May 2015

A Unified And Green Platform For Smartphone Sensing

Xiang Sheng
Syracuse University

Abstract

Smartphones have become key communication and entertainment devices in people's daily life. Sensors on (or attached to) smartphones can enable attractive sensing applications in different domains, including environmental monitoring, social networking, healthcare, transportation, etc. Most existing smartphone sensing systems are application-specific. How to leverage smartphones' sensing capability to make them become unified information providers for various applications has not yet been fully explored.

This dissertation presents a unified and green platform for smartphone sensing, which has the following desirable features: 1) It can support various smartphone sensing applications; 2) It is personalizable; 2) It is energy-efficient; and 3) It can be easily extended to support new sensors.

Two novel sensing applications are built and integrated into this unified platform: *SOR* and *LIPS*. SOR is a smartphone Sensing based Objective Ranking (SOR) system. Different from a few subjective online review and recommendation systems (such as Yelp and TripAdvisor), SOR ranks a target place based on data collected via smartphone sensing. LIPS is a system that learns the LIfestyles of mobile users via smartPhone Sensing (LIPS). Combining both unsupervised and supervised learning, a hybrid scheme is proposed to characterize lifestyle and predict future activities of mobile users.

This dissertation also studies how to use the cloud as a coordinator to assist smartphones for sensing collaboratively with the objective of reducing sensing energy consumption. A novel probabilistic model is built to address the GPS-less energy-efficient crowd sensing problem. Provably-good approximation algorithms are presented to enable smartphones to sense collaboratively without accurate locations such that sensing coverage requirements can be met with limited energy consumption.

A Unified And Green Platform For Smartphone Sensing

by

Xiang Sheng

B.E., Nanjing University of Science and Technology (2008)
M.S., Syracuse University(2010)

Dissertation
Submitted in partial fulfillment of the requirements for the degree of
Doctor Of Philosophy in Electrical And Computer Engineering

Syracuse University
May 2015

To my beloved family

Acknowledgements

Many people have contributed to making graduate career rewarding, without whom this dissertation could not have been written and to whom I am deeply indebted.

First, and foremost, I want to thank my advisor, Dr. Jian Tang, who has continuously provided me with guidance and support throughout my graduate study. He taught me about how to do research, and helped me focus on the important issues when looking at a new problem. A very special thank for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. This dissertation would have been impossible without his mentorship. I could not have imagined having a better advisor and mentor for my Ph.D study.

Besides my advisor, I would like to thank all my labmates: Jielong Xu, Chenfei Gao, Teng Li, Xuejie Xiao, Jing Wang and Jimmy Chen. Many thanks for all your collaboration, company and encouragement. The past years, without you, would be a lonely research journey. You made the time much more enjoyable.

I also wish to acknowledge the financial support from NSF grant CNS-1218203.

My final but everlasting gratitude goes to my family members, who have always supported me in my choices. Without their continuous encourage-

ment and constant love, nothing would have been possible. This dissertation is dedicated to you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Vision

Figure 1-1: Sensing the world!

As a child, a teen or an adult, have you ever had the dream that you have certain supernatural ability, such as clairsentience, clairaudience, or clairvoyance? This kind of dream is commonly shared among different cultures. Mythic figures sometimes have the superpower to perform extrasensory perception. They could receive information far beyond what they could gain through physical senses.

In Chinese mythology, there is a mythic figure called God ErLang, who has three eyes. The third eye in the center of his front head is very powerful, with which he can see wherever he wants to see. By leveraging the smartphone's sensing ability and cloud computing technology, we can bring every person ErLang's third eye and the superpower of sensing anywhere across the whole world.

This thesis explores the design, implementation and evaluation of an energy efficient unified platform for smartphone sensing. Users of this platform will have the extrasensory perception ability such that they can see, hear and sense anywhere they want at any time (Fig. 1-1).

1.2 Background

Nowadays, Smart phones have evolved as key electronic devices for communications, computing and entertainment, and have become an important part of people's daily life. People use smartphones taking pictures, recordings dialogs, browsing websites. Most of smartphones (iPhone 5, iPhone 5S, Nexus 5, etc.) are equipped with a rich set of embedded sensors such as camera, GPS, WiFi/3G/4G interfaces, accelerometer, digital compass, gyroscope, microphone and so on [35]. As shown in Fig. 1-2, iPhone 5S [30] have at least 11 types of sensors.

Although not built for sensing, smartphones' sensing ability have attracted many research attentions in various research domains [35]. Some embedded sensors could be directly used to collect useful information: front and back cameras on smartphones can be used as image sensors; network interface could be used as network signal strength sensors and coarse location sensors; GPS could be used as accurate location sensors; Microphones could be used as acoustic sensors;

Other embedded sensors, such as: accelerometers, gyroscopes, light sensors and proximity sensors, were originally designed to assist the phone's display such as the

Figure 1-2: Embedded sensors in iPhone 5S

screen orientation, screen brightness, and so on. These sensors could also be used to provide useful information to help infer users' background contexts. For example, accelerometers could be used to infer user's pace frequency. Moreover, in some existing projects, by analyzing features collected from multiple sensors, more interesting information could be discovered. Such as, in Jigsaw project [43], user's moving status (walk, running, bicycling, etc.) could be inferred by jointly analyzing continuous sensing readings from multiple sensors.

In recent years, a lot of third party external sensors have also been developed, such as Jawbone up [31], Fitbit [19], Google Glass [23] and SensorDrone [63]. These external sensors can communicate with smartphones via Bluetooth protocol and extend the smartphones' sensing ability. For example, with a SensorDrone device connected, a smartphone can sense the air pressure, brightness, temperature and humidity.

In this thesis, we propose to leverage emerging cloud computing model to build a platform. To provide sensing applications for numerous cloud users with different needs, the platform must be able to support various participatory sensing and opportunistic sensing applications on different smartphone platforms. Users could use this platform to provide information and collect useful information.

Figure 1-3: An external sensor: Sensordrone [63]

Two interesting sensing applications are built on top of the proposed platform as examples of how to use the platform to develop sensing related applications. One is used for **objective ranking** and the other for **user's lifestyle learning**.

Objective ranking system: Currently, a few online review and recommendation systems have attracted millions of users and are gaining increasing popularity. For example, Yelp serves as an online urban guide, which provides user reviews, recommendations and rankings of a large variety of local businesses including restaurants, shops, theaters, etc. Another typical example is TripAdvisor, which has become the world's largest website for travelers. It provides user reviews, ratings and ranks for hotels, restaurants, attractions in different places across the whole world. These systems usually rate and rank target places and attractions based on *subjective* ratings and opinions provided by users. It is known commonly that human beings' subjective opinions are easily biased. Since nowadays people always carry smartphones with them, data collected via smartphone sensing can be used to evaluate a target place. For example, for a coffee shop, based on readings from microphones, we can know if it is quiet; based on readings from light sensors, we can know if it is bright; based on readings from temperature sensors (on Sensordrones), we can know if it is warm. So we design, implement and evaluate an objective ranking system, which ranks target places based on (objective) data collected via smartphone sensing on top of the platform. The objective of this application is not to replace the current rank-

ing/recommendation systems that are based on subjective user ratings but to enhance them with objective data collected via various sensors to provide more comprehensive and objective rankings and recommendations for users.

Users' lifestyle learning: According to businessdictionary.com, *"Lifestyle is expressed in both work and leisure behavior patterns and (on an individual basis) in activities, attitudes, interests, opinions, values, and allocation of income."* By leverage the sensing ability of user's smartphone, a comprehensive view of the context (such as location, local weather, activities, etc.) of a mobile user over a long period of time could be obtained. The lifestyle learning system we designed is to find out what a mobile user likes to do (characterization) and what he/she will do next (prediction) based on the collected sensor data. Such a lifestyle learning system can be used to support a large variety of applications for improving life quality. For example, a major application is to recommend local businesses to mobile users based on not only his/her location but also his/her lifestyle. This work represents one of the first efforts along this line, which is focused on lifestyle learning, while leaving lifestyle-aware recommendation or lifestyle-based applications for future research.

There are primarily two smartphone sensing paradigms [64]: Participatory Sensing and Opportunistic Sensing. In participatory sensing, mobile users actively engage in sensing activities by manually determining how, when, what, and where to sense. In opportunistic sensing, sensing activities are fully automated without the involvement of mobile users.

However, a phone's main job is not sensing after all. Performing sensing tasks may consume a significant amount of energy of a smartphone. Therefore, without carefully managing very limited energy resources on smartphones, users may end up with an awkward situation after performing a few sensing tasks, in which phones run out of energy when they are needed to make phone calls. To our best knowledge, fundamental energy-efficient resource management problems have not been well studied for

smartphone sensing. Furthermore, unlike a traditional sensor network which is usually operated by a single organization, smartphones and their sensors are owned and controlled by different individual users. Hence, the mobility is totally uncontrollable and hard to predict. As shown in one of our research publications [65], in case of some sensing tasks need multiple smartphones to participate, significant data redundancies will exist if each smartphone senses independently. In this thesis we will present how to efficiently schedule sensing tasks to save total energy consumption. Moreover, most smartphone sensing applications are location-dependent. If energy-hungry GPS is turned on during the whole sensing procedure, the battery may be drained very quickly. So a GPS-free collaborative sensing tasks scheduling could significantly save the total energy consumption, which will be presented in this thesis.

We will review some existing research results in the related field in Section 1.3.

1.3 Related Work

Research effort has been made to apply smartphone sensing in multiple domains: environmental monitoring, transportation, healthcare, social network, etc. For example:

Environment monitoring: a smartphone sensing application, called PEIR (Personal Environmental Impact Report) [53], was developed to use location data sampled from smartphones to calculate personalized estimates of environmental impact and exposure. Ear-Phone [59] is a smartphone based urban noise monitoring system. NoiseTube [46] has the similar functionality. In addition, *www.sensorly.com* is a website which offers free access to 100% community powered coverage maps for various wireless networks (3G/4G/WiFi). Their mapping crowd collects data every day using a free application on smartphones, which produces a true picture of the carriers' coverage. Lu et al. proposed SoundSense in [42], a scalable framework for modeling

sound events on smartphones, which uses a combination of supervised and unsupervised learning techniques to classify both general sound types (e.g., music, voice) and discover novel sound events. SoundSense was implemented on the iPhone and represents the first sound sensing system specifically designed to work on resource limited smartphones.

Traffic monitoring: Since users also carry smartphones when they driving. Smart phone based traffic monitoring systems have been developed. For example, CalTel [29] uses smartphones to collect traffic information, WiFi information on the road. Nericell [51] is a smartphone sensing based road condition and traffic monitoring system, which uses various sensors on a smartphone to detect potholes, bumps, braking and honking. Another system is VTrack [69], a mobile sensing system which tracks the traffic delays and congestions. In VTrack, drivers' smartphones are used to provide spatio-temporal samples to monitor traffic delays. Specifically, VTrack uses WiFi signals to estimate the driver's locations, along with a hidden Markov model based map matching method, to identify the road segments and the time spent on these segments.

Personal health monitoring: UbiFit Garden [9] is a smartphone sensing system jointly developed by Intel and University of Washington, which uses small inexpensive on-body sensors and machine learning techniques on activity modeling to infer people's activities throughout everyday life. This system captures levels of physical activity and relates this information to personal health goals when presenting feedback to the user for encouraging physical activity. DietSense system [60] is a smartphone based diet monitoring system, which captures images automatically during mealtime. Users could keep track their diets while professions performing analysis. BALANCE [14] is another smartphone sensing system which monitors people's eating and activity behaviors and encourage healthier lifestyles. BALANCE automatically detects the user's caloric expenditure via sensor data from a Mobile Sensing Platform

unit worn on the hip. HealthSense [66] is a patient monitoring system which aims to provide professionals more information than he/she can manually interpret. HealthSense transmits sensor data from the patient to a server for analysis via machine learning techniques. In another project [36], Lee et al. connected the ZigBee-based built-in blood glucometer to smartphones. The measured blood glucose could be transmitted directly to the web. Leijdekkers and Gay [38], developed a heart attack self-test system. In this system, electrocardiogram sensors are wirelessly connected with a smartphone, which can collect a mobile user's symptoms and send them to a smartphone application. The mobile application can then analyze the streaming data to detect the onset of a heart attack. If the application assesses that the user is at risk, it will urge him/her to call the emergency application immediately. If the user has a cardiac arrest, the application will automatically determine his/her current location and alert the ambulance application.

Social network: Smart phone sensing could provide more information to social networks besides text, image and videos. As its name suggests, Micro-blogs [20] is a smartphone sensing system developed for social networks. This system let users post geotagged blog entries and enhance them with sensing data (e.g., audio records, pictures, accelerometer data, or WiFi coverage) captured via their smartphones; query and browse information via a digital map application; and send requests to smartphones in the region of interest. By applying machine learning algorithm to collected sensing data(e.g. acceleration, audio samples, pictures, neighboring devices, and location) captured by the smartphone, CenceMe system [49] can derive user's personal sensing presence (e.g., walking, in conversation, at the gym) and share this information through social networks.

General-purpose smartphone sensing systems have also been introduced by a few recent works. In [41], the Bubble-Sensing framework was proposed to bind sensing tasks to a specific physical locations of interest. This framework can be used for both

opportunistic and participatory sensing. Mobile users are selected to collect information, such as background noise and photo. A "bubble" task can be bound to the location of interest and remains active for a period. This sensing framework could be used to keep a living documentary of places of interest. Cornelius et al. introduced AnonySense in [11], which is a privacy-aware system for smartphone sensing. Designed for community-oriented information applications, AnonySense distributes sensing tasks among a set of anonymous mobile devices, and collect verified yet anonymous sensing results. The proposed system aims at addressing the privacy concerns in large-scale sensing applications.

In [12], Das et al. presented a Platform for Remote Sensing using Smartphones (PRISM). PRISM enables third-party applications to be packaged as executable binaries and push them automatically to an appropriate set of phones. The smartphone end of PRISM will then execute the received executable file.

Sensing scheduling and coordination have been studied in the context of smartphone sensing recently. A protocol, Aquiba [68], was proposed to exploit opportunistic collaboration of pedestrians for smartphone sensing. In [72], several mechanisms were introduced for automated mapping of urban areas that provide a virtual sensor abstraction to applications. Spatial and temporal coverage metrics were also presented for measuring the quality of acquired data. In a recent work [65], Sheng et al. presented optimal algorithms and practical heuristic algorithms for energy-efficient collaborative sensing scheduling problems.

There are several drawbacks of application-specific solution:

1) Most existing smartphone sensing systems, such as systems presented in [9, 20, 36, 38,42,50,51,53,59,69] are application-specific. Significant overhead exists in sensing application development, since some functionalities are needed to be developed repeatedly, such as how to communicate with backend server.

2) There will be a large set of sensing applications, which brings maintenance diffi-

culty. Users need to install different applications for different use cases, which is really inconvenient.

3) Even though PRISM [12] is a general-purpose mobile sensing platform, the sensing tasks delivered to smartphones are packaged as executable binaries, which are platform-dependent (Windows Mobile only) and may cause security issues. AnonySense [11] uses a customized, yet very limitedly-used Lisp dialect for implementation. Therefore, its applicability is very limited.

4) Energy-efficiency has not been addressed by these related works. It is hard to coordinate multiple sensing apps running on one smartphone, which make the sensing not energy efficient.

On energy-efficient collaborative sensing scheduling, only few recent works are related. In [72], the authors introduced mechanisms for automated mapping of urban areas that provide a virtual sensor abstraction to applications. They also proposed spatial and temporal coverage metrics for measuring the quality of acquired data. In [68], the authors proposed a protocol, Aquiba, that exploits opportunistic collaboration of pedestrians. Its performance was studied via simulations. Collaborative sensing has been well studied for mobile sensor networks. In [79], Zhou et al. considered how to deploy mobile sensors into an existing sensor network to enhance its connectivity and coverage, and presented a dynamic programming based algorithm under the assumption that each sensor is equipped with GPS. Several distributed algorithms were presented for a sensing coverage problem in [78], which do not need any location and distance information. In [61], Saipulla et al. explored the fundamental limits of sensor mobility on barrier coverage and presented a sensor mobility scheme that constructs the maximum number of barriers with minimum sensor moving distance.

However, these existing research works have certain drawbacks: 1) Closely re-

lated works [68, 72] presented heuristic algorithms that cannot provide performance guarantees. 2) The algorithms presented in [61, 78, 79] for mobile sensor networks (in which sensor mobility can be controlled to achieve certain sensing coverage) cannot be applied here because the mobility of smartphones is usually uncontrollable.

1.4 Contributions

In this section, the contributions of this thesis will be clearly presented.

First, the design and implementation of a unified platform is presented. The proposed system is general enough such that various opportunistic and participatory sensing applications(which may even involve a large variety of sensors) could run on top of it. The overhead to launch a new sensing application is very little, since the proposed platform can be easily and quickly reconfigured. Old inefficient algorithms or policies could be easily replaced with new ones. Energy-efficiency is taken into consideration during the platform design. The data could be shared among multiple running applications.

Then, two smartphone sensing applications are built on top of the platform : 1)**SOR**: SOR stands for "Sensing based Objective Ranking system". Different from a few online review and recommendation systems (such as Yelp and TripAdvisor) which usually rate and rank places and attractions based on subjective ratings provided by users, SOR ranks a target place based on data collected via smartphone sensing. 2)**LIPS**: LIPS is the abbreviation for "Lifestyle learning via smartPhone Sensing". Combining both unsupervised and supervised learning, this general system can characterize and predict users' lifestyle.

How to leverage cloud-assisted collaborative sensing to reduce sensing energy consumption for smartphone sensing applications is also studied in this thesis. By solving the modeled minimum energy sensing scheduling problem, it is shown that signif-

icant energy savings can be potentially achieved by using collaborative sensing in smartphone sensing applications. Since GPS is really energy consuming and accurate location information is not always available, this thesis also solved the GPS-less energy-efficient sensing scheduling problem for mobile crowd sensing. A probabilistic model for sensing coverage without accurate location information (provided by GPS) is proposed. A GPS-less energy-efficient protocol for sensing scheduling based on the probabilistic coverage model is proposed for the real use.

The contribution of this thesis could be summarized as follow:

1) A unified platform is designed and implemented. The architecture of the proposed system not only can be easily adjusted to meet the new sensing needs, but also can be easily extended to meet the new sensing technology.

2) A mobile sensing based objective ranking application is developed on top of the unified platform. An online scheduling algorithm is proposed and used to schedule sensing activities for coverage maximization, which has a constant approximation ratio of 1/2. A personalizable ranking algorithm is developed and used to rank target places based on various sensor readings and user preferences. The SOR system is validated and evaluated via both field tests (using real hiking trails and coffee shops in Syracuse, NY as target places) and simulation.

3) A mobile sensing based lifestyle learning application (LIPS) is built on top of the unified platform. A hybrid scheme, combining both unsupervised and supervised learning, for lifestyle learning is proposed. The system first characterizes the lifestyle of a mobile user using Places of Interest (PoIs). Then based on discovered PoIs, LIPS could predict user's future activities using a supervised classification algorithm. In addition, an adaptive sampling algorithm for improving energy efficiency is proposed for this system. LIPS has been validated and evaluated via extensive field tests.

4) This thesis also studied how to leverage cloud-assisted collaborative sensing to reduce energy consumption for smartphone sensing applications. By assuming

the moving trajectory of each mobile user is known in advance, a polynomial-time algorithm is proposed to obtain minimum energy sensing schedules, which can be used to show potential energy savings that can be brought by collaborative sensing and can serve as a benchmark for performance evaluation. This thesis also presented an algorithm to achieve a good trade-off between total energy consumption and fairness. Under realistic assumptions, two practical and effective heuristic algorithms: the prediction-based algorithm and the function-based algorithm are proposed. It has been shown by simulation results based on real energy consumption and location data that compared to traditional sensing without collaborations, collaborative sensing significantly reduces energy consumption, and the proposed function-based algorithm performs well in terms of both total energy consumption and fairness.

5) This thesis proposed a probabilistic model for sensing coverage without accurate location information. Based on that and under a strong assumption that the moving trajectories of mobile users are known in advance, a $(1 - \frac{1}{e})$-approximation algorithm is proposed to minimize the energy consumption, and a $\frac{1}{2}$-approximation algorithm is proposed to minimize energy consumption while taking fairness into consideration. Both algorithms could be solved in polynomial time respectively. Under realistic assumptions, a GPS-less and energy-efficient protocol for sensing scheduling is proposed based on the proposed approximation algorithms. It has been shown by simulation results that the proposed protocol significantly outperforms a baseline method in terms of coverage probability. Experimental results from a field test are also presented to validate the proposed protocol.

1.5 Outline Of This Thesis

The rest of this thesis is organized as follows:

In Chapter 2, the design and implementation details of the unified platform are

presented. In this chapter, first we will give an overview of the unified platform's architecture and analyze its performance requirements in Section 2.2. Then the implementation details of our proposed solutions to address these needs is presented in Section 2.3 and Section 2.4 for mobile frontend and backend, respectively.

In Chapter 3, a Sensing based Objective Ranking system(SOR) is presented. In this chapter, how to implement this application on top of the proposed unified platform is presented in Section 3.2. The scheduling and ranking algorithms are presented in Section 3.3 and Section 3.4 respectively. The experimental and simulation results are presented and analyzed in Section 3.5.

