ENERGY EFFICIENT LOCATION TRACKING AND COMPUTATION OFFLOADING FOR MOBILE DEVICES

by

Lei Zhang
B.Eng., Huazhong University of Science and Technology, 2011

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

© Lei Zhang 2013
SIMON FRASER UNIVERSITY
Summer 2013

All rights reserved.
However, in accordance with the Copyright Act of Canada, this work may be reproduced without authorization under the conditions for “Fair Dealing.” Therefore, limited reproduction of this work for the purposes of private study, research, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.
APPROVAL

Name: Lei Zhang

Degree: Master of Science

Title of Thesis: Energy Efficient Location Tracking and Computation Offloading for Mobile Devices

Examiner Committee: Dr. Greg Mori
Chair

Dr. Jiangchuan Liu, Senior Supervisor

Dr. Qianping Gu, Supervisor

Dr. Jie Liang, SFU Examiner

Date Approved: _________________________________
Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the “Institutional Repository” link of the SFU Library website (www.lib.sfu.ca) at http://summit/sfu.ca and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author’s written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, British Columbia, Canada

revised Fall 2011
Abstract

In today’s mobile devices, the battery reservoir remains severely limited in capacity, making energy consumption a key concern in the design and implementation of mobile applications. In this thesis, we closely examine two approaches to improve the energy efficiency of mobile applications: smartly utilizing the local computation resources and adaptively offloading the computation to the remote cloud.

We use location tracking as a case study for the former. We identify the defects of conventional location tracking services that rely only on GPS, and develop SensTrack, a novel solution that jointly use multiple sensor hints in modern smartphones. For the latter, we measure the energy reduction of computation offloading for two realworld mobile applications and identify the key influential factors. We then formulate the energy-efficient scheduling problem for computation offloading and present effective solutions.

Keywords: Energy Efficiency; Mobile; Location Tracking; Computation Offloading; Cloud
To my family!
“Hope for the best, prepare for the worst.”
Acknowledgments

I would first like to thank my senior supervisor Dr. Jiangchuan Liu. His attitude towards research has greatly influenced me. His significant enlightenment, guidance and encouragement on my research are invaluable for the success of my M.Sc studies. His insights and suggestions have made the road to completing this thesis smoother.

I want to thank Dr. Qianping Gu and my thesis examiner Dr. Jie Liang, for reviewing this thesis and providing helpful suggestions that helped me to improve the quality of the thesis. I also want to thank Dr. Greg Mori for taking the time to chair my thesis defense.

Many colleagues and friends of mine not only provided help in my studies but also in my everyday life. The time with them is unforgettable. I am deeply indebted to them.

Finally, I want to thank my family for their love and care. Nothing would happen without your supports, my parents. I love you all!
Contents

Approval ii
Partial Copyright License iii
Abstract iv
Dedication v
Quotation vi
Acknowledgments vii
Contents viii
List of Tables xi
List of Figures xii

1 Introduction 1
1.1 Background ......................................................... 1
1.2 Motivation ............................................................. 3
1.3 Contribution ........................................................... 4
1.4 Organization of the Thesis ........................................... 4

2 Related Work 6
2.1 Energy Consumption on Mobile Devices ................................ 6
2.2 Energy-Efficient Location Sensing ........................................ 7
2.3 Computation Offloading ............................................... 8
3 SensTrack: Energy-Efficient Location Tracking  
3.1 Challenges and Opportunities .................................. 11  
3.1.1 Limited Availability of GPS vs Multiple Location Sensing Methods . 11  
3.1.2 Unnecessary GPS Samplings vs Adaptive Sampling .................... 13  
3.1.3 Assistance from Other Sensors .................................. 14  
3.2 SensTrack: Design Details ....................................... 15  
3.2.1 Overview ....................................................... 15  
3.2.2 Track Reconstruction: Gaussian Process Regression ................. 16  
3.2.3 Switching Location Sensing Methods .......................... 18  
3.2.4 Utilizing Sensor Hints ........................................ 20  
3.3 Evaluation ....................................................... 21  
3.3.1 Data Collection and Methodology ............................... 21  
3.3.2 Accuracy ....................................................... 22  
3.3.3 Energy Efficiency ............................................. 23  
3.3.4 Energy-Accuracy Trade-off .................................... 24  
3.3.5 Transmission Overhead ....................................... 26  
3.4 Further Discussion ............................................... 27  
3.4.1 Multiple Mobility Patterns .................................... 27  
3.4.2 Energy Consumption of Accelerometer and Orientation Sensor ...... 27  
3.4.3 Other Indoor Localization Technologies .......................... 27  
3.5 Summary ......................................................... 28  

4 Energy-Efficient Computation Offloading  
4.1 Motivation ....................................................... 29  
4.2 Measurement ..................................................... 31  
4.2.1 Devices, Tools and Configurations .............................. 31  
4.2.2 Test Applications ............................................. 32  
4.2.3 Local Execution vs Offloaded Execution ......................... 34  
4.2.4 Computation Offloading under Dynamic Wireless Networks .......... 36  
4.2.5 Power Model .................................................. 40  
4.3 Problem Formulation ............................................. 41  
4.4 Offline Solution ................................................ 45  
4.5 Online Extension ................................................ 47
5 Conclusion and Future Work

Bibliography
## List of Tables

3.1 Effect of window size $T$ .................................................. 20  
3.2 Effect of threshold $\theta$ .................................................. 20  
3.3 Average error of predicted locations .................................. 22  
3.4 Average WiFi traffic ...................................................... 26  

4.1 Local execution vs Offloaded execution ............................... 36  
4.2 Delay tolerance in online gaming ...................................... 37  
4.3 Average number of frames updated before the response arrived 38  
4.4 Power Model ............................................................... 40  
4.5 Energy efficiency improvement ........................................ 53  
4.6 Number of reschedulings ................................................ 54
# List of Figures

3.1 Tracking results when $T = 5s$, $\theta = 45^\circ$, $D = 100m$, $v = 8m/s$ .................................. 12
3.2 WiFi signal strength along the track .................................................. 13
3.3 A part of system log when running a location-based application ................. 14
3.4 The system architecture ................................................................. 15
3.5 Comparison of the energy efficiency .................................................. 24
3.6 Trade-off between sampling rate and accuracy ...................................... 25
3.7 WiFi traffic of SensTrack ................................................................. 26

4.1 Framework of the chess game .......................................................... 33
4.2 Framework of a cloud gaming platform ............................................. 34
4.3 Comparison between local execution and offloaded execution for the chess game 35
4.4 Comparison between local execution and offloaded execution for OnLive .... 35
4.5 Captured frames in OnLive with low and high network latency ............... 38
4.6 Impact of packet loss ................................................................. 39
4.7 Wireless channel model ................................................................. 48
4.8 Flow chart of the online heuristic ................................................... 50
Chapter 1

Introduction

1.1 Background

Mobile devices, including smartphones and tablets, are increasingly becoming an essential part of human life as the most effective and convenient tools for communication and entertainment, which are not bounded by time and place. The recent years have seen an enormous growth in the popularity and visibility of smartphones and tablets. At the end of 2012, the number of mobile-connected devices reached 6.8 billion, up from 6.0 billion in 2011 and 5.4 billion in 2010 [9]. The global mobile data traffic is also expected to increase 13-fold between 2012 and 2017, which will surpass 10 exabytes per month in 2017 [31]. As mobile devices are penetrating into people’s everyday life, mobile users are expecting more features and requesting ubiquitous availability for the services provided on mobile platforms. Manufacturers have been adding an ever-increasing set of features to small mobile devices, which are no longer binary-use gadgets, but fully-fledged computers. In spite of the improvement in processing power, feature-set, and sensing capabilities, mobile devices continue to suffer from limited battery life. As the only power source of most mobile terminals, battery has seen relatively slow improvement in the past decade, whose capacity is growing by only 5% annually [51]. Therefore, the limited power supply has become a major impediment to providing reliable and sophisticated mobile applications to meet user demands.

Today’s smartphone not only serves as the key computing and communication mobile device of choice, but it also comes with a rich set of embedded sensors, such as an accelerometer, digital compass, gyroscope, GPS, microphone, and camera. Collectively, these sensors are enabling new applications across a wide variety of domains, such as healthcare, social
CHAPTER 1. INTRODUCTION

networks, environmental monitoring, and transportation. Among those broad-domain applications, location based applications provide one of the most fundamental services. Consumer and advertiser expenditure on location based services is expected to approach $10 billion by 2016 [48]. The reason that location based applications become so popular is two-fold. First, location based services rely on the knowledge about the user’s geographical location to obtain relevant information on the spot, and thus offer the user a plethora of options to satisfy his/her needs under that particular context. Second, a typical modern mobile device usually has the ability to locate or estimate its current position. The localization technologies used today mainly based on Global Positioning System (GPS), other technologies also obtain assistance from WiFi and GSM, each of which can vary widely in energy consumption and localization accuracy. As it is known to be more accurate, GPS is often preferred on mobile platforms over its alternatives such as GSM/WiFi based positioning systems. Unfortunately, it is well-known that GPS, the core enabler of many location-based applications, is power-hungry. The aggressive usage of GPS can cause the battery to completely drain within a few hours [22, 63]. Because of the high energy expense for localization, location based applications cannot assume continuous and ubiquitous location access in their design. To save the energy consumed by the major component of a mobile device, such as GPS, researchers have done extensively work on smartly and efficiently utilizing the ever-growing capability of local computing and sensing under various working contexts.

Even though mobile device hardware and mobile networks continue to evolve and to improve, resource poverty, especially energy scarcity, will always be the major obstacle for many applications. Empowered by the cloud computing paradigm, both academic researchers and industrial pioneers have been striving to enhance the capabilities of mobile devices with cloud resources. Mobile cloud computing, which combines the strength of clouds and the convenience of mobile devices, naturally appears and attracts tremendous attentions. For example, cloud storage and synchronization applications, represented by Dropbox [5] and iCloud [8], are embraced by most mobile users. Gaikai [6] and OnLive [12], providing cloud-based distributed interactive applications, execute video games in the cloud and deliver video stream to end users, which shift the hardware/software requirements and the necessary computing loads to cloud proxies. To extend the battery life of mobile devices as well as to overcome other limits such as computation capability, network bandwidth, and storage space, computation offloading is a widely adopted solution. The basic idea is to send the heavy computation to resourceful cloud servers and receive results
from these remote servers so that the energy for executing those tasks can be saved and mobile platforms' capabilities can be augmented. However, energy can only be saved by computation offloading with the prerequisite that the savings from offloading the computation must exceed the energy cost of the additional communication. In general, offloading is beneficial when large amounts of computation are needed with relatively small amounts of communication overhead. Yet how to implement this idea into practice needs careful analysis.

In this thesis, we investigate approaches that can improve the energy efficiency of mobile devices from two aspects: utilizing the local computation and offloading the computation to the cloud. For the local computation approach, we take location tracking, one of the most important services on mobile platforms, as the example to illustrate that the sensor hints on smartphones can be used to reduce the usage of GPS which normally consumes significant amounts of energy. For the computation offloading approach, we conduct the measurements on two test applications to examine the energy efficiency of computation offloading, and propose energy-efficient transmission scheduling schemes that make dynamic offloading decisions adaptive to the changing network conditions.

1.2 Motivation

Our motivation of this thesis consists of two aspects:

As smartphones nowadays are becoming more and more powerful, applications providing location based services have been increasingly popular. Many, if not all, smartphones are equipped with a powerful sensor set (GPS, WiFi, the acceleration sensor, the orientation sensor, etc), which makes them capable to accomplish complicated tasks. As the core enabler of most location tracking applications on smartphones, GPS incurs an unacceptable energy cost that can cause the complete battery drain within a few hours. Although GPS is often preferred over its alternatives, the coverage areas of GPS are still limited (GPS typically can not function indoors). To this end, we expect to improve the energy efficiency of traditional location tracking service as well as to expand its coverage areas by fully utilizing the local computing and sensing capability.

