2013

Improving Energy Efficiency and Security for Pervasive Computing Systems

Fengyuan Xu
College of William & Mary - Arts & Sciences

Improving Energy Efficiency and Security for Pervasive Computing Systems

Fengyuan Xu

Zhengzhou, Henan, China

Bachelor of Science, Jilin University, 2007

A Dissertation presented to the Graduate Faculty
of the College of William and Mary in Candidacy for the Degree of
Doctor of Philosophy

Department of Computer Science

The College of William and Mary
August 2013

APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

Fengyuan Xu

Approved by the Committee, August 2013

Committee Chair
Associate Professor Qun Li, Computer Science
College of William and Mary

Professor Evgenia Smirni, Computer Science
College of William and Mary

Professor Weizhen Mao, Computer Science
College of William and Mary

Associate Professor Haining Wang, Computer Science
College of William and Mary

Senior Researcher Thomas Moscibroda
Microsoft Research

ABSTRACT

Pervasive computing systems are comprised of various personal mobile devices connected by the wireless networks. Pervasive computing systems have gained soaring popularity because of the rapid proliferation of the personal mobile devices. The number of personal mobile devices increased steeply over years and will surpass world population by 2016.

However, the fast development of pervasive computing systems is facing two critical issues, energy efficiency and security assurance. Power consumption of personal mobile devices keeps increasing while the battery capacity has been hardly improved over years. At the same time, a lot of private information is stored on and transmitted from personal mobile devices, which are operating in very risky environment. As such, these devices became favorite targets of malicious attacks. Without proper solutions to address these two challenging problems, concerns will keep rising and slow down the advancement of pervasive computing systems.

We select smartphones as the representative devices in our energy study because they are popular in pervasive computing systems and their energy problem concerns users the most in comparison with other devices. We start with the analysis of the power usage pattern of internal system activities, and then identify energy bugs for improving energy efficiency. We also investigate into the external communication methods employed on smartphones, such as cellular networks and wireless LANs, to reduce energy overhead on transmissions.

As to security, we focus on implantable medical devices (IMDs) that are specialized for medical purposes. Malicious attacks on IMDs may lead to serious damages both in the cyber and physical worlds. Unlike smartphones, simply borrowing existing security solutions does not work on IMDs because of their limited resources and high requirement of accessibility. Thus, we introduce an external device to serve as the security proxy for IMDs and ensure that IMDs remain accessible to save patients' lives in certain emergency situations when security credentials are not available.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

The work in this dissertation could not have been done without my advisor, Dr. Qun Li. I would like to express my sincere gratitude to him for his guidance and support.

I am grateful to my committee members, Dr. Evgenia Smirni, Dr. Weizhen Mao, Dr. Haining Wang, Dr. Thomas Moscibroda, for their valuable advice.

I appreciate all the help received from Vanessa Godwin, Jacqulyn Johnson and Dale Hayes.

I would like to thank my wife, father and mother. Their unconditional love encouraged me to step forward along my Ph.D. path.

I would also like to thank Wei Wei, Hao Han, Yifan Zhang, Zhenrui Qin, Edmund Novak, and many of my dear friends for their company and support.

To my wife Hui
To my father and mother
For their great love and support

LIST OF TABLES

LIST OF FIGURES

1 Introduction

Pervasive computing systems are comprised of various personal mobile devices connected by wireless networks. Some of these devices can be hand-held and fulfill all kinds of computational demands through intensive interactions with users. Other devices can be embedded into the surrounding environment or on the users' bodies to specifically support services with minimal human intervention. The combination of general and specific purpose devices bring rich opportunities for personal computing. Pervasive computing systems have gained soaring popularity because of the rapid proliferation of personal mobile devices. The number of devices has been steeply increasing over recent years and will surpass the world population by 2016.

However, the fast development of pervasive computing systems is facing two critical issues, energy efficiency and security assurance. As personal mobile devices start processing more complicated tasks, power consumption increases. However, these devices are typically battery-powered because mobility is a common theme in pervasive computing systems. Given the fact that battery technique has hardly improved, the increasing energy needs of pervasive computing is outpacing provided battery capacity. Mobile security is of growing importance as a lot of private user information is stored on personal mobile devices and transmitted through wireless communications. Without proper protections, users' privacy will be violated and useful computation will be limited. Concerns rising from the energy

and security problems have slowed down the advancement of pervasive computing systems.

We select smartphones for our energy study because smartphones are the most popular devices in pervasive computing systems. Thus, their energy problems are representative and the solutions we propose for smartphones can be generalized to other personal mobile devices. Moreover, the energy problem on smartphones is more serious in comparison with other devices. Smartphones are extremely energy-hungry because users tend to run computationally demanding tasks on them. At the same time, users still have a high expectation of the battery lifetime for their devices. This conflict cannot be mitigated by slowly improving battery techniques alone. Software optimization can save energy and maximize battery usage.

As to security, we focus on implantable medical devices (IMDs), which are specialized for medical purposes. There is a lot of sensitive patient data on IMDs and it is possible to physically influence the patients' health conditions through IMDs. Comprehensive security protections are necessary to prevent malicious attacks. However, unlike smartphones, borrowing existing security schemes on IMDs does not work because of their limited computational resources and high requirement of accessibility. Additionally, this problem urgently requires solutions because real attacks have been successfully performed on IMDs with the assistance of simple equipment. Action must be taken to prevent further damage. We also hope our research on this group of specialized devices can shed light on solving similar security problems.

To help solve the energy issue, we contribute three projects. We start with an analysis of the power pattern of internal system activities, and we improve the energy efficiency by fixing energy bugs that we found. We also investigate the external communication methods employed on smartphones, such as cellular networks and

wireless LANs, to reduce the energy overhead on data transmissions.

System power models are important to power management and optimization on smartphones. However, existing approaches for power modeling have several limitations. Some approaches require external power meters, which are not convenient for people to use. Others either rely on the internal battery current sensing hardware, which is not available on many smartphones, or take a long time to generate power models. To overcome these limitations, we propose a new way of generating power models from battery voltage dynamics called V-edge. V-edge is self-constructive and does not require current-sensing. Most importantly, it builds models quickly. Our implementation supports both component-level power models and per-application energy accounting. Evaluation results using various benchmarks and applications show that the V-edge approach achieves high power modeling accuracy, and is two orders of magnitude faster than existing self-modeling approaches requiring no current-sensing hardware.

In our second project, we build upon our system power modeling technique mentioned above to improve the energy efficiency of popular applications on smartphones. Specifically, we investigate email sync over 3G. According to our study, a large portion of energy is wasted while the smartphone receives messages in sleep mode. We perform a thorough analysis of different email sync techniques, across different mobile platforms, and we identify common energy usage problems in them. Then we propose a general design guideline for developers to improve the energy consumption of other programs on smartphones.

In our third project, we propose an online access point association algorithm for pervasive computing systems. It runs in a decentralized manner and is easy to deploy without any change to the infrastructure. This project is motivated because selecting a suitable access point (AP) is important to pervasive computing devices using WiFi. When contentions occur because of unwise AP association, the link

quality is severely degraded and more energy is wasted on data re-transmissions.

Last but not least, the rapid development in pervasive computing is blurring the boundary between the physical and cyber world. This trend makes security issues critical because malicious attacks are now able to threaten human life. For example, wireless access to IMDs today is unprotected and open to any one. Recently, real attacks have been demonstrated that a patient with an IMD could die because of this vulnerability. It is not appropriate to simply enforce traditional security on IMDs due to concerns for patient safety. 24/7 protection via standard encryption stops malicious actions, but also rigidly prevents legitimate doctors, who do not have security credentials, from accessing IMDs to save the patient's life in an unexpected emergency. Therefore, our fourth contribution is to design a context-aware security scheme tailored for heart-related IMDs. There are two novel techniques incorporated. One is ECG based key paring without prior shared secrets, and the other is an access control mechanism resilient to the spoofing attacks in this context.

Specifically we address four challenging problems as follows.

1.1 Energy Modeling on the Smartphone

Smartphones already commanded 63 percent market share in the United State according to a recent report, and the percentage keeps increasing. Compared with their predecessor, featured phones, the large demand of energy has stretched their battery capacities to the limit. Frequent re-charging becomes an annoying part of daily smartphone usage. The application using WiFi is a major factor that leads to this energy starving. In order to improve the power consumption of this kind of application, it is important to first understand where the energy is drained with the use of WiFi based applications. Bottlenecks of the power consumption are then

able to be identified and solved. Thus, we propose a fine-grained power modeling on smartphones to fulfill this demand.

There are three design requirements of our modeling approach. The first requirement is fine granularity. Each major hardware component should have its own power model. This is the key building block to track the energy consumption of active applications on the smartphone. The second one is self-construction. Slight changes in hardware and software configuration may lead to different modeling outcomes. Given the fact that there are a lot of smartphones models with many customization options, we require our modeling can run on the target device in an online manner, for the sake of accuracy. The last one is widespread use. So far only part of battery interfaces are capable of sensing electrical current. In order to make our modeling suitable to all kinds of smartphones, we do not take advantage of this useful electrical current information in the procedure of modeling.

The energy waste of carelessly designed applications also intensifies this tension. Therefore, energy is a crucial resource to be managed on smartphones. In order to manage well, it is required to have good online modeling of consumes power on major smartphone hardware component like CPU and screen. Existing solutions however either cannot achieve satisfied accuracy and overhead, or need current sensing support on smartphones while a large portion of them is not capable doing that. we propose one online self-constructive energy modeling at component granularity only based upon voltage information from battery interfaces. One novel metric, V-edge, is first introduced to construct a new energy measurement system, replacing current required by the traditional way. New modeling procedures are described to estimate battery capacity and generate energy models for each major component using this new measurement system. Our proposed V-edge solution estimates power consumption 900 times faster than previous SOD based method. V-edge also outperforms SOD method in terms of accuracy for voltage-sensing-

only smartphones. Additionally, we develop kernel module Pecan for per-process energy accounting on the real smartphone device as the example to demonstrate the application and performance of V-edge approach.

The contribution of our work are as follows.

- We are the first to introduce a new metric to build an alternative energy system of traditional one. Only voltage information is required for our approach, so it is able to be applied on smartphones that are only capable of voltage sensing, as well as other embedded devices that are also manufactured like that, which are quit common.

- We provide energy transferring mechanisms between new and corresponding traditional energy systems, and whole procedures of component level modeling with concrete examples on real hardware. The overhead of energy estimation on one system operation is significantly reduced to 1 second [1] from above 900 seconds in SOD method. The modeling accuracy is improved as well, comparable to current-sensing-capable based methods.

- We implement a per-process energy accounting on Android platform, Pecan, for demonstrating the auto and self-constructive procedures from modeling to monitoring and overall performance.

- We are the first to take screen pixel colors, which have a heavy impact on power consumption, into account when modeling screen component.

1.2 Improve Energy Consumption of Email Sync

Standby time, a battery's life in standby mode, is important for smartphones to provide good power performance to users. Even though many smartphones claim

[1] Could be further reduced if voltage updating frequency is less than 1 second.

a standby time of more than 10 days in their technical specifications, in practice their standby times are often much shorter, e.g., only two or three days or even less. The main reason for such a big gap is that when a smartphone remains in standby mode, it stays connected to the Internet through its cellular data interface (e.g., 3G), waiting for various incoming events, such as emails, short messages, instant messages, notifications of social applications (e.g., Twitter and Facebook), and many other push notifications.

We study the power performance of email sync in connected standby on smartphones. Email is a killer application on smartphones for most users, who expect their email clients to keep synchronized with one or more email servers even when the phone is in standby mode. We measure the energy consumption of existing email clients on two major smartphone platforms: Android and Windows Phone (WP). Our results show that email sync is indeed a major drain on existing platforms, and we observe that existing mobile email clients do not handle incoming emails in an energy-efficient way.

Based on our findings, we formulate new design principles for energy-efficient event handling (and specifically email sync) on smartphones in connected standby. Applying these principles to the case of email sync, we develop 5 new techniques, each one addressing one of the shortcomings we have identified in existing systems. Collectively, these techniques achieve significant reductions in the energy cost of email syncing. In summary, the main contributions of this work are:

- We show that email sync in a connected standby state is a major source of energy consumption in today's smartphones, and that existing email clients are not optimized for operation in this mode.
- We derive design principles for energy-efficient event handling in connected standby, and propose techniques for reducing the energy required per email sync.

- We implement a new email client on smartphones and show that it is significantly more energy efficient.

1.3 AP Association in Wireless LAN

802.11 is the main communication method on mobile devices like laptops. Therefore, the throughput of this wireless link has a major influence on the laptop user experience. Among all operations in the 802.11 protocol, AP association procedure is one of few steps determining the throughput. As the first step of the communication procedure, an unwise selection of AP hurts one laptop or client's throughput. This performance downgrade is usually hard to be offset by other methods like efficient rate adaptations. Which AP should one laptop pick for association? This question is not easy to answer in modern wireless networks where there are many laptops and APs, because the broadcast characteristic of wireless communication cause interference within the networks. The throughput of one laptop is affected by association decisions of other network clients that interfere with it. To improve throughput in wireless networks, we study this AP selection problem in a decentralized manner with the objective of maximizing the minimal throughput among all clients. We reveal through theoretical analysis that the selfish strategy, which commonly applies in decentralized systems, cannot effectively achieve this objective. Accordingly, we propose an online AP association strategy that not only achieves a minimal throughput (among all clients) that is provably close to the optimum, but also works effectively in practice with a reasonable computation and transmission overheads on laptops. The association protocol applying this strategy is implemented on the commercial hardware and compatible with legacy APs without any modification. We demonstrate its feasibility and performance through real experiments and intensive simulations.

Our main contributions are:

1. We have designed a distributed online algorithm for AP association. This algorithm well captures interference in transmission. It only needs to be performed once on a new user when she or he joins the network, and meanwhile all existed users do not have to revoke their association decision already made. We have proved our algorithm is $e \log m$ competitive, while no online algorithm is able to do better than $\lceil \log(m+1) \rceil$ [19].

2. Our implementation is practical and does not require any modification on APs, making our technique applicable to existing wireless networks. A light-weight method is introduced to estimate one laptop's throughput on the target AP without association, reducing the operation overhead. We demonstrate its practicability in real experiments, as well as in large scale simulation settings.

1.4 Secure Wireless Communication in IMDs

Medical device technique has seen a rapid development over the past several years. Therapy utilizing IMDs has become much easier with the help of wireless communication. By leveraging external wireless programmer, doctors can wirelessly link to IMD to collect medical information and adjust treatments on devices in routine follow-ups. In the near future, patients even do not need to make a special trip to the hospitals for check up because doctors will be able to check patients' health conditions over wireless LAN. Therefore, both parties prefer wireless IMDs to other types, simulating wireless applications in IMDs. However, the current wireless protocols implemented on IMDs are not designed with the consideration of security. Recent study found out that, with simple programmer equipment obtained from off-the-shelf markets, anyone including malicious attackers could connect and reprogram the commercial IMDs that are widely used right now. This security hazard

put patients' lives in danger, not to mention potential leaking of private information.

We designed a security scheme to protect wireless communications between the IMD and legitimate external programmer. This scheme includes changes of system architecture and communication protocols specifically for heart-related IMDs. A wearable device called IMD Guardian is introduced to act like a security proxy between the IMD and programmer. This architecture change obviously benefits the energy consumption of battery-powered IMDs because they does not have to deal with many communication messages other than medical data. More importantly, this change is able to save patient's life in certain emergency case that requires IMD to remain operable when appropriate security credentials may be unavailable. Accordingly wireless communication protocols also need to be changed accordingly to meet new requirements and security standards.

Two novel techniques are incorporated to protect IMDs from potential attacks targeting at these design changes. One is an Electrocardiography (ECG) based secure key establishment without prior shared secrets. This technique ensure IMD to securely and successfully pair with correct Guardian even under circumstances adversary may present. The other is an access control mechanism resilient to adversary spoofing attacks. When applied, adversary cannot make IMD disable security functionality by making fake emergency conditions.

Our IMD security scheme makes the following contributions:

1. Previous work in ECG based key agreements did not properly extract the randomness of input data or correctly evaluate final outputs. In contrast, we are the first to propose a rigorously information-theoretic secure extraction scheme, and evaluate its performance on resource constrained embedded systems.

2. To the best of our knowledge, we are the first to finalize and implement a comprehensive secure protocol for the previously proposed architecture that

uses an external device as the authentication proxy to protect the IMD. Besides, our design is tailored for IMDs and requires no special hardware. For example, the key extraction scheme of IMD Guard is proposed based on the existing functionality of the IMD, and the wearable device does not need powerful transmitter modules to defend against the adversary's spoofing attacks.

3. We perform extensive experiments on our prototype to evaluate the validity and performance of the IMDGuard.

1.5 Organization

This proposal is organized as follows. In Chapter 2 we introduce a fast self-constructive power modeling technique that works on most of smartphones. Then we improve the email sync in the connected standby mode on smartphones in Chapter 3. In Chapter 4, we present an online AP association protocol for 802.11 that improve the network throughput. In Chapter 5, we present a security mechanism to protect wireless communications between the IMD and legitimate programmer with the consideration of patients' safety. We conclude this dissertation in Chapter 6.

2 Smartphone Power Modeling

Energy consumption is a paramount concern in all battery-powered mobile devices, including smartphones. Power modeling is a key technology and an effective way to understand the power consumption of applications running on mobile devices. Thus, it has attracted much research effort. With a power model, users can identify the power-hungry applications and better manage battery life of their smartphone [2]. Developers are able to profile, and consequently optimize, the energy consumption of their mobile applications [64].

Existing approaches for power modeling have several limitations. First, an accurate power model heavily depends on individual smartphone's hardware/software configuration, battery age, and device usage [35]. Most existing work [23, 28, 38, 69, 78, 81] relies on external power measurement equipment to generate accurate models. This is labor-intensive and requires experts' knowledge. Since the power model of individual smartphone is different and slowly changing [4, 35], it is expensive to apply this approach to build models tailored to every phone. The ``self-metering" approach [35, 39, 45] has been proposed to build individualized power models if a smartphone can read the online voltage and current values from its built-in battery interface. While most smartphones have voltage-sensing capabilities, many smartphones today, including popular models like Nexus S and some Samsung Galaxy series, do not have the ability to sense current. Therefore, the previous approach based on current sensing is not applicable to many smart-

phones. The State-of-Discharge (SOD) approach [94] sidesteps this problem by using the SOD information in battery interface. It does not require current-sensing but has very long model generation time (days) due to the very slow changing nature of SOD. This makes it impossible to have fast power model construction, which is often required to rebuild power models to adapt to various changes in hardware and software, the battery aging, and usage pattern changes (Section 2.2).

In this paper we propose a new approach for power modeling, called V-edge, to address the limitations of existing approaches. V-edge is self-constructive, does not require current-sensing, and most importantly, is fast in model building. V-edge is based on the following insight to voltage dynamics on battery-powered devices: when the discharge current of a battery is changed, the instant voltage change, caused by the internal resistance, has a reliable linear relationship with the current change. Therefore, from the voltage change, we can determine the change of current and consequently the power information (see more details in Section 2.3). The V-edge power modeling requires only voltage-sensing and thus works for most smartphones, and is able to generate power models much faster than SOD-based approaches.

We have designed and implemented a power modeling prototype based on V-edge. Our implementation supports both component-level power models and per-application energy accounting. Experimental evaluation results, using various benchmarks and real applications, show that V-edge is able to generate accurate power models, comparable to the power-meter-based approach. The building time is much shorter than SOD-based approaches.

To the best of our knowledge, V-edge is the first work to model smartphone power consumption by leveraging the regularity of instant battery voltage dynamics. Prior to our exploration, these instant dynamics are treated as irregular fluctuations during slow supply voltage dropping (i.e. SOD decreasing) [94]. Our key

contributions are as follows.

- We are the first to observe that the current information can be inferred from instantaneous changes in a battery's voltage. We demonstrate that inferring current in such a manner is fast, reliable, and accurate.
- Based on this observation, we propose V-edge to facilitate the self-constructive power modeling on most smartphones. V-edge is much faster than the existing solution, making it efficient to (re)build power models for timely adapting of hardware and software configurations with minimum interruption to users.
- We present the design and implementation of the power modeling system that applies V-edge on popular smartphones, including power models of major hardware components and the per-application energy accounting.
- We evaluate our V-edge-based implementation using a diverse set of benchmarks and applications. The results demonstrate that, given the same model, the error range of the energy estimations of V-edge is within 4%, on average, compared with those of power-meter-based approaches. The model generation is two orders of magnitude faster than SOD-based approaches.

The rest of the chapter is organized as follows. In Section 2.1, we introduce how power modeling works as background. In Section 2.2, we survey the related work and motivate V-edge. In Section 2.3, we describe our observation on battery voltage dynamics and demonstrate how to infer current information from voltage readings of battery interface. We present the V-edge energy measurement system in Section 2.4 and the power models in Section 2.5. We describe the design and implementation of a system built upon the V-edge power modeling in Section 2.6 and evaluation results in Section 2.7. We discuss limitations of V-edge and future work in Section 2.8 and conclude in Section 2.9.

2.1 Background: Power Modeling

A power model estimates the power consumption of a system, such as a smartphone, based on more readily observable system statuses. Typically, the model is generated through a *training phase*. A set of well-designed programs are run to explore various system states in this phase and corresponding power values are measured at the same time. Provided system states and their power measurements as inputs, various modeling techniques like Linear Regression (LR) can derive the relationship between these two sets of information, i.e., a power model.

It is common to simply take resource utilization as the system status, such as screen brightness, CPU usage and so on. As an example of such a *utilization-based* power model, consider a system consisting of only a CPU. To build a power model for this system, one would first design several training programs generating different CPU loads. Then one would run each training program and exploit some measurement tool, like Monsoon Power Monitor [12], to provide corresponding power value P.

Assuming that power consumption of the CPU has a linear relationship with CPU utilization, a power model can be formulated as $P_{cpu} = a * U_{cpu} + b$, where a and b are constant, U_{cpu} is CPU utilization, and P_{cpu} is the estimated power consumption. Here, U_{cpu} is called a *predicator*, as it is used to indicate the power consumption of the CPU. There can be multiple predicators in a power model. For example, if Dynamic Frequency Scaling (DFS) is enabled on the CPU, one may use two predicators, the frequency F_{cpu} and U_{cpu}, to estimate the power consumption. Besides LR, other techniques (e.g., non-linear regression) can also be used to build alternative (often more complicated) power models.

Once a power model is generated, it can be used in a *power estimation phase* to predict the power consumption of the system without requiring additional power

measurements. For example, if the CPU utilization of a program is 25% for a duration of T, the the energy consumption of this program is $E_{total} = (a * 25 + b) * T$. More generally, one can monitor and calculate the total energy consumption of a program with dynamic CPU usages as $E_{total} = \sum_i P^i_{cpu} * \Delta T$, where P^i_{cpu} is the i-th measurement of CPU utilization and ΔT is the time interval of the measurement.

Similar to CPU, a power model can be built for other hardware components, such as the screen, Wi-Fi, GPS and so on. After power models of all the components are generated, a power model of the whole system (e.g., a smartphone) can be built on top of the component power models. It is also possible to perform the energy consumption accounting of individual applications or processes, as we will describe in Section 2.5.

While a power model can give absolute values of energy cost, in practice relative values are often more meaningful to end users. It is usually hard for most users to map absolute energy values (e.g., 10 Joules) to what they concern, such as what percentage of energy has been consumed by screen or an application. As a result, most power monitoring tools on smartphones show power consumption information to users in terms of percentages rather than absolute values [2].

From the above example, we can see that power measurement is the foundation and an essential part of power modeling. As we will see in Section 2.2, however, the ways in which power measurement is currently done introduces limits to the power modeling's usability and applicability.

2.2 Related Work and Motivation

System power modeling has been an active research topic and many approaches have been proposed. Based on how power consumption is measured, existing literature can be divided into two categories: *external metering* and *self-metering*.

Table 2.1: Comparison of power modeling approaches

	Self-modeling?	Support most phones?	Fast model adaptation?
External metering	×	√	×
Self-metering except SOD	√	×	√
SOD approach	√	√	×
Ideal approach	√	√	√

Once power consumption is measured, various *training techniques* to generate models have been studied.

External metering. Most existing work on smartphone power modeling relies on external and expensive power meter to build power models [23, 28, 38, 69, 81]. Those approaches are very accurate because a dedicated power meter can precisely measure power consumption. However, they are labor-intensive and can be done only in a lab. Due to hardware and software diversity of smartphone, each type of smartphones may have a different power model. Any new configuration requires rebuilding the model back in the lab again. Therefore, these in-lab methods are very inflexible and thus not suitable to use in the wild across a large number of users.

Recently, BattOr [78] extended the external meter to mobile settings with a lightweight design. Nevertheless, it is not easy for a layman to operate BattOr because it is not deployed on smartphones. In fact, more and more smartphones use non-replaceable batteries to optimize layout, so attaching any external equipment on them becomes difficult and even dangerous to end-users.

Self-metering. Self-metering approaches [35,39,45,94] collect energy information from smartphones' built-in battery interfaces to generate power models without requiring a power meter. The battery interface consists of battery status registers that the fuel gauge integrated circuit exposes to smartphone operating systems, including voltage, temperature, State-Of-Discharge (SOD), and sometimes current information. The power can be calculated if both voltage and current are provided

by the battery interface.

However, many smartphones, including popular ones like the Nexus S and the Samsung Galaxy S2, provide battery interfaces that are only capable of sensing voltages. This means existing self-metering approaches, except [94], are unable to work on a large amount of smartphones (the number is still increasing) already in use.

Zhang et al. [94] proposed building power models based on SOD readings of batteries, which does not require current-sensing. However, the SOD-based approach has a very long model generation time and is inaccurate due to its SOD-based nature. The approach measures the remaining battery capacity (a number from 0% to 100%) to estimate the energy consumption. The granularity of energy measurement is as coarse as 1% of the whole battery capacity. It not only takes tens of minutes to observe a change of battery capacity but also introduces large errors due to the coarse energy granularity.

Motivation of fast power model construction. Fast power model construction is desirable because there are many cases requiring model rebuilding. Besides hardware and software changes, rebuilding is also necessary for changes of software configurations as a simple CPU policy modification may lead to up to 25% differences in power estimation [35]. The battery aging problem [4] also affects power modeling as battery capacity drops significantly with battery age. Thus, a power model needs to be rebuilt after a battery has been used for some time. Furthermore, Dong et al. [35] showed that power models also depend on device usage and demonstrated that a power model should be continuously refined based on usage. In addition, the complexity of modern hardware may require many training cases to generate accurate power models. For example, Mittal et al. [64] used 4096 training cases (for different R, G, B color combinations) to generate a power model for AMOLED display. If it were to take 15 minutes to observe a change of SOD

(the minimal time used by Zhang et al. [94]), then it would take the SOD-based approach more than 1,000 hours to generate a single display model, making it almost impossible for end-users to build or rebuild power models on their smartphones.

In addition, the power measuring of a training program need to be performed in a controlled environment. In fast power modeling, short measuring time largely reduces the chance that the user takes the system control back during the running of a training program. Thus, the fast power modeling is more robust because of the tolerance of users' interruptions. Also, the fast one is more flexible because it is able to quickly suspend construction after the completion of a training program and resume later.

Ideally, besides accuracy, a good power model approach should be self-modeling (i.e., it should not depend on external power meters), work for most smartphones (i.e., it should not require current-sensing), and be able to generate models quickly. As shown in Table 4.2, no existing approach can meet all three requirements. This motivates us to look for a better power modeling approach.

Training techniques for power model construction. Besides LR, other training techniques can also be used for power model construction. For example, Dong et al. [35] used Principal Component Analysis (PCA) to improve the accuracy of a power model by identifying the most effective predicators. Pathak et al. [69] proposed to construct power models using system call tracing. They created Finite State Machines (FSM) for power states of system calls, thus achieving fine-grained power modeling. Our work is complementary to those advanced (and more complicated) model construction techniques. They can be used on top of our battery voltage dynamics based power measurement approach. In this paper, we show that accurate power models can be generated using our new power measurement approach even though we only use basic training techniques for model construction.

2.3 Sensing Current from Battery Voltage Dynamics

A smartphone is powered by the battery, where supplied voltage is not constant. The voltage dynamics of the battery are exploited here to achieve all desired objectives of the ideal power modeling approach. We show that it is possible to infer discharging current information from instantaneous voltage dynamics of a battery. This inference is reliable enough to be used for power estimation. We also demonstrate that it is practical to detect instantaneous voltage changes by using battery interfaces on smartphones. Based on this, a new energy measurement system is introduced in the next section.

2.3.1 Battery Voltage Dynamics

The left part of Figure 2.1 shows the equivalent model of battery electrical circuit [51]. It indicates that at a certain point in time, the voltage reading V of the battery interface can be obtained using

$$V = OCV - V_c - R_b * I$$

where OCV is the open-circuit voltage determined mainly by the remaining capacity of battery, V_c is the voltage drop on the capacitance, R_b is one of the two internal resistors, and I is the discharge current.

When encountering a notable amount of current change, OCV and V_c remain roughly the same value in a short time frame, but the multiplication of R_b and I is sensitive to this current change. As illustrated in the right part of Figure 2.1, we can observe a sharp edge of voltage readings from the battery interface immediately after the current change. This is known as *internal resistance effect*. After the instantaneous change, the voltage then slowly decreases due to the current

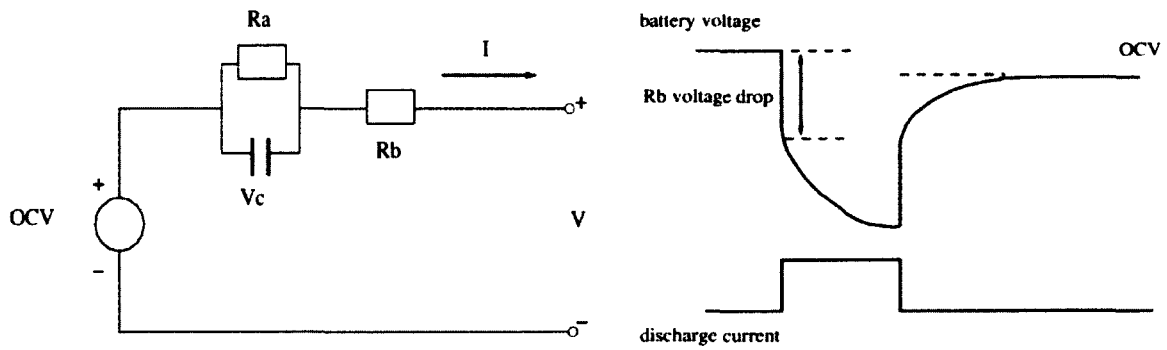

Figure 2.1: Battery voltage dynamics. Left: Equivalent electrical circuit model for batteries. Right: Battery voltage curve when discharge current is changed. The deep drop of voltage is caused by current increasing on resistor R_b.

discharging on the battery. We define this instantaneous voltage change, $R_b * \Delta I$, as *V-edge*, which is in volts. Clearly, the value of V-edge has a linearly proportional relationship with the change of current. If we measure the V-edge values with the same baseline current I_0 (this can be achieved by starting all the training programs from the same baseline when generating a power model), V-edge has a one-to-one mapping with the current.

$$V_{edge} = R_b * \Delta I = R_b * I - R_b * I_0$$

Or,

$$I = \frac{1}{R_b} * V_{edge} + I_0$$

Via this relationship, we can quickly determine the current value given the V-edge. Next we show that this linear relationship is reliable (Section 2.3.2) and V-edge can be detected accurately (Section 2.3.3). Thus, we can use V-edge to further estimate the power consumption and construct power models (Section 2.4).

Battery	Slope α	Intercept β	R^2
1	0.0048	103.4	0.9987
2	0.0054	100.9	0.9945
3	0.0050	103.3	0.9992
4	0.0054	101.8	0.9991
5	0.0054	102.3	0.9985
6	0.0057	158.9	0.9978
7	0.0056	154.5	0.9979
8	0.0051	157.0	0.9976

Table 2.2: Linear mapping between V-edge and current on eight batteries of two different smartphones, in the form of current $I = \alpha * V_{edge} + \beta$. R^2 is the metric indicating the goodness of fitting.

2.3.2 Reliable Relationship between V-edge and Current

The linear relationship between V-edge and current is evident in theory, but because it requires a simplifying assumption about the battery, we seek to understand whether the relationship holds in practice. To this end, we design a set of test trials that run various tasks with different stable workloads on the smartphone. Five batteries for a Google Nexus S phone and three for a Samsung Galaxy Nexus phone were picked for experiments with consideration of different aging stages and manufactures [1]. We ran all tests on these batteries and measured their V-edge values (in μV) respectively. The corresponding current levels (in mA) of these tests were obtained on a Monsoon Power Monitor at a constant voltage level. We then modeled the relationship between V-edge and current using LR for each battery.

Table 2.2 shows the regression results of eight batteries. The first five batteries are for the Nexus S and the last three are for the Galaxy Nexus. R^2 is the *Coefficient of Determination*, a widely used measure of how well the LR is [66]. We can see that R^2 values of these eight fittings are all above 0.99, indicating very good fitting results. More concretely, Figure 2.2 shows how well the regression fits the data of battery 2, of which the R^2 value is smallest.

[1] new to one-year-old batteries from four manufactures

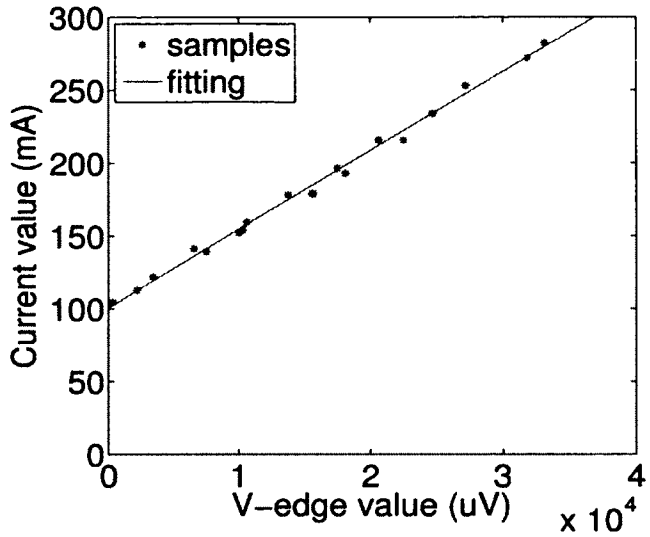

Figure 2.2: Sampling vs. fitting on battery 2.

Those real-world experimental results demonstrate that the relationship between V-edge and current is indeed reliable. This provides the foundation of our proposed fast and accurate power modeling approach.