The design, implementation and validation details of a lifestyle learning application are presented in Chapter 4. The implementation details of the proposed application are presented in Section 4.2. The proposed learning scheme and adaptive sampling algorithm are presented in Section 4.3 and Section 4.4 respectively. We evaluate the LIPS via field test and the results are presented and analyzed in Section 4.5.

The study of energy-efficient collaborative sensing will be presented in Chapter 5. With accurate location information available, we present the flow based scheduling algorithm in Section 5.2.2. A practical GPS-less collaborative sensing scheduling algorithm is presented in Section 5.3.2.

The conclusions of this thesis and future works are presented in Chapter 6.

Chapter 2

System Design Of A Unified Platform

2.1 Overview

In this chapter, we will present the design and implementation of a unified platform. As we presented in last chapter (Chapter 1), in the field of smartphone sensing, the contributions of most existing works focus on how to apply mobile sensing in a particular domain rather than propose a general system design.

This application-specific system design will bring several drawbacks:

1) Significant overhead exists in sensing application development.

2) There will be a large set of sensing applications, which brings maintenance difficulty.

3) Users need to install different applications for different use cases, which brings inconvenience to users.

4) Energy efficiency has not been addressed by these related works. It is hard to coordinate multiple sensing applications running on one smartphone, which makes

the sensing not energy efficient.

Recently, efforts have been made to develop general systems to support various smartphone sensing applications, however the existing systems have several drawbacks as we analyzed in Section 1.3: 1) PRISM [12] uses executable binaries to deliver sensing tasks to smartphones, which is platform-dependent (Windows Mobile only) and may cause security issues. 2) AnonySense [11] uses a customized yet very limitedly-used Lisp dialect for implementation. 3) Energy efficiency has not been well addressed by any of them.

To address the listed drawbacks, in this chapter, we propose a cloud coordinated unified (instead of application-specific) platform, as illustrated in Fig. 2-1. The following important issues need to be addressed in the unified platform:

1) The platform must be flexible enough such that it can support various opportunistic and participatory sensing applications. Since different sensing applications may involve different sensors, the platform must be able to collect data from the sensors and meet each application's accuracy requirement.

2) The platform must be able to be extended to new sensors that may connected to the smartphones. The platform must be designed that new sensing applications can be easily developed and quickly deployed.

3) New sensing policies which specify the sensing data type, sensing accuracy and sensing conditions can be easily adjusted without affecting other sensing applications running on the same platform.

4) The server side should be able to easily adjust the task distribution and coordinate algorithms. New backend functional modules (data analysis modules or data handling modules) should be easily deployed without affecting the existing modules.

5) Minimizing sensing energy consumption should be taken into consideration during system design.

As illustrated in Fig. 2-1, the proposed platform consists of two parts: Mobile frontend and Cloud Backend. The mobile frontend is implemented on each smartphone as an application, which can manage all sensors connected to that phone and administrate the sensing tasks locally. The cloud backend contains two types of functional modules: online modules and offline modules. Online modules are those modules used to handle the requests which need real-time response, such as users' authentications, sensing data upload and task distributions. Offline modules includes database and some modules used to handle computational expensive data processing tasks. Offline modules do not communicate with frontend directly. The processing results will be either stored back to database or be forwarded to online modules for further handling. The Online and Offline modules can be mapped to Multiple sensing servers (which can be physical servers or virtual-machine-powered servers) to scale up the system.

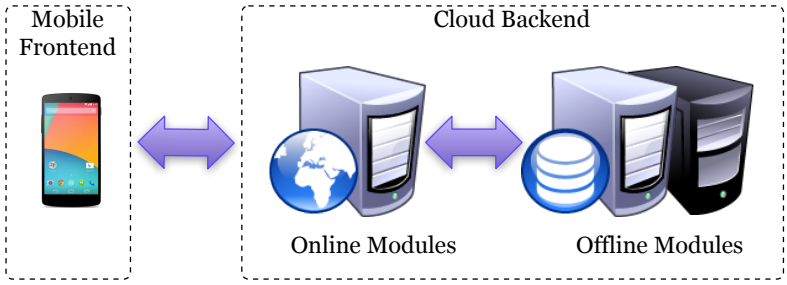


Figure 2-1: Unified platform overview

The rest of this chapter is organized as follows: we will present system architecture overview in Section 2.2. The implementation details of mobile frontend and cloud backend are presented in Section 2.3 and Section 2.4 correspondingly.

2.2 System Design

In the previous section, we analyzed the existing systems' drawbacks and proposed a cloud coordinated unified mobile platform as shown in Fig. 2-1.

In this section, we will present the overview of the system architecture and the technical details of the system.

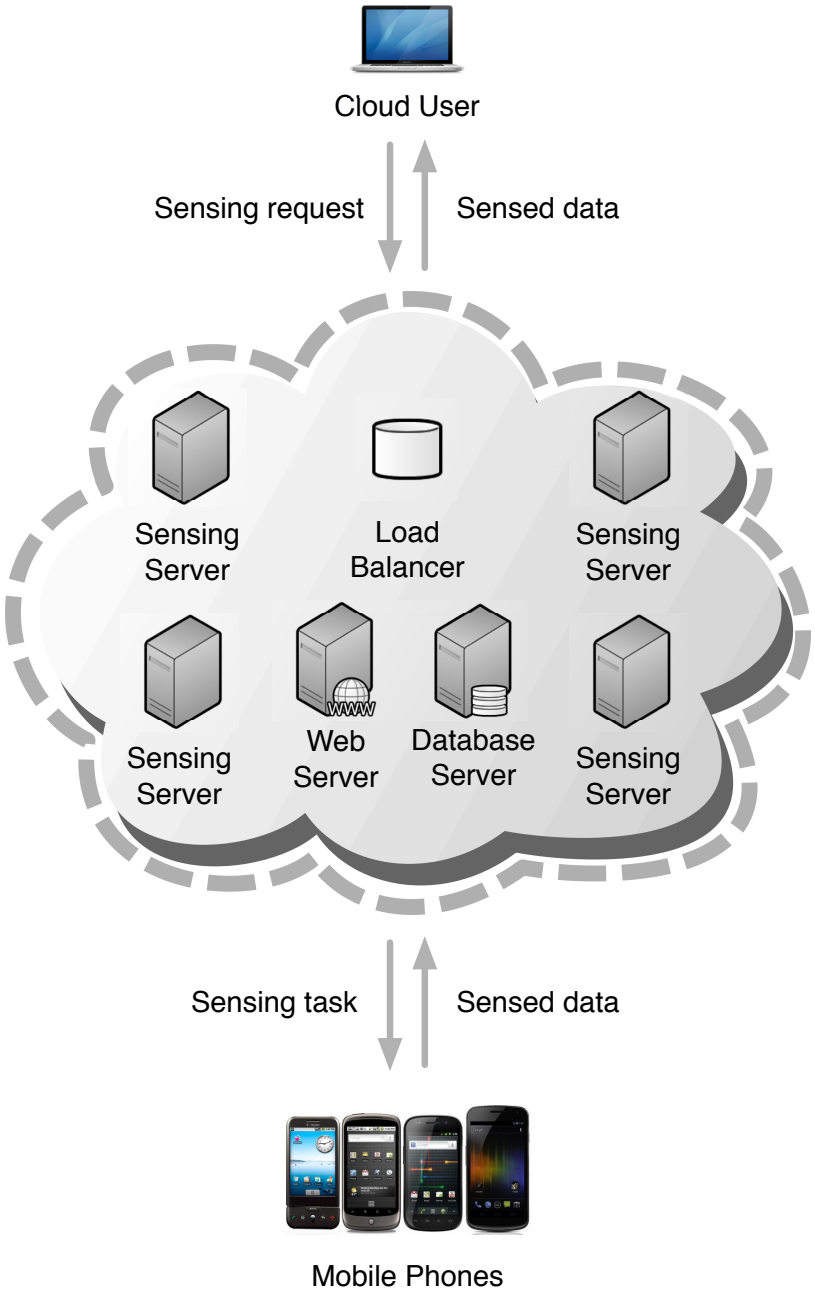


Figure 2-2: A cloud coordinated unified platform

A typical cloud coordinated unified platform is plotted in Fig 2-2. The differences between Fig. 2-1 and Fig 2-2 are: 1) Online-modules and off-line modules are plotted together into a cloud; 2) Detailed function components are plotted in Fig 2-2.

When a cloud user initiates a sensing request through a web front-end from either a smartphone or a computer (desktop/laptop), the request will be handled by one of the sensing servers to process the request. Sensing server will calculate and select a subset of smartphones that are able to fulfill the sensing tasks. For example, sensing server may select a subset of smartphones located close to the target places. Sensing server will dispatch sensing tasks to the selected smartphones and collect sensing data when tasks are fulfilled by these smartphones. The sensed data will then be stored in the database and analyzed. Analytic results will be returned to the cloud user who requests the application. An interesting feature of such a system is that a smartphone user (or simply mobile user) can be not only a cloud (application) user who can request sensing applications from the cloud but also an application provider who fulfills sensing tasks according to sensing requests from other mobile users. A load balancer could be used to balance the traffic when system is scaled up. Multiple sensing servers can be deployed to handle sensing requests from different locations.

The following functionalities should be supported by the cloud coordinated unified platform:

1) It needs to provide an interface for collecting sensing request information from cloud users, which can be accessed via a mobile device or a regular computer.

2) It needs to generate new sensing tasks in a standard format based on the sensing request (e.g., what sensors to use, what data to collect, what is the area of interest, how many readings to collect, etc.) collected from the web interface.

3) It needs to maintain important information of a list of smartphones that are available for participating in sensing tasks, including locations, available sensors, resid-

ual energy, etc.

4) It needs to provide an interface between a sensing server and smartphones for pushing sensing tasks to smartphones and collecting sensed data from them.

5) It needs to schedule sensing activities of the set of smartphones recruited for each sensing task using a scheduling algorithm or policy.

6) An application needs to be deployed on each smartphone to operate its sensors to perform the requested sensing actions, collect sensed data and send them to a sensing server.

7) It should be able to obtain sensed data from smartphones, store some useful information to the database (for future use) and/or return data reports to users.

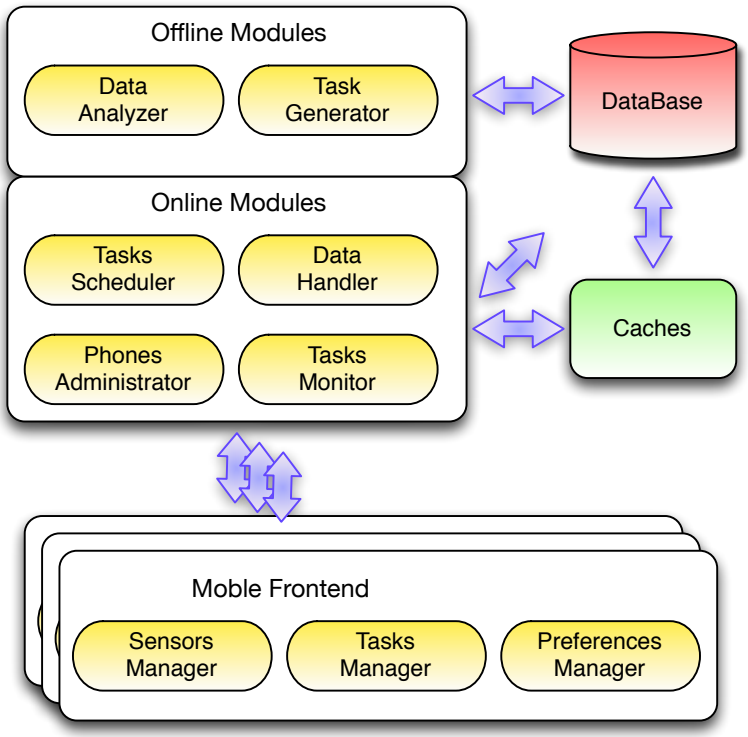


Figure 2-3: The software architecture

Next, we present the software architecture of the proposed platform, which consists of the following modules (Fig. 2-3).

1) **Task Generator:** It generates a new sensing task in a standard format based on application need. Task Generator contains a pool of sensing task templates for different use cases. Experienced users could also upload their own sensing tasks to the Task Generator for future use. Sensing tasks are a set of scripts which describe what/how to sense, what is the area of interest, etc.

2) **Data Analyzer:** It contains a set of analytic functions to process the collected sensing data.

3) **Data Handler:** It acts as a representative for the persistent database or in-memory data server (explained later). All reads/writes to the database or in-memory data server go through this module.

4) **Tasks Scheduler:** It schedules sensing activities of a set of smartphones for each sensing task according to a given sensing scheduling protocol. Scheduling protocols are application specific, and detailed discussion will be presented in the following chapters.

5) **Phones Administrator:** It maintains important information of a list of smartphones that are available for undertaking sensing tasks, including locations, available sensors, etc.

6) **Tasks Monitor:** It monitors the progresses of running tasks and serves as an interface between a sensing server and smartphones by pushing sensing tasks to smartphones and collecting sensed data from them.

7) **Sensor Manager:** It operates sensors on a smartphone to take the required sensing actions. Since sensors on modern smartphones usually work in an asynchronous manner, after sensing actions are taken, it notifies the Task Runner when sensed data are ready.

8) **Tasks Manager:** It manages the life cycle of received tasks, interprets each sensing task and fulfills this task by instructing the Sensor Manager to operate sensors, gathering sensed data, and uploading the data to a sensing server.

9) **Preferences Manager:** It manages each smartphones unique preferences for privacy protection.

The proposed system has the following desirable features:

1) The frontend's architecture is so scalable that various embedded and external sensors can be easily integrated into it.

2) The frontend is energy efficient.

3) The frontend's design takes concurrency into concern that multiple sensing tasks could run on it.

4) the communication protocol is designed to enhance the communication efficiency and security.

5) Backend server's modularized design makes it easier to be customized. More functionality could be easily added to the backend without changing the existing functionalities.

2.2.1 Technical Details

In this subsection, we will explain the issues met in the design and implementation of this platform in details. We need to make sure that all the key issues described above will be well addressed by our design and implementation.

First, to create a *unified* platform, scripts (instead of binary codes [12]) are used to describe every sensing task, which can be pushed to smartphones for execution. A mobile application was developed to execute scripts with the help of an interpreter and manage all available sensors. Essentially, the Task Generator can be implemented

as a script generator which generates scripts to specify sensing tasks based on task's type. Our platform also allow experienced users to submit scripts for their own sensing tasks to the Task Generator directly.

This design choice has the following advantages:

1) Scripting languages can enable dynamic and flexible loading of programs on smartphones. By using a scripting language, we can integrate an interpreter into the mobile application, and download and interpret scripts on-the-fly. In this way, the flexibility and speed of loading sensing tasks can be significantly boosted.

2) Scripting languages can enable portability for our system. Mobile devices may use different CPU architectures, such as ARM, MIPS, Sparc, x86, each having its own instruction set and Application Binary Interface (ABI), which is different from those of others. Moreover, if binaries are used, we have to deal with the 32bit VS. 64bit problem. With a scripting language, sensing tasks described using scripts can run on different hardware platforms, which can effectively increase the population of sensing crowd.

3) Scripting languages can also enhance security. The original scripts can be encrypted and signed by our sensing servers, which will prove they are indeed delivered from our trusted servers. Furthermore, binary codes can introduce potential security issues. Inspecting the potential security problem in binary package is much harder than doing that for scripts written in plain text, a scripting language can eliminate these potential problems by running scripts in a sandbox and only allowing them to use *white-listed* APIs such that they only interact with the hardware through trusted codes.

Lua [44] is the scripting language we chose in our platform. Lua is a powerful, fast, lightweight, and embeddable scripting language. It combines simple procedural syntax with powerful data description constructs based on associative arrays and

extensible semantics, which is exactly what we need for specifying sensing tasks. Lua interpreter is integrated into the sensing platform to interpret the sensing scripts written in Lua at run time. We enrich Lua library with self-defined sensing related functions, Lua interpreter can only interact with smartphone through the defined functions. We will give detailed discussion of sensing script in Section 2.3.

When designing the Cloud backend system, modularity was taken into account. Modularity plays a key role in designing a *reconfigurable* and *reliable* system. Every major functionality of this platform is implemented as an independent module with well-defined interfaces to interact with other components. We used Python [57] with Django [16] Framework to build the sensing server. Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. With the help of Django, we can implement components as separate modules, each of them can be used to handle requests sent to it independently. Users could specify their choice of how to combine existing modules.

In addition, with the module-based design, it is very easy to reconfigure a single part of the system for specific purposes. This can even be done on a per-application basis. For example, in order to improve *energy-efficiency*, efficient algorithms can be developed for mobile phone scheduling on the server side with the objective of minimizing and balancing power consumption. This will be discussed in greater detail in Chapter 5. Furthermore, machine learning based data modeler and predictor can also be developed as an add-on module for data analysis. We will introduce this kind of add-on in Chapter 4.

Availability and security need to be ensured for data storage. We selected a mature relational database, PostgreSQL [56] as the persistent storage solution for our platform. PostgreSQL has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity and correctness. Choosing PostgreSQL to store sensed data can give us high confidence

on the availability and safety of our data.

With all these considerations, we believe our platform has addressed our design goals. It can be used to fulfill all kinds of sensing applications with a large scale of users.

In the following sections, we will present the implementation details of the mobile frontend and cloud backend.

2.3 Mobile Frontend Implementations

In this section, we present the design and implementation details of the mobile frontend in Cloud coordinated platform. As illustrated in Fig. 2-4, the mobile frontend contains the following modules: *Message Handler*, *Task Manager*, *Local Database*, *Script Interpreter*, *Sensor Manager* and *Sensor Providers*. The mobile frontend is implemented as a mobile application that runs on each mobile user's smartphone.

The *Message Handler* serves as an interface for communications between the mobile frontend and a sensing server. HTTP is used as the communication protocol. All the communications between frontend and backend are serialized to binary data, according to our own defined communication protocol. The communication is also encrypted. Binary data is stored in the message body of an HTTP message. In this way, we can minimize traffic load and enhance security since the third party system does not know how to decode it). The Message Handler is responsible for encoding/decoding the message body.

The other functionalities of the Message Handler include: 1) it dispatches an incoming message to the intended sensing application currently running on top of the platform. The *applications Register* contains a set of mappings. Each mapping maps an incoming server's key to a receiving application. *Message Handler* uses this to decide how to distribute the incoming messages. 2) It encodes data obtained from sen-

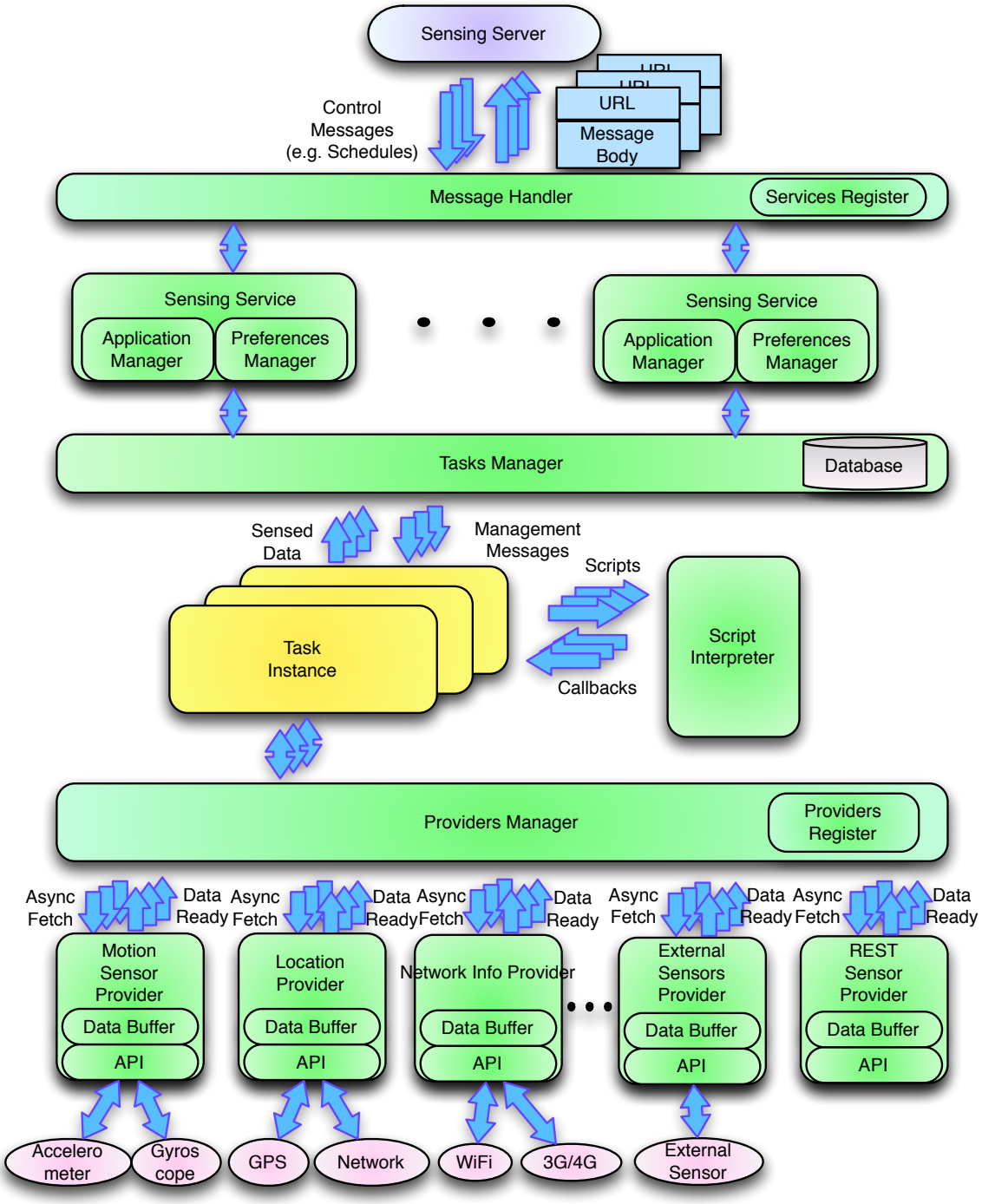


Figure 2-4: The architecture of mobile frontend

sors in a message and sends it to a sensing server. 3)It can communicate with a Google server. This is useful when a sensing server loses track of a particular smartphone, it can ask the mobile device to ping it via a Google Cloud Messaging server. 4) It can prevent a smartphone from going to sleep during communications with a server, which is implemented by using the Android system API `powerManager.newWakeupLock()`.