Other than relying on the local computation, offloading the computation to the cloud is another feasible solution that effectively expands the usability of mobile devices beyond their physical limits, and also greatly extends their battery charging intervals through potential
energy savings. Intuitively, offloading is beneficial whenever a computation-intensive task is not affordable by local resources. However, moving the task to the remote cloud may incur a large volume of data transfer, which, though may not be a severe problem for users with high-speed wired network connection, can largely contradict the benefit for mobile users with their energy-hungry wireless interfaces. The situation becomes even worse when the wireless connection is unstable and of poor quality, which indicates performance degradation or more energy consumption for successful data transfer. Therefore, it is crucial to have an energy-efficient offloading scheme for mobile devices.

1.3 Contribution

The main contributions of this chapter are as follows:

• For improving smartphones’ energy efficiency through local computation, we identify the problems of traditional location tracking service including limited availability of GPS and unnecessary GPS samplings. The opportunities of energy-efficiency improvements by utilizing the assistance from sensors on smartphones are discussed. We then present the detailed design of an energy-efficient location tracking service, SensTrack. As the main component, a track reconstruction algorithm based on Gaussian Process Regression is proposed. Other mechanisms for making smart adaptive sampling decisions are also discussed. We implement a prototype of SensTrack, and evaluate the proposed system based on the traces collected in real-world experiments.

• For offloading the computation to the cloud adaptively to the network conditions, we conduct a measurement study for two real-world mobile applications to investigate the variation in performance and energy efficiency of computation offloading under dynamic network environment and identify the key influential factors. We then formulate the energy-efficient offloading scheduling problem, and propose the offline and online solutions with simulation results.

1.4 Organization of the Thesis

The rest of this thesis is organized as follows. We present an overview of relate work systematically in Chapter 2. In Chapter 3, we first describe the defects of typical location-based
applications that utilize GPS, then present our detailed design of SensTrack, at last we evaluate our proposal and analyze the performance improvement. In Chapter 4, a measurement study for computation offloading in wireless networks is first presented, the energy-efficient scheduling problem is then formulated, and effective solutions are proposed and evaluated. Chapter 5 concludes the thesis, and notes some future directions.
Chapter 2

Related Work

In this chapter, we will give an overview of the studies previously done on the topics related to this thesis. More specifically, we will discuss the related work in three aspects: measurement studies about mobile devices’ energy consumption, energy-efficient location sensing approaches, and the recent works on computing offloading to cloud.

2.1 Energy Consumption on Mobile Devices

Energy consumption has been and will always be one of the biggest concerns in mobile/wireless networks. In order to have the mobility, mobile devices rely on battery power, and on the other hand, are limited by the battery capacity in spite of the dramatic improvement in functionality and hardware infrastructure. Understanding the energy consumption of mobile systems is essential for further reducing it. Toreveal the characteristics of energy cost in mobile systems, researchers have performed measurement studies that provide significant insights from different aspects. The measurement in [54] indicated that the power drained by the network interface constitutes a large fraction of the total power used by the personal digital assistant (PDA). The paper also examined the opportunities of application-specific optimizations to reduce the energy consumption from network interface. In order to obtain more practical knowledge, L. M. Feeney et al. measured the power consumption of an IEEE 802.11 wireless network interface operating in an ad hoc networking environment, and applied a collection of linear equations to describe the energy consumption behavior [25]. According to the resultant linear energy model, approximate power consumption of the network interface in different states (sending, receiving, idle) can be calculated. A
more recent study [15] compared the energy consumption features of three widespread mobile networking technologies: 3G, GSM, and WiFi. The following highlight findings were presented: in 3G, a significant portion of energy, referred as tail energy, is wasted in high-power states after the completion of a typical transfer; GSM also exhibits a similar trend with much shorter tail time; WiFi, although its association overhead is comparable to the tail energy, has significantly more efficient data transfer than 3G. In another work, A. Carroll et al. measured the power consumption of a modern mobile device (the Openmoko Neo Freerunner mobile phone), broken down to the devices major subsystems (CPU, memory, touchscreen, graphics hardware, audio, storage, and various networking interfaces), under a wide range of realistic usage scenarios [16]. Energy demands on mobile platforms have also been characterised in the user behavior’s perspective [56] that can help to understand the interdependencies between resources under particular context.

2.2 Energy-Efficient Location Sensing

To track the users’ locations, many energy-efficient sensing approaches with adaptive sensing policies have been proposed to minimize the energy consumption [22, 39, 41, 47]. With the objective of minimizing the localization error for a given energy budget, EnLoc [22], an energy-efficient localization framework, includes a heuristic with a local mobility tree to predict the next sensing time by utilizing the dynamic programming technique. Jigsaw [41] uses the information obtained from the acceleration sensor and the microphone to continuously monitor human activities and environmental context. According to the user’s mobility patterns, a discrete-time Markov Decision Process is employed to learn the optimal GPS duty cycle schedule with a given energy budget.

There are also works based on the observation that the required localization accuracy varies with locations. An adaptive location service for mobile devices, a-Loc [39] uses a Bayesian estimation framework to determine the dynamic accuracy requirement, and tunes the energy expenditure accordingly. It argued in [47] that given the less accuracy of GPS in urban areas, it suffices to turn on GPS adaptively to achieve this accuracy. The rate-adaptive positioning system for smartphone applications (RAPS) was then proposed to minimize energy consumption with given accuracy threshold by using the information of moving distance, space-time history, and cell tower-based blacklisting.

Utilizing the sensing power of smartphones is not a new topic in literature. M. Keally
et al. [33] presented the design of Practical Body Networking (PBN) system to provide practical activity recognition with mobile devices, which combines the sensing power of on-body wireless sensors with the additional sensing power, computational resources, and user-friendly interface of an Android smartphone through the unification of TinyOS motes and Android smartphones. Another interesting ongoing work discusses how to fuse information from Microsoft Kinect’s tracking with the smartphone’s sensor readings to improve Kinect gaming experience [44].

Inspired by many existing studies, in this thesis we take efforts to achieve a high energy efficiency by reducing the sampling rate of sensing users’ locations. However, our work uses a novel approach by utilizing the acceleration sensors and the orientation sensors on smartphones to capture the geometric features of users’ moving trajectories. We will further explain the difference between SensTrack and existing works in Chapter 3.

2.3 Computation Offloading

With recent advance in cloud computing, computation offloading has received great attention, since the attractive features of cloud computing such as high availability, high reliability, and “infinite” resources have made offloading as a very promising option for mobile platforms. As one of the pioneer works, a high-level discussion was given in [36], which investigated the opportunity of offloading applications to cloud as well as the energy overhead for issues such as privacy and security, reliability, and data communication before offloading. A simple mathematical model was presented, which suggested that offloading is beneficial to the applications with large amounts of computation and little data to send. Further, an experimental study was performed to analyze the critical factors that affect the energy consumption of mobile clients in cloud computing [45]. For the specific hardware model, the authors concluded the condition for offloading to save the mobile device’s battery: the migrated workload should perform more than 1000 cycles of computation for each byte of the generated data. Another work also examined the energy trade-off when offloading a task [37], which analyzed the benefits and drawbacks of one specific application: cloud-assisted global search and synchronization. The paper argued that the synergy between mobile platforms and cloud computing is under-utilized and should be further explored.

Systems that offload computation to Virtual Machine (VM) instances or clouds have also been introduced by researchers. MAUI [23] was proposed to offload methods from .NET
CHAPTER 2. RELATED WORK

applications to a remote runtime environment based on a history of energy consumption, which has considered some low-level challenges such as failure handling and program state transferring. One drawback of MAUI is that it required programmers to annotate methods that can be offloaded for remote execution. Similar to MAUI, CloneCloud [20] partitioned applications using a framework that combined static program analysis with dynamic program profiling and optimized execution time or energy consumption using an optimization solver. Moreover, CloneCloud focused on a detailed design for state migration and merging, and allowed remotely executing virtualized methods to call native functions. To provide secure computation offloading, Enterprise Centric Offloading System (ECOS) was proposed in [26], in which the involved data/state was categorized into different privacy levels, and the private data was transferred through the TLS-encrypted connection. It has also discussed issues like multiplexing offloads from multiple devices on a single compute resource, and churning resources during the application’s execution. For Android platforms, mobile cloud computing systems such as Hyrax [43] and Cuckoo [34] have been designed and implemented. Hyrax was proposed to allow Android smartphones porting Hadoop applications from cloud. Cuckoo was presented as a framework to assist the development of mobile applications that can benefit from runtime computation offloading.

Two key questions in computation offloading are what to offload or when to offload. While many of the previous papers [18, 38, 46, 52, 55] focused on what or how to offload, a few of them [57, 58, 60] tackled the problem of when to offload, and even fewer [20, 28] attempted to solve both of the two problems. Even in the studies that solved the two problems, the researchers often treated them separately. We argue that, for offloading in mobile cloud computing, these two problems are not independent and should be considered jointly. In this thesis, we do not focus on partitioning the program and the corresponding low-level details such as program profiling and state migration, rather we manage to optimally schedule the offloading request and the required transmission based on the application demands and the network condition.
Chapter 3

SensTrack: Energy-Efficient Location Tracking

In this chapter, we use location tracking, one of the most fundamental services as a case study, and discuss the possibility to improve mobile devices’ energy efficiency through fully utilizing local computing and sensing ability. In particular, we present the design of SensTrack, a location tracking service on smartphones that provides user’s moving trajectory while reducing its impact on the device’s battery life. By applying different localization technologies, we expand the coverage area compared to the traditional approach that only uses GPS. In addition, the sensor hints from the smartphone itself can help us make decisions about adaptive sampling. SensTrack smartly selects the location sensing methods between WiFi and GPS, and reduces the sampling rate by utilizing the information from acceleration sensor and orientation sensor, two of the most common sensors found on smartphones today. We have implemented a prototype on the Google Nexus S phone, which continuously collects data from the acceleration sensor and the orientation sensor, and records the location samples from GPS and WiFi. Experiments have been conducted on a real world path while the phone was carried by a mobile user in a region of our university campus. The collected data is further analyzed and filtered on computers. To predict the user’s original trajectory, a track reconstruction algorithm based on a machine learning technique is also implemented on the server side. Performance evaluation on the real data sets shows that SensTrack only needs 7% GPS samples of the naive approach and saves nearly 90% GPS activated time. Meanwhile, SensTrack reconstructs the user’s trajectory with high accuracy.
CHAPTER 3. SENSTRACK: ENERGY-EFFICIENT LOCATION TRACKING

and better coverage.

3.1 Challenges and Opportunities

In this section, we start by describing the defects of typical location-based applications that utilize GPS, including limited availability and unnecessary samples. We then discuss the opportunities for making improvements.

3.1.1 Limited Availability of GPS vs Multiple Location Sensing Methods

It should be noted that traditional GPS cannot work properly under the indoor environment. The standard GPS receiver requires signals from at least 4 satellites simultaneously to calculate and output 3-dimensional locations and velocity information [32]. Therefore, the mobile devices need to be in line-of-sight contact with the GPS satellites, which significantly limits the usage of typical location based applications.

Figure 3.1(a) shows one track that we took using GPS on a mobile device. Although we did not stop recording, the track ends once it entered the building (the Academic Quadrangle in our campus), which indicates the performance of GPS largely depends on the working condition. The signals from GPS satellites can be blocked not only by buildings but also by canyon walls, trees, and even thick clouds. When the user walks through buildings, GPS equipped by a normal smartphone cannot function since the lack of satellite signals. Even worse, GPS units may consume more energy than the normal situation when there is no satellite signals [19].

Besides GPS, there also exist alternate location sensing technologies. For example, Android OS provides a network-based localization mechanism, which exploits GSM footprints from cell towers and WiFi signals to obtain an approximate location. Although the network-based location sensing is not as accurate as GPS, it provides the possibility to keep tracking inside a building since it mainly relies on the WiFi connection, in which case GPS units can be deactivated to save battery. For the scenarios like university campus, hotels or hospitals, we can always assume persistent wireless local network access, which implies that other location sensing methods may provide us valid options when GPS is out of use.

Figure 3.2 shows the received WiFi signal strength along the track presented in Figure 3.1(a). The dash line indicates the time stamp (588s) at which the user entered the Academic Quadrangle. There are some spikes before 588s (201s ~ 216s, 335s ~ 368s, 387s ~ 398s,
Figure 3.1: Tracking results when $T = 5s$, $\theta = 45^\circ$, $D = 100m$, $v = 8m/s$
537s \sim 558s), which means that the user can receive some WiFi signal for a short time when passing by buildings. After entering the building at 588s, the received WiFi signal stayed at a relatively high level since the WiFi connection is assured in teaching areas of the university campus. This figure can support our argument that, when the user is inside a building, WiFi signal is usually relatively strong. Therefore, the network-based localization can be a valid choice under the indoor environment where GPS is no longer available. The idea is to use the GPS satellite signal and the wireless network connection as indicators for switching between GPS and the network-based location sensing method.

