Modeling and Evaluating Energy Performance of Smartphones

by

Rajesh Palit

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2011

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.

Rajesh Palit

Abstract

With advances in hardware miniaturization and wireless communication technologies even small portable wireless devices have much communication bandwidth and computing power. These devices include smartphones, tablet computers, and personal digital assistants. Users of these devices expect to run software applications that they usually have on their desktop computers as well as the new applications that are being developed for mobile devices. Web browsing, social networking, gaming, online multimedia playing, global positioning system based navigation, and accessing emails are examples of a few popular applications. Mobile versions of thousands of desktop applications are already available in mobile application markets, and consequently, the expected operational time of smartphones is rising rapidly.

At the same time, the complexity of these applications is growing in terms of computation and communication needs, and there is a growing demand for energy in smartphones. However, unlike the exponential growth in computing and communication technologies, in terms of speed and packaging density, battery technology has not kept pace with the rapidly growing energy demand of these devices. Therefore, designers are faced with the need to enhance the battery life of smartphones. Knowledge of how energy is used and lost in the system components of the devices is vital to this end. With this view, we focus on modeling and evaluating the energy performance of smartphones in this thesis. We also propose techniques for enhancing the energy efficiency and functionality of smartphones.

The detailed contributions of the thesis are as follows: (*i*) we present a finite state machine based model to estimate the energy cost of an application running on a smartphone, and provide practical approaches to extract model parameters; (*ii*) the concept of *energy cost profile* is introduced to assess the impact of design decisions on energy cost at an early stage of software design; (*iii*) a generic architecture is proposed and implemented for enhancing the capabilities of smartphones by sharing resources; (*iv*) we have analyzed the Internet traffic of smartphones to observe the energy saving potentials, and have studied the implications on the existing energy saving techniques; and finally, (*v*) we have provided a methodology to select user level test cases for performing energy cost evaluation of applications. All of our concepts and proposed methodology have been validated with extensive measurements on a real test bench.

Our work contributes to both theoretical understanding of energy efficiency of software applications and practical methodologies for evaluating energy efficiency. In summary, the results of this work can be used by application developers to make implementation level decisions that affect the energy efficiency of software applications on smartphones. In addition, this work leads to the design and implementation of energy efficient smartphones.

Acknowledgements

First and foremost, I express my profound indebtedness to my supervisor, *Dr. Sagar Naik*, who has supported me throughout my thesis with his patience and constant guidance while allowing me to work in my own way. This thesis would not have been completed without his utmost support and encouragement. I offer my sincere gratitude to my co-supervisor, *Dr. Ajit Singh*, for his advice, support, and encouragement. His business experience and intuitions enriched my growth as a researcher. I take this opportunity to thank all of the committee members of this thesis for their valuable time, support, and advice.

My deepest gratitude goes to my parents, *Mira Palit* and *Nani Gopal Palit*, who allowed me to pursue my degree, though, without me, they became very lonely and helpless at times. I would like to thank my uncle, my sisters and brothers-in-law for the constant support they provided to my parents. I would especially like to mention the name *Sukumar Chowdhury*, my brother-in-law whose enormous support has enabled me to come to this stage of my life. I am indebted to him more than he knows.

I cannot ignore the inspiration that I have constantly received from my son, *Roshan*, who was born in the middle of my PhD tenure. Although my life became difficult, he has helped me understand the true meaning of life and responsibility.

Finally, I like to thank my friends and individuals who helped me in any way at Waterloo to complete my degree. I would specially mention the names, *Michael* and *Sukanta*, who proof read three chapters of my thesis.

Dedication

To all of my teachers, especially *Apu Dey, Purnendu Bhattachariya,* and *Muhammad Yakub Ali*, whose teaching, love, and inspiration have been the great possession in my life.

Where The Mind is Without Fear

```
WHERE the mind is without fear and the head is held high
Where knowledge is free
Where the world has not been broken up into fragments
By narrow domestic walls
Where words come out from the depth of truth
Where tireless striving stretches its arms towards perfection
Where the clear stream of reason has not lost its way
Into the dreary desert sand of dead habit
Where the mind is led forward by thee
Into ever-widening thought and action
Into that heaven of freedom, my Father, let my country awake.
```

Rabindranath Tagore

Table of Contents

List of Tables

List of Figures

List of Acronyms

3G	Third Generation Mobile Telecommunications
AES	Advanced Encryption Standard
AP	Access Point (WLAN)
BER	Bit Error Rate
CIT	Combinatorial Interaction Testing
CPU	Central Processing Unit
DES	Data Encryption Standard
EDGE	Enhanced Data rates for GSM Evolution
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
FTP	File Transfer Protocol
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HSDPA	High-Speed Downlink Packet Access
HSUPA	High-Speed Uplink Packet Access
HTTP	HyperText Transfer Protocol
JVM	Java Virtual Machine

MAC Medium Access Control

MGF Moment Generating Function

MTU Maximum Transmission Unit

NFC Near Field Communication

NIC Network Interface Card

OS Operating System

PDA Personal Digital Assistant

PSM Power-Saving Mode

QoS Quality of Service

RAM Random Access Memory

SFTP Secured File Transfer Protocol

SNR Signal-to-Noise Ratio

SoC System-On-Chip

SoC State of Charge

TCP Transport Control Protocol

UCCI Universal Computing and Communication Interface

UDP User Datagram Protocol

UMTS Universal Mobile Telecommunications System

USB Universal Serial Bus

VoIP Voice over Internet Protocol

WLAN Wireless Local Area Network

WPAN Wireless Personal Area Network

WWAN Wireless Wide Area Network

Chapter 1

Introduction

Since the invention of the transistor, enormous progress and development in the field of solid state physics has continually reduced the size of the semi conductor devices. The capability, or, the speed of electronic devices has increased exponentially while their size and cost has decreased to the same extent. Over time, these technological advancements have allowed digital communication to evolve, and in the 1980's the cell phone emerged. Due to their inherent support for portability and mobility, hand-held devices connected to wireless networks have become widely popular.

In the early years, mobile phones were used only for voice calls and short message services (SMS). The users could also avail push/pull services to some extent with limited data service. Later on, with the overwhelming penetration of Internet into society, there was a huge demand for full-fledged data service in cellular networks. To meet the user demands, newer standards and technologies for wireless networks and hand-held devices have been brought to market, and are now widely accepted by end users all over the world. The success and growth of Internet based services have profoundly impacted global economics, as well as the ways in which people communicate and live their lives [43]. This trend is ongoing and it seems that smartphones will become as prevalent in our daily lives as electricity and motor vehicles.

1.1 Smartphone and its Components

There has been a rapid evolution in the industry of handheld device over the last couple of years. These devices include smartphones, personal digital assistants (PDA), and tablet

computers. By the end of 2010, over 75% of the world's population had subscribed to mobile phones and about 55% of them were smartphone users in developed countries [67].

Smartphones can be defined as small computers with high speed wireless communication interfaces. The smartphone market is one of the most competitive markets for semiconductor vendors. At the low end, cost pressures are pushing them to integrate their hardware components into single-chip devices. At the high end, they are required to keep up with the newest air interfaces, such as HSDPA and HSUPA, while adding TV-quality video, 3D graphics, and other multimedia functions to the processors. They need to find the right balance of cost and multimedia performance to meet the demands of carriers and end users.

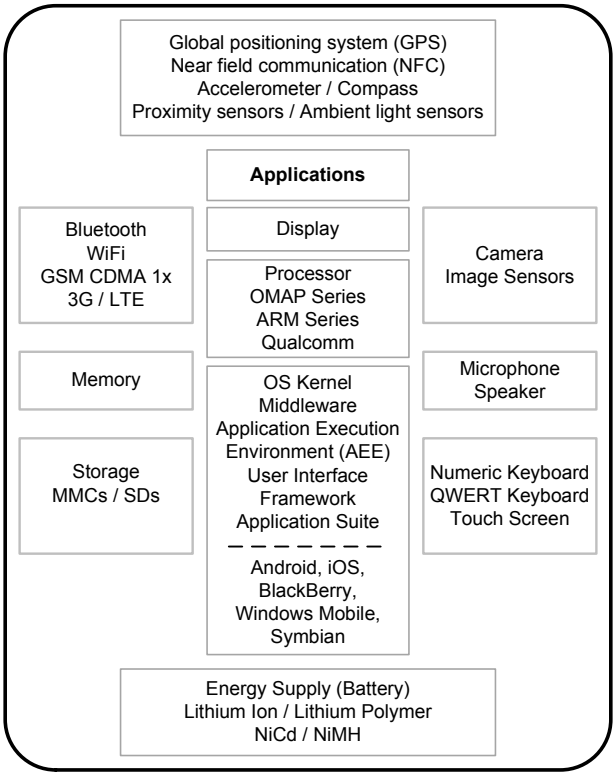


Figure 1.1: Components of a smartphone.

Figure 1.1 depicts the components of a state-of-the-art smartphone. The devices typically have low-power-consuming RISC (Reduced Instruction Set Computer) microprocessors manufactured by companies like Texus Instrument (TI) and ARM Qualcomm. They

generally have random access memory (RAM) of around 256 MB to 1GB and several gigabytes of removable flash memory. The power supply is usually equipped with a 3.7 volt lithium-ion battery ranging from 800 to 1500 milliampere hour (mAh). They have Half VGA or Quarter VGA color displays most often with touch screen capabilities. Smartphones also have cameras, GPS (Global Positioning System) receivers, and other sensors mentioned in the figure. On the software side, there are a few main streams of Operating

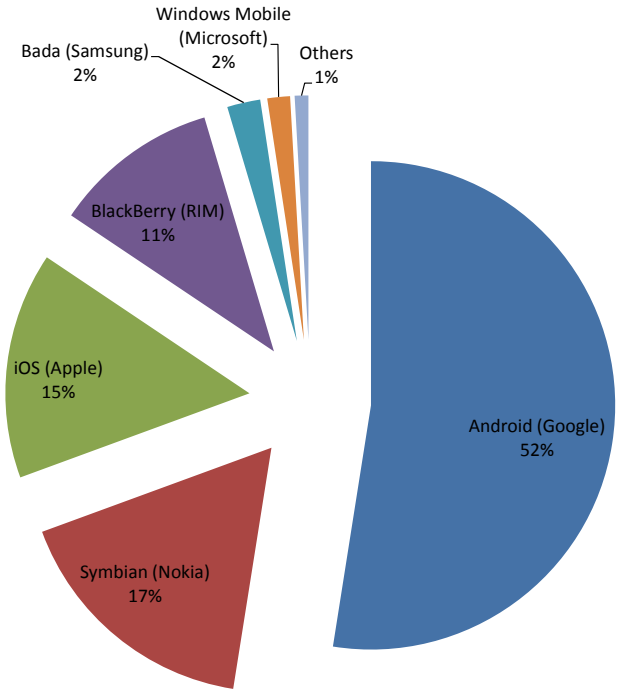


Figure 1.2: Market share of smartphone OS (Gartner, November 2011).

Systems (OS) used in the mobile phones. As shown in Fig. 1.2, about 52% of the smartphones in the market use Android OS from Google. Symbian OS by Nokia shares 17% of OS market. The iOS used in iPhone shares 15% of the market. BlackBerry uses its own proprietary OS, which covers 11% of the market. Windows Mobile OS captures 2% market shares. There also exist Palm garnet OS and some linux based OS like LiMo, Mobilinux in the market.

Smartphone applications, or mobile applications, colloquially referred to as *apps* come with the smartphone, or are downloaded by users from various mobile software distribution platforms, known as the *App Store or Market*. There are more than 325 thousand applications on Android market and more than half a million applications on Apple's *App Store*.

The number of downloads reached more than 10 billion in app store at the end of 2011, and it is expected that by the year 2015 the total *app* downloads per year will reach close to 50 billion. The categories of mobile applications include (*i*) Games; (*ii*) Social networking; (*iii*) News and weather forecast; (*iv*) Maps, navigation, search, and location based service; (*v*) Online music and video; (*vi*) Entertainment and food; (*vii*) Sports; (*viii*) Banking, finance, and mobile payment; (*ix*) Shopping; (*x*) Productivity; and (*xi*) Travel, lifestyle, and mobile health monitoring.

1.2 Resource Constraints

Among the applications on smartphones, Internet browsing, online video and music playing, gaming, social networking, news, weather, stock reports, global positioning system (GPS) aided maps, navigation, and searching are at the top of the charts [44, 93]. Uploading photos and videos directly to social networks and voice-over-IP (VoIP) clients are becoming increasingly popular. Moderate computing power, communication bandwidth, and above all, innovative development tools have enabled the creation of mobile versions of many popular desktop applications [125]. The complexity of these applications is growing in terms of computation and communication needs with increasing device functionality. As a consequence of this increased usability, the demand for increased operating time, or battery life of smartphones is rising rapidly.

Unlike other resources such as memory and processor, energy is an exhaustible resource. Energy cannot be reclaimed once it is spent [130]. Therefore, energy must be diligently used in handheld devices. The concern about energy availability has become more acute due to the volume of third-party applications available on the Internet. In addition, there has not been a satisfactory development in battery technology in terms of energy capacity. The battery capacity has not increased at the same pace as other components of handheld devices. The state-of-the-art mobile device from Google, *Nexus S* has a standby time of 18 days, whereas its talk-time is just 7 hours on 3G networks. Similarly, smartphones last only $2 - 3$ hours when online video is played. This dependence on battery energy puts a severe constraint on the availability of these devices. Therefore, devising energy management strategies have attracted the attention of hardware and software designers of smartphones. In this thesis, we focus on designing energy efficient software applications for smartphones.

1.3 Energy Management and Applications

The term *energy management* is used to mean a reduction in the overall energy consumption of a system through effective use of the system resources, and to keep the hardware components at low power consumption state as long as possible without sacrificing the system performance. Energy management can be performed at several levels in the system hierarchy [91]: the hardware component, the operating system (including protocols), the application and the user level.

Users generally lack knowledge about the power consumption of each component of the device, and are reluctant to make frequent energy management decisions. The hardware level approach may be thought to be appropriate as the hardware parts physically drain the battery energy. However, the hardware is there to fulfill the software needs, and software is the ultimate consumer of energy. Basically, hardware and software level approaches should not be considered mutually exclusive; instead they are supplementary in nature. Therefore, focus should be given on software level optimizations in the devices with hardware level energy saving features [34].

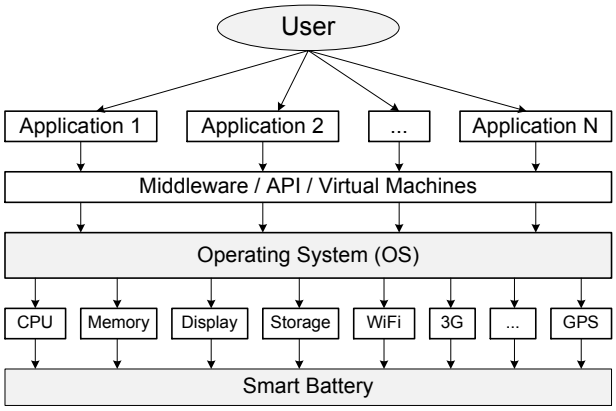


Figure 1.3: Relationships among user, applications, OS and battery.

Figure 1.3 depicts the relationship between the energy usage hierarchy of a smartphone. Users run applications on devices, and applications utilize hardware components such as the CPU and memory through middleware and OS interfaces to accomplish a task. The OS coordinates and schedules the access to hardware. However, it has no control over how efficiently the application uses hardware components. The OS also does not control how much energy a hardware component consumes to remain in a certain state. Therefore, energy efficient applications and hardware components with energy saving features play a

pivotal role in achieving overall energy efficiency of smartphones. The role of an energy-aware OS is to take advantage of the power saving features of software applications and hardware components.

1.4 Designing Energy Efficient Applications

The key challenges in designing energy efficient smartphone applications are as follows:

- **User Expectation:** Users of smartphones expect to run software applications that they usually have on their desktop computers as well as the mobile applications developed for mobile devices. The complexity, processing power, and communication requirements of these applications are high in comparison to smartphone resources, and they ultimately drain high amounts of battery energy. Most importantly, achieving energy efficiency without compromising user experience is a challenging task.
- **Application Development Environment:** As mentioned earlier, there are already more than a million applications in the market. A significant portion of these applications is designed and implemented on an *ad hoc* basis, and good software engineering guidelines are not followed. The development processes of the applications, OS and hardware are independent, and devices are diverse in terms of processor, memory, display types. Moreover, not all designers are aware of the energy saving potentials of OS, communication and other hardware components. High level energy cost information is not also available to them. All of these issues are challenges for developing energy efficient smartphones.
- **Lack of Performance Evaluation Tools:** Performance defects such as energy and delay performances are mostly invisible in desktop environments due to availability of abundant power, and high processing and communication bandwidth. However, they become visible in mobile platforms because of the lack of such resources. Due to the large number of configurations, there is a lack of consistent testing configurations across devices. The absence of a mobile test bench which is required for GPS, cellular and other location based testing, and difficulty in replicating the behavior of a real-battery are two more challenges in performance testing of smartphones.

1.5 Energy Management Strategies

We present a broad overview of existing energy management strategies for wireless portable devices in this chapter. Comprehensive details of software based energy saving methodologies for handheld devices can be found in [104]. Surveys of system-level dynamic power management techniques, energy efficient network protocols for wireless networks, and power-aware mobile multimedia applications can also be found in [11, 72, 172]. Discussions of related work have been given further with each solution approaches we have provided in the following chapters.

1.5.1 Smart Battery Aided Design

Smart batteries are re-chargeable batteries augmented with additional sensing and SoC computation logic. They play a key role in making applications adaptive to the amount of energy left in the battery. Power consumption pattern or load profile has a significant impact on battery lifetime [123]. Some profiles may let a battery to recover from time to time, whereas some other load profiles may not let the battery to recover. Therefore, task scheduling is an important system-level instrument which can adjust system load based on *state of charge* (SoC) information to prolong battery lifetime.

Chiasserini and Rao [24, 25] have mentioned that charge recovery takes place under bursty or pulsed discharge conditions. Hence, this phenomena can be exploited to enhance the *actual capacity* of the battery. They explore stochastic battery models to track charge recovery in conjunction with battery lifetime. Software designers need to be aware of the battery characteristics in order to be able to schedule the major energy consuming tasks in such a way that the resulting load profile leads to the longest battery lifetime.

1.5.2 Energy-Efficient GUI Design

Display is one of the largest energy consuming components in mobile devices. Display along with user interactions form graphical user interaction (GUI) sub-system. A number of techniques have been proposed to minimize the energy consumption of GUI sub-system, and a few categories are described below:

1. **Brightness Control of the Backlight:** These techniques, namely, dynamic luminance scaling (DLS) and concurrent brightness and contrast scaling (CBCS) keep the

perceived contrast of still images as close as possible to the original while achieving power reduction from the backlight system [19, 23].

2. **Frame Buffer Compression:** Frame buffer compression reduces the number of frame buffer accesses, and thus saves energy. Shim *et al.* [138] have shown that frame buffer compression reduces the display energy cost by $50 - 66\%$. When differential Huffman coding is used to compress the frames [139], the frame buffer activity is reduced by $52 - 90\%$.

3. **Dark Window Optimization:** An active window on display uses only about 60% of the total screen area, and the content of the screen can be displayed on much lower power displays with no apparent loss in visual quality. These are the two observations from a survey conducted by Iyer *et al.* [68]. They proposed several dark windows optimization techniques that allow the windowing environment to change the brightness and color of portions of the screen that are not of interest to the user.

Zhong and Jha [175] make the following recommendations to make GUI design energy-efficient: (*i*) Even an idle system consumes much energy so reducing the usage time for a task is an effective way of energy reduction. (*ii*) Do something while waiting for user input. (*iii*) Minimize screen changes as it costs energy to change even a single pixel. (*iv*) Since text input is much slower, avoid or minimize text input. (*v*) Features that do not accelerate usage should be avoided.

1.5.3 Energy-saving micro-Sleep Techniques

There is a significant difference in energy costs of the *sleep* and *idle* states of a processor [117]. Brakmo *et al.* [16] have introduced the concept of a $\mu Sleep$ state so that under certain conditions the processor can be put in the $\mu Sleep$ state instead of the *idle* state. If there is no process to run, the processor is moved into the OS *idle* state, and the scheduler makes a decision to move the processor to the $\mu Sleep$ state or to the processor *idle* state. A real-time clock alarm or an external event causes the processor to move back to the *running* state. Their experiments have shown that $\mu Sleep$ can reduce energy consumption by more than 60% when the experimental Itsy pocket PC is lightly loaded.

Liu *et al.* [89] proposed a micro power management (μPM) scheme that allows a WiFi transceiver to sleep for very short intervals, such as a few microseconds. It can be used to sleep even between two MAC frames. This client-based solution uses prediction to exploit short *idle* intervals, and any support from the AP is not needed. To control data loss, μPM takes advantages of the retransmission mechanism in 802.11.

1.5.4 Energy-efficient Communication Techniques

1. **MAC Level Techniques:** Around 90% of the energy can be saved by the Wireless NIC at the cost of increased delay in web-page downloads [79]. The authors of [79], Krashinsky *et al.* proposed a Bounded Slowdown protocol (BSD) to cope with the delay problem. The energy saving techniques proposed in [146, 162] use inactivity timers to decide when to switch off the Wireless NIC. Stemm and Katz [146] use knowledge of application behavior to compute the timeout duration, whereas, Yan *et al.* [162] estimated the inter-arrival times in a dynamic fashion to switch off the WNIC. Havinga and Smit [58] proposed a TDMA based energy efficient MAC protocol in which the AP schedules packet transmissions. The authors apply *mobile grouping strategy* that allows an STA to have a concatenated uplink and downlink phase, and the transceiver enters a low power mode for the remaining time of a frame. The *schedule traffic in bursts* strategy allows a transceiver to stay in a low-power mode for an extended period of time.

2. **Proxy Assisted Energy Saving:** A proxy works as an intermediary between mobile hosts and streaming servers. Applications running on the mobile user device send requests for streaming media objects to the local proxy. A proxy can enable a hand-held device to save energy in a number of ways, for example, by reducing the volume of contents to be downloaded by user devices, and/or by making data traffic bursty so that the intervals between bursts are long enough for devices to put their communication interfaces to sleep state. The detailed energy saving potentials of a Transforming Proxy, an HTTP-level Power Aware Web Proxy, a Power Aware Streaming Proxy, and a Streaming Audio Proxy can be found in [3, 127].

3. **Source-level Power Control:** Multimedia content servers apply a number of energy saving strategies. Examples of such server-based techniques are: (*i*) traffic shaping to enable a user device to put its communication interface to sleep state [1], and (*ii*) resolution control of video frames [84]. These techniques do not involve any proxy between the media servers and hand-held devices.

4. **TCP Based Energy Saving:** The Transmission Control Protocol (TCP) related energy efficiency of hand-held devices are categorized into two groups: (*i*) computational energy cost of executing the protocol details, such as congestion control, ACK generation, checksum computation, and round-trip time (RTT) [134], and (*ii*) TCP-assisted controlling the wireless NIC [2, 13], where TCP connection stays inactive for a while, thereby giving an opportunity to put the NIC in a low-power state.

1.5.5 Programming and Compilation Techniques

The potential for energy reduction through modification of software application and compilation has been studied since the mid 90s [49, 80, 105, 149, 159]. The concept of a set of base costs of instructions plays a key role in these studies. Tiwari et al. [149] have proposed a measurement-based instruction-level power analysis technique that makes it feasible to effectively analyze the energy consumption of software. A number of energy reduction techniques have been proposed which include: (*i*) reducing memory access, (*ii*) energy cost driven code generation, (*iii*) instruction re-ordering for low power consumption, and (*iv*) instruction packing and dual memory loads.

1.5.6 High-level Energy Management Techniques

A number of techniques have been studied at the application-level to achieve energy efficiency in hand-held devices. Those are data compression and download scheduling at the application-level and computation offloading. In the first approach, the decompression tasks on a client device are appropriately interleaved with downloading activities to maximize energy saving [161]. In the computation offloading approaches, some computation-intensive tasks are transferred from a user device to a server [52, 86, 152, 153, 158].

1.5.7 Integrated Power Management Techniques

Solutions for energy efficiency have been proposed at various computational levels, namely, cache and external memory access optimization, dynamic voltage scaling, dynamic power management for disk and network interfaces, efficient compilers, and application/middleware adaptations. The optimization techniques developed at each level have remained largely independent of the other abstraction levels, thereby not exploiting the opportunities for further improvements achievable through cross-level integration. The binding for this set of techniques is accomplished by system-level energy management approaches [34, 91, 119].

1.6 Problem Descriptions

In this thesis, a number of problems are addressed which share a common objective of achieving energy efficiency in smartphones. We cover two primary challenges regarding mobile application development process. The *first* challenge is to have high-level energy

cost information so that designers can take advantage of that during the design phase of energy efficient applications. A finite state machine based energy consumption model is proposed which is augmented with an existing application development process to achieve better energy efficiency. The *second* challenge is to have a consistent energy evaluation test bench for comparing test results of energy performance, we propose a measurement test bench that considers the values of different smartphone parameters during a test, and reduce the number of test configurations significantly.

In addition, we have addressed an upper-level (or, application level) energy management technique that enhances the functionality of smartphones by accessing resources on other devices. We have also investigated an energy efficient communication strategy that includes a data packet aggregation algorithm based on different burst parameters of smartphone Internet traffic. The background, motivation, related literature review, and detailed descriptions of each problem are presented in Chapter 2 through Chapter 5. These chapters are independent of each other. We provide a brief overview of the problems in the following paragraphs.

An in-depth understanding of how energy is consumed by an application, i.e., an energy consumption model, is a prerequisite for designing energy efficient applications. Energy cost analysis, the breakdown of energy consumption in different states of an application and in different hardware components such as processor, communication interface, display, and storage is important in this regard. In fact, analysis of energy performance often leads to energy-efficient application design. Designers concentrate on energy intensive components of an application and put effort to gain energy efficiency which results in reduced energy consumption.

Moreover, knowledge about the impact of design decisions on energy consumption is very useful at an early design stage as changes made in the final stage of an application are more expensive [73]; this helps reduce the time and cost of developing energy efficient applications. Suppose that the energy cost, speed, and compression ratio of a compression process are known. If the data rate and energy cost of a communication link are known, it is easy to decide whether or not compression before sending certain data is energy-efficient. We have addressed these two issues by proposing a finite state machine (FSM) based *energy consumption model* for software applications and by introducing the concept of *energy cost profile* that is comprised of high level energy cost information.

Additionally, we addressed the problem of resource constraints in smartphones to facilitate energy efficient sharing of resources among the smartphones. Smartphones are built to work alone and they typically cannot access or share each other's hardware or software resources. In the presence of a resource sharing infrastructure, these devices are able to share

and access each other's hardware, software resources, and data. The concept of *universal computation and communication interface* (UCCI) is introduced in this regard. As smartphones have become the preferred means of communication, and the smartphone traffic constitutes an increasingly large share of Internet traffic. As such, exploiting the energy saving potentials for wireless interfaces is highly relevant to smartphones. We propose a MAC level frame aggregation scheduler algorithm, *LEDAS* by observing smartphone WiFi traffic characteristics.

Finally, the problem of designing energy performance testing of smartphones is addressed. Millions of test configurations exist due to the large number of user controllable parameters in smartphones. Dealing with such volume of test configurations is quite impractical. A concept of user level test case for smartphones is introduced, and a heuristic based methodology is proposed for reducing the number of test cases for smartphone energy performance testing.

1.7 Solution Strategies

A brief overview of the solution strategies that address the problems mentioned above is given in the following. Details are provided in the following chapters.

1. **Energy Consumption Model:** The behaviors of the hardware components of smartphone are modeled as finite state machines (FSM), and the states are identified from the perspective of power consumption instead of their operational details. On the other hand, an application is viewed as a sequence of high-level activities, interspersed with idle periods. The duration of an activity coupled with the power levels associated with the corresponding hardware states allows us to calculate the energy cost of the activities. The work addresses the challenge of estimating energy cost of an application, and details of this approach can be found in Chapter 2.

2. **Energy Cost Profile of Devices:** The energy cost profile of a device contains high level energy cost information for performing application level tasks such as cost of sending a data packet of different sizes, and cost of writing a data block in the storage. It also includes the power consumption information for the different states of hardware components, such as *idle*, *active* and *sleep* states of the device's processor, and *transmission*, and *reception* states of the communication interfaces. The energy cost profile can be very useful at an early stage of application design process. This work addresses the challenge of having an energy-aware application development process, and details of this strategy can be found in Chapter 2.

3. **Universal Computing and Communication Interface:** We introduce the concept of *UCCI*, a generic model for sharing resources among portable wireless devices. The *UCCI* consists of two protocols *DCM* and *FIX* that enable any device to communicate with another device without having prior knowledge of each other. When a device finds that it either does not have the functionality, does not have sufficient resources to execute a task, or that it intends to save energy, the device exports that task to a nearby server. This work addressed the challenge of application-level energy management, and details of this technique can be found in Chapter 3.

4. **Low Energy Data-packet Aggregation Scheduler:** Data packets of the *web browser*, *YouTube video player*, and *Skype VoIP caller* on smartphones are captured using a *network packet analyzer*. The distributions of *packet sizes* and *inter-arrival times* of the packets in uplink and downlink traffic are observed. The packets are then grouped into *bursts* based on their *inter-arrival time*. The distributions of *durations, inter-arrival times* of the bursts, and *number of packets* in each burst are computed to observe the energy saving potentials. Based on the observations, a MAC level frame aggregation scheduler is proposed which considers *burst formation time*, *burst size*, and *number of packets* in a burst. The details of this energy-efficient communication technique can be found in Chapter 4.

5. **Design of Energy Performance Testing:** The large number of configurations of a smartphone is reduced in two steps. In the first step, the user settable parameters of a smartphone are identified, and categorized into *basic*, *active*, and *passive* parameters. The *active* parameters are then divided into two groups based on their impact on energy consumption. To the best of our knowledge, we are the first to address this problem of having consistent test configuration, and the details can be found in Chapter 5.

1.8 Validation Methodology

The proposed solutions were validated by conducting experiments on a real energy measurement test bench. A number of state-of-the-art smartphones, namely, *BlackBerry 9700*, *Google G1*, *Nexus One*, *Nokia E71*, and *HTC HD2* have been used in the experiments. Full details of the setup are given prior to the discussion of results for each solution. An experiment is repeated at least three times, and each time $30 - 45$ readings of power consumption are recorded. After obtaining the sets of readings, graphs are plotted to observe any discrepancy or abnormality; mean and standard deviation are then computed to check

the consistency of the data. 3G link related experiments are conducted at different times of day to observed any variations in readings. The variations in data were noted with the corresponding results. The experiments were conducted by multiple persons (research assistants) to check against individual mistakes.

Solution Specific Validation Approaches

1. **Energy Consumption Modeling:** In the absence of smartphones, initially, the experiments were conducted on laptops. The obtained results were published in [116]. Later on we conducted experiment on Google G1 and Nexus One to validate our models.

2. **Capability and Functionality Enhancement:** Two prototypes were developed on BlackBerry 9700 and Nexus One smartphones to realize the concept of universal computing and communication interface (UCCI). We have shown the efficacy of the model by awaking a laptop, transferring task, and putting the laptop into *sleep* state after completing the task.

3. **Anatomy of Smartphone WiFi Traffic:** We used *Google Nexus One*, *iPhone 3Gs*, and *BlackBerry 9700* to observe the WiFi traffic. To evaluate the performance of our proposed packet aggregation scheduler, LEDAS, we derived closed form formula through analysis, and conduct both simulation and experiments to evaluate the performance.

4. **Design of Energy Performance Testing:** In evaluating energy performances of network related applications (NRA), we used four different smartphones of *Android*, *BlackBerry*, *Nokia*, and *Windows Moblie* operating systems to show the energy performance for running network related applications.

1.9 Robustness of Solution Strategies

The robustness of our proposed ideas are explained by identifying their strength in the following areas.

1. As long as the hardware components of a device can be modeled as finite state machine based on their power consuming states, our proposed energy consumption model can be applicable to that device. Our model is also easily applicable to tablets

and other small systems such as PDAs. The proposed model is also extensible, as new hardware components can be added to the model by identifying their power consuming states.

2. The proposed UCCI concept is independent of the implementation process. In the prototype implementation, Bluetooth link was used for WPAN communication. Any other links such as NFC or ad hoc WiFi link can be used to communicate locally instead of Bluetooth link. Even infrastructure WLAN can be used when available. In fact, we have shown that infrastructure WLAN link is more energy saving.

3. The proposed data-packet aggregation scheduler is compatible with any MAC protocol as long as they support frame aggregation.

4. In our design of energy performance testing, there is a provision for checking dependency of power consumption on other hardware components. This helps to address issues where several hardware components are fabricated on the same chip.

1.10 Summary of Contributions

In this thesis, we focus on designing energy efficient smartphone applications. We explored issues that impact the energy consumption of smartphones, and introduce a formal model to better understand the energy consumption behavior of applications. The information gathered in the energy cost estimation process is utilized to build energy cost profile of devices that help design energy efficient applications. A summary of contributions in this thesis is given in the following.

- A finite state machine (FSM) based formal model is proposed to estimate the energy cost of mobile applications. We discuss the challenges involved in extracting model parameters, and propose a practical approach to estimate energy consumption of mobile applications. The concept of *energy cost profile* of handheld devices is introduced, which facilitates energy-efficient design of applications in the early stage of application development.

- The concept of *UCCI*, a generic model for sharing resources among portable wireless devices is presented. Two prototypes have been developed on Android and BlackBerry smartphones namely, *HTC Nexus One* and *BlackBerry 9700* to demonstrate the efficacy of the model. We explain how and in what situations resource sharing can be effective, and save energy with less delay.

- A test bench is described to capture and analyze the wireless access traffic generated by smartphone applications. Based on the above observations, we have identified opportunities to design new energy saving techniques specifically tuned for smartphones.
- The concept of user level test cases has been introduced to evaluate the energy cost of running applications on smartphones. A test selection technique is applied to a class of Network Related Applications, and the detailed design and implementation of a test bench is described to execute those test cases. This work provides a framework for researchers and developers to conduct experiments for evaluating the energy cost of applications on smartphones. To the best of our knowledge, this work is the first to address energy performance testing of mobile applications.

1.11 Organization of this Thesis

The solution strategies discussed in Section 1.7 are described in Chapter 2 through Chapter 5. The conclusions, summary and potential extension to the proposed work are given in Chapter 6. The detailed organization is given in Fig. 1.4.

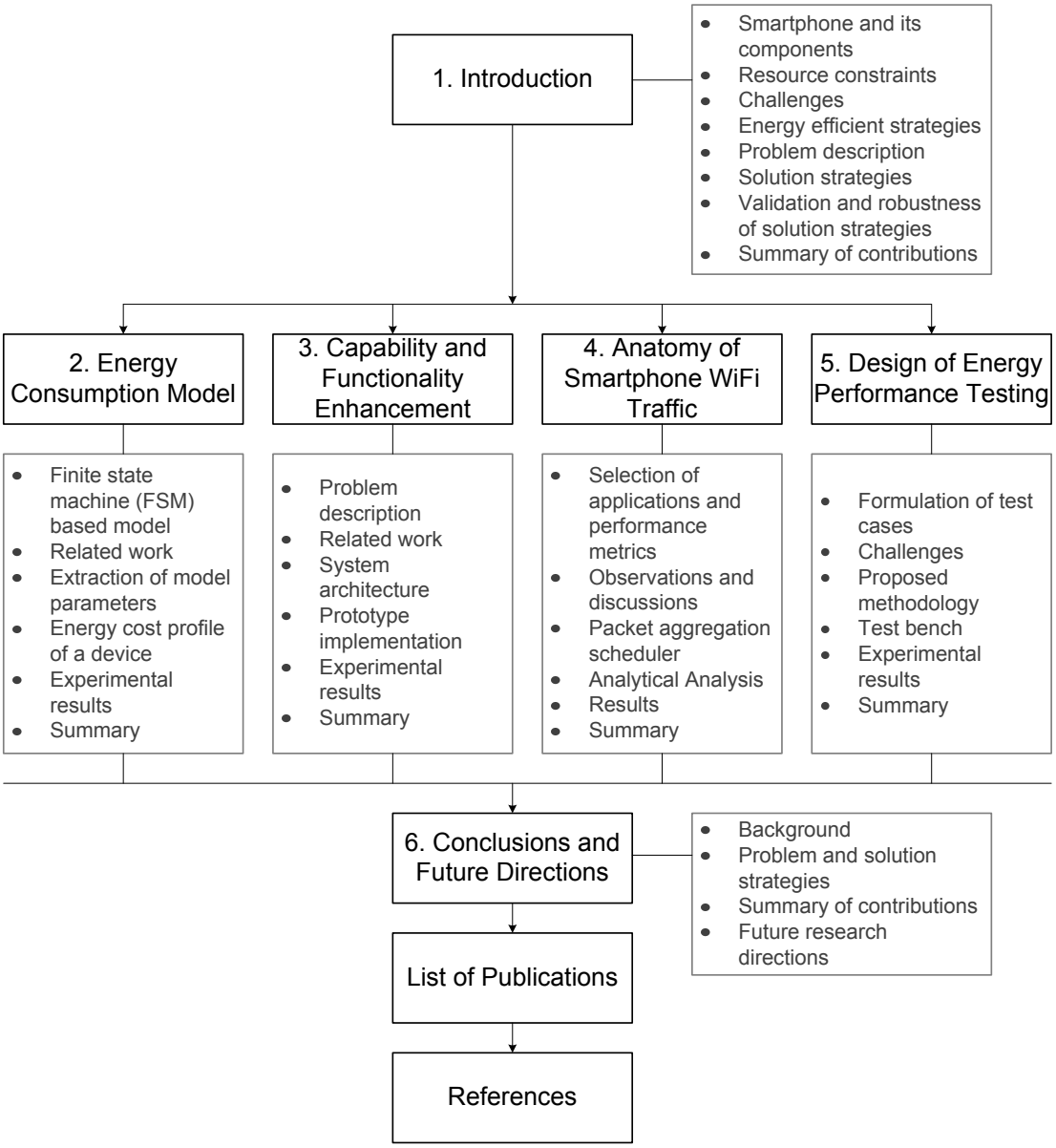


Figure 1.4: Organization of this thesis.

