured energy is the real cost of playing that video in the browser, we made sure to clear the cache of browser each time the experiment was performed so that all the web elements will be requested fresh from the web server [32].

Table 4.1 shows the results of four iterations of same experiment performed on two same smartphones with different OS versions following the experiment outlined above. In order to compute the energy consumption, we have used the same experimental setup as described in section 3.4.

The results show that there is a significant difference between the energy consumption of the devices which indicates the presence of energy bug across different OS versions.

Following figures 4.2, 4.3, 4.4 and 4.5 show the actual results of all the experiments performed to compute energy consumption across different OS versions.

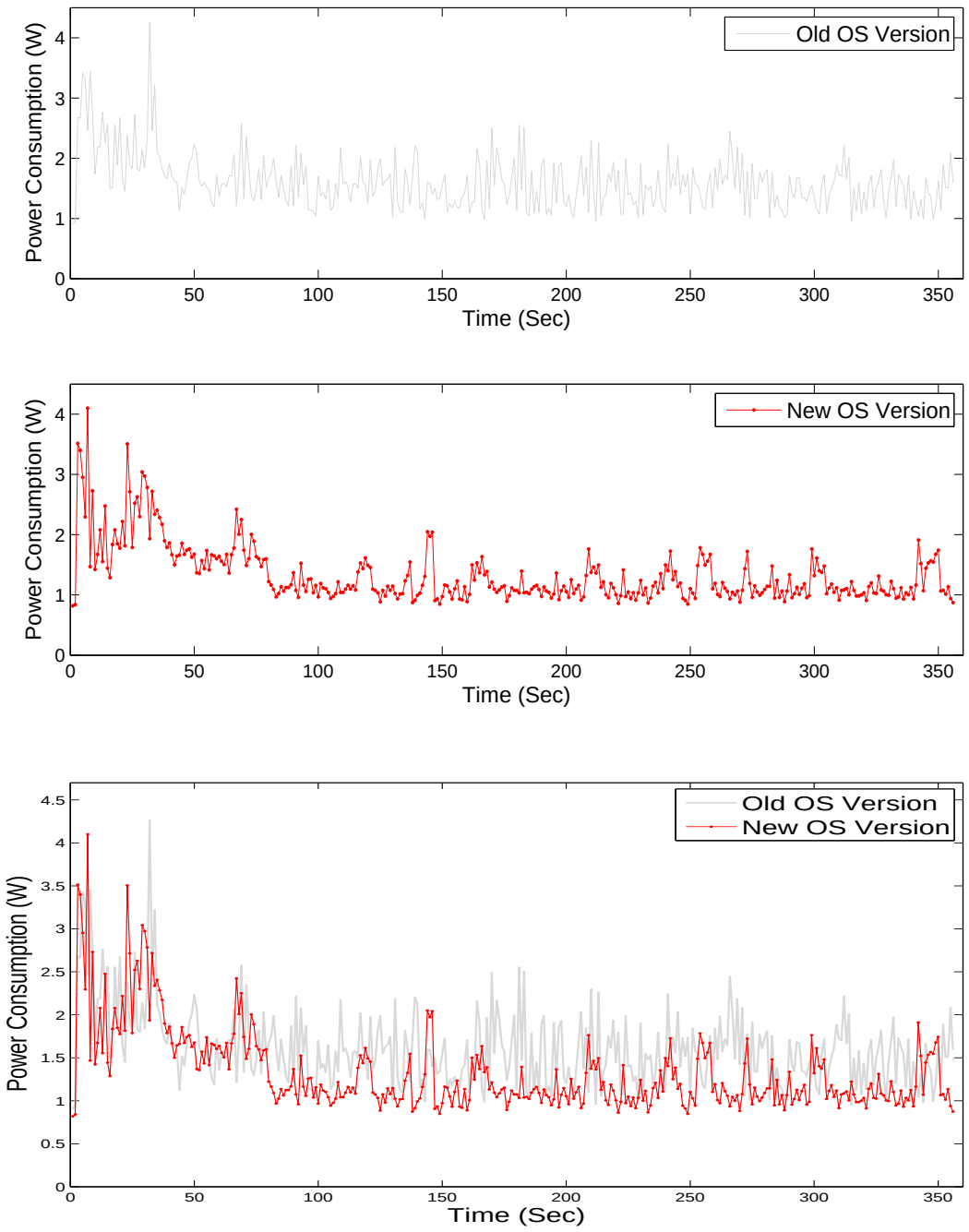


Figure 4.2: Experiment 1 - Difference in energy consumption across different OS versions.