### 3.1.2 Unnecessary GPS Samplings vs Adaptive Sampling

The GPS sensor can sample the user’s location at a relatively high rate. However, it is not ideal to record every location update since the error for each location sample varies. To make the path more smooth and fit the real trajectory, a typical location based application usually updates the user’s location only if the distance to the last valid location sample is larger than a certain threshold [14]. Therefore, with a fixed and frequent GPS location sampling policy, it probably introduces a significant amount of unnecessary GPS samples.

To demonstrate this, we collect the system log of an Android application, *My Tracks* [10], which uses the GPS sensor in mobile devices to record the paths that users take while
hiking, cycling, running, or participating in other activities. Figure 3.3 shows part of the system log, demonstrating its executing history in one run. As shown in the figure, the application usually takes several GPS samples to get one valid location update, in which case the threshold is 5 meters. Our experimental result in this case shows that up to 79% location samples of My Tracks are unnecessary. Since many of the samples are discarded, these invalid location measurements cause unnecessary energy consumption.

### 3.1.3 Assistance from Other Sensors

Nowadays smartphones become more and more powerful in terms of hardware, which usually contains various sensors. As an example, iPhone 4 is equipped with several environmental sensors, including an ambient light sensor, a magnetic compass, a proximity sensor, an accelerometer, and a three-axis gyroscope [3]. Android 4.0 (API Level 14) also supports up to 13 kinds of sensors [2], even though the sensors’ availability varies from device to device. The supported list of sensors in a Google Nexus S phone consists of: one KR3DM 3-axis Accelerometer, one AK8973 3-axis Magnetic field sensor, one AK8973 Orientation sensor, one GP2A Proximity sensor, one GP2A Light sensor, one Linear Acceleration Sensor, one Rotation Vector Sensor, one K3G Gyroscope sensor, and one Gravity Sensor [1].

To reduce unnecessary GPS samples, adaptive sampling is proposed in many existing works [22, 39, 41, 47]. Usually we need additional information to make adaptive sampling decisions, which may include the location history, the speed history, the distance information, remaining battery power, the accuracy requirement, etc. In this work, we utilize the powerful sensors equipped by smartphones to obtain the information about changes of the orientation, moving speed, and traveled distance. Based on these useful information, we are
able to make smart adaptive sampling decisions. The detailed design is described in the following section.

3.2 SensTrack: Design Details

3.2.1 Overview

To reduce the frequency of location sensing, SensTrack periodically collects data from the corresponding sensor to detect a turning point or estimate current speed and the distance from the last recorded location. The high energy efficiency of this approach is supported by the fact that the GPS sensor consumes much more energy than the acceleration sensor and the orientation sensor [35, 47]. When the GPS satellite signal is not available and the WiFi connection is active, SensTrack switches to the network-based location sensing method to obtain the raw coordinates. The last step of SensTrack is to upload the coordinates of sampled locations to an online server that uses a machine learning algorithm to reconstruct a smooth and accurate trajectory.

![System Architecture Diagram](image)

Figure 3.4: The system architecture

Figure 3.4 demonstrates the SensTrack’s system architecture. The service consists of two
stages: the first is to collect the location samples; and the second is to reconstruct the original trajectory. Given the working conditions, SensTrack switches between the GPS-based and the network-based localization methods using the GPS or WiFi sensors, respectively. By utilizing the sensor hints from the acceleration sensor and the orientation sensor, SensTrack is able to make smart adaptive sampling decisions in the GPS mode. For example, when the smartphone detects a turning point or if it estimates an unreasonable speed or an unexpected large traveling distance, it uses GPS to record the current location. After the server side receives all the collected location samples, a Gaussian Process Regression algorithm is then employed to predict the trajectory that the user has taken.

3.2.2 Track Reconstruction: Gaussian Process Regression

Once the collection of location samples is finished, it is not ideal to simply connect all the recorded locations, since the distances between any two successive locations may not be the same. For some parts of a trajectory, the recorded locations can be very sparse, while for other parts, the location samples may be relatively intensive. If we simply connect the location samples, the resultant trajectory can be very abstract. Therefore, uploading the collected data to the online server either by a wireless or wired connection to reconstruct the trajectory is our last stage. We adopt the Gaussian Process Regression (GPR), a machine learning technique to perform the interpolation. The training set of the algorithm is the recorded critical locations decided by the sensor hints which capture most of key features of a trajectory. And the testing set is the predicted locations between the successive but far-away location samples. Combing both input and output gives us the final trajectory. We next detailed describe GPR and how the user’s trajectory can be reconstructed by using GPR.

A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution, and is fully specified by a mean function and a covariance function [50]. The inference of continuous values with a Gaussian process prior is known as Gaussian Process Regression. Consider $x$ as a general random variable. We define the mean function $m(x)$ and the covariance function $k(x, x')$ of a real process $f(x)$ as

\[
m(x) = E[f(x)],
\]

\[
k(x, x') = E[(f(x) - m(x))(f(x') - m(x'))],
\]
and can write the Gaussian process as

\[ f(x) \sim N(m(x), k(x, x')). \]

For notational simplicity the mean function is usually set to be zero. In our method the covariance function will be the squared exponential covariance function, although other covariance functions may also be useful. Assuming that observations are noise-free, the covariance function specifies the covariance between pairs of random variables

\[ cov(f(x_p), f(x_q)) = k(x_p, x_q) = \exp(-\frac{1}{2}|x_p - x_q|^2). \quad (3.1) \]

For a testing data set \( X_* \), we can generate a random Gaussian vector \( f_* \) for target values with the covariance matrix calculated from Equation 3.1

\[ f_* \sim N(0, K(X, X_*)). \]

Therefore, the joint distribution of the training outputs \( f \) and the test outputs \( f_* \) according to the prior is

\[
\begin{bmatrix} f \\ f_* \end{bmatrix} \sim N \left( 0, \begin{bmatrix} K(X, X) & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix} \right). \quad (3.2)
\]

If \( X \) contains \( n \) training points and \( X_* \) contains \( n_* \) test points, then \( K(X, X_*) \) is the \( n \times n_* \) matrix of the covariances evaluated at all pairs of training and test points. And the other entries \( K(X, X), K(X_*, X), \) and \( K(X_*, X_*) \) are similar.

If observations are noisy, we can write \( y = f(x) + \epsilon \). Assuming additive independent identically distributed Gaussian noise \( \epsilon \) with variance \( \sigma^2_n \), we have the prior as

\[ cov(y_p, y_q) = k(x_p, x_q) + \sigma^2_n \delta_{pq} \]

or

\[ cov(y) = K(X, X) + \sigma^2_n I, \]

where \( \delta_{pq} \) is a Kronecker delta which is one when \( p = q \) and zero otherwise. Introducing the noise in Equation 3.2, the joint distribution of the observed target values and the function values at test points according to the prior will be

\[
\begin{bmatrix} y \\ f_* \end{bmatrix} \sim N \left( 0, \begin{bmatrix} K(X, X) + \sigma^2_n I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix} \right). \quad (3.3)
\]
The posterior distribution over functions can be obtained by restricting the joint prior distribution on the observations. Then we arrive at the key predictive equations for GPR

\[ f_\star | X, y, X_\star \sim N(\overline{f}_\star, \text{cov}(f_\star)), \] (3.4)

\[ \overline{f}_\star = E[f_\star | X, y, X_\star] = K(X_\star, X)[K(X, X) + \sigma_n^2 I]^{-1}y, \] (3.5)

\[ \text{cov}(f_\star) = K(X_\star, X_\star) - K(X_\star, X)[K(X, X) + \sigma_n^2 I]^{-1}K(X, X_\star). \] (3.6)

We then focus on explaining how to use GPR with given location samples to reconstructed the estimated trajectory. A trajectory can be considered as the path that the user follows through space as a function of time. Specifically, we have \( n \) location samples from \( x_1 \) to \( x_n \), each of which can be represented by a two-dimensional points \( x_i = (x_i, y_i) \). Then \( X \) is the sampled date set for all \((x_i, y_i)\)s. According to what we have explained, the user’s track can be represented by generated GPR functions which is determined by a covariance function and a mean function. In the case that there is only one test point \( x_\star \), we let \( k(x_\star) = k_\star \) denote the vector of covariances between the test point and the \( n \) training points. Then for a single test point \( x_\star \), Equation 3.5 and 3.6 can be reduced to

\[ \overline{f}_\star = k_\star^\top (K + \sigma_n^2 I)^{-1}y, \] (3.7)

\[ V(f_\star) = k(X_\star, X_\star) - k_\star^\top (K + \sigma_n^2 I)^{-1}k_\star. \] (3.8)

On obtaining Equation 3.7 and 3.8, we further propose the following Algorithm 1 for a single test case, in which \( \text{cholesky}(K + \sigma_n^2 I) \) is the Cholesky decomposition on the matrix of \( K + \sigma_n^2 I \), and \( \backslash \) denotes the matrix left division. The implementation addresses the matrix inversion required by Equation 3.7 and 3.8 using Cholesky factorization. For multiple test cases lines 3 \( \sim \) 6 are repeated. In our case, \( X \) is time space of the training set, \( y \) is the set of observed target values (location samples), \( k \) is the covariance function, \( \sigma_n^2 I \) is the noise, and \( x_\star \) is the testing data. The outputs are as follows. \( \overline{f}_\star \) is the mean predicted value (predicted location of \( x_\star \)), \( V[f_\star] \) is its variance, and \( \log p(y|X) \) is the marginal likelihood. A more detailed explanation can be referred to our previous work [42].

### 3.2.3 Switching Location Sensing Methods

As mentioned, it is well-known that GPS cannot function properly indoors. To expand the coverage areas, SensTrack switches between GPS and the network-based localization
through the wireless connection. Basically, we want to use GPS outdoors and the network-based localization indoors, and thus it is important to decide when to switch. Initially, SensTrack starts in the GPS mode and periodically executes a WiFi scan. When it detects the GPS signal loss as well as an active wireless network connection, SensTrack turns into the WiFi mode. If GPS becomes available again, and the phone loses the WiFi connection or the accuracy of location samples provided by the network decreases significantly, SensTrack switches back into the GPS mode.

We note that there are two conditions satisfied to switch the location sensing method: the current method fails to obtain location samples, and the other method is guaranteed to work, which prevents from switching between the two modes too often. Frequently changing location sensing mechanism can be very energy consuming, because the high-power components associated with both location providers need to be active. In some cases, both of the two methods are available when the user passing by some buildings. According to our rules, we should not change SensTrack’s working mode, since in these situations the wireless connection tends to be unstable and short. In other cases, none of the two methods are available if we simply lose the GPS satellite signal outdoors. Our rules can also avoid the unnecessary switching in these cases. It is also worth mentioning that SensTrack stops collecting the sensor hints when it switches into the WiFi mode. In another word, we passively receive location updates in this mode. The reason is that, unlike GPS, when we request the location information, the WiFi localization technology cannot respond within a tolerable delay. It means that even if we apply the sensor hints to sense the location adaptively, we cannot obtain a location sample timely in the WiFi mode. Therefore, considering the WiFi localization updates the location less frequently than GPS, we decided not to waste energy on the acceleration sensor and the orientation sensor.