Chapter 2

Energy Consumption Model

In this chapter, we present a *finite state machine* based model to estimate the energy cost of applications running on a portable handheld device. We introduce a concept of *energy cost profile* that comprises the high level energy cost information as well as the power consumption of different states of hardware components of a device. The high level energy cost information can be used by application developers to consider the impacts of design decisions on energy cost at an early stage of software design. We propose to plug in the *energy cost profile* into the existing device emulators to estimate energy cost of an application. By means of extensive experiments we show the impacts of packet sizes at the transport layer of communication, block sizes to read/write data involving micro-disk, and block sizes for data encryption and decryption on energy performance of application software.

2.1 Problem Description

This section has been organized in terms of four subsections: (*i*) motivation; (*ii*) proposed framework; and (*iii*) contributions. We summarize the organization of this chapter at the end of this section.

2.1.1 Motivation

To the developers and researchers, energy-efficiency means to enhance the capability and functionality of the devices so that they can run applications longer with the same amount

of energy. Energy cost analysis, the breakdown of energy consumption in different states of an application and in different hardware components such as *processor*, *communication interface*, *display*, and *storage* is important to them. In fact, analysis of energy performance leads to energy-efficient application design. Designers concentrate on energy intensive components of an application and put effort to gain energy efficiency which results in reduced energy consumption.

Analysis of the complexity of a software application can be mapped to the energy cost using power consumption models of different hardware components such as processor, memory, display, and storage [70]. However, this approach becomes intractable for modeling event-driven, interactive applications with different system level complexities such as memory hierarchy, input-output buffering, and many peripheral components of smartphones. This strategy is not easy to automate and is not practiced much.

Energy profiling of applications by means of physical measurements is limited to the entire chip due to chip integration, architecture and packaging [49, 66]. The final chip may not be available during software design and implementation phases. For a developer, it is not practicable to perform measurements for all applications on all devices to give the energy performance results.

The simulation-based power profiling techniques mainly focus on embedded systems with dedicated applications, and these are available only for the lower levels of hardware design, at the circuit level and to a limited extent at the logic level [132, 82, 83]. These tools are very slow and it is impractical to evaluate the power consumption of smartphone software because the application power consumption would only be known at the very last stage of the design process. The emulation based approaches speed up the power estimation process, but they need prototyping platform such as an FPGA-board and software tools to get an estimate of energy consumption and profile information [51, 56, 30].

This work focuses on how the application developers can weigh the implications of their application design decisions on power consumption. We use a concept of *energy cost profiles* of devices, and propose a measurement and software emulation based energy performance evaluation framework.

2.1.2 Framework

We, at first, formulate an analytical model for estimating energy consumption of a software application. The behaviors of the hardware components of a mobile device are modeled as finite-state machines (FSM). The states have been identified from the perspective of power consumption instead of their operational details. An application is viewed as a sequence of

high-level activities, interspersed with idle periods. Intuitively, the duration of an activity coupled with the power levels associated with the corresponding hardware states allows us to calculate the energy cost of the activities. In order to estimate the energy cost of an application, we need to map the theoretical model into a practical framework so that we are able to measure the energy performance. Our proposed framework comprises the following elements.

Energy Cost Profile of a Device: The energy cost profile of a device contains high level energy cost information for performing application level tasks such as cost of sending a data packet of different sizes, and cost of writing a data block in the storage. It also includes the power consumption of the different states of the hardware components such as *idle*, *active* and *sleep* states of the device's processor, and *transmission*, and *reception* states of the communication interfaces. This information is crucial to making application design decisions.

Software Device Emulator: The second component, *software emulator* of a device, is available to the developers from all major smartphone platforms, namely, *Android*, *iPhone*, and *BlackBerry*. When an application is executed on an emulator, accurate information about how an application uses the different hardware components can be obtained by running the application. For example, the *Nokia Energy Profiler* enables measurement of power used by an S60 device and provides details of some key power-consuming activities, such as CPU, mobile-network, and IP-traffic activity [110]. Therefore, if we plug-in the energy cost information, such as costs for CPU, send/receive data packets in the emulator, the cost for executing an application can be easily estimated.

The advantage of our approach is twofold: (*i*) we reuse existing infrastructure *i.e.*, the software emulator of the devices. Hence we do not need any extra simulator, or hardware modules. As the device emulator behaves like an actual device, we get most accurate information about the application behavior. Getting application behavior is very crucial as applications are event driven, interactive, and becoming increasingly complex. (*ii*) Since the power consumption in different hardware states remain largely constant, we need to measure the hardware state-power cost once, and that can be done by the experts.

We describe how to integrate the energy estimation framework in the development cycle of mobile applications in Fig. 2.1. Designers refer to the energy cost profile of a target device during the design phase, make high-level energy-performance trade-offs, and implement an initial version of a target application. The energy cost of the initial version of the application is obtained by running it on the device emulator with the energy-cost profile

of the target device. In fact, energy evaluation can be carried out along with functionality testing. After analyzing the energy costs in different hardware components as well as different states of an application, adjustments are made in the design to achieve better energy efficiency, if necessary. This process is iterated until the target energy performance level is achieved.

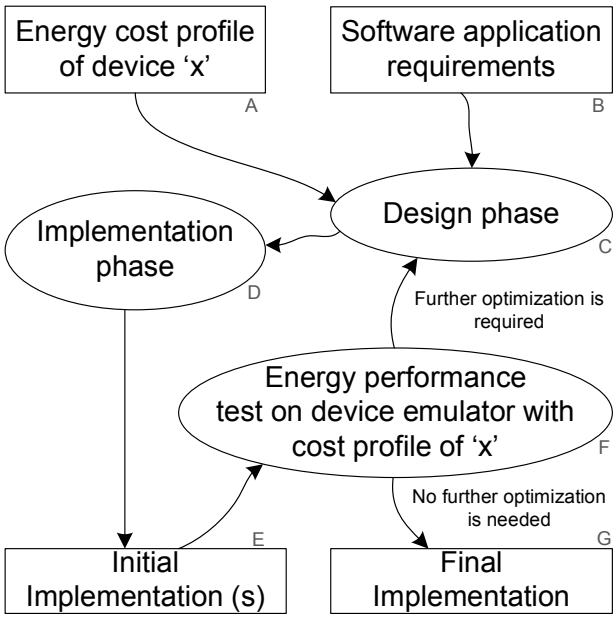


Figure 2.1: Schematic diagram of our proposed energy estimation framework.

Knowledge about the impact of the design decisions on energy consumption is very useful at an early design stage as changes made in the final stage of an application are more expensive [73]. Our proposed *energy cost profile* of a device becomes available at an early design stage, and it provides high-level energy cost information to application designers. Thus it helps reducing the time and cost of developing energy efficient applications. Here, we give a few examples below where design decisions need to be made based on energy cost information.

- Compressing data before transmission results in saving energy in some scenarios;
- Accuracy of location information can be compromised with less frequent access of the global positioning system (GPS);

- Offloading a computation intensive task to *cloud* or a nearby surrogate server costs less energy in some scenarios;
- Additional energy is required for using the *secure file transfer protocol* (SFTP) to satisfy a security requirement instead of FTP.

In the above mentioned scenarios, the designers need high level energy cost information to make decisions for energy efficiency. Suppose that the energy cost, speed, and compression ratio of a compression process are known. If the data rate and energy cost of a communication link are known, it is easy to decide whether or not compression before sending is energy-efficient. Our proposed energy cost profile contains those information to make design decisions for energy efficiency. We provide examples of energy cost profile parameters of a smartphone using a energy performance measurement testbed in Section 2.5.

2.1.3 Contributions

We summarize the contributions of this work in the following.

- We discuss the issues related to energy performance evaluation of mobile applications and present a finite-state machine (FSM) based formal model to estimate the energy cost of mobile applications (Section 2.3).
- An energy estimation framework is proposed to extract the values of the model parameters. We show how to relate them with *FSM* based energy consumption model (Section 2.4).
- The concept of *energy cost profile* of handheld devices is introduced, which facilitates energy-efficient design of applications in the early stage of application development (Section 2.5).
- We have conducted on-device experiments to validate our proposed framework and discuss the results (Section 2.5.3).

The rest of this chapter is organized as follows. We discuss the related work in Section 2.2 which is followed by the description of FSM based energy consumption model. Section 2.4 explains how to get parameters of the model (Block–F of Fig. 2.1). In Section 2.5, we describe *energy cost profile* (Block–A of Fig. 2.1), experimental setup for energy cost profile of *HTC Nexus One* smartphone. Results from the experiment are discussed in the same section. We conclude this chapter in Section 2.6.

2.2 Related Work

The key concepts of software strategies for energy management and system level power-aware design techniques have been described in [91, 150]. Lorch and Smith [91] discussed how energy management can be done at different levels of a system, and how the energy management strategies can be evaluated. Unsal and Koren [150] defined the terminologies related to energy-aware techniques. A comprehensive survey of software based energy saving techniques for handheld devices has been done by Naik [104]. Kansal and Zhao [74] discussed the research challenges in application layer energy optimization, and demonstrated how energy-profiling helps a developer choose between alternative designs in the energy performance trade-off space.

Many energy estimation models and related energy efficient solutions have been proposed in the literature at different abstraction levels such as circuit level, code level, and system level. In this section, we mainly discuss system level techniques and tools. In system level, a system is decomposed into a set of components such as display, memory, and CPU, and each component is modeled individually with its own power level. We also include few papers on simulation and emulation based energy estimation for completeness.

2.2.1 Simulation and Emulation Based Estimation Tools

There are a number of simulation based energy estimation techniques that use different levels of abstraction of the hardware such as circuit, gate, register transfer, architecture level. The level of details influences the accuracy and speed of the simulators [14, 102, 64]. The instruction level power consumption models measure power consumption of each instruction while executed in a loop [149, 132, 82]. Celebican *et al.* [18] proposed a cycle accurate energy simulator for peripheral devices. The simulation based power estimation tools focus on dedicated system-on-chip embedded systems where accuracy of estimation is important. However, these techniques require long simulation times due to their inherent nature, and thus they are impractical for complex applications and systems like smartphone platforms.

Authors of [51, 56, 30] proposed the idea of power emulation using hardware acceleration to speedup computation as compared to simulation based approaches. Hardware prototyping platforms such as FPGA-boards are used to estimate power consumption along with functional characteristics in real time. These approaches drastically accelerate the development process. However, significant challenges are there due to increased hardware complexity of devices such as smartphones.

2.2.2 Measurement Based Estimation Tools

Flinn *et al.* [49] described a tool called PowerScope for profiling energy usage of an application. It continuously samples the power for profiling application energy usage. An external hardware multimeter with an in-built clock is used to sample the monitored device. At each sampling time, the CPU status of the monitored computer is noted. This status consists of the current value of the Program Counter, the Process ID (PID) and the interrupt handling details. At the same time, the multimeter records the instantaneous current, and voltage values. Later, during a energy profiling post-process stage, the CPU values are associated with the multimeter readings for each sampled interval in order to reconstruct the power consumption details of the monitored computer.

Banerjee *et al.* [9] presented a tool called PowerSpy, which tracks and reports the battery energy consumed by the different threads of a monitored application, the operating system (OS), and other applications in a multi-threaded environment along with I/O devices. Initially, PowerSpy keeps track of an application's CPU times, I/O activities and energy consumption. Then, the energy consumption by other applications are filtered to get the energy consumption by that particular application.

Since the energy sampling in hardware method is done independent of the system being monitored, the sampling frequency can be very high and is independent of any OS activity. The instantaneous power values are also more accurate than the software counterparts. The software approach is limited by the sampling rate at which the OS gets updates from the actual battery, and suffers from the drawback that at very high sampling rates, the battery device may not be able to report the changes accurately.

Haid *et al.* [57] proposed a co-processor for run-time energy estimation in system-on-chip (SoC) designs. The performance overhead of this profiling technique is low as the estimation is done through hardware and fully parallel to the functional units on the SoC. With the presence of a such a co-processor, the energy cost of individual applications can easily be measured and profiled. This co-processor can be activated only during energy cost profiling to save energy consumed by the co-processor itself.

Zhang *et al.* [173] described a tool called PowerBooter for automatic power model construction on smartphones. It uses built-in battery voltage sensors and knowledge of battery discharge behavior to monitor power consumption. It controls the states of the hardware components to get the breakdown of energy consumption. The authors used another tool called PowerTutor that uses the power model generated by PowerBooter for online power estimation. PowerBooter makes the power models for new smartphone variants, and the PowerTutor facilitates designing and selection of power efficient software

for embedded systems. Dong and Zhong [41] proposed a high-rate automated energy model called *Sesame*. *Sesame* also uses smart battery interface of smartphones instead of external circuitry, and it includes a set of techniques to overcome the limitations of batteries for energy modeling up to 100 Hz speed.

2.2.3 Studies of Energy Consumption Behaviors

Ferreira *et al.* [47] collected 7 million battery usage information points from 4000 participating devices across the world. They analyzed charging activity, energy level, device type, temperature, voltage and uptime of batteries to assess how users charge their smartphones, and the implication of charging on battery life and energy usage. The author argued that such study helps identifying design opportunities for reducing energy cost, and also predicting when energy intensive applications should be scheduled.

Carroll and Heiser [17] stressed the need for a good understanding of where and how the energy is used. They presented a detailed measurement analysis of over all energy consumption of a smartphone along with a breakdown of power distribution to CPU, memory, display, graphics hardware, audio, storage, and networking interfaces. Though they did not use latest generation smartphone in their main experiment, it still provides a good understanding of the energy consumption of the handheld devices. Specially the energy consumption breakdown can help make high-level design decisions.

Wang and Manner [154] examined the energy costs for sending and receiving per bit of user data over *Edge*, 3G and WiFi. The energy consumption characteristics they provide help application designers in choosing communication interfaces for longevity of battery life. Qian *et al.* [121] implemented a tool called *Application Resource Analyzer* (ARO) to analyze the radio resource usage for smartphone applications. The ARO comprises two components: the data collector and the analyzers. The data collector captures data for radio interface usage, user activity, and application performance. The collected data are fed into the analyzers for offline analysis.

2.2.4 Energy Efficient Techniques

Balasubramanian *et al.* [8] studied the energy consumption patterns of 3G, GSM and WiFi technologies. They found that 3G and GSM incur a high tail energy overhead after each episode of data transfer. Based on their observation, they developed a protocol called *TailEnder* which reduces energy consumption of mobile applications. *TailEnder* basically fetches more data in advance, and improves user-specific response times with less energy.

Based on the same observation, authors of [88] developed a scheme called *TailTheft* which employ a *Dual Queue Scheduling* algorithm to prefetch and delay data transfer. Dogar *et al.* [40] proposed a technique called *Catnap* that keeps the WiFi interface in the *sleep* state even during data transfer.

Shye *et al.* [141] collected traces of user activities on a smartphone and used the information to characterize power consumption. They observed that energy consumption widely varies user to user, and the display and the CPU are the two largest power consuming components on smartphones. To reduce the energy consumption between two interactions of a user, the authors implemented a scheme that slowly reduces the screen brightness over time. Their optimization techniques save 10.6% of total system energy savings with a minimal impact on user satisfaction.

At the code level, Naik and Wei [105] studied how the designs of algorithms and implementations affect energy utilization. They observed that different instructions consume different amount of energy, and proposed three energy saving strategies: assigning live variables to registers, avoiding repetitive address computations, and minimizing memory accesses. Jain *et al.* [70] extended the work of Naik *et al.* [105] to identify some important factors and their impact on energy cost. These factors include CPU operations, memory access, I/O, and switching complexity. By distinguishing the importance of these factors on energy consumption, algorithms can be designed to achieve energy efficiency. However, the proposed model at the code level did not address I/O components including a communication subsystem involved in web-based applications.

2.2.5 Energy Efficient Systems

Rumble *et al.* [130] proposed an OS called *Cinder* that is developed on top of *HiStar* OS. *Cinder* consists of *capacitors*, and they are instruments for tracking and enforcing energy usage in a system. The *task profiles* of *Cinder* keep the statistics of application energy consumption, and let the users to express energy policies for applications in terms of minutes or hours based on previous application behavior. Thus the *capacitors* apply energy policies generated by *task profiles* according to the intent of the user of a device. Zeng *et al.* [170, 171] coined the term *first class OS resource* for energy, and proposed *currentcy model* to generate energy policy and enforce in a system.

2.3 Energy Consumption Model

In this section, we present a formal energy consumption model. In the next section, we present a practical approach to extract the model parameters for estimating the energy consumption of an application.

Smartphones comprise software applications, an operating system (OS) and hardware components. Users interact with the applications through input/output components such as keypad, display and touch screen. Communication interfaces such as cellular, WiFi, and Bluetooth connect them with other devices. For all activities, the applications utilize the hardware components which are controlled and managed by the OS. Hardware components ultimately consume energy supplied by the battery. A simplified diagram is given in Fig. 2.2 to show the energy consumption relationship of the components of a device. However, different applications require the participation of hardware components in different proportions, and thus, the energy consumed by the applications varies. If we know the energy cost of using the hardware components and the usage patterns of the different components by an application, we can calculate the amount of energy consumed by an application.

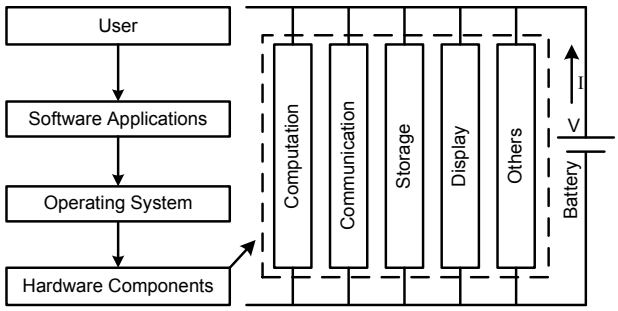


Figure 2.2: Components between application layer and battery on a portable device.

A hardware component stays in different power states based on their functionalities. Hardware components with energy saving features can stay in low power states temporarily when they are not needed. Therefore, we need to know the details of the states and state-switching behavior of the components. For this, we can use a Finite State Machine (FSM) notation as it is widely used to describe the dynamic behavior of hardware components [28, 59].

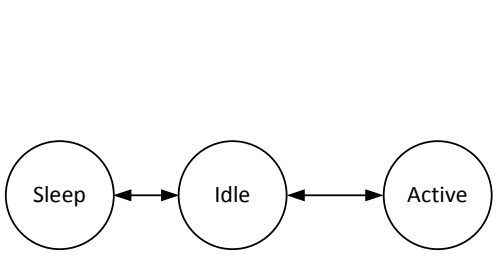
We define a general FSM as $FSM = (\Sigma, S, \Lambda, \partial)$, where Σ is an input alphabet (control messages of hardware components), S is a set of states, Λ is a state transition functions

$\Lambda : \mathrm{S} \times \Sigma \rightarrow S$, and ∂ is a power function. The power function $\partial : S \rightarrow \Re$ returns a real valued power level consumed in a state. Here, we assume constant power consumption, *i.e.*, when a hardware component remains in a state for a certain length of time, the component draws constant amount of power. In this case, if the supply voltage of a device remains constant, the consumed power is directly proportional to current. We use the idea of *state residence time* to denote a period of time in which a hardware component remains in a particular state. Under the constant power assumption, given the current level l of a state and the state residence time Δt, the energy consumption of a component is $v \times l \times \Delta t$, where v is the supply voltage. Let $FSM_i = (\Sigma_i, S_i, \Lambda_i, \partial_i)$ denote an instance of the general FSM for a hardware component i. At the abstract level, this component has n_i number of states with each state satisfying the constant power assumption. Let $S_i = \{s_{i,1}, s_{i,2}, \cdots, s_{i,n_i}\}$ denote the state set containing these states.

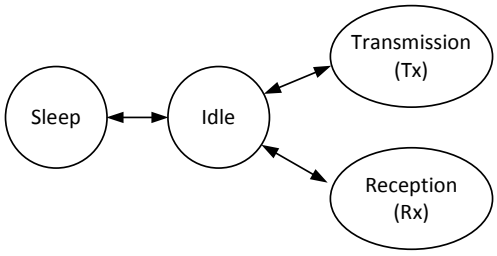
Based on the preceding discussion, we describe the power consumption states of different hardware components of a device. Figure 2.3 illustrates the power consumption states of processor, communication interface and storage. Generally, the computation component is referred to as the processor integrated with input/output element controllers. We assume that only the processor in the computation component has a power-saving feature and operates in three modes: active, idle, and sleep (Fig. 2.3(a)). In the *active* state, a processor becomes busy executing instructions and consumes its highest power. In the *idle* state, the processor does not execute any instruction, but remains ready to execute. Processor utilization is almost zero in the *idle* mode. In the *sleep* mode, most of the circuitries in a processor are turned off, whereas the timer and the *wake-up* circuitry remain enabled.

An FSM model for the communication interface is given in Fig. 2.3(b). Typically, a transceiver has four states (operating modes) with each state satisfying the constant power assumption [59]; in order of decreasing power levels, the states are *transmission* (Tx), *reception* (Rx), idle, and sleep. As shown in Fig. 2.3(c), the number of states of a storage component, e.g., internal storage or external *SD card* storage is also four: *Write*, *Read*, *Idle* and *Sleep*.

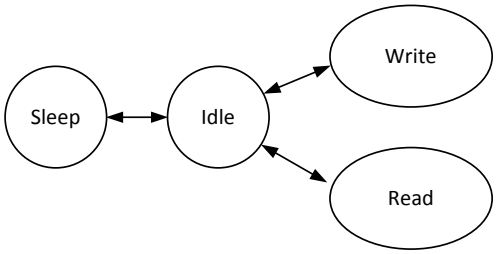
Power consumption in a state of some components varies depending upon some parameters. For example, power consumption in *active* state of display varies based on its *brightness* level. Similarly, the power consumption of memory are different in *Write* and *Read* states based on the level of memory hierarchy at which it performs an operation. The FSM models of *display* and *memory* are given in Fig. 2.4, where h is the number of memory hierarchy levels. In fact, some processors have multiple operational modes with varying power consumption and speeds, and in such cases we get multiple sub-states of the *active* state.



(a) Power states of a processor.



(b) Power states of a communication interface.

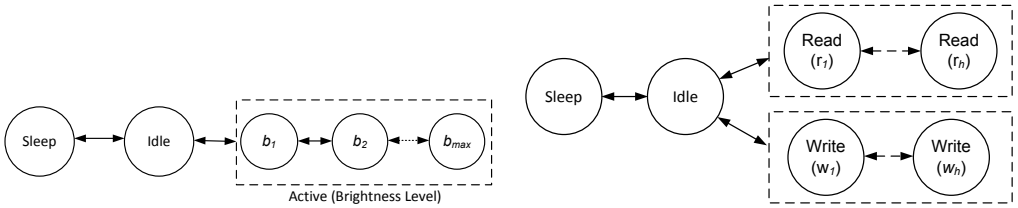


(c) Power states of a storage component.

Figure 2.3: FSM models for processor, communication interface and storage.

Figure 2.5 shows the power consumption of a *HTC Nexus One* device in different states. It consumes 35 milliwatts (mW) of power in *stand-by* mode when display is *off*. The device consumes 453 mW of power when it is ready to execute user applications with display *on*; we call it *idle* mode. To observe the processor's power consumption, we compute the exponents of pairs of random real numbers to fully load the processor. Then, the power consumption of the system becomes 1057 mW. To check the power consumption of writing to and reading from storage, we write and read data blocks of 1 kilobyte. The corresponding amounts of power consumption are 801 and 719 mW. Thus, after we subtract the idle costs, processor, storage write, and storage read costs become 954, 348, 266 mW, respectively.

While the experiment was conducted, every time we fully engaged a specific component for a while, and observed constant power consumption. However, when an application runs on a device, it utilizes one or more components at a time according to its needs and the power consumption pattern varies over time. Thus the instantaneous current consumption changes randomly as shown is Fig. 2.6. In order to get energy consumption of an application, average current consumption is computed over certain period. In Fig. 2.6, the average current consumption is 254.7 milli-ampere in 40-second time. Since the power supply voltage is 3.7 volts, energy consumption becomes 37.7 Joules.



(a) Power states for display. (b) Power states for memory.

Figure 2.4: FSM diagrams for display and memory.

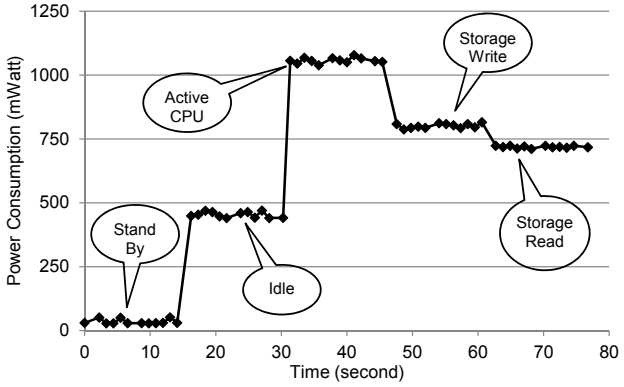


Figure 2.5: Power consumption in different states of *HTC Nexus One*.

Having the above practical examples, we now formulate the expression for energy consumption on a device. Suppose that there are n number of hardware components in a portable device. Each of the components has a fixed number of predefined states, n_i. Now state $s_{i,j}$ means the jth state of component i, where $1 \leq j \leq n_i$. Suppose that component i switches to state j, $n_{i,j}$ number of times in a given time period T and it stays $\delta t_{i,j,k}$ amount of time each time it enters $s_{i,j}$, where $1 \leq k \leq n_{i,j}$. From the above model definition and description, we can now derive the formula of energy consumption $E(T)$ by the device in time period T. As shown in Eq. 2.2, $\Delta t_{i,j}$ is the total amount of time that component i stays in state j within time period T. In Eq. 2.3, $\psi(S_{i,j})$ is the fixed current consumption

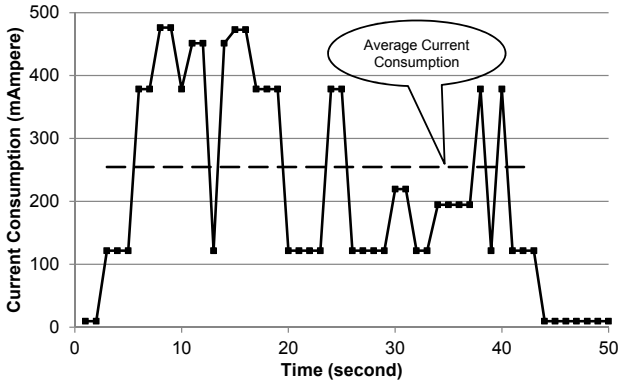


Figure 2.6: Instantaneous current consumption profile of a device.

with constant voltage supply v.

$$E(T) = \sum_{i=1}^{n}\sum_{j=1}^{n_i}\partial(s_{i,j})\sum_{k=1}^{n_{i,j}}\delta t_{i,j,k} \tag{2.1}$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n_i}\partial(s_{i,j})\Delta t_{i,j} \tag{2.2}$$

$$= v\sum_{i=1}^{n}\sum_{j=1}^{n_i}\psi(s_{i,j})\Delta t_{i,j} \tag{2.3}$$

The energy estimation given in Eq. 2.3 does not include the cost of switching states of the hardware components. As expressed in Eq. 2.1, $\Delta t_{i,j}$ is a sum of $n_{i,j}$ state residence times. For example, a processor routinely switches from idle to active, active to idle and so on while an application is executed. The residence times in these states varies depending on the attributes of applications and the scheduling algorithm followed by the OS. The energy required to switch from one state to another state is very small in comparison to the energy spent in different states of a component. The energy of switching states can be expressed as a summation of products of number of state switching and energy required for each switching. In Eq. 2.4, $\varepsilon(T)$ indicates the total energy spent in switching states in time T, where, $\epsilon_{i,j}$ is the required energy for switching into state j from any other state of

component i. Intuitively, the energy cost increases with the number of switching of states.

$$\varepsilon(T) \;=\; \sum_{i=1}^{n}\sum_{j=1}^{n_i} n_{i,j}\;\epsilon_{i,j} \tag{2.4}$$

According to Eq. 2.3, we can compute the total energy consumption, once we know the power consumption of each state of the components and the *state residence times* of those states. From an application point of view, when we want to estimate the energy consumption, we need to know the states and residence time of each state of all components used by that particular application. However, the OS schedules different applications to access different components of a device. Since there are a number of applications and the OS running simultaneously on a device, the particular time when an application gets access to a component or how long it continues to access the component cannot be known a priori. Moreover, the length of time that an application uses a component at a time can be very small, which can be even in microsecond range.

For some components such as a processor, the usage information can be extracted easily for the peripheral components, such as communication interface, it is difficult to get the timing and get the residence time of each state indicated in the model. However, the energy consumption for performing activities such as sending or receiving data can be measured for those components. Therefore, the energy cost can be calculated by taking products of power consumption and residence times of states for some components; for some other components, the energy costs can be calculated by activities performed in those components. We rewrite Eq. 2.3 in the form of Eq. 2.5 to incorporate the two approaches for estimating energy. The n components are divided into two sets of m and $n-m$ components. For the first set, the energy cost is estimated using Eq. 2.3, and for the second set of components ($m+1$ to n), energy costs are estimated based on activities. Suppose that an activity l performed on component i is denoted by $a_{i,l}$, and $\xi(a_{i,l})$ expresses the energy cost of performing activity $a_{i,l}$. Now, Eq. 2.3 takes the form of Eq. 2.5.

$$E(T) \;=\; v\sum_{i=1}^{m}\sum_{j=1}^{n_i}\psi(s_{i,j})\Delta t_{i,j} + \sum_{i=m+1}^{n}\sum_{\forall l}\xi(a_{i,l}) \tag{2.5}$$

The terms given in Eq. 2.5 are easily computed in device emulator environments, which we discuss in the next section.

2.4 Getting Model Parameters

For some components, the power consumption in all the states, and state residence times are needed according to Eq. 2.5. For the rest of the components, energy consumption based on activities performed are required to estimate the energy consumption of an application. The authors of [49, 9] implemented separate tools for computing these values. They used system level primitives to capture the state residence times on a device, and used hardware and software tools, respectively, to measure the consumed power. However, we propose to use a device emulator for state residence time and use of hardware tools to measure the power consumption of the hardware states. The advantages of using those tools are as follows:

- Device emulators are available to application developers from the OS vendors. BlackBerry has emulators for all of their devices. In the Android emulator, settings such as RAM size, OS version, and storage size can be customized to have desired device settings. Therefore, application developers do not need to buy actual devices to test their applications. In the Android emulator, the profile information is available, and most importantly, the system calls can be made on emulator by loading appropriate applications. To our understanding, a device emulator can be used to get the state residence time easily, with less effort and costs.
- As Banerjee *et al.* [9] also mentioned, software tools for measuring a system's power consumption or loss of battery energy have many drawbacks. It is limited by the sampling rate at which the OS gets updates from the battery. Moreover, the battery circuitry may not be able to report its SoC (State of charge) state accurately at high sampling rates. Thus the results obtained from software tools are dependent on the particular device and its battery. The hardware approach is costly and the developers require expertise to handle the equipments. Most importantly, the power consumption in different states of components are fixed, so, there is no need to measure them every time, which is a loss of time and effort, whereas these values can be measured once and supplied by the manufacturers.

A schematic diagram of the estimation process is given in Fig. 2.7, the hardware configuration and energy cost profile of a target device x are fed into a device emulator. When target application y is run on the emulator, the usage profile on the device x is evaluated. Usage profile contains the usage information of the hardware components by the application. Processor utilization and number and size of data packets sent/received by the application in a given time duration are example parameters of usage profile. The total

energy cost as well as the costs in different hardware components can be calculated by applying Eq. 2.5. Another energy cost is computed without running application y and the difference of costs is the energy cost of running the target application y on device x.

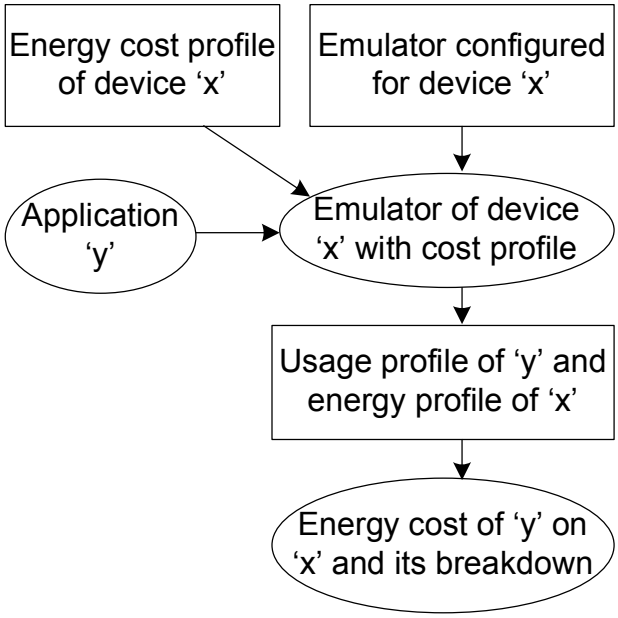
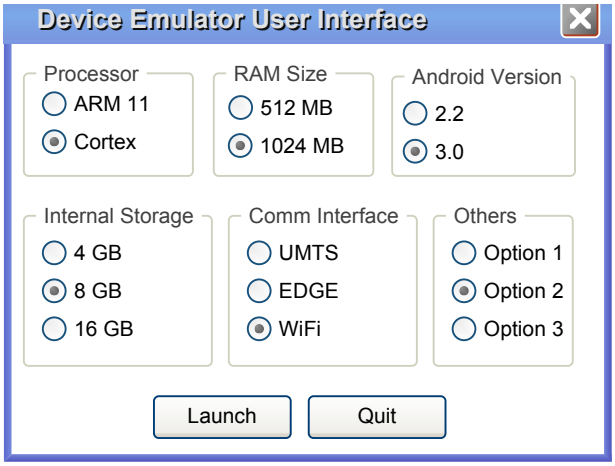


Figure 2.7: Schematic diagram of energy estimation process using device emulator.

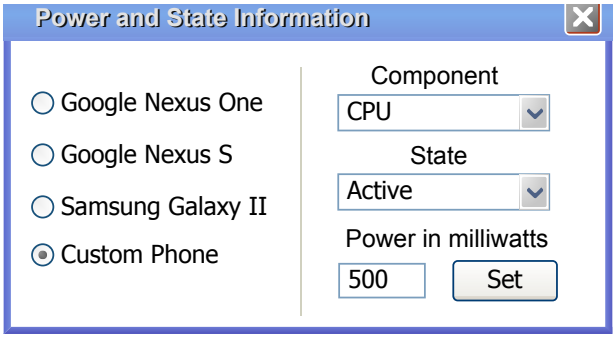
Figure 2.8 illustrates the user interfaces of the proposed emulator. A hardware configuration is chosen from the available options and the emulator is launched to execute a target application (2.8(a)). The emulator gives the values of state residence times such as the utilization of the processor which is a percentage of processor usage, storage read and write parameters, and parameters for communication interfaces. To get the amount of energy spent by the application, we need to plug-in the power consumption profile of the hardware states [42, 61]. As shown in Fig. 2.8(b), a developer can observe the changes in energy consumption by the applications by adjusting the values of power consumption. The details of how to get costs using *energy cost profile* is described in the next section.

2.5 Energy Cost Profile of a Device

In this section, we discuss the concept of *energy cost profile* in detail, and describe the experimental setup for evaluating the energy cost profile. In the experiments, we measure



(a) For choosing components of a device.



(b) For choosing power-state information of a device.

Figure 2.8: User-interfaces of a device emulator.

the parameters such as processing cost, data encryption cost, read/write costs of *HTC Nexus One* smartphone and explain the outcomes of the experiments. An example is given to show the importance of energy cost profile at the end of this section.

2.5.1 Profile Parameters

Applications get access to the hardware components via middleware, OS, and communication protocols, as depicted in Fig. 2.9. A software developer needs to choose the values of parameters such as buffer size, packet size, or block sizes during the implementation phase. However, the relationships between such application level parameters and energy

costs are not always intuitive or linear [114]. Consequently, these values should be chosen carefully in order to develop energy efficient applications. The concept of *energy cost profile* helps during design and analysis phases by providing suitable range and energy consumption trends of such parameters. In the energy estimation phase, the cost profile helps the estimation process by providing power consumption data in different states of hardware components, such as the power consumption in the *active* state of a processor, *idle* state of a communication component. It also provides energy costs of high level tasks, such as for sending a data packet or compressing a file.

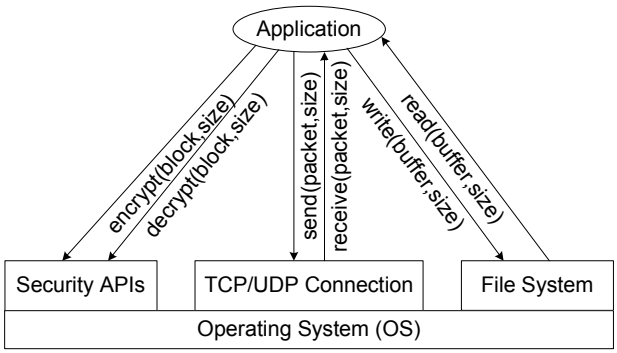


Figure 2.9: Interactions of applications with operating system.

Suppose that an application needs to transfer s bytes of data from a smartphone to an Internet server. The skeleton of the program is given in Program 2.5.1. A software developer of this application needs to make some high level design decisions such as which communication link or security protocol to use for transferring data. A comparison of energy costs helps to make these high level decisions. In addition, the energy consumption can further be optimized by choosing appropriate values of parameters of a selected high level component. For example, WiFi communication link is selected to transfer data, and by choosing suitable packet sizes, the energy consumption can be further reduced. As we observe in Program 2.5.1, the cost of computation is also affected by the read/write or packet size parameters due to the number of calls needs to make from the application.