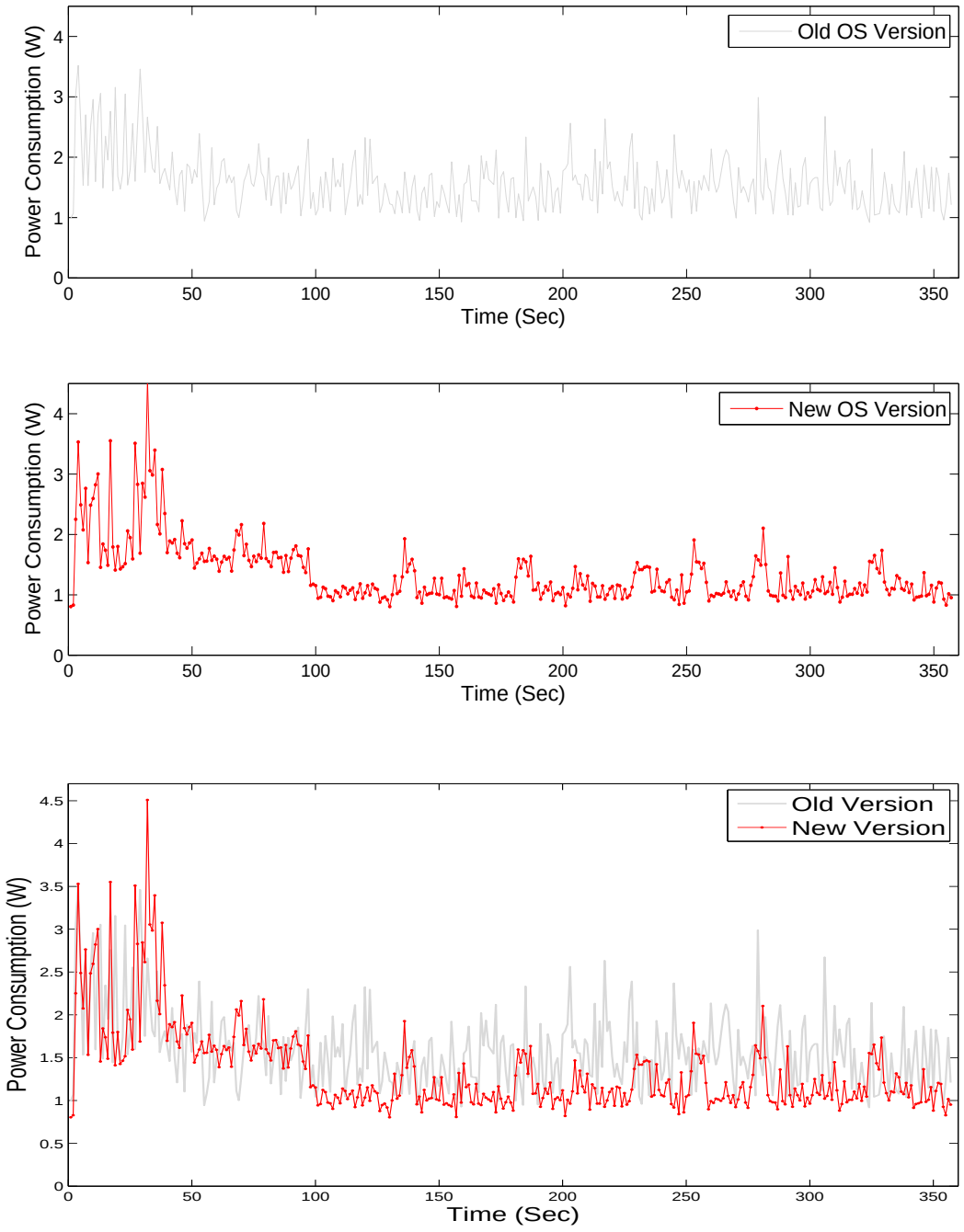


Figure 4.3: Experiment 2 - Difference in energy consumption across different OS versions.