Every time when a sensing server needs a smartphone to undertake a sensing task, it will include all necessary information (e.g., when to sense, how to sense, etc.) in an HTTP message, which will then be sent to the Message Handler on the mobile frontend. Message handler dispatches decoded messages to the signed application, based on the applications register information. How to sense, i.e., what data to acquire, is described using the Lua [44] scripting language. As introduced in Section 2.2.1, Lua is a powerful, fast, lightweight, and embeddable scripting language. It combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics, which make it quite fit for sensing task description. Sample Lua scripts (with comments) are presented in Fig. 2-5. Note that those functions such as `get_light_readings()` and `get_location()` in the scripts are functions defined by us and will be mapped to the callback functions (for asynchronous data fetching).

```
--[[ if GPS query is permitted and the location data could be obtained,
   then, get 2 GPS location readings; otherwise, get 2 coarse locations.]]--
if fine_location_permitted == true and gps_data_available() then
    thirdeye.get_location_reading(1, 'fine');
else
    thirdeye.get_location_reading(1, 'coarse');
--[[ get system's background running tasks information]]--
thirdeye.get_running_tasks_info();
--[[ get current location's weather information]]--
thirdeye.get_weather_reading();
--[[ get 10 accelerometer's reading]]--
thirdeye.get_accelerometer_reading(10);
--[[ get user's activity recognition result]]--
thirdeye.get_activity_recognition();
```

Figure 2-5: Sensing script

Multiple *Sensing applications* could run on top of the sensing platform. These applications are independent of each other and have their own *Application Manager* and *Preferences Manager*. *Application Manager* contains the setting of the application, such as the default sensing script of the application and default uploading server's IP address. If the application is a periodic application, (such as LIPS application which we will present in Chapter 4) the application manager will also set the task period and trigger the sensing task periodically by sending the stored sensing script to *Task Manager*.

The *Preferences Manager* enables a mobile user to setup his/her preferences for each application. For example, a user could indicate he/she doesn't want to use the limited data plan for transmitting/receiving sensing data. If so, data uploading occurs only when WiFi connection is available. When WiFi connection is not available, local database is used to store sensor data acquired by the task instance. The task manager will upload the sensing results when a stable WiFi connection is available. Another example is: the user might indicate a certain time period that he/she isn't willing to provide sensing data due to privacy concerns.

The *Task Manager* manages the data collection procedure, which has four functionalities: 1) tracking all these task instances. 2) triggering task instance to collect data from sensors; 3) handling failures (such as rebooting a failed sampling task instance); 4) encapsulating sensor data and notifying the message handler to upload them to the cloud backend.

After receiving a sensing message, each incoming task will be served by a task instance, which will be hosted by a thread. The *Task Manager* will manage all the sensing instances. A task instance sends the corresponding Lua scripts to the *Script Interpreter* for translation. The interpreter can interpret both Lua's own functions and the functions we defined for data acquisition. The script interpreter tells the task instance which Java function to call to obtain data from sensors since the Android

system cannot recognize Lua scripts. Note that security can be enforced here by only allowing a white list of security-ensured functions to be called. A task instance is a self-contained component, which maintains its own status (e.g., running, waiting for data, etc.), calls proper API functions to acquire data from sensors, and manages data collected from sensors. The mobile frontend is a multi-tasking system, where concurrency is well supported. At one time, there can be multiple sensing task instances running on the frontend. Those task instances can acquire data from one or multiple sensors simultaneously.

Sensors in mobile frontend are managed by the *Sensor Manager* and providers. We create a provider for one or multiple related sensors. A provider is basically a software component which actually operates a "sensor" using APIs provided by the android system or a third party to collect data. Note that the definition of "sensor" here has a much broader meaning, which refers to data sources that can provide context information of a mobile user. Therefore a sensor could be: 1) an embedded sensor (such as GPS, accelerometer, digital compass, etc.) on a mobile phone; 2) a application that can provide context information (such as local weather) to mobile users via APIs; 3) an external sensor (such as Fitbit [19], SensorDrone [63], etc.) that can be connected to a smartphone via its network interface (such as Bluetooth); 4) a restful network application which could provide useful information, such as weather information and maps information.

Data acquisition from a sensor is done asynchronously so that an operation will not block or be blocked by others. When a sampling task instance requests data, the sensor manager directs the call to the corresponding provider to actually acquire data from sensors. Moreover, the sensor manager can cancel data acquisition if timeout. Note that each Provider maintains a data buffer which buffers data collected from its sensor and can even share them with multiple different tasks. In this way, energy consumed for sensing can be reduced.

We summarize sensors supported by the current implementation, their type (embedded or external) and the corresponding information that can be obtained from them in Table 2.1

Table 2.1: Sensors supported by mobile frontend

Sensor	Type	Usage	Unit
2G/3G/4G interface	Embedded	signal strength	dBm
Accelerometer	Embedded	Acceleration	m/s^2
Gyroscope	Embedded	Orientation	rad/s
Light sensor	Embedded	light level	Lux
Magnetic Field Sensor	Embedded	Magnetic field level	Micro-Tesla
Microphone	Embedded	relative sound Level	value in $[0, 32767]$
Proximity sensor	Embedded	Proximity distance	cm
WiFi interface	Embedded	signal strength	dBm

The architecture of the mobile frontend is scalable because new sensing application can be easily developed and integrated into it. To support a new application, the developers only need to: 1) import the application's *application Manager* module, which contains the basic settings; 2) register the application in applications Register.

Also various sensors can be easily integrated into it, which is achieved by *Sensor Manager* and *Providers*. If we want to make mobile frontend to support a new sensor (embedded or external), we only need to create a *Provider* for that sensor. The provider needs to be registered with the *Sensor Manager* via the *Provider Register*, which keeps a list of currently supported sensors and the corresponding data acquisition functions we defined (such as `get_light_readings()` and `get_location()`). When a task instance requests data by calling such a data acquisition function, the *Sensor Manager* directs the call to the corresponding Provider to actually acquire data from sensors.

In this section we have presented the implementation details of mobile frontend. Implementation of cloud backend is presented in the next section.

2.4 Cloud Backend Implementations

In this section, we present the system architecture of the cloud backend for the unified platform, and discuss related implementation details.

The architecture of cloud backend is depicted in Fig. 2-6, which consists of online modules and offline modules. In our design, the online modules are designed to be light-weighted and can provide immediate responses to the mobile frontend. Online modules support a set of applications for the mobile frontend, including login, raw data handling, messaging, task monitoring, etc. The offline modules are designed to handle computational-intensive and time-consuming workload. The offline modules can be physically separated and run on powerful servers. This design ensures that online requests from the mobile frontend are not delayed by the time-consuming processes, which include data statistical analysis, machine learning model building and query information from third-party applications, etc. The system design makes it easy to scale up the system horizontally since more computing power could be added to support offline learning without affecting the online part.

As illustrated in Fig. 2-6, A sensing server consists of *User Manager, Sensing Tasks Monitor, Sensing Data handler, Data Pre-processor, Machine Learning Modules, application Modules, REST Information Providers, Database and Memory Caches.*

The *Message Handler* in the sensing server is quite similar to its counterpart in the mobile frontend. It communicates with the mobile frontend using HTTP and dispatches incoming message to different components. *Modules Register* helps *Message Handler* to dispatch messages to appropriate online modules.

User Manager handles system login, user authentication and user information update. User Manager maintains the currently available users' information. Those information could be used as input for other modules, such as scheduling modules, which we will present for each application.

Here, a sensing task is defined as a procedure of acquiring data from sensors for

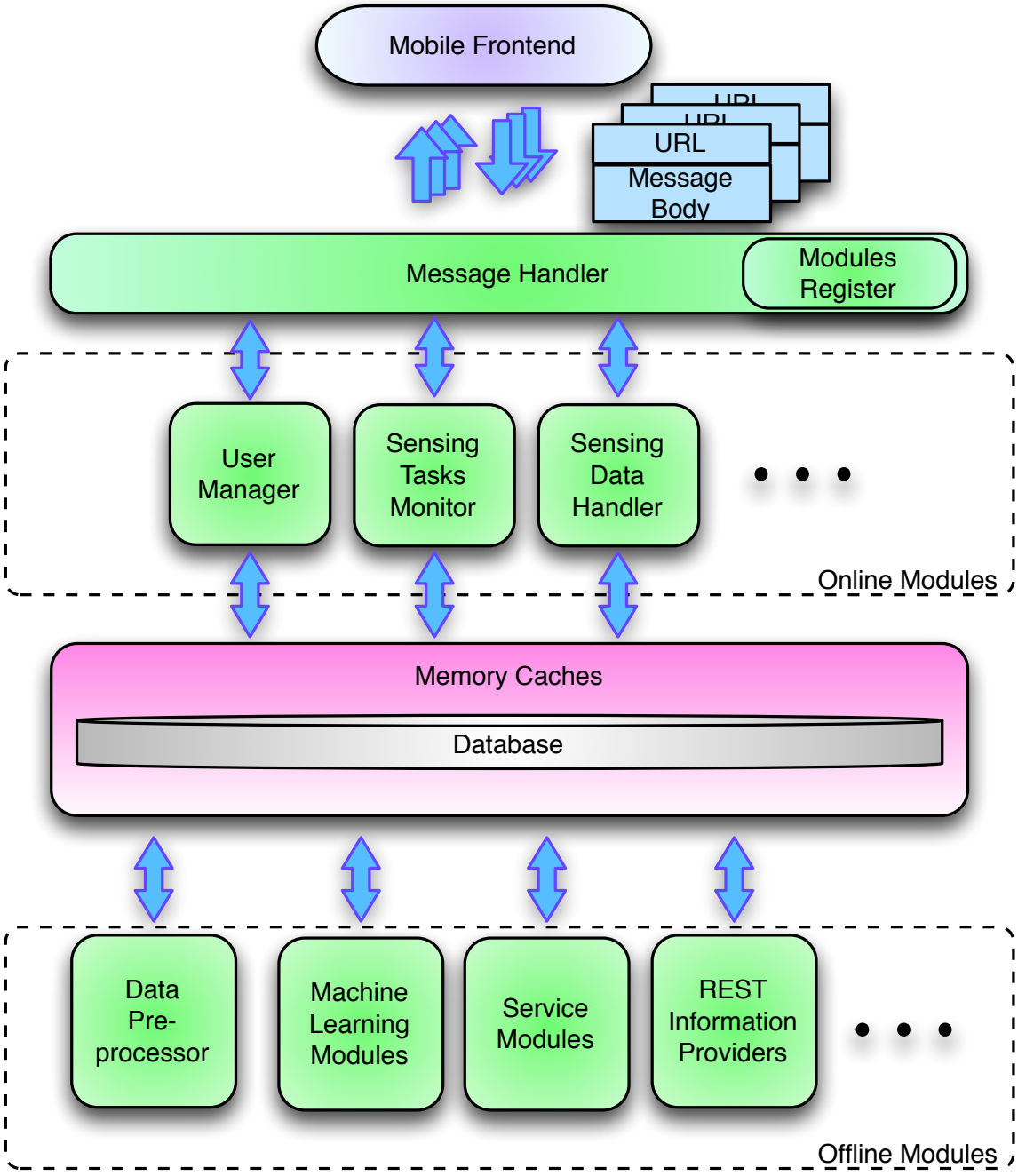


Figure 2-6: The architecture of cloud backend

a target place. The *Sensing Tasks Monitor* keeps tracking all necessary information related to sensing tasks, including its initiator (who initiates the sensing task), the Lua scripts defining the corresponding data acquisition procedure (See Fig. 2-5 for an example), and its running status(initiated, running, finished or failed).

If the received message includes sensed data, the message will be handled by *Sensing Data Handler*. The Sensing Data Handler will validate the collected sensor data: if the data doesn't comply with the corresponding sensing script (i.e. a required feature, such as location, is missing), they will be discarded; If the collected data is valid, sensing data handler will notify the *Data Pre-processor* when raw sensor data are ready.

In cloud backend, we choose the PostgreSQL [56] for storing data, which is an open-source Object Relational Database Management System (ORDBMS) with an emphasis on extensibility and standards compliance. To improve the performance, *Memory Caches* are used to cache the recent readings from the database.

Next, we introduce the offline modules. The *Data Pre-processor* decodes the binary raw data. Moreover, it also processes raw data to generate more meaningful data for various sensing features (temperature, humidity, roughness of road surface, etc), which will then be stored into the database to serve as input for other data analytic modules. The processed data are called *feature data*, which are usually statistics (average, variance, etc) of raw data.

The *Machine Learning Modules* contains frequently used machine learning algorithms for both supervised learning and unsupervised learning. *applications Modules* are a set of modules for different applications. We will present these modules in great detail for different applications.

REST Information Providers are a set of modules that collect useful information to enrich the sensing data. For example, the weather information could be obtained from weather.com by providing geo-location information. Also the place information

of an actual place (such as restaurant, coffee shop, etc.) could be retrieved from third-parties' APIs, such as Google's Place API [25], Microsoft Bing [5], HERE Maps [28] and MapQuest [47].

Since the cloud backend is designed following the module-based design idea, more online modules and offline modules could be easily added to enrich the functionalities. The added online modules and offline modules of the new application will not affect other applications' modules. All other applications' modules can be kept with no modification.

Chapter 3

Application I: *S*martphone Sensing Based *O*bjective *R*anking (*SOR*)

3.1 Overview

In this chapter, we will present the design and implementation of SOR system. SOR is a Sensing based Objective Ranking system, built on top of the proposed unified platform.

Currently, a few online review and recommendation systems have attracted millions of users and are gaining increasing popularity. For example, Yelp serves as an online urban guide, which provides user reviews, recommendations and rankings of a large variety of local businesses including restaurants, shops, theaters, etc. Another typical example is TripAdvisor, which has become the world's largest website for travelers. It provides user reviews, ratings and ranks for hotels, restaurants, attractions in different places across the whole world. These systems usually rate and rank target places and attractions based on *subjective* ratings and opinions provided by users.

As we presented in the former chapters, most of modern smartphones are equipped with a rich set of embedded sensors. Moreover, external sensors can also be connected

to a smartphone via its network interface. SensorDrone [63] is a portable and wearable multi-sensor that can turn a smartphone into an environmental monitor. This small device is equipped with 10 different sensors, including multiple gas sensors, a non-contact thermometer, a humidity sensor, a temperature sensor, a light sensor, a color sensor and a pressure sensor. It can even be connected to more sensors via an expansion connector.

So data collected via smartphone sensing can be used to evaluate a target place. For example, for a coffee shop, based on readings from microphones, we can know if it is quiet; based on readings from light sensors, we can know if it is bright; based on readings from temperature sensors (on SensorDrones), we can know if it is warm.

In this chapter, we present design, implementation and evaluation of an objective ranking system, which ranks target places based on (objective) data collected via smartphone sensing. Our objective is not to replace the current ranking/recommendation systems that are based on subjective user ratings but to enhance them with objective data collected via various sensors to provide more comprehensive and objective rankings and recommendations for users.

It is quite challenging to design such an objective ranking system. First, in order to provide a comprehensive view for target places, the system needs to leverage a large variety of embedded and external sensors to collect various data. Different types of places need different sets of sensors to collect the appropriate information for future ranking. Second, a mobile user may participate at any time. The system needs to schedule sensing activities properly to ensure a good coverage over a given scheduling period. It is certainly not desirable to have sensor readings clustered on certain short periods of time. In addition, the system needs to be able to rank a target place based on various sensed data.

In this chapter, the term *"mobile user"* refers to a person who participates in sensing activities and contributes sensed data; while the term *"user"* refers to a

person who uses SOR to find out rankings and recommendations. A person can certainly be both user and mobile user. In our design, we carefully address these challenges. SOR has the following desirable features: 1) it is easy to use because an easy 2D barcode scan triggers a sensing procedure, which is then automatically done without user involvement; 2) since the system is built on the unified platform we designed, its architecture is so scalable that various embedded and external sensors can be easily integrated into it; 3) an online scheduling algorithm is proposed and used to schedule sensing activities for coverage maximization, which has a constant approximation ratio of 1/2; 4) a personalizable ranking algorithm is developed and used to rank target places based on various sensor readings and user preferences.

We summarize our contributions as follows:

1) We design and implement an objective ranking application based on mobile phone sensing, which is the first of its kind.

2) We develop theoretically well-founded and practically efficient scheduling and ranking for the proposed system.

3) We justify effectiveness of the proposed system and algorithms via both field tests (using real hiking trails and coffee shops in Syracuse, NY as target places) and simulation.

Note that the proposed system, ranking algorithm and sensed data can be integrated into existing subjective ranking and recommendation systems to provide a more comprehensive and objective view of target places for users. However, due to space limitation, this chapter is only focused on smartphone sensing and ranking based on objective data collected by smartphones.

The rest of the chapter is organized as follows: We present the software architecture and implementation details of the proposed system in Section 3.2. The proposed scheduling and ranking algorithms are presented in Section 3.3 and Section 3.4 re-

spectively. Experimental and simulation results are presented and analyzed in Section 3.5.

3.2 Design And Implementation Of SOR

In this section, we discuss the design and implementation details of SOR application, which is built on top of the unified platform. The proposed smartphone Sensing based Objective Ranking (SOR) system is illustrated in Fig. 3-1.

In order to use it, the following components must be deployed properly: 1) *Mobile sensing frontend*: the mobile frontend which we presented in Section 2.3 needs to be installed on each participating smartphone. 2) *Cloud backend*: the cloud backend which we presented in Section 2.4 needs to be deployed to collect sensed data from smartphones. 3) *2D barcode:* A 2D barcode needs to be deployed in a target place to trigger a sensing procedure.

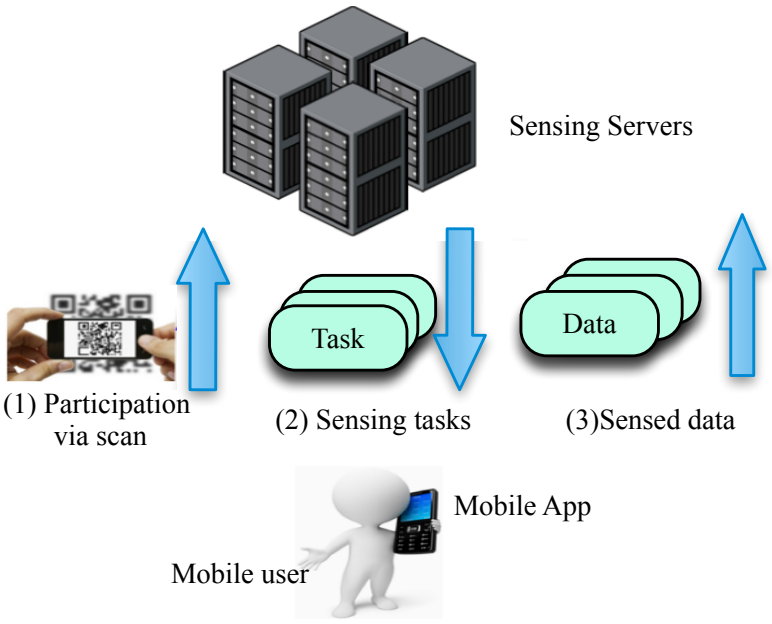


Figure 3-1: The workflow of SOR system

The workflow of SOR is as follows:

1) If a mobile user decides to participate, he/she opens the mobile application and scans the 2D barcode in the target place, which will send a notification (with information about the target place contained in the barcode) to a sensing server and trigger a sensing procedure.

2) A sensing server detects the incoming participation request, calculates a sensing schedule and sends the corresponding sensing tasks to the mobile phone.

3) The mobile application operates sensors to sense according to the provided schedule and sends sensed data back to a sensing server.

4) The sensing server collects and processes sensed data from smartphones, and stores them into a database.

5) A ranking program ranks the target place based on data collected from smartphones and user preferences.

We extend the mobile frontend of the proposed unified platform, which presented in Section 2.3, to support SOR application. As highlighted with orange in Fig. 3-2, two new modules, *SOR Module* and *SensorDrone Provider*, are integrated into the platform. Of course, new application module and new provider are need to be registered in applications Register and Providers Register accordingly. The rest of the parts works with no modification. For simplicity, we didn't plot the details of these modules with no modification, such as Existing Sensing applications, Existing Sensors Providers, Tasks Manager, Providers Manager. Reader could refer Section 2.3 for implementation details.

The *SOR module* contains the following functionalities: 1) communicate with the camera on smartphone and get picture from it; 2) interpret a picture containing QR code to a string; 3) communicate with SOR backend server by sending message through Message Handler; 4) trigger data acquisition tasks and report sensing results back to cloud backend; 5) store user's preferences and protect user's privacy.