2.3.3 Detecting V-edge

We show in this subsection that V-edge can be easily and accurately captured by battery interfaces on smartphones. Figure 2.3 illustrates the curve of voltage readings from the battery interface of a Nexus S, when CPU utilization was increased from idle to 95%. We can see a clear voltage drop immediately after CPU utilization (thus the current) was increased. After the instantaneous drop, the voltage decreases very slowly, even with the high discharging current of 95% CPU utilization (the slope will be even gentle if the current draining is smaller). By sampling voltage values from the battery interface before and after the instantaneous voltage change, we can calculate the value of V-edge. Depending on how soon we sample the voltage value after the instantaneous voltage drop, the calculation leads to

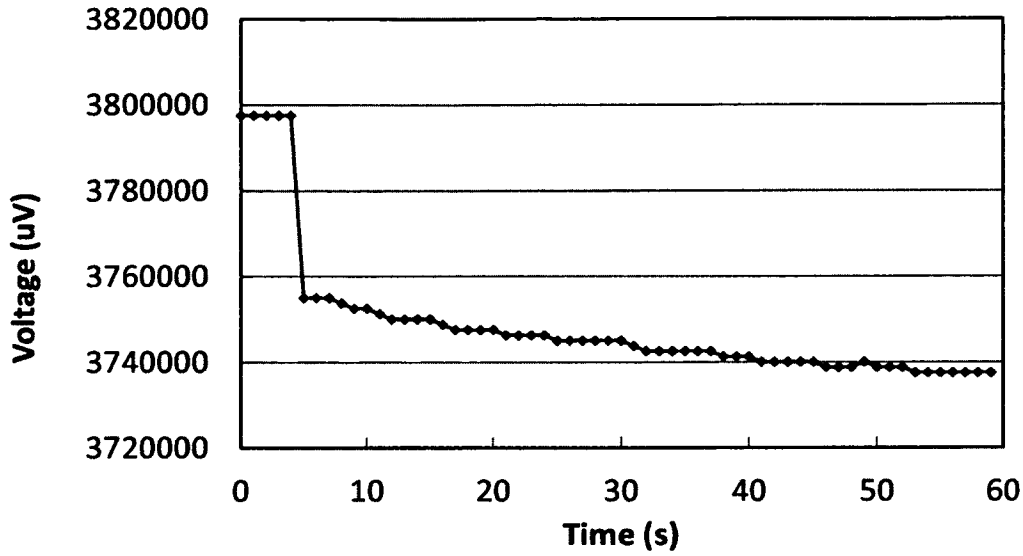

Figure 2.3: Voltage curve on a Nexus S smartphone when CPU utilization is increased from idle to 95%. Sampling rate is 1Hz.

Table 2.3: Sampling error of V-edge in different sampling delays

Sampling delay (s)	1-3	4	5	6	7	8	9	10
Sampling error (%)	0%	2.94%	5.88%	5.88%	8.82%	11.76%	11.76%	11.76%

certain error. Table 2.3 shows the errors when the sampling happens at different times (i.e., sampling delay) after the instantaneous voltage drop.

We can see that the error is zero if the sample is taken within three seconds of the instantaneous voltage drop. The error increases when the sampling delay becomes larger. If the sampling delay is 10 seconds, the error is 11.76%. Clearly, to reduce error, we should sample voltage value as soon as possible after the instantaneous voltage drop.

The battery interface of smartphones typically updates the voltage value periodically but the updating rate may vary drastically across different phones. For example, the Galaxy S2 updates ten times less frequently than the Nexus S (one update every ten seconds versus one per second). In the case of a low update rate, we should align our voltage sampling with the voltage updating. To achieve

this, we employ the following procedure to detect battery interface parameters - the updating interval and time.

We first put the smartphone into idle for a time period longer than its battery interface updating interval (e.g., tens of seconds), then (at time t_0) we increase CPU utilization to a high level and immediately start sampling voltage values at a rate of 1Hz. Once we detect a voltage value change larger than a threshold (i.e., the instantaneous voltage drop caused by increased CPU utilization) at time t_1, we put the CPU into idle again and continue to sample voltage values at 1Hz. When we detect a voltage value change larger than the threshold again (i.e., the instantaneous voltage increase caused by decreased CPU utilization) at time t_2, we stop sampling. Figure 2.4 describes this procedure. Then we treat $\Delta t = t_2 - t_1$ as the updating interval of the battery interface where t_1 and t_2 are the times when voltage updates are triggered. With the sampling rate of 1Hz, the estimation error is within two seconds. Once we know the updating interval and time of the battery interface, we can align V-edge detection with the voltage updating so that the delay of V-edge detection is limited to two seconds. Thus, we can accurately measure the value of a V-edge.

The value of a V-edge is decided by the corresponding current change. If the current change is very small, it is hard to detect the V-edge. To study how likely we can detect a V-edge, we conducted a set of tests with different current changes. Table 2.4 shows the results. For each current change, we repeated the test 50 times and report the probability that the change was detected. We can see that there is about a 64% chance that the V-edge is detected along with just a 7.5 mA increment of current value. With a 30 mA change we achieve up to 98% and 100% with 37.5 mA. On a Nexus S smartphones, 37.5 mA can be caused by a small change of only 4% CPU utilization. To build a power model, we can easily design training programs with a current change much larger than 37.5mA. We conclude

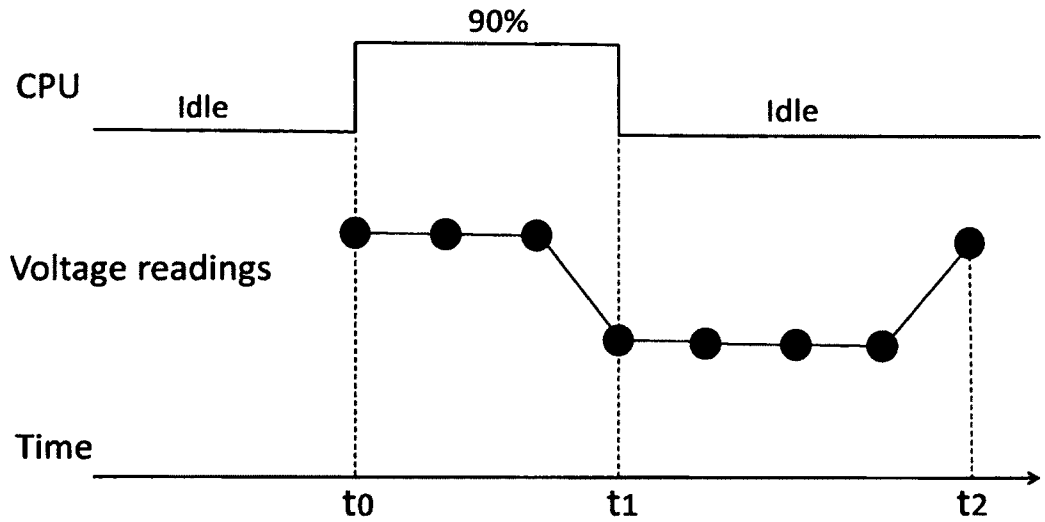

Figure 2.4: Estimate the voltage updating interval and time of battery interface.

Current increment (mA)	Probability (%)
7.5	64%
15	90%
22.5	98%
30	98%
37.5	100%

Table 2.4: Probability of capturing current changes

that V-edge is sensitive enough for component level power modeling.

2.4 V-edge Energy Measurement System

Once we derive the current information from V-edge, it is feasible to calculate the power information and further generate power models on top of it. Thus, in this section, we show how to build an alternative energy measurement system based on V-edge that is equivalent to the traditional energy measurement systems. In traditional energy measurement systems, the energy cost E of a task is measured by power P and time T, $E = P * T$. Power is decided by current I and voltage V, $P = I * V$. For simplicity, we assume that a task has a constant power consumption

during its execution time. The same analysis below can be easily extended to a task with dynamic power consumption by dividing the whole execution time into small time slots with a constant power consumption and using $\sum_i (P^i * T^i)$ to replace $P * T$. That is, the total energy cost $E = \sum_i (P^i * T^i) = \sum_i (I^i * V^i * T^i)$, where i indicates the ith slot.

In our new energy measurement system, we introduce a new term, the V-edge power P_{edge}, to replace the traditional power. The V-edge power is defined as

$$P_{edge} = V_{edge} * V$$

where V_{edge} is the V-edge at the corresponding voltage level V and the unit of P_{edge} is square volts. It does not matter whether the value of V is the voltage value before the instant voltage drop or after the instant voltage drop (see Figure 2.1) because the difference between the two voltage values is fixed as V_{edge}. That is, the two voltage values are interchangeable. In our implementation, we choose the voltage value before the instant voltage drop.

Similarly, we define the V-edge energy as

$$E_{edge} = P_{edge} * T = V_{edge} * V * T$$

to replace the traditional energy.

As we show in Section 2.3.2, V-edge and current have a linear relationship $I = \alpha * V_{edge} + \beta$. Thus, we have

$$\begin{aligned} P &= I * V = (\alpha * V_{edge} + \beta) * V \\ &= \alpha * P_{edge} + P_{edge_0} \end{aligned}$$

where P_{edge_0} is a constant value denoting the baseline V-edge power. That is, we

can calculate the real power consumption of a task from the V-edge power of the task. Similarly, we have

$$\begin{aligned} E &= P * T = (\alpha * P_{edge} + P_{edge_0}) * T \\ &= \alpha * E_{edge} + E_{edge_0} \end{aligned}$$

where E_{edge_0} is the baseline V-edge energy.

During power model generation, we can directly measure P_{edge} but not P_{edge_0}. To calculate P_{edge_0}, we employ the following procedure in the power model training phase. We first design two tasks with constant but different stable workloads. We then run the two tasks to consume the same amount of energy in terms of percentage of battery capacity (e.g., 2% of battery capacity which can be achieved by reading SOD information provided by the battery interface). The V-edge power of the two tasks is P^1_{edge} and P^2_{edge}, their traditional power is P^1 and P^2, and their execution time is T^1 and T^2. Without loss of generality, we assume that T^1 is smaller than T^2. As the tasks consume the same amount of energy, we have

$$\begin{aligned} P^1 * T^1 &= P^2 * T^2 \\ (\alpha * P^1_{edge} + P_{edge_0}) * T^1 &= (\alpha * P^2_{edge} + P_{edge_0}) * T^2 \\ P_{edge_0} &= \alpha * \Theta \end{aligned}$$

where $\Theta = \frac{P^1_{edge} * T^1 - P^2_{edge} * T^2}{T^2 - T^1}$ is a known constant value determined by running the two tasks. Note that here we only need SOD readings to derive the value of baseline power, which is done only once. In SOD-based power modeling approaches, every model training program depends on SOD readings, making the model generation time unacceptably long as shown in Section 2.7.

In fact, even the determination of P_{edge_0} can be skipped if we are only interested

in the energy profile excluding baseline, as is usually the case for end users and application developers. Thus, the V-edge energy system is a linear transformation of the corresponding traditional method.

After knowing P_{edge_0}, in the power estimation phase, when a set of tasks run together (e.g., multiple components or processes), we can obtain energy percentage consumed by each task, even without knowing the value of α. For simplicity, let us assume that there are only two tasks, i and j. We can calculate their power percentage from their V-edge power as follows. The energy consumption of task i is

$$\begin{aligned} E^i &= P^i * T^i = (\alpha * P^i_{edge} + P_{edge_0}) * T^i \\ &= \alpha * (P^i_{edge} + \Theta) * T^i \end{aligned}$$

The calculations of task j are similar to task i, so they are omitted due to space limitations.

Thus, the energy percentage of task i is

$$\%E^i = \frac{E^i}{E^i + E^j} = \frac{(P^i_{edge} + \Theta) * T^i}{P^i_{edge} * T^i + P^j_{edge} * T^j + (T^i + T^j) * \Theta}$$

In addition, we can also estimate how long the remaining battery will last. If $X\%$ of battery has been used by tasks i and j, we have

$$\begin{aligned} X\% * C &= E^i + E^j \\ (100 - X)\% * C &= (P^i + P^j) * T^{ij}_L \end{aligned}$$

where C is the battery capacity and T^{ij}_L is the remaining time of the battery if we continue to run both task i and task j at the same time. By solving the above

equations, we can get

$$T_L^{ij} = \frac{100 - X}{X} * \frac{P_{edge}^i * T^i + P_{edge}^j * T^j + (T^i + T^j) * \Theta}{P_{edge}^i + P_{edge}^j + 2 * \Theta}$$

If $T^i = T^j = T$, we simply have $T_L^{ij} = \frac{100-X}{X} * T$.

And if we run only task i in the future, the remaining battery time will be

$$T_L^i = \frac{100 - X}{X} * (1 + \frac{P_{edge}^j + \Theta}{P_{edge}^i + \Theta}) * T$$

In summary, the V-edge energy system is able to measure and estimate the power consumption of a system. In the following section, we will develop a system based on V-edge that can address common user concerns such as how much energy a particular application consumes, or how long the battery will last if the user continues running one or more applications.

2.5 Power Modeling Based on V-edge

We model the power consumption of four major hardware components of smartphones: CPU, screen, Wi-Fi and GPS. The main purpose is to demonstrate the usability of the V-edge energy measurement system underlying, so we do not introduce new power models. Instead, we use existing or modified ones which are simple and able to capture the main power characteristics of the hardware. In addition, we describe how to do per-application accounting based on generated component-level power models. Power value is provided in the V-edge power $V_{edge} * V$.

CPU Model. DFS is available on most CPUs and often enabled to save power. Thus, for the CPU power model, we consider both CPU frequency and CPU utilization. For each possible CPU frequency, we model the power consumption of the

CPU as a linear function of

$$P_{cpu} = a * U_{cpu} + b$$

where U_{cpu} is the CPU utilization.

Screen Model. The power consumption of a screen is decided not only by the brightness of backlight but also by the pixel colors. For example, at the same backlight level of 255, the power consumption of full-screen white is almost three times that of full-screen red.

Dong et al. [34] used RGB values to create a linear model of OLED (Organic Light-Emitting Diode) type displays' power consumption. However, because this model is not suitable for AMOLED (Active-Matrix OLED) types, Mittal et al. [64] proposed another model to also capture the non-linear properties of AMOLED. However, the full model requires 4096 colors, leading to high training overhead. This neutralizes the advantage of self-metering approaches, timely model adaptation. Therefore, we provide a simplified yet effective alternative using the function

$$P_{screen} = f(L) * (c_r * R + c_g * G + c_b * B)$$

where P_{screen} is the screen power consumption, $f(L)$ is a quadratic function of the brightness level L, (R, G, B) is the average RGB value of all pixels, and c_r, c_g and c_b are the coefficient of R, G, and B.

The goal of this screen model is to reduce the number of colors tested. Based on the above function, we derive a preliminary model from only 216 measured RGB colors ($6 \times 6 \times 6$) by first assuming the linear relationship between the power and RGB color. Obviously, this preliminary model does not work well on AMOLED. Additionally, we measure the power of another 125 samples uniformly distributed in the RGB color space. Then we can obtain the power differences of these 125 colors between measured and modeled values. Because the power of AMOLED

gradually changes among similar colors, the difference of one in these 125 colors can roughly represent the average offset between measured and modeled power values for all colors nearby. Therefore, the final estimated power value of a RGB color is the calculation of the above function plus the difference of one of the 125 color samples that is closest to this estimated color. In this way, the modeling error decreases to a low level, while a lot of training time is saved.

Note that when we train a screen power model, the power consumption of training programs will include the power consumption of the CPU because the CPU cannot be turned off to run any training program. Thus, we need to remove the power consumption of the CPU from the total power consumption of screen training programs. This is done by generating the CPU power model first and applying it in training screen power model.

Wi-Fi Model. We employ a simple model that considers the data throughput of both directions. A linear power function

$$P_{wifi} = d * D + e$$

is selected where D is the application data, incoming and outgoing, through the Wi-Fi interface. Similar to the screen model, we also remove the power consumption of the CPU in training the power model of Wi-Fi.

GPS Model. We model the power consumption of GPS based on the ON/OFF states, following the work [45, 93]:

$$P_{GPS} = f_{GPS} * S$$

where f_{GPS} is the the power coefficient and S is 1 when GPS is enabled or 0 otherwise.

Per-application Accounting. Users often want to know the power consump-

tion of each individual application so that they can identify where the energy was spent. This per-application power accounting can be done on top of the component-level power models. We can monitor the activities of a process on each component (CPU, screen, Wi-Fi and GPS) and account corresponding power consumption as a function of

$$P_{process} = \sum_i P^i_{cpu} + \sum_j P^j_{screen} + \sum_k P^k_{wifi} + \frac{1}{N}\sum_l P^l_{GPS}$$

where i, j, k, l are the ith, jth, kth and lth time when the process uses CPU, screen, Wi-Fi and GPS, respectively. N is the total number of processes using GPS at the same time. Zhang et al. [94] found that the sum of all component estimates is sufficient to estimate the whole system consumption. Thus, we also adopt this assumption. The power consumption of an application is the sum of the power consumption of all its processes.

2.6 System Design and Implementation

We have designed a general V-edge-based architecture, illustrated in Figure 2.5, to run on typical smartphone operating systems. In our design, V-edge runs as a system service in the background, collects data on system resource utilization and activities, generates power models and uses them for power consumption estimation. It also provides a tool with a GUI for users to review the power consumption information of each component and application.

Data Collection. The data collection part is designed to run in the kernel due to two considerations. First, running in the kernel gives us more flexibility and less latency compared with the user space. Second, it avoids the expensive user-kernel mode switching, thus introducing less system overhead. We collect three types of data: voltage readings from the battery interface, utilization information of each

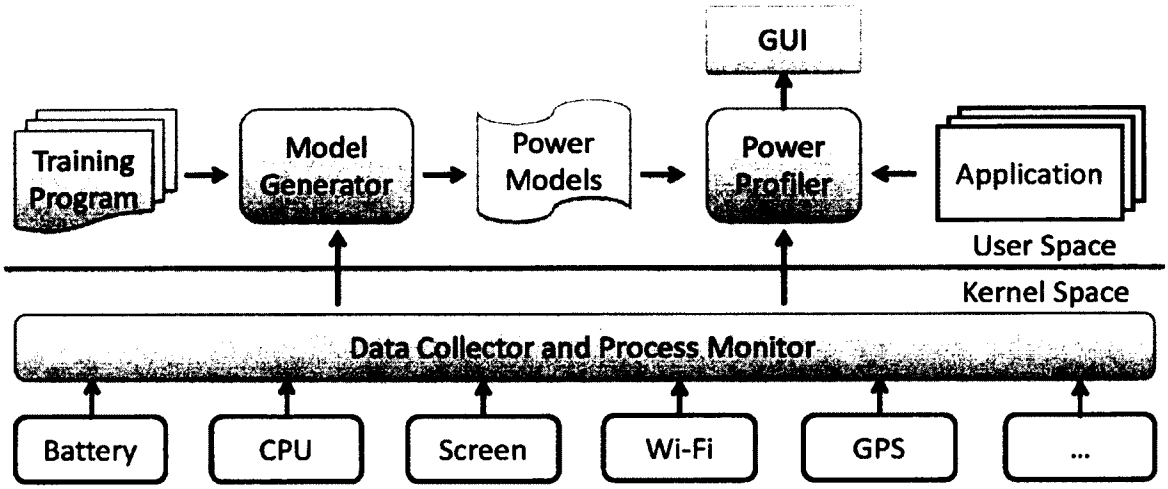

Figure 2.5: System architecture based on V-edge.

hardware component (CPU, screen, Wi-Fi and GPS), and process execution and switching information. For Wi-Fi utilization, we capture the data packets transmitted over Wi-Fi by intercepting the network stack. For process execution and switching information, we hook the kernel scheduler to collect thread scheduling information. We add a new system call to fetch the collected data from the kernel where power model generation and power estimation are done. Voltage information is only used in generating power models, while process information is only used in estimating per-application power consumption. Hardware utilization information is used for both power model generation and power estimation.

Power Model Generation. The system, on top of V-edge, automatically generates component-level power models for CPU, screen, Wi-Fi, and GPS as formulated in Section 2.5. This is done by running a set of training programs for each component in a controlled way. For example, to build the power model of the CPU, we run CPU training programs with other components in their baseline power states. All training programs of a component run from the same initial state to ensure their measured V-edge values are consistent. For example, each CPU

training program starts when the CPU is idle. The model generator also aligns runs of training programs with the voltage updating of the battery interface as we described in Section 2.3.3, to reduce errors of the voltage sampling. The modeling procedure is done without user awareness when the smartphone is idle and not plugged in. If the user suddenly interrupts the generation by using the phone, the procedure can suspend and resume later with little time penalty, thanks to the short estimation time of V-edge. We also allow power model updates adaptively or through a GUI tool described later in this section.

Power Consumption Estimation. Power estimation is done by tracking hardware resource usage and applying generated models. When users use their smartphones as normal, the data collector keeps running in the background to collect the usage information of each component (frequency and utilization percentage for CPU, brightness level and pixel colors for screen, packet size and number for Wi-Fi and usage of GPS). Thus, the power profiler is able to calculate the power consumption of each component. By tracking process switching, we can know which process is using the resources at a given time. Therefore we can associate resources usage and thus power consumption to the corresponding process, for per-process and per-application accounting.

Power Profiling GUI Tool. On top of the power profiler, we design a GUI tool to show the percentage of the energy consumed by each hardware component and provide a rebuilding option to users.

We have implemented the V-edge-based power modeling and monitoring system on the Android platform. Our implementation in total consists of 2k lines of code for the core components (data collection, model generation and power estimation) and 4k+ lines of code for the training programs.

2.7 Evaluation

We evaluate our implementation of the V-edge-based system by answering the following questions. 1) How fast can power models be generated? 2) How accurate is the power estimation using the generated models, both at component-level and in per-application accounting? 3) How much system overhead does the implementation introduce in terms of CPU and memory usage?

2.7.1 Experimental Setup

Devices. *We conduct all experiments on a Nexus S smartphone running Android* 4.0. We use a Monsoon Power Monitor to measure the actual power consumption of the experiments as the ground truth and for comparison.

Training programs for model generation. We develop a total of 412 training programs to generate power models for CPU, screen, Wi-Fi, and GPS. For each CPU frequency (there are five configurable CPU frequencies on a Nexus S), we use eight training programs with CPU usages randomly picked from eleven possible values (idle to full). Similarly, for the screen we use 347 training programs with different brightness levels and RGB colors of different pixel blocks. For the Wi-Fi, we use 24 training programs with different packet sizes and transmission rates. Finally, we use one training program for the GPS module.

Benchmarks. We design a set of benchmarks to evaluate the accuracy of our implementation on component-level power estimation. For the CPU we use four benchmarks running for 60 seconds at a CPU frequency of 200 MHz, 400 MHz, 800 MHz and 1000 MHz. For the screen, we use 15 benchmarks. Each of them displays a different picture, as shown in Figure 2.7, for 10 seconds. For Wi-Fi, we use a benchmark which sends UDP packets with a randomly selected packet size of 50, 100 or 1000 bytes and a random packet inter-arrival time from 1 to 50

milliseconds. The total run time of the Wi-Fi benchmark [2] is 60 seconds. For the GPS, we use a benchmark which uses the location service for 60 seconds.

Applications. We use six real applications to evaluation the accuracy of our implementation on power estimation of real world applications. These applications are *Gallery* where we use the default photo viewer on Android to show 20+ photos (randomly taken by the camera on Nexus S) in slide show mode, *Browser* where we use the default Android browser to read news on Bing News, *Angry Birds* where we play this free version game with commercials, *Video* where we watch a homemade video clip on the default player, *Skype* where we make a VoIP call through Wi-Fi, and *GPS Status* where we run a popular GPS-heavy app from Google Play [7]. Each test performs for one minute.

2.7.2 Model Generation Time

Model generation time is the time period to run all the training programs and construct power models. It is mainly decided by how quickly power consumption can be measured. In the V-edge approach, as shown in Section 2.3.3, we can detect the instant voltage changes and consequently measure power consumption in several seconds. However, in a SOD-based approach, power measurement time is much longer because it measures power consumption by observing changes of SOD, at least 15 minutes [94]. Given our 412 training programs, it takes the V-edge-based system for 1.2 hours in total (including the stabilization time between the training cases which can be further optimized) to generate the power models. However, it would take more than 100 hours for the SOD approach to generate the same power models. Our proposed approach is two orders of magnitude faster than the SOD approach.

More importantly, the long model building time of the SOD-based approach de-

[2]For the experiment purpose, a stable wireless environment is expected in order to remove the influence of outside factors

mands multiple rounds of the battery recharging, thereby requiring the user intervention. As such, the SOD-based approaches are difficult to automate. With the short modeling time, our approach can be easily done without the user involvement. For the sake of optimization, we can further split the whole procedure into small pieces and manage to complete them one by one. Each piece of modeling tasks just takes minutes of the smartphone idle time and consumes little energy. Thus, the V-edge-based system is transparent to end users.

2.7.3 Accuracy

We evaluate the accuracy of the V-edge approach by comparing its energy consumption estimations with both ground-truth measurements and estimations from power-meter-based models. These power-meter-based models are built by measuring power consumption of training programs using an external power meter in the model generation phase. This external-metering approach represents the highest accuracy that one model can achieve because its inputs are precise. Note that the energy comparison is stricter than direct model parameter comparison because model errors can be magnified.

Accuracy of CPU modeling. Figure 2.6 shows the energy consumption of the CPU benchmarks, including the ground truth and the estimated results of the V-edge approach and power-meter-based approach. Compared to ground truth, the errors of the V-edge approach are 1.45%, 7.89%, 9.71% and 4.18% (5.79% on average). The corresponding numbers of the power-meter-based approach are 1.32%, 5.28%, 5.92%, and 1.54% (3.51% on average). The average difference between our approach and the power-meter-based approach is only 3.65%.

The stable relationship between CPU usage and power consumption introduces small errors to both V-edge-based and power-meter-based approaches.

Accuracy of screen modeling. Figure 2.7 shows the results of the screen

benchmarks. Compared to ground truth, the average error of the V-edge approach is 5.77% (max 15.32%, min 1.22%) and the power-meter-based approach is 5.55% (max 15.81%, min 0.11%). The average difference between our approach and power meter approach is only 3.49%. Note that Figure 2.7 shows normalized results. The absolute energy consumption of the pictures are very different, as large as 3.3 times.

Our screen model is one of the most sophisticated smartphone screen models considered in self-metering approaches. Nonetheless, our experiments show that it is of limited accuracy (relatively wide error range). The reason is that this model relies on a small number of reference colors to correct initial estimations and provides final answers. Therefore, if a photo has an average pixel color similar to one reference, its estimation error is low. Otherwise, it is a bit high. The model could be optimized, but it is out of the scope of this paper.

Accuracy of Wi-Fi modeling. Figure 2.8 shows the results of the Wi-Fi benchmark. Compared to ground truth, the error of the V-edge approach is 14% and the power-meter-based approach is 10.65%. The difference between two modeling approaches is only 3.75%.

The error of Wi-Fi benchmark is relatively large. This is because our model is simple and served as the comparison platform of two modeling approaches. More predicators like packet numbers per second may improve the accuracy of modeling. Additionally, there are more CPU activities involved in both building and using the Wi-Fi model, compared with other components. Thus, some error is contributed from the CPU model.

Accuracy of GPS modeling. Figure 2.8 also shows the results of the GPS benchmark. Compared to the ground truth, the error of the V-edge approach is 10.6% and the power-meter-based approach is 4.1%. The difference between our approach and power meter approach is 6.5%.

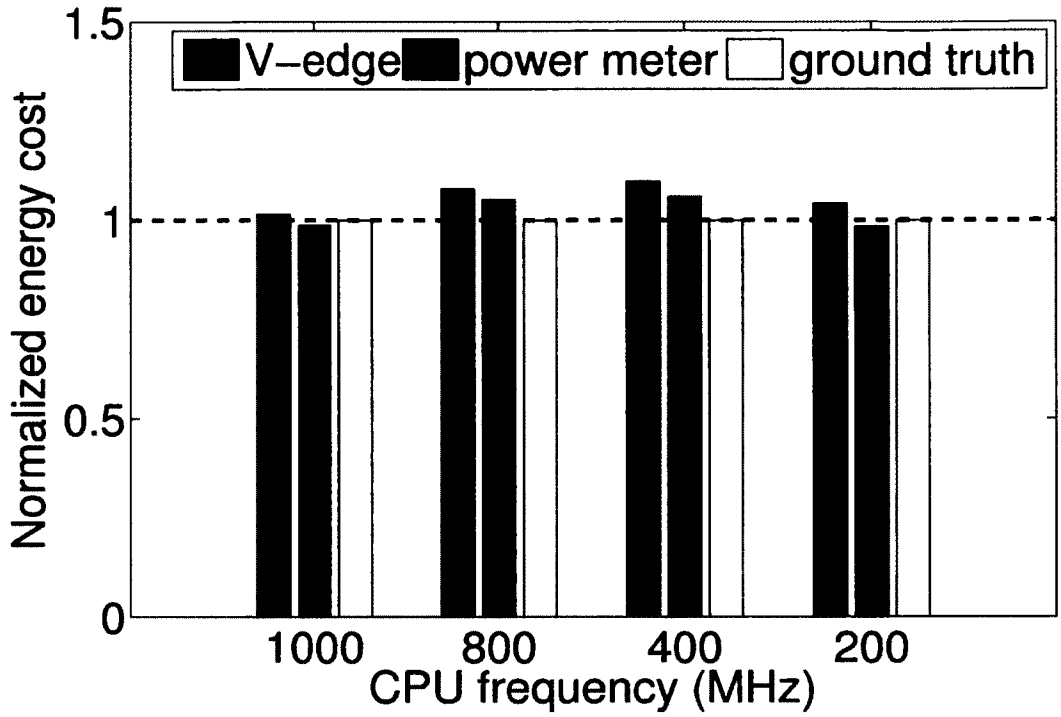

Figure 2.6: Energy consumption of CPU benchmarks. Results are normalized relative to ground truth values.

Accuracy of real applications. Figure 2.9 shows the results of the six real applications. Compared to the ground truth, the errors of the V-edge approach are 19.5%, 8.6%, 0.2%, 1.6%, 14.7% and 15.5% (10% on average). The corresponding numbers of the power-meter-based approach are 15.6%, 12.5%, 4.2%, 2%, 18.3% and 12.2% (10.8% on average). The average difference between our approach and the power-meter-based approach is only 3.8%.

The accuracy of each component model has an impact on application experiments. For example, the Wi-Fi estimation errors are accumulated quickly in the communication-intensive applications like *Skype*, leading to the relatively large difference between modeled and real results. So is the case of *GPS Status* that has a lot of interactions with the GPS module. As to estimation errors of *Galley*, displayed photos are randomly picked, so it is possible that many of them have average colors not similar to any of 125 references. Another reason is that the average RGB

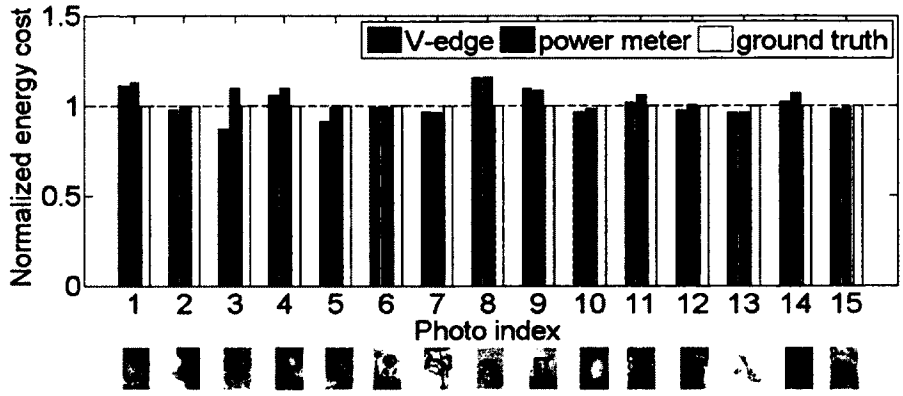

Figure 2.7: Energy consumption of screen benchmarks. Results are normalized relative to ground-truth values.

color over all pixels may not be a good predictor. We will investigate this in future. In addition to the individual model accuracy, errors are also introduced by the assumption that the linear combination of all component energy consumption is equal to the whole system consumption. Besides, we do not include power models for other minor energy consumers such as disk I/O.

Summary. All experimental results show that the accuracy of our approach is very close to the power-meter-based approach. The total average difference is only 3.7% for all component-level and application-level power estimations. This demonstrates V-edge's strength in facilitating the self-constructive power modeling.

2.7.4 System Overhead

Our implementation introduces a very small system overhead in terms of the usage of the CPU and memory. To evaluate, we measured the system CPU and memory usage when V-edge is enabled and disabled for monitoring system energy consumption. With V-edge enabled, the smartphone used only 2 MB more memory to run background V-edge code and store the collect data in memory. Such a small memory footprint is negligible compared with the large memory size of 512MB or

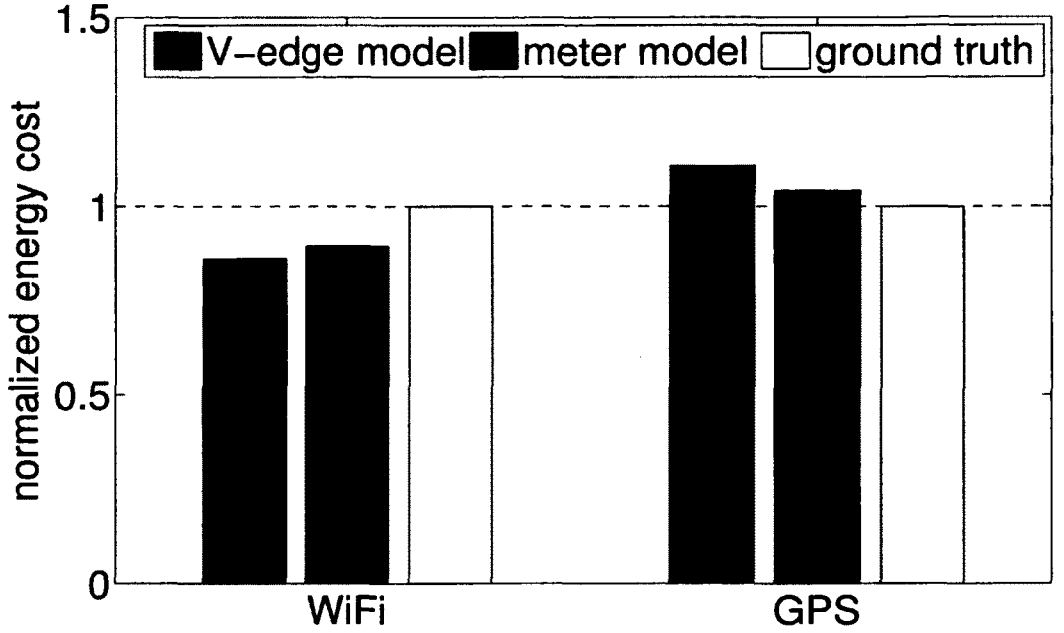

Figure 2.8: Energy consumption of Wi-Fi and GPS benchmarks. Results are normalized relative to ground-truth values.