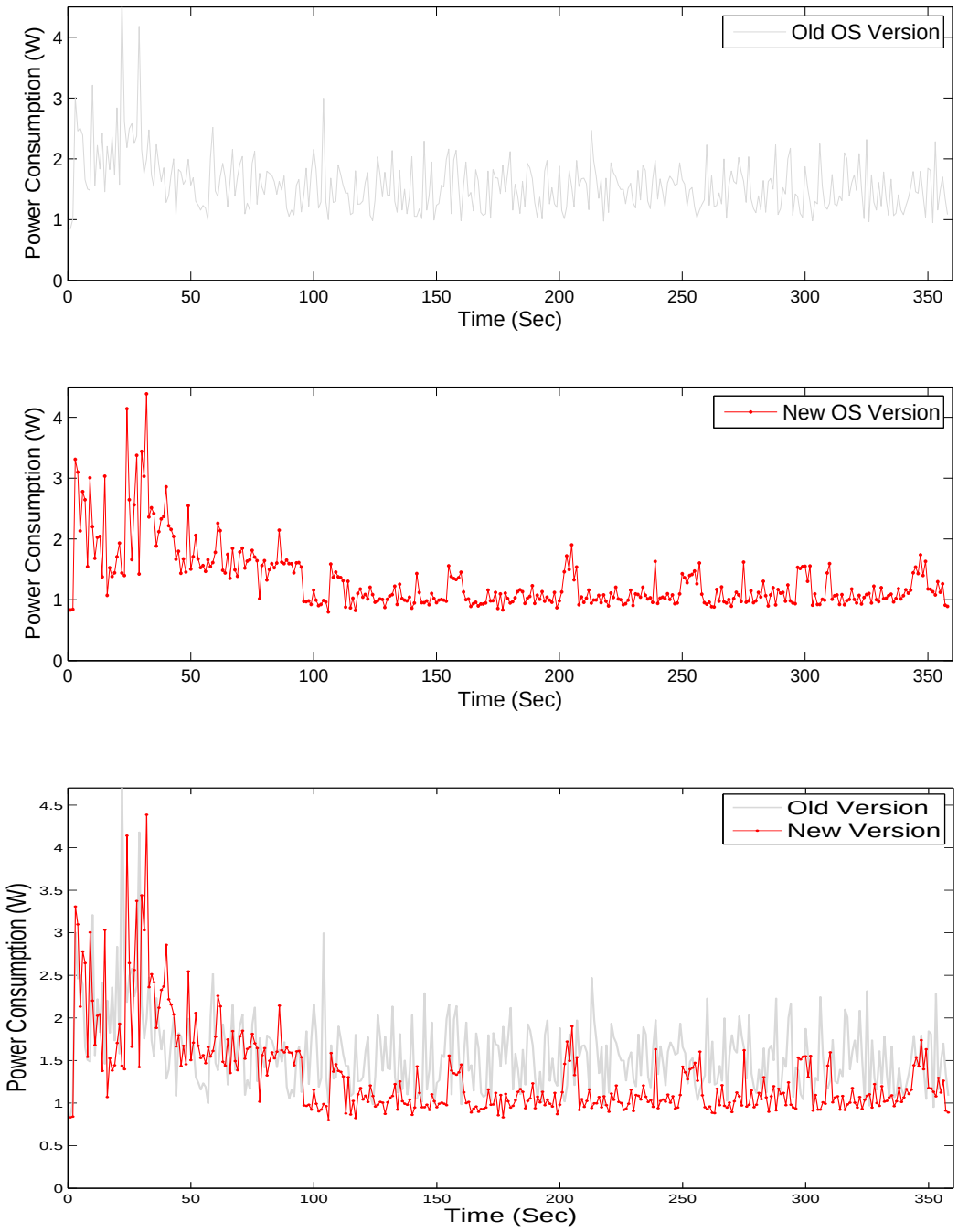


Figure 4.4: Experiment 3 - Difference in energy consumption across different OS versions.