**Algorithm 1** Predictions($X,y,k,\sigma_n^2,x_*$)

1. $L = \text{cholesky}(K + \sigma_n^2 I)$
2. $\alpha = L^\top \backslash (L \backslash y)$
3. $f_s = k_s^\top \alpha$
4. $v = L \backslash k_s$
5. $V[f_s] = k(x_*, x_*) - v^\top v$
6. $\log p(y|X) = -\frac{1}{2} y^\top \alpha - \sum_i \log L_{ii} - \frac{n}{2} \log 2\pi$
7. return $(f_s, V[f_s], \log p(y|X))$
3.2.4 Utilizing Sensor Hints

**Orientation**

SensTrack employs the orientation sensor as a detector of turning points when the user is moving. The idea is that there is no need to record the user’s location if he/she is in a steady movement without changing direction. For a sliding window of size $T$, SensTrack collects the readings of the orientation sensor, and computes the changes in direction. If user’s moving direction changes dramatically (greater than the threshold $\theta$), a location sensing of the user’s current location is executed. Considering the readings from the orientation sensor is approximately continuous, the window size $T$ should be larger enough to observe the potential direction changes. Table 3.1 shows the effect of the window size $T$. In our experiments, $T$ was set to be 5s because it would lose some turns of the trajectory for smaller window size. On the other hand, a larger window size is not necessary as it requires more memory and computation, which in turn requires more powerful hardware. The user can also decide the threshold $\theta$, the other key parameter, according to their expectations on accuracy. Table 3.2 presents the number of missing turning points for different values of $\theta$. Roughly speaking, SensTrack is more sensitive with a smaller $\theta$. However, a too small $\theta$ may cause redundant detections of the trajectory’s turns (false positives) if we consider the noises in the readings from the sensor, which potentially wastes energy in sensing locations at those false turning points.

<table>
<thead>
<tr>
<th>$T$ =</th>
<th>1s</th>
<th>3s</th>
<th>5s</th>
<th>7s</th>
</tr>
</thead>
<tbody>
<tr>
<td>key turning points</td>
<td>4 misses</td>
<td>1 miss</td>
<td>0 miss</td>
<td>0 miss</td>
</tr>
</tbody>
</table>

Table 3.1: Effect of window size $T$

<table>
<thead>
<tr>
<th>$\theta$ =</th>
<th>45°</th>
<th>60°</th>
<th>75°</th>
<th>90°</th>
</tr>
</thead>
<tbody>
<tr>
<td>key turning points</td>
<td>0 miss</td>
<td>1 miss</td>
<td>3 misses</td>
<td>4 misses</td>
</tr>
</tbody>
</table>

Table 3.2: Effect of threshold $\theta$

**Acceleration**

The acceleration sensor in a mobile device has been widely used in many existing location sensing systems, in which it acts as a binary sensor to detect user movement or non-movement. We notice that distance is theoretically a simple integral of speed, which in turn
CHAPTER 3. SENSTRACK: ENERGY-EFFICIENT LOCATION TRACKING

is an integral of acceleration. Unlike most prior works, we do not limit the acceleration sensor just to be the user’s movement detector, rather explore the possibility of calculating the distance that the user has traveled and the speed that the user is moving at.

It should be noted that the readings of the acceleration sensor on a moving device are usually noisy, especially when the user is walking. Activities with higher speed, like biking and driving, actually are more stable, whereas the movement of a pedestrian is always fluctuating. It often overestimates distance when the user is holding the phone in his/her hands, and underestimates distance when sitting quietly on a cushioned car seat [47]. When calculating the integrals, errors caused by the noise in the sensing data are accumulated. However, we argue that the estimated distance and speed obtained as integrals of acceleration are still useful even if they are inaccurate, because the location and velocity information provided by GPS can help us to calibrate the calculation. Once the estimated distance or the estimated speed exceeds the thresholds, specifically $D$ and $v$, SensTrack activates GPS to sense the current location and speed. The thresholds can be set based on the accuracy requirement or the user’s moving patterns. For example, for a pedestrian, usually the moving speed can be no more than $10\text{m/s}$ and should not be negative, and the accuracy requirement is usually higher. Moreover, the calibration of calculating the integrals can also be done when GPS is activated at the turning points.

3.3 Evaluation

3.3.1 Data Collection and Methodology

We evaluated SensTrack using a real data set collected from a Google Nexus S phone carried by a mobile user walking in our university campus. The phone is equipped with an integrated GPS, an WiFi sensor, an accelerometer, and an orientation sensor. We implemented a SensTrack prototype on Android 4.0 (API level 14). During its runtime, the prototype continuously collects data from the acceleration sensor and the orientation sensor at default rate of the system service (SENSOR_DELAY_NORMAL) in Android OS. When the GPS signal is available, a location listener is registered to request location updates from GPS periodically. Meanwhile, the prototype always tries to initiate and maintain a WiFi connection, which can be used to record the location updates from the network-based location provider. In our experiments, a PC server was used to further analyze the data collected by the smartphone and filter the GPS and WiFi location samples with the given parameters.
The trajectory reconstruction algorithm based on GRP was also implemented on the server side, which uses the filtered and valid location samples to predicted the original trajectory. For most of the presented results, our settings were $T = 5\text{s}$, $\theta = 45^\circ$, $D = 100\text{m}$, $v = 8\text{m/s}$, and a prediction was made if the time gap between two successive GPS samples is greater than 15s.

We also compared SensTrack with the naive approach, in which GPS is the only way to obtain location information and the GPS sensor is kept to be activated during the whole tracking period. Unlike SensTrack, which samples the GPS location actively, the naive approach is a passive method that records all the valid location updates from GPS. We conducted the experiments on the same real path for several times, which started from outdoor environment, came into a building, and then ended indoors. The total length of the path is around 1.1km. The results show that, without significantly losing the accuracy of tracking, SensTrack effectively reduce the number of GPS samples and the time that the GPS sensor needs to be turned on.

### 3.3.2 Accuracy

We first present the tracking results by SensTrack and the naive approach. Despite the tracking service maintained, the trajectory shown in Figure 3.1(a) ended once the user entered the building since the signals from GPS satellites were blocked by the building, which indicates the performance of GPS largely depends on the working condition. Compared to the naive approach, SensTrack demonstrates a reasonably better performance. Figure 3.1(b) shows that the trajectory reconstructed by SensTrack has a similar outdoor part, meanwhile it has the indoor part that the original one does not have. Although the indoor part of the second trajectory may be not that accurate given the limitation of WiFi localization technology, it is still good to have a approximate trajectory.

<table>
<thead>
<tr>
<th></th>
<th>recorded locations</th>
<th>predicted locations</th>
<th>average error</th>
</tr>
</thead>
<tbody>
<tr>
<td>SensTrack</td>
<td>38 samples</td>
<td>24 predictions</td>
<td>3.128m</td>
</tr>
<tr>
<td>GPS trace</td>
<td>568 samples</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.3: Average error of predicted locations

As previously stated, the resulting trajectory generated by SensTrack consists of two kinds of points: the sampled locations and the predicted locations. To evaluate the accuracy of SensTrack, we took the GPS trace as the ground truth and calculated the average error
CHAPTER 3. SENSTRACK: ENERGY-EFFICIENT LOCATION TRACKING

of the predicted locations. For every prediction, we computed the difference between the predicted location and the real location in the GPS trace at the same time. The result shown in Table 3.3 proves that SensTrack can achieve a high accuracy. The average error of the predictions is 3.128 meters, which is quite acceptable (GPS can achieve an accuracy of 5 meters in good signal conditions). It should be noted that even the GPS trace may not be the real path that the user has taken, because the performance of GPS depends on a number of factors such as the user’s position, time, surroundings, weather, etc, which means that the GPS trace itself can be inaccurate. Another result from Table 3.3 is that the naive approach recorded 568 samples over the testing path, although some of them may be unnecessary as discussed earlier. It is worth mentioning that, whether a sample is necessary should be decided case by case. For different scenarios, the ideal minimal distance (threshold) between two valid samples can vary significantly. We can adjust the number of necessary samples by setting the granularity between successive samples and filtering the recorded samples accordingly. In our experiments, the number of necessary samples does not affect the total number of GPS samples as the naive approach passively received every sample, and the granularity between successive samples cannot reflect the error of reconstructed trajectory.

3.3.3 Energy Efficiency

In modern mobile devices, the GPS receiver usually consume much more power than the accelerometer and the digital compass. For example, our testing device, a Google Nexus S phone, is equipped with a BCM4751 integrated GPS receiver (produced by Broadcom), a KR3DM 3-axis accelerometer (produced by STMicroelectronics), and an AK8973 3-axis electronic compass (produced by Asahi Kasei Microdevices). With the battery supply (3.7 volt), the power consumption (in terms of current) of the accelerometer is 0.23 mA; and the current consumption of the compass is 6.8 mA; however, the current consumption of the GPS receiver can be as much as 80 mA. To demonstrate the energy efficiency of SensTrack, we present that SensTrack can significantly reduce the number of needed GPS samples and the time that the GPS sensor needs to be activated. We did not measure the actual energy consumption of SensTrack, since we thought it is unnecessary. For different hardware, the power consumption varies, and thus the energy consumption of SensTrack on a specific hardware model only provides limited information. Therefore, it is convincing and sufficient for us to show the relative energy efficiency of SensTrack to the naive approach by comparing
the number of required sampling and the activated time of the GPS receiver.

Figure 3.5: Comparison of the energy efficiency

Figure 3.5 shows that compared to the naive approach, SensTrack only needs 7% GPS samples for the described path, and the time of the GPS sensor being active is decreased by nearly 90%. The naive approach almost updated the user’s location every second, and the GPS sensor was kept to be activated even when the user entered the building and lost the GPS satellite signals. SensTrack on the contrary only selectively activated the GPS sensor at some separate locations, and turned the GPS sensor off once the device lost the satellite signals and had an active WiFi connection. It should be pointed out that the energy efficiency of SensTrack depends on the user’s movements and the path that the user takes. If the user’s movement is very unstable and the direction changes frequently, SensTrack inevitably activates the GPS sensor more frequently, and thus consumes more energy.

3.3.4 Energy-Accuracy Trade-off

By intelligently managing the energy and localization accuracy trade-off, the battery life of a mobile device can be significantly extended, which is of great importance for the smartphone users. Since the required localization accuracy varies with locations, there is significant potential to trade-off the accuracy and the energy consumption based on the application’s needs and different working scenarios.
As mentioned before, we take the GPS sampling rate as a representative of SensTrack’s power consumption. Figure 3.6 demonstrates the trade-off between sampling rate and accuracy, which SensTrack presents under different configurations. Even though there exists some bias, we can observe a clear trend that a higher accuracy requires a higher GPS sampling rate, which means more power consumption. On the other hand, Figure 3.6 does not present a strict monotonicity. A higher energy consumption does not necessarily indicate a higher accuracy. For example, it only requires 6% samples to achieve a higher accuracy (average error is 2.66 m), whereas 11% samples are needed to produce a relatively lower accuracy (average error is 3.02 m). This is because the error of one prediction not only depends on the GPS sampling rate but also depends on the performance of the reconstruction algorithm. For GPR in our case, if the location samples have higher covariances between each other and are uniformly distributed on the path in time space, the algorithm can produce better results and achieve a higher accuracy. Therefore, besides the sampling rate, the actual samples themselves collected by the system have a huge impact on the results. The samples that have similar covariances between every two successive samples are more likely to produce highly accurate predictions.
3.3.5 Transmission Overhead

There is no doubt that exploiting network-based localization technology to obtain approximate locations would incur some extra network transmissions. To measure the extra traffic, we recorded the traffic loads of SensTrack and the baseline. As the baseline, there only maintains a valid wireless network connection. To be clear, we did not include the uploading of location samples into the transmission overhead, because unlike the indoor location sensing, the uploading process does not need to be done in real time.

<table>
<thead>
<tr>
<th></th>
<th>received</th>
<th>transmitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.38 packet/s</td>
<td>0.88 packet/s</td>
</tr>
<tr>
<td>SensTrack</td>
<td>0.94 packet/s</td>
<td>0.81 packet/s</td>
</tr>
</tbody>
</table>

Table 3.4: Average WiFi traffic

![Figure 3.7: WiFi traffic of SensTrack](image)

Table 3.4 presents the average numbers of the received and transmitted packets during the tracking process. For both SensTrack and the baseline, the average numbers of the transmitted packets were close. Although SensTrack theoretically should transmit more packets as it requests location information through the wireless link, the result is within a normal error range. On the other hand, SensTrack received more than twice as many packets as the baseline did. We argue that even if the number of received packets increases, the total transmission overhead may not be intolerable, because the size of received packets that
contains only the location information should be small. Moreover, since the WiFi connection is usually free, there is no need to worry about the wireless network traffic. Another point is that communicating with the access points consumes less energy than communicating with the GPS satellites. Figure 3.7 further shows SensTrack’s traffic pattern, which matches the result in Figure 3.2. SensTrack had WiFi traffic in the time intervals of strong WiFi signals (201s ∼ 216s, 335s ∼ 368s, 387s ∼ 398s, 537s ∼ 558s). After entering the building at 588s, SensTrack continuously transmitted and received packets.