1GB on today's smartphones. We did not observe any noticeable difference on CPU usage because of its event-driven implementation like the work [69]. Thus, our implementation is lightweight and introduces low system overhead.

2.8 Discussions and Future Work

V-edge provides prior power modeling techniques an opportunity to work on most smartphones on the market. Our power modeling system is one simple example that is built upon V-edge. It is intended to demonstrate the implementation feasibility and exhibit benefits that V-edge offers. Therefore, we only cover major energy consumers, such as the CPU and screen. In the future, we plan to complete our models by adding more components, like a 3G module, in order to create a useful system tool.

Another issue worthy of investigation is the model optimization for self-metering

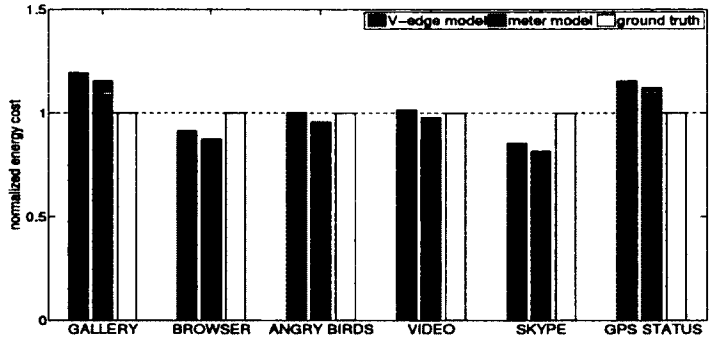

Figure 2.9: Energy consumption of applications. Results are normalized relative to ground-truth values.

approaches. Usually, the more accurate estimations are expected, the more predictors a model need to consider, and thus the more overhead the building procedure has. For example, if we only use the backlight level to model the screen like previous work, 83% building time is saved for our whole system. However, the accuracy is not acceptable. We therefore plan to study how to select more efficient predictors to balance this accuracy and overhead trade-off.

In addition, although our implementation is based on Android platform, the V-edge approach is general enough and not limited to only the Android platform. We plan to implement the V-edge-based system on other mainstream smartphone platforms such as Windows Phone.

2.9 Conclusions

In this chapter, we propose a new approach called V-edge for fast and self-constructive power modeling on smartphones. The V-edge approach is novel because it builds power models by leveraging the regular patterns of the voltage dynamics on battery-powered devices. Different from most existing self-modeling approaches, the V-edge-based approach does not require current-sensing of battery interface so that it works for most smartphones on the market. We have designed and implemented

a V-edge-based modeling prototype. It performance demonstrates that V-edge can facilitate fast and accurate power modeling with low overhead.

3 Optimizing Email Snyc on Smartphones

Standby time, a battery's life in standby mode, is important for smartphones to provide good power performance to users. Even though many smartphones claim a standby time of more than 10 days in their technical specifications, in practice their standby times are often much shorter, e.g., only two or three days or even less. The main reason for such a big gap is that when a smartphone remains in standby mode, it stays connected to the Internet through its cellular data interface (e.g., 3G), waiting for various incoming events, such as emails, short messages, instant messages, notifications of social applications (e.g., Twitter and Facebook), and many other push notifications.

This connected standby state, in which the screen is off while the network connectivity stays active, is of fundamental importance for mobile devices. Most users keep their phone in connected standby for a large fraction of time during the day. In this state, the reception of an incoming event (email, instant message...) wakes up the cellular data interface as well as the phone's operating system (OS), to receive the set of packets over the network and process received data. Collectively, these operations consume a significant amount of power. For example, our measurements show that the energy cost of receiving a small email on an Android smartphone is more than 14,000mJ, which is as high as the energy consumed by a

typical smartphone in standby mode for 10 minutes. In other words, the reception of a single email reduces the standby time of a smartphone by 10 minutes. Assuming the phone has a standby time of 10 days without receiving any events, its standby time will be reduced to less than 6 days (a reduction of more than 40%), if it receives 100 emails per day. This is despite the screen being off during the entire duration, and without any user interaction.

In this chapter, we study the power performance of email sync in connected standby on smartphones. Email is a killer application on smartphones for most users, who expect their email clients to keep synchronized with one or more email servers even when the phone is in standby mode. We measure the energy consumption of existing email clients on two major smartphone platforms: Android and Windows Phone (WP). Our results show that email sync is indeed a major drain on existing platforms, and we observe that existing mobile email clients do not handle incoming emails in an energy-efficient way. The reason is that the underlying protocols, and the overall design of today's mobile email clients do not take into account the specific characteristics and needs of operation in connected standby mode: data processing is not sufficiently de-coupled from network communication (thus preventing the 3G interface from quickly going to sleep), memory management and storage input/output (I/O) are un-optimized, and network protocols and interface operations are tailored for operations other than event reception.

Based on our findings, we formulate new design principles for energy-efficient event handling (and specifically email sync) on smartphones in connected standby. Applying these principles to the case of email sync, we develop 5 new techniques, each one addressing one of the shortcomings we have identified in existing systems.

- In current systems, long 3G tail times keep the 3G interface on for longer than is necessary. We advocate fast dormancy and adaptive 3G tail times based

on email arrival patterns, to reduce the energy cost of the 3G tail.

- In current email clients, data transmission and data processing are coupled together, e.g., a final ACK packet is sent to the email server once the received email has been completely processed. Depending on the amount of processing time required to handle an incoming email, this coupling can prevent the 3G interface from going to sleep for a long time. We propose decoupling the data transmission from data processing to improve energy efficiency.

- In some email clients, the amount of data processing and memory/storage I/O required to handle an incoming email depends on the size of the inbox, leading to a particularly high energy cost for receiving emails when there are many of them in the email client's inbox. We mitigate this problem by using a small cache for incoming emails, to enable fast email processing.

- Flash storage operations on smartphones are slow, which leads to long data processing times, thus keeping the OS awake longer. We show that in connected standby, it therefore makes sense as a design principle to perform email reception entirely in-memory.

- Finally, in some clients, a new secure network connection is established for receiving every new incoming email, resulting in repeated TCP and SSL handshakes and a waste of energy. We solve this problem by reusing network connections across the reception of multiple emails.

Collectively, these techniques achieve significant reductions in the energy cost of email syncing. Specifically, we have implemented them by modifying an existing email client on commercial smartphones. Experimental results show that our revised email client is able to significantly improve email sync's energy efficiency, reducing the average energy cost by 49.9% and up to 77.9% if we put the 3G interface into sleep immediately after an email is received.

The chapter is organized as follows. Section 3.1 gives background information on email sync in smartphones. In Section 3.2, we present a measurement study on the email-sync energy cost of existing smartphone email clients, and derive general guidelines to make event reception energy-efficient in connected standby. Section 3.2 describes our novel techniques. We describe our implementation and evaluation results in Sections 3.4 and 3.5, respectively. In Section 3.6, we demonstrate that our techniques can also be used to improve energy efficiency when receiving events in other event-triggered applications. Section 3.7 surveys related work before we conclude in Section 3.8.

3.1 Background on Email Sync

Similar to desktop or laptop PCs, smartphones usually use a client application to sync email from a server. Typically email sync can be done in one of two ways: polling or pushing. In polling, a client proactively connects to a server to check for new emails. The client can be configured to automatically poll the server periodically (e.g., every 10 minutes), or the user can manually start a polling operation. Polling has two disadvantages. First, it wastes energy if a server does not have any new emails, particularly if polling is too frequent. Second, because a new email may arrive before a client polls its server, polling causes a delay in receiving the email, in the worst case as long as the polling time interval. As a result, users may find it hard to set a proper polling time interval to balance the energy cost and email receiving delay. For these reasons, push email is widely used on smartphones. Most popular email services, such as Microsoft Exchange, Gmail and Hotmail, support push email. The Gmail application on Android supports push email, but not poll email.

In push email, instead of polling from a client, emails are received by pushing

from a server. Once a server receives a new email for a client, it pushes the new email to the client through a previously established TCP connection. Consequently, the client is able to immediately receive new emails as soon as they arrive. Push email not only has minimal delay but in some cases also saves energy since a client does not need to poll a server when there is no new email. As smartphones are usually behind a Network Address Translator (NAT) and a firewall, they often use a private IP address rather than a public one. As a result, an email server cannot initialize a TCP connection to a smartphone. Therefore, push email requires a persistent TCP connection between a client and a server so that the server can send new emails to the client over the persistent TCP connection. For example, Microsoft Exchange supports push email by Direct Push [1]. A client first connects to a server using TCP. To receive push emails, the client uses a long-standing HTTP POST (the protocol used by smartphones to talk to a Microsoft Exchange server is called Exchange ActiveSync [10] which is on top of HTTP) request to ask the server to respond within a time period, e.g., 15 minutes. Then the smartphone can go to sleep or standby mode. If the server has new emails within the 15 minutes, it pushes the emails to the client. Otherwise, the server will send a HTTP 200 OK message to the client who then sends another long-standing HTTP POST to the server.

Push email usually works only on cellular networks (e.g., 3G) but not on Wi-Fi. This is because the coverage of cellular networks is pervasive, and the cellular module of a smartphone can work independently from the smartphone's Operating System. When the OS is in sleep mode, the cellular module remains connected and can receive data from the cellular network. After receiving a data packet from an email server (or any other servers), the cellular module will generate an interrupt to wake up the OS. The execution of the email client is resumed, and the email can be pushed from the server. In contrast, Wi-Fi networks usually have a small amount

of coverage and roaming between different Wi-Fi networks breaks the persistent connection required by push email. Furthermore, on some smartphones, the Wi-Fi interface is usually turned off when the OS is asleep, and hence the Wi-Fi interface cannot receive any data. Therefore, in this chapter, we focus on studying email sync on cellular networks. Even on a cellular network, the carrier operator may tear down a TCP connection if it is idle for too long (a common timeout threshold is 10 minutes), to release the resources allocated for the TCP connection in the NAT and firewall. Email clients on smartphones handle this issue by sending a keep-alive message to the server before the timeout is reached, e.g., a PING message used in Direct Push [11].

3.2 Profiling Energy Cost of Email Sync

In this section we measure and analyze the power performance of existing email clients on smartphones. We first describe the experimental setup and then report the results and findings.

3.2.1 Experimental Setup

We have studied email sync behaviors on two smartphone platforms: Windows Phone and Android. For the WP platform, we used a Samsung Omnia 7 smartphone running Windows Phone 7.5. We used the built-in email client provided by WP. For the Android platform, we used a Samsung Nexus S smartphone running Android ICS 4.0.4. We also used the built-in email client provided by Android. Both email clients support different email services.

Our study uses two popular email services: Microsoft Exchange and Google's Gmail, examining both push email and poll email. All experiments were conducted in Beijing, China, using the 3G network of China Unicom. Monsoon Power Mon-

itor [12] was utilized to measure the power consumption of smartphones. We focused on the power consumption of the connected standby mode. That is, we measured the energy cost of receiving emails without any user interaction and the screens of the smartphones were off while receiving email. The disruption from other concurrent operations is rare in this situation, so we do not consider it in this work.

3.2.2 Measurement Results

3.2.2.1 Stages of receiving an email

We measured the power traces of receiving an email on the two platforms. Figure 3.1 shows the power trace of receiving a small-push email using the Exchange email service on Android. This email consisted of 2 KB random text content and the total data received was 10 KB including all headers and metadata. We can see there are several major stages in receiving this email. At the beginning, the smartphone was in sleep mode with little power consumption (about 30 mW). When the server pushed the email to the smartphone over the 3G network, the 3G interface and operating system of the smartphone woke up. We call this the Wakeup stage. Then the email client received and processed this email, which is called the Receive stage. The last stage was 3G tail time when the 3G interface stayed in a high power state but no network traffic was occurring. We call this the Tail stage. After the Tail stage, the 3G interface and the operating system went back to sleep. The total time duration of receiving the email is 18.42 seconds.

The 3G tail time is controlled by the cellular module using a timer. When there is no data packet transmission, the timer starts to count down. If the network remains idle after the timer expires (e.g., after 10 seconds), the 3G interface goes to sleep. If there is a data packet that is sent or received, the timer is reset. The 3G tail time

is designed to reduce network latency. From Figure 3.1, we can see that it takes about two seconds for the 3G interface to wake up from sleep mode. During the 3G tail time, the 3G interface can keep active for continuous network traffic, avoiding wakeup latency.

We also measured the power trace in receiving the same email on Android through polling. The overall procedure is very similar to the pushing case and consists of the same three major stages (i.e., Wakeup, Receive and Tail). The difference is that, in the polling case, the operating system first wakes up itself (via a timer based on the email polling interval) and then proactively wakes up the 3G interface for querying the email server instead of being interrupted by the 3G interface passively. The total time duration of receiving the email is 18.54 seconds, similar to the push email scenario.

We further measured the power trace when receiving the same email on WP through pushing. Again it consists of the same three major stages. However, compared to the Android cases, the total time for receiving the email on the WP device is much shorter, taking only 7.24 seconds. One reason is that the WP smartphone we used has a much shorter 3G tail time than the Android smartphone. To confirm this, we conducted experiments to measure the 3G tail time on the two smartphones. We found that the WP smartphone has a 3G tail time of about 5 seconds but the Android smartphone has a 3G tail time of 10 seconds. This shows that the WP smartphone enables fast dormancy (see more details in the following section), a technique to shorten 3G tail time, but the Android smartphone does not.

We also measured power traces of other combinations for receiving email. For example, using Gmail email service and pulling on WP, we observed the same three major stages in all the other power trace curves. We conclude that it is a common pattern of receiving an email. Note that the stage markers in Figure 3.1 are just for illustration purposes, used instead of showing the exact duration of each stage. In

particular the Tail stage and the Receive stage may be overlapped. For example, the 3G tail time may start immediately after all data transmissions are finished but before the data processing is done. In addition, in all the above experiments, when the email was received, the email clients on both Android and WP had already stored 200 other emails. Later in this Section we will see that the energy cost of receiving an email may depend on inbox size, particularly with WP.

3.2.2.2 Energy cost of receiving an email

To study the email receiving performance of different configurations, we measured the energy cost of receiving the same 10 KB email using various combinations of the two platforms (WP and Android), two email services (Microsoft Exchange and Google Gmail) and two email receiving approaches (push and poll), with an inbox of 200 messages. Table 3.1 shows the average results.

Our first observation was that we can see that the energy cost on Android is much higher than the one on WP. Aside from hardware differences, one reason is that the WP smartphone has a shorter 3G tail time than the Android smartphone as aforementioned. The 3G tail time is five seconds shorter on WP, leading to about 2,736 mJ energy saving. In addition, as we will show later, WP has low energy cost of email receiving when the inbox size is small. Second, we could see that push email and poll email have very similar energy costs given the same email service and platform. On Android, the difference between push and poll is only 2% for the Exchange service, while this difference is less than 4% on WP. For Gmail, the difference on WP is 13%. This is reasonable because push email and poll email are fundamentally very similar: they receive the same emails and do the same data processing work. On Android, the default email client does not support push email for Gmail service and thus we do not have the number in Table 3.1.

We also found that the energy costs of two email services are similar on the

same platform, although the Exchange service always consumes less. For example, polling-based Gmail on Android takes 20,750mJ energy, but Exchange service with polling consumes 14,674 mJ. On WP, push-based Gmail service takes 6,235mJ energy but the same service of Exchange requires 5,429 mJ energy. Despite this, different email services might be different in implementation details; they are expected to have similar patterns due to the nature of the email service. In fact, Microsoft Exchange ActiveSync (EAS) protocol has been widely used for the synchronization of emails, contacts, calendar, tasks and notes from a messaging server to a mobile device. Many companies, including Google [6] and Apple [3], have licensed and adopted EAS in their own email services for mail sync on mobile devices.

We also evaluated the Gmail application that is specifically designed for the Gmail service by Google on Android, which only supports push email. The last row of Table 3.1 shows the result. We can see that it requires less energy, compared to the default email client on Android. This suggests that Google does some Gmail specific optimizations, probably on both the client side and the server side. However, we do not know the technical details because the Gmail application is not open source.

Due to the similarity of push email and poll email, and the similarity of Exchange service and Gmail service, in the rest of this section, we focus more on Exchange push email.

3.2.2.3 Energy cost vs. inbox size

We measured the energy cost of receiving the same email in different inbox sizes. Inbox size is measured in terms of the number of emails already received in the inbox on a smartphone. We used the same small email of 10 KB that we used in the aforementioned experiments.

Figure 3.2 shows the results using Exchange push email. Interestingly, we can see that the energy cost of receiving the same email on WP highly depends on the inbox size. When the inbox size is small, the energy cost is small and increases with the inbox size significantly. For example, the energy cost when the inbox has 10,000 emails is more than three times the energy cost when the inbox is as small as 20 emails. This is probably found by updating the metadata of the inbox database. For example, to support "conversation view", the email client needs to group all the emails of a conversation thread together.

Compared to WP, Android has a much flatter energy cost in different inbox sizes. However, when the inbox size is small, the energy cost on Android is much higher than WP. For example, when the inbox size is 20 emails, Android consumes 190% more energy than WP. Even if we exclude the extra energy cost (2,736 mJ) of the long 3G tail time on Android, Android still needs 137% more energy than WP. For large inbox size like 10,000 emails, Android and WP have similar energy cost, with a small difference of 7%.

One may argue that 10,000 emails are too many for smartphones as many users only download a small part of their whole inbox onto their smartphones, e.g., only the recent emails in the last few weeks. However, with an informal survey, we found that some people do download a large number of emails or even all their emails onto their smartphones. The main reason is that they can easily search emails on smartphones, particularly when their smartphones do not have a data connection. Furthermore, email clients on smartphones usually store only email headers and limited text content of email bodies (e.g., up to 5 KB bytes text). By default, attachments or embedded images are excluded. Thus, today's smartphones have enough storage space to store a large number of emails.

3.2.2.4 Energy cost vs. email size

We measured the energy cost of receiving an email with different email sizes. As previously mentioned, smartphone email clients usually download the email header and a limited amount of the email body for a new email. Users cannot specify the amount of data to be downloaded in both built-in email clients on the two platforms. Based on our observation, the email client on WP receives very limited email data, about only 10 KB including all email headers. The email client on Android can receive up-to several hundred KB of data (see more details in Section 3.2.6), more suitable for testing. Consequently, we only measured the energy cost of receiving an email with different sizes on Android.

Figure 3.3 shows the results with an inbox of 200 emails. The email size is defined as the total received data size if the email is completely received, including email content, headers and metadata. We decided the size of an email using Outlook email client on a PC. Outlook email client can download all the email data and show the received data size. We can see that the energy cost increases as the email size gets larger but the difference is small, not proportional to the difference in email sizes. For example, when the email size is changed from 11KB to 107KB, an increase of 873%, the increase in energy cost is only 31%. This is for two reasons. First, the email size mainly contributes to the energy cost of receiving and processing the data, which is only a small part of the total energy consumption. Second, for large emails the Android email client does not receive all the email data. For instance, an email with a size of 500 KB and an email with a size of 1,000 KB have similar amounts of data written to the flash storage (336 KB vs. 340 KB).

3.2.2.5 Network activities in receiving an email

To find out what happened behind the energy cost, we profiled the network activities of the email clients when receiving an email. To do that, we captured the network

data packets received and sent by the email clients. On Android, we used Tcpdump to save all the network packets for offline analysis. For WP, we did not have a tool to capture packets. Thus, instead of using 3G, we conducted experiments on receiving emails over Wi-Fi. To make push email work over Wi-Fi, we kept the smartphone on at all times so that it could receive emails pushed from the email server. Then we used Wireshark on a laptop to capture all network packets of the smartphone transmitted in the air.

Figure 3.4 shows the packets (excluding TCP ACKs) transmitted in receiving the small email sized 10KB on WP pushed from the Exchange server. Basically, the email client first received notification of the new incoming email from the server over the previously established TCP connection. Then it started to fetch the email from the server and processed it. After that, the client sent out a PING message packet to the server to wait for the next incoming new email.

We can see that there is a network-idle time period between the PING packet and the prior burst data transmission. During this time period, the email client is processing the received email. We found that this processing time increases with inbox size. As shown in Table 3.2, with an inbox size of 200 emails, it was 0.5 seconds but increased to 10 seconds when the inbox had 10,000 emails.

Figure 3.5 shows the network packets transmitted on Android when receiving the same push email from the same Exchange server. Compared to WP, the network activities on Android are different as follows. First, instead of using the same TCP connection to fetch a new email, after receiving a notification over the existing TCP connection, the Android email client establishes a new TCP connection to the email server to receive the new email. Doing so introduces some overhead on TCP and SSL handshakes between the client and the server, where 12data packets (6.9KB in total) are transmitted. Second, on Android there are two network-idle time periods. One is between the email sync request sent from the client to the

server and the server's reply. The other one is after receiving the burst event data and before the final PING message sent from the client to the server. However, unlike WP, the network-idle time periods do not depend on inbox size. Table 3.2 also shows the total network-idle time on Android. It bears no obvious relationship to the inbox size, ranging from 3 seconds to 4.6 seconds (3.6 seconds on average).

3.2.2.6 Storage activities in receiving an email

Besides the network activities, we also profiled the storage activities in receiving an email. For Android, we developed a tool to intercept the storage access APIs (e.g., file reading and writing) to capture all the flash-storage operations of the email client. For WP, we couldn't develop such a tool due to the limited programmability offered by Windows Phone. Therefore, we only conducted the flash-storage experiments on Android.

Table 3.3 shows the total amount of data written to flash storage for different incoming email sizes. We can see that the amount of data written to flash storage increases with email size. This is because the email client needs to write the received email data into the inbox database. Thus, more data were written to flash storage for larger email. However, when the email is very large, the data amount written to flash storage does not increase any more. This is because the Android email client limits the maximum data received for emails. Unfortunately it does not provide an option for users to configure such a behavior.

The results show that the Android email client immediately writes received email data onto flash storage. All flash operations in receiving an email were distributed in a time period of one to two seconds, each writing a small amount of data. As we will show in Section 4.3, small flash writes are time and energy expensive and should be avoided in receiving emails.

3.2.2.7 Energy distribution

Finally we studied how the total energy cost of receiving an email is distributed in the three stages of the power curves: Wakeup stage for the 3G interface and operating system to wake up, Receive stage for the email receiving and processing, and Tail stage for the 3G tail time. We denote the time duration of the Wakeup stage as the time period between the time when the smartphone wakes up and the time before receiving the notification packet from the server. We define the time duration of the Tail stage as the time period between the time when the last packet is transmitted and the time when the smartphone goes to sleep again. We treat the rest of time as the time duration of the Receive stage.

Figure 3.6 shows the results. We can see that the Wakeup stage takes stable and fixed energy costs on both WP (907-936 mJ) and Android (979-1,022 mJ) no matter how big the inbox is. The energy cost of the Tail stage is also quite stable on the two platforms. On WP, the Tail stage takes 2,419-3,427 mJ (2,866 mJ on average) of energy but on Android it takes 5,832-6,307 mJ (5,962 mJ on average) of energy, due to the difference of 3G tail time on the two platforms. Different from the Wakeup stage and the Tail stage, the energy cost of Receive stage depends on the inbox size. Particularly for WP, the energy cost increases significantly with the inbox size, from only 144mJwhen the inbox size is 20 to 10,944 mJ when the inbox size is 10,000. For Android, the numbers are relatively flat: 7,387 mJ for inbox of 20 and 11,002 mJ for inbox of 10,000.

In addition, we can see that the Receive stage and Tail stage take more energy than the Wakeup stage. The Receive stage takes 30% - 71% (49% on average) of the total energy cost on WP, and takes 52% - 61% (57% on average) of the total energy cost on Android. For the Tail stage, it takes 22% - 51% (38% on average) of the total energy on WP, and takes 33% - 42% (37% on average) of the total energy cost on Android. Therefore, to reduce the energy cost of receiving emails,

we should focus on optimizing these two stages.

3.2.3 Summary of Findings

Based on above measurement results in above Section, we can see that the existing smartphone email clients are not energy efficient in following aspects.

First, the 3G tail time takes a large part of the total time of receiving an email. The 3G tail time is designed to reduce the network latency for burst data transmissions. However, the inter-arrival time of incoming events is not short in the connected standby mode. Occasional transmissions make the 3G tail time unnecessary in most cases. Therefore, the energy is wasted on the 3G interface.

Second, the data transmission and processing are coupled together, leading to more wasted energy. Receiving an email is composed of the following steps: first is communicating with the server to receive some data, processing the received data locally, and then communicating with the server again. The second step of the communication resets the 3G tail timer, causing the 3G interface on for longer time and thus wasting energy.

Third, the email clients write data onto the flash storage when receiving an email. As we will show in Section 4.3, flash operations are energy-expensive and should be batched together to save energy.

Fourth, when the inbox size is large, receiving an email costs more energy than small inbox case. It is desirable to reduce the energy cost for a large inbox size.

Finally, on Android, the email client always initializes a new TCP connection to the server to receive an email. Doing so wastes energy due to the duplicated TCP and SSL handshakes.

Next in Section 4 we propose techniques to address above energy inefficiency and reduce the energy cost of receiving email. Those techniques can also be used as general design guidelines to improve power performance of other applications

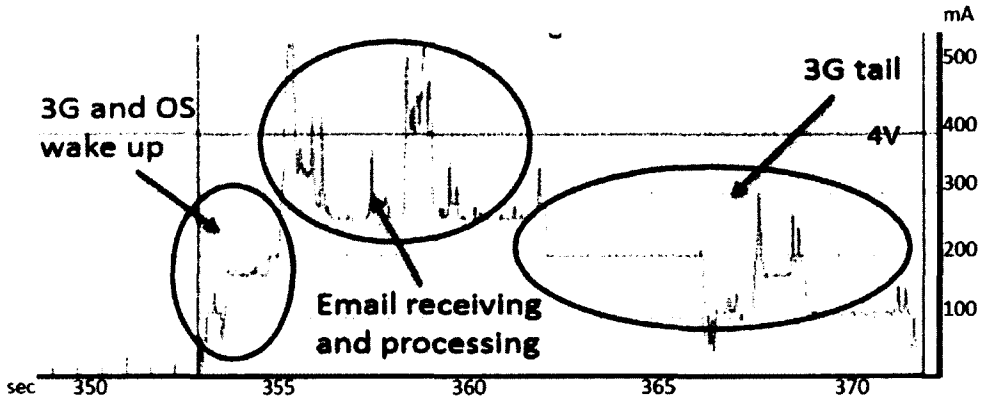

Figure 3.1: Power trace of receiving a push email on Android using Exchange email service

Table 3.1: Average energy cost of receiving a small email with various configurations

Configuration	Energy Cost (mJ)
Android + Exchange + Push	14,976
Android + Exchange + Poll	14,674
Android + Gmail + Push	N/A
Android + Gmail + Poll	20,750
WP + Exchange + Push	5,429
WP + Exchange + Poll	5,659
WP + Gmail + Push	6,235
WP + Gmail + Poll	7,027
Android + Gmail App (Push)	12,931

on smartphones.

3.3 Reducing Energy Cost of Email Sync

From Figure 3.1, we can see that the baseline power, when the system is active but idle (e.g., during the 3G tail time), can reach over 200 mW. This means that once the 3G interface and the OS wake up, the smartphone will consume a large amount of energy even without doing any tasks. Therefore, to reduce the energy cost of receiving an email, one effective way is to shorten the total time period of email receiving as much as possible. In this section we show how we achieve it with five

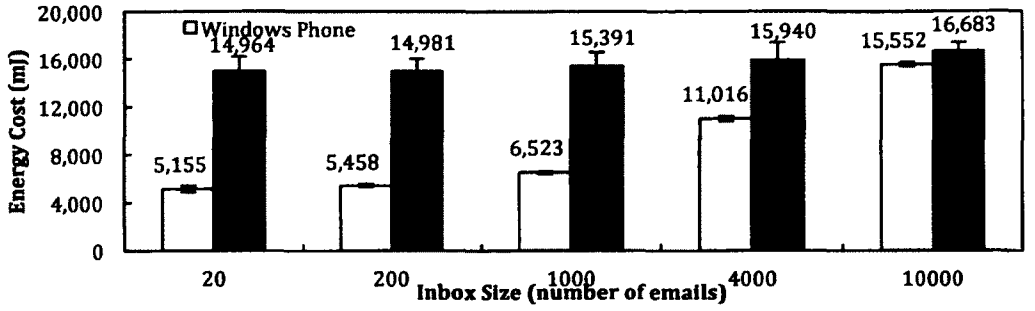

Figure 3.2: Energy cost of receiving a push email of 10 KB using Exchange service with different inbox sizes (five trials per experiment)

Table 3.2: Network idle time during email receiving

Inbox size (number of emails)	Network idle time (seconds)	
	WP	Android
20	0.4	3.5
200	0.5	4.6
1,000	0.9	3.0
4,000	3.5	3.4
10,000	10.0	3.5

techniques, each of them addressing one energy inefficiency issue we observed in Section 3.

3.3.0.1 Reducing 3G Tail Time

The 3G tail time causes a lot of energy to be wasted in receiving emails. After a new email is received and processed, the 3G interface enters the tail state in which the whole smartphone does nothing but consume energy. While the 3G tail is designed to save energy and reduce latency in continuous network data transmissions, it is not suitable for the connected standby mode. In the connected standby mode, the events received by a smartphone are pretty sparse, often arriving at a time period much longer than 3G tail time. Thus, it is likely that the 3G tail time cannot cover multiple events. To verify it, we measured the inter-arrival time of the emails of four researchers at Microsoft Research Asia. In total 31,303 emails, only 1.3% emails come within a ten-second time frame of previous one. Therefore, a 3G tail time

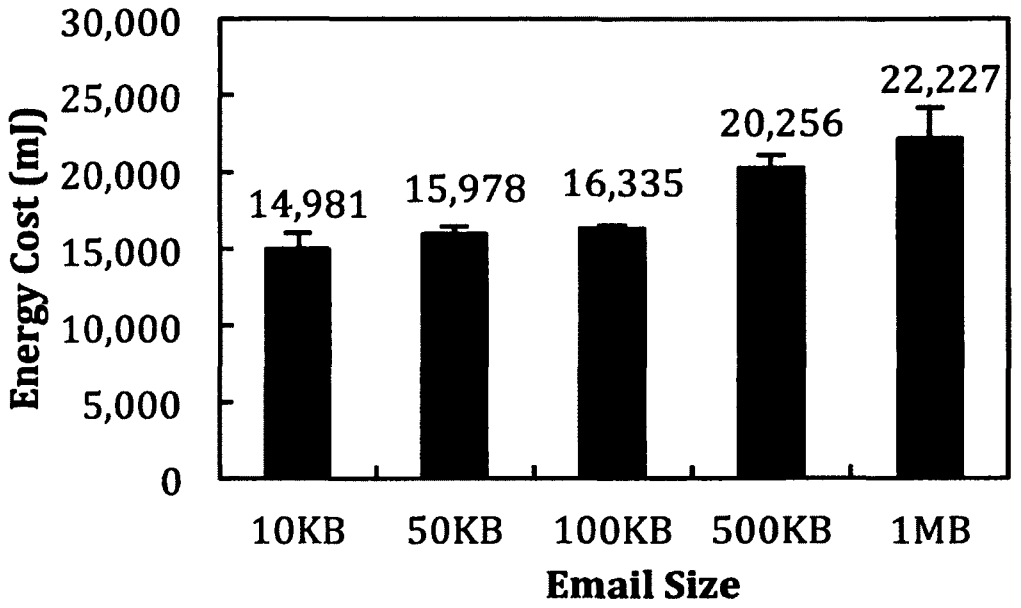

Figure 3.3: Energy cost of receiving an email with various email sizes (five trials per experiment)

Table 3.3: Data size written to flash in email receiving

Email size (KB)	10	50	100	500	1,000
Data size (KB)	139	156	188	336	340

of ten seconds rarely covers more than one incoming emails and thus wastes a considerable amount of energy.

The problem of wasted energy caused by the 3G tail time has already been identified recently. To solve the problem, a technique called fast dormancy [13] has been proposed to force the 3G interface to quickly sleep faster than before, e.g., five seconds rather than ten seconds. However, rapid dormancy increases the signaling overhead of cellular networks if the sleep timers are too short. In fact, in the early days of fast dormancy, some popular smartphones using aggressive timers have led to severe signaling channel congestion [8]. Later the network-controlled fast dormancy was adopted by 3GPP in Release 8 [1] to reduce the signaling overhead caused by the fast dormancy. Researchers have proposed to adaptively use the

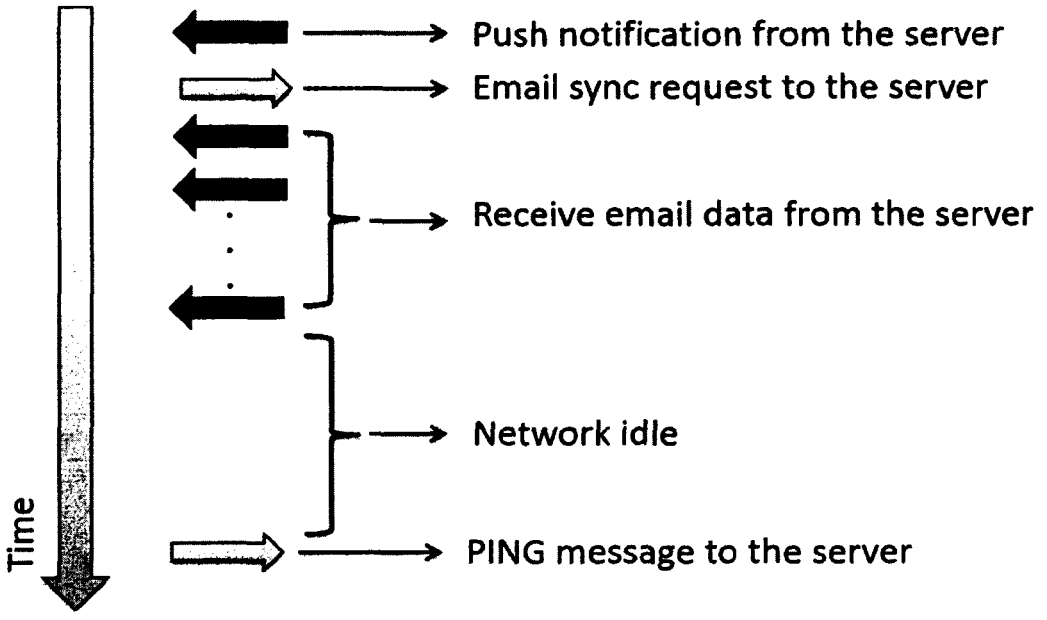

Figure 3.4: Network packets transmitted in receiving a push email using Exchange service on WP

fast dormancy based on application traffic patterns [18, 30, 72, 73]. For example, in [72] the authors proposed a tail optimization protocol based on fast dormancy. In [18] a system has been built to predict the end of communication to invoke the fast dormancy without increasing the network signaling load.