Algorithm 2.5.1: DATA TRANSFER()

$s \leftarrow$ size of data
$x \leftarrow$ buffer size for reading from storage
$y \leftarrow$ block size for encryption
$z \leftarrow$ packet size for sending data
...
for $i \leftarrow 1$ **to** $\lceil \frac{s}{x} \rceil$
$\begin{cases} \text{READ\_FROM\_STORAGE}(x) \\ \text{ENCRYPT\_READ\_DATA}(y) \\ \text{SEND\_ENCRYPTED\_DATA}(z) \end{cases}$
...

To estimate the energy cost of the given application, the costs of performing tasks involved with the application are required. We provide a sample list of very common parameters in the following.

- energy costs of reading and writing in the storage;
- energy costs of data encryption and decryption;
- power consumption when a processor is active;
- energy required to send and receive a data packet using different communication interface;
- energy costs of display with different brightness levels;

The costs of operating other components such as global positioning system (GPS) and built-in camera are important for designing applications involving them. We conducted experiments to get some of the costs on *HTC Nexus One* smartphone. Before discussing the results, we describe the experimental setup in the following.

2.5.2 Experimental Setup

We use a test bench to facilitate experimentation of smartphones to evaluate parameters of energy cost profile. As shown in Fig. 2.10, the setup includes (*i*) smartphone(s); (*ii*) power

supply with a high precision current measurement unit; (*iii*) a desktop or laptop computer to control and monitor the power supply unit; (*iv*) a wireless Access Point (AP); (*v*) a web server; and (*vi*) a cellular network connection with data access.

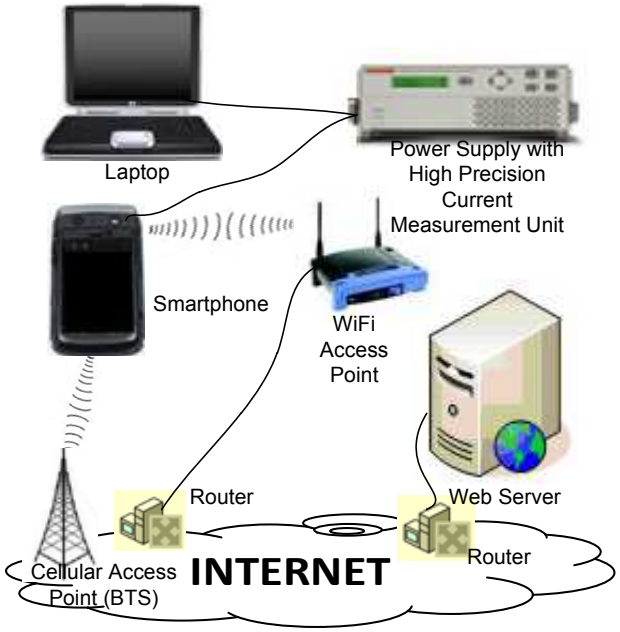


Figure 2.10: Experiment setup of test bench.

The power supply is initialized with the battery ratings of the smartphone through a controller program installed on a laptop computer. The smartphone is turned on and a test configuration is selected before conducting experiment [113]. The consumed current is measured by a monitor program installed in the same computer with and without running the test application. We used Keithley 2304A, a high speed power supply with accuracy in measuring current of $\pm(0.2\% + 400\mu A)$.

We take three sets of readings without running a target application and take another three sets of readings by running the target application. Then we check the individual set of readings whether or not the readings have similar trend and do not contain much fluctuations. Finally, we calculate the average from each set of readings, and take the difference to get the average power consumption by the application. The energy cost is computed by taking products of power consumption and application execution time.

2.5.3 Experimental Results

In this section, we discuss the results of the following energy cost parameters performed on *HTC Nexus One* smartphone.

- Power consumption of processor in active state;
- Power consumption and processing rate of DES (Data Encryption Standard) algorithm;
- Power consumption for read and write with the *sdcard* (Secure Digital Card);
- Comparison of energy costs for using 3G, WiFi and Bluetooth communication links.

We have used three standard CPU benchmarking programs to fully load the processor, and measured the power consumption of *HTC Nexus One* device. These programs do not use storage, and communication. The power consumption of the device are measured twice: when a program is executed and when that program is not executed. The difference of the averages of two sets of readings gives the cost of executing the program. As shown in Fig. 2.11, the device draws almost same power for all three benchmark programs. This energy cost information along with processor utilization (*e.g.,* 50% CPU utilization in 50 seconds means processor is fully active for 25 seconds) helps the application designers to get an estimate of the processing costs.

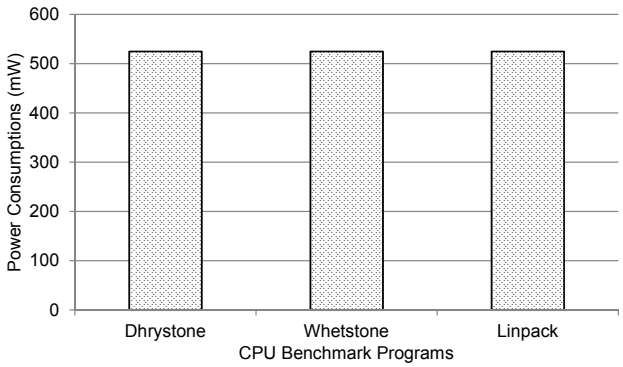
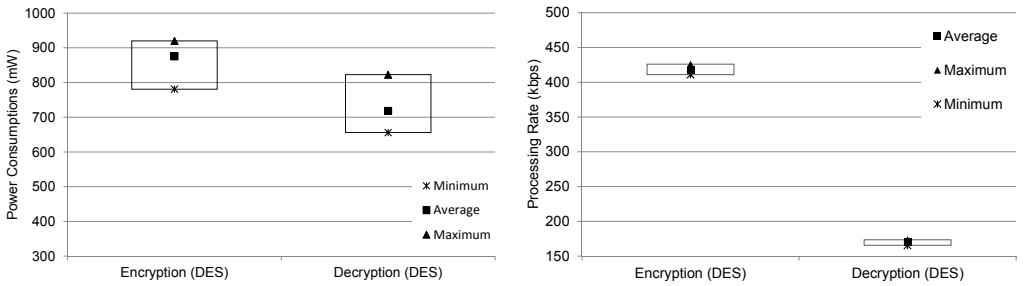


Figure 2.11: Power consumption for computation.

As we mentioned in the beginning of this section, the energy costs of security measures cannot be known until the end of the implementation phase. However, the processing

speed and corresponding energy cost for different block sizes help the designers to build energy efficient design. Fig. 2.12 shows the power cost information for the encryption and decryption processes on *HTC Nexus One* device.



(a) Power consumption for encryption/decryption.

(b) Obtained data rate for encryption/decryption.

Figure 2.12: Power consumption and data rates for encrypting and decrypting data.

Power consumption for reading and writing to storage media (specifically in *sdcard*) is given in Fig. 2.13. Data blocks of different sizes with random content were written to and read from the storage to measure the writing and reading speeds and power consumption. We observe that larger block sizes consume less power for the same reading or writing speed; however, applications will need more memory. The difference in power consumption is almost 100 mW between block sizes of 256 and 512 bytes. The designers need to make a trade-off between them.

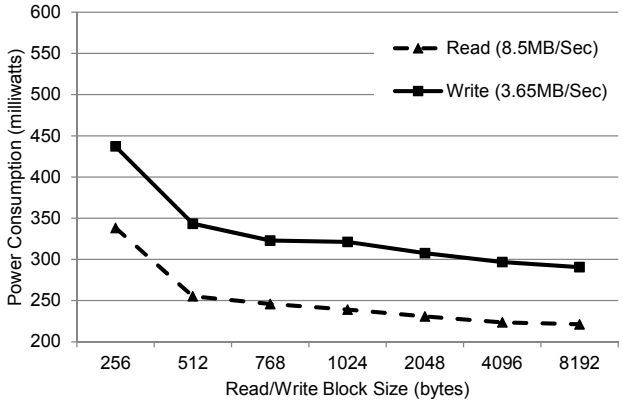


Figure 2.13: Power consumption for reading and writing data in the external storage.

The same trends were observed in case of communication components. For a fixed data rate, smaller data packets incur more protocol overheads, and consequently consume more power. Data packets of size around maximum transmission unit (MTU) are most economic. A detail study of the impact of size of packets can be found in [114]. In the presence of moderate bit-error-rate (BER), benefit of larger packet size can be offset by re-transmissions. We observe the power consumption by sending UDP data packets from the device to a laptop computer. The delays between two consecutive packets are varied to maintain fixed data rates. For example, the delay between two 256-byte packets is 2 milliseconds for 1 megabytes per second (mbps) data rate, and it is 32 milliseconds for two 2048-byte packets at 512 kilobytes per second (kbps) data rate. Figure 2.14 shows the power consumption. We observe that the difference in power consumption is about 40 mW for 256 and 1280-byte packets, and double data rate is obtained at the expense of only 20 mW. The differences in power consumption becomes significant when such an application runs for minutes or hours.

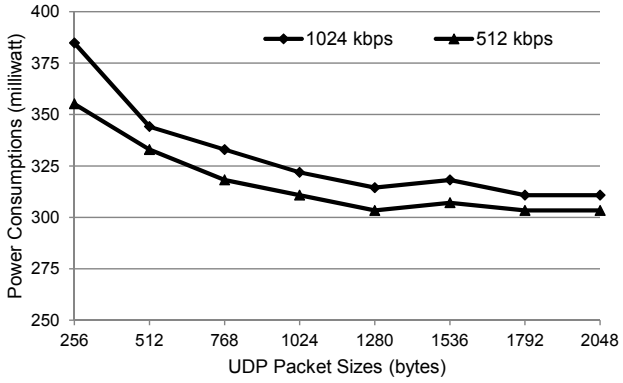


Figure 2.14: Power consumption for transmitting UDP data packets via WiFi.

We also conducted several experiments to compare the energy costs of downloading and uploading a file via 3G, WiFi, and Bluetooth links. The results shown in Fig. 2.15 appear to be counter-intuitive as downloading a file consumes more energy than uploading a file via 3G and WiFi links. This is due to the slow speed and power consumption for the decryption process involved with SFTP (Secure File Transfer Protocol) process.

This high-level energy cost information for all components of a device help the designers to get insights into how the energy costs vary with different application level parameters, and thus they come up with better energy efficient application designs.

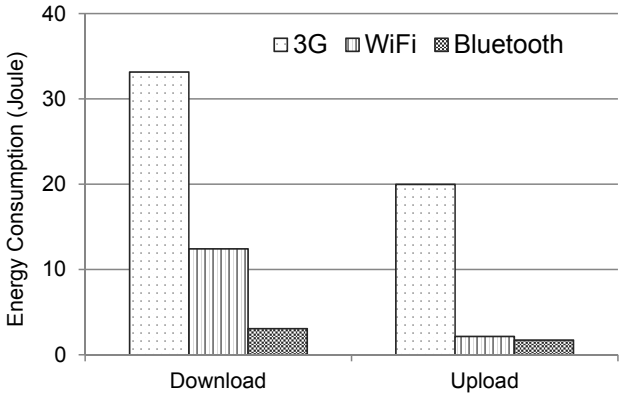


Figure 2.15: Energy consumption for transferring 1 megabyte of data.

2.5.4 An Example

In this section, we demonstrate the importance of energy cost profile at design time with an example. We developed an application to read data from external storage (*i.e., SD card*) of *HTC Nexus One* smartphone, and send the data after encryption with 56-bit DES (Data Encryption Standard) to a server located on the Internet. This application involves reading from storage, processing and data transmissions. To observe the energy consumption we considered block sizes of 256, 512, 1024, and 2048 bytes for reading, encryption and sending the data. We use 2, 4, 8, and 16 milliseconds intervals, respectively, between two consecutive data packets to achieve constant data rates. Later on, the interval between two consecutive packets is also kept fixed at 2 millisecond, which yield proportionately higher data rates for larger data packets. The data packets are sent via the WiFi link.

Figure 2.16 shows the measured energy consumed by the application for fixed and variable data rates. For fixed data rate, we see that 256-byte block transmission consumed 36% more energy than the 1024-byte block transmission. For variable data rate, 256-byte block transmission consumed 162% more energy than the 1024-byte blocks. Further increment in the block sizes requires more memory allocation without significant saving of energy.

We looked at the instantaneous power consumption to explain the energy consumption behavior while the application was executed. In case of fixed data rate transmission, larger packet sizes yield fewer packets transmitted over the same time. Since the number of packets are less, power consumption reduce for larger packet sizes. The phenomenon is evident in the first half of Fig. 2.17. For variable data rate, packets are sent at a fixed rate, and larger data packets draw more power. However, they take less time to complete the

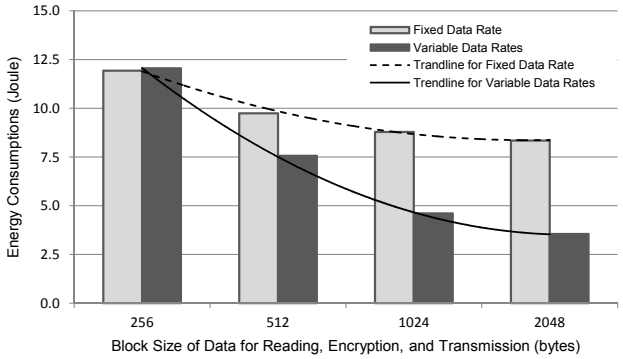


Figure 2.16: Energy consumption for transferring 2 megabytes of data.

transmission as the total data size is same for all cases. It may be noted that a device

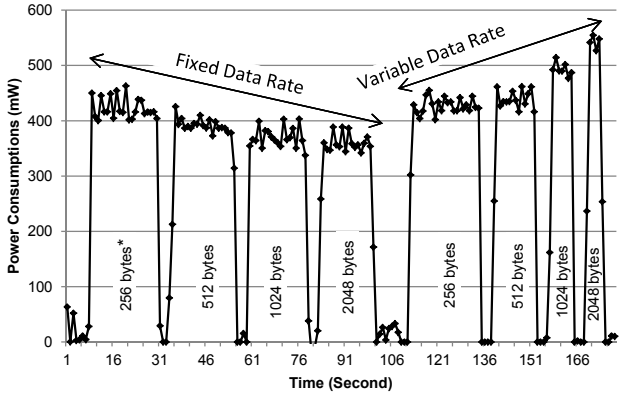


Figure 2.17: Instantaneous power consumption for transmitting data packets via WiFi.

consumes a certain amount of power to start running an application. This state is referred to as the *idle* state, and when an application actually executed it draws additional power on top of the power consumed in the *idle* state. For the *HTC Nexus One* smartphone, the power consumption in *idle* state is around 460 mW. On the other hand, a device having no application to execute can be put in *sleep* state. In the *sleep* state, a device consumes very low power, and it is about 10 mW for *HTC Nexus One*. Therefore, when an application can accomplish a task with less amount of time, it also helps reducing additional energy spent for keeping a device in *idle* state. For real time or interactive applications it is difficult to make such decisions.

* in Fig. 2.17 indicates the packet size used in the corresponding episode of transmission.

The energy cost outcomes that we obtain for the above example are fully compatible with the estimations and trend we have got in the *energy cost profile* in the last section.

2.6 Summary

We have presented a finite state machine based model to estimate the energy cost of mobile applications. We provided evidence from an energy measurement test-bench to explain the model parameters. We proposed to use smartphone device emulators to estimate the energy costs and discuss the rationale behind it. To help designers in making decisions at an early design phase, the concept of *energy cost profile* of a device was introduced. It provides energy cost information of different hardware components so that the designers take measures to improve their designs to make the applications more energy efficient. We have conducted experiments on our energy measurement testbed comprising a state-of-the-art smartphone to provide practical examples of energy cost profile of a device. Finally, we have shown the effectiveness of the whole idea with an example.

Chapter 3

Capability and Functionality Enhancement

We propose the concept of Universal Computing and Communication Interface (UCCI) that facilitates such sharing of resources between two wireless portable devices. This model comprises two basic components: Device and Connection Management (DCM) protocol and Framework for Information Exchange (FIX). *DCM* devises a unique way to save energy by allowing a server to stay in *sleep* state while its service is not needed. On the other hand, FIX enables software applications on a small device to use resources such as CPU, Internet bandwidth and storage available on a larger computer. We have used state-of-the-art smartphones *HTC Nexus One*, *BlackBerry 9700* and a laptop to develop prototypes of the proposed idea. We have conducted extensive experiments on the devices and measured the real-time energy consumption. This work explains situations under which such resource sharing can lead to energy saving. We also assessed the latency in accomplishing a task performed through sharing of resources.

3.1 Problem Description

In this section, we provide background and motivation for the work presented in this chapter. Then, we describe the system model, design criteria and research objectives of this work which are followed by the summary of contributions. We discuss the organization of this chapter at the end of this section.

3.1.1 Background

Smartphones are equipped with essential gadgets such as global positioning system (GPS), digital camera, and multiple communication interfaces. As a result, the functionality of these devices is not limited to exchanging voice calls, rather users use their phones to access email, browse Internet, and play multimedia contents. The features and functionalities of these devices are improving day by day with reduced size and price. Accordingly, the usage of these devices are becoming more and more common in daily life and user expectations in terms of running heavier applications are rising rapidly.

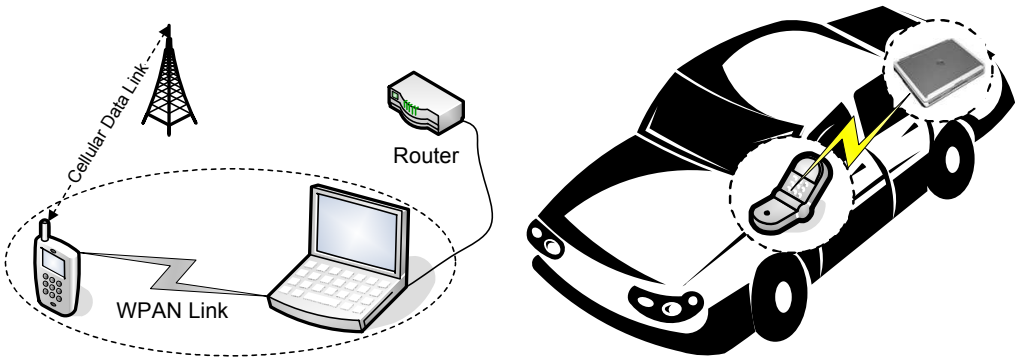
These devices are powered by small, re-chargeable batteries, and unfortunately, the growth in battery technology has not kept pace with the rapidly growing energy demand of these smart devices [120]. For example, the battery of a state-of-the-art smartphone lasts only $3-4$ hours when online video is played. A *GPS* aided navigation application runs around the same amount of time when it runs solely on battery. This dependence on battery energy puts a severe constraint on the availability of these devices [52]. Moreover, it is not feasible to equip the handheld devices with full-featured hardware components due to size and limited battery energy. Consequently, these devices are not capable of running resource intensive applications, which limits the functionality of these devices. In comparison to the size and weight of the smartphone, laptop computers are large and heavy, with relatively higher capacity CPU, battery, and communication bandwidth.

3.1.2 Motivation

Due to the complementary attributes of laptops computers and smartphones, nowadays professionals, business executives and university students use laptops as well as smartphones to meet their computing and communication needs. However, they often feel the necessity for sharing resources between these devices. For example, when they work on their laptop, they like to access some files, 3G data networks, or even the on-board camera of their smartphone; or, they may need to access the high capacity processor, high bandwidth data network available on the laptop while they work on their smartphone. Sometimes they need to access a licensed software from one device to another. Most importantly, they often need to share their data among these devices.

In Fig. 3.1(a), a smartphone user is connected to Internet through a laptop. The user might be interested to do it if the phone is not subscribed to cellular data services, or even with a subscription to data service, the user may still divert its Internet traffic through the laptop. Because the data service may not be available in some places or, the

connection speed might be poor, or the cost of the data service is high. Fig. 3.1(b) shows a scenario where a smartphone user is retrieving a file from his/her laptop as the laptop is not accessible for the time being. This may happen when the laptop is in the trunk of a car or in the airplane overhead cabin box, or, the laptop may run short of battery energy, so the user does not want to open it. The display on the laptops consumes around 36% of its total energy consumption [71], so it could be a simple case of energy awareness.



(a) A smartphone is accessing Internet through a computer.

(b) A user is retrieving a file from laptop.

Figure 3.1: Smartphone communicating with a laptop in two different scenarios.

Thus the shortcomings of resource constraints in portable devices can be overcome by sharing resources which results in functionality enhancement. In addition to that a device is able to save its battery energy by using resources of other devices instead of its own. In fact, energy saving was the most significant factor for offloading tasks from a mobile device to a server device [86, 55, 158, 53, 22]. The benefits sharing resources are threefold: (i) a device acquires some functionalities which it does not possess itself; (ii) it is able to access some resources which does not belong to it; and (iii) it saves energy. All of these benefits are very much appealing to the users, and the devices become more useful to them.

3.1.3 System Model and Design Criteria

As shown in Fig. 3.1(a), the participating devices can communicate with each other via multiple communication links such as wireless local and/or personal area networks. We refer a device as a server when it allows other devices (clients) to access its resources. A server can be a laptop, a desktop computer or some other dedicated device having computation and communication facilities. We assume that the server device permits the

clients to do so because they belong to same owner or the server device gets incentives for rendering services. In the following, we describe the design objectives to accomplish the task of resource sharing in energy efficient manner.

- *Communication Link:* Connection establishment is a prerequisite for sharing resources among participating devices, and the Internet can be used to establish such connections. However, the Internet is not available when a user remains outside of user's workplace, and also, integrity and security of the personal data always remain a major concern. Also, routing data through Internet where source and destination are in close proximity will unnecessarily burden the Internet. Therefore, a local communication link that requires low energy, and meet security constraints is suitable for this purpose. On the other hand, a wired connection requires a physical contact by cables, and it also limits the movement of a device, thus a wireless link is much preferred for this purpose.

- *Service On-demand:* A server is able to provide instant access to its resources when it remains available all the times. Consequently, it consumes energy to remain always *On* (further discussed in Section 3.3). Thus when a server is not used frequently, it wastes energy most of the times just to remain available. This situation can be avoided if a server becomes available only when its service is needed. We call it *service on-demand*. A client device awakes a server when it needs service. Here, saving of energy comes with a latency in getting service after a request is made.

- *Software Framework:* In addition to the communication link between client and sever, there must be an agreement among them so that they are able to communicate, and transfer data for sharing resources. There are diverse types of devices available in the market, and they come with different platforms (*operating systems*). Therefore, a generic framework (*cross-platform*) is essential to accomplish the task of resource sharing.

- *Energy Saving:* Sometimes a client accesses resources on a server to attain some functionality which that client does not possess, and in such scenario, energy saving is not an objective for both client and server. However, in some situations, a client accesses high capacity resources of a server to save its time and energy. In such cases, we assume that the server comes with an adequate supply of energy, resources. and we mainly focus on energy saving in the client device. The condition of saving energy is relaxed only when functionality enhancement is prime objective.

3.1.4 Research Objectives

We propose a generic architecture called *Universal Computation and Communication Interface (UCCI)* to enable communication between a mobile client and a nearby server device. Prior work advocates only for offloading a computational task to another device, but we have included the communication and sharing of resources in our model. We also consider the energy expense of the server device, because in our model the server itself can be a portable device running on battery, and the server device can remain in *sleep* state. *UCCI* consists of *Device and Connection Management (DCM)* and *Framework for Information Exchange (FIX)* protocols. *DCM* uses personal area wireless link (Bluetooth) to communicate a server, and it uses the 'Wake ON' feature of the server to *awake* it from *sleep* state to *active* state. *DCM* puts a server again in *sleep* state when service is not needed. *FIX* works on top of *DCM* and it facilitates the sharing of information and resources between a client and a server.

'Wake ON' feature allows a device to be turned on by a network message. Such a device can be kept in a very low power state by turning on the 'Wake ON' feature, and make the device available for use when necessary. We have conducted a set of experiments to observe the energy costs of a devices in different power consumption states with and without turning on the 'Wake ON' feature. Results show that the energy costs (overhead) for activating 'Wake ON' feature is very less.

We need to know the steps that take place in *UCCI* connection to evaluate the impact of resource sharing on energy consumption. The data processing rate and energy costs of basic operations such as communication, computation, storing and retrieving data from storage need to be investigated to estimate the costs of performing a task on a device. Then, we are able to compare the costs with and without resource sharing. Suppose that a smartphone, x accesses a resource, r (CPU) on a laptop, y. To accomplish this, x needs to follow the steps [55]: *(i)* needs to maintain a connection with y, *(ii)* reads data and related codes from its storage, *(iii)* sends them to y, *(iv)* waits for the results, and *(v)* receives the results. Data speed of the communication link between the devices and the data processing rate at device y contribute to the latency of the overall process, and the energy spent in transmitting and receiving data constitutes the energy cost. The reading and writing of data from and to the storage remain the same when a task is accomplished in device x. There is also some costs incurred in x for staying active while the task is processed in y, however, it can be avoided by properly scheduling the device x to awake up when the task is completed. We measured these costs on a state-of-the-art smartphone, *HTC Nexus One*, and discuss the possible scenarios when and how sharing of resources can be effective and efficient.

3.1.5 Contributions

The main strength of this work is that we have not only designed a model, but also have developed prototypes to show the validity of the model. We have implemented the system using the off-the-shelf devices. We summarize the principal contributions of our work below.

- (c_1) We introduce the concept of *UCCI*, a generic model for sharing resources among portable wireless devices. *UCCI* consists of two protocols *DCM* and *FIX* that enable any device communicate with another device without having prior knowledge of each other. when a device finds that it does not have the functionality or enough resource to execute a task or it intends to save energy, the device exports that task to a nearby server. when a device has multiple communication links available to it, the device can pick a suitable one to fit its need and for saving energy.
- (c_2) Two prototypes have been developed on Android and BlackBerry smartphones namely, *HTC Nexus One* and *BlackBerry 9700* to demonstrate the efficacy of the model.
- (c_3) To observe the energy saving aspects of *UCCI*, we conducted experiments to measure the application level energy and communication costs for different usage scenarios.
- (c_4) We discuss the experimental results and explain how and in what situations resource sharing can be effective, and save energy with less delay. We also discuss the various security aspects, and argue that the model does not give rise to any new security threats.

The rest of this chapter is organized as follows. In section 3.2, we discuss related work such as task offloading and low power personal area network communications. In section 3.3, we explain the working principle of our framework. Section 3.4 describes the prototype implementation of the proposed framework and it is followed by experimental setup in section 3.5. Section 3.6 presents the experimental results with discussions. We put conclusions in section 3.7.

3.2 Related Work

We review some substantial prior work in this section. We begin with discussing the benefits and costs of offloading followed by the different techniques or methods of offloading. We

also explain the reasons behind different design issues of our model. We finish this section with a discussion of the reasons for considering Bluetooth as the client to server device communication link.

Gitzenis *et al.* [52] studied the problem of task offloading and power management in wireless computing. They presented a Markovian dynamic control framework to optimize the task migration and processor speed/power management. They pointed out that task offloading results into energy savings at the mobile terminal (sparing its processor from computations) and execution speed gains due to (typically) faster server processor(s). However, the overheads are the energy cost for terminal-server wireless communication and the delay for uploading the task and getting back the results. The net gains (or losses) depend on network connectivity and server load. Their observation is: for a task with low communication and high computation requirements, migration is advantageous under both criteria of energy consumption and response delay. However, to accomplish this, the wireless connectivity must be strong and the server has to be lightly loaded. Weak connectivity turns migration into a less attractive option. In our model, we consider the short range personal area wireless link which is robust with relatively moderate bandwidth. For example, Bluetooth v2.1 supports 2 Mbps application to application data rate and Bluetooth v3.0 supports up to 24 Mbps. According to the observations in this research, our proposed model and its operating environment are suitable for task offloading.

Zhao *et al.* [174] studied a case where resource limitation forces offloading of a task. They propose a H.264 encoder modularization and energy models for offloading. They mainly focus on the usage of the computation offloading method to H.264 video encoder on mobile devices. Results from three of their offloading schemes show that offloading the encoding part of inter frames or the whole video encoder can save large amounts of energy. They observe that with efficient wireless link, computation offloading techniques would be more efficient to save energy on mobile devices.

Rudenko *et al.* [129] present an automation framework for computation offloading and it records the average power consumption of a repetitive task for deciding whether to offload the task. Kremer et al. [80] propose an offloading scheme that uses check-pointing techniques to handle disconnection events for wireless connection. The cost model for local computation is based on the average computation time. Rong *et al.* [126] study offloading under real-time constraints. They use multiple synthetic tasks and each task has a known constant computation time. Li *et al.* [86] propose making offloading decisions at function level. The computation of each function is assumed to be a constant and obtained by profiling. Wang *et al.* [153] propose a method of parametric compiler analysis to determine the computation time. The method considers only simple parameters, such as the command line options. It cannot analyze more complex data such as an image. All

these methods require estimating the computation time before execution in order to make offloading decisions. In contrast, Xian *et al.* [158] use a timeout method and do not require such estimation. In their study, a timeout is set for computation instead of an idle period. If the computation is longer than the timeout, the computation is offloaded to a remote server to conserve the energy for the client.

Gurun *et al.* [55] present a framework that makes computation offloading decisions in computational grid settings. The schedulers in such environment determine when to move parts of a computation to more capable resources to improve performance. They mentioned that offloading decision amounts to predicting the bandwidth between the local and remote systems to estimate costs associated with offloading. They formulated the problem as a statistical decision problem, and evaluate the efficacy of a number of different decision strategies. They found that a Bayesian approach which incorporates change-point detection in its formulation of the prior distribution is the most effective of those they investigated.

Gu *et al.* [53] propose an adaptive offloading system that includes two key parts: a distributed offloading platform and an offloading inference engine. They mainly focus on the memory. When the application memory requirement approaches the mobile device's maximum memory capacity, the system initiates offloading. The system partitions the application's program objects into two groups, offloading some to a powerful nearby surrogate to reduce the device's memory requirement. With the offloading inference engine, runtime offloading can effectively relieve memory constraints for mobile devices. Having studied the pros and cons of the methods of offloading, we designed a very light and simple Offloading Decision Maker (ODM) engine. At first, it does not partition a task running parts in the mobile device and parts in the server. Rather it decides before executing a task whether to offload or not. In case of offloading, the whole task is migrated to server and an interface is executed for sending data and receiving the results. In making the decision, *ODM* considers the applications resources requirements, availability of offloading service, system's resource meter and energy state and above all user's permission.

Mahmud *et al.* [92] emphasize on having an energy-efficient scheme for simultaneous or single operation of the wireless interfaces attached to Multi-service User Terminal (MUT). MUT stands for the devices that have multiple wireless interfaces for receiving various classes of services from the networks. They propose a simple model for predicting energy consumption in a terminal attributed to the wireless network interfaces. Then, the actual consumption patterns are measured to estimate the parameters of the model. They observe that each access technology has a different data rate, network latency, interaction capability, mobility support, and cost per bit because each has been designed with specific services in mind. They stress on the need to have comprehensive understanding of the

power consumption of the devices/modules in various operational states. Complying with this understanding we explored the energy cost of communication on different communication links that a smart phone typically possesses. And we use the results in designing our system model.

In mobile to server device communication we use Bluetooth link, because, Bluetooth is widely adopted as short range communication protocol and almost all the smart phones come with a Bluetooth connectivity. More importantly the next generation Bluetooth v3.0 module is going to be more energy efficient, more secure and is supposed to provide higher data rates of around 24Mbps. We summarize the strength of this work below:

- The concept of awaking a server *on demand* basis is very crucial. Instead of spending energy by keeping the server awake all the times, the server device saves significant portion of energy by being in *sleep* mode. We may compare the situation with constant polling versus interruption when the an event occurs.
- The idea of task exporting has been around for quite sometime and the design objective or target platforms were assumed to be grid or mesh networks. We view task offloading between peer devices, where functionality enhancement is the key objective and energy saving comes as a by product.

We have not only proposed a design or concept, we have implemented the whole system with existing hardware available in the market. This is the most important strength of this work.

3.3 Architecture

In our model, a device is termed as a client or a server based on its role or functionality in an ongoing session. A client device in a session may act as a server during some other session when another device seeks a service from it. So the terms client and server are not tightly coupled with a device. For example, when a smartphone accesses resources of a laptop, the laptop becomes a server and the smartphone becomes a client. On the other hand, the laptop becomes a client when it accesses the files of the same smartphone. In this case, the smartphone acts as a server. The working principles of the three *UCCI* components are given below.

3.3.1 Device and Connection Management (*DCM*)

A server device can stay in several states of operation as shown in Fig. 3.2 and it is able to provide service in *active* state. In *sleep* (or *inactive*) state, it suspends all of its operations except the 'Wake On' feature. 'Wake On' feature refers to the capability of waking up from *sleep* state to *active* state after getting a special message through a communication interface such as LAN, Wireless LAN (WLAN) and Universal Serial Bus (USB). To enable 'Wake On' feature, a device needs to scan for particular message while sleeping. When both client and server stay in the same network, a client device needs to know the Internet protocol (IP) address of the server device. However, a client needs to know the Medium Access Control (MAC) address of the server to 'Wake Up' the server by sending *magic* packet.

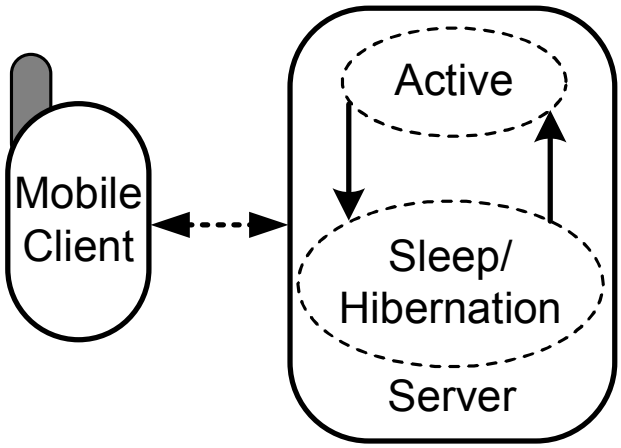


Figure 3.2: State transition diagram of server device.

Now, let us explain how a client obtains the address of a server device:

- Server in *active* state: An *active* server periodically broadcasts its service list with its device address. Before initiating a connection, a client scans for devices surrounding it and makes a list of devices. The user of the client is then asked to select the server from the server list.

A magic packet is sent to *wake up* a device from sleep state. This packet contains 255 in consecutive 6 bytes, followed by 16 repetitions of the target device's 6-byte MAC address anywhere within its payload. The magic packet is typically sent as a UDP datagram to port 7 or 9.

- Server in *sleep* state: When a server is in *sleep* state, it does not broadcast its address, and in this case a client needs to get the address from the user or from history data. If the server device belongs to the same owner, he/she knows the device address and for a public server, the address of the device needs to be printed on it. Otherwise, a client won't be able to wake it up. Usually, a client device keeps a short list of devices which are connected quite often, and thus a user does not require to input the server address all the times. As all the devices have user-friendly names, users are not expected to input bizarre hexadecimal device addresses.

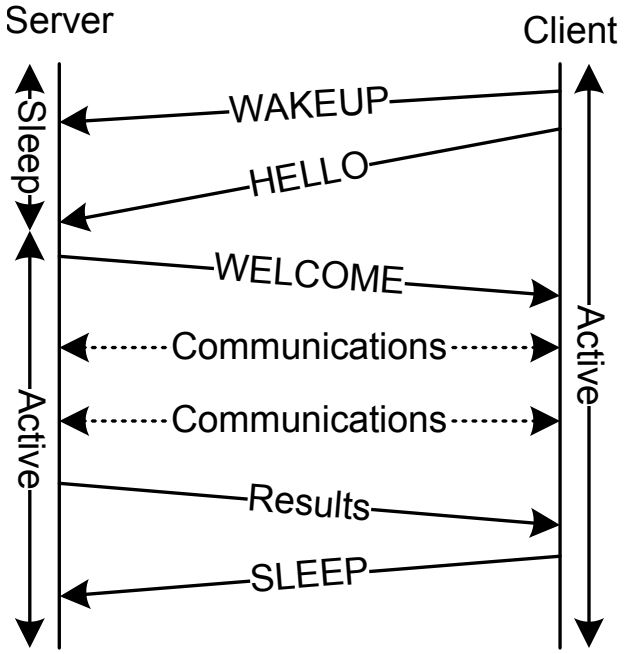


Figure 3.3: Timing diagram of task offloading using UCCI.

A client does not need to send a WAKEUP message to turn on the server as it is already in *active* state. However, client sends a HELLO message to check whether the server is ready to accept a connection. For an *inactive* server, a client sends a WAKEUP message to activate the server. The relative timing of the events is given in Fig. 3.3. After sending the WAKEUP message, the client waits for a while (10 seconds) for the server to come in *active* state. It then sends a HELLO message to check whether the server becomes active. The server replies with a WELCOME message. If the client does not receive a WELCOME message within a time frame (5 sec), it again sends HELLO message. In the HELLO message, the client mentions its identity and service request. Thus the server

transfers the controls to specific handlers. After completing the task, the client sends a SLEEP message to put the server in sleep mode.

Hardware and Software Requirements

The required hardware and software tools necessary to implement the *UCCI* model are already available in the marketplace and we now describe hardware and software requirements for server and client devices below:

- Hardware requirements for server: A server device is equipped with a low power short range high speed wireless communication module to communicate with the client. The server also has 'Wake On' feature so that it can be awaken from *sleep* state. Otherwise, a server needs to be ON all the times to make its services available, and for a portable device it is not affordable.
- Hardware requirements for client: A client device includes the same low power wireless communication module to connect with the server. It is capable of sending special 'Wake UP' signal using the device address of the server so that the communication module at the server can wake up the server from *sleep* mode. The client's module also capable of searching surrounding server modules and services.
- Software requirements for server/client: After establishing the connection between server and client, the server verifies identity of the client, and based on the requested service type, the server transfers the handle to the specific handler. In our implementation, we used Java and no specialized tools was necessary to develop the applications on the client and server sides.