3.4 Further Discussion

3.4.1 Multiple Mobility Patterns

Although our work focuses on the pedestrians, it can be easily extended on multiple mobility patterns, such as running, biking, driving, etc, which are often with higher speeds. Intuitively these movements are more stable, and thus the trajectories are likely less complex, and thus the sensors on smartphones can easily capture the features of the path. Therefore, our approach at least paves the road of designing the efficient tracking service for multiple mobility patterns. However, given the characteristics of different movements, modifications should be carefully considered.

3.4.2 Energy Consumption of Accelerometer and Orientation Sensor

In this work, to make our point clear, we assume a continuous sampling of the acceleration sensor and the orientation sensor, which may cause unnecessary energy cost. It is not necessarily the case. Given that the energy-efficiency is a major goal of our design, users can further employ a low duty cycle on the usage of the acceleration sensor and the orientation sensor. Since the high speed movements are more stable, a low duty cycle can still allow the sensors to capture the features of the users’ movements.

3.4.3 Other Indoor Localization Technologies

Our work chose the network-based method, which is mainly based on the WiFi positioning system, as our indoor localization approach. The primary reason is that the implementation of this method is already provided as APIs in Android platforms (since API level 1). Other methods for the indoor localization can also be employed such as the specialized real-time
locating systems (RTLS) [17] or the inertial measurement unit (IMU)-based navigation systems [59]. However, many of these methods also require a costly infrastructure or additional hardware, which hardly satisfy the need for a cost-effective solution. On the other hand, indoor localization is not our main concern in this work, rather it is a supplementary of GPS to extended the coverage of SensTrack.

3.5 Summary

In this chapter, we have presented how to improve smartphones’ energy efficiency by smartly managing local resources. A novel location tracking service, SensTrack, has been proposed. We first discussed the limitations of the traditional GPS-based approach and opportunities of improvements. Next, the detailed design of SensTrack was presented including: the trajectory reconstruction algorithm based on the Gaussian Process Regression, the rules of switching between two location sensing methods, and the principles for exploiting the sensor hints. We then used the real traces to evaluate the performance of SensTrack, which shows that SensTrack can significantly reduce the usage of GPS and generate accurate tracking results. The design of SensTrack and evaluation presented above reveal several interesting challenges which remain for future work including resilient accelerometer data processing, tracking for multiple mobility patterns, and joint optimization of energy and accuracy.
Chapter 4

Energy-Efficient Computation
Offloading

The previous chapter has shown that energy can be saved on mobile devices through local computation using the example of location tracking service on smartphones. In this chapter, we discuss how to further improve mobile devices’ energy efficiency by utilizing the cloud resources and offloading the computation adaptively to the network environment.

4.1 Motivation

Cloud computing and mobile systems perfectly suit each other: mobile platforms need the “infinite” computation ability and the “unlimited” storage from cloud to achieve better availability and reliability; on the other hand, cloud computing is also looking for its killer application to reinforce its dominant position. As an approach to bridge cloud resources and mobile platforms, offloading can alleviate the computation burden of mobile devices at the cost of extra transmission. In order to extend the battery life of mobile devices by offloading, the tradeoff between the energy consumption for computation and that for transmission needs to be carefully considered.

Before utilizing cloud resources to assist the execution of certain application, one needs to know how to partition the application into the local part and the cloud part. Normally, decoupling the target application is not a difficult task for desktop PCs in local area networks (LANs) that visit cloud resources through Internet, since the wired connection is fast, stable,
persistent, and at low cost. In most cases, a determinate part of computation can be shifted from PCs to the cloud, and the traffic generated during the migration is not a serious concern for LANs.

When it comes to mobile or wireless networks, the situation is totally different, and the problem becomes much more challenging. The key observation is that, mobile devices access Internet via WiFi or cellular networks, and the connection between them experiences spacial or temporal instability. For cellular data networks, the signal strength, which may vary significantly across different areas, has a direct impact on radio energy consumption, since both radio power and data rate are affected [53]. While WiFi radios also exhibit significant variation in energy cost per bit at different locations, this is mainly due to the variation in data rate rather than radio power [25]. Even for the WiFi users who keep stationary, the network condition still changes from time to time under different circumstances. For example, if a WiFi network has only one effective connection, the link quality is likely to be good, however, when multiple devices communicate with the access point simultaneously, they may suffer from severe interference. On the other hand, poor network connection can make computation offloading infeasible, since both of the two goals of offloading (saving energy and improving performance) fail under certain conditions: higher radio power leads to more energy consumption, and lower data rate increases the transmission delay and thereby lengthens the elapsed time for the application’s execution. When offloading is no longer applicable or beneficial, computation should be kept locally. Therefore, it is critical to make offloading scheme adaptive to the changing network condition. Unlike offloading from PCs through wired connection, for mobile platforms, the offloading decisions should be made dynamically together with the transmission decisions during the process according to the quality of wireless connection.

To make optimal computation offloading decisions, we need to answer two key questions: what to offload; and when to offload. For traditional offloading from PCs through wired connection, we only need to consider the first question, and it is not difficult to find the answer, since the entire computation can be easily shifted to remote servers with very low overhead. However, it is not a straightforward issue for mobile platforms to decide what and when to offload. Previous studies have shown that answering either of the two questions requires considerable efforts. In this chapter, we argue that, for mobile systems, these two questions should be considered jointly, as one can have direct impact on the other in terms of energy consumption and performance due to the nature of wireless channel’s instability.
Therefore, our goal is to find an optimal schedule of both offloading and transmission for mobile platforms, which can adapt to the highly-variable network environment.

4.2 Measurement

In this section we conduct a measurement study with the following goals:

- Compare the energy efficiency of local execution and offloaded execution for the target applications, and verify the potential benefit from computation offloading.

- Identify the variation in performance and energy efficiency of computation offloading under different network conditions.

- Build the appropriate power model of the test device for computation and transmission during offloading.

4.2.1 Devices, Tools and Configurations

Our experiments were performed on a 32 GB Google Nexus 7 tablet [7] that can be connected to Internet through WiFi and mobile networks. The test device runs the Android 4.2.2 OS, and has a battery of 4325 mAh at 3.7 volts. In order to measure the actual power consumption, we used a digital multimeter to record the current transferred between the battery and the tablet since the supply voltage can remain stable in a long period of time. The current can be sampled once per second with the accuracy of 10 mA, and be recorded by a software. We unplugged the battery from the device, and wired them up with the digital multimeter as a serial connection in the circuit. Besides recording the current consumption, we also collected the execution traces of our test applications. We developed a software profiler to profile the CPU usage and the wireless traffic on the mobile device. Another analysis tool, Application Resource Optimizer (ARO) [49], was also adopted in our experiments, which can capture the packets and analyze the radio states during the execution. The two profilers were running as the background processes when our testing mobile applications executed.

Since one of our goals is to investigate the energy impact of dynamic wireless networks during computation offloading, we need to be able to emulate different network conditions. In our experiments, a desktop PC that runs the Linux OS was used to bridge the router...
CHAPTER 4. ENERGY-EFFICIENT COMPUTATION OFFLOADING

that has the access to Internet and the wireless AP which the test tablet was connected to. As for emulating the network properties, we used the netem tool [11] on the desktop PC, which provides the network emulation functionality and is available in current Linux systems. Its features include wide area network delays with different delay distribution, packet loss, packet duplication, packet corruption and packet re-ordering. In order to ensure that the variation of energy consumption is caused by the changing network conditions, the interferences from other factors should be eliminated as much as possible. First, we disabled the CPU frequency auto-scaling on the testing mobile device, and kept the CPU working at a fixed frequency. Second, as the screen usually takes up most of the consumed energy on a mobile device, the auto adjustment of screen brightness was also disabled. At last, we killed all the irrelevant and unnecessary processes before profiling the execution of our test applications.

4.2.2 Test Applications

Two realworld mobile applications were selected as the test applications to conduct the experiments. The first application is an open source chess game, DroidFish [4], which has an integrated chess engine in the program and is also able to configure a network chess engine as shown in Figure 4.1. A chess engine is a computer program that analyzes chess positions and makes decisions on the best chess moves. Although the chess engine decides what moves to make, it typically does not interact directly with the user, instead, it communicates with the graphical user interface (GUI) via the Chess Engine Communication Protocol. The chess game has three game modes: player vs player, player vs computer, and computer vs computer. To test this application, we set the chess game into computer vs computer mode, in which no human input is required, and the program communicates with the chess engine and uses it to search the best moves. In this mode, a significant amount of computation needs to be done by the chess engine, and only a little communication is required since both the chess positions and the chess moves can be expressed as plain text. Therefore, the chess game is promising for energy saving from computation offloading as it is very computation-intensive and requires little data transmission. The possibility of using different chess engines in the application allow us to compare the local execution with the offloaded execution: when the chess game works with a local chess engine, the computation are executed locally on the mobile device, while the computation can be offloaded remotely if a network chess engine is configured. In our experiments, the network chess engine was
configured on a laptop with Core i3-2310M CPU and 4 GB RAM which was connected to Internet through a wireless AP.

We chose OnLive, one of the pioneering commercial cloud gaming platforms, as the other test application. Advances in cloud technology have expanded to allow offloading not only of traditional computations but also of such more complex tasks as high definition 3D rendering, which turns the idea of cloud gaming into a reality. Figure 4.2 shows the various functions and modules required by a cloud gaming system. As can be observed, a players commands must be sent over the Internet from the its thin client to the cloud gaming platform. Once the commands reach the cloud gaming platform they are next converted into appropriate in-game actions, which are interpreted by the game logic into changes in the game world. The game world changes are then processed by the cloud system graphical processing unit (GPU) into a rendered scene. The rendered scene must be compressed by the video encoder, and then sent to a video streaming module, which delivers the video stream back to the thin client. Finally, the thin client displays the video frames to the player. To test OnLive, we ran a cross-platform game, Osmos [13], through an OnLive mobile client as well as locally on our test device, as the game is also sold as a mobile app in Google Play Store.

Figure 4.1: Framework of the chess game
4.2.3 Local Execution vs Offloaded Execution

To prove that computation offloading can bring potential energy savings on mobile devices, we now compare the local execution and the offloaded execution of the two test applications. For the chess game, we profiled the current consumption and the execution trace when the program used the local chess engine and the network chess engine respectively. For the second application, we first ran the mobile game as an independent app on the test tablet. As its comparison, we then played the same game integrated in OnLive. As we cannot guarantee that every time we have exactly the same inputs in the game, in order to eliminate the inference from human interactions, we did not generate any user input when we recorded the current consumption and the execution trace during the video playback.

Figure 4.3(a) shows the CPU usage and the total traffic during one execution of the chess game, which provides the strong evidence that, for computation-intensive tasks, computation offloading can mitigate the burden on CPUs at the low cost of extra data transmission. From Figure 4.3(b) we can see that it significantly reduces the power consumption if the test device offloads the heavy calculation to the cloud. It should be noted that, at the end
CHAPTER 4. ENERGY-EFFICIENT COMPUTATION OFFLOADING

Figure 4.3: Comparison between local execution and offloaded execution for the chess game of the chess game, the data transfer rate of the offloaded execution increases, and the CPU usage of the local execution decreases. The reason is that, there are usually only a few pieces left on the board when the chess game comes to the end, and it needs less computation to decide the best move at that time, and thus the network engine responds more quickly and the communication becomes more frequently.

Figure 4.4: Comparison between local execution and offloaded execution for OnLive

The previous results prove that computation offloading has the potential to save mobile devices’ energy for computation-intensive applications such as the chess game. However, the execution traces for the mobile game and OnLive show us the different results. In Figure 4.4, both the CPU usage and the current consumption are higher when the mobile game
executed in the OnLive platform. This can be explained by the fact that the two programs have totally different implementations. When the mobile game running as a installed app, it needs more GPU processing to render the game scenes, while when it runs in OnLive it requires more CPU processing to decode the received video. Moreover, the goal of OnLive is to provide ubiquitous gaming experience that normally cannot get from mobile devices rather than saving their batteries, and the test game itself does not belong to the kind of games that require high performance hardware, which OnLive is mainly designed for. Therefore, it is not surprising that OnLive consumes slightly more power than the mobile game’s local execution.