We advocate for the fast dormancy to be used in connected standby. As shown by our measurement results in Section 3, the WP smartphone we used enables fast dormancy and thus has a shorter 3G tail time than the Android smartphone. We also agree with the authors in [72] and [18] for adaptive 3G tail time based on application traffic patterns.

In particular, because that network events are very sparse in the connected standby, 3G tail time still wastes energy even if fast dormancy is enabled. In an ideal case, the 3G interface should go to sleep immediately after the event receiving is finished. However, to avoid interference with other applications, individual applications must not directly control the 3G tail time. Instead, we propose that the

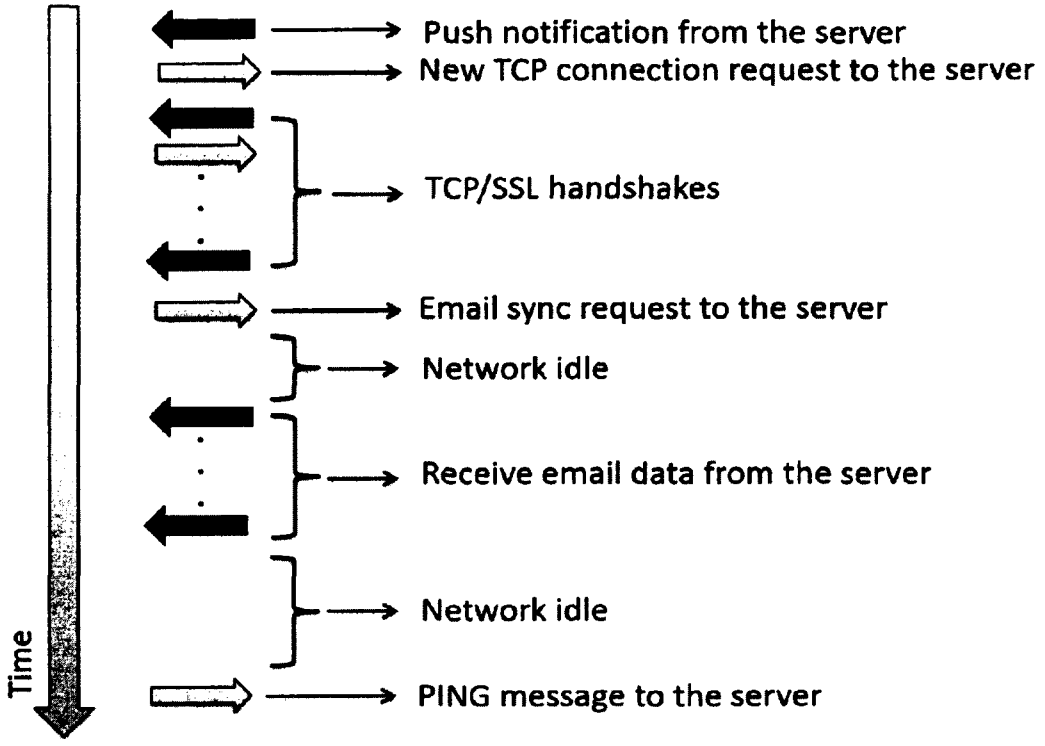

Figure 3.5: Network packets transmitted in receiving a push email using Exchange service (the same one used for Fig. 3.4) on Android

OS should provide a global service that collects the network usage information of all the applications running in connected standby, decides the shortest length of 3G tail time, and collaborates with the cellular network to put the 3G interface into sleep mode as quickly as possible, minimizing the energy waste of 3G tail time. Due to the long inter-arrival time of network events in the connected standby, the signaling overhead is small.

3.3.1 Decoupling Data Transmission from Data Processing

Due to the 3G tail effect, data transmission should be decoupled from data processing to save energy. In an ideal case, an email client should first finish all network communication with a server and then process the received data, so that the 3G interface is able to go to sleep as soon as possible. In receiving a push email,

there are three steps: 1) fetching the email content from the server, 2) processing the received email locally (e.g., updating the inbox database), and 3) telling the server that it is ready to receive the next push email and then go to sleep. As we show in Section 3, in such a "transmission-processing-transmission" process, existing email clients do the second transmission part after the processing part is finished, which seems a natural design choice due to the sequential nature but is not energy efficient. During the data processing part, the 3G interface stays awake unnecessarily and the second data transmission resets the 3G tail timer, wasting energy. For example, assume that the data processing needs two seconds and the power is 200 mW, 400 mJ energy will be wasted. Therefore, to save energy, an email client should batch all its data transmissions together. It should first fetch new email content and immediately tell the server that it is ready to receive the next push email before waiting for the email processing to be finished.

Batching all data transmissions together may be difficult if a network protocol depends on the result of data processing. For example, if an email server requires an email client to tell whether the email processing is successful or not, the email client cannot start the second data transmission part before the data processing part is finished. We propose to solve this problem using speculative execution [50]. That is, we predict the result of the data processing part and send the predicted result to the server without waiting for the data processing part to be finished. If we find that the predicted result is wrong after the data processing part is finished, we can re-sync with the server. If we can correctly predict the data processing results with a high probability, we can still batch data transmissions together to reduce the 3G tail effect. To determine feasibility, we conducted an experiment to measure the failure rate of receiving and processing 10,000 emails. We found that all the 10,000 emails were successfully processed without any error. This indicates that the email receiving is reliable and we can correctly predict the email processing results with a

high probability. Therefore, it is possible to leverage speculative execution to save energy. Furthermore, for Microsoft Exchange and Gmail services on smartphones, they do not depend on the email processing results. Thus, we can easily batch data transmissions and decouple them from data processing.

3.3.2 In-Memory Data Processing

Writing data to flash storage is slower than writing data to memory, particularly for small, random data writes [24, 30, 47]. We conducted experiments to measure the performance difference of the flash storage and memory. We used a Nexus S smartphone running Android 4.0.4. We inserted an email of 270 bytes into a SQLite database for 1,000 times and measured averaged energy and time cost of inserting one email. We used the SQLite database because the Gmail client and the default email client on Android use SQLite to store emails. The SQLite library provides two types of APIs to write data into a database: individual writes and batching multiple writes together as a transaction. Thus, we measured the performance of writing the 1,000 emails one by one and batching the 1,000 email writes together.

Table 3.4 shows the results. For individual writing, when the entire database was loaded in memory, the average energy and time cost per email was just 1 mJ and 1 ms. When the database was stored on the flash storage, the corresponding energy and time costs increased to 32 mJ and 55 ms. For batching writes, when the database was loaded in memory, the average energy and time costs were 0.59 mJ and 0.58 ms. When the database was stored on flash storage, the corresponding figures were 0.64 mJ and 0.65 ms. Those results show that for small writes, flash storage costs much more energy (32 times) and takes much more time (55 times) than memory. However, for writing a large amount of data together, the flash storage and memory have similar performance in terms of the energy and time costs.

Therefore, to reduce the energy and time cost in email receiving, an email client should not issue many small flash writes. Instead, it should batch multiple small writes together and commit them to flash in a batch. As mentioned before, email clients on smartphones only download email headers and up to several kilobytes of email bodies. Thus, the data received for a new email is small. However, as we show in Section 3, existing email clients write received data of every email to the flash storage immediately, which is not energy efficient. Therefore, when an email client receives a new email, it should cache the received data in memory rather than writing them to flash immediately. Once the client has received a sufficient number of emails (e.g., 10 emails) or the cached data are larger than a threshold, it can flush all the data to flash storage together. Furthermore, the client may also piggyback data writes with other activities. For example, when the user turns on the smartphone to check emails or use other applications, the client can write its cached data to flash. As those flash writes share the same baseline power with other activities of the user, they will not introduce much extra energy or time overhead.

Delaying flash writes may cause data loss if the email client or the smartphone crashes before the cached data are written to flash storage. However, modern smartphone operating systems including Windows Phone, Android and iOS all run each application in a separate protection domain (i.e., the so called "application based security model"). Failures of one application will not affect other running applications. As a result, today's smartphones are more reliable and crash less than before. Even when running out of battery, the operating system will terminate applications gracefully, to give them chance to save data onto flash. Smartphone email clients are also pretty reliable. We measured the default email client on Android by receiving 10,000 emails and did not observe any failure or crash. In addition, even if the email client crashes before writing the cached data to flash storage, it can re-sync the emails with the server. Therefore, in-memory data processing is

able to reduce the energy cost of email receiving.

3.3.3 Reusing Existing Network Connections

In receiving emails, email clients should reuse existing network connections rather than making new ones. In particular, for the push email, because an email client already maintains a persistent network connection with a server to receive notification of new incoming emails, it should receive a new email over the continuous network connection to save energy. Even for a poll email, when an email client does a second polling before the network connection of the last polling times out, it should also reuse the previous network connection.

While it may be easy and natural to make a new network connection to receive every new email, the energy cost may not be negligible. As most email services require a secure network connection, making a new network connection includes not only TCP layer handshakes but also SSL layer handshakes to negotiate the encryption method and exchange security keys. As shown in our experiments on Android in Section 3, the overhead of making a secure TCP connection is not negligible. Without transmitting any application data packets, it consumed 1,757 mJ energy to make a TCP/SSL connection. In total 6.9 KB data were transmitted between the client and server. WP is more energy efficient. It always reuses the same TCP connection to receive new emails.

3.3.4 Data Structure Partitioning

One interesting finding on WP is that the energy cost of receiving the same email increases significantly when the inbox size is large. On Android, the energy cost also increases with inbox size but the increase is not as significant as WP. The possible reason is that when storing a new email into the database of an inbox, it takes more time to update the metadata. For example, the email client needs

Table 3.4: Energy and time cost of database writing

	Energy cost per email (mJ)		Time cost per email (ms)	
	DB in mem	DB on flash	DB in mem	DB on flash
Individual writes	1	32	1	55
Batch writes	0.59	0.64	0.58	0.65

to search the whole inbox to associate the new email with the right conversation thread. The cost of doing so depends on the inbox size. In addition, when the inbox is too large to be loaded into the memory of the email client, searching the inbox takes more energy and time due to the frequent flash operations.

To solve this problem and reduce the energy cost of receiving an email, we propose partitioning a large inbox into two parts: one small inbox with recently received emails (e.g., emails received in last two weeks) and one large inbox containing all remaining emails. The email client only uses the small inbox to handling email receiving. That is, when receiving an email, the client only inserts it into the small inbox and updates the metadata without using the large inbox. Thus, the energy cost of receiving emails is reduced even if a smartphone stores many emails. This approach is based on a key observation: most of the time users only need to check recent emails on their smartphones. Therefore, it is not necessary to touch all the emails already stored in a smartphone in receiving new emails. When a user does need to access all the emails, e.g., in searching the whole inbox, the client can search both the small inbox and the large one to return the combined results. As searching a whole inbox happens much more rarely than receiving emails, inbox partitioning is able to save energy.

3.4 Implementation

Implementing the techniques proposed in Section 4 only requires modifications on the email client side. Since the Gmail application on Android and the built-in email

client on WP are not open source, implementation is done on the built-in email client of Android, which is accessible and widely used. The source code we used is vanilla Android 4.0.4 with kernel 3.0.8.

Figure 3.7 illustrates the architecture of our implementation, focusing on email syncing and processing. On Android the built-in email client consists of two parts, each running in a separate process. The first part is the network communication part which implements the email protocol and handles all the communication details with the email server to receive and send emails. For Exchange email service, this part runs as an app named ``Exchange". It runs in the background without a UI. The other part is the data processing which processes received emails and provides user interfaces to handle all the interaction details with users including reading emails and composing emails to send out. It also deals with email storage, saving emails to and loading emails from the SQLite database. This part runs as an app named ``Email". The Email app process and the Exchange app process exchange data through Remote Procedure Calls (RPCs).

To decouple data transmission from data processing, we revised the Email Protocol Handler in the Exchange application to do all data transmissions without waiting for the Email Processor to finish processing a received email. Specifically, PING messages are sent via the PING Engine in Figure 3.7. In the original client, the PING Engine checks up in a two-second interval whether the Email Protocol Handler allows it to resume PING messaging after an email-receiving event ends. So there is a delay between the completion of receiving an email and the start of PING. The energy is wasted because the cellular module has to delay its sleep mode. We improved the signaling between the Email Protocol Hander and the PING Engine so that a PING message can be sent out immediately. Thus, all data transmissions are batched together to save energy. We also revised the Email Processor to report an error if anything is wrong in processing an email so that the Email Protocol

Handler can re-sync with the email server. In addition, we revised the Email Protocol Handler to receive emails by re-using the existing TCP connection rather than making a new one, avoiding energy waste of TCP/SSL handshakes.

In the Email application, we added a new Database Manager, loading a small cache inbox into the memory to implement in-memory email receiving. On the flash storage, all the emails are stored in a big database. The Database Manager manages the data sync between the in-memory cache inbox and the persistent one on the flash storage. When the user interacts with the smartphone or we receive more than a customized number of emails, the Database Manager commits the new email data onto the flash storage.

Android does not provide an API for fast dormancy and we cannot control the 3G tail time on the fly. We have managed to enable fast dormancy on the Nexus S smartphone by flashing a new baseband firmware.

3.5 Evaluation

We evaluate our implementation by measuring the total energy saved in receiving an email and the individual ones contributed by each technique we proposed and implemented. For all the experiments, we used Exchange push email service with the original built-in email client provided by Android and our revised one on a Nexus S smartphone. For each experiment, we repeated for the procedure five times and report the average results with standard deviations.

Total energy saving. Figure 3.8 shows the total energy saving of our revised email client, compared to the original one when the inbox size is 200 emails but the new incoming email has different sizes. On average the total energy saving is 44.3%, with a narrow range from 41.2% to 46.9%. These results demonstrate that our proposed techniques are able to significantly reduce the energy cost of email

receiving. The energy savings mainly come from the shortened email receiving time. On average our revised email client reduces the time period of receiving the emails by 47.6%, compared to the original one.

Figure 3.9 shows the total energy savings of our revised email client in receiving a small email of 10 KB with different inbox sizes. Similar to Figure 8, we can see that our revised email client is able to effectively reduce the total energy cost of email receiving no matter how many emails the inbox has. The average energy saving is 45.8%, also with a small range from 39.9% to 51.3%. Compared to Figure 8, the average energy saving is higher because when the email is large, the received data volume becomes large and more energy will be saved by our proposed techniques, as shown later in this Section.

Energy saving of each technique. Figure 3.10 shows the anatomy of the total energy saving in receiving the 10 KB email with an inbox of 200 emails. For the total energy savings of 7,690 mJ, the largest part comes from reducing 3G tail time which saves 2,448 mJ (32%) energy. Decoupling data transmission from data processing part and reusing TCP connections also save a significant energy cost, each with 1,757 mJ (23%). The in-memory processing part and using small cache inbox part reduce relatively less energy, saving 806 mJ (10%) and 922 mJ (12%)energy, respectively.

Reducing the 3G tail time and reusing the TCP connections have fixed energy savings, independent from incoming email size and inbox size. However, the energy saving of other techniques may depend on the email size or inbox size. Figure 11 shows how much energy can be saved by decoupling data transmission from data processing for different email sizes. We can see that generally the energy saving increases when the email size becomes large. The decoupling technique tries to batch all data transmissions together. When the email is large, more data is transmitted over the network and thus the energy saved by batching becomes

large.

Similarly, Figure 3.12 shows how much energy can be saved by in-memory processing for different email sizes. We can see that the energy saving is high for large emails. This is because the processing cost (e.g., writing operation) heavily depends on email size. Thus, when the email size is large, processing the email entirely with memory operations will save more energy.

Figure 3.13 shows the benefit of using a small cache inbox for email receiving when the total inbox size is different. The size of the small cache inbox we used is 20. Thus, the benefit for the inbox size of 20 in Figure 3.13 is zero. For a larger inbox size, we can see that the extra energy saving caused by using the small cache inbox increase as the total inbox size becomes large. For the total inbox size of 10,000 emails, the extra energy saving is 2,750 mJ. These results demonstrate the effectiveness of using the small cache inbox.

Note that in all the above experiments, we measured the energy cost of receiving individual emails without counting the energy saving of batch writing multiple emails onto flash. In our implementation we batch 20 emails together for writing their data onto flash. By doing so, we can further save 634 mJ energy per email. On average this increases the total energy saving of receiving a 10 KB email to 49.9% in different inbox sizes.

Furthermore, we also calculate how much extra energy can be saved if we can put 3G interface into sleep immediately after the end of receiving an email. Doing so we can further save 2,448 mJ of energy. For receiving a 10 KB email with an inbox size of 200, this will increase the total energy saving by 77.9%.

3.6 Event Receiving in Other Applications

Despite this chapter focusing only on email receiving, the techniques we proposed may also be applied to other applications. Example applications include various social network applications such as Facebook and Twitter, locations based applications which receive notifications upon location changes, and many other applications which receive push notifications when new updates or new in-application content are available. Similar to email, when in standby mode, those applications maintain their own persistent connection or use a separate push notification service like Google client notification service [16] to keep connected to a server for receiving various events or poll changes from a server. Such event receiving is very similar to email receiving and follows the "transmission-processing-transmission" pattern. Each event receiving also wakes up the 3G interface and a smartphone's entire operating system for a small amount of data transmission and processing. Due to the similarity between receiving emails and receiving other events, those applications may also have the issues with energy inefficiency we found in email receiving, such as sparse data transmissions resetting 3G tail timer, frequent small flash writes, and making new network connections unnecessarily. Therefore, our proposed techniques can be used as general design guidelines for those applications rather than specific techniques designed for email receiving only.

We plan to investigate more applications to further study how the techniques we proposed in this chapter may be used to reduce the energy cost of receiving events in connected standby. In addition, recent tablet devices and other types of mobile devices have started to function like smartphones, keeping connected in standby mode to receive various network events including emails. For example, Windows 8 introduces a new connected standby power mode [82] to provide such support. We also plan to study the power performance of applications on those devices and

look for potential improvements.

3.7 Related Work

System power management. Power management is important to mobile devices including smartphones due to the limited power supply of batteries. All smartphone operating systems come with components to leverage hardware capabilities and manage system activities to save power, e.g., adjusting screen backlight levels, running CPU at low frequency, putting network interfaces into sleep or killing power-hungry applications in low battery mode. Various techniques have been proposed to reduce the power consumption of each key component of a smartphone such as the screen [64], CPU [74] and network communication [33,42]. However, even with good system-level and component-level power management, applications that are not well designed can still cause high energy consumption.

Finding energy bugs of applications. Recently researchers have started to study how to improve the power performance of mobile applications by finding energy bugs. Energy bugs are not real program bugs causing failures but they lead to unnecessarily high energy consumption. For example, in [70] the authors proposed techniques for automatically finding non-sleep energy bugs in applications. However, such a tool depends on pre-configured program patterns to find energy problems, e.g., requiring a Wakelock to prevent the system from sleeping without releasing it, and only works for serious non-sleep bugs. Authors in [64] built tools for developers to estimate the energy consumption of their applications and find potential energy inefficiency issues. Our work focuses on finding energy inefficiency when receiving email by measuring the energy performance of existing email clients. The techniques we proposed are complimentary with existing work and can be used as general design guidelines for other applications.

Fast dormancy. The problem of the cellular data network tail has drawn a lot of attention. Smartphone manufacturers have developed their own and inconsistent ways to reduce the energy cost of a cellular tail, which results in the standard approach called network-controlled fast dormancy in 3GPP [13, 72]. Researcher also proposed schemes to reduce the power consumption of applications by using fast dormancy adaptively [18, 72]. We believe that fast dormancy should be used in connected standby. However, even with fast dormancy, there is still wasted energy in the event receiving during the connected standby due to very sparse network activities. We propose that the OS should provide a global service to collect the network usage information of all the applications running in the connected standby. When all the applications finish their event receiving, the service works with the cellular network to immediately put the 3G interface into sleep mode and thus minimizes the energy waste of the 3G tail time in connected standby.

3.8 Conclusion

In this chapter we have studied the power performance of email sync on the Windows Phone and Android operating systems. We conducted experiments to measure the energy cost of email receiving in existing email clients, using both push and poll emails, with different email services, email sizes and inbox sizes. Together with our analysis of the network and flash storage activities, the measured results indicate that existing email clients are not energy-efficient in several aspects. Besides 3G tail time, the main source of inefficiency stems from the coupling of the data transmis-sion and data processing, which is a bad design choice in connected standby mode. Frequent storage access, making new TCP connections to receive new emails and updating a large inbox are further components that increase the time duration of email sync and lead to energy waste.

Based on the experiment, we advocate that the fast dorman-cy should be used in the connected standby and propose other techniques to address the energy in-efficiency of ex-isting email clients, including the decoupling of data trans-mission from data processing, in-memory processing, reus-ing existing network connec-tions and using a small cache inbox to handle the email sync. We have imple-mented these techniques and evaluation results show that average energy sav-ings reach 49.9%. If we can put the 3G interface into sleep mode immediately after receiving an email, the total energy savings can be increased to 77.9%.

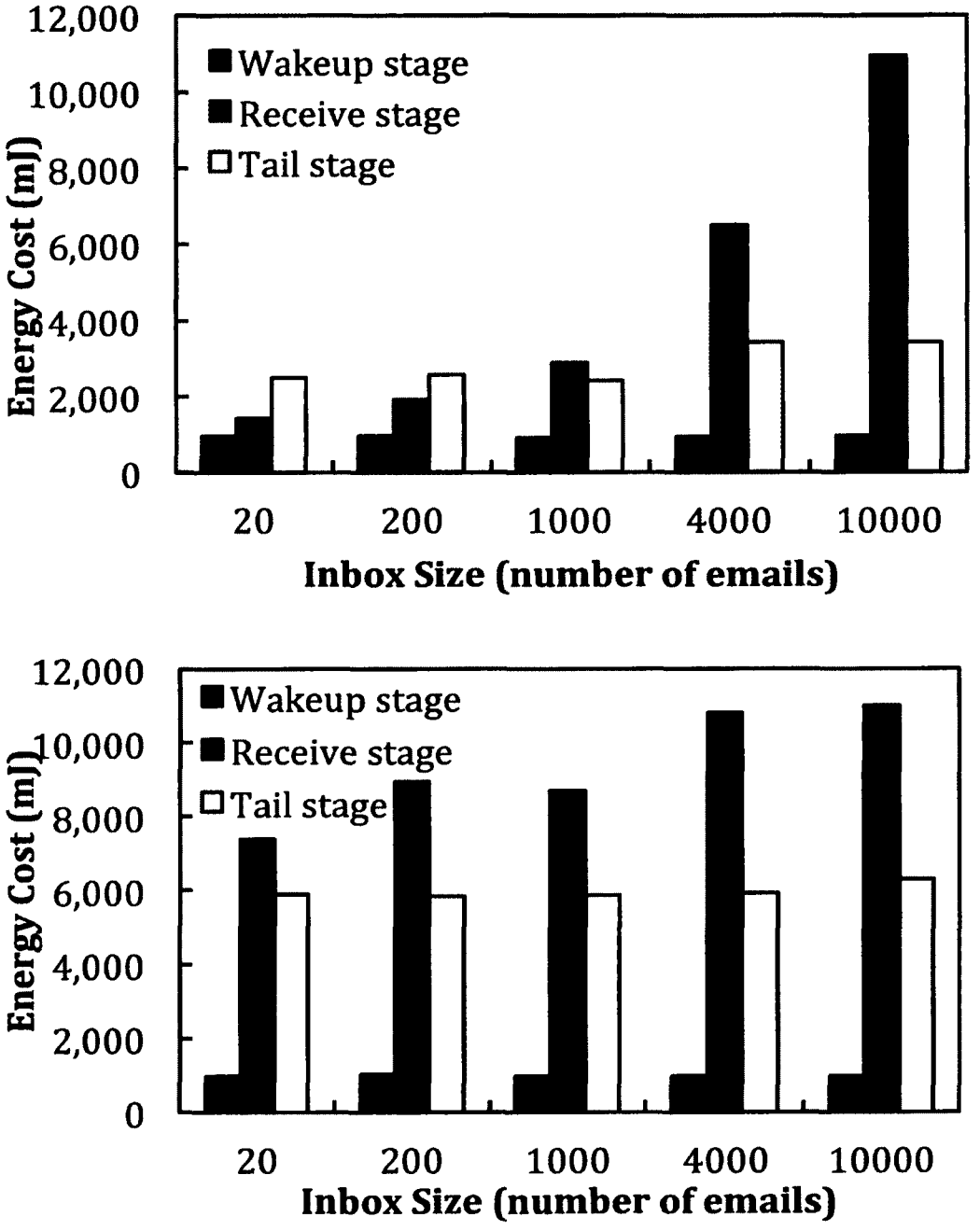

Figure 3.6: Energy distribution among the three stages in receiving an email with different inbox sizes. Top: on WP. Bottom: on Android.

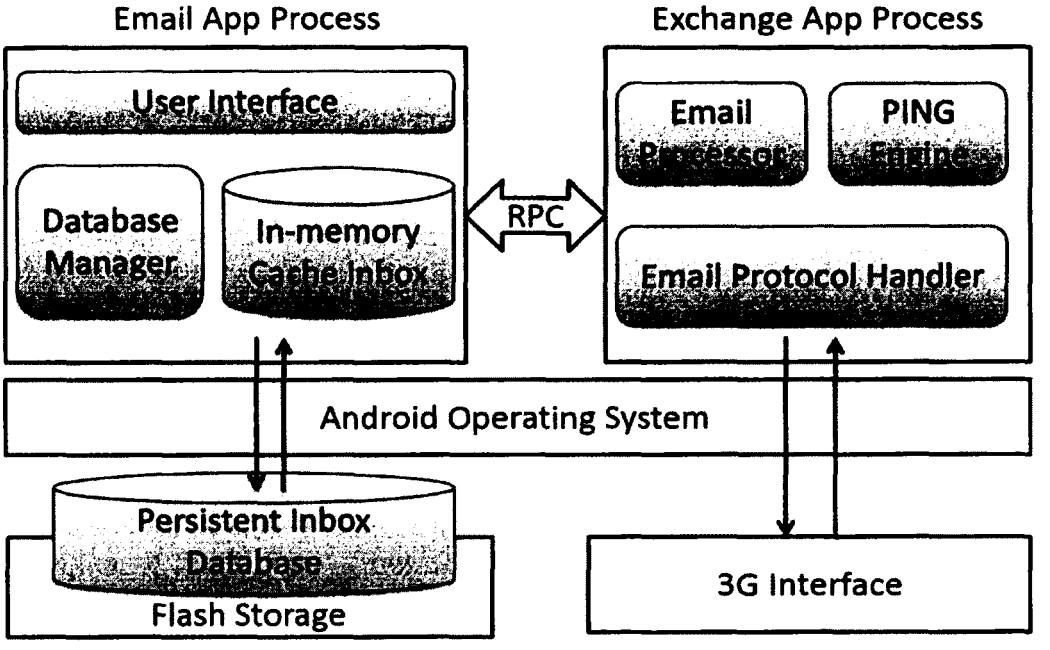

Figure 3.7: Implementation architecture

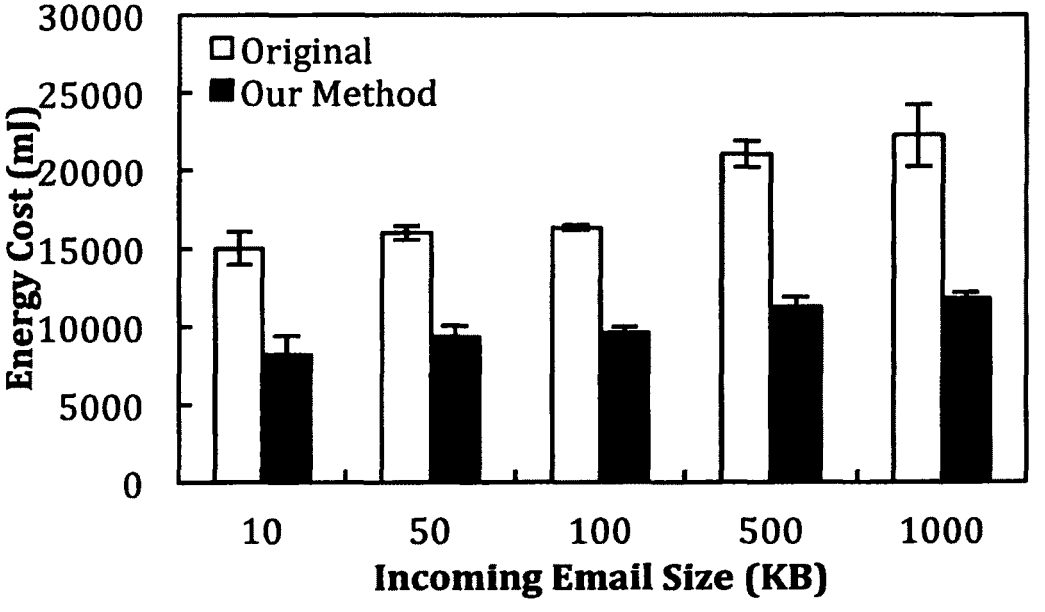

Figure 3.8: Energy saving with different email sizes

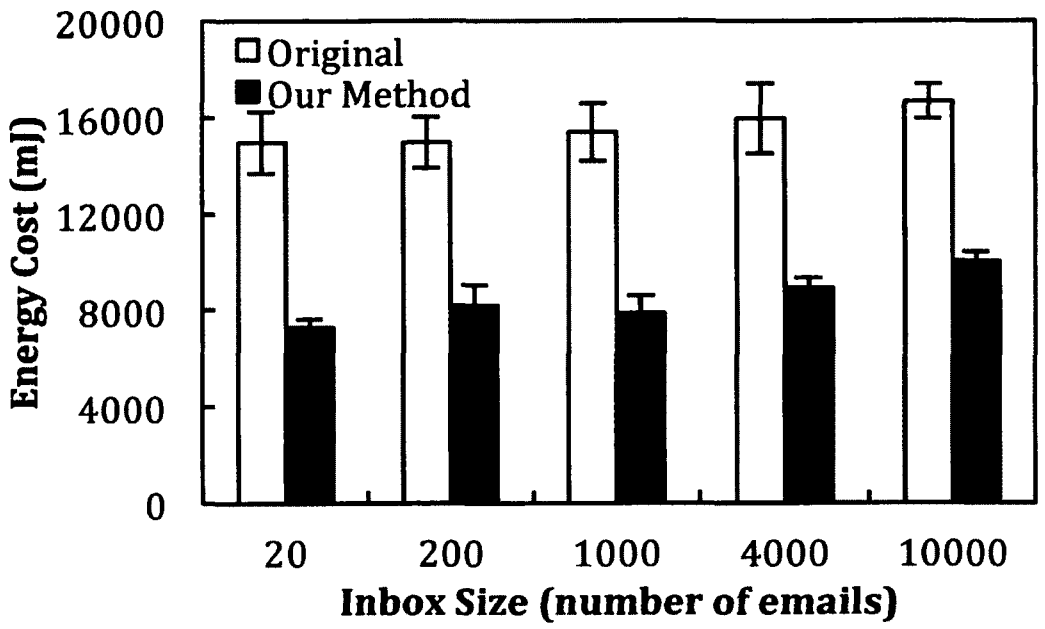

Figure 3.9: Energy saving with different inbox sizes

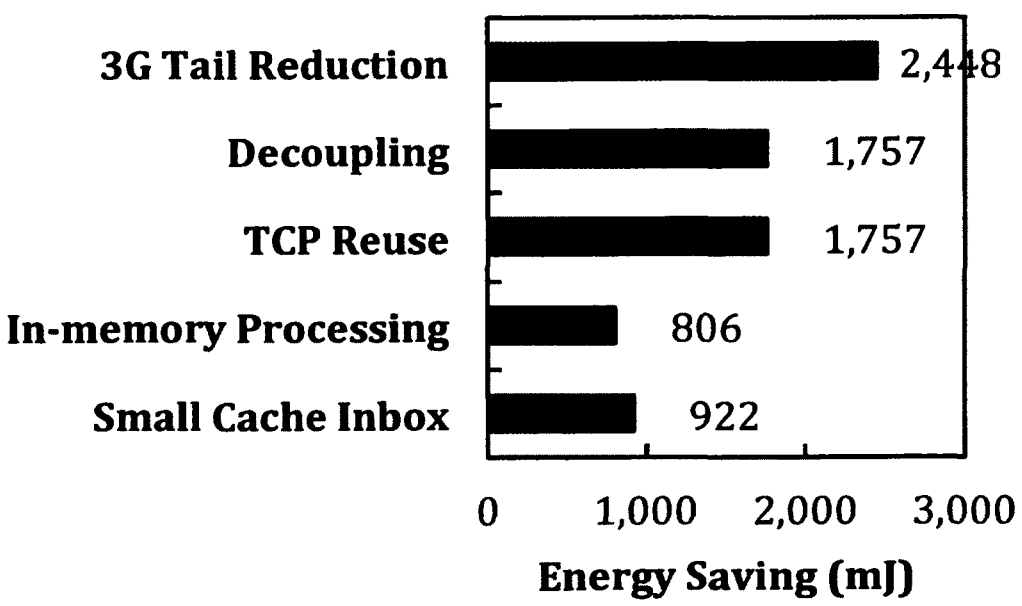

Figure 3.10: Anatomy of total energy saving

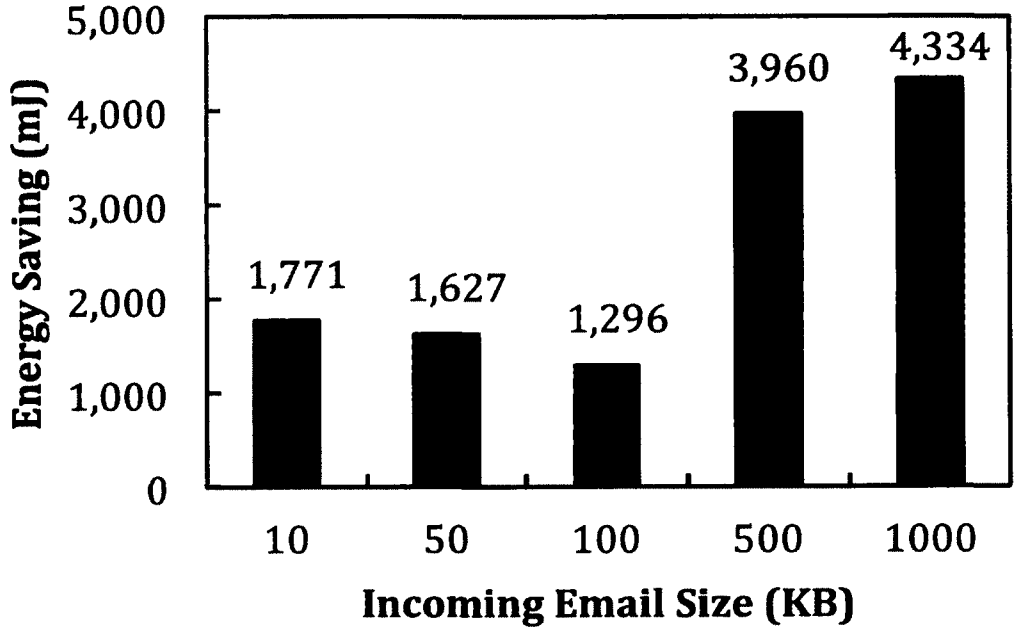

Figure 3.11: Energy saving of decoupling data transmission from data processing in different email sizes