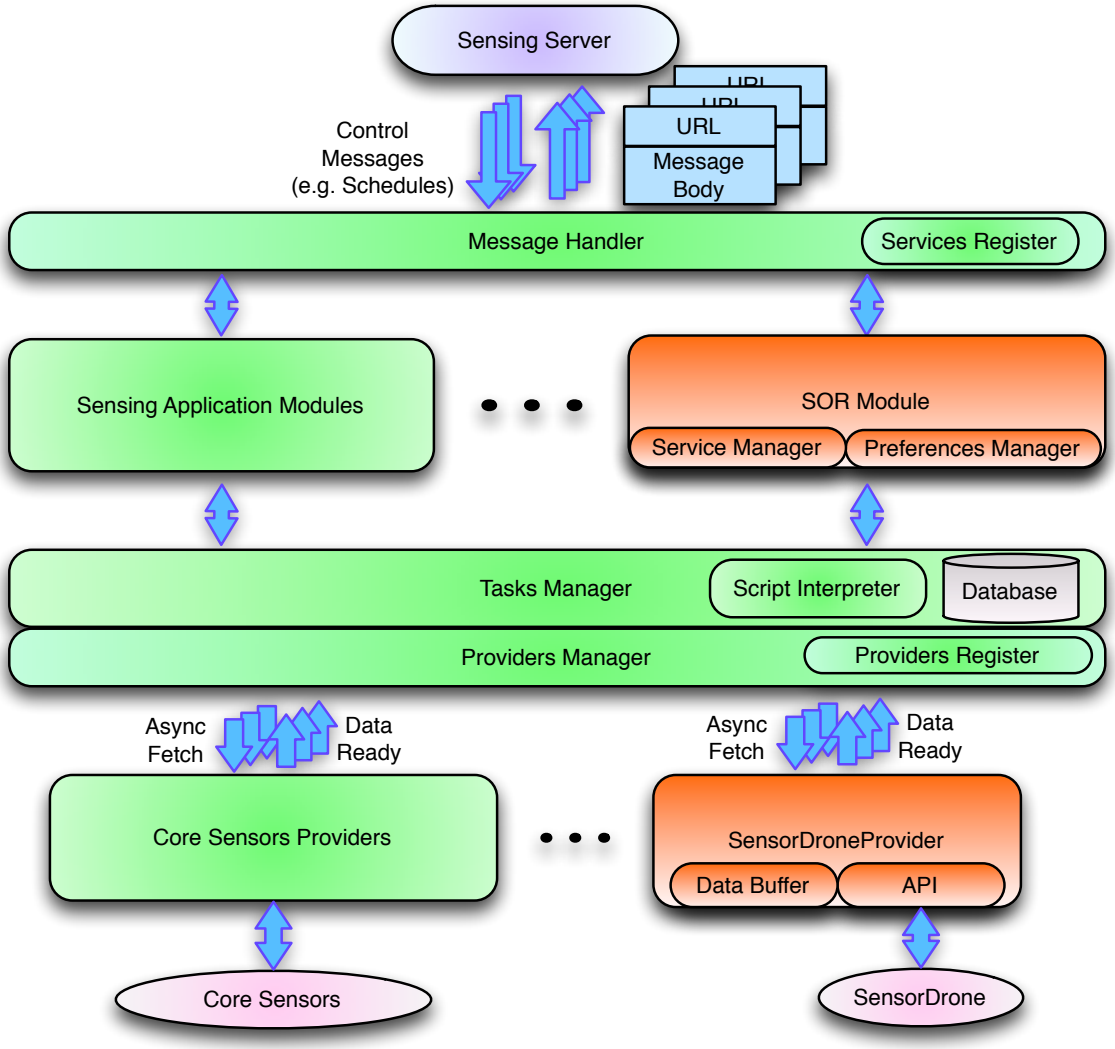


Figure 3-2: Mobile frontend of SOR

The *SensorDrone Provider* wraps the official APIs of SensorDrone, and perform data collection from SensorDrone asynchronously. SensorDrone provider has a local cache which enables data sharing among tasks with same data requirements. After integrating SensorDrone Provider into the platform, more sensors are supported by mobile frontend, as show in Table 3.1:

Similarly, the backend for SOR application is also built on top of the sensing platform's cloud backend, as illustrated in Fig. 3-3. The *Existing Online Modules* and *Existing Offline Modules* in the figure are those modules with no modifications, include: *User Info Manager*, *Sensing Tasks manager*, *Sensing Data Handler* and *Data*

Table 3.1: Sensors supported by SensorDrone provider

Sensor	Type	Usage	Unit
Air pressure sensor	External(SensorDrone)	Air pressure	m/s^2
Altitude sensor	External(SensorDrone)	Altitude	Feet
Gyroscope	Embedded	Orientation	rad/s
Humidity sensor	External (SensorDrone)	Relative humidity	Ratio
Light sensor	External(SensorDrone)	light level	Lux
Temperature sensor	External(SensorDrone)	Temperature	Fahrenheit

Pre-processor. Detailed introduction is giving in Section 2.4

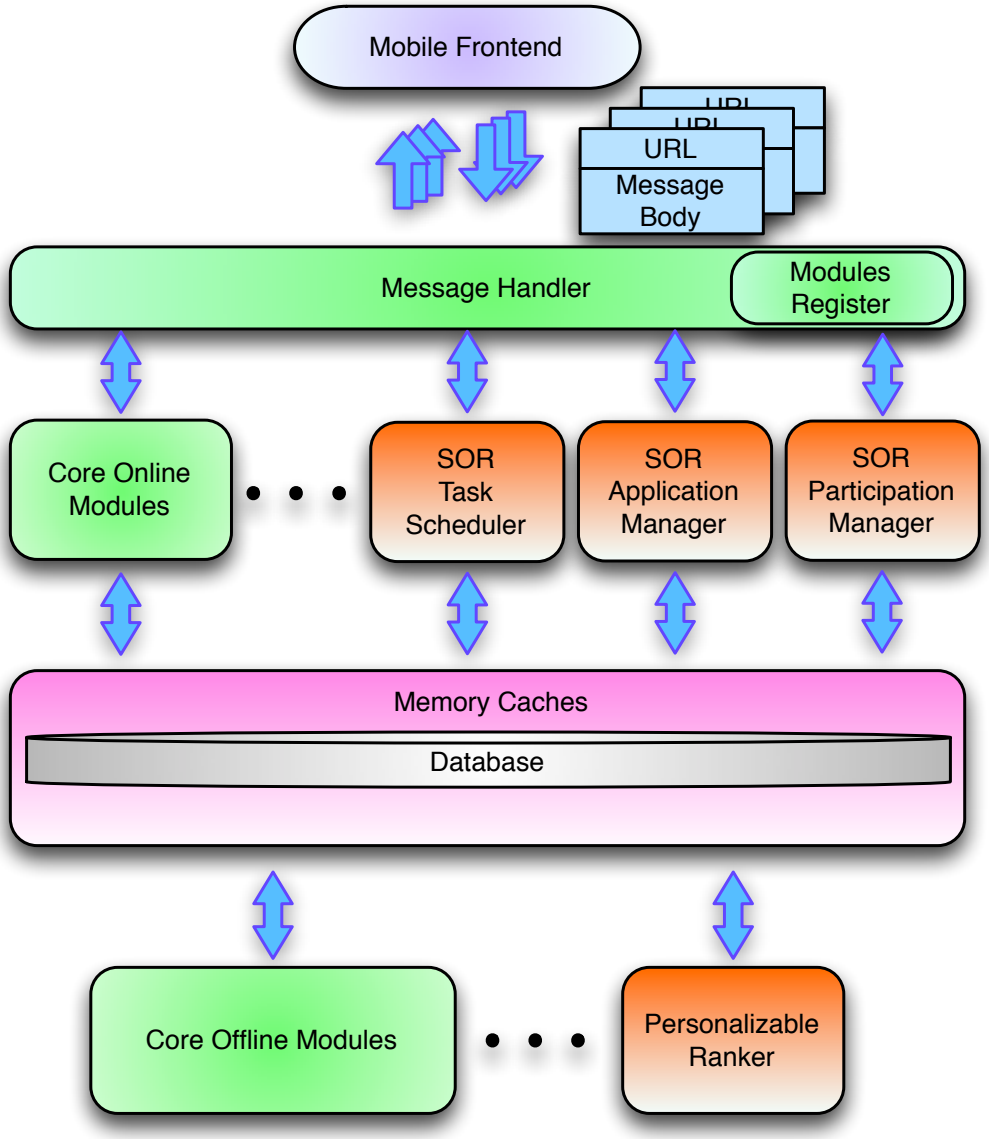


Figure 3-3: Cloud backend of SOR

The added-on modules for SOR application is highlighted in orange color. Those

modules include: *SOR Application Manager*, *SOR Participation Manager* and *Personalizable Ranker*.

The *SOR Application Manager* manages all necessary information related to each SOR application. Here, an *application* is defined as a procedure of acquiring data from sensors for a target place, which may include multiple sensing tasks. The SOR Application Manager handles the SOR applications' information including its AppID, its creator (which could be the owner/manager/operator of the corresponding target place), the Lua sensing scripts defining the corresponding data acquisition procedure and the progress status of the application.

The *SOR Participation Manager* keeps track of a list of sensing tasks and their information, including participating userID, the corresponding token, the corresponding application, the location of the target place, the sensing budget and its status (such as running, waiting for sensing schedule, finished, error, etc.). Note that the sensing budget is an integer number specifying how many times the corresponding mobile user can perform data acquisition task defined by the corresponding Lua scripts. Initially, it is set to the maximum number of times the mobile user is willing to acquire data from its sensors and it is updated at runtime. Every time when a mobile user scans a 2D barcode, the Participation Manager will first verify whether the user is actually in the target place by acquiring its location and comparing it against the location stored in the Application Manager, and then create a task for it if the user is considered as a truthful user. Moreover, a mobile user's status in the Participation Manager will be changed to "finished" if according to his/her location, he/she leaves the target place.

For each application, the *SOR Task Scheduler* applies an online algorithm to calculate a sensing schedule (that specifies when to sense for each participating user) for a scheduling period based on runtime tasks' information (such as current participating users, their sensing budgets, etc.) provided by Sensing Tasks Manager. Task

Scheduler will select a suitable subset of users to fulfill the sensing tasks. The Tasks Scheduler will also distribute the calculated schedules along with the corresponding Lua scripts to participating mobile phones, and store them into the database. The scheduling algorithms are varying among different sensing applications. We will describe the proposed scheduling algorithm in greater detail in Section 3.3.

In the *Existing Offline Modules*, The *Data Pre-processor* processes the data and stores useful information into corresponding tables in the database for various sensing features (temperature, humidity, roughness of road surface, etc.). These features data will then be served as input for the *Personalizable Ranker*. The processed data is called *feature data*, which are usually statistics (average, variance, etc.) of raw data. The *Personalizable Ranker* leverages a personalizable ranking algorithm to rank target places according to feature data and user preferences. Both data processing and personalizable ranking will be discussed in Section 3.4.

3.3 Scheduling Algorithm

In this section, we will our sensing model, and then present the online scheduling algorithm used in the SOR system.

In our sensing model, we use a set $\mathbf{T}$ of N time instants to divide the time domain within a sensing scheduling period $[t^S, t^E]$ into small time intervals with equal durations. The measurements are scheduled to be taken only at these time instants. Of course, the larger the N, the more accurate the measurement, however, the higher the sensing cost (such as energy consumption). If a sensing feature is measured at time t_i, then we say time instant t_j can be *estimated* by using the data collected at time t_i with a probability of $p(t_i, t_j)$. Note that we aim to come up with a general sensing model, in which any method or distribution model can be used to obtain this probability according to application needs. In our implementation, we chose to use a

bell-shaped probability distribution $N(\mu, \sigma)$, such that the closer t_j is to t_i, the more likely that the sensor reading at t_j stays the same as that at t_i. This is consistent with most sensing features, such as temperature, humidity, accelerometer, wireless signal strength, etc. Moveover, different variances σ can be used to model different sensing features. A large σ can be used for those sensing features whose readings do not change drastically over time (such as temperature, humidity, etc.), while a small σ can be used for those whose readings may change quickly (such as acceleration, orientation, wireless signal strength, etc.). This model has been adopted in a few related works such as [40].

A sensing schedule can be given as a set $\mathbf{\Phi}$ of time instants. Since samplings are independent from each other, the probability that time instant t_j can be covered by the given sensing schedule $\mathbf{\Phi}$ is:

$$p(t_j, \mathbf{\Phi}) = 1 - \prod_{t_i \in \mathbf{\Phi}} (1 - p(t_i, t_j)). \tag{3.1}$$

Suppose that we are given a set $\mathbf{T}$ of equally spaced instants within a scheduling period $[t^S, t^E]$ as well as the duration a mobile user k participating in sensing activities $[t_k^S, t_k^E]$, then $\mathbf{T}_k \subseteq \mathbf{T}$ is a subset of time instants in $\mathbf{T}$ that falls in $[t_k^S, t_k^E]$. A sensing schedule of user k, $\mathbf{\Phi}_k$, is a subset of time instants in $\mathbf{T}_k$. In addition, every mobile user k has a sensing budget N_k^B, which is the number of times he/she is willing to sense during a scheduling period. We are interested in solving the following problem:

$$\max_{\{\mathbf{\Phi}_1, \cdots, \mathbf{\Phi}_K\}} \sum_{t_j \in \mathbf{T}} \sum_{k=1}^{K} p(t_j, \mathbf{\Phi}_k) \tag{3.2}$$

Subject to:

$$|\mathbf{\Phi}_k| \leq N_k^B, k \in \{1, \cdots, K\}. \tag{3.3}$$

The scheduling problem is to maximize the total sensing coverage probability by selecting a sensing schedule $\mathbf{\Phi}_k$ for each participating mobile user k, with a cardinality

no more than the given budget N_k^B. The goal here is to spread measurements across the whole sensing period and in the meanwhile, ensure fairness by preventing certain mobile users from being abused.

We construct a collection of subsets of the ground set $\mathbf{T}$, $\mathbf{\Lambda} = \{\mathbf{\Psi} : \mathbf{\Psi} \subseteq \mathbf{T}, |\mathbf{\Psi} \bigcap \mathbf{T}_k| \leq N_k^B, k \in \{1, 2, \cdots, K\}\}$. Next, we show that $(\mathbf{T}, \mathbf{\Lambda})$ is a matroid.

Definition 1 (Matroid [21]) *A pair (Q, Z) consisting of a ground set Q and a collection Z of subsets of Q is a matroid if:*

1) $\emptyset \in Z$;

2) If $X \in Z$ and $Y \subset X$, then $Y \in Z$;

3) for all $X, Y \in Z$, if $|X| > |Y|$ then there exists some $x \in X \setminus Y$ such that $Y \bigcup \{x\} \in Z$.

Theorem 1 $(\mathbf{T}, \mathbf{\Lambda})$ *is a matroid.*

It is easy to see that $\emptyset \in \mathbf{\Lambda}$. Suppose that $\mathbf{\Psi}_1 \in \mathbf{\Lambda}$. According to the definition of $\mathbf{\Lambda}$, $\mathbf{\Psi}_1$ satisfies the constraint $|\mathbf{\Psi}_1 \bigcap \mathbf{T}_k| \leq N_k^B, k \in \{1, \cdots, K\}$. And if $\mathbf{\Psi}_2 \subset \mathbf{\Psi}_1$ then we have $|\mathbf{\Psi}_2 \bigcap \mathbf{T}_k| \leq |\mathbf{\Psi}_1 \bigcap \mathbf{T}_k| \leq N_k^B, k \in \{1, \cdots, K\}$. So $\mathbf{\Psi}_2 \in \mathbf{\Lambda}$.

We prove that condition 2) is also satisfied by contradiction. Suppose that $\mathbf{\Psi}_1 \in \mathbf{\Lambda}, \mathbf{\Psi}_2 \in \mathbf{\Lambda}$, and $|\mathbf{\Psi}_1| > |\mathbf{\Psi}_2|$, but there does not even exist any element x such that $x \in \mathbf{\Psi}_1 \setminus \mathbf{\Psi}_2$ and $\mathbf{\Psi}_2 \bigcup \{x\} \in \mathbf{\Lambda}$. If this statement is NOT true, then $\mathbf{\Psi}_2 \bigcup \{x_k\} > N_k^B, \forall x_k \in \{\mathbf{\Psi}_1 \setminus \mathbf{\Psi}_2\} \bigcap \mathbf{T}_k, k \in \{1, \cdots, K\}$. This means for any k, if $\{\mathbf{\Psi}_1 \setminus \mathbf{\Psi}_2\} \bigcap \mathbf{T}_k \neq \emptyset$, then $|\mathbf{\Psi}_2 \bigcap \mathbf{T}_k| = N_k^B$. So $|\mathbf{\Psi}_1 \bigcap \mathbf{T}_k| \leq |\mathbf{\Psi}_2 \bigcap \mathbf{T}_k|, \forall \{\mathbf{\Psi}_1 \setminus \mathbf{\Psi}_2\} \bigcap \mathbf{T}_k \neq \emptyset, k \in \{1, \cdots, K\}$. And since $\{\mathbf{\Psi}_1 \setminus \mathbf{\Psi}_2\} \subset \mathbf{\Psi}_1$, and all the elements in $\mathbf{\Psi}_1 \bigcap \mathbf{\Psi}_2$ are shared by both $\mathbf{\Psi}_1$ and $\mathbf{\Psi}_2$, $|\mathbf{\Psi}_1| \leq |\mathbf{\Psi}_2|$, which is in contradiction to our assumption. This completes our proof.

The scheduling problem can be re-formulated as:

$$\max_{\mathbf{\Psi}\in\mathbf{\Lambda}} \sum_{t_j\in\mathbf{T}} p(t_j, \mathbf{\Psi}). \tag{3.4}$$

This scheduling problem falls in a class of problems of maximizing a sub-modular set function over a matroid [21] because its objective function $f(\mathbf{\Psi}) = \sum_{t_j\in\mathbf{T}} p(t_j, \mathbf{\Psi})$ has been shown to be a non-negative, monotone and sub-modular function in [74] and we show that $(\mathbf{T}, \mathbf{\Lambda})$ is a matroid in Theorem 1. We present a simple greedy algorithm to solve it in the following.

Algorithm 1 The sensing scheduling algorithm

Input: $\mathbf{T}$ $\mathbf{\Lambda}$;
Output: $\mathbf{\Psi}$;

1: $\mathbf{\Psi}_0 := \emptyset$; $l := 1$;
2: **while** $\exists x \in \mathbf{T} \setminus \mathbf{\Psi}_{l-1}$ s.t. $\mathbf{\Psi}_{l-1} \bigcup \{x\} \in \mathbf{\Lambda}$ **do**
3: $\quad x^* := \text{argmax}_{x'\in\mathbf{T}\setminus\mathbf{\Psi}_{l-1}} f(\mathbf{\Psi}_{l-1} \bigcup \{x'\}) - f(\mathbf{\Psi}_{l-1})$;
4: $\quad \mathbf{\Psi}_l := \mathbf{\Psi}_{l-1} \bigcup \{x^*\}$;
5: $\quad l := l + 1$;
6: **end while**
7: **return** $\mathbf{\Psi}_l$;

The basic idea of the proposed algorithm is to keep adding the time instant, that can result in the maximum incremental coverage until no mobile users can be scheduled to sense more without violating their budget constraints, into the solution. The running time of this algorithm is $O(|\mathbf{T}|^2 \cdot g(|\mathbf{\Lambda}|))$, where $g(|\mathbf{\Lambda}|)$ is the running time for testing whether $\mathbf{\Psi}_{l-1} \bigcup \{s\} \in \mathbf{\Lambda}$ or not. In our algorithm, this can be quickly done in constant time by maintaining a counter for each mobile user and checking if its value exceeds the given budget. So the overall time complexity is $O(|\mathbf{T}|^2) = O(N^2)$. Hence the proposed algorithm is time efficient.

In addition, according to Theorem 1, the scheduling problem is to maximize a non-decreasing sub-modular set function over a matroid. It has been shown in [21],

that a simple greedy algorithm (similar to that shown above) gives a $\frac{1}{2}$-approximation for a class of such problems. Therefore, Algorithm 1 is a $\frac{1}{2}$-approximation algorithm for the scheduling problem (3.4).

3.4 Data Processing And Personalizable Ranking

In this section, we discuss how raw data collected from mobile frontend are processed and fed to the ranking algorithm as input to calculate ranks for a target place.

3.4.1 Data Processing

In SOR, for a target place, data collected by sensors of certain type in a given scheduling period are stored as a set of 3-tuples $(t, \Delta t, \mathbf{d})$. t is the timestamp, whereas, Δt is a short period of time (typically several seconds). Note that SOR takes multiple (instead of one) readings within $[t, t+\Delta t]$ to ensure high sensing quality. The number of readings to be taken during this period can be specific in the Lua scripts. $\mathbf{d}$ is the corresponding set of readings.