Hardware components with energy saving features [116] are available in the digital system since long. Personal computers are equipped with 'Wake On LAN', 'Wake On USB' features. Some of the Bluetooth and WLAN hardware components have 'Wake On' capability and some Operating Systems (OS) such as Mac OS supports these features. However, 'Wake On Bluetooth' can easily be incorporated on the OS if hardware supports this feature. Another way to enable 'Wake On' option is by exploiting 'Wake On USB' feature. It is done by attaching an USB Bluetooth module in the device's USB port.

Energy Saving Measures

The graph in Fig. 3.4 depicts the benefits of *sleep/Wake-UP* model of the server. A laptop that we have used in our experiment consumes 19.2 and 14.4 watts in *active* state, with

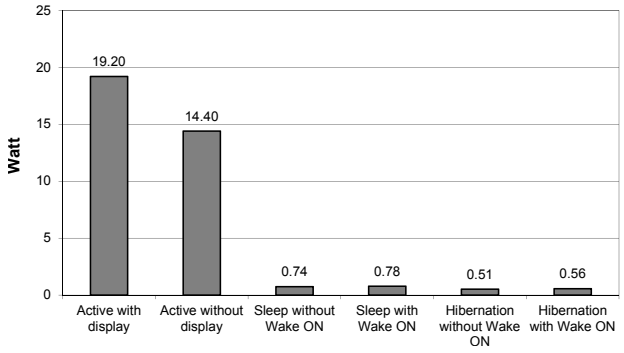


Figure 3.4: Power consumption of a laptop in different states.

and without display, respectively. Whereas, in *sleep* and *hibernation* states, the laptop consumes only 0.78 and 0.56 watts, which are just 4.06% and 2.92% of the *active* state, respectively.

Fig. 3.4 also shows that the increased energy consumption for turning on the 'Wake On' feature is only 5.4% and 10% more in *sleep* and *hibernation* states respectively. However, 'Wake On' feature is needed to put an *active* server to *sleep* state and put it back to *active* state when it is requested for service. As a result, it is definitely energy saving to adopt the *sleep/Wake-UP* model instead of being always *active*. It is worth mentioning that a device takes time to switch from *hibernation* to *active* or *sleep* to *active* states. These delays usually range from 5 to 20 seconds and they vary system to system. The delay for switching from *hibernation* to *active* state is longer, and the decision whether to put a server in *hibernation* or *sleep* state depends on the type of application. For quicker response time, a client needs to put the server in *sleep* state rather than in *hibernation* state.

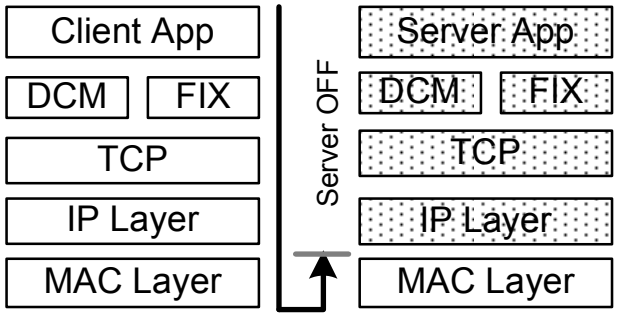


Figure 3.5: Placement of UCCI in protocol stack.

Placement of UCCI in OSI Model

Fig. 3.5 shows the position of *UCCI* in the network protocol stack. When a server device is in *sleep* state, a client can only reach its MAC layer by sending a WAKEUP message. After getting the WAKEUP message, server switches to *active* state and the client is able to access the server application, and further interactions take place.

3.3.2 Framework for Information Exchange (FIX)

The software applications such as remote file browsing, sharing, remote desktop sharing, Internet sharing, sharing camera or *GPS* data basically involve exchange of data. They are productivity or utility tools which are simple yet they provide the users of these devices with very useful services. One practical example would be very relevant here, users tend to take weeks to download the files of captured videos and photos from digital cameras. Because they need to connect devices to computers via cables. In the following, we describe how *UCCI* facilitates these types of tasks using *FIX*.

When a client connects a server device, it initially retrieves service information from the server. Then, the client device lays out the service list, and allows its users to choose an option as shown in Fig. 3.6. When a user selects an option, the client device sends the corresponding code to the server, and server execute corresponding module to further communicate with the client device.

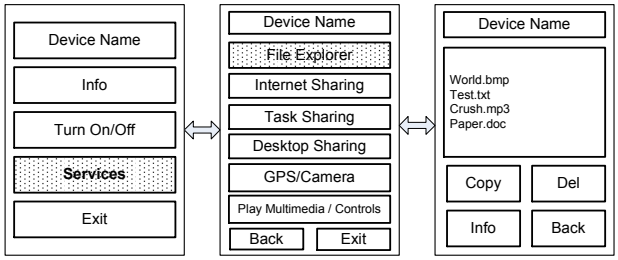


Figure 3.6: User interface of an UCCI based application.

Some software applications require extensive computation power and large memory capacity and claim good amount of energy. Such types of applications can be outsourced to take the advantage of CPU and memory on server devices. As we discussed in section 3.2, many techniques have been proposed in the literature regarding task offloading. For task offloading, the application need to have options for local and remote processing and when

user select remote processing, the application transfer and manage through application programming interface (API). In our prototype, we have used Java to implement task outsourcing feature.

3.3.3 Possible Security Issues

We point out the possible security threats that are involved in *UCCI* model, and discuss below how these issues can be resolved by taking appropriate measures.

- Authentication: The UCCI service should be restricted to the authorized users and devices. In our model, we consider Bluetooth which has built-in authorization module using PIN codes. Therefore, no user can connect to another device without knowing the PIN code.
- Secure communication link: As the data and tasks are migrated from one device to another, we need to ensure that the communication link is secure in the first place. Likely, secure personal area wireless communication is available and Bluetooth v3.0 supports 128-bit Advanced Encryption Standard (AES) which is state-of-the-art security protocol. However, encryption and decryption of data consumes much energy and its use is recommended only when there is a need.
- Software Security: When a task is transferred to a server, we need to ensure that it does not break server's security, and also no other application on server can intercept the data or code of the task. In UCCI, an exported task is executed on a Java Virtual Machine (JVM), and therefore, in the controlled environment, the task cannot break the security of the server, or any other application can access the information of others.
- Hardware Security: In our model, the server or client devices typically belong to the same owner, so the threat of hardware intimidation is less in this case. Hardware intimidation is not a new security threat arises from our model, and users need to be cautious about this when they connect to a public device.

3.4 Prototype Implementation and Model Validation

To implement prototypes for the proposed *UCCI*, we have used a *Toshiba Tecra R10-ES1* laptop as a server device with Windows 7 operating system on it. A *HTC Nexus One*

and a *BlackBerry 9700* smartphones were used as clients to build our prototypes. The smartphones communicate with the laptop through its built-in Bluetooth link. However, the operating system does not allow the Bluetooth device to *wake* it up from *sleep* state. We used an external mouse (*Microsoft Wireless Laser Mouse 8000*) to facilitate that. An USB dangle is attached with laptop and the mouse communicates with the dangle using Bluetooth link. The USB dangle connects the laptop as human interface device (HID), and thus the mouse is able to *awake* the laptop from *sleep* state. If the OS supports the 'Wake On' feature of the built-in Bluetooth device, the external mouse would not be needed. With an appropriate device driver, features of these two Bluetooth devices can undoubtedly be combined. However, we avoided that as we have developed only the prototype of our concept.

Figure 3.7: Snapshot of an Android based UCCI application.

We developed an application for server and two versions of the client application to implement *UCCI*. All applications are written in Java, and the client applications are modified to fit with Android and BlackBerry OS. A screenshot of the client application is given in Fig. 3.7. It shows Android OS version of the application which is running on the *HTC Nexus One* smartphone. The user interface of the BlackBerry version is the same and is installed on a *BlackBerry 9700* device. The applications are able to establish *UCCI* connections via both the WLAN and Bluetooth links. The 'Wake On' feature of the WLAN works only in presence of AP, and it is not supported in WiFi adhoc mode. Moreover, WLAN link becomes useless while the user on the road or in the place where there is no support 'Wake On' packet forwarding.

Using the application shown in Fig. 3.7, the smartphone (HTC Nexus One) wakes up the server (Toshiba laptop) from *sleep* state, and transfers data using Bluetooth link. After processing the data, the results are sent back to the smartphone, and the laptop is put back into *sleep* state again. In fact, once the laptop is connected with the smartphone, we can access and use the resources on it, it is just a matter of attaching appropriate software applications. We play audio/video, and control the volume on the laptop from the smartphone, and similar activities can be performed on the smartphone from a laptop.

3.5 Experimental Setup

As discussed in section 3.1, we need to know the energy costs of the basic operations such as computation, communication and storing data on a device, so that we are able to estimate the energy saving when sharing of resources takes place. If a device spends much energy in transferring data to the server, the net energy gain becomes less, in fact, it can be negative in some situations. In this section, we describe the experimental setup that we used to conduct experiments for evaluating energy costs.

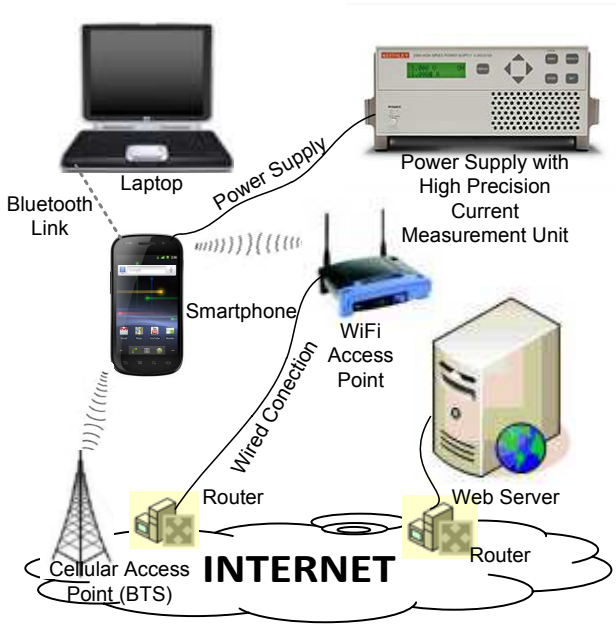


Figure 3.8: Logical view of our experimental setup.

As shown in Fig. 3.8, we connect the *HTC Nexus One* smartphone with a Keithley

2304A high speed power supply to measure the current consumption of the smartphones. The power supply is connected to a desktop PC via USB port, and using a controller program on that PC, we set required output voltage (3.7 volts) on the power supply. We sampled the consumed current by the smartphones one second interval and readings were taken throughout the execution of an operation. For example, before initiating a file transfer using SFTP (Secure File Transfer Protocol) we start probing current, and stop after the transfer stops. We repeat the experiments at least 3 times, and each time we take 5 sets of readings for each scenario. Each set contains 50 to 150 readings based on the duration of the operations. We show the variations in the readings by providing the maximum, minimum and mean values of the readings. Prior to conducting experiments, we configured the settings and battery connections of the smartphone as described in [113].

3.6 Results and Discussions

In this section, we present the energy costs of some basic operations such as computation, communication and data storage and retrieval for *HTC Nexus One*. Then we show the costs of transferring a file from the same smartphone to a server located in the Internet.

3.6.1 Energy Costs for Basic Operations

We chose three widely used CPU benchmark programs, namely, *Linpack*, *Whetstone* and *Dhrystone* to evaluate the cost of computation on *HTC Nexus One* smartphone. We used another Android application which computes the exponents of two random real numbers continuously. We executed these four applications for 20 seconds each time, and measure the current consumption. Fig. 3.9 shows the power consumption of the smartphone for each of the applications, and we see that the benchmark programs consume about 961 milliwatts. On the other hand, the exponent computation application consumes about 1100 milliwatts as it involves only floating-point operations. Once we know the energy cost of CPU for full utilization, the energy cost of an application for computation can be calculated based on its CPU utilization [116].

We obtained the energy costs of reading and writing in the built-in storage of the same smartphone by a simple application which reads and writes data in blocks of different sizes. We varied the block size from 256 bytes to 8 kilobytes, and the application reads a file of 256 megabytes, and write a file of 128 megabytes with random contents. The reading process takes around 30.5 seconds, and writing takes around 36 seconds irrespective of block size. The results in Fig. 3.10 show that smaller block size consumes more power.

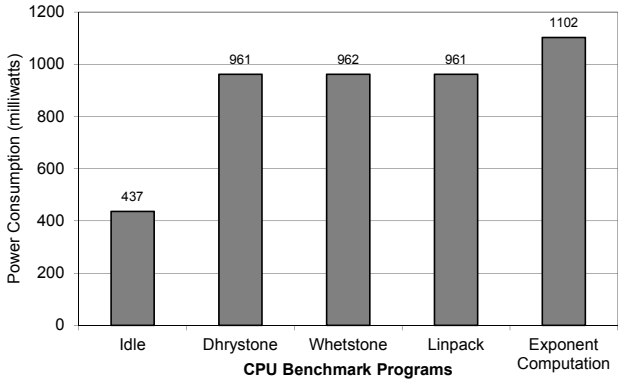


Figure 3.9: Power consumption for computation.

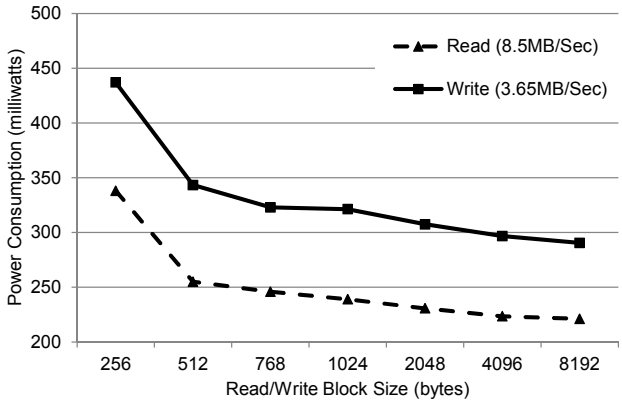


Figure 3.10: Power consumption for reading and writing data in the internal storage.

Fig. 3.11 shows the power consumption for transmitting UDP data packets of 1472 bytes via WiFi interface (802.11g). The experiment was done on the same *HTC Nexus One* smartphone. The data rates were varied as we changed the transmission interval between two packets. It is interesting to observe that for smaller transmission intervals the power consumption amounts are same. The maximum size of an unsegmented UDP packet that can be sent from application layer is 1472 bytes in this scenario. More information on the impact of packet size and delay on power consumption can be found in [114]. Though we have shown only for WiFi interface, the effect of packet length and intervals is also significant in other interfaces.

We also measured the cost of data encryption and decryption with an application that uses Java standard API for 64-bit DES (Data Encryption Standard) encryption and

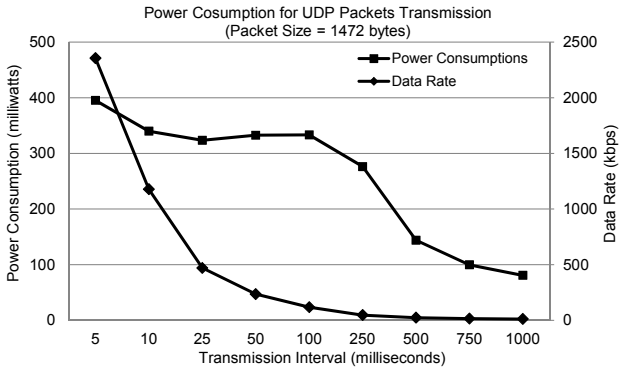
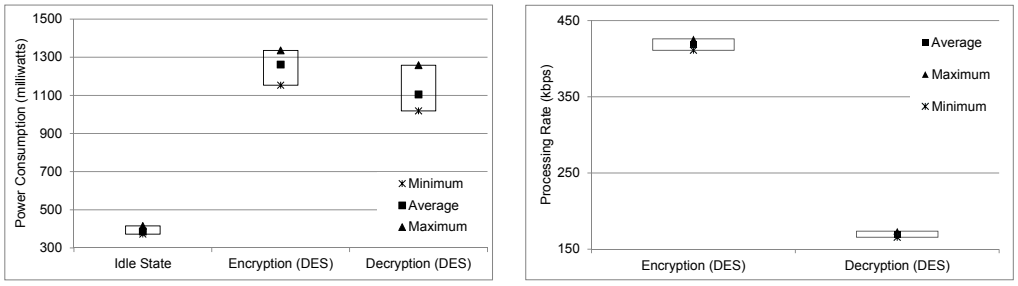


Figure 3.11: Power consumption for transmitting data packets via WiFi.

decryption algorithm. The application reads a large file of 32 megabytes and writes the same file after encryption. Later on, the encrypted file is read back by the application, and stored as a separate file after decryption. The power consumption and processing speeds of the processes are given in Fig. 3.12. We see that though the power consumption for decryption is less than the encryption process, the data processing speed for decryption is significantly less in compare to the encryption process. We see the impact of this during secure file transfer process that we discuss later in this section.



(a) Power consumption for encryption/decryption.

(b) Obtained data rate for encryption/decryption.

Figure 3.12: Power consumption and data rates for encrypting and decrypting data.

With the help of energy cost information presented above, we are now in a position to make energy-aware decisions. For example, it requires 1233 Joules of energy to store in the internal storage, and another 412 Joule to read the file when necessary. If we want to store the file in a storage server in the Internet, with basic cost information, we can calculate the required energy for transmitting and receiving the file over the communication

with appropriate security measures. Of course, storing in the network server involves more energy, specially when we need to access the file frequently. However, when the device is running short of storage space, we compel to store it on the network server.

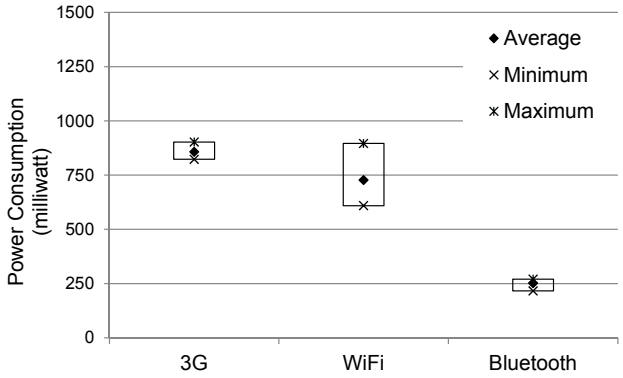
3.6.2 Energy Costs for Transferring a File

Now we investigate the energy consumption of a task which involves resource sharing via *UCCI*. We choose to transfer a file from the smartphone to a SFTP server located in the Internet. This task of transferring a file from the smartphone can be accomplished in the following ways:

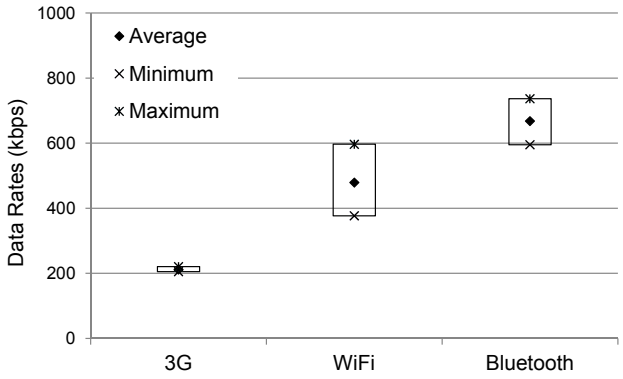
- **Scenario 1 (S1):** send the file from smartphone directly via 3G link,
- **Scenario 2 (S2):** send the file to the laptop via Bluetooth link, and then send it to the SFTP server from the laptop,
- **Scenario 3 (S3):** compress the file in the smartphone, and then send the compressed file via 3G link, and
- **Scenario 4 (S4):** send the uncompressed file to the laptop, compress it there, receive it from the laptop via Bluetooth link, and send the compress file via 3G link.

We measured the energy consumed by the smartphone in each of the above scenarios. For that, we, at first, evaluated the power consumption and data rates for 3G (WWAN) and Bluetooth (WPAN) links to estimate the energy costs for transferring data. We also measured the cost of transferring data over WiFi link as it falls in between WWAN and WPAN in terms of coverage. We conducted the experiments during different times of the day, and repeated the experiments to avoid any temporary fluctuations in the measurements. It is worth mentioning that these results are dependent on the location of the mobile tower, WiFi access point and local interference, but we took no measures that might affect the data rates of any of these links. Therefore, we may consider these results as an instance of a typical scenario experienced by users.

Fig. 3.13 and Fig. 3.14 show values of the mean, minimum and maximum power consumption and obtained data rates during downloading and uploading of files from/to the SFTP server. We observe that the data rates in uploading are higher than that of downloading, which is counter intuitive. We performed some additional experiments to verify this result. We used simple FTP protocol to transfer files, and found that data exchange rates are much higher in this case, and download rates are higher than upload rates. We



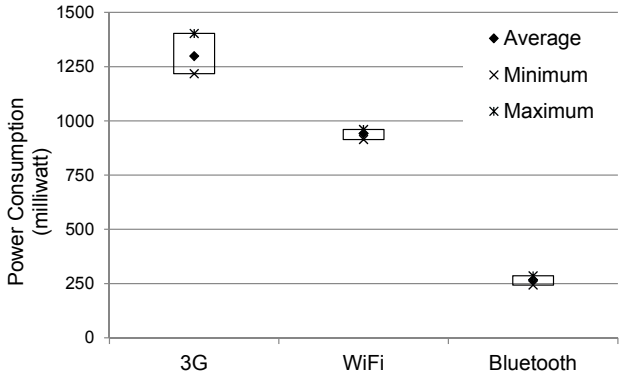
(a) Power consumption for downloading.



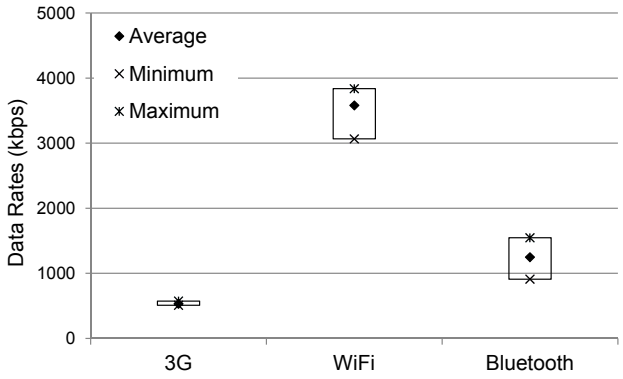
(b) Obtained data rate for downloading.

Figure 3.13: Power consumption and data rates for downloading data.

also examined the energy costs of reading and writing data from/to storage. We found that writing data on *sdcard* consumes 35% more power than reading data whereas writing is done at 42% of the reading speed. Actually, when we send a file through SFTP, the following operations take place: reading the file from storage, encryption the file data, transmission of the data. On the other hand, when we receive a file via SFTP - reception of file data, decryption of the received data, and writing the data on the storage take place. In SFTP process, operations other than the transmission/reception speeds dominate the overall data processing rates. Further study is necessary to investigate this matter. Data are more vulnerable in the WWAN or WLAN than in WPAN and for that reason we used SFTP to transfer data. Fig. 3.15 shows the energy expense for downloading and uploading a one megabyte file. We see that 3G link costs 10 times more than the Bluetooth link when we transfer the file. Data transfer costs over WiFi link is also lower than that of 3G



(a) Power consumption for uploading.



(b) Obtained data rate for uploading.

Figure 3.14: Power consumption and data rates for uploading data.

link.

To measure the energy costs for scenario S3 and S4, we measured the energy cost and data processing rates of compression and decompression process on the *HTC Nexus One*. As shown in Fig. 3.16(a), power consumption for compression and decompression are 1236 and 1184 milliwatts, respectively. The data processing rates during compression and decompression are about 2456 and 4818 kilobytes, respectively (shown in Fig. 3.16(b)). Using the results from the experiment, we are now able to calculate energy costs of transferring files via 3G, WiFi and Bluetooth links with or without compressing the data. Compression ratio is the most important factor in deciding whether to compress before transferring data. The compression ratios of JPEG (Joint Photographic Experts Group) and MPEG (Moving Picture Experts Group) format files (users typically encounter in smartphone) are very low and sometimes become negative even after using efficient compression algorithms

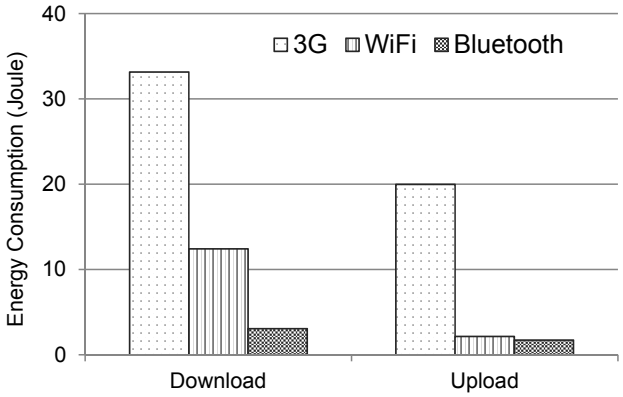


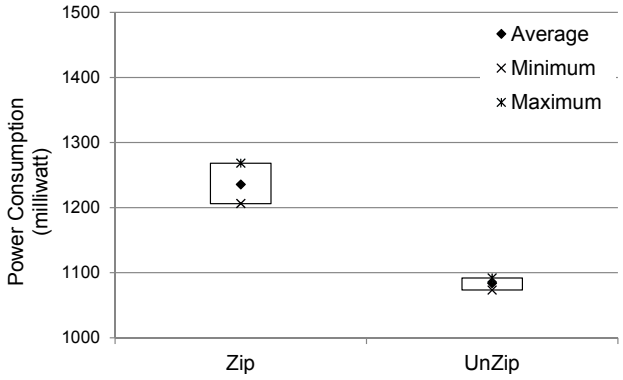
Figure 3.15: Energy consumption for transferring a file of 1 MB size.

[33]. In our experiment, we used Android's *ZipInputStream* and *ZipOutputStream* class for compressing *portable data format* (pdf) files and the compression ratio was only 12.5%.

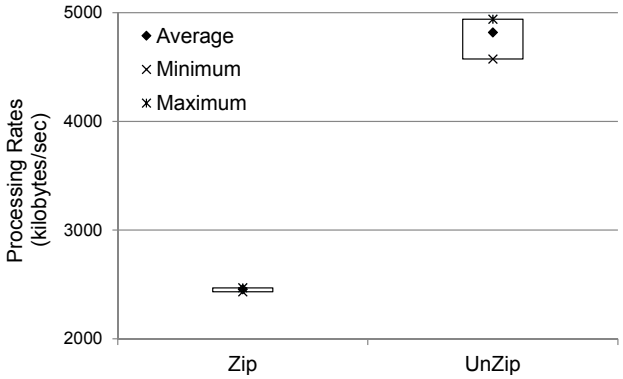
Table 3.1: Comparison of required time and energy cost for different scenarios

Scenarios	**Time Required** (Second)	**Energy Consumption** (Joule)
S1	76.99	99.93
S2	32.79	8.76
S3	69.45	90.02
S4	153.81	109.72

We calculated the energy costs and processing times for uploading a file of size 5 (five) megabytes using four scenarios mentioned in the beginning of this section. The results are given in Table.3.1. In scenario S2, the consumed energy is the least among the four scenarios. We assume that server has much higher Internet bandwidth, and we did not include the time to upload the file from *UCCI* server to SFTP server in scenario S2. In scenario S3, the cost is less than that of S1, but the difference is not significant. In fact, the compression ratio is a critical factor here, higher compression ratio yields better energy saving. The completion time of whole uploading process is also important. Presented energy costs are the costs uploading process only. During the file uploading, the device needs to be active which consumes additional energy (about 450 milliwatts on our smartphone). Therefore, energy cost in scenario, S4 is the worst in all respects. The data rates and



(a) Power consumption (milliwatt).



(b) Processing rate (kilobytes).

Figure 3.16: Power consumption and processing rate for compression/decompression.

energy consumption for WiFi link are much better than that of 3G link, and therefore, a secure WiFi link (if available) should be preferred over 3G link.

3.7 Summary

We have proposed *UCCI*, a generic architecture for facilitating communication between two wireless portable devices such as a smartphone and a laptop. This framework allows a server device to remain in *sleep* state unless its service is needed by a client device. It exploits the 'Wake On' feature of the server device and uses the personal area low power Bluetooth link, and then, optionally, puts the server back into sleep state. To validate our model, we have developed two prototypes using state-of-the-art *BlackBerry 9700* and *HTC Nexus*

One smartphones. We discuss the impact of this model by measuring power consumption on the client in different states through on-device experimentation. We compared the costs of transferring a file over 3G, WiFi, and Bluetooth links and measured the energy costs of data compression and decompression processes to show how they affect the energy budget of a smartphones.

Chapter 4

Anatomy of Smartphone WiFi Traffic

We analyze the WiFi access traffic of Android based *HTC Nexus One*, Apple's *iPhone 3GS*, and *BlackBerry 9700* smartphones for different classes of applications, namely, *web browsing*, *YouTube video playing*, and *Skype VoIP calling*. We set up a bench to capture the WiFi access traffic data of smartphone applications, and analyzed the data in terms of *packet size*, *packet inter-arrival time*, *burst duration*, *burst inter-arrival time*, and *burst size*. We discuss the implications of these observed parameters on existing *MAC* level energy saving techniques. Then, we propose a Low Energy Data-packet Aggregation Scheduler (LEDAS) that accumulates a number of upper layer packets into a burst at medium access control (MAC) level based on *formation time*, *size*, and *number of packets*. We have given a flowchart description of the technique. By means of analysis, we have derived expressions for the average values of *burst size*, *burst inter-arrival times*, and *number of packets* in a burst. Finally, we have evaluated the energy saving potential of LEDAS on a smartphone.

4.1 Problem Description

Today's smartphones are equipped with good processing capabilities, graphical user interface (GUI), and multiple radio interfaces, because of significant development in microelectronics technology. More capabilities are being built into these phones, and they are able to support resource intensive applications. These applications include web browsing, social networking, email client, online gaming, online multimedia playing, global positioning system (GPS) based navigation, and weather and stock updates. Due to these network related applications, smartphones draw a significant amount of wireless access traffic while

producing much uplink traffic. This traffic volume is growing rapidly and significantly faster than broadband traffic volume [157].

Though wireless access traffic of smartphones is routed through cellular and 802.11 based WiFi data networks, Universal Mobile Telecommunications System (UMTS) based 3G cellular data networks typically require more energy with less data rates compared to WiFi based networks [117]. Moreover, WiFi hotspots have become very common at homes, institutions and public places. Accordingly, smartphones use WiFi data link by default due to its accessibility, higher bandwidth, and less cost in comparison with cellular networks. Cellular data networks are mainly used while the users walk around or stay on transports. Analysis of residential digital subscriber lines (DSL) of a large European ISP shows that there is a significant and increasing number of active smartphone connections [93]. Therefore, novel energy saving measures need to be addressed for both cellular and WiFi based data networks.

Unfortunately, the growth in battery technology has not kept pace with the rapidly growing energy demand of these smart devices. The battery of a state-of-the-art smartphone generally lasts only $3-4$ hours when online video is played. A GPS aided navigation application runs around the same amount of time when it runs on battery. This dependence on battery energy puts a severe constraint on the availability of these devices. Therefore, there is a strong motivation for designing all aspects of smartphones from the perspective of battery energy [104].

Battery driven portable devices that are capable of saving energy, comprise of hardware components with energy saving features. Such a component has several operational modes, *i.e.* states, with different levels of energy consumption. For example, the communication component can have four operational modes, namely, transmission, reception, idle, and doze [114]. The power consumption levels in the reception and transmission modes are much higher than the doze and idle modes [137]. Intuitively, energy can be saved by keeping a transceiver in the doze mode for as long as possible, as determined by an operating system (OS). Similar energy-saving features are incorporated in processor design, display, and other hardware components [16].

The energy saving strategies for communication interfaces can broadly be classified into two categories: (*i*) inactivity threshold strategy, and (*ii*) micro power management (μPM). *Inactivity threshold strategy* is based on the principle that the longer a component has been inactive, the longer it will continue to be inactive [91]. When a user does not interact with a smartphone for a while, the device turns off the display and further saves energy by keeping all the hardware at minimum energy level. Only minimal interaction is maintained with the network to trace call and to update applications' status. A recent

study [45] revealed that users interact with their smartphones in a bursty manner, and each session of interactions (*usage burst*) generally lasts for $10 - 250$ seconds. A smartphone starts a timer to observe the idle period after each usage burst, and goes into idle mode to save energy once the period exceeds a preset threshold.

In contrast to the *inactivity threshold strategy* discussed above, *micro power management (μPM)* is applicable during a usage burst, and this technique deals with keeping a component in low energy state (*pause* or *doze* mode) for a very short interval so that the functionality of the device is not compromised. The fundamental requirement for these strategies is the ability of the related hardware component to go into low power mode for a very short period of time. The short interval is in the range of microseconds to couple of milliseconds [16, 89]. This strategy enables a communication interface to enter into power-saving mode even between two medium access control (MAC) frames. The gaps between successive frame exchange can further be extended by aggregating couple of data packets into one MAC frame, which make room for hardware to be in low energy state for longer period of time [114, 78, 176]. Though the concept of packet aggregation on the device side is new, various forms of traffic aggregation have been used in data backbone networks, and new techniques are being proposed to achieve higher bandwidth and energy efficiency [98, 143].

The process of packet aggregation adds delay to data packets and creates packets of larger size than that of the original packets. However, some kinds of traffic such as VoIP and real-time multimedia are very much delay sensitive. For each type of communication interface, there is a threshold value for data packets, known as *maximum transmission unit* (MTU). For example, a 1500-byte packet is the largest packet allowed by Ethernet at the network layer [148]. If the size of a data packet to be transmitted exceeds MTU size, that packet is segmented into multiple packets. As a result, it is not advantageous to have an aggregated packet whose size is more than MTU; rather, it causes more overheads. Therefore, while applying μPM, clear understanding of the nature of traffic and distributions of the inter-arrival times and sizes of the network data packets is required to maintain a certain level of performance and quality of service (QoS).

In this chapter, we present an in-depth insight into the characteristics of wireless access traffic, which will spur the development of *micro power management* strategies for the WiFi communication interface of smartphones. We have collected wireless access traffic data of smartphones for some representative applications [44], and studied the characteristics of individual packets as well as packet bursts for both uplink and downlink traffic.

We gathered the Internet traffic data of three state-of-the-art smartphones, namely, *HTC Nexus One*, *iPhone 3GS*, and *BlackBerry 9700*, connected to Internet via WiFi net-

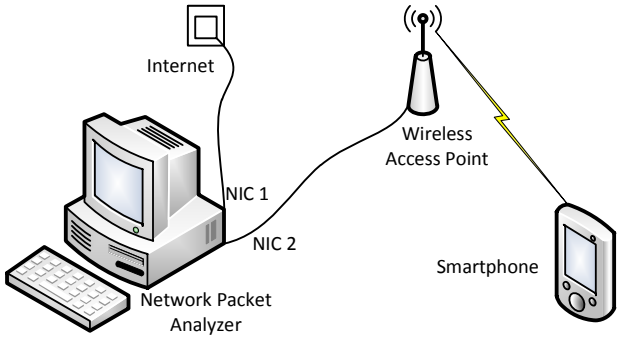


Figure 4.1: Connection details of network packet probing setup.

work. We route the traffic from and to a smartphone via a desktop computer as shown in Fig. 4.1. The WiFi *access point* (AP) is attached to the computer and only one smartphone is associated with the AP at a time. Therefore, all data packets traveled to and from smartphone can be sniffed by the packet analyzer running on the desktop computer. The traffic data for 3G networks could be collected at the cellular base station (BTS) or by placing a packet probing application in the smartphones. However, we did not have access to the BTS of cellular networks or have packet probing applications for the smartphones. The analysis of wireless access traffic in 3G networks is beyond the scope of this work, and we aim to investigate that in future.

We carried out *web browsing*, *YouTube video playing*, and *Skype VoIP calling* on the smartphones. A *network packet analyzer* installed on the desktop computer captured all the data packets. In the dataset, we observed some network management packets originated from the WiFi AP and we exclude them from further processing. We examine the distribution of *packet sizes* and *inter-arrival times* of the packets in uplink and downlink traffic. The packets are grouped into *bursts* based on their *inter-arrival time*. Then, we compute the distribution of *durations, inter-arrival times* of the bursts, and *number of packets* in each burst. As we discuss in Section 4.6, the parameters are very important in designing energy saving techniques for communication interface of a smartphone. For example, the duration of bursts is crucial in designing packet aggregation techniques at the MAC level. This work makes the following contributions.

- We present a test bench to capture and analyze the wireless access traffic generated by smartphone applications.
- We run representative applications on smartphones and analyze the characteristics of the wireless access traffic over the wireless network by means of *duration, inter-arrival*

time, size of burst and *number of packets per burst.*

- We have collected traffic data from *iPhone 3GS*, *BlackBerry 9700* and *HTC Nexus One*, and observed whether there is any significant difference in the distributions of the above mentioned parameters.
- We observed that the size of almost 80% of the uplink data packets is less than 66 bytes, and about 60% of the downlink packets have less than 1 millisecond inter-arrival time. For uplink traffic, it is about 50%. The inter-arrival times of bursts in both directions is more than 10 millisecond for more than 90% of the bursts.
- Based on the above observations, we have identified opportunities to design new energy saving techniques specifically tuned for smartphones.

The rest of this chapter is organized as follows. We review some substantial work that focus on energy saving measures for communication module in Section 4.2. The details of our test bench is described in Section 4.4, analysis of the results is given in Section 4.5. We propose a packet aggregation scheduler at the end of this chapter.

4.2 Related Work

We discuss the relevant prior work in energy saving in wireless networks. One paper proposes a scheme for optimizing inactivity periods of a mobile device in $3G$ cellular networks, and the rest of the papers focus on 802.11 based infrastructure wireless networks.