<table>
<thead>
<tr>
<th></th>
<th>Chess</th>
<th>Mobile Game/OnLive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>local</td>
<td>offloaded</td>
</tr>
<tr>
<td>avg power</td>
<td>1792.2 mW</td>
<td>347.5 mW</td>
</tr>
<tr>
<td>avg CPU usage</td>
<td>83.3%</td>
<td>8.4%</td>
</tr>
<tr>
<td>avg data rate</td>
<td>N/A</td>
<td>1.944 KB/s</td>
</tr>
<tr>
<td></td>
<td>local</td>
<td>offloaded</td>
</tr>
<tr>
<td>avg power</td>
<td>1812.6 mW</td>
<td>1848.2 mW</td>
</tr>
<tr>
<td>avg CPU usage</td>
<td>29.1%</td>
<td>55.7%</td>
</tr>
<tr>
<td>avg data rate</td>
<td>N/A</td>
<td>288.89 KB/s</td>
</tr>
</tbody>
</table>

Table 4.1: Local execution vs Offloaded execution

Table 4.1 lists the numerical results of this experiment, which confirm the previous discussion. One thing should be pointed out is that, when the two applications both execute locally, the chess game consumes less power than the mobile game even though the chess game requires much higher CPU utilization. The difference in power consumption comes from the other components of the mobile device: the mobile game puts more calculation on GPU, updates the screen much more frequently, and uses more memory than the chess game.

4.2.4 Computation Offloading under Dynamic Wireless Networks

In this section, we investigate how the performance and energy efficiency of computation offloading change when the network condition is getting worse. As described earlier, we used a desktop PC as the network bridge between the test device and Internet. We first test the two applications under the normal connection, then we increase the network latency and the packet loss rate to emulate the dynamic wireless connection.
Impact of Delay

Our measurements show that, when the network latency is increased, the power consumption does not vary much for the two test applications. Although the energy efficiency of computation offloading remains stable, we argue that the increasing delay may hurt the performance of the offloaded tasks. Higher network latency implies that more time is needed to get the response when the application interacts with the cloud. For some applications such as Email, the delay can be masked so that it does no harm to the user experience. For example, in the chess rules, each player has a fixed interval of time to think and make the move; when the network latency increases, the network chess engine has less time to search the best move (less thinking time) as it takes longer to send back the result, but it does not affect the user experience as long as the total delay does not exceed the whole thinking interval. However, for the delay-sensitive applications, the high network latency can result in a significant performance degradation. Studies on traditional online gaming systems have found that different styles of games have different thresholds for maximum tolerable delay [21]. Table 4.2 summarizes the maximum delay that an average player can tolerate before the Quality of Experience (QoE) begins to degrade.

<table>
<thead>
<tr>
<th>Example Game Type</th>
<th>Perspective</th>
<th>Delay Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Person Shooter (FPS)</td>
<td>First-Person</td>
<td>100 ms</td>
</tr>
<tr>
<td>Role Playing Game (RPG)</td>
<td>Third-Person</td>
<td>500 ms</td>
</tr>
<tr>
<td>Real Time Strategy (RTS)</td>
<td>Omnipresent</td>
<td>1000 ms</td>
</tr>
</tbody>
</table>

Table 4.2: Delay tolerance in online gaming

To measure the actual impact that the network delay has on OnLive, we recorded the tablet’s screen (the frame rate was approximately 15fps) when we ran the game in OnLive, and analyzed the video frame by frame. Figure 4.5 shows how the screen updated after one touch was detected on the screen. The left side lists OnLive’s responses with the normal connection, while the right side presents the updated frames after we increased the delay by 150ms. With the better wireless connection, it only needs 5 frames to get the response of that action from OnLive, whereas 8 frames are needed for the worst case. As shown in Table 4.3, the average response time increases as the delay gets higher, which indicates that the performance of OnLive degrades.
Figure 4.5: Captured frames in OnLive with low and high network latency

Table 4.3: Average number of frames updated before the response arrived

<table>
<thead>
<tr>
<th>Added delay</th>
<th>Num of frames updated in average</th>
<th>Represented response time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5 frames</td>
<td>0.33 s</td>
</tr>
<tr>
<td>50 ms</td>
<td>5.7 frames</td>
<td>0.38 s</td>
</tr>
<tr>
<td>100 ms</td>
<td>6.5 frames</td>
<td>0.43 s</td>
</tr>
<tr>
<td>150 ms</td>
<td>7.5 frames</td>
<td>0.50 s</td>
</tr>
</tbody>
</table>

Table 4.3: Average number of frames updated before the response arrived
Impact of Packet Loss

We now examine the influence of packet loss rate, another important parameter in networking analysis. In our measurements, we find that the power consumption for the two test applications when the packet loss rate varies does not change significantly. However, it is not necessary that the energy efficiency of computation offloading remains unchange. The packet loss can have great negative effects on the quality of network communication, which means that it may cost more energy to transfer the same amount of data successfully. In other words, the effective throughput on the wireless link drops when the packet loss rate becomes higher.

![Figure 4.6: Impact of packet loss](image)

(a) Unnecessary packets in the chess game
(b) Transfer rate of OnLive

We captured the transferred packets in the data transmission during the computation offloading for the two test applications. The chess game’s GUI communicates with the network chess engine through a TCP connection. We classify the packets that should not appear in the normal data transfer as unnecessary packets, such as duplicate ACKs, timeout retransmissions, fast retransmissions (after 3 duplicate ACKs), ACKs of unseen segment and other special packets. For each run of the chess game, we counted the total number of packets and the number of unnecessary packets, and calculated the percentage of unnecessary packets. As shown in Figure 4.6(a), with the normal wireless connection there are only 0.1% unnecessary packets, whereas the abnormal traffic takes up 2.4% of the total traffic if the packet loss rate is increased to 1.5%. Different from the chess game, OnLive streams the gaming videos to users via UDP. We measured the data transfer rate between
the mobile client and the OnLive’s cloud sever instance. Figure 4.6(b) presents the mobile client’s average downloading data rate under different packet loss rates, which shows that the downloading rate drops by 24.1% when the wireless connection loses more packets, although the data rate remains higher than 200KB/s. It should be pointed out that the decreasing data rate not only implies the lower energy efficiency, but it may also hurt the application’s performance (in OnLive’s case, the image quality of the gaming video may be affected).

4.2.5 Power Model

In this section, we present the power model that characterizes the power consumption properties for the test device based on the traces collected in our measurements. Since the focus of this work is not to build an accurate power model, we simplify the power model proposed by L. Zhang et al. [62]. We only care about the power consumption from the components that are related to computation offloading, and thus our model mainly considers the power consumption of CPU and WiFi as shown in Table 4.4. The CPU power consumption, as we kept the working frequency fixed in our experiments, is then strongly influenced by its utilization. As for the WiFi power consumption, similar to [62], the WiFi interface has low-power state and high-power state in our model. When the WiFi interface is idle or transmitting at a very low rate, it stays in the low-power state, while if the data rate is high, the WiFi interface works in the high-power state. Other components, such as the screen and other sensors, may also consume a significant amount of power. It should be noted that a more complex power model can achieve higher accuracy, but this simple model is sufficient for our purpose.

<table>
<thead>
<tr>
<th>Model</th>
<th>( \beta_u \times \text{util} + \beta_{cpu} \times \text{CPU}_{on} + \beta_l \times \text{WiFi}<em>l + \beta_h \times \text{WiFi}<em>h + \beta</em>{others} \times \text{Oth}</em>{on} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>Variable</td>
</tr>
<tr>
<td>CPU</td>
<td>util</td>
</tr>
<tr>
<td></td>
<td>( \text{CPU}_{on} )</td>
</tr>
<tr>
<td>WiFi</td>
<td>( \text{WiFi}_l )</td>
</tr>
<tr>
<td></td>
<td>( \text{WiFi}_h )</td>
</tr>
<tr>
<td>Others</td>
<td>( \text{Oth}_{on} )</td>
</tr>
</tbody>
</table>

Table 4.4: Power Model
4.3 Problem Formulation

The previous measurement study has shown that the performance and energy efficiency of computation offloading can be largely affected by the various network conditions. In this section, we give a formal description of the computation offloading scheduling problem in wireless networks. We will focus on making energy-efficient offloading decisions and assigning intervals for data transmission. Consider a set of $n$ jobs $J = \{j_1, j_2, ..., j_n\}$, with $m$ time slots to execute them $\{t_1, t_2, ..., t_m\}$. Each job consists of five elements $j_i = (c_i, t^a_i, t^d_i, d^s_i, d^r_i)$, where $c_i$ is the computation load, $t^a_i$ is the job’s release time, $t^d_i$ is the deadline for the job to be done, $d^s_i$ is the amount of data to be sent, and $d^r_i$ is the amount of data to be received. These demand/cost information can be predicted by forecast algorithms (such as the algorithms proposed in [29]).

Assume that we have the knowledge of wireless channel state for all the $m$ slots $NS = \{ns_1, ns_2, ..., ns_m\}$. The function $R(ns_{t_i})$ maps the channel state to the effective throughput in slot $t_i$ (the amount data can be transferred during this slot) considering the bandwidth and the packet loss. Given the power model presented in Section 4.2.5 and the network throughput, we are able to calculate the energy consumption $E_c(c)$ for the local execution of computation $c$, and the energy cost $E_t(d)$ for the wireless transmission of data $d$.

Our goal is to find an optimal schedule for all the jobs in terms of energy consumption with the guarantee of performance. Let $s_i = (d_i, [s^s_i, s^e_i], [t^s_i, t^e_i])$ denote the schedule for $j_i$, where $d_i$ is the offloading decision ($d_i = 0$ indicates executing $j_i$ locally, while $d_i = 1$ indicates offloading $j_i$ to the cloud), $[s^s_i, s^e_i]$ is the slot interval for sending data, and $[t^s_i, t^e_i]$ is the slot interval for receiving data. Note that we only care about the sending and receiving intervals for the jobs that are chosen to be offloaded. If a job is decided to be kept locally, we do not consider its schedule any more, since originally all the jobs are running on the mobile device. We apply sequential scheduling in this work, which means in every time slot at most one transmission can be scheduled. The reason is that, given the same network throughput and the same starting point, both sequential and parallel scheduling consumes the same amount of energy, while the former has better performance in terms of the finish time (all the parallel jobs have the same and long finish time, however, the sequential scheduled tasks have different and increasing finish times with the same maximum value). Assume that $F_c$ is the processing capability of the cloud server. Therefore, the problem can be formulated...
to find a schedule $S = \{s_1, s_2, ..., s_n\}$ that minimizes the total energy consumption

$$E_{total} = \sum_{i=1}^{n}\{(1 - d_i) \cdot E_c(c_i) + d_i \cdot E_t(d_i^s + d_i^e)\}. \quad (4.1)$$

The following constraints should be satisfied:

1. **Causality constraint:**
   \begin{equation}
   \forall i \in [1, n], t_i^{ss} \geq t_i^a; \quad (4.2)
   \end{equation}

2. **Performance constraint:**
   \begin{equation}
   \forall i \in [1, n], t_i^{se} \leq t_i^d; \quad (4.3)
   \end{equation}

3. **Data transmission constraint:**
   \begin{equation}
   \forall i \in [1, n], \sum_{k=t_i^{se}}^{t_i^{te}} R(ns_k) \geq d_i^s,
   \text{and} \sum_{k=t_i^{se}}^{t_i^{te}} R(ns_k) \geq d_i^e; \quad (4.4)
   \end{equation}

4. **Execution constraint:**
   \begin{equation}
   \forall i \in [1, n], t_i^{rs} - t_i^{se} \geq c_i/F_c; \quad (4.5)
   \end{equation}

5. **Energy saving constraint:**
   \begin{equation}
   \forall i \in [1, n], \text{if } d_i = 1, E_t(d_i^s + d_i^e) < E_c(c_i); \quad (4.6)
   \end{equation}

6. **Feasibility constraint:**
   \begin{equation}
   \forall i \neq j, [t_i^{ss}, t_i^{se}] \cap [t_j^{ss}, t_j^{se}] = \emptyset,
   [t_i^{ss}, t_i^{se}] \cap [t_j^{rs}, t_j^{re}] = \emptyset,
   [t_i^{rs}, t_i^{re}] \cap [t_j^{ss}, t_j^{se}] = \emptyset,
   \text{and} [t_i^{rs}, t_i^{re}] \cap [t_j^{rs}, t_j^{re}] = \emptyset. \quad (4.7)
   \end{equation}

The causality constraint (Equation 4.2) implies that a job cannot be scheduled before its release time. The performance constraint (Equation 4.3) follows that computation offloading should not cause performance degradation in terms of user experience even though
our goal is to save energy. Meanwhile, the release times and the deadlines can be set to ensure the dependencies among jobs. The data transmission constraint (Equation 4.4) allows the necessary amount of data to be transferred. The execution constraint (Equation 4.5) indicates that the cloud server need enough time to process the offloaded task. The energy saving constraint (Equation 4.6) guarantees that energy savings can be achieved from the scheduled offloading. Finally, the feasibility constraint (Equation 4.7) ensures that there is no conflict in the sequential scheduling.