Ranking is conducted based on values of a set of *humanly understandable features*, such as temperature, WiFi signal strength, roughness of road surface. For a target place, raw data need to be processed to calculate a value for each feature, which will then be used by the ranking algorithm as input. Note that the methods for calculating these values from raw data may vary with features. For example, for temperature, we take an average over all temperature sensors' readings; however, for roughness of road surface, we take an average of standard deviations of accelerometers' readings within Δt. Readings from different types of sensors may be combined to generate the value for a feature too. SOR calculates these statistics (*feature data*) and stores them into the database. When they are needed for ranking, they are read from the database into a matrix $\mathbf{H} =< h_{ij} >, i \in \{1, \cdots, N\}, j \in \{1, \cdots, M\}$, where N and

M are the numbers of target places and features respectively. For simplicity, we focus on places belonging to a certain category (such as coffee shop, hiking trail, etc) here. SOR can certainly deal with multiple categories by using multiple such matrices.

However, data in $\mathbf{H}$ cannot be directly used for ranking since the purpose of ranking is to recommend suitable places for individual users. If ranking is done on the absolute temperature, then a very hot (or cold) place may be ranked one of top places, which is certainly not preferred by most people. Hence, our personalizable ranking algorithm will further process these values based on user preferences, which will be discussed next.

3.4.2 Personalizable Ranking Algorithm

In this section, we present a personalizable ranking algorithm based on user preferences. Our algorithm uses the same sensed data as input for all users but can produce different rankings for different users based on their preferences. The input of the algorithm includes: 1) $H =< h_{ij} >, i \in \{1, \cdots, N\}, j \in \{1, \cdots, M\}$ (read from the database); 2) $\mathbf{U} =< u_j >, j \in \{1, \cdots, M\}$, where u_j is the value preferred by the user on feature j; 3) $\mathbf{W} =< w_j >, j \in \{1, \cdots, M\}$, where w_j is the weight given by the user on feature j to express his/her emphasis; We outline the proposed algorithm in the following and then explain every step in details.

Algorithm 2 Personalizable Ranking Algorithm

1: Process $\mathbf{H} =< h_{ij} >$ further and store results to a new matrix $\mathbf{\Gamma} =< \gamma_{ij} >$ according to user preferences by $\gamma_{ij} := |h_{ij} - u_j|, i \in \{1, \cdots, N\}, j \in \{1, \cdots, M\}$;
2: Sort the target places in $\mathbf{\Gamma} =< \gamma_{ij} >$ on a column by column basis to produce an individual ranking $\mathbf{R}_j$ for every feature j;
3: Aggregate individual rankings to output a final ranking based on $\mathbf{W} =< w_j >, j \in \{1, \cdots, M\}$ using a min-cost flow based algorithm (described below).

In the first step, the algorithm calculates the distances between numbers in $\mathbf{H}$ and the values preferred by a user and then store them into another $N \times M$ matrix

$\mathbf{\Gamma} =< \gamma_{ij} >$. For example, suppose that the temperature (suppose it is the jth feature) in target place i is h_{ij}, then $\gamma_{ij} := |h_{ij} - u_j|$, where u_j is the temperature preferred by the user. If the user does not input a desirable temperature, the system provides a default value, e.g. 73°F, based on common sense. Moreover, for some features (such WiFi signal strength), if it is always the larger (the smaller) the better, then a very large (small) default value is always used as the preferred value.

In the second step, for all target places belonging to a category (such as coffee shop or hiking trail), the algorithm produces a ranking $\mathbf{R}_j$ (i.e a sorted list) on each feature j by sorting all the target places in ascending order of the corresponding feature values on the column by column basis. We call such rankings *individual rankings* in the following.

In the third step, the algorithm *aggregates* individual rankings produced (based on a single feature) in the second step to generate the final ranking. In order to do it, we need a metric measuring distance between two rankings. In SOR, the *Kemeney distance* [32, 33] is chosen for this purpose. It has been shown in [17] that Kemeney distance based ranking aggregation has good spam resistance, compared to other ranking algorithms. This method has been widely and successfully used for ranking in various applications such as webpage ranking, consensus and etc.

Suppose that an index function $\pi(i, \mathbf{R})$ returns the index of item i (target place i in our case) in ranking $\mathbf{R}$.

Definition 2 (Kemeney Distance [32, 33]) *The Kemeney distance between two rankings* $\mathbf{R}_1$ *and* $\mathbf{R}_2$*,*

$$d_K(\mathbf{R}_1, \mathbf{R}_2) = \sum_{i=1}^{N} \sum_{i'=1}^{N} \mathbf{1}(sgn((\pi(i, \mathbf{R}_1) - \pi(i', \mathbf{R}_1)) * (\pi(i, \mathbf{R}_2) - \pi(i', \mathbf{R}_2))) < 0), \tag{3.5}$$

where $\mathbf{1}(\cdot)$ *is the indicator function and* $sgn(\cdot)$ *is the sign function.*

Intuitively, the Kemeney distance counts the number of pairwise violations between two rankings. For example, given two rankings of three items $\{A, B, C\}$:

$$\begin{aligned} \mathbf{R}_1 &: A, B, C \\ \mathbf{R}_2 &: B, C, A \end{aligned} \tag{3.6}$$

then the Kemeney distance between them is $d_K(\mathbf{R}_1, \mathbf{R}_2) = 2$, since there are two pairwise violations, (A, B) and (A, C).

The Kemeney distance based method can aggregate multiple individual rankings to produce a single ranking. In SOR, we enhance personalizable ranking further by allowing users to assign different weights to emphasize (or de-emphasize) different features. This leads to a more complicated weighted ranking problem, which has not been well studied. Specifically, let $\mathbf{\Omega}$ be a collection of M individual rankings on all features $\mathbf{\Omega} = \{\mathbf{R}_j : j \in \{1, \cdots, M\}\}$. We come up with a new metric, called *weighted K-ranking distance*, to evaluate the quality of a ranking based on user preferences.

Definition 3 (Weighted K-Ranking Distance) *The weighted K-ranking distance from a ranking* $\mathbf{R}$ *to a collection of individual rankings* $\mathbf{\Omega}$ *is:*

$$\kappa_K(\mathbf{R}, \mathbf{\Omega}) = \sum_{j=1}^{M} w_j * d_K(\mathbf{R}, \mathbf{R}_j), \tag{3.7}$$

where w_j *is the weight assigned to feature* j *by the user.*

The ranking problem is to find a ranking $\mathbf{R}^*$ such that its weighted ranking distance to $\mathbf{\Omega}$ is minimized among all rankings, i.e.,

$$\mathbf{R}^* = \underset{\mathbf{R}}{\operatorname{argmin}}\, \kappa_K(\mathbf{R}, \mathbf{\Omega}). \tag{3.8}$$

Unfortunately, as showed by [17], computing such an optimal (aggregated) ranking $\mathbf{R}^*$ is NP-hard, even for the unweighted case with $|\mathbf{\Omega}| = 4$. So we need to have an effective heuristic algorithm.

Spearman's footrule distance [15], $d_f(\cdot)$, has been widely used to approximate the Kemeney distance:

$$d_f(\mathbf{R}_1, \mathbf{R}_2) = \sum_{i=1}^{N} |\pi(i, \mathbf{R}_1) - \pi(i, \mathbf{R}_2)|, \tag{3.9}$$

where $\mathbf{R}_1$ and $\mathbf{R}_2$ are two rankings as discussed above. The footrule distance has the following property [15]:

$$d_K(\mathbf{R}_1, \mathbf{R}_2) \leq d_f(\mathbf{R}_1, \mathbf{R}_2) \leq 2d_K(\mathbf{R}_1, \mathbf{R}_2). \tag{3.10}$$

Similarly, we define the *weighted f-ranking distance* as:

$$\kappa_f(\mathbf{R}, \mathbf{\Omega}) = \sum_{j=1}^{M} w_j * d_f(\mathbf{R}, \mathbf{R}_j). \tag{3.11}$$

Instead of solving the original ranking problem defined above, we can solve a footrule distance based ranking problem:

$$\mathbf{R}^* = \underset{\mathbf{R}}{\operatorname{argmin}}\, \kappa_f(\mathbf{R}, \mathbf{\Omega}). \tag{3.12}$$

It has been shown in [17] that the *unweighted* of the footrule distance based ranking problem could be transferred to a minimum cost perfect matching problem, which can be solved efficiently in polynomial time.

Next, we show that our *weighted* version can be efficiently solved by constructing an auxiliary flow graph and using a min-cost flow based algorithm. First, we construct a flow graph to assist computation $\mathbf{G}(\mathbf{V} \bigcup \mathbf{V}' \bigcup \{s, z\}, \mathbf{E})$. In this graph, each vertex

$v_i \in \mathbf{V}$ corresponds to a target place i and each vertex $v_{i'} \in \mathbf{V}'$ corresponds to a rank. There is a directed edge $e \in \mathbf{E}$ from each $v_i \in \mathbf{V}$ to every $v_{i'} \in \mathbf{V}'$, whose cost is set to $\sum_{\mathbf{R}_j \in \mathbf{\Omega}} w_j * |\pi(i, \mathbf{R}_j) - i'|$ and capacity is set to 1. Note that for a target place i, the cost here basically gives the sum of distances to all individual rankings (suppose that its final rank is i'). Moreover, to complete a flow graph, we introduce a virtual source s, which has a directed edge to each $v_i \in \mathbf{V}$ with a cost of 0 and a capacity of 1. And, there is a virtual sink z, which has a directed edge coming from each $v_{i'} \in V'$ with a cost of 0 and a capacity of 1 too.

The importance of the flow graph lies in the fact that a min-cost $s - z$ flow with an amount of N on the graph gives a ranking that minimizes the weighted f-ranking distance. It is known that the min-cost flow in such a flow graph (whose link capacities are all 1) can be efficiently found by a linear programming based algorithm [2], which is guaranteed to generate an integer flow since the corresponding co-efficient matrix is totally unimodular. Moreover, it can be easily shown that the optimal solution to our footrule distance based ranking problem is a $\frac{1}{2}$-approximate solution to the original (Kemeney distance based) problem due to the property (3.10) described above.

3.5 Validation And Performance Evaluation

We validated and evaluated SOR via both field tests and simulation. Specifically, we field-tested two kinds of places, hiking trails and coffee shops, in or around the city of Syracuse; and we evaluated the performance of the proposed scheduling algorithm via simulation.

3.5.1 Field Tests For Hiking Trails

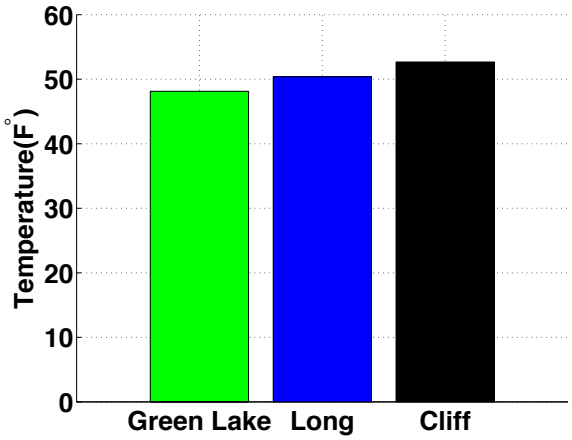
In the first sets of field tests, we collected data from three hiking trails in or around Syracuse, namely, the *Green Lake Trail* [26] (in the Green Lake State Park), the *Long*

Trail and the *Cliff Trail*(both of them are in the Clark Reservation [7]) The field tests were conducted during 11:00AM-2:00PM Nov. 17, 2013. In each test, there were 7 participating smartphones, which are all Google's Nexus4 smartphones.

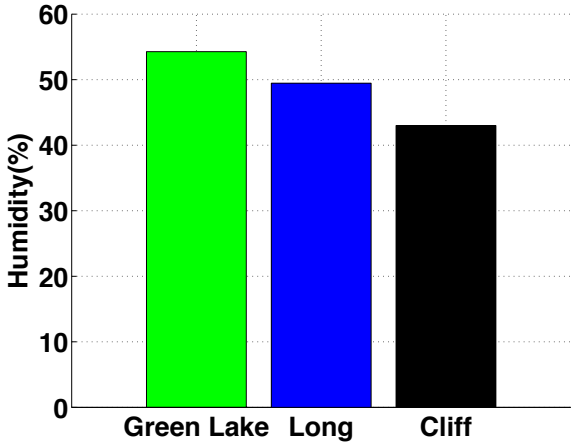
For hiking trails, we collected data of 5 sensing features that hikers usually care about most (listed below). We used the following methods to process sensed data to produce values for each feature: 1) temperature: it is an average of all temperature sensor readings; 2) humidity: it is an average of all humidity sensor readings; 3) roughness of road surface: it is an average of the standard deviations of all accelerometer's readings within Δt (a short sampling period described in section 3.4); 4) curvature: it is calculated based on GPS locations using the method presented in [39]; 5) altitude change: it is the standard deviation of averages of all altitude sensor readings within Δt. The feature data are presented in Figure 3-4.

In order to justify effectiveness of personalizable ranking in SOR, we came up with three virtual hikers, namely, *Alice, Bob* and *Chris*, whose preferences are described using hiker profiles shown in Fig. 3-5. Note that a user can express his/her preferences by setting preferred feature values and weights. The weight can be set to an integer in $\{0, 1, 2, 3, 4, 5\}$ with '0' meaning that he/she doesn't care and '5' indicating he/she really cares. For example, Alice is assumed to be an experienced hiker who prefers difficult trails. So she sets all the preferred values for the roughness, curvature and altitude change to MAX (a relatively large integer pre-configured in SOR), and sets all their weights to 5. We then present the rankings of the three target hiking trails computed by SOR via smartphone sensing for three hikers in Table 3.2.

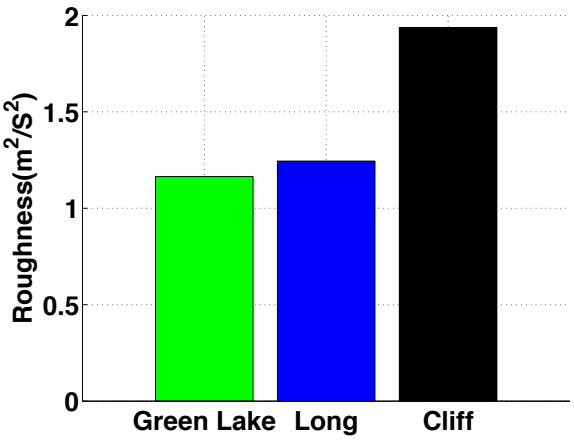
To validate these ranking results, we established the *ground truths* using pictures taken during field tests and real user comments collected from Internet via Google (mainly from *www.cnyhiking.com, www.outdoorexperiencereview.com www.hikespeak.com, nysparks.com, etc*), which are shown in Fig. 3-6 and Fig. 3-7 respectively. Note that in the table, we also summarize user comments as key opinions for quick references.



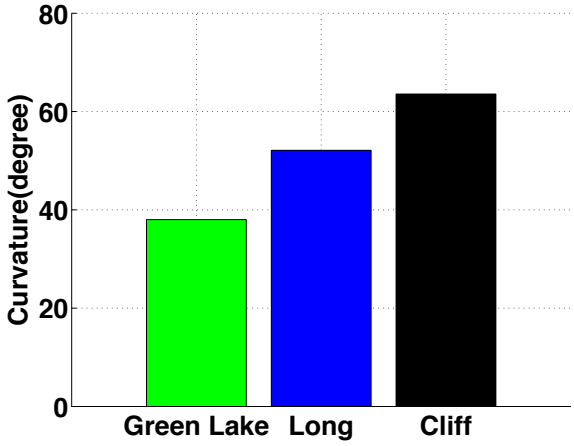
(a) Temperature



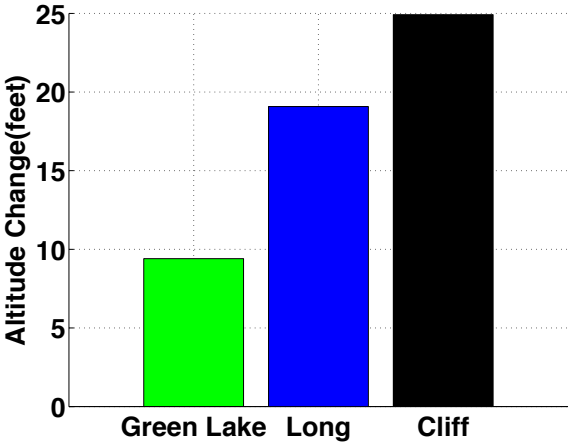
(b) Humidity



(c) Roughness of road surface



(d) Curvature



(e) Altitude change

Figure 3-4: Feature data for hiking trails

Alice

An experienced hiker who prefers difficult trails

Feature	Pre. Val.	Weight
Temp.	73 (Default)	3
Humi.	30%(Default)	3
Roug.	Max	5
Curv.	Max	5
Alti.	Max	5

(a) Alice's profile

Bob

A beginner who likes dry and even trails.

Feature	Pre. Val.	Weight
Temp.	73 (Default)	3
Humi.	Min	5
Roug.	Min	5
Curv.	Min	0 (does not care)
Alti.	Min	3

(b) Bob's profile

Chris

A beginner who likes jogging near a lake/sea/river.

Feature	Preferences	Weight
Temp.	73 (Default)	0 (does not care)
Humi.	55%	5
Roug.	Min	5
Curv.	Min	5
Alti.	Min	5

(c) Chris's profile

Figure 3-5: Hiker profiles

Table 3.2: Rankings of hiking trails computed by SOR

User	No. 1	No. 2	No. 3
Alice	Cliff Trail	Long Trail	Green Lake Trail
Bob	Long Trail	Cliff Trail	Green Lake Trail
Chris	Green Lake Trail	Long Trail	Cliff Trail

From these ground truths, we can see that the Cliff Trail is rocky so it is indeed a difficult trail. The other two trails are flat and fairly easy, especially the Green Lake trail (according to a real user comment "...This trail is almost entirely flat"). In addition, the Green Lake Trail is around a lake (see its picture) so it is supposed to be humid and a little cooler. According to rankings produced for Alice (an experienced hiker who prefers difficult trails), Cliff Trail is ranked No. 1, followed by the Long Rail (which is a little more difficult than the Green Lake Trail). Similarly, for Bob (a beginner who likes dry and even trails), the Long Trail is recommended as the top choice, followed by the Cliff trail, which is difficult but drier than the Green Lake Trail. Since Bob cares more about humidity than difficulty (according to the corresponding weights), so Cliff Trail is ranked higher than Green Lake Trail. For Chris (a beginner who likes jogging near a lake/sea/river), the Green Lake Trails is certainly recommended as the first choice. We can conclude that data collected and processed by SOR can well capture characteristics of target places, and personalizable rankings produced by SOR can well match user preferences.

Figure 3-6: Ground truth 1: pictures of the target hiking trails

3.5.2 Field Tests For Coffee Shops

In the second sets of field tests, we collected data from three coffee shops in Syracuse, namely, the *Tim Hortons* (985 East Brighton Avenue Syracuse, NY 13205), the *Barns & Noble (B&N) Cafe* (3454 E. Erie Blvd, Syracuse, NY, 13214) and a *Starbucks* (177

	Real Users' Comments	Key Opinions
Green Lake	"…trails around the lake to run…" "…this is a great running trail…" "…trails… are wide and mostly flat …quite suitable for beginner…" "…The trail is almost entirely level…"	very flat, easy.
Long	"...wood land and meadow…" "... which runs through a meadow""...parts are easy…"	flat, easy.
Cliff	"…rocky outcrops…""...There are a couple of steep billy climbs…""This trail is extremely difficult,..."	rocky, difficult.

Figure 3-7: Ground truth 2: real user comments for the target hiking trails

Marshall St, Syracuse, NY 13210). The field tests were conducted during 11:00AM-2:00PM Nov. 15, 2013. In each test, there were 12 participating smartphones, which are all Google's Nexus4 smartphones.

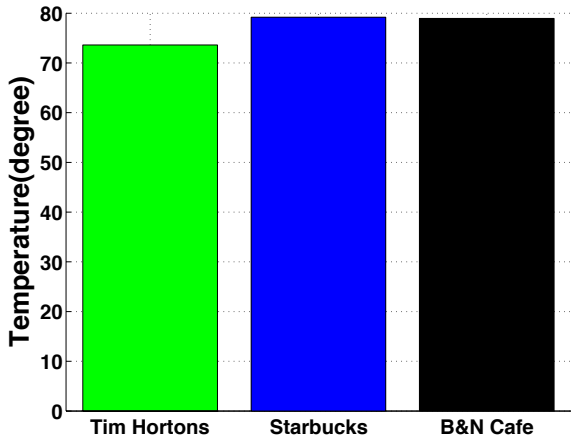
For coffee shops, we collected data of 4 sensing features that customers usually care about most: 1) temperature (temperature sensor on the Sensordrone), 2) brightness (light sensor on the Sensordrone), 3) WiFi signal strength (WiFi interface), and 4) background noise level (microphone). For all these four features, we took averages of all corresponding sensor readings. The feature data are presented in Figure 3-8.

Similarly, we came up with two virtual customers, namely, *David* and *Emma*, whose preferences are described using customer profiles (with preferred values and weights) shown in Fig. 3-9. We then present the rankings of the three target coffee shops computed by SOR for both customers in Table 3.3.