Yang [163] investigated the discontinuous reception (DRX) mechanism for saving energy of mobile devices in the Universal Mobile Telecommunications System (UMTS) networks. The DRX mechanism is controlled by two parameters: the inactivity timer threshold t_I and the DRX cycle t_D. Based on a M/G/1 queueing model with vacations, the author studies the optimal values for t_I and t_D which maximize the energy saving in mobile devices. The author also presents an adaptive algorithm called dynamic DRX (DDRX) that dynamically adjusts the values of t_I and t_D close to the optimal values.

Liu *et al.* [89] proposed micro power management (μPM) scheme that works in the client devices. It allows a WiFi radio to sleep for short intervals, in the range of a few microseconds. The communication interface can be used to sleep even between two MAC frames to save energy. The μPM technique uses predictions to exploit short idle intervals, and it relies on 802.11 retransmissions to recover from any mis-predictions.

Tan *et al.* [147] proposed an application-independent protocol, called *power save mode* (PSM) throttling. This transport level technique reshapes TCP traffic into periodic bursts with the same average throughput as the server transmission rate. Clients accurately predict the arriving time of packets, and turn on/off the wireless interfaces accordingly. PSM-throttling can minimize power consumption on TCP-based bulk traffic by effectively utilizing available Internet bandwidth without degrading the performance of application perceived by the user.

Rozner *et al.* [128] claimed that depending on the PSM implementation strategies, traffic of the other clients in the same network (competing background traffic) causes upto 300% more energy consumption in a client device. Moreover, the capacity of a wireless network reduces due to the unnecessary retransmissions and unfairness. They propose Network-Assisted Power Management (NAPman) algorithm for WiFi devices that addresses the above issues. NAPman distinguishes traffic of a PSM client from the traffic of competing constantly awake mode (CAM) clients and other PSM clients. Then it enforces a work-conserving first-come-first serve (FCFS) policy only to the packets of clients that are awake at any given time. This energy-aware fair scheduling minimizes client energy and unnecessary retransmissions.

Agrawal *et al.* [2] proposed an algorithm named Opportunistic PSM (OPSM). This scheme is effective when all the connected devices are engaged in web browsing, which is characterized by small file downloads over TCP, with a short duration of inactivity or think time in between two downloads. It performs better than static PSM when the number of associated devices with an AP increases. The reason behind the improved performance is that OPSM only permits one download at any time, due to which a device gets the maximum throughput and this results in least energy consumption. In static PSM this is not the case, since it allows simultaneous downloads, which leads to longer file download times and hence consumes more energy than OPSM.

Kim *et al.* [78] present a MAC level frame aggregation scheme, which can improve the throughput performance. By aggregating small-size frames into a large frame, it reduces MAC and PHY layer overheads. Their measurement results show that the throughput performance can be improved by 2 to 3 Mbps by applying the frame aggregation technique in the IEEE 802.11*b* standard. They have also proposed that frame aggregation can easily be performed above the MAC service access point (SAP) easily with device driver modifications. This work dealt with the impact of frame aggregation on throughput performance, and they did not consider any traffic pattern or impact of frame aggregation on QoS.

Zhu *et al.* [176] address the power saving problem by developing a model for stochastic analysis of timer-based power management in infrastructure WLANs. Based on this

model, the probabilities that a device is active, idle, or dozing are derived, and the power consumption of the device, number of frames buffered, and average delay per frame are obtained. This scheme produce bursty traffic to keep the communication interface in doze mode for longer period of time. However, it does not reduce the MAC/PHY layer data overheads. Specifically, the PHY layer overhead is very much significant in WLAN.

Nath *et al.* [106] proposed to transmit multiple beacons, one for every client associated to the AP. Each client estimates the round-trip-time (RTT) of the current TCP connection and sends this information to the AP, based on this information the AP schedules the beacon frames to the clients.

Ra *et al.* [122] mention a class of applications that is often naturally delay-tolerant so that it is possible to delay data transfers until a lower-energy option becomes available. They present an optimal online algorithm for energy-delay trade-off using the Lyapunov optimization framework. Their results show that their algorithm can be tuned to achieve a broad spectrum of energy-delay trade-off, and it can save $10-40\%$ of battery energy for some workloads.

Our work explores the characteristics of wireless access traffic which must be taken into account in designing micro power management (μPM) techniques for smartphones. Smartphone's communication hardware is capable of being in doze mode for short intervals (as low as 4 milliseconds), and this feature can be utilized even when the device interacts with its user. In this work, we investigate the requirements of traffic patterns of network related applications, and suggest guidelines for developing novel energy saving techniques.

4.3 Selection of Applications and Performance Metrics

To get representative statistics of the wireless access traffic of smartphones, it is very important to consider a set of relevant applications that constitute smartphone traffic. We have considered a set of applications according to usage rating given in [44]. In this section, we also describe the metrics that we measured from the traffic data.

4.3.1 Chosen Applications

We selected three state-of-the-art smartphones, namely, *HTC Nexus One*, *iPhone 3GS* and *BlackBerry 9700* which run on the most popular mobile operating systems (OS). Then, we

chose a set of representative network related applications on smartphones to gather traffic data [44]. The applications are: (*i*) random web browsing, (*ii*) social networking website, *facebook.com* browsing, (*iii*) *YouTube* video playing, and (*iv*) *Skype* VoIP calling. In case of web browsing, we randomly browsed news websites such as `cnn.com` and `www.cbc.ca/news/`, searched in `google.com`, and accessed emails on `gmail.com`. During *facebook.com* browsing, we followed links, status on *friends' wall*, and viewed photos. We played two videos on *YouTube.com* for online video data, and made VoIP calls using *Skype* to gather VoIP trafiic. The duration of each of the tasks was about 10 minutes. Though we collected data from three different smartphones, in this work, we used the data obtained from HTC Nexus One handset, and only a subset of the results is shown due to lack of space.

4.3.2 Performance Metrics

In the traffic data, we intended to observe the distribution of packet inter-arrival times, packet sizes, and burstiness of traffic. We were mainly interested to see the attributes of the bursts. Fig. 4.2 shows different parameters of a burst.

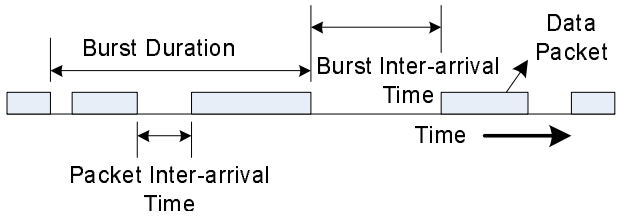


Figure 4.2: Schematic diagram of performance metrics.

We explain the performance metrics and discuss their importance below.

- *Burst Duration* is the difference of arrival times between the first and last packets in a burst. We do not consider the duration of a burst containing only one packet.
- *Burst Size* is the sum of sizes of all data packets in a burst. The packet includes application data, and headers of transport, network and MAC levels. The MAC level header size is 14 bytes in all cases as the packets are captured at wired portion of the link.
- *Packets per Burst* is the total number of packets in a burst.

- *Burst Inter-arrival Time* is the time gap between the arrival of the last packet of a burst and arrival of first packet of the next burst is referred as the burst inter-arrival time.

A burst containing a couple of data packets, and with less duration can easily be aggregated at MAC level before transmission. Aggregation process reduces MAC and PHY level overheads. A communication interface can utilize the larger burst inter-arrival times by being into low energy mode during those intervals. Energy saving by overhead reduction, and by staying longer in low energy mode are our concerns.

4.4 Experimental Setup

The network connection details for collecting traffic data packet information from a smartphone is shown in Fig. 4.1. A 802.11*g* based access point (AP) is connected to a network interface card (NIC) of a desktop computer which is connected to the Internet through another NIC. Internet connection is shared between the two NICs of the desktop computer. If only one smartphone is connected with AP, most of the packets captured by the packet analyzer, originate from the smartphone. There is a small number of network management packets exchanged by the AP, and those packets are discarded during analysis.

As a packet analyzer, we used *Wireshark* (`http://www.wireshark.org/`), an open source and widely used network packet analyzer in the industry and educational institutions. *Wireshark* does not manipulate things on the network, it only examines packets on a network. To collect smartphone's Internet traffic data, we connect one smartphone with the AP at a time, and run an application. Then we collect the packet information from the *Wireshark*, installed on the computer. The captured data packets are exported as spreadsheet from *Wireshark* for further analysis. Then, the packets are separated into uplink and downlink packets based on the source and destination IP addresses.

One or more packets are marked as a *group* when the inter-arrival times between two consecutive packets are less than a *threshold* value. Each of the groups is referred to as a *burst*. A burst may contain only one packet if the time gaps with its previous and next packets are more than the *threshold*. The time span between the first and the last packets in a burst can be any positive time period. To the best of our knowledge, the state switching timing (active to sleep and sleep to active) of state-of-the-art device circuitry is around 4 milliseconds. Therefore, the threshold value is set to 5 milliseconds.

As we collect traffic information at the desktop computer, a data packet travels via AP from a smartphone before arriving at the analyzer. Thus, the AP adds some delay to

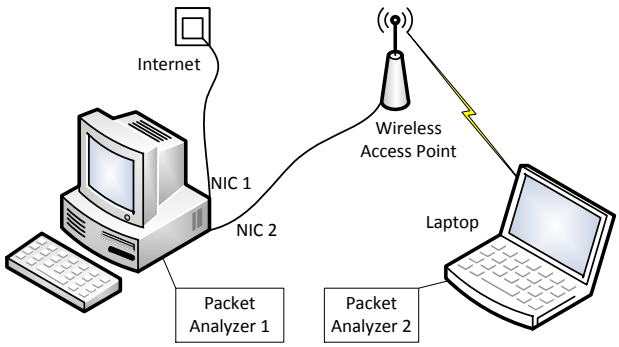


Figure 4.3: Connection setup for verifying the impact of access point (AP).

each packet and that may not be uniform for all packets due to buffering effect. Presence of uneven delays corrupt the timing of packet inter-arrival times, and to investigate this effect we use a laptop computer with another packet analyzer on it (Fig. 4.3). We run applications on laptop computer and gather packet information at both analyzers and compare the statistics obtained from both sources. The results are discussed in the next section.

4.5 Observations and Discussions

The information regarding sizes and inter-arrival times of uplink and downlink data packets is crucial for designing an effective *micro power management* strategy for smartphones' communication interfaces. With this objective, we at first compare the total number of uplink and downlink packets for different application scenarios. Then we show the distribution of size and inter-arrival time of individual packets in uplink and downlink traffic. The same attributes of bursts are discussed later in this section. Finally, we discuss the impact of the AP and operating systems (OS).

In social networking website (*facebook.com*) browsing, the amount of uplink packets is about 80% of the downlink packets, and in case of random web browsing, it is in the range of 70% to 95%. The numbers of packets in uplink and downlink traffic are almost equal in VoIP traffic. Only for *YouTube* video traffic, the amount of uplink packets is only 18% of the downlink traffic. Thus, the amount of uplink packets is significant as compared to the downlink packets except for *YouTube* video traffic.

The size of the packets in the uplink traffic is usually smaller, which are mostly ACKs (80% for random web browsing). For downlink traffic, about 60% of packets are of above

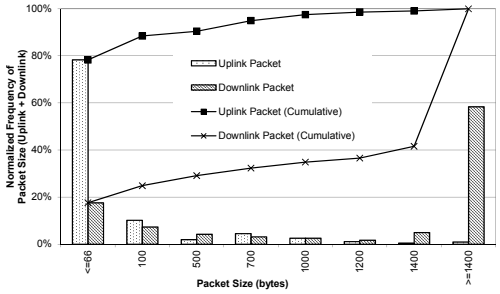


Figure 4.4: Distribution of uplink and downlink packet size for random web browsing.

1400 bytes in random web browsing, and almost all packets are of MTU size in case of *YouTube* video playing. The distribution of the packet size is given in Fig. 4.4. The uplink and downlink packet size in *Skype* VoIP traffic is around 71 bytes. A significant portion of the packets in both uplink and downlink traffic have very little inter-arrival times. As shown in Fig. 4.5, more than 45% of the uplink packets have inter-arrival time of below 1 millisecond, and above 60% of the downlink packets have inter-arrival time of less than 1 millisecond. In the following paragraphs, we present the characteristics of *bursts* formed by the uplink and downlink traffic.

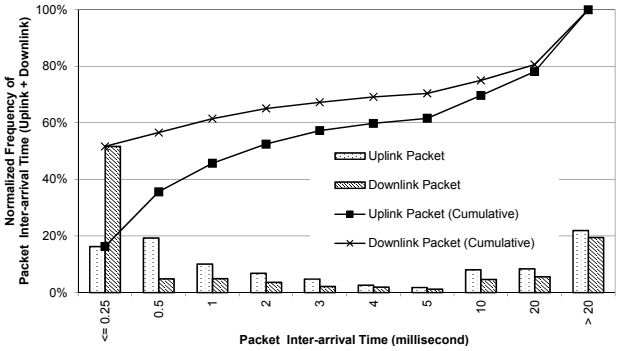


Figure 4.5: Distribution of uplink and downlink packets' inter-arrival time for random web browsing.

Burst Duration Fig. 4.6 shows the distribution of *burst durations* of uplink traffic. More than 95% of the bursts last less than 10 milliseconds. Almost 40% of the bursts live at best 1 millisecond. This phenomenon is very much interesting. This kind of shorter burst duration also indicates that packets within a burst are independent of each other.

No response is required from the destination node before sending all the packets in a burst. We observe exactly the same trend with the downlink traffic.

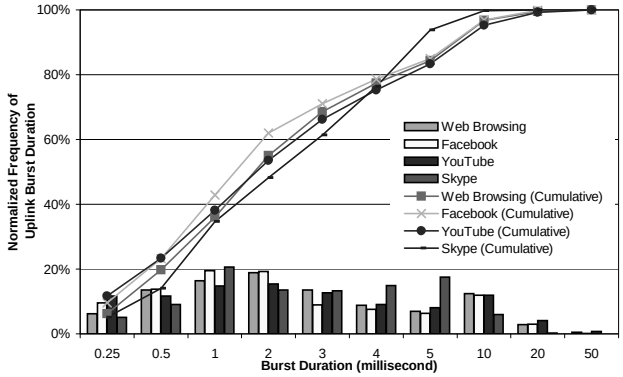


Figure 4.6: Distribution of uplink burst durations.

Burst Size The distribution of uplink *burst size* is given in Fig. 4.7. Most of the burst sizes fall below 1500 bytes. Almost 70% of the bursts formed in *YouTube* uplink traffic are single-packet bursts of 66 bytes in length. Those are basically transport level ACKs. Again, for *Skype* uplink traffic all bursts contain one packet of size around 70 bytes. In more than 50% of the cases, the *burst sizes* in uplink traffic for web browsing fall in between 200 to 1000 bytes. The *burst sizes* in the downlink traffic are larger compared to sizes of

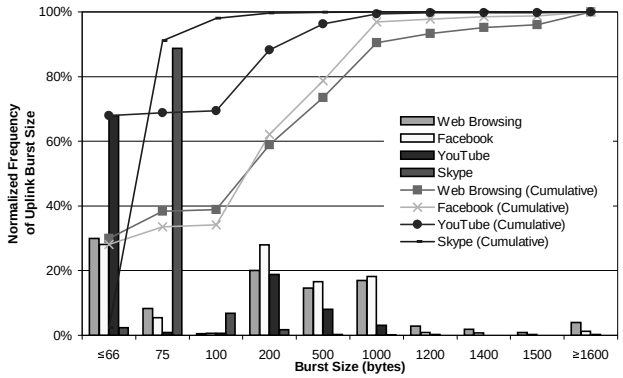


Figure 4.7: Distribution of uplink burst sizes.

uplink traffic (shown in Fig. 4.7 and Fig. 4.8). For web browsing and online video playing, the burst size generally remains above 2000 bytes. The burst size becomes as large as 50

kilobytes for YouTube video traffic. This differences in burst sizes of uplink and downlink traffic require different strategies for energy saving in both directions.

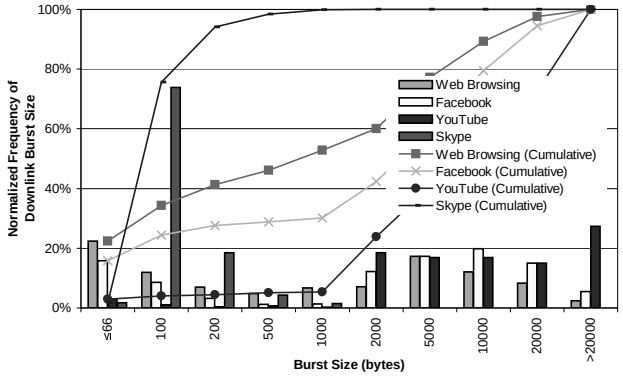


Figure 4.8: Distribution of downlink burst sizes.

Packets per Burst The distribution of *packets per burst* in uplink traffic is given in Fig. 4.9. More than 50% of the bursts contain 2 or more packets during web browsing. In *YouTube* uplink traffic, about 70% of the bursts contain single packet, and almost all bursts are single packet burst for *Skype* uplink traffic. In case of downlink traffic, similar trend is observed for web browsing and *Skype* calling. However, *YouTube* traffic contains 25% single packet burst, and 25% bursts with more than 25 packets. This bursty traffic is an indication of energy saving measure at transport level, which provides more idle time to the communication interface.

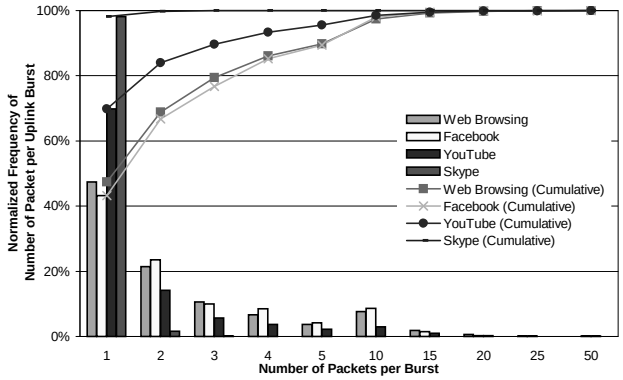


Figure 4.9: Number of data packets in uplink bursts.

Burst Inter-arrival Time Fig. 4.10 shows the distribution of *burst inter-arrival times* for downlink traffic. Only 15% of the bursts has less than 10 milliseconds of inter-arrival time, and 60% of the bursts have inter-arrival times of more than 20 milliseconds. Same trend is also observed in uplink traffic.

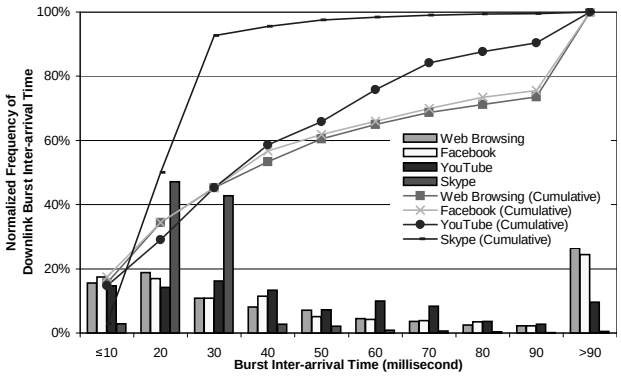


Figure 4.10: Distribution of downlink burst inter-arrival times.

The results discussed so far are based on separate uplink and downlink traffic data, and separate analysis is required for packet aggregation techniques. However, the communication module of a smartphone needs to be active for sending and receiving of data packets, and therefore, we examine a dataset which contains both the uplink and downlink data. Fig. 4.11 shows the distribution of *burst inter-arrival times* in both directions of traffic for *webpage* and *Facebook* browsing. Around 30% of the bursts have inter-arrival times of less than 10 milliseconds. In case of web browsing, half of the bursts come more than 20 milliseconds apart.

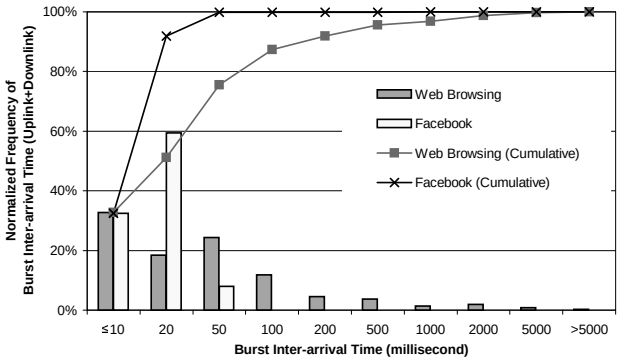


Figure 4.11: Distribution of burst inter-arrival times in both directions.

Placement of Packet Analyzer To observe the effect of AP on the route of traffic data, we used a laptop with packet analyzer on it instead of a smartphone. We browsed random web pages, and captured packets on the laptop as well as on the desktop computer as usual (Fig. 4.3). On the packet analyzer of the laptop, 40% of the packets in the uplink traffic have inter-arrival time of 0.25 millisecond or less, and for downlink traffic it is 42% of the packets. On the other hand, the packet analyzer on the desktop PC, we found 33% of the uplink packets have inter-arrival time of 0.25 millisecond or less and for downlink packets it is 68%. These numbers suggest that the AP is reducing the *burstiness* of the traffic in both directions by introducing uneven delays, and therefore, the presented results give a conservative estimate about the traffic bursts.

Impact of OS To observe any possible impact of operating system (OS) or device on the wireless access traffic, we conducted the same set of experiments on three different smartphones namely, *HTC Nexus One*, *iPhone 3GS* and *BlackBerry 9700* by keeping all other parameters unchanged. However, we did not observe any significant differences in the distribution of the parameters discussed above.

4.6 Impacts on Energy Saving Methods

We summarize the characteristics of uplink and downlink traffic, and discuss the potential application of those attributes in designing energy saving techniques. The characteristics of uplink traffic are as follows:

- Durations of more than 95% of the bursts are below 10 milliseconds (Fig. 4.6);
- Inter-arrival times of more than 80% of the bursts are grater than 10 milliseconds (Fig. 4.10);
- Sizes of almost all uplink bursts are less then 1500 bytes (Fig. 4.7);
- Number of packets per burst is less than 16 (Fig. 4.9);

We observe the same statistics for downlink traffic except for the burst size. The downlink burst size is usually larger, and in case of *YouTube* video traffic, burst size becomes up to 50 kilobytes (Fig. 4.8). When we consider the bursts in both directions, only 30% of the bursts come less than 10 milliseconds apart (Fig. 4.11). Based on the analysis given above, we discuss its implication on different MAC level energy saving techniques in Table. 4.1. We explain the effects of different burst parameters in the following.

4.6.1 Impact of Burst Duration and Size

Different applications on a smartphone communicate with different servers. Sometimes even one application communicates with more than one server. However, all communications are routed through the AP in infrastructure wireless networks. Small burst durations (couple of milliseconds) create opportunities for holding data packets in a burst without affecting the applications' performance. As most of the burst size is less than MTU in uplink traffic, all the packets in a burst can be aggregated into a MAC frame. On the other hand, for larger burst size as in downlink traffic, several MAC frames containing individual packets can be accumulated before transmitting them at a time. There are several derivatives of *frame aggregation* techniques [89, 176], and they basically create longer idle periods for a communication interface. In addition to that *frame aggregation* technique reduces MAC and PHY layer overheads significantly. However, large burst sizes introduce longer packet delays, and re-transmission rate also increases in presence of moderate bit error rate (BER).

4.6.2 Impact of Burst Inter-arrival Time

The inter-arrival time of bursts gives us insight into how long the communication interface should go into doze mode for saving energy. Results show that about 35% of the bursts have inter-arrival time of less than 10 milliseconds. Therefore, a millisecond-level dozing is essential for uninterrupted flow of the data traffic. However, the beacon interval is 100 milliseconds in existing WiFi networks, and therefore, the current beacon interval is unable to accommodate energy saving management scheme in presence of online multimedia or VoIP traffic. Moreover, sending PS-POLL for receiving data after each tiny doze interval is impractical, because energy saved from short doze mode would be spent in sending PS-POLL messages. Therefore, coordinating the state information of the devices with the AP is a crucial design issue.

4.6.3 Coordination between Device and AP

The values of device's doze interval and AP's PSM (*Power Save Mode*) message interval must be chosen in such a way that an AP is able to track the state of an attached device without getting an explicit PS-POLL message. On the device side, a device needs to wait for a PSM message after waking up from doze state. It either expects data packets from the AP or goes into doze mode again according to the status value in the PSM message. The

challenge in the device side is to reduce the wait time before going into doze mode further. To achieve these objectives, one or more of the following measures worth investigation.

Natural Coordination As we mentioned in the beginning of this section, the number of uplink packets is comparable to downlink data packets, and the distribution of burst inter-arrival times in uplink traffic is similar to the downlink traffic. A device can take advantage of this natural phenomenon by synchronizing the doze interval with the uplink frame rate. An AP informs a device of any buffered data using the Acknowledgment's (ACK) *more data* field, and the device receives subsequent frames from the AP. No extra PSM message is needed here. However, this scheme may not work when the uplink frame rate is low as compare to downlink rate (as in *YouTube* traffic).

PSM Message with ACK In infrastructure wireless networks, in any data exchange, the access point becomes either a sender or a receiver, and it often needs to send ACKs. Since the beacon message is large, and beacon interval is long, an AP can tag the buffer status of connected devices with MAC level ACKs.

For example, if an AP supports 256 devices, it needs only a 8 bytes vector to indicate the status for each device. This technique does not require extra PHY or MAC level overheads, and the status message can be sent more frequently. The client devices can update themselves accordingly.

Extra PSM Message When the network is under-loaded, AP does not need to send ACKs very often. In such situations, AP itself can send PSM messages time to time containing buffer status. AP does not run on battery and in low traffic scenario, this frequent PSM messaging will not reduce network throughput.

Table 4.1: Impact of the analysis on different MAC-level energy saving techniques

Traffic Parameters	**Observations**	**Standard PSM (*e.g.,* [50])**	**Flexible Beacon Period Technique (*e.g.,* [106])**	**Micro-Power Saving Techniques (*e.g.,* [16, 89])**	**Frame Aggregation Techniques (*e.g.,* [2, 78, 176])**
Packet Size (Fig. 4.4)	About 85% of the uplink packets are smaller than 100 bytes and 60% of the downlink packets are greater than 1400 bytes.	x	x	x	Smaller packets in uplink are suitable for aggregation as a number of them fit into a MAC frame. Larger packets in downlink are suitable for accumulation into a physical layer frame, so that physical layer overhead is reduced.
Burst or Packet Inter-arrival Time (Fig. 4.10 & Fig. 4.11)	The inter-arrival times of 80% of the uplink packets is less than 20 milliseconds, and it is less than 1 millisecond for 60% of the downlink packets. The inter-arrival times of 70% of the bursts in both uplink and downlink traffic is more than 10 milliseconds.	Small inter-arrival times of packets are not suitable. The lowest beacon interval in standard PSM is 100 milliseconds, whereas, only 20% of the packets (see Fig. 4.5) have inter-arrival time of more than 20 milliseconds. Therefore, standard PSM is not feasible to save energy when applications (used in the analysis) run on a device. However, when a device is in idle state, standard PSM can be used to save energy utilizing the *doze* mode.	Longer burst inter-arrival times cause less number of beacon messages. For the kind of traffic we observed, this technique requires transmission of frequent beacon messages as the inter-arrival times of 60% of the packets (see Fig. 4.5 and Fig. 4.11) are less than 20 milliseconds. Significant portion of bandwidth would be spent on sending beacon messages with increased number of clients attached to an AP.	Inter-frame space dozing is always beneficial (if achievable). However, longer inter-arrival times can be utilized in dozing if coordination with the AP can be maintained. These techniques suggest to utilize very small inactive periods even in between MAC inter frame spaces. Though dozing for a couple of micro-seconds is quite challenging given the present state of the wireless interfaces, it does not require co-ordination with the AP.	Small inter-arrival times are also good as they result in small burst durations. 80% of the uplink bursts have less than 5 millisecond of inter-arrival times and these bursts consist of packets of smaller sizes. These features are attractive for frame aggregation techniques. Inter-arrival times of packets can also be controlled by transport level technique such as PSM-throttling [147] to facilitate frame aggregation.
Burst Duration (Fig. 4.6)	About 80% of the bursts has duration of less than 10 milliseconds.	x	x	x	Data packets incur less delay for small burst durations. The observed burst durations are smaller than the packet inter-arrival times of *VoIP* traffic.
Burst Size (Fig. 4.7 & Fig. 4.8)	The size of the 90% of the uplink bursts is less than 1000 bytes. In downlink traffic, the burst sizes vary with the type of the applications, and for *YouTube* video, the burst sizes go beyond 20 kilobytes.	x	x	x	Smaller burst durations sometimes lead to larger burst sizes, and based on traffic, longer burst durations can also result into smaller burst sizes. Thus, both timer and size based thresholds need to be used in aggregation techniques.

x denotes that a technique does not deal with that parameter.

4.7 Packet Aggregation Scheduler

Motivated by the observations discussed earlier in this chapter, we investigated the energy saving potential of data packet aggregation at MAC layer. In infrastructure wireless networks, clients forward all data packets from different applications to the WLAN or cellular access point. The packets that arrive as a burst are good candidates for aggregation process for reducing delay incurred by the packets due to accumulation process. If the aggregation process is accomplished in network layer, several queues would be needed to maintain individual source-destination pair, and the resultant traffic in each flow exhibits less burstiness. On the other hand, some might argue that applications send as much data as possible at a time, so gathering consecutive packets from an application might reduce the performance, even impede the functionality of the application. Though developers do not always send optimum size data packets as they do not keep energy efficiency in mind, that claim is valid to some extent. This situation can be avoided by aggregating packets from several applications in MAC layer. However, consecutive packets with very little inter-arrival time can be aggregated as the short time interval implies the independence of each packet.

The IEEE 802.11n standard enables high data speed connection which is about 100 Mbps measured at MAC layer. To accommodate such data rate, it supports two MAC level frame aggregation techniques, namely, Aggregate-MAC Service Data Unit (A-MSDU) and Aggregate-MAC Protocol Data Unit (A-MPDU). These two aggregation techniques can also be combined in two levels [87, 135, 145]. However, the standard does not specify the scheduler for these schemes, and it is left as vendor's choice. Here, we propose a packet aggregation scheduler named as Low Energy Data-packet Aggregation Scheduler (LEDAS) in this regard.

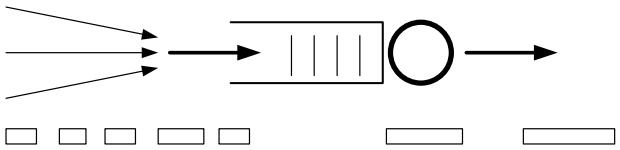


Figure 4.12: View of the aggregator as a queuing system.

LEDAS accumulates a number of upper layer packets into a burst at medium access control (MAC) level, based on *formation time*, *size*, and *number of packets*. With this scheme, larger bursts lead to longer *inactivity periods* during which the communication module can be kept in doze mode. Figure 4.12 shows a schematic diagram of the aggregation process. Fewer MAC frames lead to less overheads and contentions in the wireless

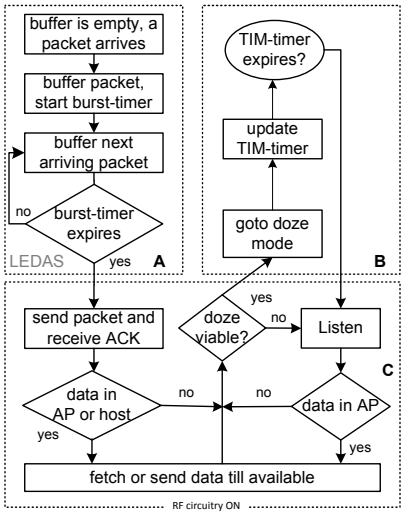


Figure 4.13: Flow diagram of the aggregation process.

medium. However, the data packets incur delays due to the accumulation process. We have given a detail flowchart description of the technique. By means of analysis, we have derived the distributions of *burst size*, *burst inter-arrival times*, and *number of packets* for three different burst selection criteria. We evaluated the efficacy of the technique by simulations and showed the energy-delay trade-offs. Finally, we evaluated the energy saving potential of LEDAS on a state-of-the-art *HTC Nexus One* smartphone.

4.8 Low Energy Data-packet Aggregation Scheduler

A flow diagram of the aggregation process is given in Fig. 4.13 and an algorithm showing the working principle of LEDAS is given in 4.8.1. It basically receives packets originating from different applications through Logical Link Control (LLC) sub-layer. The packets are held here until a hold time expires (τ), or the total size of the packets exceeds some threshold (α) or the number of packet crosses some limit (γ). The aggregated packet is termed as burst and hold time is termed as burst formation time. After a burst is formed, it is pushed into the MAC module. Normally, size of a burst is kept in a range of $[\alpha, \beta]$. When the size of a burst exceeds α, it is sent to MAC. In some cases, the size of an ongoing burst is less than α, but the size exceeds β after the arrival of the next packet. In such situations, burst is formed with existing packets and the newly arrived packet is considered for next burst.

Algorithm 4.8.1: LEDAS()

comment: Initialization

α is minimum burst length
β is maximum burst length
γ is maximum number of packets in a burst
n is number of packets in buffer
b is length of packets in buffer
w is length of a new packet
t_timer is TIM timer
b_timer is burst timer
status is state of LEDAS

$n \leftarrow 0, b \leftarrow 0$
buffer $\leftarrow \phi$, *status* $\leftarrow$ *idle*

comment: Buffer Empty (Idle Mode)

while *status* = *idle*
do
- **if** *t_timer* = *clock*()
  - **then**
    - *listen beacon*
    - RECEIVE-BUFFERED-DATA()
- **if** *a packet arrives in host*
  - **then**
    - ADD-TO-BUFFER(*packet*)
    - *status* $\leftarrow$ *active*
- **if** *no buffered data in AP or host*
  - **then**
    - *put RF circuitry in doze mode*
    - *till t_timer expires*

comment: Buffer NOT Empty (Active Mode)

while *status* = *active*
do
- **if** $b \geq \alpha$ **or** *b_timer* = *clock*() **or** $n = \gamma$
  - **then**
    - ADD-TO-MAC-BUFFER()
    - *status* $\leftarrow$ *idle*
  - **else if** *a packet arrives*
    - **if** $(b + w) < \beta$
      - **then** ADD-TO-BUFFER(*packet*)
      - **else if** $(b + w) > \beta$
        - **then**
          - ADD-TO-MAC-BUFFER()
          - ADD-TO-BUFFER(*packet*)

procedure ADD-TO-BUFFER(*packet*)
buffer $\leftarrow$ *packet*
$n \leftarrow n + 1, b \leftarrow b + w$
return

4.9 Analysis

Figure 4.14 shows the timing diagram of the aggregation process. The burst timers starts as the first packet arrives at the empty buffer. As shown in the figure, the size of the buffer increases as the subsequent packets arrive. A burst is released based on the burst size (α, β), burst timer (τ) and number of packets (γ) in a burst. We consider the distribution of inter-arrival time of the packets (A) and the distribution of incoming packet sizes (S) as exponential with mean λ and $1/\mu$, respectively. We analyze the process and present a summary of the findings here. The general idea is taken from [124] and [98]. The inter-arrival time of bursts (T) is the difference of arrival time of the first packet of two consecutive bursts. The distributions of burst size and number of packets in a burst are B and N, respectively.

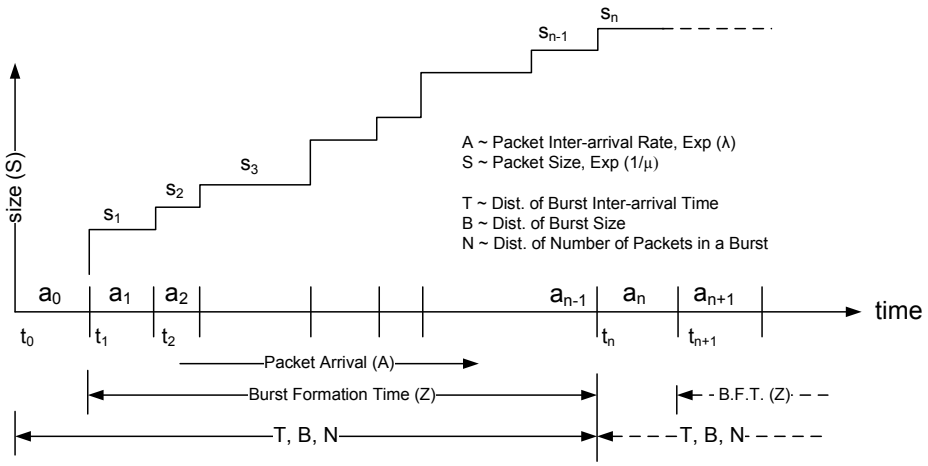


Figure 4.14: Timing diagram of the aggregation process.