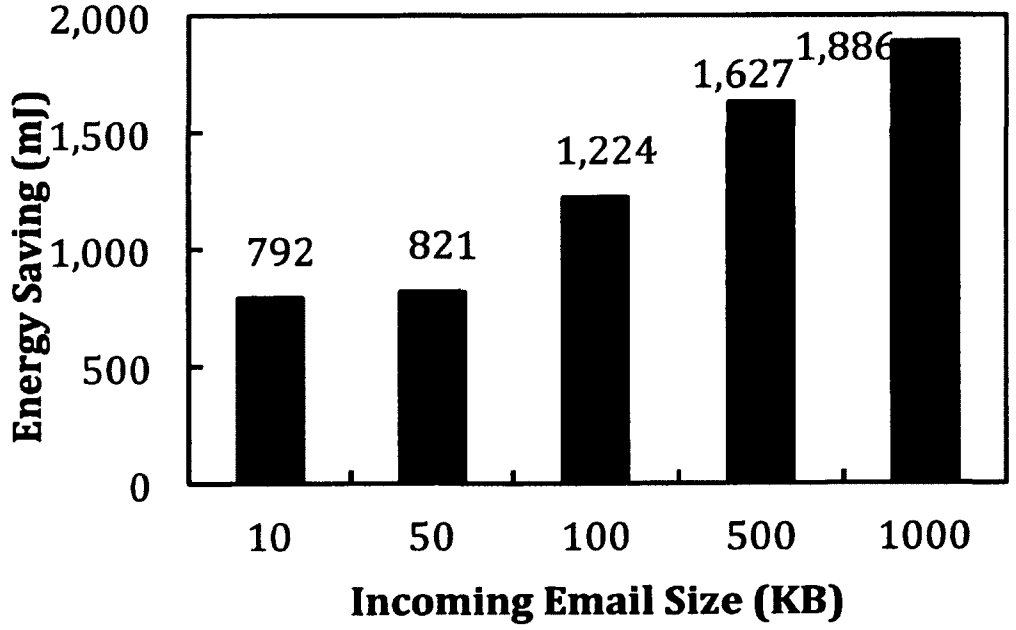

Figure 3.12: Energy saving of in-memory data processing in different email sizes

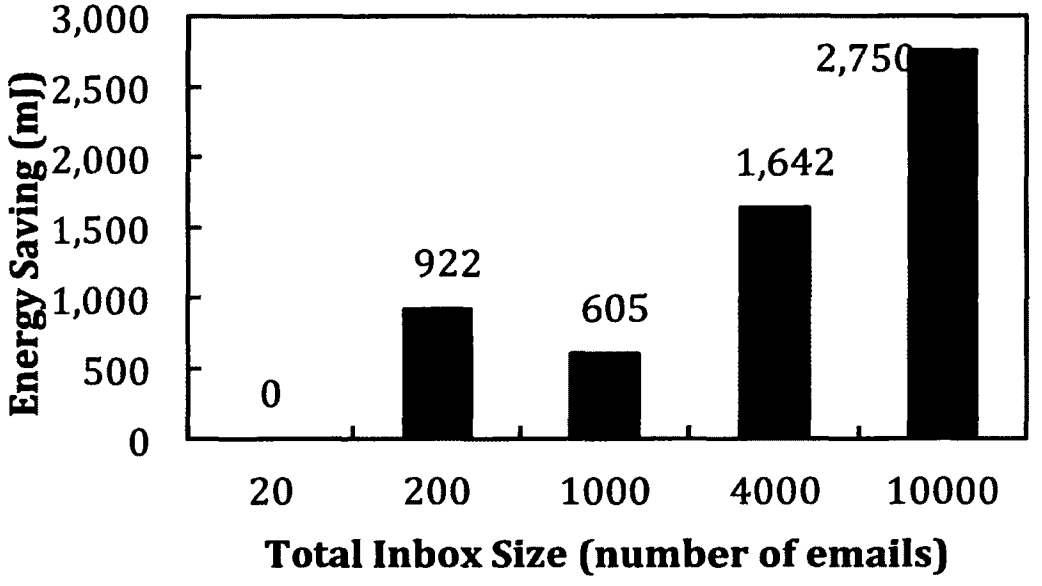

Figure 3.13: Energy saving of using a small cache inbox for email receiving with different total inbox sizes

4 Access Point Association in Wireless LAN

It is already very common for Wireless LAN clients like mobile phones and laptops to face multiple choices of APs because of the high density deployment. Which AP to attach is not a trivial question especially when there are a lot of nearby users who may inference with each other. A user selecting an inappropriate AP will experience bad service, or even hurt other users' throughput. The current technique of AP selection is for the user to selfishly pick the AP with the strongest signal, or RSSI value. The intuition is that factors like multipath effect and path loss which reduce throughput's will have a smaller effect when the user is communicating with an AP with a larger RSSI.

This simple strategy might fail when there is a large number of users crowded together. Consider the case when we have two APs on orthogonal channels, one with much stronger signal strength than the other, and a collection of users. All the users will simply pick the same AP (with the largest RSSI), so that the actual throughput of each user is very small because of channel contention between users. Based on this observation, alternative criteria such as selecting the AP which yields the largest throughput have been suggested.

However, it is unclear how well this *selfish* strategy will perform when every user attempts to connect to the AP which is able to maximize their own throughput.

Unlike an AP's RSSI value measured by a user which is not affected by additional users associating with that AP, the AP's throughput will change as more users join in. Therefore, a user selecting an AP based on throughput may have to switch APs constantly, hence lowering overall performance.

In practice, we believe a good performance is to achieve the maximized minimal throughput for all clients. Using this simple but reasonable metric, we seek to design a practical distributed protocol for AP association. We theoretically analyze the worst-case performance of the selfish strategy, and introduce an online algorithm that achieves a better worst-case performance. Incoming user employing this algorithm determines an irrevocable association, only making use of the load information on the nearby APs, in order to minimize the $\mathbb{L}_p$ norm of the loads on all APs at the moment. Based on our online algorithm, we have implemented an association protocol, SmartAssoc, for commodity hardware driver at the client side. This protocol works well with current legacy 802.11 APs. Using a combination of real experiments and extensive simulation, we demonstrate that our online association protocol performs better than the RSSI-based and selfish AP selection.

The rest of this chapter is as follows. We give an overview of the related work in Section 4.1, and explain our motivation in Section 4.2. Section 4.3 examines the selfish strategy in distributed wireless AP selection scenario in a quantitative way. Section 4.4 proposes our online algorithm as the association strategy for the protocol design of SmartAssoc, and characterize its performance properties through theoretical analysis in a realistic model. We present the practical and efficient implementation of proposed association protocol on the off-the-shelf wireless LAN adapter in Section 4.5. SmartAssoc is demonstrated through real experiments in Section 4.6, and simulation results are provided in Section 4.7. Finally, Section 4.8 concludes.

4.1 Related Work

AP association plays an important role in improving wireless performance [43, 48, 55, 60, 62, 67, 75, 79, 84, 90]. The signal-noise ratios, which is popularly used for access point selection in 802.11, has demonstrated as a bad idea by G. Judd et al. [44]. In their paper, some other criteria like AP load information are taken into consideration for improving the network performance and achieving load balancing. Metrics evaluating the AP load are interested by both academic community and industry. [86] relies on passive measurement of delay intervals between the time when a beacon is scheduled for transmission and its eventual transmission to estimate an AP's load. In [68], a combined metric is applied, which includes the number of stations associated, mean RSSI for the station set of AP and regular RSSI. Similar AP-assisted approach is proposed in [29], to give associated clients the information about load through beacons.

Some solutions concerning this association issue are proposed in other directions. [26] creates multiple virtual interfaces based on one single wireless card, and make them communicate with associated APs like simultaneously. However, it needs an efficient association mechanism to define which AP and how long to connect, as well as a low handoff overhead. [46] aggregates the bandwidth available at accessible APs and also balances their loads by introducing a fast switching and a scheduler to distribute client's time across APs so as to maximize throughput. Both of them differs from techniques above in that they do not pick a single AP, but rather multiplexes the various APs in a manner that maximizes throughput. In paper [21]and [65], however, they believe association decisions must rely on a global view of the entire WLAN, rather than the local viewpoint of an individual client or AP. [65] introduces a central controller to aggregate information received from all APs and also control them based on this information. In paper [21], more com-

plicated central scheme for AP association is theoretically discussed. It claims to achieve both balanced load and max-min fairness by providing central algorithmic solution according to a formulation indicating strong correlation between AP loads and clients' bandwidth allocations. And it concerns multiple association model, which is not mentioned in [65]. Actually it is hard to change a wireless network architecture already deployed, and not practical as all users are operating in the decentralized manner.

The selfish behavior of users in a congestion game has been studied theoretically. A special case where each user's decision is a singleton set is considered in [85], while [61] describes a class of congestion game where the payoff function associated with each resource is user specific. The convergences under different load balancing scenarios are provided in [36]. In this work we model the decentralized AP selection with selfish users as an extension of the weighted singleton congestion game in which the weight of a user varies as the associated AP changes. Other modelings of the wireless infrastructure selection include [63] and [25], targeting at different scenarios and goals. The major distinction of our work is that we design and implement on the commercial chipset an online greedy AP selection protocol in the distributed manner. The performance is supported by theoretical proof and demonstrated both in real experiments and simulations.

Compared with decentralized methods, work by [21] and [65] uses the idea that better AP association decisions can be obtained by relying on a global view of the entire WLAN, or an extra centralized controller. AP side system is modified in [65] to aggregate workload information and provide association control according to it. In [21], a more complicated central scheme for AP association is discussed.

There are also other papers discussing how to multiplex multiple APs. In [26], it created multiple virtual interfaces based on one single wireless card, and made them communicate with associated APs like simultaneously. The paper [46] built a

multi-interface association mechanism to distribute a client's data traffic on multiple accessible APs in a scenario where the backhaul link is the bandwidth bottleneck.

We take a practical wireless model that well capture the complicated interference among APs and clients. Considering interference like [89] makes great impact on the real world performance of algorithms designed upon this model. From the technical perspective, the load balancing literature provides the foundation of our solution to the formulated problem of AP association. We borrow results from the literature [19, 53, 80] to better understand the hardness of our formulated problem. We elaborate each work later when it is used. Load balancing also finds applications in other fields, such as wireless sensor networks [56].

4.2 Motivation

We consider an IEEE 802.11 infrastructure network [59], in which there are m APs and n stationary clients or users. Given no central controller and local information, All the clients are allowed to freely choose an AP within the transmission range to associate with. The goal of this project is *to maximize the minimum throughput over all clients.* Through this chapter, we consider the MAC layer throughput if no specification.

The AP association protocol currently employed in IEEE 802.11 networks lets a client associate with the AP that gives the strongest signal. We term this the Best-RSSI strategy. However, the Received Signal Strength Indication (RSSI) may not be a good indicator of throughput changes. [91] provides an experimental example to demonstrate this poor correlation between RSSI and throughput.

It is easy to observe that the relatively stable state of the RSSI does not reflect the relatively intensive fluctuation of the sampled throughput. Thus, RSSI is not suitable and accurate enough to evaluate an AP's performance. It is possible to

end up with a situation in which all clients connect to a single AP. In this case, the competitive ratio[1], in terms of average client throughput, can be as bad as $1/m$. When m is large, this plummeting performance becomes unacceptable.

4.3 Selfish User Strategy

One natural alternative to solving the above problem, with respect to our goal, is to let the clients behave myopically by applying in decentralized AP selection the *best-reply* policy. Explicitly, it means that every user keeps moving to associate with the AP that could offer it the best throughput *until no user can gain higher throughput by unilaterally deviating from its current decision (Nash Equilibrium).*

To simplify the analysis for selfish users, we make two assumptions in this section. In the next section, we will use a more realistic assumptions. First, we assume that the interference between the communications of two APs is not considered, i.e., the nearby APs operate on orthogonal channels. Second, the association procedure of a user is considered as an atomic operation, so only one user perform association at a time. The time at which a user makes a decision to change APs is marked as a *decision step*. However, we do not require users to follow a certain decision order, which means in each decision step the user who is picking a new AP could be any one. Under these assumptions, we will show that such selfish user game converges to a Nash Equilibrium often having non-optimal performance. More complicated scenarios even cannot grantee the existence of the Equilibrium state [36, 61].

We denote by U_a the set of users connecting with AP a. So let $n_a = |U_a|$ represent the cardinality of this set. We designate by st_u the percentage of service time

[1]competitive ratio is the performance ratio, with respect to some metric, between worst outcome of certain association protocol and the optimal strategy case. Here we mean the competitive ratio in terms of throughput. Later, after we transform the problem, we mean the competitive ratio in terms of load.

the user u gains from associated AP, and T_u corresponds to the throughput of u. And for any user u and AP a, we use R_{ua} to denote the *transmission rate* under the situation only u is associating with a. R_{ua} varies even for the same user. For the rest of this section, unless otherwise specified, the transmission rate refers to the effective transmission rate, which considers the overhead caused by retransmissions, random backoff and so on.

To examine the performance of this protocol, we consider two aspects: convergence and competitive ratio. The competitive ratio here is equivalent to the *price of anarchy*[2](PoA) using minimum user throughput as social cost. The following subsections show first whether the selfish user protocol will eventually stabilize and how fast the protocol will achieve convergence in general, and after that give the competitive ratio of the protocol.

4.3.1 Convergence of the Selfish Strategy

In this subsection, we will show how to model this selfish throughput strategy as a special case of the weighted congestion game, where the weight of a user varies as the associated AP set, which is singleton, changes. This game is proved to be converged with not ideal speed by leveraging the technique similar to [36]. We start with a lemma to characterize the throughput calculation.

Lemma 1. *All the users on an AP a have the same throughput T_a*

$$T_a = \frac{1}{\sum_{i \in U_a} \frac{1}{R_{ia}}} \tag{4.1}$$

Proof. Owing to *Carrier-Sense Multiple Access with Collision Avoidance* (CSMA/CA) protocol, all the users associated to the same AP, no matter what their transmission rates are, have a fair chance to seize the channel for packet transmission.

[2]the ratio of the worst-case social cost among all Nash Equilibria over the optimal cost

Therefore, given the same packet size z, any user u connecting to AP a with a transmission rate R_{ua} is assigned a portion of service time from a:

$$st_u = \frac{\frac{z}{R_{ua}}}{\sum_{i \in U_a} \frac{z}{R_{ia}}} = \frac{\frac{1}{R_{ua}}}{\sum_{i \in U_a} \frac{1}{R_{ia}}} \tag{4.2}$$

It is obvious that every user on the same AP has throughput:

$$T_a = T_u = st_u \times R_{ua} = \frac{1}{\sum_{i \in U_a} \frac{1}{R_{ia}}} \tag{4.3}$$

□

Given the Lemma 1 and assumptions we made, the selfish AP selection can be modeled as an extension of the congestion game. For sake of clarity and consistence in terminology, we describe this model in the wireless LAN scenario. Consider a set M of APs, each having a load function depending on the total weight of the users associated (Definition 1), and a set U of users, each of whom only can choose one AP from a permissible subset of M (in the absence of a coordinating authority). The weight of a user i on AP j (i.e. the load imposed by the user) is defined as the reciprocal of the transmission rate, $\mathbb{L}_{ij} = \frac{1}{R_{ij}}$. Accordingly, maximizing the minimum throughput over all clients is equivalent to minimizing the maximum load over all APs.

Definition 1. *The load of an AP a, $\mathbb{L}_a$, is*

$$\mathbb{L}_a = \sum_{i \in U_a} \mathbb{L}_{ia} = \sum_{i \in U_a} \frac{1}{R_{ia}}$$

For the convergence proof, we introduce a sorted vector (in ascending order) of all users' throughput as the potential function. According to $Lemma$ 1, we can simplify this vector to a new one $\vec{\mathcal{T}}$ by using T_a above to represent respectively the current throughput of every user associated with AP a (for $\forall\, a \in M$, where M is the

set of all APs). Put differently, given a user i associated with AP a, its throughput is replaced with T_a in the $\vec{\mathcal{T}}$.

The following defines the *lexicographic order* on different vectors ($\vec{\mathcal{T}}$).

Definition 2. *One vector $\vec{\mathcal{T}}$ defined above is called lexicographically larger than the other one $\vec{\mathcal{T}}'$ if $\vec{\mathcal{T}}$'s first unequal element is larger than its corresponding position index one in $\vec{\mathcal{T}}'$, where both vectors are in ascending order.*

Definition 3. *In $\vec{\mathcal{T}}$, $T_a(s)$ denotes the throughput T_a at the decision step s.*

We show the convergence of the protocol by identifying a potential function and showing that this potential strictly increases after each step. Consider the vector $\vec{\mathcal{T}}$, in an ascending order of all users' throughput.

Theorem 1. *$\vec{\mathcal{T}}$ lexicographically increases when a user i moves from AP j to k for better throughput.*

Proof. Based on the assumption that interference is not considered in this section, we know this migration only influences two components of $\vec{\mathcal{T}}$: one corresponding to the throughput for AP j that user i just left, while the other corresponds to the AP k that i has joined. Other components remain unchanged. Suppose this is the $s+1$-th decision step. Because user i moves for a higher throughput, $T_k(s+1) > T_j(s)$; and because AP j has one less client, its throughput increases: $T_j(s+1) > T_j(s)$. In a word, if $T_j(s)$ is the pth component in $\vec{\mathcal{T}}$ at step s, $T_j(s+1)$ and $T_k(s+1)$ reside at two positions whose indexes are no smaller than p^{th} (at the right side of pth position including p).

Assume in $\vec{\mathcal{T}}$ at step s, $(m-q)^{th}$ (recall m is the number of APs) is the first position in which the value is larger than the one in position p, i.e., there are q throughput larger than $T_j(s)$ in $\vec{\mathcal{T}}$. Note that only if no throughput is equal to $T_j(s)$ in $\vec{\mathcal{T}}$ at step s, $m-q=p$. After step $s+1$, this number q increases by 1 for the reason mentioned above (T_j moves to the right). Thus, the $(m-q-1)^{th}$ position

becomes the first position that holds different values for step s and step $s+1$ in $\vec{\mathcal{T}}$. Obviously, according to the definition of lexicographical order, the vector $\vec{\mathcal{T}}$ at step $s+1$ is larger. □

Above theorem is also applicable to the extended scenario that there are new users coming into the network.

Since we have shown that users' migration always increases the potential - $\vec{\mathcal{T}}$, this gives us an upper bound (Theorem 2) on the convergence time in general.

Theorem 2. *Without specifying the concrete underlying configuration, this network (m APs and n clients) reaches the equilibrium in at most m^n steps.*

Proof. It is equal to the number of different sorted vectors, which is bounded by the number of network topology snapshots. In other words, after performing at most m^n steps, this potential function $\vec{\mathcal{T}}$ will come to a state at which it will not be larger any more. This means that no matter which user has the chance to make a decision at the next step, it will stay on its current AP from its selfish point of view. □

We have shown that the network will finally converge to an equilibrium within bounded steps. However, the number of steps may be exponentially growing with respect to the network size, which is presented in the following theorem. Whenever an AP accepts a new user (maybe users from other APs), we count it as a new AP association.

Theorem 3. *For a network with m identical APs and m^2 users, there exists a scenario where the number of AP associations is at least 2^m.*

Proof. Our construction is inspired by, but quite different from, the identical machine scheduling in [37].

The network is as follows. The users are divided into m groups $G_1, G_2, \ldots, G_m$, each of which has m users. Users in the same group have the same weight.

We set the weight of each group recursively. For G_1, its weight is 1. For G_i, its weight is set as more than the sum of the weights of previous groups' users, say $w_{G_i} = m\sum_{j=1}^{i-1} w_{G_j} + 1$. Such weight setting can guarantee that two APs have the same load if and only if they serve the same number of users from each group. In the equilibrium after all users join the network, each AP serves m users, each of which comes from a different group. Since each user's choice is deterministic, i.e., it chooses the lightest load AP (note here APs are identical), the number of AP associations is determined by the order in which users join the network and the order in which they make adjustments. In the following, we construct an adversary that controls such orders to cause more than 2^m AP associations. Whenever the adversary *releases* a user, the user can join the network, or adjust APs, based on the best-reply policy.

Let $F(m)$ be the resulting number of AP associations for an m-AP network under the adversary's strategy $\mathcal{S}(m)$ described as follows. We will prove $F(m) \geq 2^m$ by induction.

For $m = 2$, the adversary's strategy $\mathcal{S}(2)$ is to release users in an arbitrary order (e.g., light weight first). It holds trivially that $F(2) \geq 2^2$ since there are 4 users and each of them invokes at least one AP association.

Suppose $F(k) \geq 2^k$ and this is resulted from the adversary's strategy $\mathcal{S}(k)$. Consider the case when $m = k+1$. Now each group has $k+1$ users. The adversary will use the following strategy $\mathcal{S}(k+1)$. First, it applies the following strategy, creating a k-AP network sub-problem.

- Release one user from G_{k+1}.
- Select k users from each G_i where $i = 1, 2, \ldots, k$, making k^2 users. Apply $\mathcal{S}(k)$ to them.

The user from G_{k+1} will occupy an AP. Denote this AP by AP_a. The later released k^2

users will only consider the other k APs, because even the sum of their weights is less than AP_a's current load. Therefore, these k^2 users, together with the k available APs, create a k-AP instance, and the number of involved AP associations is $F(k)$.

In the next, the adversary releases several users, whose associations we ignore but will lead to an interesting equilibrium.

- Release the rest users from $G_1, G_2, \ldots, G_k$ (i.e., one user per group).
- Release $k-1$ users from G_{k+1}. Wait until equilibrium.

Consider the equilibrium. There must be k APs, each of which serves exactly one user from G_{k+1} and does not serve other users. All the $k(k+1)$ users from the first k groups are crowded in one AP (denote it by AP_b).

At last, the adversary creates another k-AP sub-problem.

- Release the last user. It is from G_{k+1}, and it will choose AP_b.
- For users at AP_b, select k users from each G_i where $i = 1, 2, \ldots, k$, making k^2 users. Apply $\mathcal{S}(k)$ to them.

The released user from G_{k+1} will choose AP_b due to lightest load. After it associates with AP_b, all the other users at AP_b have the incentive to move to the other k APs. None of the selected k^2 users will associate back with AP_b at any time until the equilibrium, because AP_b has at least $k+1$ users (one user per group) so that there must be some AP that is less loaded. (Consider i from $i = k+1$ to $i = 1$. If there exists an AP serving two users from G_i, then there is at least one AP serving no user from G_i. This AP is less loaded than AP_b.) The k^2 users and the other k APs create a k-AP instance, contributing to $F(k)$ AP associations.

We can see that, under strategy $\mathcal{S}(k+1)$, the resulting $F(k+1)$ is composed of at least two $F(k)$, so we have $F(k+1) \geq 2F(k) \geq 2^{k+1}$. The theorem follows immediately. □

Note that this theorem considers identical APs, a special case of our model that assumes unrelated APs. The implications of the theorem are two folds. First, it is impossible to bound the convergence time by polynomials of m and n. Second, some user may need to change AP for exponentially many times ($2^m/m^2$), which significantly degrades performance.

4.3.2 Competitive Ratio

In the following, we obtain the competitive ratio with respect to the minimal throughput over all clients in the selfish protocol. We still assume that the number of APs is m and the number of clients (or users) is n. We also assume that, among all users, the maximal available transmission rate is R_{max} and the minimal available transmission rate is R_{min}. Recall that we define the load a client imposes to an AP as the reciprocal of the transmission rate of a client when connecting to an AP. Therefore, we define $\mathbb{L}_{max} = \frac{1}{R_{min}}$, and $\mathbb{L}_{min} = \frac{1}{R_{max}}$.

In a Nash Equilibrium of the selfish strategy, suppose the most loaded AP is k, which has a load $\mathbb{L}^S$, i.e., $\mathbb{L}_k$. Since this selfish user game reaches the equilibrium, any client connecting to AP k is not willing to move to any other AP j. That is, $\mathbb{L}^S \leq \mathbb{L}_{max} + \mathbb{L}_j$ for $\forall j \in \{a | a \in M \& a \neq k\}$. Thus, $\mathbb{L}^S \cdot m \leq \mathbb{L}_{max} \cdot (m-1) + \sum_{j=1}^{m} \mathbb{L}_j \leq \mathbb{L}_{max} \cdot (m-1) + \mathbb{L}_{max} \cdot n$

$$\therefore \quad \mathbb{L}^S \leq \frac{\mathbb{L}_{max} \cdot (n+m-1)}{m}$$

Then in the optimal strategy, the maximal load over all the APs is $\mathbb{L}^O \geq \frac{n \cdot \mathbb{L}_{min}}{m}$ In sum, the price of anarchy is

$$\frac{\mathbb{L}^S}{\mathbb{L}^O} \leq \frac{n+m-1}{n} \cdot \frac{\mathbb{L}_{max}}{\mathbb{L}_{min}}$$

This bounds the PoA from above. In the following, we bound the worst-case

PoA from below.

Theorem 4. *There exists a scenario where the competitive ratio is no smaller than* $(m-1)/(1+\epsilon)$ *where* m *is the number of APs and* ϵ *is a small positive constant.*

Proof. The example below first appears in [17], and recently in [52] for slightly different purposes (centralized greedy load balancing and sequential load balancing game). We show that this example also works in our context (distributed selfish user association).

Suppose there are m APs and $m-1$ users. Each user $i \in U$ can only associate with two APs, i and $i+1$. The weights are set as follows. For $i \in U$, $L_{i,i+1} = i$ and $L_{i,i} = 1+\epsilon$. For this example, the maximum load of the optimal solution is $1+\epsilon$.

The adversary can release users sequentially as $1, 2, \ldots, m-1$. It is easy to see that user i will associate with AP $i+1$ in the equilibrium, causing the maximum load to be $m-1$. Therefore, its competitive ratio is $(m-1)/(1+\epsilon)$. □

The theorem shows that the price of anarchy is $\Omega(m)$. In the following section, we use an online algorithm with competitive ratio $O(\log m)$, which is an exponential improvement.

In summary, the selfish user protocol has a high convergence time and a poor performance in some network scenarios.

4.4 Online Algorithm

We have shown that both the Best-RSSI and selfish user protocols perform poorly under certain scenarios. In this section, we introduce our practical online association strategy. Our online algorithm considers merely communication load including interference and congestion, which provides a more realistic model.

The protocol is simple without assuming a complex interaction among clients and APs. We merely assume that, when a new client joins the network, it can

measure the loads (recall the Definition 1) of all APs within its hearing range. If the client does not affect an AP, or does so with negligible influence, it does not need to know the load on that AP. For example, if a client is far away from an AP, the interaction between them or the influence would be marginal and the client will not consider the information on that AP when it makes its AP association decision. We will show by implementation in section 4.5 that this assumption can be approximately achieved through a practical and low-overhead measurement method. We do not even assume how the loads will be changed when a client joins -- although we do assume the load on each AP will be non-decreasing when a client joins.

In the following, we examine several scenarios to show the ramifications of our assumptions and demonstrate how much our assumptions conform to the reality.

- **Interference with APs**: When client i joins the network, it might interfere with the transmission and reception on several APs. We denote the loads imposed on AP j after i makes its association decision as $\mathbb{L}_{ij}$. Note that j may not be the AP that client i associates with.

- **Interference with clients**: When client i joins the network, it might interfere with another client k. Even though i may not directly interfere with the AP (say AP j) that k is associated with (possibly due to being out of transmission range), the interference of i on k's communication may change the load on AP j. If the load on AP j is visible to client i, this scenario is amenable to our analysis; otherwise, we will ignore the load imposed by this indirect influence because the load change due to this rippling effect is marginal.

- **Myopic network configuration**: When a client i joins the network, it may not see all the APs because of the limited communication range. If the client does not see an AP, we assume the load change on that AP owing to the joining of client i is negligible. Furthermore, in this case, client i will not be able to

associate with that AP because there is no usable bidirectional link between the client and the AP.

In summary, we study a more practical and complicated wireless LAN model here than the one used in the previous section. The weight (communication load) a user adds on the associated AP might vary as the local network configuration changes or new incoming clients appear.

We now formally define the problem. Due to interference, a user may increase load to other APs, in addition to the AP that it associates with. To capture such interference effects, we denote by $\delta_{i,j,k}$ the increased load to AP j if user i chooses AP k. The generalized AP association problem (GAS) can be formulated as an integer linear programming problem.

$$\begin{aligned} \min_{\mathbf{x}} \max_{j} \quad & \sum_{i,k} x_{i,k}\delta_{i,j,k} \\ \text{subject to} \quad & \\ & \sum_{k} x_{i,k} = 1, \quad \forall i \in U \\ & x_{i,k} \in \{0,1\}, \quad \forall i \in U, k \in M \end{aligned} \tag{GAS}$$

where $\mathbf{x} \in \{0,1\}^{|U|\times|M|}$ is the association matrix with elements $x_{i,k}$. The two constraints force each user to associate with exactly one AP.

This model generalizes the previous model in Section 4.3. If each user can only increase its associated AP's load, i.e.,

$$\delta_{i,j,k} = \begin{cases} \mathbb{L}_{i,j} & \text{if } j = k \\ 0 & \text{otherwise} \end{cases}$$

then it becomes the previous problem.

In the new problem, for a specific user i and an AP j, even if i is not associated

	AP 1	AP 2
AP 1	3,3	4,2
AP 2	4,2	3,3

Table 4.1: The cost matrix for two users under different AP associations (left for user 1, and top for user 2). Each entry (a, b) is the cost for users 1 and 2 respectively.

with j, the exact load induced to j is dependent on which AP user i associates with. Specifically, $\delta_{i,j,k}$ may be different from $\delta_{i,j,l}$ for $k \neq l$. We do not assume any relationship between different $\delta_{i,j,k}$, only assuming that they are non-negative (a new association won't decrease any AP's load). To see the difficulty of this problem, we should note that the input information can no longer be modeled by a bipartite graph.

The selfish strategy in the previous section has worse performance under this general model. There may not exist an equilibrium. To see this, we encode the classic game, matching pennies, in the model. Consider the 2-user and 2-AP scenario. Let $\delta_{1,1,1} = \delta_{1,2,2} = \delta_{2,1,2} = \delta_{2,2,1} = 2$ and all others are set as 1. The resulting cost matrix is in Table 4.1. We can see that user 1 prefers to have the same choice as user 2 does, while user 2 prefers to differ. In distributed scenarios, they will keep switching APs forever.

In fact, the problem is a generalized version of unrelated machine load balancing problem. Therefore, all the hardness results of the latter problem trivially hold for the new problem. Specifically, no polynomial algorithm can achieve a competitive ratio less than 3/2 unless $P = NP$ [53]. No online algorithm can achieve a competitive ratio less than $\lceil \log(m+1) \rceil$ [19].[3] For the load balancing problem, all the constant approximation algorithms are based on linear relaxations. There are two kinds of rounding techniques [53, 80]. Among them, [80] is not applicable to our problem. It requires to sort the users by their loads to the same AP, while in our case, one user

[3]Their work considers a special case of unrelated machines. The machines have identical speed but each user can only select a subset of the machines. This case can be easily encoded by our model.

may contribute different loads to the same AP. The algorithm in [53] is applicable to our problem, but it is $\Omega(m)$-competitive, not a constant approximation algorithm, as shown in the following theorem.

Theorem 5. *To solve problem GAS, algorithm [53] is $(m+1)$-competitive where m is the number of APs. The bound is almost tight in that there exist a class of GAS instances that the algorithm gives m-competitive solutions.*

Proof. We first show how the algorithm works, then prove its competitive ratio, and then construct the set of bad input instances. The competitive ratio is proved by following the same procedure as in [53], where they proved the 2-approximation ratio for the traditional load balancing problem.

The algorithm performs binary search for the optimum load. In each iteration, it checks whether the optimal solution could yield maximum load T. To answer this question, it drops the integral restrictions and checks the feasibility of the following linear program.

$$\begin{cases} \sum_{i,k} x_{i,k}\delta_{i,j,k} \leq T, & \forall j \in M \\ \sum_{k} x_{i,k} = 1, & \forall i \in U \\ x_{i,k} \geq 0, & \forall i \in U, k \in M \\ x_{i,k} = 0 & \text{if } \exists j \in M, \delta_{i,j,k} > T \end{cases}$$

It is easy to see that, if the optimal solution yields maximum load T, then the linear program above is feasible. Suppose the linear program is feasible. We can find an extreme solution **x** (a vertex of the polytope defined by the constraints). The algorithm then rounds fractional solution to integral assignment by finding a maximum matching for the fractionally assigned users (users correspond to fractional variables in **x**). We relax the maximum matching requirement to an arbitrary

semi-matching if no maximum matching exists.

We now prove the competitive ratio. We will prove that the rounding of the fractional extreme solution **x** produces an integer solution $\widetilde{\mathbf{x}}$ such that

$$\begin{cases} \sum_{i,k} x_{i,k}\delta_{i,j,k} \leq T + mT, & \forall j \in M \\ \sum_{k} x_{i,k} = 1, & \forall i \in U \\ x_{i,k} \in \{0,1\} & \forall i \in U, k \in M \end{cases}$$

The extreme solution **x** contains at most $m+n$ non-zero variables, among which the non-integer variables correspond to at most m users. To see this, note that by definition an extreme solution must have mn^2 (the total number of variables) linearly independent constraints satisfied with equality. Among the constraints appeared in the linear program, at most $m+n$ of them could yield non-zero solution (the m constraints for APs and the n constraints for users). Therefore, there are at most $m+n$ non-zero variables in **x**. Since each user who corresponds to non-integer variables could contribute at least 2 non-zero variables, there are at most m such users by a counting argument.