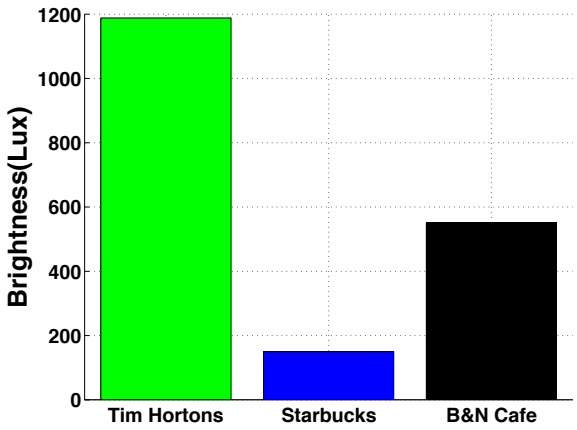
Table 3.3: Rankings of coffee shops computed by SOR

User	No. 1	No. 2	No. 3
David	Starbucks	B&N Cafe	Tim Hortons
Emma	B&N Cafe	Tim Hortons	Starbucks

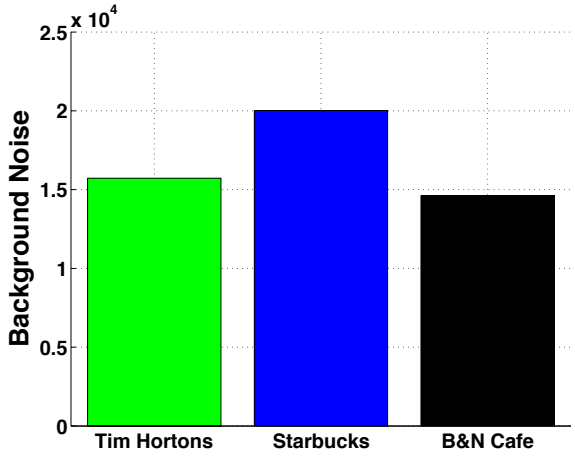
Again, we compare the ranking results with *ground truths*, which are pictures



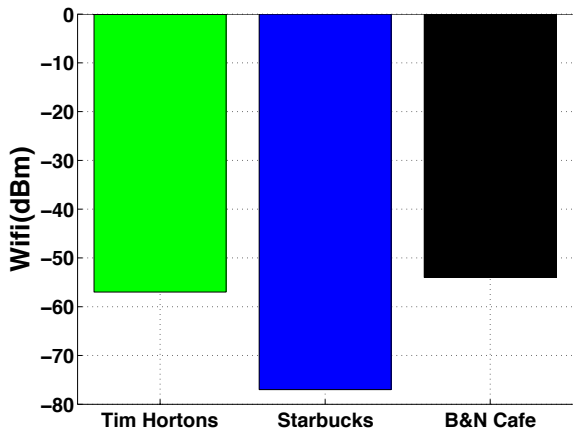
(a) Temperature



(b) Brightness



(c) Background noise



(d) Wifi

Figure 3-8: Feature data for coffee shops

David

An social person who likes to hangout with friends in coffee shops

Feature	Pre. Val.	Weight
Temp.	80	4
Brig.	Min	5
Soun.	Min	0
Wifi	Max	3

(a) David's profile

Emma

An student who likes to study and read in coffee shops

Feature	Pre. Val.	Weight
Temp.	80	3
Brig.	800	5
Soun.	Min	5
Wifi	Max	3

(b) Emma's profile

Figure 3-9: Customer profiles

taken during field tests and real user comments collected from Internet via Google (mainly from Yelp and Foursquare), shown in Fig. 3-10 and Fig. 3-11 respectively.

Figure 3-10: Ground truth 1: pictures of the target coffee shops

	Real Users' Comments	Key opinions
Tim Hortons	"…it is not as noisy as I expected. ..not many people…" "I felt a little bit cold when I was in the store…" "…large windows…It feels very bright…"	quiet, cold, bright.
Starbucks	"…it was a very loud theater…" "…This location is dark and kind of gloomy…" "… Always crowded, never a place to sit, long lines…" " … way too small… considering the noise level and proximity of the tables to each other…"	crowded, loud, dark.
B&N Cafe	"The in-store Starbucks has plenty of tables and is fairly quiet." "…Only place to study…" "…It is very quiet… It is the best public place for studying or reading in Syr…"	quiet, good for study.

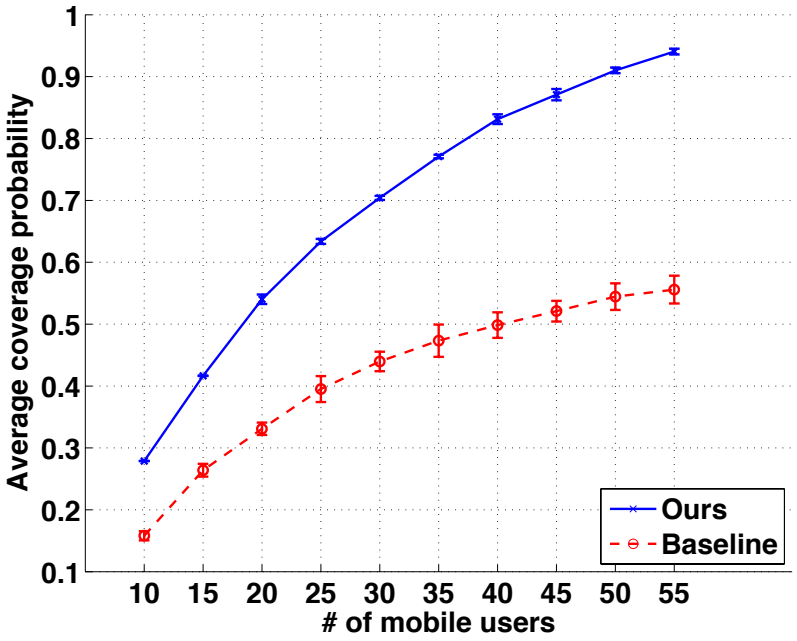
Figure 3-11: Ground truth 2: real user comments for the target coffee shops

From the ground truths, we can see that the Starbucks is crowded, noisy and dark. While the other two coffee shops are quiet and bright. The Tim Hortons is a little colder than the B&N Cafe, however, very bright due to a big window (See Fig. 3-10). David is a social person who likes to hang out with friends in coffee shops so he prefers a not-so-bright and warm place but does not really care about noise. According to the ranking produced for him, the Starbucks is ranked No. 1, followed by the B&N Cafe (since it is not as bright as the Tim Hortons). For Emma (a student who likes to read and study in relatively warm coffee shops), the B&N is recommended as the top choice, followed by the Tim Hortons which is a little colder than B&N Cafe. In the coffee shop case, we can make the same conclusion as the hiking trail case.

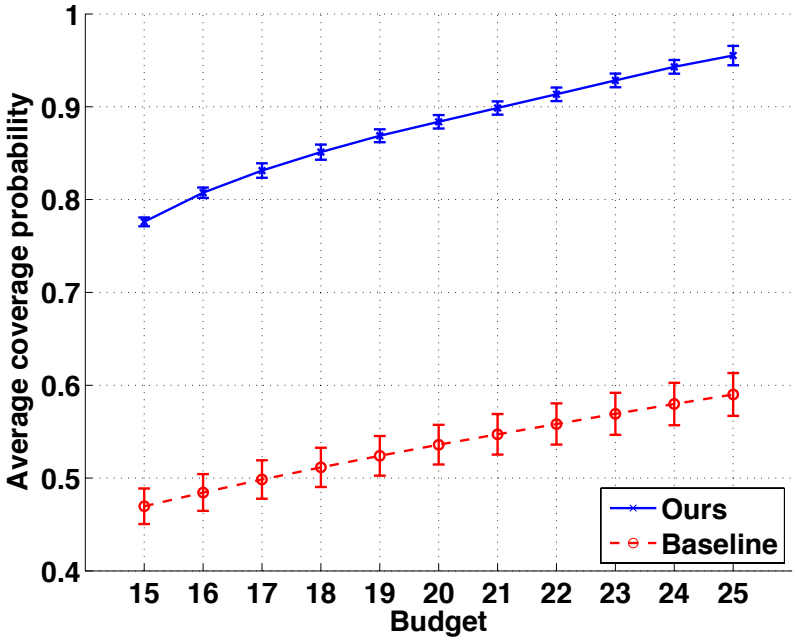
3.5.3 Simulation For The Sensing Scheduling Algorithm

We evaluated performance of the proposed sensing scheduling algorithm in large cases via simulation. In the simulation, the duration of sensing scheduling period was set to 3 hours, which is divided by 1080 time instants. The arrival (leaving) times of mobile users were randomly generated, following a uniform distribution between 0 (the corresponding arrival time) and 10800s. We used a bell-shaped Gaussian distribution (with $\mu = 0$ and $\sigma = 10$s) to model coverage, as discussed in section 3.3. A simple scheduling algorithm served as the baseline: a smartphone starts to sense every 10s since its arrival for N_k^B times, where N_k^B is the corresponding budget. The average coverage probability was used as performance metric, which is the sum of coverage probabilities (objective function) divided by the total number of time instants in the scheduling period (i.e., 1080). In the first simulation scenario, we changed the number of mobile users from 10 to 50 with a step size of 5 and the budgets of all mobile users were fixed to 17. In the second scenario, we changed the budget from 15 to 25 with a step size of 1 and the numbers of mobile users were fixed to 40. We presented the results in Fig. 3-12. Note that every number in the figure is an average over 10 runs.

From the figure, we can see that on average, our scheduling algorithm outperforms the baseline algorithm by 65% in terms of average coverage probability. From Fig. 3-12(a), we can see that when 55 users participate in sensing, our algorithm leads to almost 100% coverage. In order to achieve an average coverage probability of 80%, our scheduling algorithm need no more than 40 users (with a budget of 17) while the baseline algorithm can only reach an average coverage probability of 50% with 40 users. Similar observations can be made from Fig. 3-12(b). No matter which method is used, the average coverage probability always increases with the number of mobile users and budget as expected. In addition, we observe that the variance of the coverage probability given by our scheduling algorithm is always less than that given by the baseline algorithm, which means our algorithm is more stable and is suitable



(a) Varying # of mobile users



(b) Varying budget

Figure 3-12: Performance of the sensing scheduling algorithm for various situations.

Chapter 4

Application II: *LI*festyle Learning Via *P*hone *S*ensing (*LIPS*)

4.1 Overview

As analyzed in the previous chapters, the sensors of a smartphone can easily detect the context (such as location, local weather, activities, etc.) of its mobile user. However, how to leverage this unique capability for learning lifestyles of mobile users and making their life better has not yet been fully exploited. One of the early efforts in this regard is that some urban guide mobile applications (such as Yelp [77] and Urbanspoon [70]) have used locations (collected by smartphones) to recommend local businesses (such as restaurants, bars, local applications, etc.) to mobile users. We believe that with smartphones and their sensors, we can do much better and do much more than this.

Some related works are focusing on analyzing location data collected by GPS. In the project TraClass [37], Lee etc. studied how to predict moving objects' types by analyzing their trajectories and other features. They proposed to generate a hierarchy of features by partitioning trajectories and exploring different types of clustering.

In [22], the authors proposed a sequential pattern mining paradigm that can be used to analyze the trajectories of moving objects. In [76], the authors proposed a data mining framework to retrieve association patterns from raw individual GPS data. Note that all these works leveraged only location and time information for analysis without using other features (such as moving states, weather, etc.) that can be collected via smartphone sensing.

In this chapter, we present the design, implementation and evaluation of a sensing application, LIPS, which can learn LIfestyles of mobile users via smartPhone Sensing (LIPS). According to businessdictionary.com, *"Lifestyle is expressed in both work and leisure behavior patterns and (on an individual basis) in activities, attitudes, interests, opinions, values, and allocation of income."* By leverage multiple sensors on a smartphone, we can obtain a comprehensive view of a mobile user's context (such as location, local weather, activities, etc.) over a long period. Based on those context information, we aim to find out what a mobile user likes to do (characterization) and what he/she will do next (prediction) based on the collected sensor data. Such a lifestyle learning application can be used to support a large variety of applications for improving life quality. For example, a major application is to recommend local businesses to mobile users based on not only his/her location but also his/her lifestyle. This work represents one of the first efforts along this line, which is focused on lifestyle learning, while leaving lifestyle-aware recommendation or lifestyle-based applications for future research.

LIPS is built on top of the platform. Both mobile frontend and cloud backend need to be extended with new application modules. The LIPS frontend application module on the mobile frontend reports the context information collected by sensors of the smartphone to the learning application module on the backend server periodically. Based on this information, the learning application module builds models for lifestyles of mobile users.

Combining both unsupervised and supervised learning, we propose a hybrid scheme for lifestyle learning, which consists of two parts: characterization and prediction. Specifically, we present a two-stage algorithm to characterize the lifestyle of a mobile user using Places of Interest (PoIs), which leverages two different algorithms for coarse-grained and fine-grained clustering in two stages respectively. Based on discovered PoIs, we present a supervised learning based algorithm to build a model for predicting the future activities of a mobile user.

In addition, operating smartphone sensors (such as GPS) could be energy consuming. Note that even though some sensors (such as accelerometer) are always active, a thread needs to be spawned to collect its readings, which will prevent the smartphone enter sleep mode and affect the energy consumption too. A phone's main job is not sensing after all. Extensive smartphone sensing may drain its battery quickly, leaving it dead when it is needed to perform its regular duties such as phone calls, web surfing, etc. To enable green lifestyle learning, we present an adaptive sampling algorithm, which adaptively control the sampling rate according to discovered PoIs and the lifestyle model. To the best of our knowledge, we are the first to build a smartphone sensing based system to learn and analyze lifestyles of mobile users based on various context information (collected from smartphones).

We build a novel smartphone sensing based application for lifestyle learning, and propose practical and effective solutions to fundamental problems (learning and energy-efficient sampling). Specifically, we summarize our contributions in the following:

- We present design and implementation of LIPS application, which learns lifestyles of mobile users via smartphone sensing.
- We present an effective hybrid scheme for lifestyle learning, which combines both unsupervised and supervised learning.

- We present an energy-efficient sampling algorithm, which leverages the discovered PoIs and the lifestyle model for adaptively controlling the sample rate.
- We performed extensive field tests to validate and evaluate LIPS. The experimental results well justify the effectiveness and efficiency of LIPS on lifestyle learning.

The rest of the chapter is organized as follows: We present the software architecture and implementation details of the proposed LIPS application in Section 4.2. The proposed learning scheme and adaptive sampling algorithm are presented in Section 4.3 and Section 4.4 respectively. Experimental results are presented and analyzed in Section 4.5.

4.2 Design and implementation of LIPS

In this section, we present the design and implementation of LIPS application.

Similar to SOR application, LIPS also consists of two parts: mobile frontend application modules and cloud backend application modules.

The mobile frontend applications modules is developed and integrated into the unified platform, that runs on each mobile user's smartphone. The cloud backend modules is integrated into backend server,which runs in the cloud. We can simply deploy multiple learning servers and load balancers if we need to serve a large number of mobile users from different locations.

In our design, backend application modules are composed of online modules and offline modules. The online modules are designed to be light-weighted, which provides immediate online responses to the mobile frontend, including notification, user request handling and updating, etc. However, lifestyle learning involves compute-intensive and time-consuming workload, which includes running the clustering-based

characterization algorithm (Section 4.3.1) and the supervised learning based prediction algorithm (Section 4.3.2) on collected sensor data. In our design, a set of the offline modules are designed and deployed to handle this compute-expensive workload in an offline manner on powerful servers. This design ensures that online requests from the mobile frontend are not delayed by the time-consuming learning process.

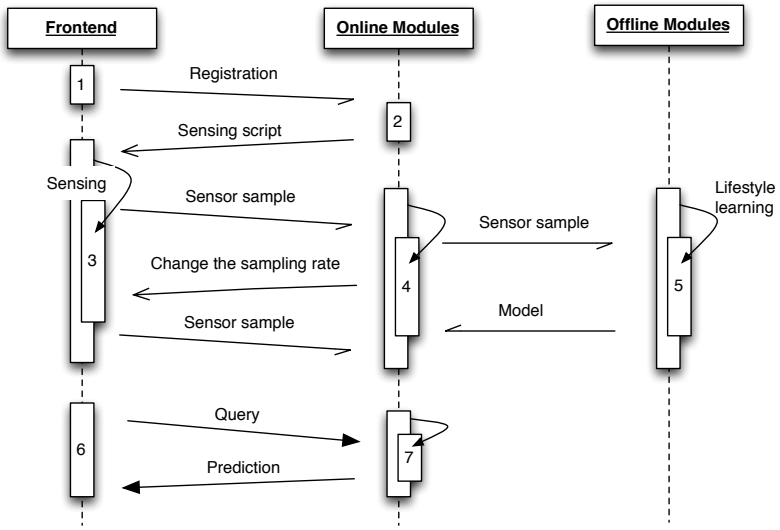


Figure 4-1: The workflow of LIPS

We illustrate how the mobile frontend modules, the online modules and the offline modules interact with each other and how LIPS works in Figure 4-1, which are further described as follows:

1) A mobile user registers for the lifestyle learning application by sending his/her personal information along with his/her preferences (e.g. only allowing coarse locations rather than fine locations) to the online modules.

2) The online modules accepts (or rejects) the registration request and sends sensing scripts according to user preferences to the mobile frontend.

3) The mobile frontend collects the mobile user's context information using smartphone sensors periodically and sends sensor data to the online modules, which then stores them into a database. The mobile frontend adaptively adjusts its sam-

pling rate to trade off energy efficiency and learning performance (will give detailed introduction in Section 4.4).

4) The offline modules periodically pulls sensor samples from the database.

5) The offline modules discovers PoIs and builds the lifestyle model for activity prediction on a daily basis according to received sensor samples. And, it updates the online modules with PoIs and lifestyle model.

6) The mobile frontend periodically sends a query with the current context information (Section 4.3) to the online modules. This query can also be sent in an ad-hoc manner.

7) The online modules replies with the prediction results (Section 4.3.2).

In order to support LIPS application, two types of modules need to be added to the mobile frontend of platform: LIPS application module and providers. Add-on modules are highlighted with orange color in fig. 4-2. The functionality of other modules (in green color) have been introduced in details in Section. 2.3.

Similar to the *SOR application* module, *LIPS application* is independent of other sensing application modules and needs to be registered in *applications Register*. In *LIPS application* module, the local *Preferences Manager* enables the platform user to setup his/her own preferences. For example, a user could indicate he/she doesn't want to use his/her limited and expensive data plan for transmitting/receiving sensor data for lifestyle learning. If so, sensor data will be uploaded to the learning server only whenever WiFi connection is available. Another example is: the user might indicate a certain time period that he/she isn't willing to provide data to the learning server due to privacy concerns.

The *Sampling application Manager* manages the data collection procedure, which has four functionalities: 1) updating and storing sensing scripts; 2) triggering sampling task instances to collect data from sensors periodically; 3) handling failures

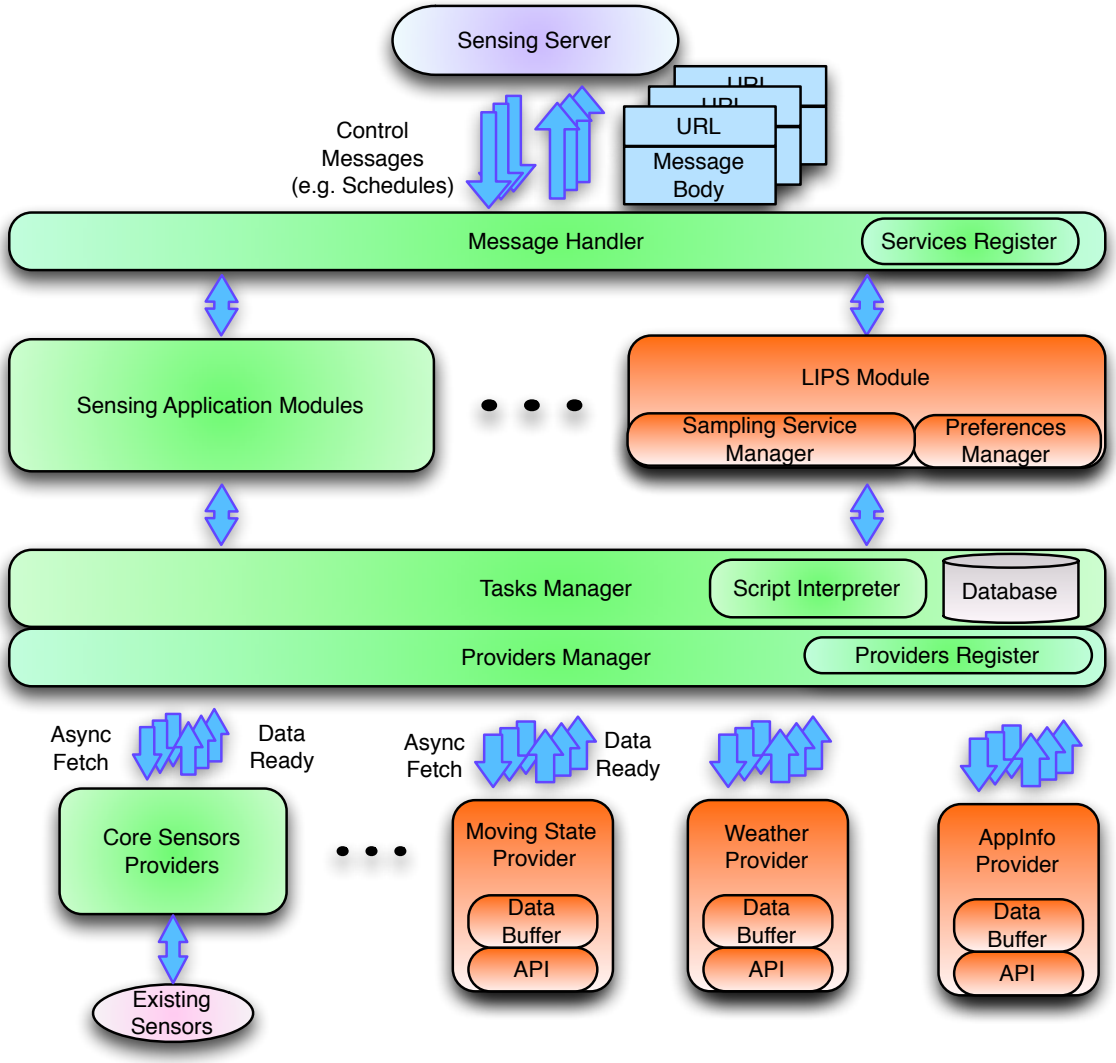


Figure 4-2: The architecture of the mobile frontend

(such as rebooting a failed sampling task instance); 4) Encapsulating sensor data and notifying the Message Handler to upload them to the learning server. Note that data uploading occurs only when network connection is available. Otherwise, The Sampling Task Manager will store results into the local database.