The objective function consists of the computation energy cost for the local executed jobs and the transmission energy cost for offloaded jobs, which can also be represented as the difference between the total energy cost without offloading and the possible energy saving brought by offloading:

$$E_{\text{total}} = \sum_{i=1}^{n} E_c(c_i) - \sum_{i=1}^{n} \{d_i \ast [E_c(c_i) - E_t(d^s_i + d^r_i)]\}.$$  (4.8)

As the first part of Equation 4.8 is fixed, we focus on maximizing the energy saving part, and thus our goal is reformed as:

$$\text{maximize } E_s = \sum_{i=1}^{n} \{d_i \ast [E_c(c_i) - E_t(d^s_i + d^r_i)]\}$$

subject to constraints 4.2, 4.3, 4.4, 4.5, 4.6, 4.7.  (4.9)

This problem is challenging as each task has two intervals (sending and receiving) to schedule with the restriction of release time, deadline, and distance (processing time) in-between. The criteria of making offloading decision for a specific task is the amount of energy saving, which can vary significantly for different schedules under dynamic network environment. Moreover, it is possible that, for a single job, more than one schedule can achieve the same amount of energy saving. Considering the scheduling for multiple jobs, the problem becomes even harder, as each scheduled job directly affects the available time slots for the following jobs.

For the problem formulation, we have the following theorem:

**Theorem 1** The described offloading scheduling problem is a NP-hard problem.

**Proof.** In order to prove this optimization problem is NP-hard, we must prove the corresponding decision problem is NP-complete. Therefore, we must show that to decide
whether there is a schedule that satisfied all the previous constraints and can save at least $W$ units energy is NP-complete. Given an instance of the decision problem, a certificate that it is solvable would be a specification of the offloading decisions for each job, and the sending and receiving intervals for the offloaded jobs. We could then check that each job runs under the listed constraints, and the total saved energy is no less than $W$.

We now show that Subset Sum is reducible to this decision problem. Thus, consider an instance of Subset Sum with numbers $w_1, ..., w_n$ and a target $W$. In constructing an equivalent scheduling instance, one is struck initially by the fact that we have so many parameters to manage. The key is to sacrifice most of the flexibility, producing a simpler “skeletal” instance of the problem that still encodes the Subset Sum Problem. Assume that the cloud server has infinite processing ability ($F_c = \infty$), so that, for each job, no time is needed for processing it in cloud, and only the time for the data transfer is required. Another important assumption is that the wireless channel is in good state for the first $W$ time slots in which the link throughput is a constant, and it becomes extremely bad since then so that no job can be offloaded.

Let $S = \sum_{i=1}^{n} w_i$. Define jobs 1, 2, ..., $n$; job $i$ has a release time of 0, a deadline of $S$, and a duration of $w_i$ for the data transfer if offloaded (the processing time is omitted as we have assumed). For this set of jobs, we have the freedom to arrange them in any order, and they will all finish on time. Assume that each job can have exactly one unit energy saved in one time slot from offloading when the wireless channel remains in the good state. Now consider any feasible solution to this instance of the decision problem. Since the schedule saves at least $W$ units energy, there must be some job being offloaded for each of the first $W$ time slots (the wireless channel must not have any idle time for the first $W$ time slots), otherwise the total amount of saved energy can only be less than $W$ (no energy can be saved in the bad channel state, since offloading is no longer beneficial in this state). In particular, if jobs $i_1, ..., i_k$ are the ones that are offloaded in the first $W$ slots, then the corresponding numbers $w_{i_1}, ..., w_{i_k}$ in the Subset Sum instance add up to exactly $W$.

Conversely, if there are numbers $w_{i_1}, ..., w_{i_k}$ that add up to exactly $W$, then we can schedule these jobs to be offloaded in the first $W$ time slots, and the remainder are kept locally, which is a feasible solution to the decision problem.

Now we have proven the decision problem of the original optimization problem is NP-complete. Therefore, the described offloading scheduling problem is a NP-hard problem.
4.4 Offline Solution

To solve this problem, we use a dynamic programming technique. Let the array $T$ of length $m$ denote the usage of $m$ time slots by the scheduling algorithm: initially, all slots are available, and thus $T$ is set to be all 0s; if a slot is assigned to job $i$, we modify the corresponding element of $T$ with the job’s index. Also let $F(n, T)$ be the maximum energy saving that can be achieved after $n$ jobs have been scheduled with the given $T$. For a single job $i$, there may be multiple valid schedules that can save energy within the interval of its release time and deadline, and we assume $S(i)$ is the set of all these schedules. We have the following recurrence relation:

$$F(n, T) = \max\{F(n - 1, T), \max_{s \in S(n)}[E_s(n) + F(n - 1, T')]\}, \quad (4.10)$$

where $E_s(n) = E_c(c_n) - E_t(d^*_n + d^*_n)$, $s$ is a schedule in the set $S(n)$, and $T'$ is the update of $T$ according to the schedule $s$.

For boundaries, we have

$$F(1, T) = \begin{cases} 
0, & \text{if there is no valid schedule within } T; \\
\max_{s \in S(1)}\{E_s(1)\}, & \text{if task 1 can be scheduled within } T. 
\end{cases} \quad (4.11)$$

We propose Algorithm 2, which solves the scheduling problem recursively. In each iteration, the algorithm first checks if there is an opportunity to save energy for the current job with available time slots in $T$ (line 11). If the algorithm cannot find a schedule that saves energy, it drops the current job to execute locally (line 12-13); otherwise, it goes through all the possible schedules (including the one that drops the job) to get the schedule with the maximum energy saving, and updates $T$ accordingly (line 15-16). The boundary condition is checked when there is only one job left (line 1-9). In each iteration, let $M$ denote the maximum number of available time slots in the scheduling interval $[t^a_n, t^d_n]$ after the transmission units for job $n$ are arranged, which means that the size of the search space for the schedule of job $n$ is $M$. In one iteration, there are $M^2$ schedules that need to be checked, and for each schedule constant operations are performed to calculate the energy cost. When scheduling the jobs in the best case (for example, when all the jobs have non-overlapping intervals of release times and deadlines), the algorithm can find the unique best schedule in each iteration, and saves it once for the final schedule. As for the worst case, in each iteration all the $M^2$ schedules may have the same energy saving, and thus the algorithm needs to step into the next iteration to decide which schedule is the best, and the...
Algorithm 2 max-ESaving($n,T$)

1: if $n == 1$, then
2:   search the interval $[t^a_n, t^d_n]$ in $T$ and compute the corresponding $E_s(1) = E_c(c_1) - E_l(d^a_1 + d^r_1)$;
3:   if there is no valid schedule such that $E_l(d^a_1 + d^r_1) < E_c(c_1)$, then
4:     return 0;
5:   else
6:     return $\max_{s \in S(1)} \{E_s(1)\}$;
7:     update $T$ according to the selected schedule of task 1;
8:   end if
9: output the complete schedule $S$ according to $T$;
10: else
11:   search the interval $[t^a_n, t^d_n]$ in $T$ and compute the corresponding $E_s(n) = E_c(c_n) - E_l(d^a_n + d^r_n)$;
12:   if there is no valid schedule such that $E_l(d^a_n + d^r_n) < E_c(c_n)$, then
13:     return max-ESaving($n - 1,T$);
14:   else
15:     return $\max \{\max$-ESaving($n - 1,T$), $\max_{s \in S(n)} [E_s(n) +$-ESaving($n - 1,T')]\}$;
16:     update $T$ to $T'$ according to the selected schedule of task $n$;
17:   end if
18: end if
overall optimal cannot be found until it recurses to the one job case. In the worst case, the
algorithm is indeed doing the exhaustive search, and its recursion tree grows exponentially,
and thus it becomes very inefficient with the total running time of $O(M^{2^n})$.

### 4.5 Online Extension

The previous section has presented the offline solution which solves the problem with strong
assumptions. In this section, we discuss some practical issues that need to be considered in
the implementation design, and we further propose an online scheduling heuristic.

#### 4.5.1 Job Dependency and Data Sharing

Now we consider two kinds of relationships that can exist among the jobs. First, one can
observe the job dependency when a job needs to use the results of other jobs. If a job $a$
is dependent on another job $b$, job $a$ can only be executed after job $b$. In this case, it is
equivalent that job $a$ is released when job $b$ is done, and thus the release time of job $a$ is
decided by the schedule of job $b$. When this dependency exists, we should schedule job $b$
first, and then update the status of job $a$. Note that our goal is to choose and schedule the
offloaded jobs, and we do not care about scheduling the local jobs. Therefore, we do not
consider the dependency any more if either of job $a$ and $b$ is scheduled to run locally (in this
case, assume that the dependency is already considered by the fixed release time). Second,
other than the dependency, it is also possible that multiple jobs share the same data. When
one of the jobs is scheduled to be offloaded, the shared data must be transferred, and thus
all the following jobs that share the data can save this traffic. Therefore, when a group of
jobs use the same data, we should schedule from the earliest job, and if a job is scheduled
to run on the cloud side, we can then decrease the amount of data that needs to be sent for
all the following jobs.

#### 4.5.2 Wireless Channel Model

For the online scheduling, the status of the wireless channel is not known as a priori. To
describe a wireless channel, many Markov chain based models have been proposed, from
the two-state Gilbert-Elliott model [24, 27] to the hierarchical hidden Markov model [61].
Complex models can capture the higher order statistics of the wireless channel, but result in
a nearly exponential increase in model complexity [30]. In this work, we adopt a two-state channel model like the Gilbert-Elliott model, which has two channel states: “good” and “bad” channel conditions. As illustrated in Figure 4.7, $P_G$ denotes the probability that the channel will stay in the good state in the next time slot given that the current state is good, and $P_B$ denotes the probability that the channel will stay in the bad state in the next time slot given that the current state is bad. Accordingly, we have $1 - P_G$ as the state transition probability from the good state to the bad state in the next time slot, and $1 - P_B$ as the state transition probability from the bad state to the good state in the next time slot. Therefore, the expectation time that the wireless channel remains in the good and bad state can be given as $T_G = \frac{1}{1-P_G}$ and $T_B = \frac{1}{1-P_B}$ respectively. Note that it is straightforward to extend the two-state model to a higher-order model with more states to achieve higher accuracy, but a two-state wireless channel model is sufficient for our purpose. We further assign a packet error rate to each state ($P_{eg}$ is for the good state, and $P_{eb}$ is for the bad state), which is defined as the ratio of the number of retransmitted packets to the total number of packets.

We assume that any packet that is corrupted during transmission and cannot be fixed by error-correction techniques such as CRC coding must be retransmitted. Let $t$ denote the time required for transmitting $n$ packets over an error-free wireless channel. The expected time for transmitting the same number of packets over an error-prone wireless channel can be calculated as $t_e = \frac{t}{1-P_e}$. 

Figure 4.7: Wireless channel model
4.5.3 Online Scheduling

We now propose a heuristic for online scheduling, whose flow chart is given in Figure 4.8. The details in each step are presented as follows.