Now we assign users with integer variables in **x** to the corresponding APs. After this assignment, the load at any AP is at most T. Then we assign users with non-integer variables. Recall that we assign fractionally assigned user to any machine that they have positive fractional assignment to, thus each fractionally assigned user contributes at most T load to each AP. Otherwise, we should have $x_{i,k} = 0$ due to the existence of j such that $\delta_{i,j,k} > T$. Given that there are at most m fractionally assigned users, the additional load to each AP can not exceed mT. Thus the claim is proved.

In the following, we construct the class of bad instances. On these instances, the algorithm gives m-competitive ratio (even with maximum matching strategy).

Consider the network with the same number of users and APs, i.e., $n = m$. We set

$$\delta_{i,j,k} = \begin{cases} s_{i,j} & \text{if } k = i \\ \frac{1-s_{i,j}}{m-1} & \text{if } k = i+1 \\ \infty & \text{otherwise} \end{cases}$$

where $s_{i,j}$ are to be defined later and $i+1$ is defined to be 1 for $i = m$. In other words, we set each user i to be able to associate with two APs, APs i and $i+1$. For $s_{i,j}$, we set

$$s_{i,j} = \begin{cases} 1 & \text{if } i+j \leq m+1 \\ 0 & \text{otherwise} \end{cases}$$

Now we have finished constructing the set of instances. We show an example of the inequalities imposed by APs for $m = 3$.

$$\begin{pmatrix} 1 & 0 & \infty & \infty & 1 & 0 & 0 & \infty & 1 \\ 1 & 0 & \infty & \infty & 1 & 0 & \frac{1}{2} & \infty & 0 \\ 1 & 0 & \infty & \infty & 0 & \frac{1}{2} & \frac{1}{2} & \infty & 0 \end{pmatrix} \begin{pmatrix} x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{3,3} \end{pmatrix} \leq \begin{pmatrix} \mathrm{T} \\ \mathrm{T} \\ \vdots \\ \mathrm{T} \end{pmatrix}$$

For the constructed instance, the optimal solution is to assign user i to AP $i+1$, yielding maximum load 1. In the following, we show that the algorithm may assign user i to AP i, resulting in maximum load m, which is a m-competitive solution.

Set $T = m$. Consider the solution

$$x_{i,k} = \begin{cases} \frac{1}{m} & \text{if } k = i \\ \frac{m-1}{m} & \text{if } k = i+1 \\ 0 & \text{otherwise} \end{cases}$$

It satisfies with equalities all the non-zero constraints imposed by the APs and users. We can check that it is an extreme solution (note that it is the unique solution to the $2m$ non-zero constraints). Since the algorithm only requires an arbitrary maximum matching from the fractional assignment, we can see that setting $x_{i,i} = 1$ gives a maximum matching. □

Since the known constant approximation algorithms in load balancing literature fail to provide constant approximation guarantee for our problem, it is natural to ask whether constant approximation algorithm exists. Answering this question leads to the work of [95], and we report it in a separate work due to independent interest.

Theorem 6 ([95]). *For any constant $c > 1$, there does not exist a polynomial time algorithm that is c-competitive for GAS, unless $NP = ZPP$.*

In this work, we are more interested in online algorithms, since users in a wireless network may come at any time. Currently, one of the best online algorithm for load balancing on unrelated machines is due to [22]. Their solution is $e \log m$-competitive. We show in the following that this algorithm is still applicable to our problem. More importantly, we prove that competitive ratio is still $e \log m$. This result is surprising in that (1) the competitive ratio is irrelevant of $\delta_{i,j,k}$ (2) the generalization does not increase difficulty for online algorithms. In addition, this competitive ratio is not far from the best we can do (up to a multiplicative constant e) for an online algorithm, since no online algorithm can do better than $\lceil \log(m+1) \rceil$ as mentioned before.

The online AP selection algorithm runs as follows. When a new client appears (in online fashion), it will make an irrevocable association with one of the visible AP so that the $\mathcal{L}_p$ norm of the loads on all the APs within its transmission range, after its join, will be minimized at the moment. Since the client is unable to affect the other APs that are not in its hearing range, this algorithm will minimize the $\mathcal{L}_p$ norm

of all the APs' loads in the system $(\mathcal{L}_1^p + \mathcal{L}_2^p + \cdots + \mathcal{L}_m^p)^{1/p}$ in each new association event. Then we have

Theorem 7. *If the protocol is to minimize the $\mathcal{L}_p$ norm of the loads (rather than to minimize the maximum load), then the online protocol gives a $r \leq \frac{1}{2^{1/p}-1}$-competitive ratio*

Proof. Suppose client $1, 2, 3, \cdots, i, i+1, \cdots, n$ join the network sequentially. Consider the situation when client i is joining the network. Let $\mathcal{L}_{ij}$ be the resulting load of AP j if users 1 to i all follow our protocol (i may not select j). Let $x_{i,k}$ be the indicator variable for the optimal choice such that $x_{i,k} = 1$ if and only if user i chooses AP k in the optimal solution. With a little abuse of notation, we let $\delta_{i,j} = \sum_k x_{i,k}\delta_{i,j,k}$. Let $\mathcal{L}_j^*$ be the load on AP j in the optimal solution for minimizing the $\mathcal{L}_p$ norm. We have $\mathcal{L}_j^* = \sum_i \delta_{i,j}$. If users 1 to $i-1$ follow our protocol but user i chooses the optimal AP, then the load of the current system is $(\mathcal{L}_{i-1,1} + \delta_{i,1}, \mathcal{L}_{i-1,2} + \delta_{i,2}, \ldots, \mathcal{L}_{i-1,j} + \delta_{i,j}, \ldots)$.

Following the main idea of [22], we derive that

$$\begin{aligned} & \sum_j (\mathcal{L}_{ij}^p - \mathcal{L}_{i-1,j}^p) \\ \leq & \sum_j ((\mathcal{L}_{i-1,j} + \delta_{ij})^p - \mathcal{L}_{i-1,j}^p) \\ \leq & \sum_j ((\mathcal{L}_{n,j} + \delta_{ij})^p - \mathcal{L}_{n,j}^p) \end{aligned}$$

The first inequality is true because in this step client i tries to minimize the $\mathcal{L}_p$ norm. The second one is true because $(x+\delta)^p - x^p$ is increasing with x when $p > 1$ and $\delta \geq 0$.

$$
\begin{aligned}
& \sum_j \mathcal{L}_{nj}^p \\
= & \sum_i \sum_j \left(\mathcal{L}_{ij}^p - \mathcal{L}_{i-1,j}^p\right) \\
\leq & \sum_i \sum_j \left((\mathcal{L}_{n,j} + \delta_{ij})^p - \mathcal{L}_{n,j}^p\right) && (4.4) \\
= & \sum_j \sum_i \left((\mathcal{L}_{n,j} + \delta_{ij})^p - \mathcal{L}_{n,j}^p\right) \\
\leq & \sum_j \left(\left(\mathcal{L}_{nj} + \sum_i \delta_{ij}\right)^p - \mathcal{L}_{nj}^p\right) && (4.5) \\
= & \sum_j \left(\mathcal{L}_{nj} + \mathcal{L}_j^*\right)^p - \sum_j \mathcal{L}_{nj}^p \\
\leq & \left(\left(\sum_j \mathcal{L}_{nj}^p\right)^{\frac{1}{p}} + \left(\sum_j \mathcal{L}_j^{*p}\right)^{\frac{1}{p}}\right)^p - \sum_j \mathcal{L}_{nj}^p && (4.6)
\end{aligned}
$$

where (4.4) is due to (4.4); (4.5) is due to the fact that $\sum_{i=1}^k ((x+\delta_i)^p - x^p) \leq (x + \sum_{i=1}^k \delta_i)^p - x^p$ for $p > 1$, $x \geq 0$, and $\delta_i \geq 0$;[4] (4.6) is due to Minkowski Inequality. Then

$$
2\sum_j \mathcal{L}_{nj}^p \leq \left(\left(\sum_j \mathcal{L}_{nj}^p\right)^{\frac{1}{p}} + \left(\sum_j \mathcal{L}_j^{*p}\right)^{\frac{1}{p}}\right)^p
$$

Let $r = (\sum_j \mathcal{L}_{nj}^p)^{\frac{1}{p}} / (\sum_j \mathcal{L}_j^{*p})^{\frac{1}{p}}$. We then have $2r^p \leq (r+1)^p$ and $r \leq \frac{1}{2^{1/p}-1}$.

Thus, our protocol is a $r \leq \frac{1}{2^{1/p}-1}$-competitive online algorithm to minimize the L_p norm. □

Theorem 8. *The online protocol is a $e \log m$ competitive protocol for minimizing the maximal load (or $\frac{1}{e \log m}$, w.r.t. maximizing the minimal throughput).*

Proof. Let the heaviest load among all APs running our protocol be $\mathcal{L}_m$ and the heaviest load among all APs in the optimal minimizing heaviest load protocol be $\mathcal{L}_m^*$.

[4] It can be proved by considering the function $f(s+t) - f(s) - f(t)$ where $f(s) = (x+s)^p - x^p$.

Thus, $m^{1/p}\mathcal{L}_m^* = (m \cdot \mathcal{L}_m^{*p})^{1/p} \geq (\sum_j \mathcal{L}_j^{*p})^{\frac{1}{p}} \geq \frac{1}{r}(\sum_j \mathcal{L}_{nj}^p)^{\frac{1}{p}} \geq \frac{1}{r}(\mathcal{L}_m^p)^{1/p} = \frac{1}{r}\mathcal{L}_m$. In other words, the $\mathcal{L}_p$ norm protocol is a $rm^{1/p}$-competitive online algorithm for minimizing the heaviest load on all APs. $rm^{1/p} \leq \frac{m^{1/p}}{2^{1/p}-1} \leq m^{1/p}\frac{p}{\ln 2}$. When $p = \ln m$, $m^{1/p}\frac{p}{\ln 2}$ reaches its minimum value, $e \log m$, in the positive real number domain $\mathcal{R}^+$. Thus, we choose $\mathcal{L}_{\ln m}$ norm, and the competitive ratio, correspondingly, is $e \log m$. □

Instead of being related to the number of APs and the ratio between the maximal and minimal rates, the competitive ratio of this protocol is linear to the logarithm of the number of APs, an almost constant competitive ratio for a small number of APs, which is deemed very promising since a constant competitive ratio algorithm usually gives a very good practical performance.

Furthermore, this algorithm has the advantage of computational simplicity and feasibility for practical implementation. The expected performance bound, for each client joined, is ensured just by the local network information at the moment it was coming as a new client. It is not necessary to reconsider its decision once a new association event occurs. In other words, our online algorithm takes exactly n steps to finish. In the next section, we demonstrate that this algorithm can be employed as a practical and light-weighted association protocol for off-the-shelf wireless LAN adapters.

4.5 Protocol Implementation of SmartAssoc

In this section, we describe how to efficiently realize SmartAssoc with low costs. Although aforementioned online algorithm is carefully designed to avoid large overhead caused by the measurement of nm^2 input parameters $\delta_{i,j,k}$, SmartAssoc still needs to address the real-world issue before we apply it in practice. That is how to estimate APs' load within a short time frame. It should not require any modification at the infrastructure side, so our implementation can be used in any open

802.11 networks or networks the client is able to access. To solve this problem, we propose a distributed light-weight AP load probing method in our SmartAssoc implementation, so that message exchanges between users and/or APs are not necessary. The load information is obtained by the user from target AP without association.

We consider the scenario that wireless link is the bottleneck of a communication connection. The discussion of other cases is out of the scope of this chapter. Thus, the workload of an AP is reflected by the wireless traffic on air for this AP. Here we monitor the uplink stream traffic, ignoring the downstream, because accuracy improvement of throughput measurement is small compared with the extra complexity in the implementation experience of [46]. Every channel is considered to be interference-free with others, as this type of interference is ignorable compared to interference inside the channel. Thus, the computation of the $\mathcal{L}_p$ norm of the AP's workload can be reduced to per-channel based computation, while the comparison is still among all channels.

The implementation of SmartAssoc is taken on the popular commercial wireless LAN adapter by taking advantage of the legacy standard 802.11 protocol. User space control is provided by the Click Modular Router toolkit [5], while association functionality of the MadWifi driver v0.9.4 [9] is directly taken over by SmartAssoc in the monitor model.

As mentioned above, SmartAssoc requires to measure every AP's load on the same channel when a client i joins a candidate AP j [5]. A natural way to obtain this information is to let the client i perform an association operation with j, and generate traffic on j while at the same time capturing an uplink data stream for each AP by passive listening. The packet retransmission and duplication does not count. However, the association process consumes a lot of time, especially for

[5]Assume i always has some communication demand after association

encrypted wireless networks. When the set of association candidates is not small, the user is not able to bear waiting so long. Nevertheless, sending data packets without association will lead to rejection from the object AP because of the IEEE 802.11 standard. Thus, we find a more light-weight way to obtain the equivalent load information without association. Currently 802.11 standards require an AP to respond to *probe requests*, even if the request is sent by a station not associated with the AP [6]. We leverage this to create a packet type to replace the real data packet in the MAC layer. The intuition is to generate modified probe request traffic to the object AP j, similar to the data traffic, to estimate other APs' loads as if the client i is associating with j. The detail modifications of the probe request packet are made as follows.

- We make the probe request uni-cast, forcing the target AP to return an ACK upon receiving the packet. This behavior is similar to a station transmitting data packets to an AP. And this process is important as well for calculating the throughput.
- We change the subtype flag in the packet header to prevent the AP from returning a probe response, in order not to introduce unnecessary traffic to the network.
- The packet size, transmission rate, and inter-arrival time of modified probe requests are packet-wisely customizable by the user, which is able to provide more accurate throughput information for specific estimation based on upper-layer applications. This feature is implemented in the probing generator module. Its performance is shown in the next section.

We also implement an AP filter to make the candidate AP list programmable. The user can select a preferred channel, network, and minimal RSSI threshold to cus-

[6] It is done automatically by the firmware, transparent to the upper layers

tomize the list. Only qualified APs will be considered for estimation to reduce overhead.

The implemented protocol is described as pseudo code (Algorithm 1). A list of candidate APs is determined in the first place for estimation according to the beacons. In the list, the candidates from the same channel i group as a set $\mathcal{C}_i$. For each target AP j in $\mathcal{C}_i$, the user injects a probing traffic, which consists of modified probe request packets, to the AP j, while measuring all members' loads. After these measurements, the user can calculate the $\mathcal{L}_p$ norm loads for all members within this channel set, $(\sum_{k \in \mathcal{C}_i} \mathcal{L}_k^p)^{1/p}$, where p is the natural logarithm of the number of APs. This $\mathcal{L}_p$ norm value, influenced by association with the target AP, will be compared with the current best candidate AP among all channels. The comparison strategy applied in the best candidate updating stage is controlled through two programmable parameters. The first one is the norm-difference threshold T_{nd}, and the second one is RSSI-difference threshold T_{rd}. If the norm for the target AP are at least T_{nd} smaller than the norm for the current best AP, the target AP will be the best candidate AP instead of current one. Otherwise, the algorithm continues to check whether this norm is just smaller than the current best candidate's, as well as whether the target AP's RSSI is at least T_{rd} more than the best candidate's. If so, the target AP will be the best; for all other cases, we keep the current best AP without updating. The user treats every member of a channel set as the target member respectively, and repeats this process for each channel set. After the evaluation of all candidate APs, the user will associate with the final best candidate AP.

4.6 Evaluation

We verify the feasibility of our online algorithm and demonstrate the performance of SmartAssoc implementation in this section. Each client is powered by a 1.66GHz

Algorithm 1 Protocol Description

```
Discovers all available APs
Applies the filter on discovered APs
Puts all filtered APs into candidate list
for each C_i do
  for each AP j in C_i do
    Generates a probing traffic to j
    Estimates the new load of each AP within C_i
    Calculates the L_p norm change
    Updates the best candidate AP according to T_nd and T_rd
  end for
end for
Associates with the best candidate AP
```

CPU with 1 GB RAM, running on Linux kernel 2.26.24. A D-Link WNA-2330 with Atheros 5312 chipset wireless card is used.

4.6.1 Application Aware Probing

Since our modified probing stream, used to emulate the real data stream, is programmable in terms of the packet size, inter-arrival time, and transmission rate, it is easy to generate specific streams to mimic the data stream of a certain application. Thus, the client is able to find out the "best" association AP with respect to the application it wants to use. Figure 4.1 demonstrates how similar our probing can be to the secure copy (SCP) and VoIP protocols, respectively. The SCP protocol used is the Unix *scp* command line program. SCP transfers a single file from a laptop to a remote desktop on the Internet through a commercial AP on the channel 6 with RSSI -58. The packet size of the probe emulating SCP is 1500 bytes, and the inter-arrival time is presented at the right of Figure 4.1. On the left side of Figure 4.1, we choose Skype as the VoIP application to set up a communication between two laptops through the same commercial AP on the channel 1 with RSSI -33. The probing packet used in this experiment is 200 bytes on average. For both experiments, we first brought up our driver module to create virtual interfaces in

the kernel for all upper-layer applications. Then a script was executed to associate with the target AP in each experiment. After association and IP assignment, we ran the application and our probing generator to generate the traffic, respectively. The traffic traces are captured by Wireshark for cumulative distribution function (CDF) calculation of the inter-arrival time. In these two experiments, the transmission rate for all streams is fixed at 36Mbps.

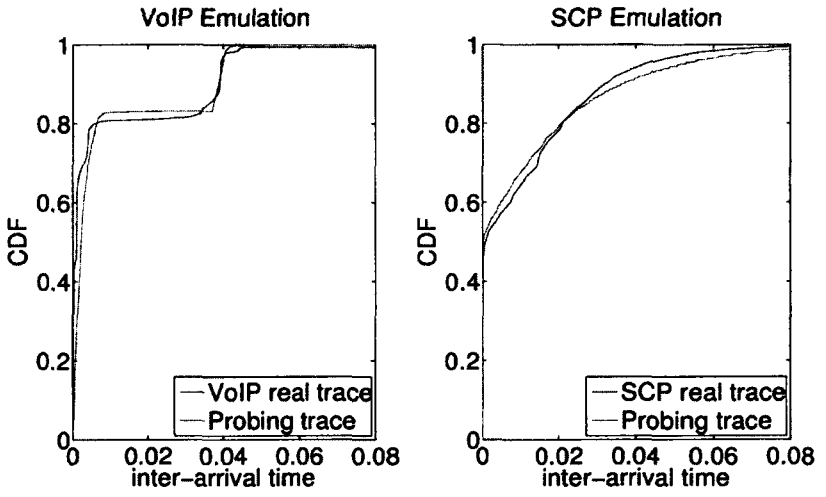

Figure 4.1: Upper-layer application stream emulations.

4.6.2 Measurement Accuracy

The APs' load information needed in SmartAssoc is derived by monitoring the wireless channels. Although it is not necessary for monitoring to capture all packets on the air, a relatively accurate measurement can help to make a better association decision. Thus we conducted a series of experiments to investigate the capture missing, which is the main factor to cause the measurement error of the load. In this chapter, we are focusing on the Atheros 5212 chipset, while other chipsets can be easily studied like this as well. We set up two laptops with a distance of 10 feet between them. One is the target laptop, which is used to generate a data stream for measurement. The other laptop acts as the monitor to estimate the data throughput

from the target laptop. To make our experiment comprehensive, we use different transmission rates when transmitting the data streams. The rates used are 1Mbps, 2Mbps, 5.5Mbps, 11Mbps, 18Mbps, 36Mbps, and 54Mbps. We also use different inter-packet times of 5ms, 10ms, 15ms, and 20ms at each rate, respectively. The packet size in all trials is 1000 bytes, and every trial last 5 seconds for data stream generating and capturing. The experiment results are shown in Figure 4.2. The x axis presents the captured packet number in each trial at the monitor laptop side, while the y axis presents the number of transmitted packets counted at the target laptop side. It is clear that if there is no error, all points (star or circle) should fall on the green dash line $y = x$. Based on these experimental results (excluding the six outliers), we are able to calculate the best linear fitting by using the Least Square method, which is shown as the red line, $y = 0.8806 \times x$. 0.8806 is used for estimation calibration with respect to the Atheros 5212 chipset, i.e., $estimation = \frac{measurement}{0.8806}$

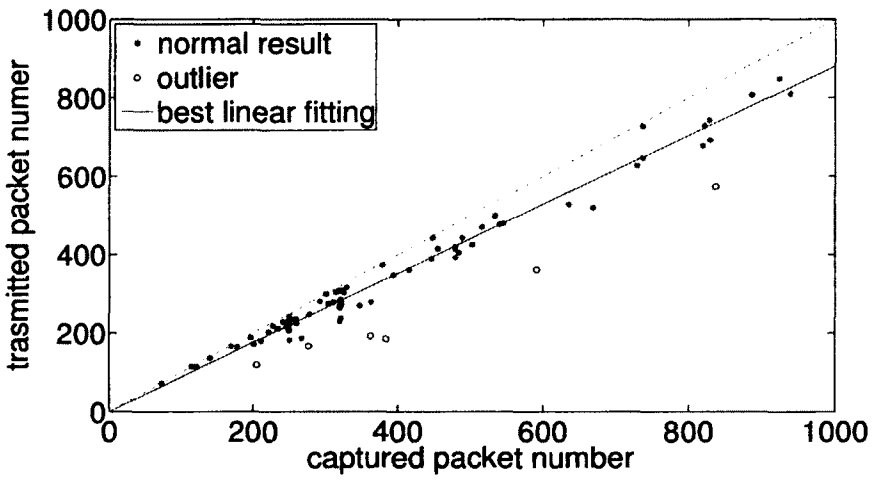

Figure 4.2: Calibration Experiments for Atheros 5212 chipset

4.6.3 Measurement Duration

The long measurement time of channel traffic, although it benefits the accuracy of workload estimations, extends the delay time before associations and consumes

more power. Thus, we conducted experiments to find out what is the minimum measurement duration given certain measurement error. To facilitate experiments, we implemented a testing program based on `libpcap`. It has two functionalities. The first one is a standalone component of AP workload estimation described in our protocol. The second one is a wireless traffic generator that can assemblies 802.11 packets of our choice and transmits them at given traffic patterns. Two laptops installed this testing program are set in 5.18 GHz of 802.11a. One of them is used to generate a traffic to one AP set in the same channel. The inter-arrival time of generated wireless traffic follows the exponential and normal distributions respectively. The other one is to estimate this AP's workload in different time windows. The experimental results are illustrated in Fig 4.3 and Fig 4.4. We found out 50 ms is a proper duration that can achieve less than 3 percent estimation errors.

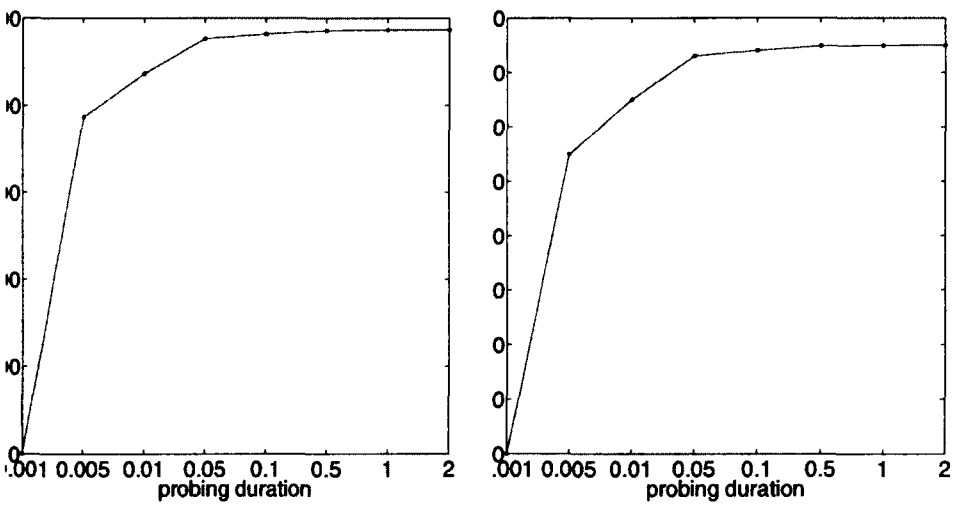

Figure 4.3: Estimation of workloads generated by two traffic patterns following exponential distributions with mean of $\frac{1}{973}$ and $\frac{1}{751}$ respectively. X axis is different traffic measurement time from 0.001 seconds to 2 seconds. Y axis is the estimated workloads in packets per second.

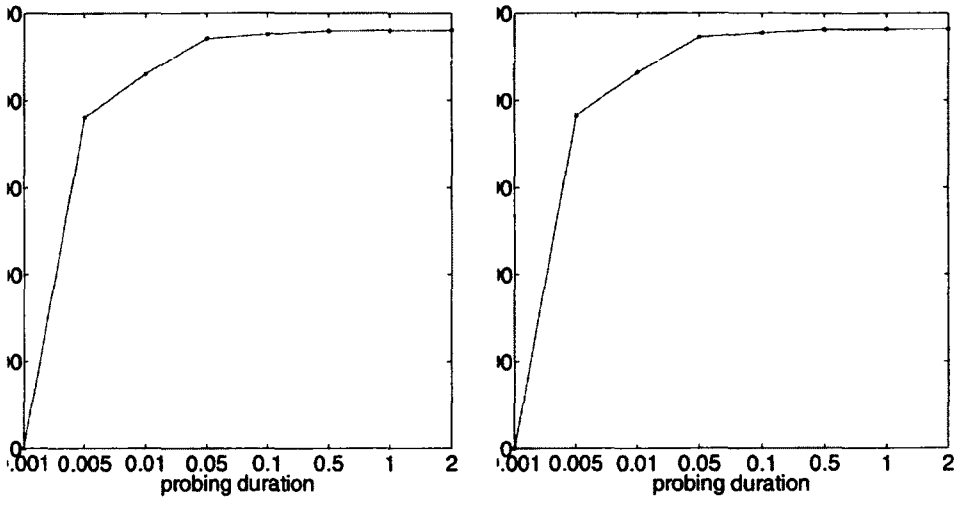

Figure 4.4: Estimation of workloads generated by two traffic patterns following normal distributions. In the left figure, the mean of distribution is 961 with standard deviation of 1, while in the right one, the mean is 964 with standard deviation of 4. X axis is different traffic measurement time from 0.001 seconds to 2 seconds. Y axis is the estimated workloads in packets per second.

4.6.4 Comparison Experiment

We conduct an experiment to compare our association method of SmartAssoc with other practical ones, the Best-throughput and Best-RSSI strategies. Best-throughput here is one special case of the selfish strategy, of which the convergence speed can be bad. It means every client will make an irrevocable association decision to maximize its own throughput. In the experiment, there are three APs consisting of an *extended service set* (ESS) and four wireless LAN clients. Two of the APs (AP-1 and AP-3), whose process capabilities are relatively stronger than the thirds , are set in channels 1 and 11, respectively, and the third one (AP-2) is set in channel 11 as well. Three of the four clients, STA-1, STA-2 and STA-4, are put close to each other. Detailed settings are shown in Table 4.2.

The experiment includes four trials. In each trial, clients came to join the ESS one by one by using the same association strategy. It is reasonable because that, in real world, the time to perform the association operation is statistically much

Table 4.2: Comparison Experiment Settings

APs				
ID	Model		Channel	
AP-1	LinkSys WRT54G		CH 11	
AP-2	D-Link DI-713P		CH 11	
AP3	LinkSys WRT54G		CH 1	
Clients				
ID	Adapter	Pkt payload (byte)	Inter-pkt time(ms)	Transmission rate(Mbps)
STA-1	internal	1000	5	2
STA-2	internal	1000	5	11
STA-3	external	1000	5	11
STA-4	external	1000	5	11

smaller than the inter-arrival time between new users, so the possibility that two clients will want to join the ESS at the same time is low. After joining, the client will generate traffic using the configuration in table 4.2 to the associated AP. A trial was repeated three times, one for each association strategy.

The minimum client throughput for each strategy in the four trials are presented in Figure 4.5. Our algorithm performs better than the other two because it can balance the APs' workloads and reduce the interference among all wireless nodes. The performance of the Best-throughput is unstable, and it also indicates that the selfish strategy cannot compete with ours under similar overhead costs. Meanwhile the Best-RSSI strategy often makes all clients associate with the same AP.

4.6.5 Overhead

All three protocols mentioned above need an AP discovery phase. However, Best-RSSI does not have any other overhead, whereas Best-throughput and our protocol need extra time to evaluate every discovered AP. For each channel, our protocol only spends a small amount of time on the channel measurement, compared with the Best-throughput one. Nevertheless, the Best-throughput protocol requires real de-association and re-association operations, including the IP assignment, before every measurement. These operations consume a lot of time, from 3 *sec* to 8 *sec*.

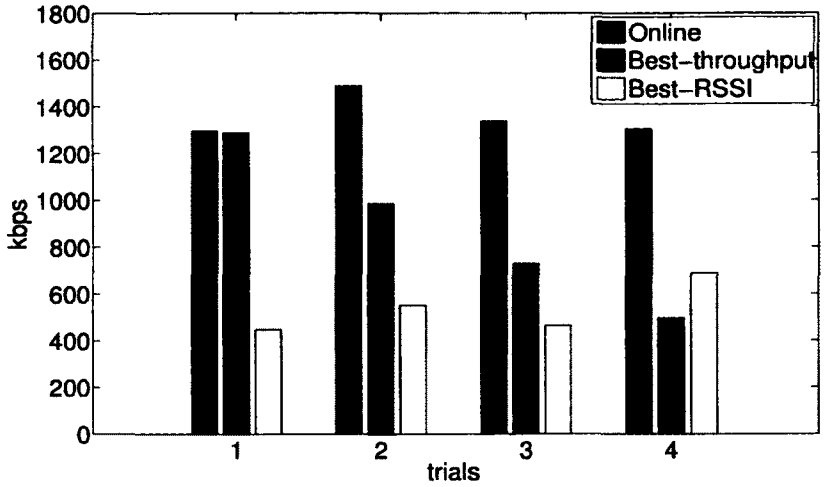

Figure 4.5: Comparison Experiment Results

When the AP number grows, the Best-throughput protocol spends much more time than ours. Therefore, the overhead of the selfish protocol, which is equivalent to multiple runs of the Best-throughput protocol, is even worse. Thus, our protocol is the most efficient one.

4.7 Simulation Result

We use simulations to evaluate our association protocol SmartAssoc on a larger scale with more wireless nodes and various configurations. We use NS2 version 2.33 as our simulator. The multiple channel feature is patched into the NS2 wireless portion following the instructions of [76]. The MAC layer type is 802.11, while the radio propagation model is two-ray-ground. Ad-Hoc routing protocol is disabled since we are focusing on the infrastructure type of wireless LAN. The RTS/CTS mechanism is also disabled. The data traffic for users is a UDP stream with a packet size of 1000 bytes and average inter-arrival time of 1 ms. The transmission rate is set to 11 Mbps. The selected channels include 1, 4, 5, 6 , and 11 for covering the orthogonal and adjacent channel cases. The throughput measurements are

between the wireless nodes.

We implement the following three practical association protocols for comparison.

1. **Online**. This is our proposed online algorithm in SmartAssoc. It is implemented as described in Algorithm 1.

2. **Selfish**. The behavior of *Selfish* is defined in section 4.3. For the same reason that the convergence speed of selfish strategy can be very bad, we demonstratively run the protocol for 5 rounds. In the first round, the clients come to join the wireless LAN one by one. Each client will associate with every AP to measure the UDP throughput and pick the one who is able to offer the highest value. In each of the next four rounds, every client will repeat the above process to adjust its association based on current wireless LAN association topology. Finally, every client will keep its association with the AP it picks in the last round.

3. **Ideal**. This is the globally optimal association solution in terms of maximizing the minimum throughput over all clients. *Ideal* is obtained by enumerating all possible association topologies, given a specific scenario setting only including the location and channel assignment information. In the real world, it is not practical because of its complexity.

Every experiment conducted below consists of many trials. Each trial has its own scenario configuration. The configuration provides the locations of all wireless nodes and the APs' channels. Both of these two information are randomly generated. Every throughput measurement, no matter whether it is for a data stream or probing stream, takes 3 seconds in the simulation.

The first simulation is to study the competitive ratio of *online* compared with *ideal* from the perspective of empirical experiments. The experimental value of the

competitive ratio is a good indicator of the performance gap on average between the *online* and *ideal*. The statistically stable performance is also a concerned issue to the users in the real world. This experiment randomly picked 50 scenarios for testing. For every scenario, the competitive ratio in terms of the minimal client throughput, shown in Figure 4.6, is calculated based on the test result of *online* and *ideal*. The theoretical upper bound is also provided for comparison. The simulation results shows that about 86 percent of competitive ratio is above 0.47, and 70% are quit stable, just around 0.5. The worse competitive ratio among these 50 trials is 0.313, while the theoretical upper bound, computed from $\frac{1}{e \log m}$, is 0.232.

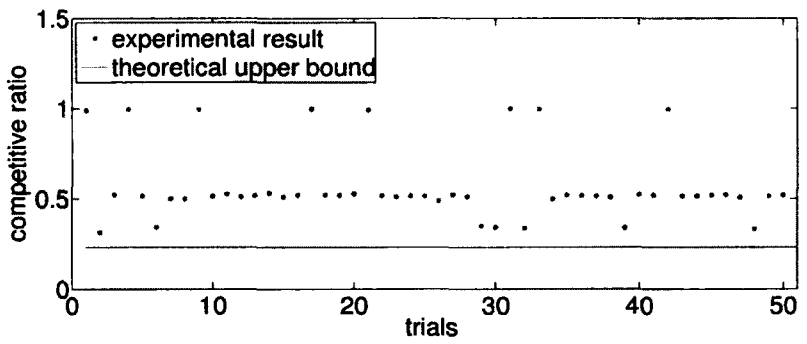

Figure 4.6: Competitive ratio results, w.r.t. minimal client throughput, for 5 clients and 3 APs within 20×20. The simulation repeated 50 times with different configurations.