There are two ways to run periodic tasks on Android, 1) Using the `Handler` class to set a `Runnable` task. The handler will execute the `Runnable` task after a given delay. 2) Setting a system level alarm. The alarm will be triggered at a certain time and broadcast a message. The alarm listener (that has been registered to this alarm

beforehand) invokes a `Runnable` task when the message is received. Since the former method cannot wake up the Android system from sleep, the scheduled sampling tasks may not be executed on time when the Android system is sleeping, which will lead to inaccuracy on lifestyle learning. In LIPS, we employ the second method, i.e., the system level alarm, to implement periodic sampling. And *Sampling application Manager* manages the LIPS application related alarms. With the sampling alarm triggered, the Sampling Task Manager will create a sampling task instance, which contains a time-stamp, a sensing script, and a data buffer. A sampling task instance sends the corresponding Lua scripts to the Script Interpreter for translation. The interpreter can interpret both Lua's own functions and the functions we defined for data acquisition, as we have introduced in Section 2.3.

The add-one providers of the platform are: *Moving State Provider*, *Weather Provider* and *AppInfo Provider*. A Provider is basically a software component which actually operates a "sensor" using APIs provided by the Android system or a third party to collect data. Please note that, "sensor" here has a much broader meaning, which refers to data source that can provide context information of a mobile user. Therefore a sensor could be: 1) an embedded sensor (such as GPS, accelerometer, digital compass, etc.) on a smartphone; 2) a application that can provide context information (such as local weather) to mobile users via APIs; or 3) an external sensor (such as Fitbit [19], SensorDrone [63], etc.) that can be connected to a smartphone via its network interface (such as Bluetooth).

The *Weather Provider* uses REST APIs to communicate Weather.com with remote servers and get the weather information for current location. The *Moving State Provider* wraps the APIs provided by Google Play application. It monitors user's Moving State On-foot, Driving, Bicycling, Still, Unknown, manages the information collection life cycle Initiated, Data Ready, Finished, Timeout and shares the results. *AppInfo Provider* use Android system APIs collect system related information. The

Data acquisition is done asynchronously so that an operation will not block or be blocked by others. When a sampling task instance requests data, the Sensor Manager directs the call to the corresponding Provider to actually acquire data from sensors. Moreover, the Sensor Manager can cancel data acquisition if timeout.

The backend of LIPS application is built on top of the platform's cloud backend, as illustrated in fig. 4-3. The add-on modules are divided into two types: online modules and offline modules. The online modules are: *LIPS Sensing Script Manager*, *LIPS Sensing Controller* and *LIPS Sensing Script Manager*. The offline modules are *PoI Discover*, *Lifestyle Modeler* and *Place Information Provider*. Online and offline modules are connected by a database for data exchange.

First, we introduce the online modules.

In order to learn lifestyle, the mobile frontend needs to collect readings from multiple sensors periodically. Lua [44] is used to specify what sensor data to acquire and how frequent to sample. The *LIPS Sensing Script Manager* generates a sensing script when accepting a registration request from a mobile user, and sends it to the mobile frontend for execution. The Sensing Script Manager can change sensing activities of the mobile frontend by updating its sensing script. For example, the sampling period of the mobile frontend is set to 5min by default, the Sensing Script Manager can slow down sampling by sending a new script specifying a longer sampling period (e.g., 10min) to the mobile frontend. We will discuss how to adaptively adjust the sampling rate at runtime for energy saving in Section 4.4. By changing the sensing scripts, the server is able to modify the sensing types (e.g. location, temperature, weather condition) and sensing frequency for each user at runtime. If sensing scripts are changed for a user, Sensing Script Manager will send that specific user a notification to ask the user to connect to the online application and update the changed sensing scripts.

The *LIPS Sensing Controller* performs the following duties: 1) validating collected

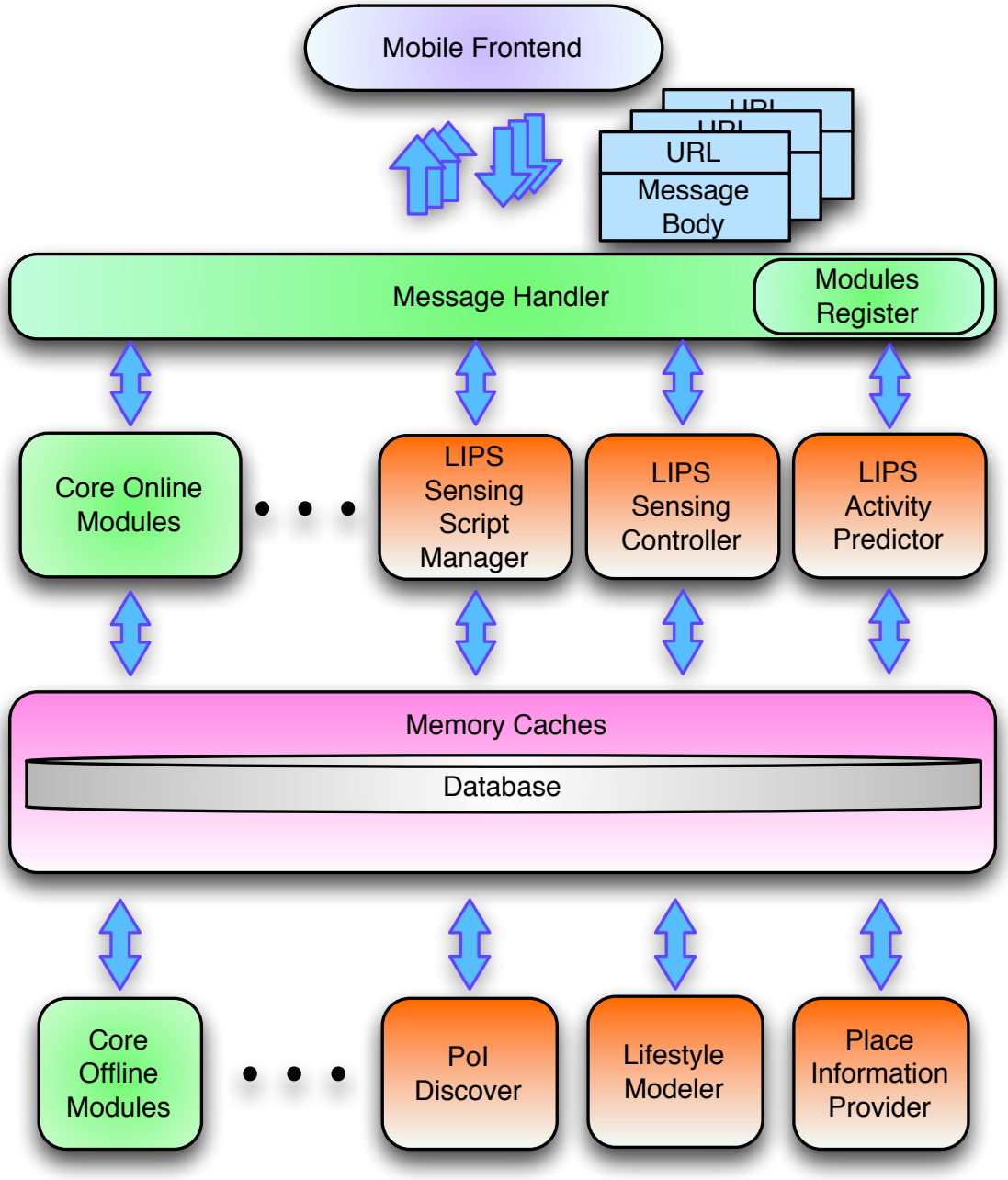


Figure 4-3: The architecture of the learning server

sensor data: if the data don't comply with the corresponding sensing script (i.e. a required feature, such as location, is missing), they will be discarded; 2) notifying the Data Pre-processor when raw sensor data are ready; Updating users' sampling profile (e.g. when the sampling data is reported, how many samplings the user has provided). 3) adjusting the sampling rate of the mobile frontend adaptively (Sec-

tion 4.4); 4) collecting some statistics about collected sensor samples, e.g., counting how many valid samples have been received. 4) Profile information will be explored to determine when machine learning should be performed and when sampling rate should be adjusted.

The *LIPS User Activity Predictor* handles the prediction query from the mobile front end by predicting the user's future activities based on his/her current states and the lifestyle model learned from historical data (Section 4.3.2).

Next, we introduce the offline modules. The *PoI Discover* uses the output of *Data Pre-processor*(introduced in Section 2.4) to analyze each user's feature data and discover his/her Places of Interest (PoIs), which will be described in details in Section 4.3.1). The *Lifestyle Modeler* builds a model for predicting future activities of a mobile user based on discovered PoIs using a supervised learning algorithm, which will be introduced in details in Section 4.3.2). The *Place Information Provider* is used to retrieve the actual place (such as restaurant, coffee shop, etc.) information given the location of a PoI, which will be used for building the lifestyle model. In our system, we use Google's Place API [25] to obtain such information. The other location/map applications can also be used to serve this purpose, such as Bing Maps [5], HERE Maps and MapQuest [47].

4.3 Lifestyle Learning

In this section, we will discuss how to learn the lifestyle of a mobile user in details.

From the definition of *lifestyle* presented above, we know that lifestyle reflects an individual's preferences, and it can be expressed in both work and leisure activity patterns.

For example, the lifestyle of a mobile user, say Alice, could be given as a set of activities: 1) Alice normally goes to work from home at around 9:00AM every

weekday; 2) Alice has her lunch in the restaurants near her workplace at around 12:30PM every weekday; 3) Alice usually leaves work and go home at around 6:00PM; 4) Alice usually does grocery shopping on Saturday; 5) Alice goes to a movie theater every Saturday; 6) Alice goes shopping in a local shopping mall every Saturday; and 7) Alice goes to library every Sunday afternoon.

As described above, the goal of lifestyle learning is to find out what a mobile use likes to do (characterization) and what he/she will do next (prediction).

Mobile phones are usually carried by their users almost all the time, which make them a perfect device for providing useful context information to learn the lifestyle of mobile users. In LIPS, *sensor samples* are collected periodically by the mobile frontend for lifestyle learning. A sensor sample $\mathbf{s}$ is defined by a 3-tuple $(t, l, \mathbf{D})$, where t is the timestamp, l is the location, and $\mathbf{D}$ is a set of raw sensor readings that are used to produce feature data (described below).

In LIPS, a list of features are extracted from a sensor sample, which are then used as input for discovering of PoIs of a mobile user (Section 4.3.1) and for predicting his/her activities (Section 4.3.2):

1) *Day and Time:* the day (Monday, Tuesday, etc.) and the time at the sampling instant. Note that both features are very important since the period of many people's lifestyles is one week and usually their activities in a day are highly time-dependent.

2) *Location and Speed:* the location and the moving speed at the sampling instant, which are obtained via either GPS (fine) or Google's Location applications (coarse) according to user's preferences.

3) *Moving State*: {On-foot, Driving, Bicycling, Still, Unknown}, which can be obtained by calling the activitity recognition API [1] in the Google Play applications.

4) *Step Frequency*: the numbers of steps per second (if on-foot). Each sensor sample

includes 10 continuous accelerometer readings, which are then used to estimate the step frequency using the method introduced in [10].

5) *Weather Condition*: {Sunny, Cloudy, Raining, Snowing}, which can be obtained by calling the REST Weather Channel API [71]. (e.g. NOAA Weather application API, Weather Channel API and WeatherBug API).

6) *Local Temperature:* the outdoor temperature at the sampling instant and location, which can be obtained by calling the REST Weather Channel API [71] too.

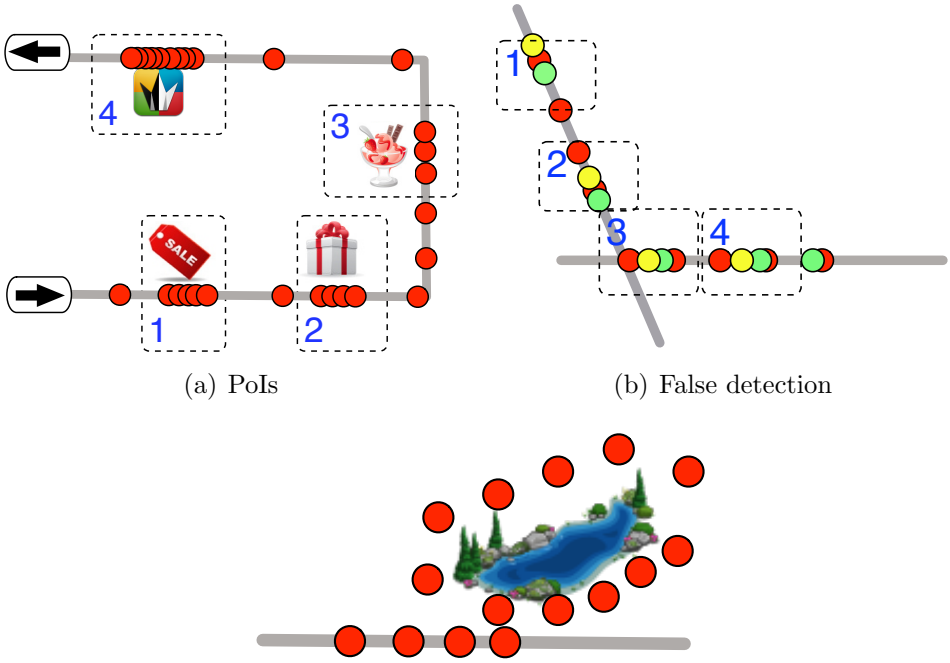
7) *User State*: {Active, Inactive}, which shows whether or not the user is actively using the smartphone. This can be obtained by using the Android system API to check if any app is launched in the past sampling period.

We select these features to build the lifestyle model because we believe they may all have a significant impact on a mobile user's activities. For example, a mobile user usually goes to a restaurant on Saturday night, however, if the weather happens to be pretty bad (e.g., snowing), he/she may decide not to go out.

4.3.1 Lifestyle Characterization With Places Of Interests

In order to learn lifestyle of a mobile user, we first need to know which places he/she likes to go, which, however, is hard to tell simply based on a set of collected sensor samples since some of them may be taken when he/she moves from one place to another. We characterize the lifestyle of a user using PoIs. A PoI is a place that a mobile user has visited, which could be a grocery store, a shopping mall, a restaurant, etc. Discovering PoIs for a mobile user is the first step of lifestyle learning.

In an example illustrated by Fig. 4-4(a), a mobile user entered a commercial district. He first spent some time in a shop with goods on sale. Then he walked into a gift shop. A popular ice cream shop caught his attention on his way to the movie theater. He bought an ice cream, which caused him to be late for his movie. So he



(a) PoIs (b) False detection

(c) Miss detection

Figure 4-4: Problems associated with the PoI discovery

ran to the theater from ice cream shop. Each red dot in this figure is the location of the user at a sampling instant. If the user stays at the same place for more than one sampling period, we will have more than one red dots in that location. The user's moving speed determines spacial sparsity of samples: generally the faster the user moves, the more sparse the samples we will have in the spacial domain. To analyze the lifestyle of a user, we need to discover his/her PoIs. In this example, PoIs are marked in the figure: 1) A (shop with goods on sale); 2) B (gift shop); 3) C (ice cream shop); and 4) D (movie theater).

From the example above and the field-test, we can see that sensor samples have the following two properties: 1) The set of collected sensor samples contains both samples related to PoIs, and samples corresponding to movements between PoIs, which may not be relevant. 2) The number of PoIs are not known beforehand.

Intuitively, we should apply a clustering algorithm to find clusters based on collected samples, which can then be used to identify desired PoIs. However, we find

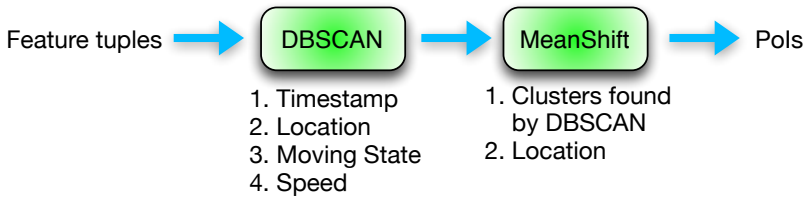


Figure 4-5: The PoI discovery algorithm

most existing clustering algorithms are not suitable for our problem due to the following reasons: 1) Those clustering algorithms, in which the number of clusters is required as input, are not suitable, since the number of PoIs are not known beforehand. 2) A naive approach, in which clusters are determined simply based on the amount of time a user stays in an area, is not applicable since a user may stay on certain part of a road for a long time due to traffic jam. 3) Clustering simply based on locations without taking time into consideration, may lead to many false PoIs. As shown in Fig. 4-4(b), samples collected over several days overlap (yellow, green and red dots mark samples collected in different days). In such a scenario, areas A, B, C and D are likely to be discovered incorrectly as PoIs. 4) It is not reasonable to determine whether a place is a PoI or not simply based on a moving speed threshold.In Fig. 4-4(c), the user jogs around a lake with samples evenly distributed around it. By just setting up a fixed speed threshold for clustering, the lake may not be discovered as a PoI.

In summary, clustering should be done according to multiple relevant features rather than a single feature.

Based on our observations, we design a two-stage algorithm to discover PoIs from a set of sensor samples, which is illustrated in Fig. 4-5. As described above, instead of directly using raw sensor data, we extract useful information from collected sensor samples to produce feature data as input. We choose the DBSCAN [18] and MeanShift [8] algorithms for coarse-grained and fine-grained clustering in the first and second stage respectively.

In the first stage, our main goal is to filter out those samples related to movements between two PoIs rather than actual PoIs. For each collected sample $\mathbf{s} \in \mathbf{S}$, we extract its *timestamp, location, moving state and moving speed* to build a feature tuple $\mathbf{f} = (t, l, a, v) \in \mathbf{F}$. We feed those feature tuples into the DBSCAN algorithm to produce a set of clusters. DBSCAN is a density-based clustering algorithm, which ensures that the output clusters are areas of high density while outside of the clusters are areas of low density [18]. This is desirable for our problem since DBSCAN can efficiently filter out those sparsely distributed samples related to movements between two PoIs. Moreover, we perform clustering based on multiple features, which avoids the potential issues related to single feature based clustering described above.

Specifically, we first perform DBSCAN based on the feature tuples to discover a set $\mathbf{C}$ of clusters $\mathbf{C} = \{\mathbf{C}_1, \cdots, \mathbf{C}_N\}$ from $\mathbf{F}$. Then for each cluster $\mathbf{C}_j \in \mathbf{C}$, apply DBSCAN again to further divide this cluster into a set of sub-clusters $\mathbf{C}_j = \{\mathbf{C}_{j,1}, \cdots, \mathbf{C}_{j,M}\}$. Note that for most cases, one round of DBSCAN is sufficient, i.e., the second round of DBSCAN will not be able to divide each $\mathbf{C}_j \in \mathbf{C}$ further into multiple smaller clusters. However, the second round of DBSCAN is necessary in some cases. For example, if a mobile user visits places in two different cities, only two clusters (each corresponds to a city) will be returned after the first round because DBSCAN does clustering based on the density of samples. This is obviously too coarse so clustering needs to be done again to improve granularity.

After clustering using the DBSCAN algorithm, we can have a set of clusters of feature tuples. However, since DBSCAN is not a centroid-based clustering algorithm, it does not return cluster centers, which are needed by us. In addition, we find that if multiple PoIs are close to each other (e.g., multiple PoIs in a plaza), the related feature tuples may be put into the same cluster. So we still need to do fine-grained clustering following the first stage. In the second stage, we use the MeanShift algorithm on each cluster found in the first stage. In this stage, only locations are used for clustering.

The MeanShift algorithm uses a similar idea for clustering but can return cluster centers (if each cluster has a convex shape) [8]. These cluster centers will then be returned as the set of PoIs.

We formally present our PoI discovery algorithm as Algorithm 3, in which the first stage starts from Step 3, and the second stage starts from Step 8. We use c'_i and r'_i to denote the center and the corresponding radius of cluster $\mathbf{C}'_i$ respectively.