4.9.1 Used Terms and Symbols

The symbols and terms used in this analysis are as follows:

- Distribution of packet arrival time (A) is exponential with rate λ and pdf is denoted by $f_A(t)$;
- Distribution of packet size (S) is exponential with mean μ and its pdf is denoted by $f_S(l)$;

- Probability density function (pdf) of burst formation time (Z) is denoted by $f_Z(t)$;
- Probability density function (pdf) of inter-arrival time between two consecutive bursts(T) is denoted by $f_T(t)$;
- Probability mass function (pmf) of the number of packets in a burst (n) is denoted by $P_N(n)$;
- Probability density function (pdf) of the length of a burst (B) is denoted by $f_B(x)$;
- Minimum burst length, α;
- Maximum burst length, β;
- $f_{S_n}(x)$ expresses the convolution of $f_S(x)$ with itself n times;
- $\Phi_S(u)$ is the Moment Generating Function (MGF) of some distribution, S;
- s_n is the length of n^{th} packet and
- L_k is the sum of lengths of k packets

4.9.2 Bursts sent on formation time

When a packet arrive at empty buffer, a timer is started and a burst is sent when the timer reaches τ. So, the burst formation time is τ. The burst inter-arrival time, T can be expressed as Eq. 4.1. The mean of packet arrival time (A) is $\frac{1}{\lambda}$, therefore, mean of T can be expressed as Eq. 4.2.

$$T = A + \tau \tag{4.1}$$

$$E[T] = \frac{1}{\lambda} + \tau \tag{4.2}$$

A burst contains n packets if $(n-1)$ packets arrive in time τ after the first packet arrive. Thus the probability that a burst contains n packets can be given as Eq. 4.3.

$$P_N(n) = \frac{(\lambda\tau)^{n-1}e^{-\lambda\tau}}{(n-1)!} \tag{4.3}$$

To compute the mean number of packets in a burst, we use moment generating function (MGF) of N. Once the MGF is known, the distribution of N will be known. The MGF

of N can expressed as in Eq. 4.4. The expected number of packets in a burst is given in Eq. 4.5.

$$\begin{aligned}
\Phi_N(u) &= \sum_{n=1}^{\infty} e^{un} P_N(n) \\
&= \sum_{n=1}^{\infty} e^{un} \frac{(\lambda\tau)^{n-1} e^{-\lambda\tau}}{(n-1)!} \\
\Phi_N(u) &= e^{-\lambda\tau} e^{u+\lambda\tau e^u} \qquad (4.4) \\
E[N] = \Phi'_N(0) &= 1 + \lambda\tau \qquad (4.5)
\end{aligned}$$

To obtain the properties of burst size, B, we used the Theorem 6.12 of [164]. The probability distribution of B is given in Eq. 4.6 and MGF of B can be expressed as Eq. 4.7. The MGF of packet size distribution, S is in Eq. 4.8.

$$\begin{aligned}
f_B(x) &= \sum_{n=1}^{\infty} f_{S_n}(x) P_N(n) \qquad (4.6) \\
\Phi_B(u) &= \Phi_N(\ln \Phi_S(u)) \qquad (4.7) \\
\Phi_S(u) &= \int_0^{\infty} e^{ul} f_S(l) dl = \int_0^{\infty} e^{ul} \frac{1}{\mu} e^{-\frac{l}{\mu}} dl = \frac{1}{1-\mu u} \qquad (4.8)
\end{aligned}$$

Now the MGF of burst size distribution, B and expected burst size $E[B]$ are given in Eq. 4.9 and Eq. 4.10, respectively.

$$\begin{aligned}
\Phi_B(u) &= e^{-\lambda\tau} \frac{e^{\frac{\lambda\tau}{1-\mu u}}}{1-\mu u} \qquad (4.9) \\
E[B] = \Phi'_B(0) &= \mu(1+\lambda\tau) \qquad (4.10)
\end{aligned}$$

4.9.3 Bursts sent on size

When bursts are released based on their size, two situations can take place: (i) current burst size is below α, and when a new packet arrives, the burst size falls in between $[\alpha, \beta]$. In this case, the newly arrived packet is included and sent with the current burst; (ii) current burst size is below α, but when a new packet arrives, the burst size exceeds β. In this case, the new packet is included in the next burst, and current burst is sent

with size less than α. The probability that a burst contains n number of packets can be expressed Eq. 4.11.

$$\begin{aligned} P_N(n) &= Pr(L_{n-1} < \alpha \text{ and } L_n \leq \beta) \text{ or } Pr(L_n < \alpha \text{ and } L_{n+1} > \beta) \\ &= Pr(\alpha < L_n < \beta | L_{n-1} < \alpha) + Pr(L_{n+1} > \beta | L_n < \alpha) \end{aligned} \tag{4.11}$$

Now, the probability of each part can be expressed as Eq. 4.12 and Eq. 4.13. The first part,

$$\begin{aligned} Pr(\alpha < L_n < \beta | L_{n-1} < \alpha) &= \int_0^\alpha Pr(\alpha < L_n \leq \beta | L_{n-1} = l) f_{S_{n-1}}(l)\, dl \quad n \geq 1 \\ &= \int_0^\alpha Pr((\alpha - l) < s_n \leq (\beta - l)) f_{S_{n-1}}(l)\, dl \\ &= \int_0^\alpha \left[F_S(\beta - l) - F_S(\alpha - l)\right] f_{S_{n-1}}(l)\, dl \\ &= \int_0^\alpha (e^{-\frac{\alpha}{\mu}} - e^{-\frac{\beta}{\mu}}) e^{\frac{l}{\mu}} \times Erlang(l; n-1, \mu)\, dl \\ &= \int_0^\alpha (e^{-\frac{\alpha}{\mu}} - e^{-\frac{\beta}{\mu}}) e^{\frac{l}{\mu}} \times \frac{l^{n-2} e^{-\frac{l}{\mu}}}{\mu^{n-1}(n-2)!}\, dl \\ &= \frac{(e^{-\frac{\alpha}{\mu}} - e^{-\frac{\beta}{\mu}})}{\mu^{n-1}(n-2)!} \times \int_0^\alpha l^{n-2}\, dl \\ &= \frac{(e^{-\frac{\alpha}{\mu}} - e^{-\frac{\beta}{\mu}})(\frac{\alpha}{\mu})^{n-1}}{(n-1)!} \end{aligned} \tag{4.12}$$

And, the second part,

$$
\begin{aligned}
Pr(L_{n+1} > \beta | L_n < \alpha) &= \int_0^{\alpha} Pr(L_{n+1} > \beta | L_n = l) f_{S_n}(l)\, dl \quad n \geq 0 \\
&= \int_0^{\alpha} Pr(s_{n+1} > (\beta - l)) f_{S_n}(l)\, dl \\
&= \int_0^{\alpha} (1 - F_S(\beta - l)) \times Erlang(l; n, \mu)\, dl \\
&= \int_0^{\alpha} e^{-\frac{\beta - l}{\mu}} \times Erlang(l; n, \mu)\, dl \\
&= \int_0^{\alpha} e^{-\frac{\beta - l}{\mu}} \times \frac{l^{n-1} e^{-\frac{l}{\mu}}}{\mu^n (n-1)!}\, dl \\
&= \int_0^{\alpha} \frac{e^{-\frac{\beta}{\mu}} l^{n-1}}{\mu^n (n-1)!}\, dl \\
&= \frac{e^{-\frac{\beta}{\mu}} (\frac{\alpha}{\mu})^n}{n!}
\end{aligned} \tag{4.13}
$$

The probabilities that a burst contains n number of packets are given in the following equation (Eq. 4.14).

$$
P_N(n) = \begin{cases} e^{-\frac{\beta}{\mu}} & n = 0 \\ \frac{(e^{-\frac{\alpha}{\mu}} - e^{-\frac{\beta}{\mu}})(\frac{\alpha}{\mu})^{n-1}}{(n-1)!} + \frac{e^{-\frac{\beta}{\mu}} (\frac{\alpha}{\mu})^n}{n!} & n = 1, 2, \ldots \end{cases} \tag{4.14}
$$

The MGF of distribution, N is derived in Eq. 4.15.

$$\begin{aligned}
\Phi_N(u) &= \sum_{0}^{\infty} e^{nu} P_N(n) \\
&= \sum_{1}^{\infty} e^{nu} \frac{(e^{-\frac{\alpha}{\mu}} - e^{-\frac{\beta}{\mu}})(\frac{\alpha}{\mu})^{n-1}}{(n-1)!} + \sum_{0}^{\infty} e^{nu} \frac{e^{-\frac{\beta}{\mu}}(\frac{\alpha}{\mu})^{n}}{n!} \\
&= (e^{-\frac{\alpha}{\mu}} - e^{-\frac{\beta}{\mu}}) \sum_{1}^{\infty} \frac{e^{nu}(\frac{\alpha}{\mu})^{n-1}}{(n-1)!} + e^{-\frac{\beta}{\mu}} \sum_{0}^{\infty} \frac{e^{nu}(\frac{\alpha}{\mu})^{n}}{n!} \\
&= (e^{-\frac{\alpha}{\mu}} - e^{-\frac{\beta}{\mu}}) e^{u+\frac{\alpha}{\mu}e^{u}} + e^{-\frac{\beta}{\mu}} e^{\frac{\alpha}{\mu}e^{u}} \qquad (4.15)
\end{aligned}$$

We get,

$$E[N] = \Phi'_N(0) = 1 + \frac{\alpha}{\mu} - e^{-\frac{\beta-\alpha}{\mu}} \qquad (4.16)$$

The probability distribution of burst inter-arrival time can be expressed as Eq. 4.17.

$$\begin{aligned}
f_T(t) &= \sum_{n=1}^{\infty} f_{A_n}(t) P_N(n) \\
&= \sum_{n=1}^{\infty} Erlang(t; n, \lambda) P_N(n) \qquad (4.17)
\end{aligned}$$

The MGF of T can be expressed as Eq. 4.18.

$$\Phi_T(u) = \Phi_N(\ln \Phi_A(u)) \qquad (4.18)$$

Now, the MGF of the distribution of packet arrival (A) is,

$$\begin{aligned}\Phi_A(u) &= \int_0^\infty e^{ut} f_A(t)dt \\ &= \int_0^\infty e^{ut}\lambda e^{-\lambda t}dt \\ &= \lambda\int_0^\infty e^{-(\lambda-u)t}dt \\ &= \frac{\lambda}{\lambda-u} \end{aligned} \tag{4.19}$$

Hence, $\Phi_T(u)$ becomes,

$$\begin{aligned}\Phi_T(u) &= \Phi_N(\ln\Phi_A(u)) \\ &= (e^{-a}-e^{-b})\Phi_A(u)e^{a\Phi_A(u)} + e^{-b}e^{a\Phi_A(u)} \end{aligned} \tag{4.20}$$

The expected values of burst inter-arrival time, $E(T)$ and expected value of burst formation time, $E(Z)$ become (given in Eq. 4.21 and Eq. 4.22),

$$E[T] = \frac{1}{\lambda}[1+\frac{\alpha}{\mu}-e^{-\frac{\beta-\alpha}{\mu}}] \tag{4.21}$$

$$E[Z] = \frac{1}{\lambda}[\frac{\alpha}{\mu}-e^{-\frac{\beta-\alpha}{\mu}}] \tag{4.22}$$

Now, the probability distribution function of burst size can be expressed as Eq. 4.23. The expected value of the burst size can also be computed by taking product of average packet size and expected number of packets in a burst. The expected value of burst size is given in Eq. 4.24.

$$\begin{aligned} f_{B,N}(x,n) &= \int_0^L f_S(x-L)f_{S_{n-1}}(l)dl \qquad L<x\leq\beta \\ f_B(x) &= \sum_{n=1}^{\infty}\int_0^L f_S(x-L)f_{S_{n-1}}(l)dl \end{aligned} \tag{4.23}$$

$$\begin{aligned} E[B] &= \mu E[N] \\ E[B] &= \alpha+\mu(1-e^{-\frac{\beta-\alpha}{\mu}}) \end{aligned} \tag{4.24}$$

4.9.4 Bursts sent on number of packets

In this case, a burst is released when it accumulates γ packets. Since the packet arrival process is exponential, the distribution of burst inter-arrival time, T takes Erlang distribution (Definition 3.7 in [164]). The probability distribution function can be expressed as Eq. 4.25. The expected value of T is given in Eq. 4.26.

$$f_T(t) = \frac{\lambda^\gamma t^{\gamma-1} e^{-\lambda t}}{(\gamma-1)!} \quad (4.25)$$

$$E[T] = \frac{\gamma}{\lambda} \quad (4.26)$$

$$f_B(x) = f_{S_\gamma}(x) \quad (4.27)$$

$$\Phi_B(u) = (\Phi_S(u))^\gamma \quad (4.28)$$

$$E[B] = \Phi'_B(0) = \gamma\mu \quad (4.29)$$

The probability distribution function and MGF of burst size, B are given in Eq. 4.27 and Eq. 4.28, respectively. Intuitively, the expected burst size is $\gamma\mu$, and the number of packets in each burst is γ.

4.10 Simulation and Experimental Results

We evaluate the performance of the proposed LEDAS scheme through simulation and experiments. For simulation, we used *Pawan*, a simulator for Infrastructure WLAN with PSM. We have integrated LEDAS with Pawan and measured the values of different performance metrics such as reduction in energy consumption, and average packet delay. The simulation parameters are given in Table 4.2. At the end of this section, we present the results of our experiments on HTC G1 smartphone, the first smartphone from Google with Android operating system.

Figure 4.15 shows the reduction in energy cost for different packet aggregation time (also referred as burst formation time, BFT). The values are normalized with energy cost without packet aggregation or with 0 (zero) burst formation time. In the range of 20 milliseconds to 40 milliseconds of BFT, the consumed energy reduced to 40% to 60% in transmission and reception, respectively. These values are proportional to reduction in transmission and reception times which is a result of reduced number of MAC level frame transmissions and hence reduced overheads.

Table 4.2: Simulation parameters

Parameter	Value	Parameter	Value
SIFS	10 μs	Transmit (Tx)	2000 mW
DIFS	50 μs	Receive (Rx)	1000 mW
EIFS	268 μs	Listen (Ls)	800 mW
Slot Time	20 μs	Doze	25 mW
PLCP Header Time	96 μs	CW_{min}	31
MAC Header Size	34 bytes	CW_{max}	1023
MAC ACK	14 bytes	$[\alpha, \beta]$	$[1200, 1400]$
PS-POLL Size	20 bytes	γ	16

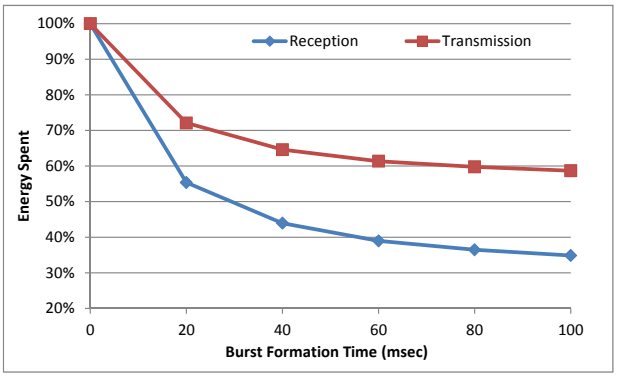


Figure 4.15: Reduction in energy costs.

We present the average packet delay, which can be seen as a cost of packet aggregation in Fig. 1.16. Without packet aggregation, the delay is negligible although it varies from 15 to 45 milliseconds for 20 to 100 milliseconds of BFT. For most of the applications, these delays, especially for the lower BFTs, are very much tolerable except for few, which require high quality of service (QoS) constraints. This figure also shows the average number of network level packets in each burst. As the number of concurrent applications running on smartphones grow, more packets can be combined in one MAC frame and more benefit can be obtained from LEDAS.

We present the average packet delay, which can be seen as a cost of packet aggregation in Fig. 4.16. The delay varies from 15 to 45 milliseconds for 20 to 100 milliseconds of BFT. The delay is negligible in absence packet aggregation. For most of the applications, these delays, especially for the lower BFTs, are very much tolerable except a few, which require to maintain high quality of service (QoS) constraints. This figure also shows the average

number of network level packets in each burst. As the number of concurrent applications running on smartphones grow, more packets can be combined in one MAC frame and more benefit can be obtained from LEDAS.

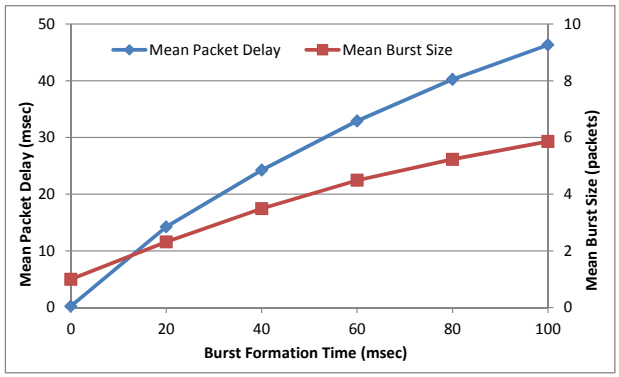


Figure 4.16: Average packet delays

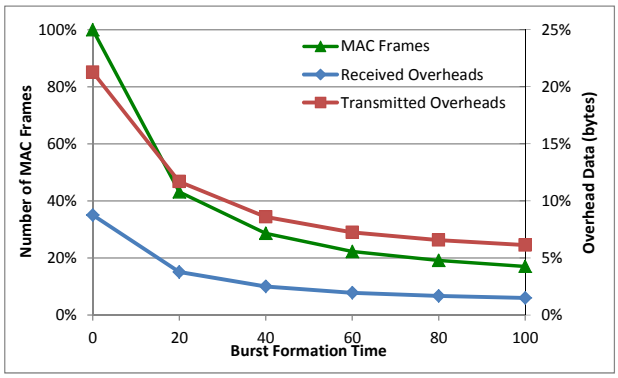


Figure 4.17: Received and transmitted overheads.

In WLANs, for every MAC level transmission of data packets, there is MAC level acknowledgment. Therefore, there are both transmission and reception overheads to send a data packet. The amount of overhead is shown in Fig. 4.17. The overheads are about 8% and 3% for 40 milliseconds of BFT, whereas the normal overheads are 22% and 9%, respectively. This is simply due to the reduction in number of MAC frames. Due to the aggregation process of upper layer packets, number of MAC frames reduced as the BFT increases. The side effect of this phenomenon is that there will be less contention for the acquisition the of the wireless channel and more bandwidth is utilized for exchanging data.

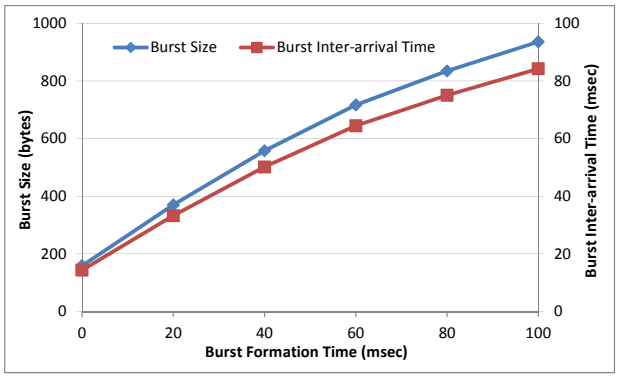


Figure 4.18: Average burst size and inter-arrival time at MAC layer.

In Fig. 4.18, we show the average burst size and average inter-arrival time of each burst at the MAC layer. As the burst formation time increases, more upper-layer packets are accumulated, and thus, the size of the burst increases. The increase in inter-arrival time between two consecutive bursts is particularly important to us. As the inter-arrival time increases, the RF circuitry can be turned off for longer period of time and more energy is saved by being in doze mode.

In the experiment, we used a Toshiba Tecra R10-ES1 laptop as server with Windows 7 operating system on it. A HTC G1 smartphone was used to carry out the experiment as a client. To measure the current consumption, we connect the smartphone with a Keithley 2304A high speed power supply [76]. The power supply is connected to a desktop PC on the USB port, and using a controller program on that PC, we set required output voltage on the power supply. Then we monitor the consumed current by the target device in millisecond intervals with ± 0.002mA accuracy.

The reduction in current consumption is given in Fig. 4.19. The current readings drop to 19 mAmp from 23 mAmp which may seem insignificant. However, the actual benefit of LEDAS comes from the utilization of the idle time between two consecutive bursts by keeping the communication interface to doze mode. We did not have enough access to control the interface in HTC G1. Therefore, the result yields partial benefit and the outcome is substantial.

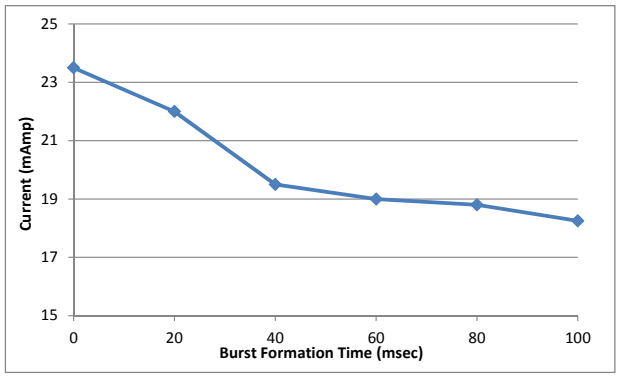


Figure 4.19: Current consumption for different burst formation time.

4.11 Summary

We have discussed potential energy saving issues by anatomizing the WiFi access data traffic of smartphones. We study the challenges associated with designing energy saving strategies for communication module. We have derived certain values and bounds from practical usage data. It will help the system designers to come up with better design and implementations. We also proposed low energy data packet aggregation scheduler (LEDAS). The performance of the proposed scheme is evaluated through analysis, simulations and experiments on a smartphone.

Chapter 5

Design of Energy Performance Testing

In this chapter, we provide a methodology to select user level test cases for performing energy cost evaluation of smartphone applications. We define the concept of a user level test case for smartphones and show that, due to configuration settings, there exist millions of such test cases. We propose a test selection technique to reduce the number of test cases. The proposed technique is applied to four different smartphones and their energy costs are evaluated for running common *network related applications*. We have developed a state-of-the-art test bench to execute those test cases for real applications on smartphones, and measure their actual energy costs.

5.1 Problem Description

There has been a rapid evolution in the smartphone industry over the last couple of years. Moderate computing power, communication bandwidth, and above all, innovative development tools have enabled the creation of mobile versions of the favorite desktop applications [125]. Among the applications, Internet browsing, online video and music playing, social networking, news, weather, stock reports, global positioning system (GPS) aided maps, navigation, searching, games, and productivity software are at the top of the charts [44, 93]. Uploading photos and videos directly to social networks and voice-over-IP (VoIP) clients are becoming more and more popular.

These applications are resource intensive in terms of bandwidth, processing power, and/or display, and naturally they draw a good amount of battery energy. However, there

has not been much improvement in energy density of battery [123]. Development in battery technology is not keeping pace with development in other sub-systems of smartphones. Consequently, battery energy puts the ultimate bottleneck on the performance capability, and availability of a smartphone. Therefore, there is a strong urge for designing all aspects of smartphones from the perspective of battery energy.

Users are generally not concerned with the speed of the processor, amount of memory, and the operating system of a smartphone though they care about a brand-name. On the other hand, what really matters to them is whether or not it contains interesting applications and how long the smartphone runs applications without recharging its battery. Thus energy cost measurement of applications is useful to the users, rather than the hardware level energy cost information. Another interesting phenomenon regarding users and applications of smartphones is the diversity of smartphone usage. Not all the users use all classes of applications on their smartphones. Different clusters of users favor different groups of applications.

Falaki *et al.* [45] characterized four key dimensions of smartphones which includes user interactions and usage of applications. Their survey reveals that users differ by one or more orders of magnitude along all the dimensions. For example, the mean amount of traffic per day varies from 1 to 1000 megabytes. They observed that despite these quantitative differences, qualitative similarities also exist among users. Users can be clustered according to their activities on smartphones. Therefore, when the performance of a smartphone is mentioned, the type of performance should also be noted. Otherwise, it will likely be marginally helpful only to a small percentage of users. A user level test case should consider a group of applications that correspond to a user group. Thus the end users will benefit from the outcomes of the tests.

The energy performances of two mobile applications can be compared by measuring the energy costs while executing the applications on a smartphone. In this case, the applications are intended for the same purpose, *e.g.,* applications for playing videos. The energy performances of two smartphones can also be compared by running the same application (perhaps, different versions) on those devices. As we explore in Section 5.4, typical smartphones possess a number of user settable parameters, and settings of these parameters have significant impact on the energy costs of smartphones. Hence the settings of these parameters, termed as configurations of the smartphones, should be consistent to have an unbiased outcome. For example, comparison of energy cost performances of two smartphones conducted at different levels of brightness (*e.g.,* 25% and 75%) is unfair. Therefore, the configurations of a smartphone must be considered during testing for fair and representative test results.

Besides producing inconsistent test configurations, the large number of user settable parameters of smartphones lead to millions of test configurations [4]. This makes the energy performance testing of smartphones impractical. Thus covering the test space, in other words, measuring the impacts of all parameters with reduced number of test configurations is the most crucial challenge in evaluating energy performance of mobile applications. We introduce the concept of *user level test case* and *test configurations* of smartphones. In this work, we consider *network related applications*, applications that require network communications, and present a methodology to measure energy costs with reduced and consistent test configurations. To reduce the number of test configurations, we categorize the smartphone parameters into three groups, namely, *basic*, *active* and *passive*. We have chosen five smartphones: *BlackBerry 9700*, *HTC HD2*, *HTC Nexus One*, *Nokia E71* and *Apple iPhone 3GS* and studied their configurable parameters. Tables containing the parameters are given in Appendix-A. The *active* parameters are further divided into *primary* and *stand-along* parameters to reduce the number of test configurations. A detailed methodology is presented in Section 5.5.

In performance testing, definition of test metrics play an important role to comprehend test results [156]. The energy cost of a test case is found as an average current for a given test configuration and the amount of energy is calculated by multiplying the average current by voltage and duration of the experiment. However, measured energy cost may not be a good metric for comparing two smartphones as they might have same type of battery but with different energy capacities. In that case, an application with same energy cost in both devices runs longer on the device with higher battery energy. Hence an energy cost metric should fully reflect the energy consuming behavior of an application and a smartphone. We propose an energy cost metric based on the average current consumed by an application, and energy capacity of the battery.

The methodology to evaluate the energy cost can be applied by developers as well as users if they have access to a test bench that we propose in this chapter. Thus, the users or product critics will have better instrument to test smartphones. On the other hand, the manufactures and developers will be able to enhance the system design and development by focusing on specific user groups. In this work, we make the following contributions:

- We define the concept of user level test cases to evaluate the energy cost by running applications on smartphones. We discuss why different sets of test cases are needed for evaluating energy cost of smartphones.
- We explain why the configurations of smartphones are so important for conducting experiments. We observe the challenges in selecting test configurations and the attributes of the test applications (Section 5.4).

- We discuss how to identify all such test cases. We apply a test selection technique to a class of applications called Network Related Applications (NRAs) (Section 5.5).
- We present the detailed design and implementation of a test bench to execute those test cases (Section 5.6), and execute few test cases on four smartphones and compare their results (Section 5.7).

Finally, we identify the limitations in performing energy tests of smartphones (Section 5.8). There has been little work reported describing approaches to performance testing of software applications. To the best of our knowledge, this work is the first that focuses on energy performance testing of mobile applications. In the next section, we present a general review of literature related to our work.

5.2 Literature Review

Although no reported work in the literature has direct similarity with our proposed work, in this section, we review a few papers to understand the concept. The software applications for mobile devices are generally developed in traditional computing environments such as on a desktop or a laptop, and are deployed in smartphones. As a result, the fundamentals of software engineering principles, i.e., analysis, design, implementation, and testing are still applicable to the engineering of mobile devices and applications [155]. Our proposed work involves three key elements: (*i*) software performance testing, (*ii*) testing on smartphone, and (*iii*) *combinatorial interaction testing (CIT)*. We divide this section into three subsections to review papers under each topic, and at the end of each subsection, we explain how our work is related to the discussed work. In the smartphone testing subsection, we cite some additional work for completeness.

5.2.1 Software Performance Testing

Weyuker and Vokolos [156] observed that lack of sufficient planning for performance issues often causes problems while an application is in use. They examined software performance testing issues, identified objectives for performance testing, and prescribed a methodology to form performance test cases. In performance testing, design of test case selection strategies means to test for performance criteria rather than functional correctness criteria. The criteria for performance measurement such as throughput, stimulus-response time, and scalability are considered from a user's perspective. According to the authors, formation of

performance test cases starts with identifying the software processes that directly influence the overall performance of the system. The subsequent steps are as follows:

- For each process, input parameters that significantly affect the performance of the system are selected. The number of input parameters is kept small to avoid a large test space.
- Values of the selected parameters are chosen from both average and heavy workloads, which reflect desired usage scenarios.
- When a parameter forms a range, representative values from within this range are chosen. Each selected value forms a separate test case.

Denaro *et al.* [38] argue that performance evaluation of distributed software application is useful in early development stages when important architectural choices are made. They observed that most of the critical performance faults are often introduced initially because of wrong architectural choices. Their proposed approach for early performance testing of distributed component-based applications consists of four phases: (*i*) for a given set of architecture design, select use-case scenarios relevant to performance, (*ii*) map the selected use cases to the actual deployment technology and platform, (*iii*) produce stubs of components that are not available in the early stages of the development, but are needed to implement the use cases, and (*iv*) execute the test. Our proposed work falls in the category of performance testing, and we followed the fundamental steps described in [156, 38] in our testing methodology.

5.2.2 Testing on Mobile Devices

Efficiency is one of the most important issues of mobile software engineering because of the resource scarcity. It applies to many aspects of mobile applications such as algorithms, power consumption, network access, data storage, visualization, user interface, and ergonomics. However, much work has not been done in these areas. Dedicated devices can be optimized for maximum battery life, but mobile applications may inadvertently make extensive use of battery draining resources [97]. A comprehensive review of energy saving methodologies is presented in [104]. Naik discusses the issues related to application to physical layer from energy saving point of view. Naik and Tripathi [103] explain the need for automating system test process as they are easily repeatable, time consuming, and easy to automate. Due to the number of configurations and test cases, automation of mobile

software test process is essential. Operating system (OS) assisted utility programs can be used to set configuration and get usage profiles [101] of the applications.

Bo *et al.* [15] mentioned that mobile devices are highly resource constrained in terms of processing ability, memory capacity and communication ability. Hence, an intrusive testing approach may affect the results of the testing. The diversity of the devices and platforms reduces the re-usability and maintainability of test cases. Moreover, the typical mobile applications highly interactive, and the devices often accept activations from the users and send responses back for user to take further actions. Thus automation of usage scenarios is difficult because of unpredictive user's actions. The authors discussed the drawbacks of commercially available *TestQuest* and *Digia AppTest* in detail. They proposed a black box testing approach called *MobileTest*, based on sensitive-event, such as reception of a Short message service (SMS). *MobileTest* uses capture-replay mechanism to help testers automate their test case generation task. This approach involves automatically capturing user interactions, and later replaying the interactions.

Satoh [133] developed a framework called *Flying Emulator* to test mobile applications to validate the applications on their real environment. This mobile agent based emulator performs application transparent emulation of its target device for applications written in Java. It facilitates the evaluation of application behavior on wireless networks without the need of the tester to move across different networks to validate a given mobile application behavior. Its main purpose is to enable application-level software to be executed and tested with the services and resources provided through its current network as if the application was being moved and executed on that target device when attached to the network. Other work based on *black box* and *mobile agent* based testing can be found in [136, 90, 167, 65].

Niranjan *et al.* [8] present a measurement study for energy consumption in smartphones with 3G, GSM and WiFi. They proposed a protocol to reduce the energy consumption for some applications. Jose *et al.* [12], emphasize that context-aware applications in mobile phone consume a large amount of energy, hence the number of operations in those applications may be decreased to reduce the energy consumption.

Huang *et al.* [63] developed a methodology to compare the user perceived performance of network applications in terms of some performance metrics. Those metrics fluctuate very much with carriers and devices. Xiao *et al.* [160] have shown that *WLAN* consumes less energy than *WCDMA* during video download and upload via YouTube. Similarly, Nurimen *et al.* [112] discussed that if contents are uploaded for other users during active downloading then it consumes little amount of extra energy in peer to peer applications.

Kim *et al.* [77] proposed a performance testing method at the *unit test* level for emulator based testing environment. Since the performance defects found at *system level* testing

often takes more time and efforts to reveal, the overall cost of the development process goes high. They utilize a database called *MObilePBDB* based on benchmark data from target device and integrate the test tools called *PJUnit* in the development environment like *Eclipse*. The setup basically predicts performance time of the application unit in the target mobile device.

5.2.3 Combinatorial Interaction Testing (CIT)

A *System Under Test (SUT)* generally comprises a number of parameters that take different values. Exhaustive testing that considers all possible combinations of test cases is desirable [26]. However, it is not feasible to carry out exhaustive testing because of the exponential explosion of the number of test cases. Suppose a smartphone has 10 user settable parameters, and each can take on 5 different values. The total number of combination becomes 10^5. Hence only a subset of all test cases can practically be considered as a test space [96]. *Combinatorial Interaction Testing (CIT)* is a technique to reduce the number of test cases of a software system systematically for testing [21, 31, 36, 81].

For a given *SUT* with k parameters, the main idea of combinatorial testing is to cover every possible value combination of no more than t parameters. Here, t is called the test strength. The value of t is a small number and the corresponding combinatorial test is called t-way testing [21, 81, 6, 108]. Experience suggests that not every parameter contributes to each fault and many faults are caused by interactions between a relatively small number of parameters, and this understanding is the key for t-way combinatorial testing. The 2-way testing popularly known as *pairwise testing* is based on the observation that software faults often involve interaction between two parameters.

Kuhn *et al.* [81] explains combinatorial testing with example and discussed the importance of it. Combinatorial testing is accomplished by constructing a set of tests known as *Mixed Covering Array or Covering Array* that covers all t-way combinations of parameter values. Results from their investigations show that many faults are caused by a single parameter value, and the progressively fewer faults are observed for higher oder interactions. For example, in a web server application about 40% of the failures were caused by a single value, another 30% were caused by the interaction of two parameters, and a cumulative total of almost 90% were triggered by three or fewer parameters.

While building the *Mixed Covering Array (MCA)*, it is assumed that different parameters are independent of each other. However, configurations included in MCA may become invalid due to application constraints [31, 32]. Ignoring constraints leads to inaccurate test planning and waste of effort. The authors of [32] discussed constraints with examples

from highly configurable system like Nokia 6000, and proposed a technique to represent constraints. They describe ways to integrate constraint checking with the existing CIT algorithms.

Chen *et al.* [21] addressed the issues of *shielding* parameters in combinatorial testing. The *shielding parameters* in many real-life applications disable other parameters in certain conditions. They proposed *Mixed Covering Array with Shielding Parameters (MCAS)* to explain the problem caused by shielding parameters. The authors mentioned that in realistic software applications, there may exist both regular constraints and shielding parameters, and dealing with combination of these two kinds of relationship is more difficult.

Smartphone parameters such as *3G data connection*, *WiFi*, *Bluetooth*, *volume*, and *brightness* are not independent with respect to an application, and some of them are shielding parameters at the same time. For example, for an image viewer application, there is no role of *volume*, and when *WiFi* data connection is turned on, 3G data connection automatically shuts down. Thus the testing environment contains interactions with constraints and shielding parameters, which is quite complex as Chen *et al.* [21] mentioned. Analysis of such systems has not yet been reported in the literature, and our proposed methodology is based on heuristic.

5.3 Formulation of Test Cases

Suppose that a smartphone has m number of user settable parameters denoted by $U_d = \{B_1, B_2, B_3, \ldots, B_m\}$. A parameters B_j takes a value (b_j) from an ordered set $\nu(B_j)$, and the k^{th} instance of parameter B_j is referred to as b_j^k. The total number of different states of B_j is denoted by $\eta(B_j)$. Another representation of $\eta(B_j)$ is $|\nu(B_j)|$, which is the number of elements in $\nu(B_j)$. For example, *brightness* (referred as B_{32} in Table 5.5) of a smartphone's display is a parameter that a user can set at different levels. Therefore, for brightness, $\nu(B_{32}) = \{0, 25, 50, 75, 100\}$, $\eta(B_{32}) = 5$, and an instance of B_{32}, $b_{32}^2 = 50$.

A *configuration* of a smartphone is an instance of all user settable parameters for executing a test case. The set of all *configurations*, S, can be given as $\{\nu(B_1) \times \nu(B_2) \times \ldots \times \nu(B_m)\}$. A *configuration* ($\beta_i$) is a member of S, and is expressed as $\{b_1, b_2, \ldots, b_m\}$. Let us consider another parameter, *Data Access Mode* (referred as B_{37} in Table 5.5), where, $\nu(B_{37}) = \{$OFF, WiFi, EDGE, 3G$\}$ and $\eta(B_{37}) = 4$. If we consider only these two parameters B_{32} and B_{37}, then $\{25, WiFi\}$ and $\{50, 3G\}$ are two examples of configuration.

An application setting, α_i comprises $\{A_i, C_i, T_i\}$. A_i represents the class and description of an application and C_i indicates types of content for A_i. T_i is the execution time of A_i

on a smartphone. Suppose that there is a video file player application (a_1) that supports only MPEG (c_1^0) and Flash (c_1^1) format of video files. The number of different contents is denoted by $\eta(C_i)$ and for this application, $\eta(C_1) = 2$. The time durations for executing these two types of video files are t_1^0 (for example, 60 sec) and t_1^1 (for example, 90 sec), respectively.

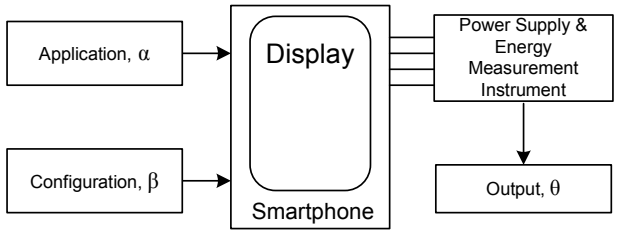


Figure 5.1: System model of proposed test configuration.

A test case γ_i is represented as a pair <input; expected output>. An application setting, α_i and a device configuration, β_i constitute an input. The output θ_i of a test case is expressed as energy cost metric that we propose in this work. We formally express $\gamma_i : \langle \alpha_i, \beta_i; \theta_i \rangle$ and Fig. 5.1 shows a logical diagram of the test configuration. An application is executed by setting the user settable parameters according to a specific configuration and the consumed energy is measured by a high resolution current measurement unit.

5.4 Challenges

There are two main challenges involved in test case selection process: (i) large number of configurations, (ii) large number of applications, and a variety of contents for these applications. We describe the details of each challenge in this section.