Next, we conducted a scale-up comparison simulation between *online* and *selfish*, which includes three experiments to show the performance in large scale deployments. In the first experiment, there are 10 clients and 3 APs within a rectangle of 30×30; the second one has 20 clients and 6 APs located in a rectangle of 90×90; 30 clients and 9 APs are involved in the third experiment within a rectangle of 150×150. Each experiment ran 30 trials. For each trial scenario, both our strategy and the selfish strategy were applied for the association processes of all clients on this setting, respectively. After finishing all users' association processes, we measured the UDP traffic throughput for every client and found the minimum, T_{online} for *online* and $T_{selfish}$ for the *selfish* strategy. Then the minimal

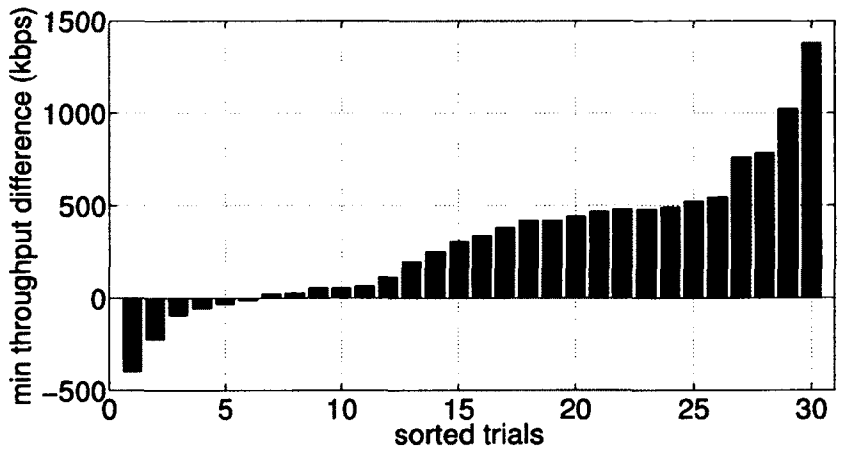

Figure 4.7: Simulation result for 10 clients and 3 APs within 30×30. The simulation repeated 30 times. Each bar is the representative of minimum throughput difference for each trial. The results are sorted in ascending order

client throughput difference, shown in Figures 4.7, 4.8, and 4.9, is calculated by using $diff = T_{online} - T_{selfish}$. These figures show that, even through the selfish protocol is allowed to consume more time, our strategy is more often to perform better in terms of maximizing the minimum client throughput.

In the *online* strategy, since every client only needs to run our association once, the following clients in the future will not affect the behaviors of current associated clients. Meanwhile, for the *selfish* protocol, the unexpected new clients can easily break the current equilibrium into an unstable state, which will interrupt the usage of users. Thus, the *online* is more practical and less-intrusive. From the figures, it is shown that ours can, despite not knowing who will come to join the network, reduce the performance downgrade for the client who has the minimum throughput.

4.8 Conclusion

In this chapter, we consider the problem of AP association in WLAN. We present a theoretical analysis of the performance of two commonly used AP selection protocols, and propose an online algorithm with a provable good competitive ratio. The

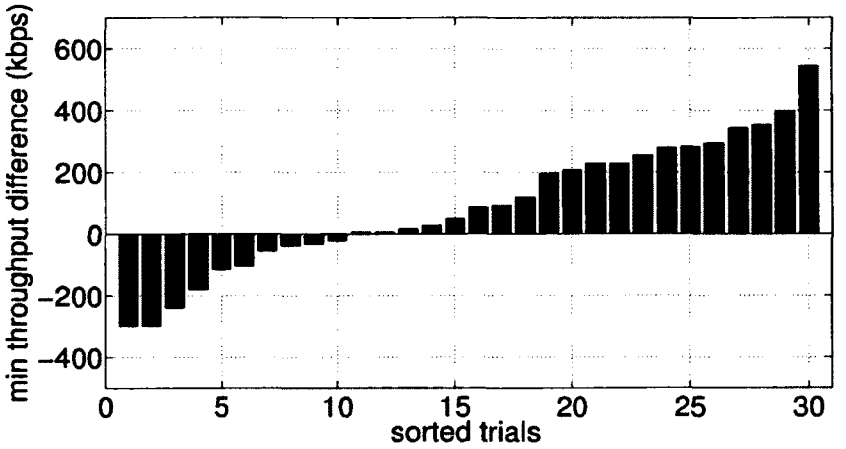

Figure 4.8: Simulation result for 20 clients and 6 APs within 90×90. The simulation repeated 30 times. Each bar is the representative of minimum throughput difference for each trial. The results are sorted in ascending order

association protocol based on this algorithm is implemented on the real testbed in a light-weight way. We evaluated our scheme using a combination of implementation on commodity hardware and extensive simulation. We demonstrate that it is promising that by combining our theoretical understanding and real system implementation experience, a new, practical, and better AP association protocol is possible.

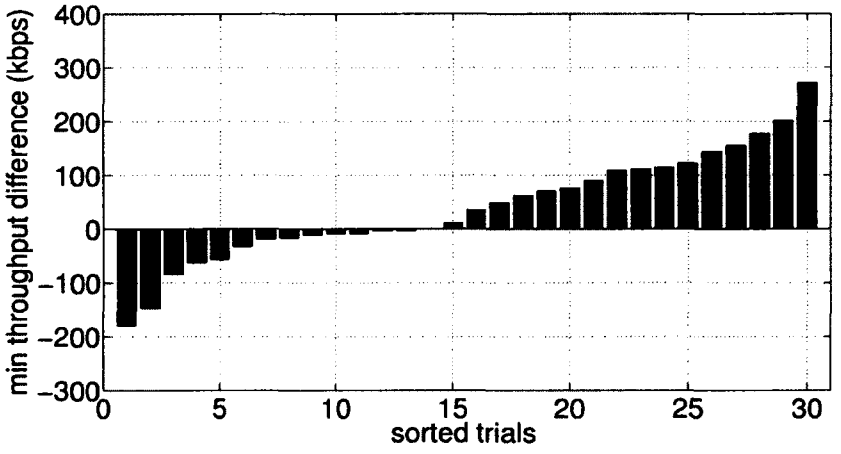

Figure 4.9: Simulation result for 30 clients and 9 APs within 150×150. The simulation repeated 30 times. Each bar is the representative of minimum throughput difference for each trial. The results are sorted in ascending order

5 Implantable Medical Device Security

The rapid advances of bioengineering are introducing a boom of wireless accessible IMDs. Millions of patients experience the benefits from IMDs in regulating heart rhythm, controlling blood pressure, improving hearing, providing visual sight, and so on. In the near future, IMDs are expected to be Internet aware, and become a crucial component in pervasive systems such as smart homes and hospitals, making IMDs' security important. Researchers have identified that IMDs are facing potential security threats which may cause life threatening consequences. Recent investigations on pacemakers [41] revealed security vulnerabilities on existing commercial offerings that allow, among other attacks, a malicious entity to reprogram the IMD. Thus, any vulnerability has to be addressed before further integration of IMDs into an intelligent environment can be realized.

Unlike conventional embedded systems, engineering security into IMDs presents the unique challenge. Security mechanism enforcing protection all the time may lead to troubles when safety tops secure operations of IMDs. To illustrate, consider an ER doctor, who is not recognized as legitimate operator in terms of security, may have to access the IMD to save the patient's life in an emergency situation. Temporary authorization to the doctor is not a reliable solution since the IMD owner in this circumstance may be physically incapable of doing this or remote trusted

authority is not available.

Intuitively, we want a security mechanism resembling an on/off switch to control security protections. The switch can be triggered off in an emergency without assistance from the patient, but in a non-emergency situation, the patient has full control over who has access to his device. (This does open the IMD to attack in an emergency, but life-threatening conditions trump such concerns.) Researchers have advocated pairing the IMD with an external device to provide security for the IMD, where in an emergency, the doctors can simply remove the external device and proceed to interact with the IMD without further hindrance.

There are two challenges when using an external device to protect the IMD. First, the external device and the IMD should have a means of establishing a secret *without prior knowledge*. In other words, we should not pre-deploy any secret inside the IMD. This is to avoid situations where the user is unable to recall the pre-deployed secret and needs to rekey the IMD. The conventional solution is to use a manual switch to ``reset" the IMD, but since the IMD is implanted inside the patient's body, this solution is unsuitable. The second challenge is to have a reliable method to prevent an adversary from convincing the IMD that the external device is absent. Since the IMD is inside the patient's body and the external device is placed outside the body, we have to rely on the wireless communication to relay information. This opens up a possibility for the adversary to jam the channel to create the appearance that the external device is absent. Thus it is crucial to ensure IMDs to correctly distinguish between real emergency and an attack.

In this chapter, we propose IMDGuard, a security scheme for implantable cardiac devices[1], which are implanted to monitor or treat *cardiac* medical conditions. Those IMDs are one widely utilized group of medical devices, and examples include implantable cardioverter-defibrillator, pacemaker, and ECG (electrocardio-

[1]We refer IMDs as implantable cardiac devices in the rest of chapter

gram) sensor. IMDGuard leverages the Guardian, an external wearable device, to coordinate interactions between the IMD and the doctor in such a way that provides the security in a regular condition, and safely allows access in an emergency. The patient's ECG signals are exploited to extract keys shared exclusively between the IMD and Guardian. This key extraction scheme does not need any pre-distributed secret so that it is easy to rekey the IMD when the Guardian is lost or malfunctioning. Moreover, it makes the adversary unable to forge fake Guardians except through physical contacts with the patient. IMDGuard also can effectively prevent the adversary capable of jamming from spoofing the IMD that the Guardian is absent. The adversary's deception will be revealed by collaborations between the IMD and Guardian through the notification mechanism based on the defensive jamming.

The rest of the chapter is as follows. We review the related work in Section 5.1, and the background and problem formulation in Section 5.2. Sections 5.3 and 5.4 detail the IMDGuard scheme including running time protocols and key initialization between IMD and Guardian, and Section 5.5 describes our prototype implementation. We provide evaluation on our scheme in Section 5.6, and conclude in Section 5.7.

5.1 Related work

The increase use of IMDs has motivated researchers to study the security issues on such devices [32, 40, 41]. Their proposed solution while secure, does not address what happens in an emergency situation where the doctors are unable to obtain the necessary keys from the patient.

Later work by [31] explored the concept of safety, and proposed the idea of *fail-open*, a property to physically circumvent the IMD's security protection in an emergency, through the use of an external device. This introduces a new security

threat whereby an adversary may attempt to induce the fail-open state to access the IMD. Our proposed protocol also provides the fail-open property, but differs from [31] in three aspects. First, our design avoids the periodic message broadcasting which consumes considerable battery power and exposes patients to privacy risks. Second, our solution protects the IMD without any assumption on the adversary's transmission capability. Third, our scheme is comprehensive and evaluated on resource constrained embedded systems.

Our solution includes a spoofing attack resistant mechanism related to jamming. Jamming in sensor networks have been studied by [49, 58, 92]. However, such jamming protocols do not consider the features of the IMD, and cannot be directly used in our problem. Other anti-jamming strategies like [83] and Direct-Sequence Spread Spectrum modulation also cannot be applied because of the hardware limitation and the band regulation [15].

Our solution also includes a key extraction algorithm from ECG signals to secure the link between the IMD and the Guardian. The idea of using physiological signals to secure inter-sensor communications was first introduced in [27], and Poon *et al.* [71] put this scheme into practice for ECG and PPG (photoplethysmogram) signals. Inter-pulse intervals (IPIs) of heartbeats are exploited to extract keys in [20]. For 16 consecutive individual IPIs, the ending time in millisecond (ms) of each IPI is calculated, setting 0 as the start time of the first IPI. Then the 7th and 8th digits of the binary representations of the ending times are extracted to form the key. Even though the extracted binary sequences can pass several NIST [77] randomness tests, they are actually not random as what they look. Since the average IPI is about 850 milliseconds, the 7th and 8th digits of the ending time are not random at all. The randomness lies in the lower digits, so does the error. Compared with it, our solution explored a new way to correctly utilize IPIs for extracting randomness.

A faster scheme was proposed by [87] where the coefficients of Fast Fourier

Transform (FFT) on sampled ECG signals are used to extract keys, however, the paper also does not give a rigorous analysis whether input samples contain sufficient entropy to generate a key with required entropy bits, and it evaluates the key after hashing, which is not logically correct. Our key extraction scheme differs from this work in two facets. First, we give rigorous information theoretical study on the randomness of the physiological feature from which the key is extracted. Second, we show that the adversary cannot get any knowledge about the generated keys except he can measure the ECG signals simultaneously without being caught.

5.2 Background and Formulation

In this section, we first show the configuration of IMDGuard, then the adversary model, and finally the approach against the adversary.

5.2.1 IMDGuard Configuration

IMDGuard has three components, the IMD, Guardian, and programmer. The IMD, once implanted, is expected to remain inside the body for an extended period of time. The programmer, as an outside controller, provides doctors an interface to interact with IMD through radio frequency transmission for adjusting running parameters, changing operation modes, or retrieving stored data. Above two are standard wireless programmable medical instruments. The Guardian is a wearable device with more power and computational resources than the IMD. This Guardian works as a proxy for the IMD and performs the authentication on its behalf. Both the IMD and the Guardian are capable of measuring ECG signals. The interactions of these components are illustrated in Fig. 5.1. Link α represents the access control process between the Guardian and the programmer. Link β represents the initial pairing process between the IMD and the Guardian. Link γ represents the secure

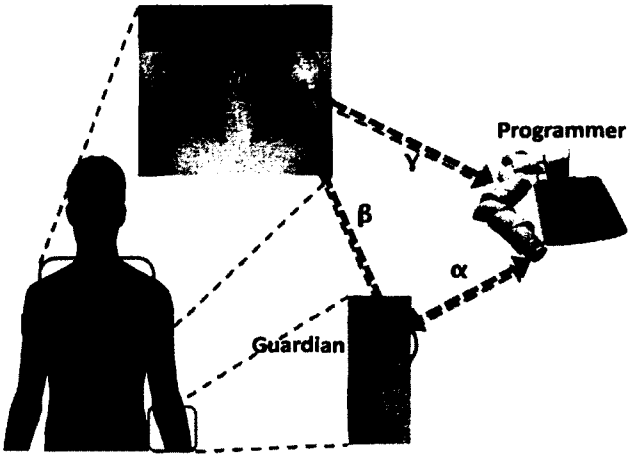

Figure 5.1: Communication interactions in IMDGuard.

communication protected by the key assigned by the Guardian to both the IMD and the programmer.

5.2.2 Adversary Model

We consider an adversary whose goal is trying to program to or retrieve data from the IMD without being caught. The adversary succeeds if he is able to access information from the IMD in the presence of the Guardian. Disruption attacks like denial-of-service are excluded in our adversary model. We assume the adversary cannot physically measure the patient's real-time ECG signals without being detected. We also assume that there is no adversary in an emergency situation. This is reasonable since in such a scenario, the patient with the IMD is likely to be in a secure facility like an ER room in a hospital which can limit the presence of adversaries.

We classify the attack strategy of adversary against IMDGuard into two aspects. The first is when the adversary tries to impersonate the Guardian by deriving the key shared between the IMD and the Guardian from either brute force searching or historic medical records of the patient. The second is when the adversary may

spoof the absence of the Guardian by selectively jamming the messages from it, in order to convince the IMD to disable safety protection and switch to the emergence status.

5.2.3 IMDGuard Overview

In IMDGuard, the Guardian performs two essential functions. First, the Guardian is used to control which mode, *regular* or *emergency*, the IMD should enter. When the patient is wearing the Guardian, the IMD should function under the regular mode. In a regular mode, the programmer requiring to interact with the IMD will first be authenticated by the Guardian, which will then issue the appropriate keys to both the IMD and programmer. When the IMD does not detect the presence of the Guardian, the IMD should enter emergency mode. The advantage is that in an emergency, the doctor will be able to physically remove the Guardian and have unfettered access to the IMD.

Second, the Guardian will authenticate the programmer on behalf of the IMD. This will conserve the IMD's battery by reducing the number of operations performed by the IMD. This also simplifies overall IMD design, since the IMD does not have to maintain cryptographic materials such as asymmetric keys and access control lists.

We assume that the Guardian will always be worn by the patient. It is reasonable since the Guardian can take the form of a watch and the patient can wear it all the time. We also assume that the adversary cannot physically remove the Guardian without the patient being aware of it.

We do not assume the IMD must associate exclusively with one Guardian. Thus before making the Guardian effective, it needs to be initialized by sharing a secret key between the IMD and this Guardian, so that they recognize each other. Moreover, in the extreme case that the Guardian current worn is broken or lost, a new

Guardian can be paired with the IMD easily without the need of retrieving old key or resetting IMD.

To realize this functionality, one key feature of IMDGuard is a secure key establishment scheme based on ECG signals. Both the IMD and the Guardian locally sample the same random source simultaneously, the patient's ECG, and then extract a symmetric key from ECG features after ECG delineation. Unlike the Diffie-Hellman key exchange or wireless based key extractions, this scheme is robust against man-in-the-middle attacks as long as the adversary cannot physically measure real-time ECG signals of the patient.

The other key feature of IMDGuard is the spoofing attack resistant mechanism. If the adversary attempts to persuade the IMD to enter the emergency mode by jamming all messages transmitted from the Guardian, the Guardian still can announce its presence to the IMD by *jamming the IMD's transmission of the challenge message*. The intuition is that the Guardian may hardly block the transmission from the adversary to the IMD, since it has no knowledge about the adversary's hardware and capabilities. Instead, the Guardian can be calibrated to the parameters of its own IMD, and can always successfully jam its IMD's transmissions.

5.3 Protocol Design in IMDGuard

Here, we present the protocols of IMDGuard. We assume that the IMD and Guardian have already paired with a shared secret key after key establishment phase, which is described in the following section. We assume the Guardian has a list of legitimate programmers and their corresponding public keys. This information can securely be installed when in the hospital. Table 5.1 summarizes the notations used.

Table 5.1: Table of notations

G	the Guardian
P	the Programmer
ni_j	the ith nonce generated by j, $j \in \{IMD, G, P\}$
$H(\cdot)$	standard cryptographic hash function, e.g., SHA-1
SK	the shared secret key between the IMD and the Guardian using ECG based key extraction (Section 5.4)
PK_j^+	the public key of j, $j \in \{G, P\}$
PK_j^-	the private key of j, $j \in \{G, P\}$
TK	the temporary symmetric key used for one session
ID	the identification of the IMD

5.3.1 Basic IMD Protocol

The IMD will periodically wake up to determine whether there is any request from the programmer. After the IMD receives a request from the programmer, the IMD will execute Algorithm 2. The IMD will send back its ID, and a random nonce $n1_{IMD}$, which is used as the session identity to resist against the replay attack. Then, the IMD starts a timer T_1 to wait for the Guardian, if present, to notify it to run the *regular condition protocol*. In the case that there is no message from the Guardian when T_1 times out, the IMD will run the *emergency condition protocol*.

Algorithm 2 Basic IMD algorithm

```
Send back to the P, ID and n1_IMD
Start waiting timer T_1
(Guardian, if present, will execute authentication protocol (Fig. 5.2) during T_1)
while T_1 time out == FALSE do
    if receive valid message from G then
        Regular condition, not an emergency.
        Execute Regular Condition Algorithm (Section 5.3.3)
    end if
end while
Possible emergency condition.
Execute Emergency Condition Algorithm (Section 5.3.4)
```

5.3.2 Guardian Authenticating Programmer

In Step 2 of Algorithm 2, the Guardian will authenticate the programmer within the time period T_1. The authentication protocol is shown in Fig. 5.2.

A random nonce $n1_G$ is signed by the programmer, and sent back to the Guardian along with another random nonce $n1_P$. The signature of the programmer is verified by the Guardian. If it is not valid, the Guardian will inform the IMD to deny the session request through the message $\{NO, n1_{IMD}\}_{SK}$ (it will also inform the programmer the authentication is failed and session is denied). If valid, the Guardian will assign a temporary session key TK to both the IMD and the programmer for the secure communication. We let the IMD use symmetric keys when communicating with the programmer and Guardian to reduce the computational load on the IMD. The Guardian and programmer use public keys to authenticate each other for better key management.

G : Overhears msg in step 1 of Algorithm 2 (5.1)

$G \rightarrow P : n1_G$ (5.2)

$G \leftarrow P : \{n1_G\}_{PK_P^-}, n1_P$ (5.3)

G : Verify the signature with PK_P^+ (5.4)

If incorrect, then

Deny the request and inform IMD and P

Authentication phase completed

If correct, then

Accept the request, and continue step(5)

$G \rightarrow IMD : \{YES, TK, n1_{IMD}\}_{SK}$ (5.5)

$G \rightarrow P : \{YES, TK, n1_P\}_{PK_P^+}$ (5.6)

Figure 5.2: Guardian authentication Programmer

$G \rightarrow IMD : \{R\}_{SK}$ (5.1)

IMD : Decrypt to derive R (5.2)

If $R == \{NO, n1_{IMD}\}$, then

Deny the request from P, and go to sleep

If $R == \{YES, TK, n1_{IMD}\}$, then

Accept req from P, and continue step(3)

$P \rightarrow IMD : \{command\}_{TK}$ (5.3)

$P \leftarrow IMD : \{response\}_{TK}$ (5.4)

Figure 5.3: IMD regular condition protocol

$P \leftarrow IMD : n2_{IMD}$ (5.1)

IMD : Wait for time T_2 (5.2)

$P \leftarrow IMD : n3_{IMD}$ (5.3)

$P \rightarrow IMD : ans$ (5.4)

IMD : Check ans (5.5)

if $ans! = H(n2_{IMD} \oplus n3_{IMD})$, then

Deny the request from P, and go to sleep

if $ans == H(n2_{IMD} \oplus n3_{IMD})$, then

Accept req from P, and continue step(6)

$P \rightarrow IMD : command$ (5.6)

$P \leftarrow IMD : response$ (5.7)

Figure 5.4: IMD emergency condition protocol

5.3.3 Regular Condition Protocol

When the IMD enters the regular condition (Step 6 in Algorithm 2), it will execute the protocol shown in Fig. 5.3. After decrypting the message R, the IMD will deny access if it receives a NO message. Otherwise, a YES message indicates that the programmer has been authenticated, and the session key for Steps 3 and 4 is TK.

5.3.4 Emergency Condition Protocol

When the IMD enters the emergency mode, it will execute the protocol in Fig. 5.4, and send the first portion of the challenge, a random nonce $n2_{IMD}$. After waiting T_2 time, the IMD sends the second portion of the challenge, $n3_{IMD}$. Assuming that the Guardian has been physically removed, the programmer will transmit back the correct answer to the challenge, $H(n2_{IMD} \oplus n3_{IMD})$. If the Guardian is present, the programmer will be unable to return the correct answer. We explain how the Guardian disrupts this in the next subsection.

5.3.5 Spoofing Attack Resistant Protocol

Here, we show how our protocol is resilient to adversary's spoofing attack based on jamming. The adversary can attempt to jam the communications between the IMD and Guardian to induce the IMD to enter the emergency mode in Step 7 of Algorithm 2. In other words, the adversary will jam the channel for length of time period T_1. Since the IMD does not receive any response from the Guardian, the IMD will proceed to execute the emergency condition algorithm. For this scenario,

G : Overhear the msg in Step 1 of Fig. 5.4	(5.1)
G : Jam the msg in Step 3 of Fig 5.4	(5.2)
G : Raise a warning alarm if Step 1 occurs frequently	(5.3)

Figure 5.5: The notification mechanism of the Guardian

the Guardian function, the defensive jamming described in Fig. 5.5, is triggered to block this session. When the Guardian hears the first portion of the challenge message (Fig. 5.4 Step 1) sent by the IMD to the programmer, the Guardian will realize that the communication link between the IMD and itself is blocked by an adversary with high probability. Then the Guardian will jam the second portion of challenge

message from the IMD (Fig. 5.4 Step 3). This operation is feasible in practice based on the Guardian's loose synchronization through the message in Step 1 of Fig. 5.4. In other words, the Guardian will be aware that there is another message, which is the jamming target, going to send in T_2 time later. This information can help the Guardian to jam the target message with less effort.

There are two advantages in letting the Guardian to jam the IMD's message instead of the adversary's message. First, the adversary's hardware may be much more powerful than the Guardian, making it difficult to calibrate the Guardian's broadcast strength needed to successfully jam the adversary's signal. Second, the the Guardian can time exactly when to be jamming since it is aware when the IMD will begin broadcast. This conserves the Guardian's power by reducing jamming period.

5.4 Key Establishment in IMDGuard

In the previous section, we assume there is a secret key already shared between the IMD and Guardian to secure their communication. However, this key establishment is challenging if the IMD and Guardian do not share any secret beforehand. In this section, we introduce a secure key extraction scheme based on the ECG delineation to establish a symmetric secret key bonding the IMD and Guardian together, making adversary impossible to forge the Guardian.

5.4.1 ECG Delineation

We conduct the ECG delineation with the wavelet-based algorithms mentioned in [54, 57]. Fig. 5.6 shows an example result of our wavelet transform based delineation. Using the information of local maxima, minima and zero crossings at different scales in the wavelet transform, the algorithm is able to detect all the sig-

nificant points of ECG in a heart beat cycle, first the R peak, then Q peak and S peak, followed by T wave and P wave.

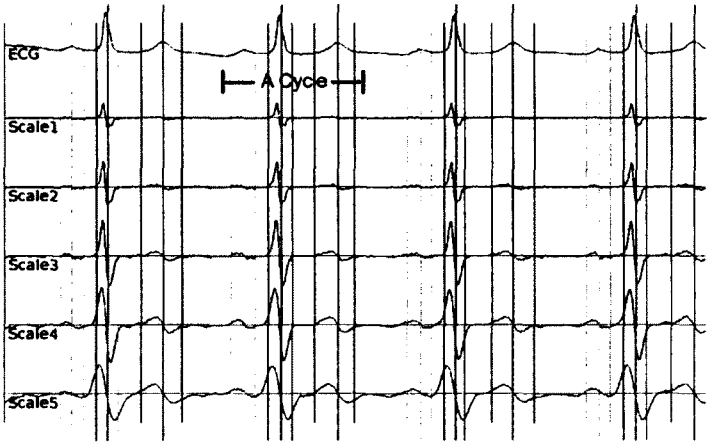

Figure 5.6: Wavelet transform of ECG waves at the first five scales. The first panel is the ECG signal, the other five, from top to the bottom, are the corresponding wavelet transforms from scale 2^1 to scale 2^5.

As shown in Fig. 5.6, in each heart beat cycle, the three blue lines in the middle denote the onset, R peak and offset of QRS complex respectively. The three cyan lines on the left denote the onset, P peak and offset of P wave respectively. The three red lines on the right denote the onset, T peak and offset of T wave respectively.

We implement the algorithms with TinyOS 2.1, with about 1200 lines of code. The high accuracy is achieved to reduce the mismatch rate of IPIs, making the following key extraction efficient.

5.4.2 An ECG Feature for Key Extraction

Given the two ECG measurements that are taken at different parts of the human body, we want to extract a symmetric key from them after the delineation. As a fundamental requirement, the key much be *random*. Thus, the key must be extracted from an ECG feature such that: (1) the feature itself is random; and (2) the feature has common places for both the IMD and Guardian.

Table 5.2: The quality of randomness of each digit.

i	P_{00}(%)	P_{01}	P_{10}	P_{11}
1	29.2	24.4	24.4	21.9
2	28.9	24.3	24.4	22.4
3	25.2	24.6	24.7	25.5
4	27.9	25.6	25.6	20.9
5	57.5	18.8	18.8	4.9
6	2.5	13.2	13.2	71.1
7	99.1	0.4	0.4	0.0
8	99.7	0.1	0.1	0.0

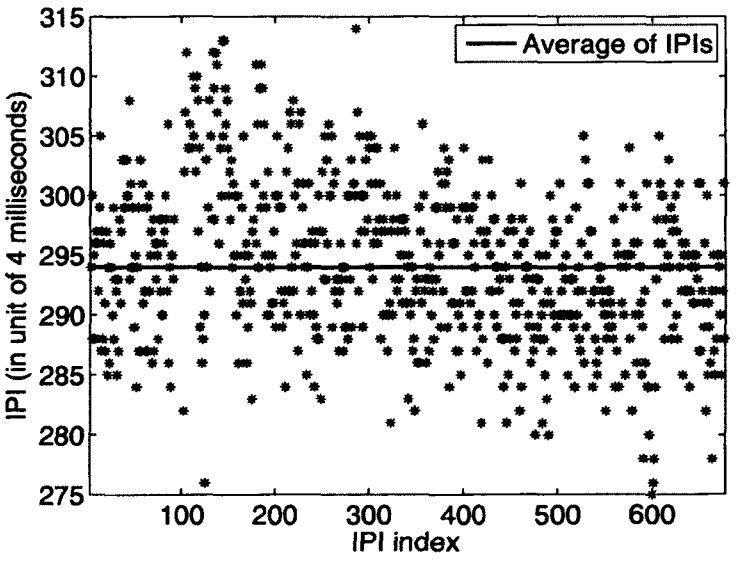

Figure 5.7: The fluctuation of IPIs against the average, which is 294 in unit of 4 ms (250Hz sampling rate).

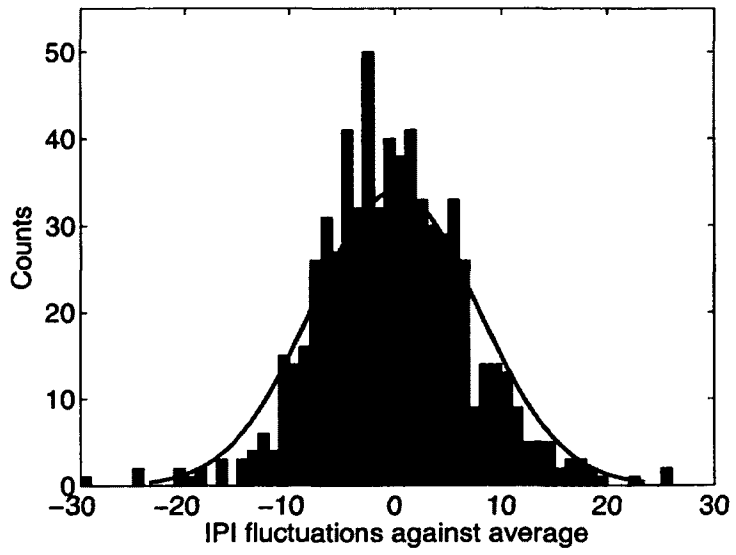

Figure 5.8: The normal distribution fitting to the fluctuation of IPIs against the average, with $\mu = 0$ and $\sigma^2 = 61$.

After the ECG delineating, we have the timing information of all the ECG significant features, namely P wave, QRS complex and T wave, for every heart beat cycle. Since the ECG signals are periodic, to ensure the randomness, we cannot directly use all the delineation points at the same time. Once a feature is chosen, other features in the same heart beat cycle are not totally random any more. For instance, if we know the position of the R peak, we can easily guess into what ranges the Q peak, S peak, T peak and P peak in the same cycle fall. Even for features in different cycles, the positions of features are not totally random. For example, given the position of R peak in one heart beat cycle, that of the following R peak will fall into a small range because the common inter-pulse interval (IPI) is known. (For adults, the common IPI is about 850 ms)

We will use the information of R peaks, which is most salient, to extract keys. Given a consecutive sequence of R peaks, IPIs are obtained by calculating the difference in time of the two consecutive R peaks. Suppose R_i denoting the time of the ith R peak, then $IPI_i = R_{i+1} - R_i$. Since the average value of IPI is quite

known, we have to exclude the average value when extracting the key. First we empirically estimate how many random bits can be extracted from each IPI value. We convert IPI values to binary representations, then examine the randomness of each digit of the binary representations. It is clear that the random bits lie in the low digits. For the ith digit, we count the number of samples for the following cases:

(1) n_{00}: if the ith digit of sample j is 0, and so is that of sample $j+1$, then increment n_{00} by 1;

(2) n_{01}: if the ith digit of sample j is 0, and that of sample $j+1$ is 1, then increment n_{01} by 1;

(3) n_{10}: if the ith digit of sample j is 1, and that of sample $j+1$ is 0, then increment n_{10} by 1;

(4) n_{11}: if the ith digit of sample j is 1, and so is that of sample $j+1$, then increment n_{11} by 1;

where $1 \leq j < n$, n is the total number of consecutive IPI samples.

We then calculate the four possibilities $P_{lk} = n_{lk}/(n-1)$, where $lk \in \{00, 01, 10, 11\}$. If the ith digit is random and independent, all the four possibilities should be around 25%. We list the possibilities for the lower 8 digits of IPI samples in Table 5.2. We set a threshold of 5%. As shown in the table, the last 4 digits are random, while the 5th digit is not. For the 5th digit, P_{00} is more than 50% while P_{11} is less than 5%. We also calculate the entropy of the fluctuations directly from the original data, which is around 5. Therefore, we can confidently extract 4 bits from each IPI.

5.4.3 Quantization

This subsection will show how to extract 4 random bits from each IPI sample. We cannot use the last 4 digits of IPI's binary representations directly, because the slight difference between the data at both sides may cause big differences in the last 4 digits of the binary representations, leading the mismatch rate to as high

as 20%. Actually, the lower 4 digits are the fluctuations of IPIs. Fig. 5.7 shows the IPI fluctuations against the average value. In terms of entropy, the fluctuations don't lose any entropy of IPI samples. Though the average IPI is quite stable, the difference between individual IPI and the average value is unpredictable. It can be positive, negative or zero. And the value of the fluctuation is quite random in a certain range. So the fluctuations can be chosen as the basis to extract the key.

In the perspective of statistics, the fluctuations shape into a normal distribution. Fig. 5.8 shows the histogram of the samples in Fig. 5.7, with a normal distribution fitting. The fitted normal distribution also results in an entropy $\frac{1}{2}log(2\pi e\delta^2) \cong 5$, which is within the range of that resulted from the original data. And we have proved that the last 4 digits are random, which implies that the real entropy is at least 4. In this sense, the distribution of the fluctuations is indeed a normal distribution, or at least close to.

IMDGuard provides the following algorithm to do the quantization. This algorithm is based on the assumption that the fluctuations form a normal distribution. For a normal distribution $N(\mu, \sigma)$, given μ and σ, we can divide the probability density function into 16 consecutive sections such that, in each section, the cumulative possibility density is $1/16$. The domain of each section can be denoted by a function of σ and μ, as shown in Table 5.3. If μ or any starting/ending point of any domain is an integer, we split samples with that value into two portions, with each going into one of the nearby domain. The splitting depends on the sample index. The samples with odd index form one portion, and those with even index form the other. Note that σ is big enough such that every domain contains at least one integer, since the entropy indicates that σ is not small. The purpose of the division is to roughly but not precisely equalize the number of samples in each domain, making the quantization unpredictable. The 16 domains are one-to-one mapping to the 4-bit gray codes.

Table 5.3: Normal distribution divided into 16 equal sections.