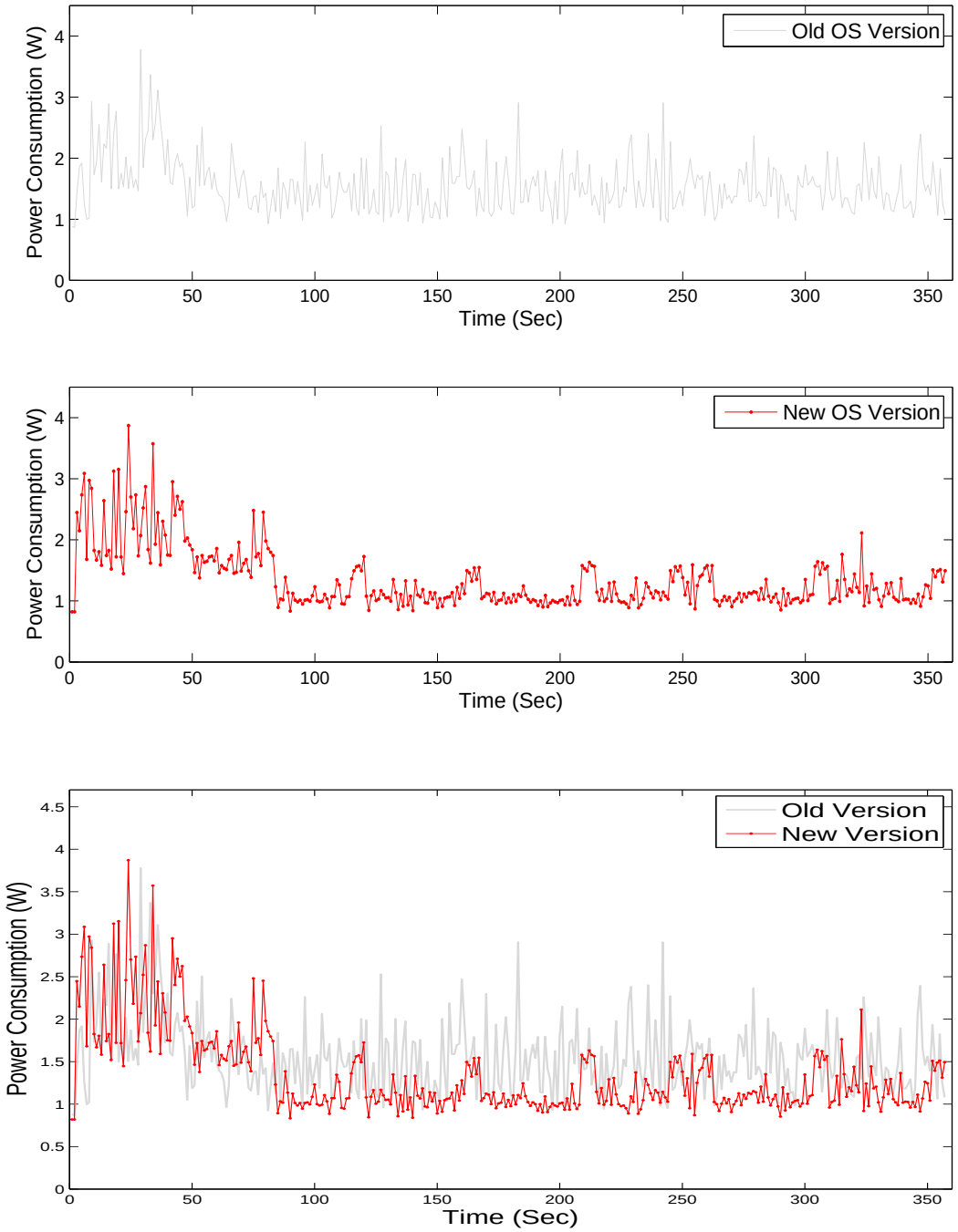


Figure 4.5: Experiment 4 - Difference in energy consumption across different OS versions.

4.4 Energy Bug Across Different Applications with Same Functionality

As discussed earlier that the smartphone applications should be energy efficient but usually there are several applications available that provide the same functionality. How to select between them ? What about the energy consumption ? In the following, we have investigated the same questions:

what happens to the power consumption of the device when we use different applications with same functionality on our smartphones?

In order to investigate this issue, we used a new smartphone and made sure that four different browser applications were available for it. The next step was to devise an experiment. We have used the same experiment as discussed in section 4.3. In the following, we will outline the steps for the experiment ComputeEnergy():

Begin Experiment ComputeEnergy()

Step 1: Open the browser application in smartphone.

Step 2: Goto the following URL: `http://youtube.com` and select the full website option.

Step 3: Play a video for 6 minutes and then close the browser.

End

To have a consistency in our experiments, we performed the same experiment using the same device with same configurations but different *browser applications*. We performed the same experiment multiple times to verify our results. We also used the same video for all the experiments. In order to ensure that the measured energy is the real cost of playing that video in the browser, we made sure to clear the cache of browser each time the experiment was performed so that all the web elements will be requested fresh from the web server [32].

In order to compute the energy consumption, we have used the same experimental setup as described in section 3.4. Table 4.2 show four iterations of same experiment performed

	Browser 1 - Energy (J)	Browser 2 - Energy (J)	Browser 3 - Energy (J)	Browser 4 - Energy (J)
Experiment 1	582.73	473.18	569.90	505.48
Experiment 2	560.59	462.56	548.93	491.89
Experiment 3	555.67	462.29	555.86	490.02
Experiment 4	559.47	464.80	562.87	490.56

Table 4.2: Energy consumption by different different browsers.

for energy consumption on same smartphone with four different browsing applications following the experiment outlined above.