5.4.1 Number of Configurations

We examined various user settable parameters of smartphones and identified the ones which are expected to influence the energy consumption while running an application on smartphones. We identified the configurable parameters for a set of smartphones, and some of them are listed in Table 5.5 and 5.6. In these tables, 'Yes' means the parameter is available, 'No' implies it is not available and 'Alternative' implies that a similar parameter is available. Most of the parameters can be set as On/Off or Manual/Auto while others

have a list of options to select from. Now, the number of test cases (N_c) on smartphones can be given as:

$$N_c = S_d \times \sum_{i=1}^{M} X_d(A_i) \times \eta(C_i) \tag{5.1}$$

Here, A_i is a selected application, S_d is the number of configurations in device d, M is the number of selected applications. $X_d(A_i)$ can be expressed as:

$$X_d(A_i) = \begin{cases} 1 & \text{if } A_i \text{ is executable in device } d; \\ 0 & \text{otherwise.} \end{cases} \tag{5.2}$$

At this point, the challenge is to identify the values of S_d, M and $\eta(C_i)$ such that it provides a feasible set of test cases for running target group of applications on smartphones for different user groups and further determining the impact of an individual parameter on energy consumption.

A straightforward way to identify all the configurations is to consider all possible combinations of the available parameters. We present a general expression for a single smartphone. Later, we exemplify this expression for HTC Nexus One smartphone. We define,

$$S_d = \prod_{B_j \in U_d} \eta(B_j) \tag{5.3}$$

Descriptions of U_d, B_j and $\eta(B_j)$ are given in Section 5.3. Based on the discussion, the number of experiments (N_c) for HTC Nexus One can be calculated as follows: we found that there are 19 parameters in U_2 for HTC Nexus One. By applying Eq. 5.3, we get, $S_2 \approx 47X10^6$. If we further consider the number of applications and contents, the total number of test cases, (N_c), will be huge. To develop a feasible methodology for energy measurement, a subset of the existing configurations needs to be chosen so that selected test cases disclose the behaviors of energy cost patterns.

5.4.2 Choosing Applications, Contents, and Durations

It is not feasible to measure the energy cost of running each of the applications, and as such we need to select few applications (termed as *test applications*) from each class which represent a group of applications. For example, an online video playing application requires

Internet access and at the same time, it needs to decode the video frames. Therefore, an online video playing program needs to be developed (or, chosen from existing applications) which very much represents the attributes of online video playing applications.

The content or input to an application may impact the energy costs. For example, a video playing application may consume different amount of energy for different video files. Different video format require different amount of processing time and also, they need to fetch different size of Internet data. Based on the number of connected players through networks and/or settings of a game, the energy costs of game vary. Therefore, a note of content type in the test case description is important.

The duration of running a test case is also critical as it captures the variation of energy costs over time. We need to measure the energy costs for a *complete cycle* of execution, so that the subsequent phase of execution is just a replication of the measured stage of an application. By *complete cycle*, we refer the time duration, during which, an application fully exhibits its energy consumption behavior.

5.5 Proposed Methodology

The objectives of our proposed methodology are (*i*) to have a consistent test configuration across smartphones, so that we will be able to compare the results; and (*ii*) to capture the energy consumption behaviors of an application with reduced number of experiments. An experiment is an execution of a test application on a given test configuration. A test application is a software realization of a test case and a test configuration is a particular setting of all user settable parameters in a smartphone.

To ensure a consistent test configuration, we at first need to know all the parameters of a smartphone, which involve in the energy consumption. For that, we examined such parameters of five (5) smartphones, namely BlackBerry 9700, HTC Nexus One, Nokia E71, HTC HD2 and iPhone 3GS.

5.5.1 Categorization of Parameters

We explored the parameters of the smartphones, and make a comprehensive list of the parameters found on those devices. The values of some parameters cannot be changed or fixed, and the values of some parameters can be set at different levels. Since the combination of different values of all parameters are vast in number, it requires much time

and effort to deal with these parameters individually. Intuitively, categorization of these parameters according to their impacts on the energy consumption is useful in this situation to reduce number of test cases. With this view, we proposed categorizing the parameters of a smartphone into three main groups which are illustrated in Fig. 5.2. Parameters with fixed values are categorized into *basic* parameters, and the rest of the parameters are called *active* and *passive* parameters. We define the groups as following.

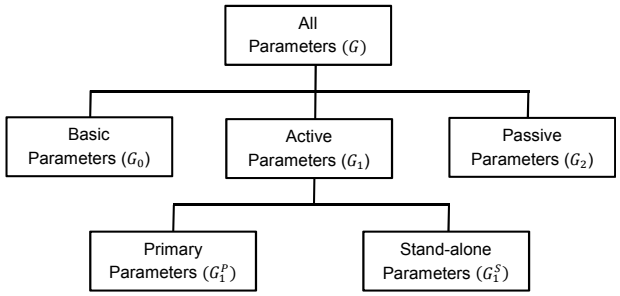


Figure 5.2: Categorization of smartphone parameters

- *Basic Parameters* (G_0): This group contains all the parameters whose values are fixed. A user cannot change or adjust the values of those parameters. Processor, Memory and size of display fall in this group. Table 5.4 in the appendix shows the basic parameters of the five smartphones mentioned earlier. These parameters affect the energy consumption of a device, and their impacts just remain the same throughout the testing process. For this reason, we can keep them aside during the testing process. However, when we compare two or more smartphones on energy performance, the differences in these parameters must be noted during the analysis of the test results.

- *Active Parameters* (G_1): This group contains the parameters whose values are settable. The user can control and adjust the values of those parameters. Basically, these parameters are some utility programs on top of operating system (OS), which control hardware components. Applications for controlling volume of a device, brightness of the display, turning on/off the global positioning system (GPS) are examples of active parameters. Table 5.5. The active parameters are further divided into two groups as described below.

  - *Primary Parameters* (G_1^p): An active parameter is regarded as primary parameter. It appears to consume a significant proportion of the total power

consumption. Variation of it values yields more informative test results. The primary parameter selection criteria are further discussed in Section 5.5.3.

- *Stand-alone Parameters* (G_1^s): The rest of the *active* parameters are *stand-alone* parameters.

- *Passive Parameters* (G_2): This group contains the rest of the parameters whose values are also adjustable by the users. They are also utility programs incorporated in the OS or from third parties. For example, WiFi Hotspot and Bluetooth Discoverable are programs which use WiFi and Bluetooth interfaces, respectively. They do not control the hardware components, but they use hardware components, and thus, they affect the test outcomes. During the testing process, we need to take out the effect of other applications by removing them or by keeping their effect fixed throughout the testing process. Thus, we turn off or fix certain values for these parameters. We particularly concern about the listed parameters given in Tab. 5.6 as they come with most of the smartphone operating systems or they are third party applications that are becoming more common in smartphone environments.

The central idea of our proposed technique is to keep the value of $G2$ parameters to certain levels and then vary the $G1$ parameters to see their impact on the energy costs of an application.

5.5.2 Number of Configurations for Active Parameters

We pick an application (A_i) based on a test case and select a potential parameter (B_i) from $G1$ that expects to have more impact on energy cost for A_i. For instance, *Network data access mode* is chosen for NRAs. Then, we vary (B_i) and observe the energy cost of A_i. During this time, all other parameters in $G1$ are kept fixed to certain values. Parameters of $G2$ are kept constant throughout all experiments. We refer this set as *primary* configurations. To measure the impact of other parameters of group $G1$, we choose another parameter (B_j) from $G1$ and observe the joint impact of B_i and B_j on the energy cost of application (A_i). This set of configurations is termed as *stand-alone* configurations. Numbers of *primary* and *stand-alone* configurations are denoted by S_d^p and S_d^s, respectively.

Now, the total number of test cases according to our scheme is given in Eq. 5.4. S_d in Eq. 5.1 becomes $(S_d^p + S_d^s)$ in Eq. 5.4. In primary configuration, only one parameter (B_i) is considered by keeping others fixed. Therefore, the value of S_d^p can be given as in

Eq. 5.5. For example, *YouTube Video Player* application is an NRA and we consider *Data Access Mode* (B_{37}) for primary configurations. Therefore, we observe the energy costs for $\{WiFi, EDGE, 3G\}$ connections. The values of other parameters such as volume (B_{31}), brightness (B_{32}) are kept at 25% during this period.

$$N_c^{'} = (S_d^p + S_d^s) \times \sum_{i=1}^{M} X_d(A_i) \times \eta(C_i) \tag{5.4}$$

$$S_d^p = \eta(B_i) \tag{5.5}$$

In Table 5.1 and 5.2, we present the energy costs of different combinations of the parameters B_i and B_j. Table 5.1 refers to the costs of *primary configurations*. Here, the modes of B_i are varied while the value of B_j is kept fixed to some value, b_j^1 for instance. In the stand-alone configurations, we choose another value of B_j, suppose, b_j^2 and conduct experiments for different values of B_i. In Table 5.2, second row contains the energy costs of primary configurations and third row contains costs of some stand-alone configurations. Let, $\Delta_k = |\theta_{2,k} - \theta_{1,k}|$, where $k\,(0 \leq k \leq 3)$ refers to the state of B_i. If values of Δ_k are equal, we conclude that energy costs for both parameters are additive, and in this case, we need to conduct experiments only across the values of B_j with any value of B_i. On the other hand, if values of Δ_k are not equal, we need to consider all the combination of $\{\nu(B_i) \times \nu(B_j)\}$ to fully extract the energy consumption behavior of B_i and B_j for application A_i.

$\Downarrow$ Other parameters	Parameter B_i			
$B_1, \ldots, B_{i-1}, B_{i+1}, \ldots, B_g$	b_i^0	b_i^1	b_i^2	b_i^3
$b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_g$	θ_0	θ_1	θ_2	θ_3

Table 5.1: Primary configuration

To identify the stand-alone configurations for the same *YouTube Video Player* application, let us consider another parameter brightness (B_{32}) from $G1$. In primary configuration, the value of B_{32} is 25%, for stand-alone configuration, we need to choose another value, 50% for instance. Now, the stand-alone configurations are $\{WiFi, EDGE, 3G\}$ with brightness 50%. If the differences of energy costs for brightness levels 25% and 50% across $\{WiFi, EDGE, 3G\}$ are same, we can say that both parameters are independent of each other and we just need to find the energy cost at brightness levels 0%, 75% and 100%. Otherwise, we need to find energy costs for all data connections at all brightness levels. We need to repeat this process for other parameter in $G1$ such as *Volume*.

Parameter, B_j	Parameter, B_i				
	States	0	1	2	3
	0	θ_{00}	θ_{01}	θ_{02}	θ_{03}
	1	θ_{10}	θ_{11}	θ_{12}	θ_{13}
	2	θ_{20}	θ_{21}	θ_{22}	θ_{23}
	3	θ_{30}	θ_{31}	θ_{32}	θ_{33}

Table 5.2: Dependency check table

The number of stand-alone configurations can be expressed formally. Let Q_j is the number of stand-alone configurations corresponding to B_j such that $B_j \in G1$. Then, Q_j can be expressed as shown in Eq. 5.6.

$$Q_j = \begin{cases} \eta(B_j) & B_i, B_j \text{ are independent;} \\ (\eta(B_j) - 1) \times S_d^p & B_i, B_j \text{ are dependent.} \end{cases} \tag{5.6}$$

One value of B_j is excluded from stand-alone configurations, as it is considered in primary experiments. Now, S_d^s becomes,

$$S_d^s = \sum_{B_j \in G1 \bigwedge j \neq i} Q_j \tag{5.7}$$

5.5.3 Choosing A Primary Parameter

In performance testing parameters are chosen according to their degree of influence on the performance of the system [156]. In our methodology, the larger is the number of primary parameters, the larger is the number of test configurations. For NRAs, *network access mode* plays a significant role on their energy consumption. Hence we choose to have *network access mode* as the primary parameter.

5.5.4 Parameter with Continuous Value

The settable values of a parameter may appear as different forms on different smartphones. For instance, *brightness* can be set at any point between 0 and 100 in HTC Nexus One, but in BlackBerry 9700, it has to be set at 0, 10, 20, and so on. In such cases, we choose to opt the discrete levels and set the continuous levels only to the discrete levels. Another interesting issue is that the same levels of *volume* or *brightness* may not represent the same

level across the smartphones. In this case, we need to use external calibration tools such as decibel meter, photometer to ensure same actual levels across the smartphones.

5.5.5 Energy Cost Metric

A smartphone consumes some current even when we do not run any user application, and most of this current is consumed by the display. Suppose that the smartphone consumes I_x amount of current when applications A_x is executed. Since the battery capacities are different across smartphones, we suggest that the corresponding energy cost metric (θ_x) of A_x for a given configuration is ξ/I_x hour. Here, ξ denotes the battery capacity. Some examples of energy cost metric calculation are given in Tab. 5.3. It requires further analysis and experimentation to compute the energy cost when more applications are executed concurrently.

ξ	I_0	I_a	I_x	I_y	θ_x	θ_y
(mAh)	(mA)	(mA)	(mA)	(mA)	$(hour)$	$(hour)$
1500	7.5	50	350	80	4.3	18.7
1230	7	43	188	97	6.5	12.7
1400	9	175	196	375	7.1	3.7

Table 5.3: Examples of energy cost metrics

5.6 Test Bench

We describe a test bench to facilitate experimentation of smartphones to measure energy costs of applications. As shown in Fig. 5.3, the setup mainly includes (*i*) smartphone(s); (*ii*) power supply with a high precision current measurement unit; (*iii*) a desktop or laptop computer to control and monitor the power supply unit; (*iv*) a wireless Access Point (AP); (*v*) a web server; and (*vi*) a cellular network connection with data access.

We use the connections as shown in Fig. 5.4 to carry out the experiment. With this connection setup, smartphone is able to read battery condition, but gets energy from the power supply. The connected battery also does not get any power supply. We keep the battery fully charged so that no power-saving mechanism is activated in the smartphone due to low energy situation in the battery.

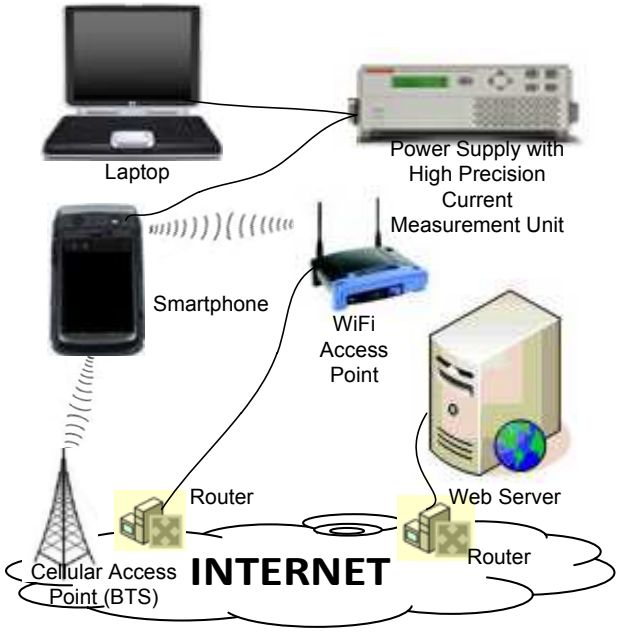


Figure 5.3: Experiment setup of test bench.

The power supply is initialized with the battery ratings of the smartphone through a controller program installed in a desktop or laptop computer. The smartphone is turned on and a test configuration is selected before conducting experiment. Consumed current is measured by a monitor program installed in the same computer with and without running the test application. We used Keithley 2304A, a high speed power supply with accuracy in measuring current of $\pm(0.2\% + 400\mu A)$.

5.7 Experimental Results

We used BlackBerry 9700, Nokia E71, HTC Nexus One and HTC HD2 smartphones in our experiments. We did not execute our test cases on iPhone 3GS. Because we could not access the battery interface as illustrated in Fig. 5.4 for the specific way the phone has been packaged. When we discuss the results of the experiments, we do not refer to specific smartphones, rather we use the terms *Phone A*, *Phone B*, *Phone S* and *Phone W*. We are more interested in how to measure the performance of the devices than their brand names. We setup the experiments as described in Section 5.6.

In Fig. 5.5, 5.6, and 5.7, we show the performances of the smartphones for different

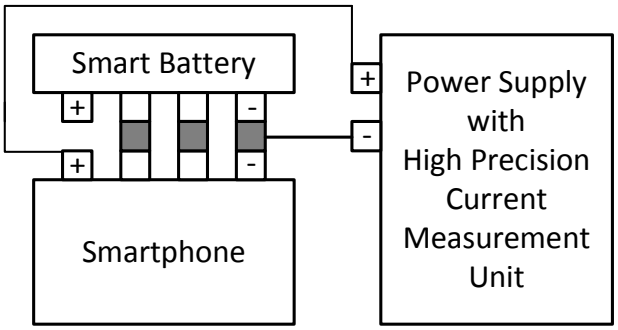


Figure 5.4: Connection details of device, battery and power supply.

NRAs, namely, YouTube video player, Internet browsing, email composing. Then we present the findings of some stand-alone experiments on individual devices in 5.8, and 5.9. Throughout the experiments, we fixed the parameters of Table 5.5 and 5.6 to certain values. We do not present the details due to space limitation. The energy metric that we proposed can be interpreted as expected lifetime of a device for a specific application. Therefore, a higher energy metric implies better performance.

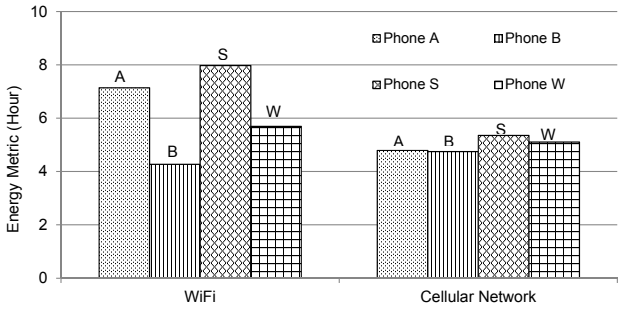


Figure 5.5: Energy metrics for YouTube video.

Figure 5.5 shows the comparison of the four smartphones when we played a YouTube video (animusic-pipe-dream) on them. *Phone A* and *Phone S* perform better than the other two in case of WiFi connection, but all of them perform the same when they use cellular data network. The average energy metric is also better for WiFi connection which suggests the use of WiFi when available.

A similar trend is observed while we browse Internet on the smartphones. As shown in Fig. 5.6, the energy metrics for *Phone B* and *Phone S*, are very high for WiFi connection, compare to cellular data connection.

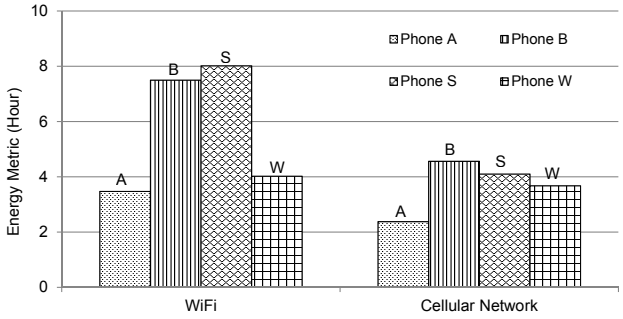


Figure 5.6: Energy metrics for Internet browsing.

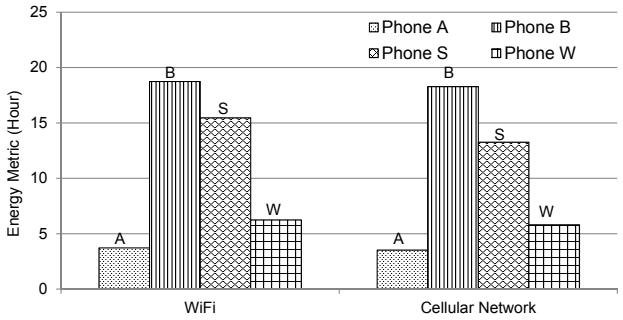


Figure 5.7: Energy metrics for composing email.

The smartphones show a different tendency while we compose an email. As we observe in Fig. 5.7, the performances of all smartphones are almost similar for both WiFi and cellular connection. While composing an email, there is minimum interaction with the network connections, and therefore, the network access modes do not impact the performance of smartphones.

We run three video files on *Phone A* over WiFi connection to the full length of their durations. The performance metrics for all the videos are almost same. This result implies that the content does not matter while running an online video application.

Next, we observe the energy metrics for WiFi, EDGE and 3G connections on Phone B. We ran YouTube video player using different network connections. As shown in Fig. 5.8, we found that EDGE connection performs better than the 3G and WiFi connection. This trend may not be similar across the smartphones.

Finally, we examine if there is any impact of *data access mode* (B_{37}) on *brightness* (B_{32}) in regards to energy consumption. We discussed the technique to check the dependency between two parameters in section 5.5. As shown in Fig. 5.9, the differences in current

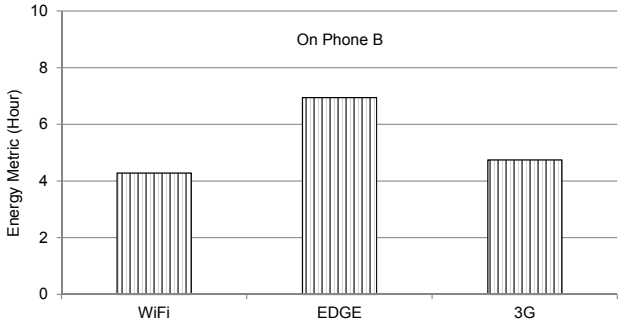


Figure 5.8: Energy metrics for various network connections.

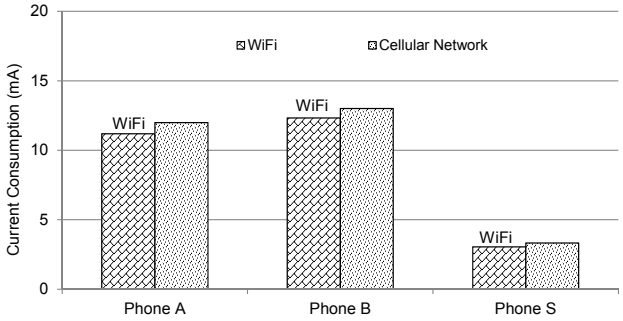


Figure 5.9: Differences in current consumption at brightness levels of 50% and 75%.

consumption are more or less equal as we change the brightness level from 50% to 75%. This implies that the network access mode do not impact the brightness levels in smartphones and hence the two parameters can be considered as independent.

5.8 Limitations

We describe the limitations of energy performance testing in this section, which are mainly due to inherent characteristics of the smartphones and their operating environments.

- Device Specifications: A device may be popular among users for some of the *basic* (G_0) parameters, such as display size, display intensity, user-friendliness and operating system. As a result, the energy cost of some applications can be high. The energy performance is not a concern in such cases. This phenomenon can be seen as a trade-off between energy cost and user perception, over which we have no control.

- Network Condition: We used a stand-alone access point (AP) to conduct WiFi related experiments. However, we had no control over the EDGE and 3G networks. Due to the fluctuations in the channel condition, variation in the results with space and time is inevitable.

- Battery Properties: A constant voltage source was used during testing to power the smartphones. However, the devices run on batteries, and real batteries are non-linear devices [104]. Therefore, the measured cost metric (θ) will be slightly different from the obtained energy cost metric from real batteries.

- Stand-alone Parameters: Same set of experiments should be conducted for all the four devices in order to compare the test results. We set similar values for primary parameters on all devices accordingly. However, it was not possible to obtain similar settings for the stand-alone parameters because all the stand-alone parameters were not available on all the devices.

We used the default applications available on the smartphone platforms to conduct the experiments. Executing compatible versions of the same application (*i.e.,* same design is followed in every version) on the smartphone platforms is logical for fair comparison. However, our objective is to evaluate the performance from user perspective, thus default applications were purposely chosen to conduct the experiments.

B_i	Parameters	Description	BB 9700	HTC Nexus One	Nokia E71	HTC HD2	iPhone 3GS
01	Processor	Processor class and speed	XScale 624 MHz processors	Qualcomm Snapdragon QSD8250 1 GHz	ARM 11 369 MHz processor	Qualcomm Snapdragon QSD8250 1 GHz	ARM Cortex A8 600 MHz, PowerVR SGX535 graphics
02	RAM	Size of RAM	256 MB RAM	512MB RAM	128 MB RAM	448 MB RAM	256 MB RAM
03	Display	Size of display	480 x 360 pixels 2.44"	480 x 800 pixels 3.7"	320 x 240 pixels 2.36"	480 x 800 pixels 4.3"	320 x 480 pixels 3.5"
04	Operating System (OS)	Name of the OS	BlackBerry OS	Android	Symbian	Windows CE	iPhone OS 3
05	Battery	Type & capacity	Li-Ion 1500 mAh	Li-Ion 1400 mAh	Li-Po 1500 mAh	Li-Ion 1230 mAh	Li-Ion 1250 mAh

Table 5.4: Examples of *basic* parameters (G_0)

5.9 Summary

We proposed a methodology to measure the energy performances of smartphones and *network related applications* by means of reduced number of test cases. Such a testing

B_i	Parameters	Description	BB 9700	HTC Nexus One	Nokia E71	HTC HD2	iPhone 3GS
31	Volume	Allows the user to change the volume level of the device	Volume levels (0 to 10)	Option 1: Sounds (Silent) Option 2: levels: (0 to 15)	Volume levels (0 to 10)	Option 1: Sounds (Silent) Option 2: levels (0 to 15)	Volume levels (0, 1 to 16)
32	Brightness	Allows the user to change the brightness level of the device	Brightness (0, 10, 20 to 100)	Brightness Continuously (0 to 100%)	Display Light Sensor (0, 25, 50, 75, 100)	Brightness Continuously (0 to 100%)	Brightness Continuously (0 to 100%)
33	Color Contrast	Allows the user to select the color contrast of the display	Contrast (Normal, Reverse contrast, Grey Scale)	No	Different Color Theme	No	No
34	Bluetooth	Allows the user to turn on/off their bluetooth connection whenever required.	ON/OFF	ON/OFF	ON/OFF	ON/OFF	ON/OFF
35	GPS	Allows the user to turn on/off their GPS connection whenever required.	Location Data (Enable / Disabled)	Use GPS satellites (Select / Deselect)	GPS data (navigation, position, trip distance)	Location Service Settings (ON/OFF)	Location Services (ON/OFF)
36	Data Encryption	The user can enable or disable the encryption	Encryption (Enabled / Disabled)	No	Phone Memory/ Memory Card (ON/OFF)	Alternative (Encrypt files when placed on storage)	No
37	Data Access Mode	Allows the user to select from WiFi/EDGE/3G connections	Yes	Yes	Yes	Yes	Yes

Table 5.5: Examples of *active* parameters (G_1)

framework will help the system designers to come up with better design and implementations of software applications. The challenges associated with the measurement of energy costs and the comparison of such costs across smartphones were addressed thoroughly. We identify the potential parameters and categorize them into different groups. Four different state-of-the-art smartphones were used to conduct test on our energy measurement test bench.

B_i	Parameters	Description	BB 9700	HTC Nexus One	Nokia E71	HTC HD2	iPhone 3GS
61	Airplane Mode	Allows the user to disable all wireless connection	ON/OFF	Select / Deselect	No	ON/OFF	ON/OFF
62	Maps	User can let the address of the device recognizable	ON/OFF	No	Alternative (Select GPS / Maps)	No	Alternative (Select Maps)
63	Browser Push	Allows the user to enable Push, WAP, MDS	ON/OFF	No	No	No	No
64	Network Selection Mode	Let device to select the network manually or automatically	Auto/Manual	Search automatically	Manual/Automatic	Select / Deselect / Auto	Alternative Carriers (Automatic / Select / Deselect)
65	WiFi Settings -Network Notification	Prompts the user whenever any network notifications are available	Alternative (Prompt when manual login is required)	ON/OFF	Option 1: Show WLAN availability (Yes/No) Option 2: Scan for Networks (1 to 10 min)	No	ON/OFF
66	Bluetooth - Discoverable	Makes the device discoverable by other Bluetooth devices	No	ON/OFF	Shown to all, Define Period (1 to 60 min), Hidden	No	No
67	Scanning Other Bluetooth Devices	Scans other Bluetooth devices in the vicinity	Search/Listen	Scan for devices	No	Search	No
68	Portable WiFi Hotspot	Leads the mobile to act as a WiFi hotspot	No	Portable Wi-Fi Hotspot (Select / Deselect)	No	Alternative Internet sharing (Select / Deselect)	Set up Internet Tethering
69	Pulse Notification / Light Event Sounds	A click sound will be made whenever some events take place	Event Sounds (ON/OFF)	Vibrate when pressing soft keys and on certain UI interactions	Alternative Blink light for (Off,5,15,30hrs)	Alternative Notification Sounds (reminders, new message, new email, voice mail)	ON/OFF
70	Automatic Brightness	Allows the user to set the brightness to default level	No	ON/OFF	No	ON/OFF	ON/OFF
71	Automatic Dim Backlight	Dims the display light automatically	ON/OFF	No	Light time-out (5 to 60 sec)	Dim backlight [10 sec to 5 min] (battery) & [1 to 10min] (external power)	No
72	Auto-rotate Screen	The screen rotates to have two orientations of display	No	ON/OFF	No	No	Alternative (Screen automatically auto-rotates)
73	Animations	Some window animations are shown	No	Animation (No, Some, All)	Alternative (None, Playing, Animation)	No	No
74	Screen Timeout	Allows the user to set the display timeout for the screen	Timeout (10 sec to 2 Min)	Timeout (15 sec to 30 min)	Timeout (5 to 90 sec)	Timeout [1 to 10 min] (battery) & [1 to 30 min] (external power)	Timeout (1 - 5 min, Never)
75	LED Coverage Indicator	Indicates the presence of network	ON/OFF	No	No	Alternative G-Sensor (ON/OFF)	No
76	Use Location Aiding For GPS	Uses addition network information for GPS	Location Aiding (Enabled / Disabled)	No	No	Alternative HTC Location service (Enabled / Disabled)	Location Services (ON / OFF)
77	Security - Firewall	User can enable or disable the Firewall	Firewall (Enabled / Disabled)	No	No	No	No

Table 5.6: Examples of *passive* parameters (G_2)

Chapter 6

Conclusions and Future Directions

6.1 Conclusions

Smartphones are emerging as the preferred personal gadget of users. Increasing number of resource intensive software applications are draining their limited energy resource. As a result, extending the battery life of smartphones has been in the focus of researchers for more than a decade. Numerous energy management techniques have been investigated at different levels of system design, starting from silicon at the bottom, to application design at the top, with communication protocols and operating systems in between. However, the well-known end-to-end argument suggests that functions placed at low levels of a system may be redundant or of little value when compared with the cost of providing them at that low level. In portable devices, software applications use hardware resources through the operating system, which in turn consume battery power when they are in use. Thus, being on the top level, software applications play the most important role in the energy consumption of smartphones, and as such draw considerable attention for research.

The challenges in designing energy-efficient software applications are described in Chapter 1. In the mobile application development process, there is a need for high-level energy cost information so that designers can apply that during design phase. From the designers or developers point of view, the impact of design decisions on the energy cost is important. Thus fewer design choices need to change after the initial energy performance test of an application. A standard energy evaluation test bench is also crucial for comparing energy test results of smartphones performance tests. Specifying the values of different smartphone parameters during a test, and reducing the number of test configurations are two potential concerns in this regard.

In this thesis, we have addressed the above two major issues related to mobile application development process. In addition, we have addressed an *application level energy management* technique that enhances the functionality of smartphones by accessing resources on other devices. We have also investigated an *energy-efficient communication* strategy that proposed a data packet aggregation algorithm based on different *burst* parameters. To observe the efficacy of the approach, WiFi Internet traffic data of different smartphones were observe and analyzed. A summary of each of the strategies is described below.

We presented a finite state machine (FSM) based model to estimate the energy cost of mobile applications. We provided evidence from an energy measurement test bench to explain the model parameters. We proposed the use of smartphone device emulators to estimate energy costs and discuss the rationale behind doing so. To help designers in making decisions at an early design phase, the concept of *energy cost profile* of a device was introduced, which provides the energy cost information of different hardware components so that designers may take measures to improve their designs and make their applications more energy efficient. We have conducted experiments on our energy measurement testbed comprising of a number of state-of-the-art smartphones to provide practical examples of energy cost profiles for devices.

The proposed *UCCI* framework facilitates communication between two wireless portable devices such as smartphones and laptops. This framework allows a server device to remain in the *sleep* state unless its service is needed by a client device. It exploits the 'Wake On' feature of the server device, uses the personal area low power Bluetooth link, and optionally returns the server back to the *sleep* state. To validate our model, we have developed two prototypes using state-of-the-art *BlackBerry 9700* and *HTC Nexus One* smartphones. We investigated the potential energy saving issues by anatomizing the WiFi access data traffic of smartphones, and a packet aggregation scheduler has been proposed to keep the communication interface in *sleep* states as long as possible.

To evaluate the energy cost evaluation of smartphone applications, we have defined the concept of a user level test case for smartphones and showed that, due to configuration settings, there exist millions of such test cases. We proposed a test selection technique to reduce the number of test cases. The proposed technique is applied to four different smartphones and their energy costs are evaluated for running common *network related applications*. We have developed a state-of-the-art test bench to execute those test cases for real applications on smartphones, and measure their actual energy costs.

We have shown the efficacy of our proposed ideas by means of extensive experiments throughout the thesis. The results of this thesis can be used by application developers

to make implementation level decisions that allow the implementation of energy efficient software applications, leading to the designing of energy efficient smartphones.

6.2 Future Directions

The need for energy efficiency in smartphones will become more acute in future for the following reasons.

1. **Growing Miniaturization of Hardware Components**: Hardware components are becoming increasingly smaller with more capacity. Smartphones with a quad-core processor are already available in the market. On the other hand, the space for battery is reducing with overall size of the devices.

2. **Addition of more Sensors:** New components are being added to the devices over time. The accelerometer, proximity sensor, near field communication (NFC) are few examples, and more sensors means more energy consumers in the system.

3. **Diversification of Usage:** With the development of new applications and tools, the usage of the smartphones will be further diversified. Users will use smartphones for mobile payment, GPS navigation, Camera and etc.

4. **Development of Energy-hungry Applications:** Everyday thousands of new applications are coming to application market-place to meet the user demand. Some types of application such as gaming and map based *apps* are resource intensive in terms of processing and communication need.

5. **Enhanced Data Transfer and Storage Security:** Security and integrity of data has been an important concern, and it will keep growing in future. Additional computation intensive measures will be required to satisfy the demand in this regard.

6. **Integration of New Technology:** New technologies will be integrated with smartphone such as *cognitive radio* which require constant energy for spectrum sensing.

This list is not exhaustive, these are just a few examples. Technology keeps changing at a rapid pace, and we need new and innovative approaches to cope with the changes. In the following, we have proposed a few potential extensions of our work and a couple of general issues that need to be addressed for building energy efficient smartphones. These points address the limitation of our approaches as well.

1. We have investigated the energy consumption of applications on top of operating systems. During the execution of the application system resources such as CPU and memory were readily available. However, in the presence of other applications, available resources may become saturated. For example, the OS may need to swap memory to accommodate an application, or an application may be kept in waiting for a while. This kind of situation is not rare, and it needs to be addressed to get a clear picture of energy consumption.

2. The cloud computing facilitates the offloading of computationally intensive tasks to cloud servers for the smartphones. However, issues such as when or in what situations offloading would be effective in terms of delay and energy are yet to be properly addressed. There is also a need for a generic framework for offloading task to a surrogate server.

3. When we analyzed the smartphone Internet traffic, we used only WiFi traffic, because we did not have access to 3G traffic data. To explore the energy saving potential of 3G data network for smartphones, availability of cellular data traces is essential.

4. In the design of energy performance testing, we were able to reduce the number of configurations, but the settings of device parameters still need to be changed to configure a device for testing. The task of setting values of parameters can be automated, and automation of the testing is another potential extension to our work.

5. During the experimentation, we used only one smartphone of different types. There is a need to observe the variation of energy consumption among the devices of same kind. For example, the variations of energy consumption on two BlackBerry 9700. This validation technique will add more confidence to our results.

6. To receive the full benefit of energy efficient software applications, operating systems need to properly coordinate the applications, and manage hardware components. Thus, a system-wide integrated energy saving framework is essential. This feature is still in its infancy.

7. Finally, there is the need for a review of existing software engineering methodologies for mobile application development. New methodologies need to be augmented with existing software engineering processes to address the performance testing issues related to the mobile application development cycle.

List of Publications

Patent Applications

P01 US Patent Filed. 2008. Subject: Estimating Energy Cost of Software Applications on Communications Devices.

P02 US Patent Filed. 2010. Subject: Method and Apparatus Pertaining to Offloading Task Execution.

Journal Papers (Accepted and Submitted)

J01 Rajesh Palit, Ajit Singh, Kshirasagar Naik. An Architecture for Enhancing Capability and Energy Efficiency of Wireless Handheld Devices, International Journal of Energy, Information and Communication (IJEIC), 2(4):117-136, November 2011.

J02 Rajesh Palit, Kshirasagar Naik, Ajit Singh, Renuka Arya. Designing User Level Test Cases for Energy Performance Evaluation of Smartphones, submitted as an invited paper to special issues of Journal of Systems and Software (JSS) for Automation of Software Test, November, 2011.

J03 Rajesh Palit, Kshirasagar Naik, Ajit Singh. Anatomy of WiFi Access Traffic of Smartphones and Implications for Energy Saving Techniques, submitted to Elsevier Computer Communications, October, 2011.

J04 Rajesh Palit, Kshirasagar Naik, Ajit Singh. Modeling and Evaluating Energy Performance of Software Applications on Smartphones, submitted to Journal of Pervasive and Mobile Computing (Elsevier), November, 2011.

Conference Papers

C01 Rajesh Palit, Ajit Singh, Kshirasagar Naik. Enhancing the Capability and Energy Efficiency of Smartphone using WPAN, in the 22nd IEEE Symposium on Personal, Indoor, Mobile and Radio Communications (IEEE PIMRC 2011), pages 1025–1030, September 2011.