	Domain		Domain
1	$(-\infty, \mu - 1.534\sigma)$	9	$(\mu, \mu + 0.157\sigma)$
2	$(\mu - 1.534\sigma, \mu - 1.151\sigma)$	10	$(\mu + 0.157\sigma, \mu + 0.319\sigma)$
3	$(\mu - 1.151\sigma, \mu - 0.887\sigma)$	11	$(\mu + 0.319\sigma, \mu + 0.489\sigma)$
4	$(\mu - 0.887\sigma, \mu - 0.675\sigma)$	12	$(\mu + 0.489\sigma, \mu + 0.675\sigma)$
5	$(\mu - 0.675\sigma, \mu - 0.489\sigma)$	13	$(\mu + 0.675\sigma, \mu + 0.887\sigma)$
6	$(\mu - 0.489\sigma, \mu - 0.319\sigma)$	14	$(\mu + 0.887\sigma, \mu + 1.151\sigma)$
7	$(\mu - 0.319\sigma, \mu - 0.157\sigma)$	15	$(\mu + 1.151\sigma, \mu + 1.534\sigma)$
8	$(\mu - 0.157\sigma, \mu)$	16	$(\mu + 1.534\sigma, +\infty)$

Since the IMD is measuring the ECG signals all the time, it is able to calculate σ and μ for a long period, say 15 minutes, and store it. During key generation, the IMD can send these parameters to the Guardian. This process doesn't leak any information about the key, since the adversary still doesn't know which sample is in which domain and how many samples are in each domain. The quantization is shown in Algorithm 3.

Algorithm 3 Quantization Algorithm

```
Input: n consecutive IPIs from ECG, IPI_1, IPI_2, ..., IPI_n.
Output: 4n bit binary string.
Obtain parameters μ and σ
Calculate 16 domains based on Table 5.3: D_1, D_2, ..., D_16
Number the 4-bit gray codes: G_1, G_2, ..., G_16
Output = φ
for i ← 1 to n do
  for j ← 1 to 16 do
    if IPI_i falls into D_j then
      Output = Concatenate(Output,G_j)
    end if
  end for
end for
```

5.4.4 Reconciliation

Due to the high accuracy of the ECG delineation, the two binary strings quantized respectively by the IMD and Guardian have a low mismatch rate. For two 4-bit

blocks corresponding to the same heart beat cycle on both sides, one bit is different in most cases if there is a mismatch. In a very few cases, there are two bits different. There is no case that 3 or 4 bits are different. Based on these observations, we design a 2-round reconciliation algorithm. It carries out Round 2 only if Round 1 fails.

Round 1: For each IPI, both the IMD and Guardian get a 4-bit block. Both sides calculate the parity of its own block and exchange this information. If the parities are different, the block is discarded. Otherwise, each side extracts the first 3 bits of the block; the 4th bit is discarded because the parity leaks one bit information. This process continues until both sides get 129 bits. The IMD then hashes it with SHA-1 hash function and sends the hash value to the Guardian. The Guardian compares this hash value with its own, and notifies the IMD. If the two hash values match, the algorithm terminates. Otherwise, Round 2 will be carried out.

Round 2: For the 43 IPIs chosen in Round 1, both the IMD and Guardian calculate the parity of the last 2 bits of each 4 bit block, and exchange this information. Again those blocks whose parities don't match are discarded. For the blocks left, both sides extract the 2nd and 3rd bits; the first bit is discarded since the second parity also leaks one bit information. Obviously, the length of the key is less than 128. Then both sides continue to analyze the following IPIs. At this time, they check two parities at the same time and extract 2 bits from each block which passes the parity check. The process continues until both sides get 128 bits.

5.5 Prototype Implementation

A challenge involving IMD experiments is the difficulty in obtaining source codes and open platforms from commercial vendors. In our prototype system, we choose the TelosB with TinyOS 2.1, an open research platform of the resource constrained

embedded system as a replacement of the IMD. The related details are described below.

Transmitter Comparison: The TelosB utilizes the CC2420 transmitter for wireless communication. The CC2420 is comparable to the typical Medical Implant Communication Service (MICS) radio like ZL70101 [14] used in IMDs. They both are low power radio devices with similar amount of power consumption during transmission. The ZL70101 expends 5 mA, while 8.5 mA is achievable for the CC2420. Besides, both of them share other common features such as multiple channel and duty-cycle support. The difference between CC2420 and ZL70101 is that MICS radio operates lower frequency band between 402-405 MHz because of the reasonable signal propagation characteristics in the human body. This has no impact on our evaluation since our implementation does not rely on the frequency or number of available channels.

Code size: The code size of each component after compilation is shown in Table 5.4. ECC [88] is the Elliptic Curve Cryptography we develop to provide public key scheme between the programmer and the Guardian. For reference, a typical IMD produced in 2002 is able to contain 2MB memory [31].

Table 5.4: Code size of our prototype implementation

Module	ROM(*bytes*)	RAM (*bytes*)
IMD	20656	1056
Programmer	20754	1060
Guardian	20614	1050
ECC	42190	1931
Key Extraction	10078	887
ECG Delineation	18720	9652

5.6 Evaluation of IMDGuard

The performance evaluation of IMDGuard is divided into three portions. First, we comprehensively assess the quality of the key extracted from ECG signals. Then we conduct a series of experiments on the effectiveness of defensing adversary's spoofing attacks. Finally, we present the efficiency of each critical components in our implementation.

5.6.1 Key Establishment

In this section, we will evaluate the key generated according to the algorithms in Section 5.4. The ECG signals are from PhysioBank database (http://www.physionet.org/physio We will address three important characteristics of the key: (1) temporal variance; (2) efficiency; (3) randomness.

5.6.1.1 Temporal Variance

Given a 128-bit key generated by the IMD and Guardian, we want to know whether the adversary can get any help if he can access historic/future records of ECG signals of the patient. Metric used is the hamming distance between the key and any other 128 bit random string before or after it. The hamming distance between two binary strings of equal length is the number of positions at which the corresponding symbols are different. Given two random strings, if they are independent, the possibility of having hamming distance k follows a binomial distribution, which is $P(k) = \binom{n}{k} p^k (1-p)^{n-k}$, where $n = 128$ and $p = 1/2$. And the mean value of k, a.k.a expected value, is $E(k) = np = 1/2 \times 128 = 64$.

We examine these hamming distances. There is no signal value equal to 0 or 128, and all the points lie between 45 and 85. The closer to the mean hamming distance, which is 64, the denser the points. We plot the possibility of the hamming

distances, as shown in Fig. 5.9. As we can see, the measured data matches very well with the theoretical binomial distribution with $n = 128$ and $p = 1/2$. From the statistical prospective, the 128 bit key generated by our scheme does not relate to the historic ECG or future ECG signals. Thus, even the adversary gets historic or future ECG data of the patient, he cannot get any help from it. This also indicates the randomness of the 128 bit key.

We also conduct the same evaluation between keys generated from ECG signals from different persons, and we get the same result as expected. The historic/future record of the same person does not help the adversary, neither does that of other people. [87] did similar evaluation about their scheme. However, they did so after hashing an identical string between two parties with a one-way hash function. Though they got similar results, their results could not prove what they claimed. Hashing will make a string random, no matter the original string is random or not.

5.6.1.2 Efficiency

In the reconciliation phase, there are two rounds. In the first round, 3 random bits are extracted from each IPI. So it needs 43 IPI to get a 128 bit key. If the first round fails, the algorithm will carry out the second round, extracting 2 bits from each IPI. In this case, it needs 21 more IPIs besides the 43 IPIs in the first round. In 88% cases, the first round succeeds. The second round succeeds all the time, at least we did not find a single failure in all our traces. Thus, on average, it needs 45.5 IPIs, without counting the IPI discarded. During Round 1, about 25% samples are discarded, and during Round 2, only 0.3% samples are discarded. Taking into account the samples discarded, it needs 61 IPIs, corresponding to 45 seconds or so, to generate a key successfully.

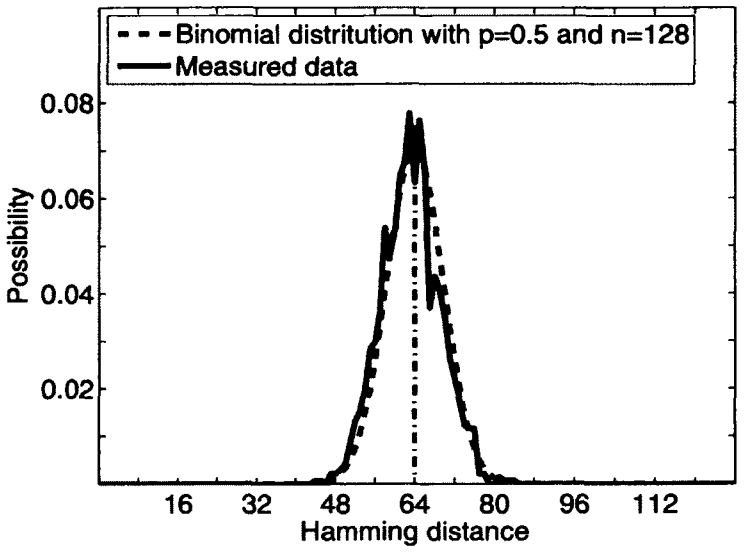

Figure 5.9: The hamming distances between the 128 bit stream from current ECG signal and those from historic records fit into a binomial distribution with $n = 128$ and $p = 1/2$.

5.6.1.3 Randomness

To evaluate randomness of the generated bit stream employed as secret keys, we run the randomness tests in the NIST test suite [77]. There are totally 15 different statistical tests, and we run 9 of them. The other 6 require a very long bit stream that we cannot generate from PhysioBank database. Our bit stream passes all the 9 tests, showing a good quality of randomness.

Evaluations show that even the patient's ECG records cannot help the adversary to predict the key shared between the IMD and Guardian, unless he physically measure patient's ECG signals simultaneously during the key establishment. However it is impossible for the adversary to physically measure patient's ECG without the patient being aware of it. This key establishment is robust to man-in-the-middle attack. If a symmetric key is successfully established, then the Guardian must be legitimate.

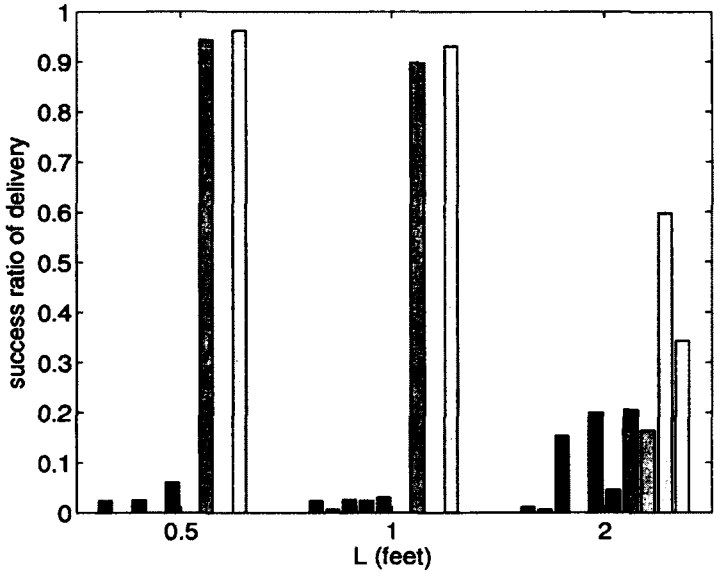

Figure 5.10: Results for the effectiveness of Guardian's jamming when targeting at malicious programmer. X-axis represents the distance between the IMD and Guardian, and Y-axis is the success ratio of delivered messages. For each distance, ten trials were conducted.

5.6.2 Jamming Related Experiments

There are two jamming related experiments. First, we experimentally validate our decision to jam the IMD's communication instead of the adversary's messages. Second, we examines our defensive jamming method using different settings in terms of the power level and the distance.

We let three TelosB motes to act as the IMD, the Guardian and the malicious programmer (adversary). To concentrate on the jamming performance of the Guardian, these motes are installed with the simplified IMDGuard which will be described in each experiment below, as well as the carrier sensing and random backoff on motes are disabled. All the experiments are taken on a large office table in an indoor environment.

Experiment 1: We vary the distance between the IMD and Guardian from 0.5 feet to 2 feet, and set the malicious programmer 11 feet away from the center point

of these two devices. The transmission power of the Guardian is configured to be -15 dBm. This is 10 dB higher than the power of the IMD but is 10 dB lower than the malicious programmer's power. The transmission interval, i.e. the inter arrival time between any two messages, is 20 ms. We then let the adversary send messages to the IMD, while the Guardian is jamming. After the transmission is over, we determine the ratio of messages successfully received by the IMD. The results are shown in Fig. 5.10. As we can see, messages toward the IMD are able to escape from being jammed with an uncertain probability, low in some cases but high in others. This observation indicates that jamming the adversary's transmission does not work in practice since our malicious programmer settings, such as the relative power strength (10 dB) and location (11 feet), or even more rigorous conditions, can be achieved by an adversary.

We then repeat the experiment again, this time letting the Guardian jam the IMD's transmission. The Guardian is able to *successfully jam all the messages.* This approach is more reliable and effective than jamming the adversary. We omit the figure for the results. The success of defensive jamming is due to the fact that the Guardian is aware of all of the IMD's settings, and that the Guardian is more powerful than the IMD by design.

Experiment 2: In this experiment, the Guardian jams the message the IMD sends to the malicious programmer in the same way as above experiment but under various settings. The distance between the IMD and the malicious programmer is fixed at 1 foot, which is considered as the closest position the malicious could have without being detected by the patient. The Guardian is placed away , from 1 foot to 7 feet, from the center point of IMD and programmer at each different power level. The successful delivery ratios of all transmitted messages in every condition are recorded in Fig. 5.11. It is evident that, as long as the Guardian is close enough, e.g. within 2 feet to the IMD, the transmission from the IMD toward the malicious

programmer is totally blocked even in the extreme testing case that Guardian's jamming power is *20 dB less than the IMD's*. This observation is important because the IMD usually is fallen into this distance range if the Guardian is worn by the patient.

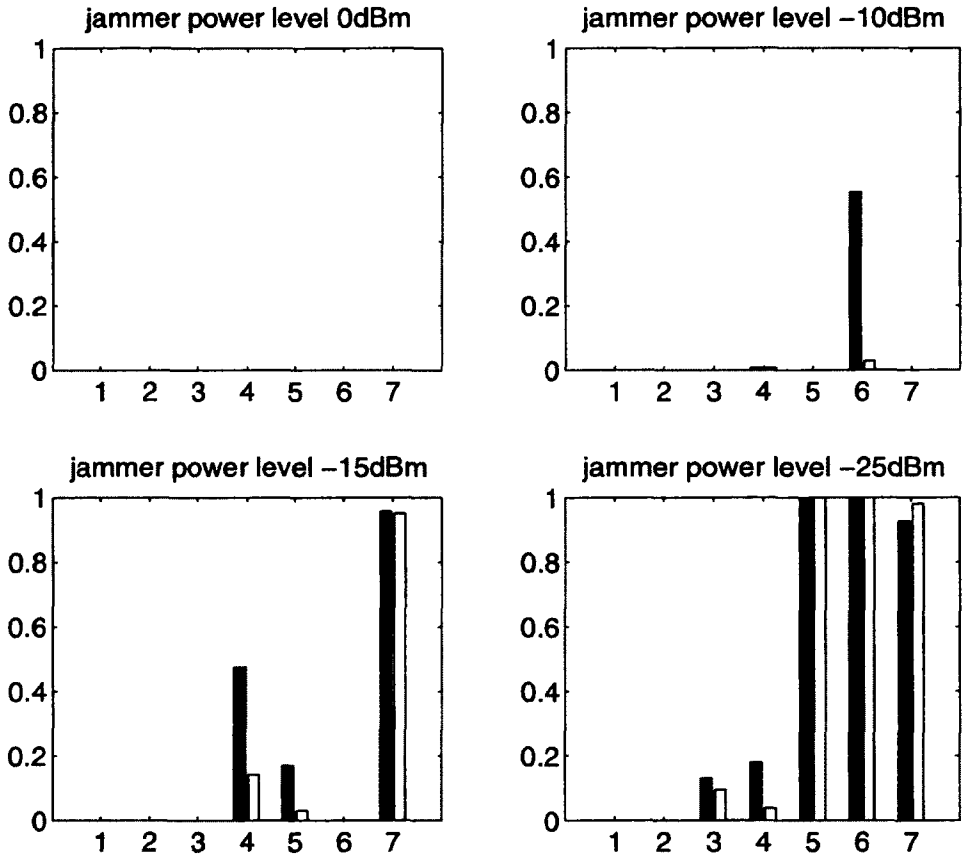

Figure 5.11: Results of the effectiveness of Guardian's defensive jamming(acted as jammer) when targeting at the IMD. X-axis means the distance between the Guardian and center point of the IMD and malicious programmer in feet, and Y-axis is the success ratio of delivery. The transmission power of the IMD is set to be -5 dBm during the whole experiment. For each distance, two trials were conducted.

5.6.3 Overhead Evaluation

Cryptographic overhead: We write a testing program on TelosB to record the average timing of ECC-based encryption/decryption, SHA-1 hash, Advanced Encryption Standard (AES) for a 20 byte message, and data is shown in Table 5.5.

Table 5.5: Security Timing Information

Encryption	Decryption	SHA-1 Hash	AES
3.3 s	1.7 s	4 ms	1 ms

Communication and Operation Overhead: The timing information for the critical operations under different scenarios is provided in the Table 5.6. In an authentication case, on average the Guardian takes 3821 ms to authentication a programmer in total. It is broken down to (1) 1550 ms for programmer to generate a signature of the given challenge (20 byte random data), (2) 2221 ms for Guardian to verify this signature, and (3) 50 ms for other communication overhead. When the Guardian is not present, the process of emergency condition (Fig 5.4) costs roughly 512+14=526 ms before the IMD accepts the programmer. If the defensive jamming occurs, the session will be denied by the IMD in about 1501 ms since receiving the request.

Table 5.6: Prototype Timing Information

Overhead in Time (ms)		
Situation	Operation	Overhead
Authentication	Signing(20bytes) Verification(20bytes) Others	1550 2221 50
Guardian Removed	Challenge Transfer Others	512 14
Guardian Jamming	Session Deny	1501

5.7 Conclusion

In this chapter, we propose IMDGuard, a comprehensive security scheme for protecting implantable cardiac devices in terms of both security and reliability. Prototype of IMDGuard is implemented to demonstrate its functionality of securing IMDs in practice.

6 Conclusion

More and more people rely on pervasive computing systems, including smartphones, body area sensors, laptops and so on. The performance of these systems directly influences user experience, and some of their drawbacks may cause serious outcomes in the real world. Among all the existing problems, security and energy efficiency are the most critical because of following reasons. First, personal computing devices consume a lot of energy in order to complete desired heavy-duty tasks. The lifetime of a battery is reduced a lot because of this and users are not satisfied. Second, many personal mobile devices access a lot of valuable user data under risky circumstances, but they are not protected by comprehensive security schemes because of their limited resources. These two serious problems make people hesitant to adopt and further depend on pervasive computing systems.

There are four projects involved in our research. We introduce a new way to measure the power consumption of smartphones without any external hardware assistance. This enables self-constructive modeling for every individual phone. As such generated power models are accurate and can adapt to any system configuration changes in real time. Based on this technique, we improve the energy efficiency of smartphone applications, taking the email sync over the cellular network as one example. The findings of our study on email sync can guide application developers to improve the energy efficiency of their codes. Additionally, we are the first to investigate the energy consumption of smartphones in the con-

nected standby mode. We also study the network performance of the wireless LAN, which is another popular communication approach employed on pervasive computing systems. Proposed access point selection algorithm can help to save power and secure the communication. Lastly, we propose a new security scheme for IMDs. An external device is introduced to act as a security proxy for the IMD inside the patient's body. The advantage of this design is that the patient's safety is still effectively protected in unexpected situations. In order to apply this scheme in the real world, we solve two challenging problems, secure key pairing without any prior knowledge and defense against powerful spoofing attacks.

From our studies we also derive some valuable knowledge that may benefit the general research area of pervasive computing. First, our new power modeling method provides the necessary tool and foundation for energy issues on all battery-powered devices. Second, as the first investigation of the sleeping mode on smartphones, our findings are helpful to research that explores comprehensive energy solutions for pervasive computing systems. Third, our solution for IMD security can be the framework to manage and protect various similar devices as well. We hope our research experience can help shed light on intriguing studies in the future.

In summary, we improve the energy efficiency of power-hungry pervasive computing systems and extend their battery lifetime; we also enhance the security mechanism of other resource-limited personal mobile devices in order to protect valuable and sensitive information. Our research efforts advance existing pervasive computing systems and can help design the next generation of pervasive computing systems.

In the future, we will continue to work on the security and energy problems of smartphones. The specifications of smartphones keep changing, e.g., from single core to multiple cores. Complicated hardware means simple power models are

unable to capture all consumption variations. We would therefore like to research these emerging hardware and provide more accurate estimations of power consumption. Additionally, we want to enable the ``Bring-Your-Own-Devices" paradigm by enforcing strong isolation of system activities in real time.

References

[1] 3gpp release 8. http://www.3gpp.org/Release-8.

[2] Antutu battery saver. https://play.google.com/store/apps/details?id=com.antutu.powersaver&feature=search_result#?t=W251bGwsMSwxLDEsImNvbS5hbnR1dHUucG93ZXJzYXZlciJd.

[3] Apple finally acknowledges iphone's exchange support. http://www.zdnet.com/blog/microsoft/apple-finally-acknowledges-iphones-exchange-support/1246.

[4] Battery performance characteristics. http://www.mpoweruk.com/performance.htm.

[5] Click. http://read.cs.ucla.edu/click/.

[6] Google adopts microsoft sync protocol. http://www.windowsfordevices.com/c/a/News/Google-adopts-Microsoft-sync-protocol/.

[7] GPS status. https://play.google.com/store/apps/details?\id=com.eclipsim.gpsstatus2&hl=en.

[8] How smartphones are bogging down some wireless carriers. http://arstechnica.com/gadgets/2010/02/how-smartphones-are-bogging-down-some-wireless-carriers/.

[9] Madwifi. http://madwifi-project.org/.

[10] Microsoft activesync command reference protocol speficiation. http://msdn.microsoft.com/en-us/library/dd299441(v=exchg.80).aspx.

[11] Microsoft direct push. http://technet.microsoft.com/en-us/library/cc539119.aspx.

[12] Monsoon power monitor. http://www.msoon.com/LabEquipment/PowerMonitor/.

[13] Ue fast dormancy behavior. 3GPP discussion and decision note RP-090960, 2009.

[14] Ultra Low-Power RF Communications for Implanted Medical Applications and Low Duty-Cycle Systems. http://www.zarlink.com/zarlink/lowpower-rf_dutycycle_wp_nov06.pdf.

[15] FCC rules and regulations, MICS Band Plan. 2003.

[16] A. Adya, G. Cooper, D. Myers, and M. Piatek. Thialfi: A client notification service for internet-scale appli-cations. In *SOSP*, 2011.

[17] James Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *J. ACM*, May 1997.

[18] P. K. Athivarapu, R. Bhagwan, S. Guha, V. Navda, R. Ramjee, D Arora, V. N. Padmanabhan, and G. Varghese. Radiojockey: Mining program execution to optimize cellular radio usage. In *Mobicom*, 2012.

[19] Yossi Azar, Joseph Naor, and Raphael Rom. The competitiveness of on-line assignments. *J. Algorithms*, 18(2), 1995.

[20] SD Bao, CCY Poon, YT Zhang, and LF Shen. Using the timing information of heartbeats as an entity identifier to secure body sensor network. *IEEE Trans Inf Technol Biomed*, 12(6):772–9, 2008.

[21] Yigal Bejerano, Seung-Jae Han, and Li Li. Fairness and load balancing in wireless LANs using association control. *ACM/IEEE Transactions on Networking*, 2007.

[22] I. Caragiannis. Better bounds for online load balancing on unrelated machines. In *SODA 2008*.

[23] A. Carrol and G. Heiser. An analysis of power consumption in a smartphone. In *USENIX ATC*, 2010.

[24] A. Carroll and G. Heiser. An analysis of power con-sumption in a smartphone. In *ATC*, 2010.

[25] M. Cesana, I. Malanchini, and A. Capone. Modelling network selection and resource allocation in wireless access networks with non-cooperative games. In *IEEE MASS*, 2008.

[26] Ranveer Chandra, Paramvir Bahl, and Pradeep Bahl. Multinet: connecting to multiple ieee 802.11 networks using a single wireless card. In *INFOCOM 2004*.

[27] S. Cherukuri, K.K. Venkatasubramanian, and S.K.S. Gupta. BioSec: A biometric based approach for securing communication in wireless networks of biosensors implanted in the human body. *International Conference on Parallel Processing Workshops*, 2003.

[28] T. Cignetti, K. Komarov, and C. Ellis. Energy estimation tools for the Palm. In *MSWIM*, 2000.

[29] Cisco System Inc. Data sheet for cisco aironet 1200 series. 2004.

[30] S. Deng and H. Balakrishnan. Traffic-aware tech-niques to reduce 3g/lte energy consumption. In *CoNEXT*, 2012.

[31] T. Denning, K. Fu, and T. Kohno. Absence makes the heart grow fonder: new directions for implantable medical device security. In *HotSec 2008*.

[32] T. Denning, Y. Matsuoka, and T. Kohno. Neurosecurity: security and privacy for neural devices. *Neurosurgical Focus*, 2009.

[33] F. Dogar, P. Steenkiste, and K. Papagiannaki. Catnap: Exploit high bandwidth wireless interfaces to save energy for mobile devices. In *Mobisys*, 2010.

[34] M. Dong and L. Zhong. Chameleon: A color-adaptive web browser for mobile oled displays. In *MobiSys*, 2011.

[35] M. Dong and L. Zhong. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In *MobiSys*, 2011.

[36] E. Even-Dar, A. Kesselman, and Y. Mansour. Convergence time to Nash equilibria. *Lecture Notes in Computer Science*, pages 502--513, 2003.

[37] Eyal Even-Dar, Alex Kesselman, and Yishay Mansour. Convergence time to nash equilibrium in load balancing. *ACM Trans. Algorithms*, August 2007.

[38] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *WMCSA*, 1999.

[39] S. Gurun and C. Krinz. A run-time, feedback-based energy estimation model for embedded devices. In *WMCSA*, 2006.

[40] D. Halperin, T.S. Heydt-Benjamin, K. Fu, T. Kohno, and W.H. Maisel. Security and privacy for implantable medical devices. *IEEE Pervasive Computing 2008*.

[41] D. Halperin, T.S. Heydt-Benjamin, B. Ransford, S.S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W.H. Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *IEEE Symposium on Security and Privacy, 2008*.

[42] H. Han, Y. Liu, G. Shen, Y. Zhang, and Q. Li. Dozyap: Power-efficient wi-fi tethering. In *Mobisys*, 2012.

[43] Hao Han, Bo Sheng, Chiu C. Tan, Qun Li, and Sanglu Lu. A measurement based rogue ap detection scheme. In *INFOCOM*, 2009.

[44] Glenn Judd and Peter Steenkiste. Fixing 802.11 access point selection. *ACM SIGCOMM Computer Communication Review*, 2002.

[45] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha. Devscope: A nonintrusive and online power analysis tool for smartphone hardware components. In *CODES+ISSS*, 2012.

[46] Srikanth Kandula, Kate Ching-Ju Lin, Tural Badirkhanli, and Dina Katabi. Fatvap: Aggregating ap backhaul capacity to maximize throughput. In *NSDI 2008*.

[47] H. Kim, N. Agrawal, and C. Ungureanu. Revisiting storage for smartphones. In *FAST*, 2012.

[48] T. Korakis, O. Ercetin, S. Krishnamurthy, L. Tassiulas, and S. Tripathi. Link Quality based Association Mechanism in IEEE 802.11h compliant Wireless LANs. In *RAWNET 2005*.

[49] L. Sang and A. Arora. Capabilities of low-power wireless jammers. Technical Report OSU-CISRC-5/08-TR24, Ohio State Univ., 2008.

[50] B. W. Lampson. Lazy and speculative execution. talk at OPODIS, 2006.

[51] S. Lee, J. Kim, J. Lee, and B. H. Cho. State-of-charge and capacity estimation of lithium-ion battery using a new open-circuit voltage versus state-of-charge. *Journal of Power Sources*, 2008.

[52] Renato Paes Leme, Vasilis Syrgkanis, and Éva Tardos. The curse of simultaneity. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, New York, NY, USA. ACM.

[53] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46, February 1990.

[54] C. Li, C. Zheng, and C. Tai. Detection of ECG characteristic points using wavelet transforms. *IEEE Trans. on Biomedical Engineering*, 42(1), 1995.

[55] M. Lu and J. Wu. Localized Access Point Association in Wireless LANs with Bounded Approximation Ratio. In *ICCCN 2008*.

[56] Dijun Luo, Xiaojun Zhu, Xiaobing Wu, and Guihai Chen. Maximizing lifetime for the shortest path aggregation tree in wireless sensor networks. In *INFOCOM'11*, pages 1566 --1574.

[57] J. Martinez, R. Almeida, S. Olmos, A. Rocha, and P. Laguna. A wavelet-based ECG delineator: evaluation on standard databases. *IEEE Trans. on Biomedical Engineering*, 51(4), 2004.

[58] I. Martinovic, P. Pichota, and J.B. Schmitt. Jamming for good: a fresh approach to authentic communication in WSNs. In *Wisec 2009*.

[59] G. Matthew. *802.11 wireless networks: the definitive guide*. 2002.

[60] Vivek Mhatre and Konstantina Papagiannaki. Using smart triggers for improved user performance in 802.11 wireless networks. In *MobiSys 2006*.

[61] I. Milchtaich. Congestion games with player-specific payoff functions. *Games and economic behavior*, 13(1):111--124, 1996.

[62] Arunesh Mishra, Minho Shin, and William Arbaugh. An empirical analysis of the ieee 802.11 mac layer handoff process. *SIGCOMM Computer Communication Review 2003*.

[63] K. Mittal, E.M. Belding, and S. Suri. A game-theoretic analysis of wireless access point selection by mobile users. *Computer Communications*, 2008.

[64] R. Mittal, A. Kansal, and R. Chandra. Empowering developers to estimate app energy consumption. In *MobiCom*, 2012.

[65] Rohan Murty, Jitendra Padhye, Alec Wolman, and Brian Zill. Designing high performance enterprise wi-fi networks. In *NSDI 2008*.

[66] R. Myers. *Classical and modern regression with applications*, volume 2. Duxbury Press Belmont, CA, 1990.

[67] Anthony J. Nicholson, Yatin Chawathe, Mike Y. Chen, Brian D. Noble, and David Wetherall. Improved access point selection. In *MobiSys 2006*.

[68] Ioannis Papanikos and Michael Logothetis. A study on dynamic load balance for ieee 802.11b wireless lan. In *COMCON 2001*.

[69] A. Pathak, Y. Hu, M. Zhang, P. Bahl, and Y. Wang. Fine-grained power modeling for smpartphones us-ing system call tracing. In *EuroSys*, 2011.

[70] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff. What is keeping my phone awake? characterizing and de-tecting no-sleep energy bugs in smartphone apps. In *Mobisys*, 2012.

[71] CCY Poon, Y.T. Zhang, and S.D. Bao. A novel biometrics method to secure wireless body area sensor networks for telemedicine and m-health. *IEEE Communications Magazine*, 44(4):73--81, 2006.

[72] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Top: Tail optimization protocol for cellular radio resource allocation. In *ICNP*, 2010.

[73] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Profiling resource usage for mobile ap-plications: A cross-layer approach. In *Mobisys*, 2011.

[74] M. Ra, B. Priyantha, A. Kansal, and J. Liu. Improving energy efficiency of personal sensing applications with heterogeneous multi-processors. In *Ubicomp*, 2012.

[75] I. Ramani and S. Savage. Syncscan: practical fast handoff for 802.11 infrastructure networks. In *INFOCOM 2005*.

[76] A. Raniwala and T. Chiueh. Architecture and algorithms for an IEEE 802.11-based multi-channel wireless mesh network. In *INFOCOM 2005*.

[77] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker. A statistical test suite for random and pseudorandom number generators for cryptographic applications. *NIST*, 2001.

[78] A. Schulman, T. Schmid, P. Dutta, and N. Spring. Demo: Phone power monitoring with BattOr. In *MobiCom*, 2011.

[79] Srinivas Shakkottai, Eitan Altman, and Anurag Kumar. Multihoming of users to access points in wlans: A population game perspective. *IEEE JSAC 2007*.

[80] David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62, December 1993.

[81] A. Shye, B. Scholbrock, and G. Memik. Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures. In *MICRO*, 2009.

[82] P. Stemen and S. Berard. Understanding connected standby. Microsoft Build conference talk, 2011.

[83] M. Strasser, C. Pöpper, S. Capkun, and M. Cagalj. Jamming-resistant key establishment using uncoordinated frequency hopping. In *IEEE Symposium on Security and Privacy, 2008*.

[84] Karthikeyan Sundaresan and Konstantina Papagiannaki. The need for cross-layer information in access point selection algorithms. In *IMC 2006*.

[85] S. Suri, C.D. Tóth, and Y. Zhou. Selfish load balancing and atomic congestion games. *Algorithmica*, 47(1):79--96, 2007.

[86] S. Vasudevan, K. Papagiannaki, C. Diot, J. Kurose, and D. Towsley. Facilitating access point selection in ieee 802.11 wireless networks. In *IMC 2005*.

[87] K. Venkatasubramanian, A. Banerjee, and S.K.S. Gupta. Ekg-based key agreement in body sensor networks. In *Computer Communications Workshops*, 2008.

[88] H. Wang, B. Sheng, and Q. Li. Elliptic curve cryptography-based access control in sensor networks. *International Journal of Security and Networks*, 1(3):127–137, 2006.

[89] Yin Wang, Yuan He, Xufei Mao, Yunhao Liu, Zhiyu Huang, and XiangYang Li. Exploiting constructive interference for scalable flooding in wireless sensor network. In *INFOCOM'12*.

[90] Haitao Wu, Kun Tan, Yongguang Zhang, and Qian Zhang. Proactive scan: Fast handoff with smart triggers for 802.11 wireless lan. *INFOCOM 2007*.

[91] F. Xu, C.C. Tan, Q. Li, G. Yan, and J. Wu. Designing a practical access point association protocol. In *INFOCOM*, 2010.

[92] Wenyuan Xu, Wade Trappe, Yanyong Zhang, and Timothy Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *MobiHoc 2005*.

[93] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha. Appscope: Application energy metering framework for android smartphone using kernel activity monitoring. In *USENIX ATC*, 2012.

[94] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. Dick, Z. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *CODES+ISSS*, 2010.

[95] Xiaojun Zhu, Qun Li, and Guihai Chen. Online vector scheduling and generalized load balancing. *Technical Report WM-CS-2012-01*, 2012.

VITA

Fengyuan Xu received his Bachelor degree of Science in Computer Science from Jilin University, China, in 2007. Then he began to pursue his Doctor of Philosophy degree of Computer Science at the College of William and Mary under the supervision of Dr. Qun Li. His research interests span a broad range of topics in the space of the pervasive computing, energy efficiency, security & privacy and wireless systems.