Table 4.3 and table 4.4 show the comparison and difference in energy consumption of four different browsers. The results show that there is a significant difference between the energy consumption which indicates the presence of energy bug and validates our view point.

	Difference in Browser 1 and Browser 2 - Energy (J)	Difference in Browser 1 and Browser 4 - Energy (J)
Experiment 1	20.75 %	14.19 %
Experiment 2	19.10 %	13.05 %
Experiment 3	18.30 %	12.50 %
Experiment 4	18.40 %	13.80 %

Table 4.3: Difference in energy consumption by different browsers.

	Difference in Browser 3 and Browser 2 - Energy (J)	Difference in Browser 3 and Browser 4 - Energy (J)
Experiment 1	18.31 %	11.83 %
Experiment 2	16.80 %	10.80 %
Experiment 3	18.30 %	12.50 %
Experiment 4	19.10 %	13.0 %

Table 4.4: Difference in energy consumption by different browsers.

Figure 4.6 show the results of all four experiments performed to compute energy consumption across different applications in a graphical form.

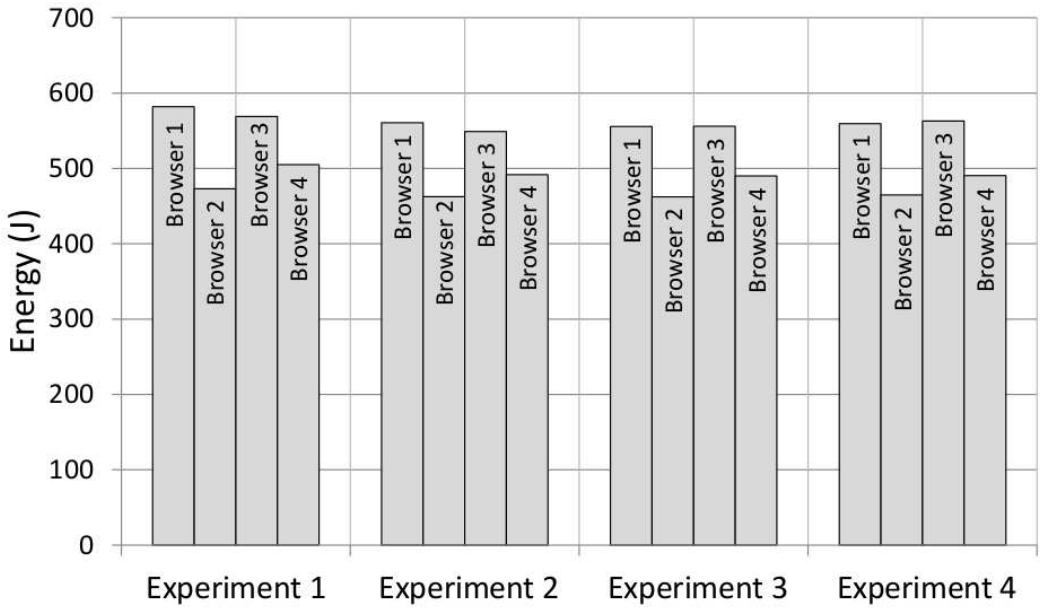


Figure 4.6: Energy consumption across different applications with same functionality.

Our experiments for two different scenarios, one with two different versions of operating systems on same smartphone and the other with different application (providing the same functionality) on same operating system and same smartphone, reveal the following interesting observations:

- Different versions of operating systems have different energy consumption patterns.
- Difference in energy consumption across operating system versions shows the presence of energy bugs.
- Different applications (although having same functionality) have different energy consumption patterns.
- Difference in energy consumption shows that there are energy issues in the applications and indicates the presence of energy bugs.

Chapter 5

Conclusion

Smartphones with all the state of the art technology, latest hardware components, new software and exciting applications are still constrained to their battery. There has been a tremendous growth and advancement in smartphone's software and hardware but in comparison the progress in smartphone battery technologies is very slow. The smartphone applications act as primary source for battery consumption as they utilize the smartphone's resources to complete different tasks. Therefore there is a need for applications to be energy efficient in their operations. The energy issues in smartphones have been a source of user frustration and a concern for the smartphone manufacturers. Therefore in Chapter 2, we have provided a literature survey of these energy issues in smartphones. We have discussed energy bugs and different reasons that could lead the smartphones to these energy bugs. We have also provided taxonomy of these energy bugs. At the end we have discussed related research in this domain.