As discussed earlier, the jobs should be classified first, so the heuristic assigns tags to the jobs that cannot be taken as fully independent ones. Each tag is a tuple, and consists of (category, group ID, index/type). Category 1 indicates job dependency, and category 2 indicates data sharing. More specifically, the tags (1, \(X\), 1) are assigned to the jobs on which other jobs rely; the jobs that are dependent on others are tagged with (1, \(X\), 2); as for the jobs that share data, the heuristic assign them the tags (2, \(X\), index), where \(X\) denotes the group ID. As there can be multiple dependencies or different combinations of jobs sharing different data, the same group ID is assigned to the jobs that are in the same dependency or data sharing relationship. The jobs that are in the same data sharing group are then sorted according to the release time, and the indexes are decided accordingly. It should be noted that one job can have multiple tags.

After the tag assignment, the heuristic selects some of the tagged jobs that should be scheduled with higher priority. As stated in the previous discussion, the jobs with tags (1, \(X\), 1) should be selected first for scheduling. The jobs with tags (1, \(X\), 2) are not considered until their tags are removed after all the jobs with tags (1, \(X\), 1) have been scheduled. In each round of scheduling, for the jobs tagged as category 2 (with tags (2, \(X\), index)), the job with the smallest index is selected from each group, which is the job coming earliest in that group, (if the job is also tagged with (1, \(X\), 2), the job with the next smallest index is chosen). Besides the tagged jobs, it also needs to choose the untagged jobs that should be scheduled together with those previous selected jobs. Define the ratio of the computation load to the transmission load for job \(i\) as \(r_i = c_i/(d_{is}^t + d_{ir}^t)\). A higher value of \(r_i\) usually implies a higher possibility to save energy, since our energy model is linear and thus this ratio is close to the corresponding ratio of energy consumption. The heuristic calculates the ratio \(r\) for the selected tagged jobs as well as for the untagged jobs. Assume \(r_{min}\) is the minimum ratio for the selected tagged jobs. Then the untagged jobs that have higher ratios than \(r_{min}\) are considered for scheduling. Therefore, in each round of scheduling, there are three kinds of jobs being scheduled: the jobs with tags (1, \(X\), 1); the jobs with smallest index in each group of category 2; the untagged jobs has higher \(r_i\) than the current \(r_{min}\).

When scheduling the selected jobs, the heuristic adopts the following rules: 1) Sort the
Figure 4.8: Flow chart of the online heuristic
jobs by $r$ in a descending order, and schedule the jobs following this order; 2) Given the predicted wireless channel state, if no energy saving can be achieved ($E_t(d_t^n + d'_n) < E_c(c_n)$), schedule the job to run locally; otherwise, schedule the job to be offloaded as early as possible; 3) If there are enough free time slots, schedule the job directly; otherwise, if the available interval, which is not long enough to offload the current job, is followed by the previous schedule of an untagged job, postpone the schedule to extend the available interval as long as the new schedule does not break the time constraints for the rescheduled job; 4) If no available schedule can be found, the job is scheduled for local execution. The intuition behind rule 1 is to schedule the jobs that save more energy with higher priority. Rule 2 and rule 3 ensure that the heuristic can adjust the schedules to offload more jobs for energy saving and guarantee the performance at the same time. It should be noted that only the untagged jobs are allowed for rearrangement. The reason is that rescheduling the tagged jobs will affect all the jobs that have related tags to them.

After a round of scheduling is done, the heuristic checks the schedules for the tagged jobs and perform proper updates. If a job with tag $(1, X_i, 1)$ is scheduled to be offloaded, it updates the release time of unscheduled jobs that have tag $(1, X_i, 2)$ as the scheduled finish time. Further, if all the jobs with tag $(1, X_i, 1)$ have been scheduled, the tag $(1, X_i, 2)$ is removed from those unscheduled jobs. On the other hand, when a job with tag $(2, X_i, j)$ is scheduled to execute on cloud, the heuristic decreases the amount of data to be sent for the unscheduled jobs with tag $(2, X_i, j')$ and then removes the tag. After multiple rounds of scheduling, the heuristic returns the final schedule when all jobs have been scheduled.

Another important component of the heuristic is monitoring and predicting the wireless channel state. During the computation offloading, the number of retransmitted packets is counted. Let $P_{re}$ denote the percentage of retransmitted packets during the $n$th time slot, the predicted value of the packet error rate for the wireless channel is calculated as $P_e(n) = \alpha * P_{re} + (1 - \alpha) * P_e(n-1)$, where the coefficient $\alpha$ is between 0 and 1 ($0 \leq \alpha \leq 1$). In a fast-changing wireless channel, $\alpha$ should be set to a large value, whereas a small value should be assigned to $\alpha$ in a slow-changing channel. The predicted packet error rate $P_e(n)$ can be used to identify the current wireless channel state in the $n$th time slot with a proper threshold $P_{thres}$. Therefore, the length of time that the wireless channel stays in one state can be obtained, which can further be used to estimate the state transition probability. In this way, the wireless channel state in a short future can be predicted. If the prediction of the channel state is wrong for $m$ successive time slots, the heuristic triggers another round
of scheduling, where the parameter $m$ controls how quick that the heuristic responds to the channel state change.

## 4.6 Evaluation

We evaluate the performance of our proposed solutions through simulation in this section. As shown in previous studies [40], the probability distribution of applications’ demand, such as the cycle demand, can be described by Gamma distribution and some other distributions. In the simulation, we set the job in-coming rate to be approximately 1 job per second with the average delay tolerance of 2 seconds, and we assume that the transferred data during offloading is generated with a Normal distribution. We apply the power model presented in Section 4.2.5. The link throughput is set to be 1 MB/s when the channel is error-free, and the corresponding throughput can be calculated given the packet error rate in the error-prone channel. The average packet error rate for the good state is assumed to be 0 and that for the bad state is set to 0.15. Our first result is obtained from our measurement traces in Section 4.2 after simple calculation. Roughly speaking, on our test device, for computation offloading to be beneficial the workload needs to perform more than 3000 cycles of computation for each byte of data. This indicates that the type of workload has a huge influence on the energy efficiency of computation offloading. It is not surprising that a computation-intensive task, such as the chess game, can still save a lot of energy with a wireless connection of low quality.

To test the performance of our proposed solutions, we simulated a computation-intensive application with synthetic demand traces, in which the majority of jobs have high cycle-to-data ratio (many may be even over 5000 cycles/byte) and a small portion of jobs are data-intensive. Table 4.5 shows the energy efficiency improvements of offloaded executions over the local execution. We vary the channel state transition ($1-P_G$, $1-P_B$) probability as shown. Our solutions show better performance than the naive offloading approach even if the channel is nearly continuously changing. The improvement comes from two aspects: first, our methods select the jobs that are really computation-intensive, and leave the data-intensive jobs locally; second, our methods adjust the schedules according to the state change of the wireless channel. When $P_G = 0.999$, the channel can be seen as staying in one state, since the expected length largely exceeds the total execution time of multiple offloaded jobs, and thus all the offloading schemes achieve the best performance. The performance of our
solutions drops a lot when $P_G = 0.8, P_B = 0.8$, as the wireless channel is in the good state only for half of the total time, decreased from 83.3% previously. Further, when the expected length of the good state is reduced to 2 slots, the performance continues to degrade as there are much less opportunities for the effective scheduling. On the other hand, the expected length of the good state has little influence on the performance of the naive offloading approach, while its performance is mainly affected by the percentage time that the channel is in the good state. The reason is that the naive approach offloads all the jobs without selection, and its energy efficiency depends on how much traffic is transmitted in the good channel state. It should be noted that these results come with the application’s nature of computation-intensity. Other computation-intensive applications may achieve even higher numbers, while some other applications may waste energy when offloading their computation to the cloud.

<table>
<thead>
<tr>
<th>$P_G$</th>
<th>0.999</th>
<th>0.99</th>
<th>0.8</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_B$</td>
<td>0.995</td>
<td>0.95</td>
<td>0.8</td>
<td>0.5</td>
</tr>
<tr>
<td>Percentage of time in good state</td>
<td>83.3%</td>
<td>83.3%</td>
<td>50%</td>
<td>50%</td>
</tr>
<tr>
<td>Expected length of good state</td>
<td>1000</td>
<td>100</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Offline improvement</td>
<td>40.3%</td>
<td>38.5%</td>
<td>30.9%</td>
<td>28.6%</td>
</tr>
<tr>
<td>Online improvement</td>
<td>37.1%</td>
<td>33.7%</td>
<td>26.5%</td>
<td>21.3%</td>
</tr>
<tr>
<td>Naive offloading improvement</td>
<td>28.9%</td>
<td>29.1%</td>
<td>15.4%</td>
<td>14.7%</td>
</tr>
</tbody>
</table>

Table 4.5: Energy efficiency improvement

As the key component of our online heuristic solution, the channel state prediction scheme is now examined. We set $P_{thres} = \frac{1}{2}(P_{eg} + P_{eb})$ as the threshold. Every time the predicted packet error rate crosses this threshold, it is identified as a wireless channel state change. After a period of time of observation, the state’s expected length can be estimated as well as $P_G$ and $P_B$. The heuristic compares the predicted channel states with the observed channel states in the running window of $m$ slots. If there are $m$ successive misses, the heuristic reschedules the jobs. Table 4.6 shows the average number of reschedulings with different $P_G$ using $m = 10$. The total length of the predicted period is $1 \times 10^5$ slots, and $P_B$ is kept to be 0.7. It is not surprising that the more stable channel is easier to predict. As $P_G$ decreases, we have seen more wrong predictions, and thus more reschedulings, which may suggest us to adjust the value of $m$ according to the expected state length and to have multiple thresholds for identifying the channel status change.
### 4.7 Summary

In this chapter, we have presented the computation offloading approach to improve mobile devices’ energy efficiency. Unlike using local resources, offloading the computation to the remote cloud in wireless connection requires us to carefully manage of the communication. Our measurements have identified the key influential factors that affects the performance and energy efficiency of offloading in wireless networks. Motivated by our findings, we have formulated the energy-efficient offloading scheduling problem, and further proposed the effective solutions.
Chapter 5

Conclusion and Future Work

In this thesis, we have closely examined two approaches that can effectively extend the battery life of mobile devices. We first presented our work on energy-efficient location tracking with smartphone sensors as a case study. Our design fully utilizes smartphones’ increasing capabilities of computing and sensing, which has shown us the significant potential to save energy by smartly utilizing the local computation resources on mobile platforms. Further, we have discussed and measured the energy reduction of computation offloading for two realworld mobile applications. With the key influential factors identified, we formulate the offloading scheduling problem and then provide energy-efficient solutions. Our work has proved that mobile clients can achieve higher energy efficiency by adaptively offloading the computation to the remote cloud.

To extend our work that has been presented in this thesis, our future work may also investigate the energy efficiency related topics following two directions. One is to further explore the effective utilization of other local resources. Today, with the GPUs rapid evolution from a configurable graphics processor to a programmable parallel processor, the ubiquitous GPU has been seen on mobile devices more and more frequently. As the many-core multi-threaded multiprocessor that excels at both graphics and computing applications, GPU has its natural advantages in massively parallel processing. Therefore, GPU computing can be the cost-effective alternate for traditional CPU processing, which is very promising on mobile systems.

Our discussion about computing offloading in this thesis stays at the higher layer. Although it is very tempting and promising to use the much cheaper and more powerful
resources on cloud, the interaction between a mobile terminal and the cloud needs care-
ful examination to avoid excessive transmission overhead. It is necessary to partition the
application with a finer granularity into local and remote parts when considering the im-
plementation of computation offloading. Even the application can be perfectly decomposed
into relatively independent modules and all the information available, including mobile ter-
minal and cloud capabilities, it is still not an easy job to obtain an optimal partitioning. It
would become more complex if the intrinsic QoS requirements or constraints of applications,
for example real-time or bandwidth requirement, are considered. Further, the modules of
some applications have inherent dependency on each other, which further complicates the
partitioning. Besides those, such other issues as privacy and security also need to be con-
sidered. Both privacy and security could be compromised in offloading, since the programs
and data are stored and managed by cloud providers and are not under users or application
providers control. Encryption could be used to protect the outsourced data. However, it
incurs high overhead, and thus can be undesirable for mobile users. More importantly, the
unique data of mobile users, including locations and mobility patterns, could be exposed
during offloading, which need specific protections if necessary. All these topics remaining as
our future work need carefully investigation.
Bibliography