C02 Rajesh Palit, Kshirasagar Naik, and Ajit Singh. Impact of Packet Aggregation on Energy Consumption in Smartphones, in the 11th International Wireless Communications and Mobile Computing Conference (IWCMC 2011), pages 589–594, July 2011.

C03 Rajesh Palit, Renuka Arya, Kshirasagar Naik, and Ajit Singh. Selection and Execution of User Level Test Cases for Energy Cost Evaluation of Smartphones, in the proceedings of ICSE Workshop on Automation of Software Test (ACM/IEEE AST 2011), pages 84–90, May 2011.

C04 Rajesh Palit, Kshirasagar Naik, and Ajit Singh. Estimating the Energy Cost of Communication on Portable Wireless Devices, in the proceedings of IFIP/IEEE Wireless Days 2008, pages 1–5, November 2008.

C05 Rajesh Palit, Ajit Singh, and Kshirasagar Naik. Modeling the Energy Cost of Applications on Portable Wireless Devices, in the proceedings of ACM MSWiM 2008, pages 346–353, October 2008.

C06 Rajesh Palit, Paul S Ward, Ajit Singh and Kshirasagar Naik. Energy-aware Cooperative (ECO) Relay-Based Packet Transmission in Wireless Networks, in the proceedings of IEEE WCNC 2008, pages 2875–2880, April 2008.

C07 Renuka Arya, Rajesh Palit, and Kshirasagar Naik. A Methodology for Selecting Experiments to Measure the Energy Consumptions in Smartphones, in the 11th International Wireless Communications and Mobile Computing Conference (IWCMC 2011), pages 2087–2092, July, 2011.

C08 Rajesh Palit, Majid Altamimi, Kshirasagar Naik, Ajit Singh. Challenges and Opportunities in Designing Next Generation Energy-efficient Smartphones, in the proceedings of the 1st International Conference on Advanced Computing (ICoAC 2011), plenary paper, December 2011.

Conference Papers (Submitted)

C10 Rajesh Palit, Abdulhakim Abogharaf, Kshirasagar Naik, Ajit Singh. A Generalized Strategy for Selecting User Level Test Cases for Energy Performance Evaluation of Smartphones, submitted to the 34th International Conference on Software Engineering (ICSE 2012), Zurich, Switzerland, June 2–9, 2012.

Book Chapter

B01 Rajesh Palit, Ajit Singh, Kshirasagar Naik. Energy Costs of Software Applications on Portable Wireless Devices, as a chapter in the book Energy Scavenging and Optimization techniques for Mobile Devices by CRC Press, Taylor and Francis Group, USA, March 2012.

References

[1] Andrea Acquaviva, Tajana Simunic, Vinay Deolalikar, and Sumit Roy. Remote power control of wireless network interfaces. *Journal of Embedded Computing*, 1:381–389, August 2005.

[2] Pranav Agrawal, Anurag Kumar, and Ramachandran Ramjee. OPSM - opportunistic power save mode for infrastructure IEEE 802.11 WLAN. In *Proceedings of the IEEE International Conference on Communications Workshops*, pages 1–6, May 2010.

[3] Giuseppe Anastasi, Marco Conti, Enrico Gregori, Andrea Passarella, and Luciana Pelusi. An energy-efficient protocol for multimedia streaming in a mobile environment. *International Journal of Pervasive Computing and Communications*, pages 301–312, March 2005.

[4] Renuka Arya, Rajesh Palit, and Kshirasagar Naik. A methodology for selecting experiments to measure energy costs in smartphones. In *Proceedings of the 7th International Wireless Communications and Mobile Computing Conference*, IWCMC '11, pages 2087–2092, July 2011.

[5] Hakan Aydin, Rami Melhem, Daniel Mosse, and Pedro Mejia-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 53(5):584–600, May 2004.

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

[7] Valeria Baiamonte and Carla F. Chiasserini. Investigating MAC-layer schemes to promote doze mode in 802.11-based WLANs. In *Proceedings of the IEEE 58th Vehicular Technology Conference*, pages 3:1568–3:1572, October 2003.

[8] 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 IMC*, pages 280–293, November 2009.

[9] Kutty S. Banerjee and Emmanuel Agu. PowerSpy: fine-grained software energy profiling for mobile devices. In *Proceedings of the International Conference on Wireless Networks, Communications and Mobile Computing*, pages 2:1136–2:1141, June 2005.

[10] CPU benchmark Programs. `http://www.cse.dmu.ac.uk/~bb/Teaching/ComputerSystems/SystemBenchmarks/BenchMarks.html`.

[11] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transaction on Very Large Scale Integrated Systems*, 8:299–316, June 2000.

[12] Jose F. M. Bernal, Luca Ardito, Maurizio Morisio, and Paolo Falcarin. Towards an efficient context-aware system: Problems and suggestions to reduce energy consumption in mobile devices. In *Proceedings of the International Conference on Mobile Business and Global Mobility Roundtable*, pages 510–514, June 2010.

[13] Davide Bertozzi, Anand Raghunathan, Luca Benini, and Srivaths Ravi. Transport protocol optimization for energy efficient wireless embedded systems. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, pages 706–711, March 2003.

[14] Chris J. Bleakley, Miguel Casas-Sanchez, and Jose Rizo-Morente. Software level power consumption models and power saving techniques for embedded DSP processors. *Journal of Low Power Electronics*, 2(2):281–290, August 2006.

[15] Jiang Bo, Long Xiang, and Gao Xiaopeng. MobileTest: a tool supporting automatic black box test for software on smart mobile devices. In *Proceedings of the 2nd International Workshop on Automation of Software Test*, AST '07, pages 37–43, May 2007.

[16] Lawrence S. Brakmo, Deborah A. Wallach, and Marc A. Viredaz. μsleep: A technique for reducing energy consumption in handheld devices. In *Proceedings of the MobiSys*, pages 48–56, June 2004.

[17] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the USENIX Annual Technical Conference*, USENIXATC '10, pages 1–14, June 2010.

[18] Ozgur Celebican, Tajana S. Rosing, and Vincent J. Mooney. Energy estimation of peripheral devices in embedded systems. In *Proceedings of the Great Lakes Symposium on VLSI*, pages 430–435, April 2004.

[19] Naehyuck Chang, Inseok Choi, and Hojun Shim. DLS: dynamic backlight luminance scaling of liquid crystal display. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(8):837–846, August 2004.

[20] Naehyuck Chang, Kwanho Kim, and Hyung Gyu Lee. Cycle-accurate energy consumption measurement and analysis: case study of ARM7TDMI. In *Proceedings of the International symposium on Low power electronics and design*, ISLPED '00, pages 185–190, July 2000.

[21] Baiqiang Chen, Jun Yan, and Jian Zhang. Combinatorial testing with shielding parameters. In *Proceedings of the Asia-Pacific Software Engineering Conference*, pages 280–289, December 2010.

[22] Guangyu Chen, Byung-Tae Kang, Mahmut Kandemir, Narayanan Vijaykrishnan, and Rajarathnam Chandramouli. Studying energy trade offs in offloading computation/compilation in java-enabled mobile devices. *IEEE Transaction on Parallel Distributed Systems*, 15(9):795–809, September 2004.

[23] Wei-Chung Cheng and Massoud Pedram. Power minimization in a backlit TFT-LCD display by concurrent brightness and contrast scaling. *IEEE Transactions on Consumer Electronics*, 50(1):25–32, February 2004.

[24] Carla F. Chiasserini and Ramesh R. Rao. Energy efficient battery management. *IEEE Journal on Selected Areas in Communications*, 19(7):1235–1245, July 2001.

[25] Carla F. Chiasserini and Ramesh R. Rao. Improving battery performance by using traffic shaping techniques. *IEEE Journal on Selected Areas in Communications*, 19(7):1385–1394, July 2001.

[26] Kyoung Youn Cho, Subhasish Mitra, and Edward J. McCluskey. Gate exhaustive testing. In *Proceedings of the IEEE International Test Conference (ITC)*, pages 1–7, November 2005.

[27] Sayantan Choudhury, Irfan Sheriff, Jerry D. Gibson, and Elizabeth M. Belding-Royer. Effect of payload length variation and retransmissions on multimedia in 802.11a WLANs. In *Proceedings of IWCMC*, pages 377–382, July 2006.

[28] Sue H. Chow, Yi C. Ho, and Tingting Hwang. Low power realization of finite state machines a decomposition approach. *ACM Transactions on Design Automation of Electronic Systems*, 1(3):315–340, July 1996.

[29] Cisco. Cisco visual networking index: Global mobile data traffic forecast update, 2009-2014. White Paper FLGD 08867, Cisco, February 2010.

[30] Joel Coburn, Srivaths Ravi, and Anand Raghunathan. Power emulation: a new paradigm for power estimation. In *Proceedings of the 42nd Design Automation Conference*, pages 700–705, June 2005.

[31] David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton. The AETG system: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444, July 1997.

[32] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Transactions on Software Engineering*, 34(5):633–650, September 2008.

[33] Lossless data compression software benchmarks. `http://www.maximumcompression.com/`.

[34] Gerard Bosch I. Creus and Petri Niska. System-level power management for mobile devices. In *Proceedings of the 7th International Conference on Computer and Information Technology*, pages 799–804, October 2007.

[35] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. MAUI: making smartphones last longer with code offload. In *Proceedings of the MobiSys*, pages 49–62, June 2010.

[36] Jacek Czerwonka. Pairwise testing in real world: Practical extensions to test case generators. In *Proceedings of the 24th Pacific Northwest Software Quality Conference*, pages 419–430, October 2006.

[37] Marcio E. Delamaro, Auri Marcelo R. Vincenzi, and Juan C. Maldonado. A strategy to perform coverage testing of mobile applications. In *Proceedings of the International workshop on Automation of Software Test*, AST '06, pages 118–124, May 2006.

[38] Giovanni Denaro, Andrea Polini, and Wolfgang Emmerich. Early performance testing of distributed software applications. In *Proceedings of the 4th International Workshop on Software and Performance*, pages 94–103, January 2004.

[39] Madhav P. Desai, Hariharan Narayanan, and Sachin B. Patkar. The realization of finite state machines by decomposition and the principal lattice of partitions of a submodular function. *Discrete Applied Mathematics*, 131(2):299–310, September 2003.

[40] Fahad R. Dogar, Peter Steenkiste, and Konstantina Papagiannaki. Catnap: exploiting high bandwidth wireless interfaces to save energy for mobile devices. In *Proceedings of the MobiSys*, pages 107–122, June 2010.

[41] Mian Dong and Lin Zhong. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In *Proceedings of the MobiSys*, pages 335–348, June 2011.

[42] Android emulator. Available online: `http://developer.android.com/guide/developing/devices/emulator.html`, October 2011.

[43] Kamran Etemad. Overview of mobile wimax technology and evolution. *IEEE Communications Magazine*, 46(10):31–40, October 2008.

[44] Hossein Falaki, Dimitrios Lymberopoulos, Ratul Mahajan, Srikanth Kandula, and Deborah Estrin. A first look at traffic on smartphones. In *Proceedings of the 10th annual conference on Internet measurement*, pages 281–287, November 2010.

[45] Hossein Falaki, Ratul Mahajan, Srikanth Kandula, Dimitrios Lymberopoulos, Ramesh Govindan, and Deborah Estrin. Diversity in smartphone usage. In *Proceedings of the MobiSys*, pages 179–194, June 2010.

[46] Laura M. Feeney and Martin Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proceedings of IEEE INFOCOM*, pages 3:1548–3:1557, April 2001.

[47] Denzil Ferreira, Anind K. Dey, and Vassilis Kostakos. Understanding human-smartphone concerns: a study of battery life. In *Proceedings of the 9th International conference on Pervasive computing*, pages 19–33, June 2011.

[48] Jason Flinn and Mahadev Satyanarayanan. Energy-aware adaptation for mobile applications. *SIGOPS Operating Systems Review*, 33:48–63, December 1999.

[49] Jason Flinn and Mahadev Satyanarayanan. PowerScope: a tool for profiling the energy usage of mobile applications. In *Proceedings of the 2nd IEEE Workshop on Mobile Computer Systems and Applications*, WMCSA '99, pages 2–10, February 1999.

[50] Matthew S. Gast. *802.11 Wireless Networks: The definitive Guide*. O'Reilly, second edition, April 2005.

[51] Andreas Genser, Christian Bachmann, Josef Haid, Christian Steger, and Reinhold Weiss. An emulation-based real-time power profiling unit for embedded software. In *Proceedings of the International Symposium on Systems, Architectures, Modeling, and Simulation*, SAMOS '09, pages 67–73, July 2009.

[52] Savvas Gitzenis and Nicholas Bambos. Joint task migration and power management in wireless computing. *IEEE Transactions on Mobile Computing*, 8(9):1189–1204, September 2009.

[53] Xiaohui Gu, Alan Messer, Ira Greenberg, Dejan Milojicic, and Klara Nahrstedt. Adaptive offloading for pervasive computing. *IEEE Pervasive Computing*, 3(3):66–73, July 2004.

[54] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mary J. Irwin, Narayanan Vijaykrishnan, and Mahmut Kandemir. Using complete machine simulation for software power estimation: the SoftWatt approach. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, pages 141–150, February 2002.

[55] Selim Gurun, Rich Wolski, Chandra Krintz, and Dan Nurmi. On the efficacy of computation offloading decision-making strategies. *International Journal of High Performance Computing Applications*, 22(4):460–479, November 2008.

[56] Josef Haid, Christian Bachmann, Andreas Genser, Christian Steger, and Reinhold Weiss. Power emulation: Methodology and applications for hw/sw power optimization. In *Proceedings of the 8th IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pages 133–138, July 2010.

[57] Josef Haid, Gerald Kaefer, Christian Steger, and Reinhold Weiss. Run-time energy estimation in system-on-a-chip designs. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 595–599, January 2003.

[58] Paul J.M. Havinga and Garard J.M. Smit. QoS scheduling for energy-efficient wireless communication. In *Proceedings of the International Conference on Information Technology: Coding and Computing*, pages 167–171, April 2001.

[59] Paul J.M. Havinga and Gerard J.M. Smit. Energy-efficient wireless networking for multimedia applications. *Wireless Communications and Mobile Computing, Wiley*, 1(2):165–184, March 2001.

[60] Jung Ha Hong and Khosrow Sohraby. On modeling, analysis, and optimization of packet aggregation systems. *IEEE Transactions on Communications*, 58(2):660–668, February 2010.

[61] Jeff Hopper and Jim Wilson. Developers guide to the arm emulator. Available online: `http://msdn.microsoft.com/en-us/library/bb630224.aspx`, July 2007.

[62] Guoqiang Hu, Klaus Dolzer, and Christoph Gauger. Does burst assembly really reduce the self-similarity? In *Proceedings of the Optical Fiber Communications Conference*, OFC '03, pages 1:124–1:126, March 2003.

[63] Junxian Huang, Qiang Xu, Birjodh Tiwana, Z. Morley Mao, Ming Zhang, and Paramvir Bahl. Anatomizing application performance differences on smartphones. In *Proceedings of the MobiSys*, pages 165–178, June 2010.

[64] Mostafa E. A. Ibrahim, Markus Rupp, and Hossam A. H. Fahmy. A precise high-level power consumption model for embedded systems software. *EURASIP Journal of Embedded Systems*, 2011:1:1–1:14, January 2011.

[65] Sergio Ilarri, Eduardo Mena, and Arantza Illarramendi. A system based on mobile agents to test mobile computing applications. *Journal of Network and Computer Applications*, 32(4):846–865, July 2009.

[66] Texus Instrument. Analyzing target system energy consumption in *Code Composer Studio*^{TM} IDE. Technical Report SPRA074A, Texus Instrument, November 2002. Software Development Systems.

[67] Key global telecom indicators for the world telecommunication service sector. Available online: `http://www.itu.int/ITU-D/ict/statistics/at_glance/KeyTelecom.html`, October 2010. International Telecommunication Union.

[68] Subu Iyer, Lu Luo, Robert Mayo, and Parthasarathy Ranganathan. Energy-adaptive display system designs for future mobile environments. In *Proceedings of the MobiSys*, pages 245–258, March 2003.

[69] Antti Jaaskelainen, Antti Kervinen, and Mika Katara. Creating a test model library for gui testing of smartphone applications (short paper). In *Proceedings of the 8th International Conference on Quality Software*, pages 276 –282, August 2008.

[70] Ravi Jain, David Molnar, and Zulfikar Ramzan. Towards a model of energy complexity for algorithms [mobile wireless applications]. In *Proceedings of the Wireless Communications and Networking Conference*, pages 3:1884–3:1890, March 2005.

[71] Saxena Jayashree and C.S. Ram Murthy. A taxonomy of energy management protocols for ad hoc wireless networks. *IEEE Communications Magazine*, 45(4):104–110, April 2007.

[72] Christine E. Jones, Krishna M. Sivalingam, Prathima Agrawal, and Jyh Cheng Chen. A survey of energy efficient network protocols for wireless networks. *Wireless Networks*, 7:343–358, September 2001.

[73] Joon-Myung Kang, Chang-Keun Park, Sin-Seok Seo, Mi-Jung Choi, and James Won-Ki Hong. User-centric prediction for battery lifetime of mobile devices. In *Proceedings of the 11th Asia-Pacific Symposium on Network Operations and Management: Challenges for Next Generation Network Operations and Service Management*, pages 531–534, October 2008.

[74] Aman Kansal and Feng Zhao. Fine-grained energy profiling for power-aware application design. *SIGMETRICS Performance Evaluation Review*, 36:26–31, August 2008.

[75] Thomas Karagiannis, Mart Molle, Michalis Faloutsos, and Andre Broido. A nonstationary poisson view of internet traffic. In *Proceedings of the IEEE International Conference on Computer Communications*, INFOCOM '04, pages 3:1558–3:1569, March 2004.

[76] Keithley website. `http://www.keithleycom/products/fasttransient/?mn=2304A`.

[77] Heejin Kim, Byoungju Choi, and W. Eric Wong. Performance testing of mobile applications at the unit test level. In *Proceedings of the 3rd IEEE International Conference on Secure Software Integration and Reliability Improvement*, SSIRI '09, pages 171–180, July 2009.

[78] Youngsoo Kim, Sunghyun Choi, Kyunghun Jang, and Hyosun Hwang. Throughput enhancement of IEEE 802.11 wlan via frame aggregation. In *Proceedings of the IEEE 60th Vehicular Technology Conference*, pages 4:3030–4:3034, September 2004.

[79] Ronny Krashinsky and Hari Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. *Wireless Networks*, 11:135–148, January 2005.

[80] Ulrich Kremer, Jamey Hicks, and James M. Rehg. A compilation framework for power and energy management on mobile computers. In *Proceedings of the 14th International Workshop on Languages and Compilers for Parallel Computing (LCPC)*, pages 1–12, August 2001.

[81] Rick Kuhn, Raghu Kacker, Yu Lei, and Justin Hunter. Combinatorial software testing. *IEEE Computer*, 42(8):94–96, August 2009.

[82] Marcello Lajolo, Anand Raghunathan, Sujit Dey, and Luciano Lavagno. Cosimulation-based power estimation for system-on-chip design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(3):253–266, June 2002.

[83] Ikhwan Lee, Hyunsuk Kim, Peng Yang, Sungjoo Yoo, Eui-Young Chung, Kyu-Myung Choi, Jeong-Taek Kong, and Soo-Kwan Eo. PowerViP: SoC power estimation framework at transaction level. In *Proceedings of the Asia and South Pacific Conference on Design Automation*, pages 551–558, January 2006.

[84] Yonghee Lee, Heejung Lee, and Heonshik Shin. Adaptive spatial resolution control scheme for mobile video applications. In *Proceedings of the IEEE International Symposium on Signal Processing and Information Technology*, pages 977–982, December 2007.

[85] Xiang Li, Wen-Zhan Song, and Weizhao Wang. A unified energy-efficient topology for unicast and broadcast. In *Proceedings of the ACM MobiCom*, pages 1–15, August 2005.

[86] Zhiyuan Li, Cheng Wang, and Rong Xu. Computation offloading to save energy on handheld devices: a partition scheme. In *Proceedings of the International conference on Compilers, architecture, and synthesis for embedded systems (CASES)*, pages 238–246, November 2001.

[87] Yuxia Lin and V.W.S. Wong. Frame aggregation and optimal frame size adaptation for ieee 802.11n wlans. In *Proceedings of the IEEE Global Telecommunications Conference*, pages 1–6, November 2006.

[88] Hao Liu, Yaoxue Zhang, and Yuezhi Zhou. TailTheft: leveraging the wasted time for saving energy in cellular communications. In *Proceedings of the 6th International workshop on MobiArch*, MobiArch '11, pages 31–36, June 2011.

[89] Jiayang Liu and Lin Zhong. Micro power management of active 802.11 interfaces. In *Proceeding of the MobiSys*, pages 146–159, June 2008.

[90] Zhifang Liu, Xiaopeng Gao, and Xiang Long. Adaptive random testing of mobile application. In *Proceedings of the 2nd International Conference on Computer Engineering and Technology (ICCET)*, pages 2:297–2:301, April 2010.

[91] Jacob R. Lorch and Alan J. Smith. Software strategies for portable computer energy management. *IEEE Personal Communications*, 5(3):60–73, June 1998.

[92] Khaled Mahmud, Masugi Inoue, Homare Murakami, Mikio Hasegawa, and Hiroyuki Morikawa. Energy consumption measurement of wireless interfaces in multi-service user terminals for heterogeneous wireless networks. *IEICE Transaction on Communication*, E88-B(3):1097–1110, March 2005.

[93] Gregor Maier, Fabian Schneider, and Anja Feldmann. A first look at mobile handheld device traffic. In *Proceedings of the 11th Passive and Active Measurement Conference*, pages 161–170, April 2010.

[94] Massimiliano Marcon, Marcel Dischinger, Krishna P. Gummadi, and Amin Vahdat. The local and global effects of traffic shaping in the Internet. In *Third International Conference on Communication Systems and Networks*, COMSNETS '11, pages 1–10, January 2011.

[95] Sue Marek. Battling the battery drain, January 2002. Wireless Internet Magazine.

[96] Darko Marinov, Alexandr Andoni, Dumitru Daniliuc, Sarfraz Khurshid, and Martin Rinard. An evaluation of exhaustive testing for data structures. Technical Report MIT-LCS-TR-921, MIT Computer Science and Artificial Intelligence Laboratory Report, 2003.

[97] Robert N. Mayo and Parthasarathy Ranganathan. Energy consumption in mobile devices: Why future systems need requirement-aware energy scale-down. Technical Report HPL-2003-167, Hewlett Packard Labs, 2007.

[98] Xenia Mountrouidou and Harry Perros. On the departure process of burst aggregation algorithms in optical burst switching. *Computer Networks*, 53:247–264, February 2009.

[99] Xenia Mountrouidou and Harry G. Perros. Characterization of the burst aggregation process in optical burst switching. In *Proceedings of Networking*, volume 3976 of *Lecture Notes in Computer Science*, pages 752–764, May 2006.

[100] Radu Muresan and Catherine Gebotys. Instantaneous current modeling in a complex VLIW processor core. *ACM Transactions on Embedded Computing Systems*, 4(2):415–451, May 2005.

[101] John D. Musa. Operational profiles in software-reliability engineering. *IEEE Software*, 10:14–32, March 1993.

[102] Anish Muttreja, Anand Raghunathan, Srivaths Ravi, and Niraj K. Jha. Automated energy/performance macro-modeling of embedded software. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(3):542–552, March 2007.

[103] Khirasagar Naik and Priyadarshi Tripathy. *Software testing and Quality Assurance: Theory and Practice*. John Wiley and Sons, Inc., First edition, August 2008.

[104] Kshirasagar Naik. A survey of software based energy saving methodologies for handheld wireless communication devices. Technical Report 2010-13, Department of Computer and Electrical Engineering, University of Waterloo, October 2010.

[105] Kshirasagar Naik and David S. L. Wei. Software implementation strategies for power-conscious systems. *Mobile Networks and Applications*, 6(3):291–305, June 2001.

[106] Suman Nath, Zachary Anderson, and Srinivasan Seshan. Choosing beacon periods to improve response times for wireless HTTP clients. In *Proceedings of the 2nd International workshop on Mobility management & wireless access protocols*, MobiWac '04, pages 43–50, October 2004.

[107] Mahadevamurty Nemani and Farid N. Najm. Towards a high-level power estimation capability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(6):588–598, June 1996.

[108] Changhai Nie and Hareton Leung. A survey of combinatorial testing. *ACM Computer Survey*, 43(2):11:1–11:29, February 2011.

[109] Nachi K. Nithi and Adriaan J. Wijngaarden. Smart power management for mobile handsets. *Bell Laboratory Technical Journal*, 15:149–168, March 2011.

[110] Nokia energy profiler. Website: `http://www.developer.nokia.com/Resources/Tools_and_downloads/Other/Nokia_Energy_Profiler/`.

[111] Jussi K. Nurminen. Parallel connections and their effect on the battery consumption of a mobile phone. In *Proceedings of the 7th IEEE Consumer Communications and Networking Conference*, CCNC '10, pages 1–5, January 2010.

[112] Jussi K. Nurminen and Janne Nyrnen. Energy-consumption in mobile peer-to-peer quantitative results from file sharing. In *Proceedings of the IEEE Consumer Communications and Networking Conference*, pages 729–733, January 2008.

[113] Rajesh Palit, Renuka Arya, Kshirasagar Naik, and Ajit Singh. Selection and execution of user level test cases for energy cost evaluation of smartphones. In *Proceeding of the 6th international workshop on Automation of Software Test*, AST '11, pages 84–90, April 2011.

[114] Rajesh Palit, Khirasagar Naik, and Ajit Singh. Estimating the energy cost of communication on portable wireless devices. In *Proceedings of the Ist IFIP Wireless Days*, WD '08, pages 1–5, November 2008.

[115] Rajesh Palit, Kshirasagar Naik, and Ajit Singh. Impact of packet aggregation on energy consumption in smartphones. In *Proceedings of the 11th International Wireless Communications and Mobile Computing Conference*, IWCMC '11, July 2011.

[116] Rajesh Palit, Ajit Singh, and Kshirasagar Naik. Modeling the energy cost of applications on portable wireless devices. In *Proceedings of the 11th international symposium on Modeling, analysis and simulation of wireless and mobile systems*, MSWiM '08, pages 346–353, October 2008.

[117] Rajesh Palit, Ajit Singh, and Kshirasagar Naik. Enhancing the capability and energy efficiency of smartphones using wpan. In *Proceedings of the 22nd IEEE Personal Indoor Mobile Radio Communications (PIMRC)*, pages 1025–1030, September 2011.

[118] Gian P. Perrucci, Frank H.P. Fitzek, Giovanni Sasso, Wolfgang Kellerer, and JJorg Widmer. On the impact of 2G and 3G network usage for mobile phones' battery life. In *Proceedings of the European Wireless Conference*, EW '09, pages 255–259, May 2009.

[119] Christian Poellabauer and Karsten Schwan. Energy-aware traffic shaping for wireless real-time applications. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 48–56, May 1994.

[120] Robert A. Powers. Batteries of low electronics. *Proceedings of IEEE*, 83(4):687–693, April 1995.

[121] Feng Qian, Zhaoguang Wang, Alexandre Gerber, Zhuoqing Mao, Subhabrata Sen, and Oliver Spatscheck. Profiling resource usage for mobile applications: a cross-layer approach. In *Proceedings of the MobiSys*, pages 321–334, June 2011.

[122] Moo-Ryong Ra, Jeongyeup Paek, Abhishek B. Sharma, Ramesh Govindan, Martin H. Krieger, and Michael J. Neely. Energy-delay tradeoffs in smartphone applications. In *Proceedings of the MobiSys*, pages 255–270, October 2010.

[123] Ravishankar Rao, Sarma Vrudhula, and Daler N. Rakhmatov. Battery modeling for energy aware system design. *IEEE Computer*, 36(12):77–87, December 2003.

[124] Manual de Vega Rodrigo and Jurgen Gotz. An analytical study of optical burst switching aggregation strategies. In *Proceedings of the 3rd International Workshop on Optical Burst Switching*, WOBS '04, pages 20–30, October 2004.

[125] Joshua J. Romero. Smartphones: The pocketable PC. *IEEE Spectrum Magazine*, January 2011.

[126] Peng Rong and Massoud Pedram. Extending the lifetime of a network of battery-powered mobile devices by remote processing: a markovian decision-based approach. In *Proceedings of the 40th annual Design Automation Conference*, pages 906–911, June 2003.

[127] Marcel C. Rosu, C. Michael Olsen, Chandra Narayanaswami, and Lu Luo. PAWP: a power aware web proxy for wireless LAN clients. In *Proceedings of the 6th IEEE Workshop on Mobile Computing Systems and Applications*, pages 206–215, December 2004.

[128] Eric Rozner, Vishnu Navda, Ramachandran Ramjee, and Shravan Rayanchu. Napman: Network-assisted power management for WiFi devices. In *Proceedings of the MobiSys*, pages 91–106, June 2010.

[129] Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. The remote processing framework for portable computer power saving. In *Proceedings of the 1999 ACM symposium on Applied computing*, SAC '99, pages 365–372, August 1999.

[130] Stephen M. Rumble, Ryan Stutsman, Philip Levis, David Mazières, and Nickolai Zeldovich. Apprehending joule thieves with cinder. *SIGCOMM Computer Communication Review*, 40:106–111, January 2010.

[131] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM Transaction on Computer Systems*, 2:277–288, November 1984.

[132] Mariogiovanna Sami, Donatella Sciuto, Cristina Silvano, and Vittorio Zaccaria. An instruction-level energy model for embedded vliw architectures. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 21(9):998 – 1010, September 2002.

[133] Ichiro Satoh. A testing framework for mobile computing software. *IEEE Transactions on Software Engineering*, 29(12):1112–1121, December 2003.

[134] Alaa Seddik-Ghaleb, Yacine Ghamri-Doudane, and Sidi-Mohammed Senouci. TCP computational energy cost within wireless mobile ad hoc network. In *Proceedings of the IEEE/ACS International Conference on Computer Systems and Applications*, AICCSA '09, pages 955–962, May 2009.

[135] Tamizh Selvam and Subramanian Srikanth. A frame aggregation scheduler for IEEE 802.11n. In *Proceedings of the National Conference on Communications (NCC)*, pages 1–5, January 2010.

[136] Sakura She, Sasindran Sivapalan, and Ian Warren. Hermes: A tool for testing mobile device applications. In *Proceedings of the Australian Software Engineering Conference*, pages 121–130, April 2009.

[137] Eugene Shih, Paramvir Bahl, and Michael j. Sinclair. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *Proceedings of the ACM MobiCom*, September 2002.

[138] Hojun Shim, Naehyuck Chang, and Massoud Pedram. A compressed frame buffer to reduce display power consumption in mobile systems. In *Proceedings of the Asia and South Pacific Design Automation Conference*, ASP-DAC '04, pages 818–823, January 2004.

[139] Hojun Shim, Youngjin Cho, and Naehyuck Chang. Frame buffer compression using a limited-size code book for low-power display systems. In *Proceedings of the 3rd Workshop on Embedded Systems for Real-Time Multimedia*, pages 7–12, September 2005.

[140] Victor Shnayder, Mark Hempstead, Bor rong Chen, Geoff Werner Allen, and Matt Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 188–200, November 2004.

[141] Alex Shye, Benjamin Scholbrock, and Gokhan Memik. Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 168–178, December 2009.

[142] Tajana Simunic and Stephen Boyd. Managing power consumption in networks on chips. In *Proceedings of the Design, Automation and Test Europe (DATE)*, pages 110–116, January 2004.

[143] Suresh Singh and Candy Yiu. Putting the cart before the horse: Merging traffic for energy. *IEEE Communications Magazine*, 49:78–82, June 2011.

[144] Rishi Sinha, Christos Papadopoulos, and John Heidemann. Internet packet size distributions: Some observations. Technical Report ISI-TR-2007-643, USC/Information Sciences Institute, May 2007.

[145] Dionysios Skordoulis, Qiang Ni, Hsiao-Hwa Chen, Adrian P. Stephens, Changwen Liu, and Abbas Jamalipour. IEEE 802.11n MAC frame aggregation mechanisms for next-generation high-throughput WLANs. *IEEE Wireless Communications*, 15(1):40–47, February 2008.

[146] Mark Stemm and Randy H. Katz. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Transactions on Communications*, 80:1125–1131, January 1997.

[147] Enhua Tan, Lei Guo, Songqing Chen, and Xiaodong Zhang. PSM-throttling: Minimizing energy consumption for bulk data communications in WLANs. In *Proceedings of the IEEE International Conference on Network Protocols*, pages 123–132, October 2007.

[148] Andrew Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, New Jersey, USA, 4th edition, 2002.

[149] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power analysis of embedded software: a first step towards software power minimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4):437–445, December 1994.

[150] Osman S. Unsal and Israel Koren. System-level power-aware design techniques in real-time systems. *Proceedings of the IEEE*, 91(7):1055–1069, July 2003.

[151] Keith S. Vallerio, Lin Zhong, and Niraj K. Jha. Energy-efficient graphical user interface design. *IEEE Transactions on Mobile Computing*, 5(7):846–859, July 2006.

[152] Cheng Wang and Zhiyuan Li. A computation offloading scheme on handheld devices. *Journal of Parallel and Distributed Computing*, 64(6):740–746, June 2004.

[153] Cheng Wang and Zhiyuan Li. Parametric analysis for adaptive computation offloading. *SIGPLAN Notes*, 39(6):119–130, June 2004.

[154] Le Wang and Jukka Manner. Energy consumption analysis of WLAN, 2G and 3G interfaces. In *Proceedings of the IEEE/ACM GREENCOM-CPSCOM*, pages 300–307, December 2010.

[155] Anthony I. Wasserman. Software engineering issues for mobile application development. In *Proceedings of the FSE/SDP workshop on future of software engineering research*, FoSER '10, pages 397–400, November 2010.

[156] Elaine J. Weyuker and Filippos I. Vokolos. Experience with performance testing of software systems: issues, an approach, and case study. *IEEE Transactions on Software Engineering*, 26(12):1147–1156, December 2000.

[157] Nick Wood. Mobile data traffic growth 10 times faster than fixed over next five years. *Total Telecom*, September 2009.

[158] Changjiu Xian, Yung-Hsiang Lu, and Zhiyuan Li. Adaptive computation offloading for energy conservation on battery-powered systems. *International Conference on Parallel and Distributed Systems*, 1:1–8, December 2007.

[159] Changjiu Xian, Yung-Hsiang Lu, and Zhiyuan Li. A programming environment with runtime energy characterization for energy-aware applications. In *Proceedings of the International symposium on Low power electronics and design*, ISLPED '07, pages 141–146, August 2007.

[160] Yu Xiao, Ramya S. Kalyanaraman, and Antti Yla-Jaaski. Energy consumption of mobile YouTube: Quantitative measurement and analysis. In *Proceedings of NG-MAST'08*, pages 61–69, September 2008.

[161] Rong Xu, Zhiyuan Li, Cheng Wang, and Peifeng Ni. Impact of data compression on energy consumption of wireless-networked handheld devices. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, pages 302–311, May 2003.

[162] Haijin Yan, Rupa Krishnan, Scott A. Watterson, and David K. Lowenthal. Client-centered energy savings for concurrent HTTP connections. In *Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '04, pages 62–67, June 2004.

[163] Shun-Ren Yang. Dynamic power saving mechanism for 3G UMTS system. *Mobile Network Applications*, 12:5–14, January 2007.

[164] Roy D. Yates and David J. Goodman. *Probability and Stochastic Processes*. John Wiley and Sons, Inc., Second edition, May 2005.

[165] Wei Ye, John Heidemann, and Deborah Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Transaction on Networking*, 12:493–506, June 2004.

[166] Wu Ye, Narayanan Vijaykrishnan, Mahmut Kandemir, and Mary J. Irwin. The design and use of simplepower: A cycle-accurate energy estimation tool. In *Proceedings of the Design Automation Conference (DAC)*, pages 340–345, August 2000.

[167] Yongfeng Yin, Bin Liu, Chen Wang, and Hongying Ni. Research on automatic testing technology oriented intelligent mobile terminal software. In *Proceedings of the International Conference on Communications and Mobile Computing (CMC)*, pages 1:274–1:278, April 2010.

[168] Masaki Yoshio, Ralph J. Brodd, and Akiya Kozawa. *Lithium-ion Batteries*. Science and Technology. Springer, First edition, January 2005.

[169] Xiang Yu, Yang Chen, and Chunming Qiao. A study of traffic statistics of assembled burst traffic in optical burst switched networks. In *Proceedings of the Optical Networking and Communication Conference*, pages 149–159, July 2002.

[170] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. ECOSystem: managing energy as a first class operating system resource. *SIGOPS Operating System Review*, 36:123–132, October 2002.

[171] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. Currentcy: a unifying abstraction for expressing energy management policies. In *Proceedings of the USENIX Annual Technical Conference*, pages 43–56, June 2003.

[172] Jiucai Zhang, Dalei Wu, Song Ci, Haohong Wang, and Aggelos K. Katsaggelos. Power-aware mobile multimedia: a survey. *Journal of Communications*, 4(9):600–613, October 2009.

[173] Lide Zhang, Birjodh Tiwana, Robert P. Dick, Zhiyun Qian, Zhuoqing M. Mao, Zhaoguang Wang, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the IEEE/ACM/IFIP CODES+ISSS*, pages 105–114, October 2010.

[174] Xiaoli Zhao, Pin Tao, Shiqiang Yang, and Fei Kong. Computation offloading for H.264 video encoder on mobile devices. In *Proceedings of the IMACS Multi-conference on Computational Engineering in Systems Applications*, pages 2:1426–2:1430, October 2006.

[175] Lin Zhong and Niraj K. Jha. Graphical user interface energy characterization for handheld computers. In *Proceedings of the International conference on Compilers, architecture and synthesis for embedded systems*, CASES '03, pages 232–242, October 2003.

[176] Yi-hua Zhu and Victor C. M. Leung. Efficient power management for infrastructure IEEE 802.11 WLANs. *IEEE Transaction on Wireless Communications*, 9:2196–2205, July 2010.