In Chapter 3, we have discussed energy performance testing methodologies and presented a methodology to reduce the configuration parameters of smartphones. We also verified our methodology by measurements and experiments on four different smartphones. Our results showed that this methodology will help developers to efficiently test their applications for energy inefficiency.

Then in Chapter 4, we have focused on energy bugs and energy issues in smartphones. We defined energy bug in context of software testing and presented several scenarios and cases. We investigated the energy bugs across different versions of operating system and across different applications with same functionality. We also performed experiments on

latest smartphones and validated our definition with measurements and experiments. Our results showed that there are energy bug across different versions of operating systems in smartphones and also across different applications with same functionality.

In the future, we will work to improve the testing methodologies so that the number of test cases for energy performance testing can be further reduced. With new smartphones, new operating systems and new applications the energy bug issue will be of critical importance. So we will work on mechanisms and tools to detect these energy bugs (by using our definition as a reference). We will also work on solution strategies to avoid and overcome these energy bugs.

References

[1] Proximity sensor. http://www.wikipedia.org/wiki/Proximity_sensor#cite_note-Proximity_sensor_on_Android_smartphones-1.

[2] Proximity sensor on android gingerbread. http://www.thecodeartist.blogspot.ca/2011/01/proximity-sensor-on-android-gingerbread.html.

[3] Sd card corruption. http://code.google.com/p/android/issues/detail?id=2500.

[4] Setcpu for root users. http://www.setcpu.com/.

[5] Android phones more prone to hardware problems. http://www.pcmag.com/article2/0,2817,2387493,00.asp, June 2011.

[6] Apple confirms battery life problems are ios 5 related. http://www.wired.com/2011/11/iphone-4s-battery-issues/, November 2011.

[7] Apple press release. https://www.apple.com/pr/library/2011/10/04Apple-Launches-iPhone-4S-iOS-5-iCloud.html, October 2011.

[8] iphone 4 ambient light sensor problem or software bug? https://discussions.apple.com/thread/3055905, May 2011.

[9] iphone 4s battery problem maybe found. https://discussions.apple.com/thread/3507356?tstart=0, November 2011.

[10] Possible issue leading to iphone 4s battery drain. https://discussions.apple.com/thread/3491965, November 2011.

[11] What will the smartphone market look like in 2015? http://www.mashable.com/2011/04/07/what-will-the-smartphone-market-look-like-in-2015/, April 2011.

[12] [bug] ios 6 - battery drain : Siri raise to speak/proximity sensor always on/flashing when screen active. https://discussions.apple.com/thread/4312957?start=30&tstart=0, October 2012.

[13] ios 6 iphone 4s battery drain. https://discussions.apple.com/thread/4310494?tstart=0, September 2012.

[14] iphone 4s- infrared sensor still stays on constantly with 5.1 update. https://discussions.apple.com/thread/3932190, May 2012.

[15] Iphone 5 proximity sensor problem? https://discussions.apple.com/thread/4590331, December 2012.

[16] Smartphone sales overtake pcs. http://mashable.com/2012/02/03/smartphone-sales-overtake-pcs/, February 2012.

[17] Smartphone sales will dwarf pc sales. http://www.businessinsider.com/smartphone-sales-forecast-2012-2?nr_email_referer=1, February 2012.

[18] Why can't i turn off proximity sensor? - apple support communities. https://discussions.apple.com/message/21261383#21261383, October 2012.

[19] Are smartphones getting larger because they have to? http://www.extremetech.com/computing/163636-the-ever-expanding-smartphone-or-why-are-phablets-so-darn-popular, August 2013.

[20] Battery problem. https://discussions.apple.com/thread/4812604, February 2013.

[21] iphone 4s battery headaches persist for ios 6.1.1 users. http://www.wired.com/2013/02/iphone-4s-battery-problems/, February 2013.

[22] Note 3 proximity sensor issue during calls. http://forums.androidcentral.com/verizon-samsung-galaxy-note-3/323701-note-3-proximity-sensor-issue-during-calls.html, October 2013.

[23] Android statistics: Number of android applications. http://www.appbrain.com/stats/number-of-android-apps, August 2014.

[24] Chart: Global web access shifts to smartphones. http://www.thenextweb.com/shareables/2014/08/19/watch-world-move-towards-smartphones-one-simple-chart/, August 2014.

[25] Faulty proximity sensor in galaxy note 3. https://community.verizonwireless.com/thread/814958, January 2014.

[26] Global mobile statistics 2014: Mobile subscribers; handset market share; mobile operators. http://www.mobithinking.com/mobile-marketing-tools/latest-mobile-stats/a#topmobilemarkets, May 2014.

[27] itunes app store now has 1.2 million apps, has seen 75 billion downloads to date. http://www.techcrunch.com/2014/06/02/itunes-app-store-now-has-1-2-million-apps-has-seen-75-billion-downloads-to-date, June 2014.

[28] Proximity sensor issues and signal issues. https://community.verizonwireless.com/thread/814442, January 2014.

[29] Ahmed Abdelmotalib and Zhibo Wu. Power management techniques in smartphones operating systems. *IJCSI International Journal of Computer Science Issues*, 9(3), 2012.

[30] A. Abogharaf, R. Palit, K. Naik, and A. Singh. A methodology for energy performance testing of smartphone applications. In *7th Int. Workshop on Automation of Software Test*, pages 110–116, 2012.

[31] Abdulhakim Abogharaf. A client-centric data streaming technique for smartphones: An energy evaluation. Masters thesis, ECE Department, University of Waterloo, Ontario, Canada, 2013.

[32] Abdurhman Albasir. An evaluation of smartphone resources used by web advertisements. Masters thesis, ECE Department, University of Waterloo, Ontario, Canada, 2013.

[33] R. Arya, R. Palit, and K. Naik. A methodology for selecting experiments to measure energy costs in smartphones. In *7th Int. Wireless Comm. and Mobile Comp. Conf. (IWCMC)*, pages 2087 –2092, July 2011.

[34] James Bach and Patrick J. Schroeder. Pairwise testing: A best practice that isn't. In *22nd Annual Pacific Northwest Software Quality Conference*, pages 180–196, 2004.

[35] N. Balasubramanian and et al. Energy consumption in mobile phones: a measurement study. In *Proc. of the 9th ACM SIGCOMM Internet Measurement Conference*, IMC '09, pages 280–293, 2009.

[36] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 280–293. ACM, 2009.

[37] Kyoung Youn Cho, S. Mitra, and E.J. McCluskey. Gate exhaustive testing. *Test Conference, International*, pages pp. 771–777, 2005.

[38] D. Cohen and et al. The aetg system: An approach to testing based on combinatorial design. *IEEE Trans. on Software Engineering*, 23:437–444, 1997.

[39] Jacek Czerwonka. Pairwise testing in real world. In *Proceedings of 24th Pacific Northwest Software Quality Conference*, 2006.

[40] J. Bernal et al. Towards an efficient context-aware system: Problems and suggestions to reduce energy consumption in mobile devices. In *9th Int. Conf. ICMB-GMR*, pages 510 –514, 2010.

[41] Zachary C Fluhr and Eric Nussbaum. Switching plan for a cellular mobile telephone system. *Vehicular Technology, IEEE Transactions on*, 22(4):197–202, 1973.

[42] Shuai Hao, Ding Li, William GJ Halfond, and Ramesh Govindan. Estimating mobile application energy consumption using program analysis. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 92–101. IEEE, 2013.

[43] J. Huang, Q. Xu, and et al. Anatomizing application performance differences on smartphones. In *8th*, MobiSys '10, pages 165–178, 2010.

[44] International Telecommunication Union (ITU). The world in 2014: Ict facts and figures. April 2014.

[45] Abhilash Jindal, Abhinav Pathak, Y Charlie Hu, and Samuel Midkiff. Hypnos: understanding and treating sleep conflicts in smartphones. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 253–266. ACM, 2013.

[46] Abhilash Jindal, Abhinav Pathak, Y Charlie Hu, and Samuel Midkiff. On death, taxes, and sleep disorder bugs in smartphones. In *Proceedings of the Workshop on Power-Aware Computing and Systems*, page 1. ACM, 2013.

[47] J.-M. Kang and et al. User-centric prediction for battery lifetime of mobile devices. In *11th Asia-Pacific Symposium on Network Operations and Management*, pages 531–534, Berlin, Heidelberg, 2008. Springer-Verlag.

[48] Youhuizi Li, Hui Chen, and Weisong Shi. Acm hotmobile 2013 poster: Bugu: an application level power profiler and analyzer for mobile devices. *ACM SIGMOBILE Mobile Computing and Communications Review*, 17(3):27–28, 2013.

[49] Yepang Liu, Chang Xu, and SC Cheung. Where has my battery gone? finding sensor related energy black holes in smartphone applications. In *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on*, pages 2–10. IEEE, 2013.

[50] Xiao Ma, Peng Huang, Xinxin Jin, Pei Wang, Soyeon Park, Dongcai Shen, Yuanyuan Zhou, Lawrence K Saul, and Geoffrey M Voelker. edoctor: Automatically diagnosing abnormal battery drain issues on smartphones. In *NSDI*, pages 57–70, 2013.

[51] D. Marinov and et al. An evaluation of exhaustive testing for data structures. Technical report, MIT Computer Science and AI Lab. Report MIT -LCS-TR-921, 2003.

[52] Robert N. Mayo and P. Ranganathan. Energy consumption in mobile devices: Why future systems need requirements-aware energy scale-down. In *PowerAware Computer Systems*, pages 26–40, 2003.

[53] H. Muccini, A. Di Francesco, and P. Esposito. Software testing of mobile applications: Challenges and future directions. In *7th Int. Workshop on Automation of Software Test*, pages 29–35, 2012.

[54] Henry Muccini, Antonio Di Francesco, and Patrizio Esposito. Software testing of mobile applications: Challenges and future research directions. In *Automation of Software Test (AST), 2012 7th International Workshop on*, pages 29–35. IEEE, 2012.

[55] K. Naik and P. Tripathy. *Software Testing and Quality Assurance: Theory and Practice*. John Wiley & Sons, first edition, 2008.

[56] Kshirasagar Naik, Yasir Ali, Veluppillai Mahinthan, Ajit Singh, and Abdulhakim Abogharaf. Categorizing configuration parameters of smartphones for energy performance testing. In *Proceedings of the 9th International Workshop on Automation of Software Test*, pages 15–21. ACM, 2014.

[57] Shin Nakajima. Model-based power consumption analysis of smartphone applications. In *ACESMB@ MoDELS*, 2013.

[58] Adam J Oliner, Anand Iyer, Eemil Lagerspetz, Sasu Tarkoma, and Ion Stoica. Collaborative energy debugging for mobile devices. In *Proceedings of the Eighth USENIX conference on Hot Topics in System Dependability*, pages 6–6. USENIX Association, 2012.

[59] R. Palit, R. Arya, K. Naik, and A. Singh. Selection and execution of user level test cases for energy cost evaluation of smartphones. In *6th Int. Workshop on Automation of Software Test*, pages 84–90, 2011.

[60] Abhinav Pathak. *Energy debugging in smartphones*. PhD thesis, ECE Department, Purdue University, Indiana, USA, 2012.

[61] Abhinav Pathak, Y Charlie Hu, and Ming Zhang. Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, page 5. ACM, 2011.

[62] Abhinav Pathak, Y Charlie Hu, and Ming Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 29–42. ACM, 2012.

[63] Abhinav Pathak, Y Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the sixth conference on Computer systems*, pages 153–168. ACM, 2011.

[64] Abhinav Pathak, Abhilash Jindal, Y Charlie Hu, and Samuel P Midkiff. What is keeping my phone awake?: characterizing and detecting no-sleep energy bugs in smartphone apps. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 267–280. ACM, 2012.

[65] Robert A Powers. Batteries for low power electronics. *Proceedings of the IEEE*, 83(4):687–693, 1995.

[66] Suresh Siddha, Venkatesh Pallipadi, and AVD Ven. Getting maximum mileage out of tickless. In *Linux Symposium*, volume 2, pages 201–207. Citeseer, 2007.

[67] Narseo Vallina-Rodriguez and Jon Crowcroft. Energy management techniques in modern mobile handsets. *Communications Surveys & Tutorials, IEEE*, 15(1):179–198, 2013.

[68] Claas Wilke, Sebastian Richly, Sebastian Gotz, Christian Piechnick, and Uwe Aßmann. Energy consumption and efficiency in mobile applications: A user feedback study. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pages 134–141. IEEE, 2013.

[69] Jack Zhang, Ayemi Musa, and Wei Le. A comparison of energy bugs for smartphone platforms. In *Engineering of Mobile-Enabled Systems (MOBS), 2013 1st International Workshop on the*, pages 25–30. IEEE, 2013.

[70] Lide Zhang, Mark S Gordon, Robert P Dick, Z Morley Mao, Peter Dinda, and Lei Yang. Adel: An automatic detector of energy leaks for smartphone applications. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 363–372. ACM, 2012.