Algorithm 3 The PoI Discovery Algorithm

Input: The set of feature tuples $\mathbf{F}$;
Output: The set of PoIs $\mathbf{P}$;

1: $\mathbf{P} \leftarrow \emptyset$;
2: $\mathbf{C} \leftarrow \emptyset$;
3: $\{\mathbf{C}_1, \cdots, \mathbf{C}_N\} \leftarrow \texttt{DBSCAN}(\mathbf{F})$;
4: **for** $\mathbf{C}_j \in \{\mathbf{C}_1, \cdots, \mathbf{C}_N\}$ **do**
5: $\quad \{\mathbf{C}_{j,1}, \cdots, \mathbf{C}_{j,M}\} \leftarrow \texttt{DBSCAN}(\mathbf{C}_j)$;
6: $\quad \mathbf{C} \leftarrow \mathbf{C} \bigcup \{\mathbf{C}_{j,1}, \cdots, \mathbf{C}_{j,M}\}$;
7: **end for**
8: **for** $\mathbf{C}_{i,j} \in \mathbf{C}$ **do**
9: $\quad \mathbf{F}' \leftarrow \emptyset$;
10: $\quad$ **for** $\mathbf{f} = (t, l, a, v) \in \mathbf{C}_{i,j}$ **do**
11: $\quad\quad \mathbf{F}' \leftarrow \mathbf{F}' \bigcup \{l\}$;
12: $\quad$ **end for**
13: $\quad \{\mathbf{C}'_1, \cdots, \mathbf{C}'_n\} \leftarrow \texttt{MeanShift}(\mathbf{F}')$;
14: $\quad \mathbf{P} \leftarrow \mathbf{P} \bigcup \{(c'_1, r'_1), \cdots, (c'_n, r'_n)\}$;
15: **end for**
16: **return** $\mathbf{P}$

The output of this algorithm is a set $\mathbf{P}$ of PoIs (with center locations and radii). Obviously, these locations cannot be directly used to predict activities of the mobile users. In LIPS, the Place Information Provider uses the Google Place API [24] to find the actual places according to these locations.

4.3.2 Lifestyle Modeling For Activity Prediction

In this section, we describe how to predict a mobile user's activities in the next T hours according to the discovered PoIs. Note that PoIs tell us exactly which places the

mobile user visited. For a mobile user, PoIs $\mathbf{p}$ and $\mathbf{p}'$ may be two different restaurants he/she usually goes to, but they both correspond to the same activity "dining". So we are actually interested in knowing what kind of activities a mobile user will do in the near future (rather than exactly which place he/she will visit) such that we can provide related and useful information (such as recommendation) to him/her.

In LIPS, we define a set of activities: mall shopping, dining, grocery shopping, outdoor recreation, indoor recreation, movie, gas station, car wash, exercise, laundry, library, and schooling. This set can certainly be expanded according to the new PoIs and needs. In addition, we need to map PoIs to activities. We again use Google Place API to find the type of each PoI. For example, given a hiking trail, it will return its type as "park". In LIPS, We then create a table to map each type to certain activity. For example, if the type of a PoI is "park", its corresponding activity is "outdoor recreation".

In order to build a lifestyle model for future activity prediction. We need to have a training set, in which each item is a *feature-activity* tuple $(\mathbf{f}; \pi)$. $\mathbf{f} = (d, t, l, a, v, w, c, u)$, where d is the day (Monday, Tuesday, etc.), t is the time, l is the location, a is the moving state, v is the moving speed, w is the weather condition, c is the outdoor temperature, u is the user state (active or not); and π is the associated activity. Note that here the feature tuple includes all features discussed in the beginning of this section, which is different from that introduced in the previous section. In addition, we aim to predict the activities in the next T_s to T_e hours so when building the training set, the activities need to be the activities discovered in that future period. For example, suppose a feature tuple of a mobile user, Alice, at 11:00AM is $\mathbf{f}$, and we want to predict her activities in the next 1 to 2 hours (i.e., between 12:00PM and 1:00PM), and her activity during that period turned out be "dining" according to some collected samples, then we will add a feature-activity tuple ($\mathbf{f}$, "dining") into the training set. Of course, if there were more than one activity, say "dining" and "indoor recreation"

during that period, we will add both ($\mathbf{f}$, "dining") and ($\mathbf{f}$, "indoor-recreation") into the training set.

After the training set is built, any supervised classification algorithm [27] can be used here to make prediction. We tested a few of them extensively with the collected sensor data and found that Support Vector Machines (SVM) [27] turns out to be the most effective one. Moreover, it is known that SVM is usually very effective in the high-dimensional spaces (many features), fast and memory-efficient. So in LIPS, we employ SVM to predict future activities of a mobile user. SVM can return a model such that when given a feature tuple (as shown above) of a mobile user, it can return the probability of each possible activity he/she may perform in the future.

4.4 Lifestyle-aware adaptive sampling

Energy resource of a smartphone is very limited. Therefore, it is crucial to carefully manage its energy usages on sensing; otherwise, LIPS may drain its battery quickly. A simple and practical approach for saving energy is to adaptively adjust the sampling rate.

On one hand, if the sampling rate is reduced (i.e., sampling period is increased), energy spent for sensor data acquisition and communications can certainly be reduced. Moreover, the smartphone system will have a much higher chance to enter the sleep mode, which is known to consume much less energy than the active mode does. On the other hand, reducing the sampling rate may lead to less samples, which will have a negative impact on the performance of PoI discovery and activity prediction. Hence, we need to develop an effective algorithm that adaptively adjusts the sampling rate to trade off energy efficiency and learning performance.

We consider the following three cases when we try to make a decision on whether or not to reduce the sampling rate: 1) If the user is at some place, which is not one

of discovered PoIs, the sampling rate should not be reduced since otherwise there may not be sufficient samples for discovering this possibly new PoI. 2) If the user is at one of discovered PoIs and the activity prediction is *stable* (explained below), the sampling rate can be reduced. 3) If the user is at one of discovered PoIs, but the prediction result is not stable, the sampling rate should not be reduced.

In our adaptive sampling algorithm, we make sure that the sampling period falls in the range of $[T_{\min}, T_{\max}]$. We set the initial sampling period to $T_{\min}$. $T_{\min}$ and $T_{\max}$ were set to 5min and 20min respectively in our implementation. Every time (say at time t) when a new feature tuple is collected, the algorithm checks whether or not its location l_t falls in the radius of any discovered PoI. If so, the algorithm further employs the developed activity prediction model $\mathcal{M}(\cdot)$ (described above) to predict his/her future activities and stores the results to $\mathbf{\Pi}_t$. Then the algorithm compares $\mathbf{\Pi}_t$ with the previous results to see if there is any significant change. If yes, the sampling period is doubled, otherwise it remains the same as before. Here $\pi_t^{\max}$ ($\pi_{t-T}^{\max}$) and $p_t^{\max}$ ($p_{t-T}^{\max}$) denote the most likely activity and the corresponding probability predicted according to the current sample $\mathbf{f}_t$ (the previous sample $\mathbf{f}_{t-T}$), respectively. α is a threshold, which is used to define the condition that triggers adjustment of the sampling period. The larger the α is, the more likely the sampling period will be increased. For all the other cases, the algorithm stays with the minimum sampling period (i.e. 5min in our implementation). We formally present our adaptively sampling algorithm as Algorithm 4. Note that the feature tuple (sample) $\mathbf{f}_{t-T}$ collected at the last sampling instant $t-T$ and the corresponding prediction results $\mathbf{\Pi}_{t-T}$ are given as input.

Note that since the PoI discovery algorithm depends on the density of sensor samples, we need to duplicate sensor samples to maintain the sample density if the sampling rate is reduced by the adaptive sampling algorithm.

Algorithm 4 Lifestyle-aware Adaptive Sampling Algorithm

Input: $\mathbf{f}_t$, $\mathbf{f}_{t-T}$, $\mathbf{\Pi}_{t-T}$, T;
Output: T';

```
1: if f_{t-T} = nil then
2:     return T_min;
3: end if
4: T' ← T_min;
5: if ∃p = (c, r) ∈ P s.t. ||l_t − c|| ≤ r then
6:     Π_{t-T} ← M(f_{t-T});
7:     Π_t ← M(f_t);
8:     if π^max_{t-T} = π^max_t & |p^max_{t-T} − p^max_t| < α * p^max_{t-T} then
9:         if 2 * T ≤ T_max then
10:            T' ← 2 * T;
11:        else
12:            T' ← T;
13:        end if
14:    end if
15: end if
16: return T';
```

4.5 Validation and performance evaluation

In this section, we present the experimental results collected from field tests to validate and evaluate LIPS.

The field tests were conducted in March and April of 2014, with a group of volunteers from 6 major cities in USA.

During the experiments, all the volunteers used Android-based Nexus 4 or Nexus 5 phones.

To protect their identities, we use a single capital letter as their names in the following.

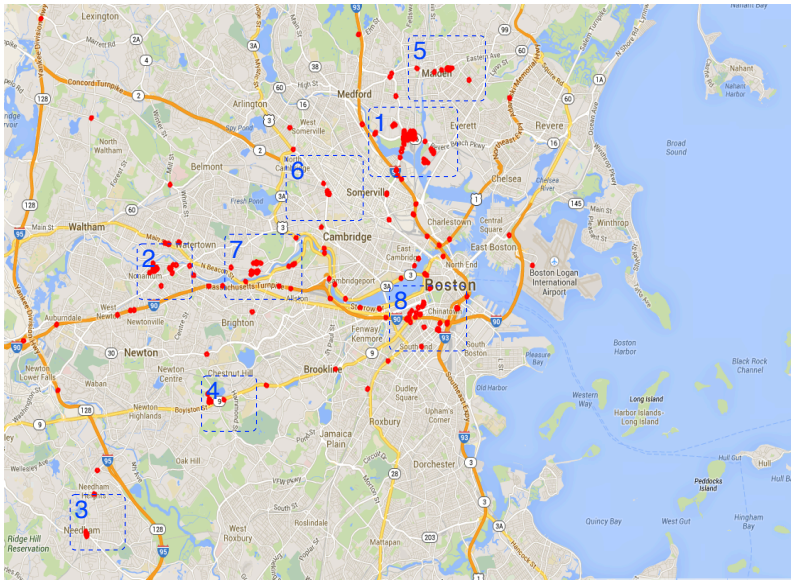
Sensors were first sampled every 5 minutes. In the last three days of experiments, we started to apply the proposed lifestyle-aware adaptive sampling algorithm to adaptively adjust the sampling rate, and in the meanwhile, we still collected sensor samples every 5 minutes for comparisons.

4.5.1 Validation and Evaluation of Lifestyle Learning

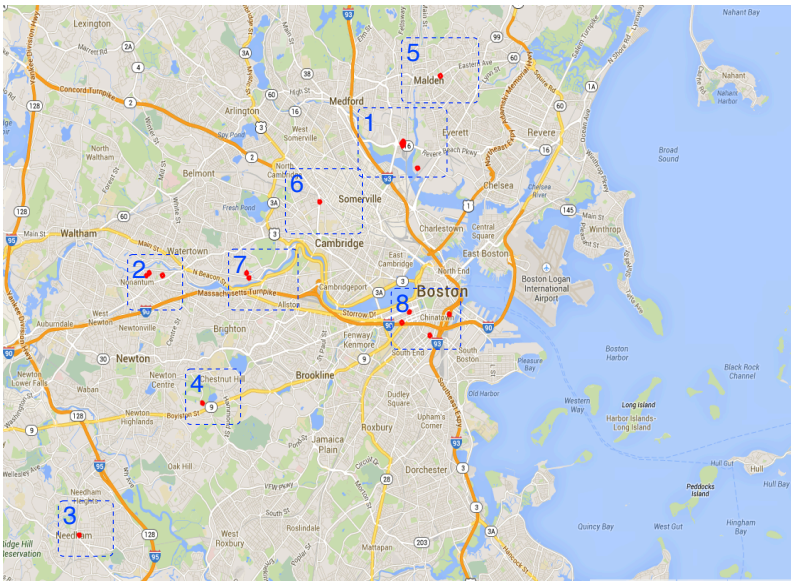
First of all, we present experimental results to validate the proposed PoI discovery algorithm. We used the Google Maps to show the sensor samples and the corresponding PoIs. Due to the space limitation, we chose to present results of two representative volunteers: Ms. A, Mr. B, who are from two different cities. Since many volunteers are friends or relatives of one of the authors, we first briefly describe their lifestyles (according to our best knowledge) as ground truths and then show the PoI discovery results.

Ms. A is a businesswoman living in the Great Boston area. Most of her activities happen in the region shown in fig. 4-6(a). Her home is located in area 1. Across the river is a grocery store, where she usually goes for grocery shopping. Area 2 is a small commercial district, where her company and several restaurants are located. On weekdays, she usually leaves home and goes to work in the morning. But occasionally, she needs to meet customers in areas 3 and 4. She likes shopping very much. On weekends, sometimes, she meets her friends and has breakfast together in area 5; sometimes, she meets her friends in Harvard University in area 6. Then they go shopping in areas 7 and 8. There is a large mall in area 7, and area 8 is the downtown of Boston, where a lot of shops and restaurants are located. On weekdays, she usually has lunch in the restaurants close to her company. On weekends, she usually goes to some restaurants in the downtown area.

The periodically collected sensor samples and the PoIs discovered by our algorithm are shown in Fig. 4-6(a). From Fig. 4-6(b), we observe that the following places are discovered as PoIs: 1) Ms. A's home and the grocery store near her home in area 1; 2) her workplace and the restaurants that she likes to go to in area 2. 3) the customers' sites in area 3 and 4; 4) Harvard University in area 6; and 5) the shopping malls and restaurants in areas 7 and 8. We can also see that sensor samples related to movements between PoIs are successfully filtered out by the proposed algorithm.

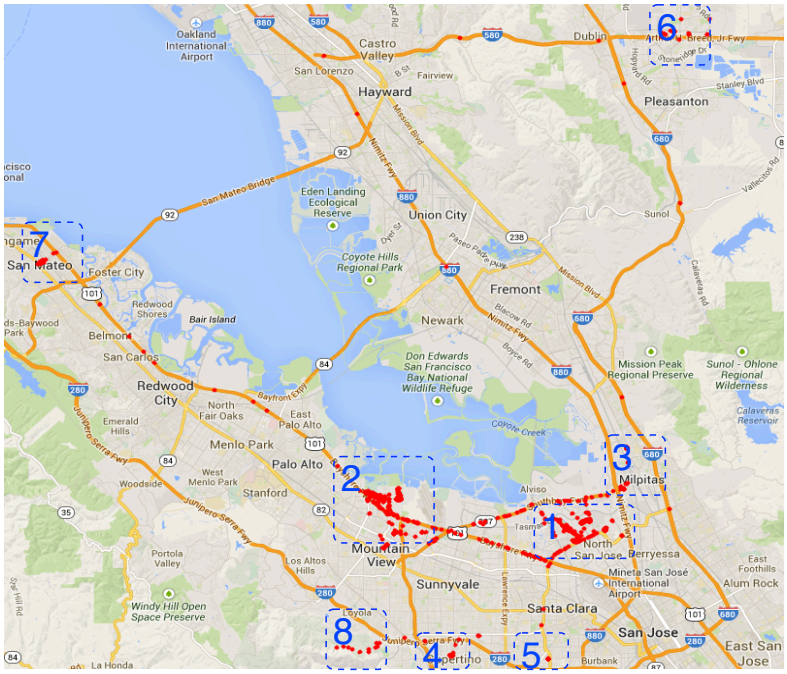


(a) Sensor samples

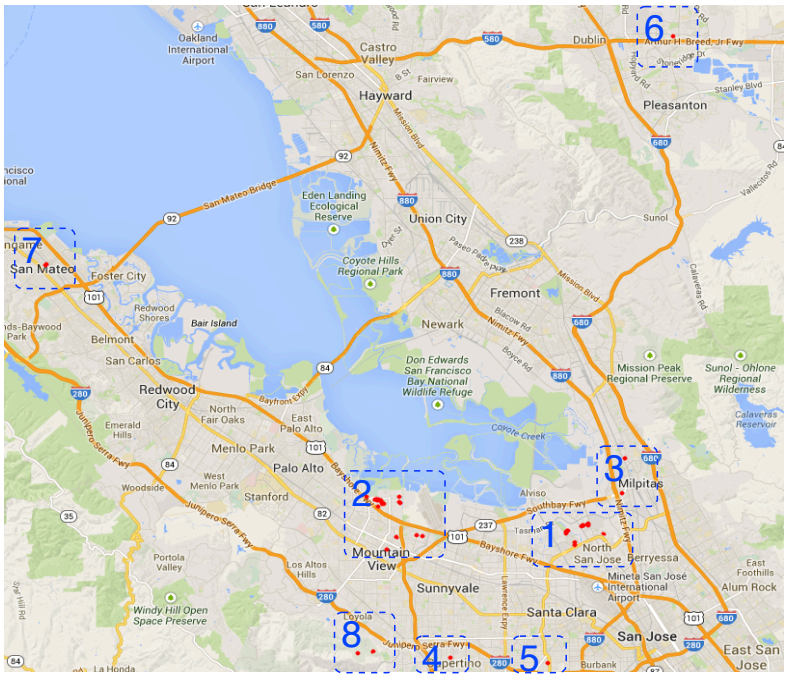


(b) Discovered PoIs

Figure 4-6: PoI discovery for Ms. A



(a) Sensor samples



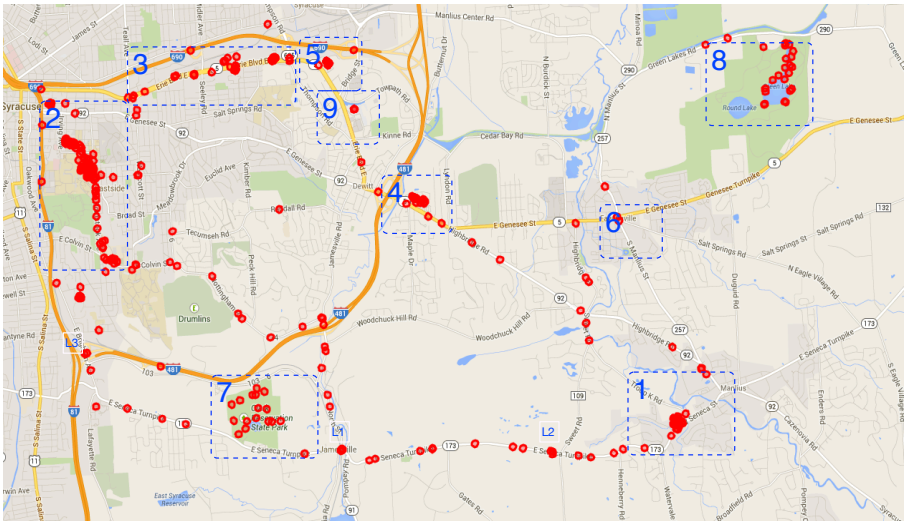
(b) PoIs

Figure 4-7: PoIs discovery for Mr. B

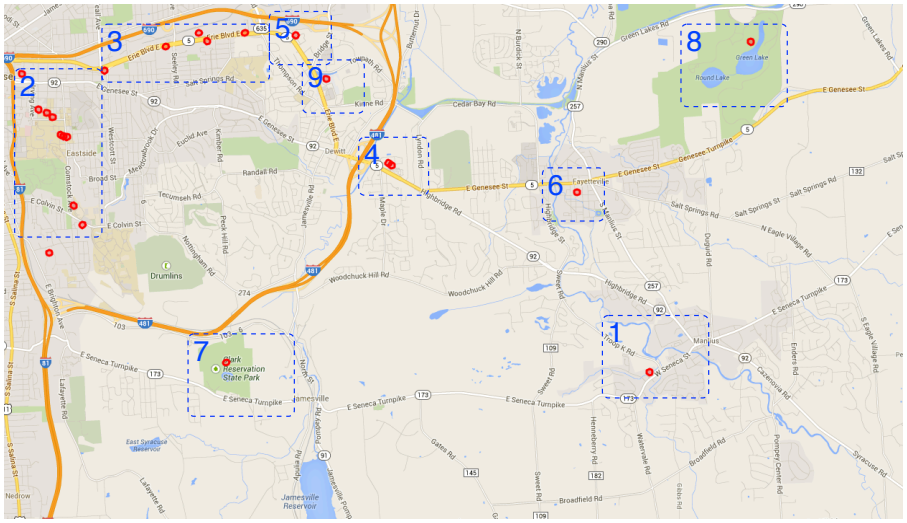
The second user Mr. B is an engineer who works and lives in the Great San Francisco area. Most of his activities occur in the region shown in Fig. 4-7(a). Area 1 is a residential community, where he and a lot of his colleagues live. A grocery store is located in this area, where he does grocery shopping almost every weekend. He and his friends take turns hosting parties over weekends, so most of his social activities occur in this area. He works in a company located in area 2, which is a high-tech company and has several buildings forming a large campus. The company provides its employees with free food and free facilities such as gyms. So Mr. B usually has breakfast, lunch and dinner in the cafes located on campus. He also does exercises in the gyms on campus. Sometimes he makes a little change to his life by having dinner in the restaurants near his workplace. On weekends, he and his family go shopping and have lunch in the commercial districts in areas 3–7, where shopping malls and great restaurants are located. In addition, if the weather is good, he likes to hike in area 8.

From Fig. 4-7(b), we observe that the following places are discovered as PoIs: 1) Mr. B's home in area 1; 2) his workplace, cafes and gyms on campus, and the restaurants he occasionally visits in area 2. 3) shops and restaurants in areas 3–7; and 4) hiking trails in area 8. Similarly, samples corresponding to movements between PoIs are filtered out.

The third volunteer Mr. C is a student studying and living in Syracuse, NY, whose activities occur in the area shown in Fig. 4-8(a). He lives in a suburban area 1. He drives to one of the two parking lots of his university almost every weekday morning in area 2. After parking his car, he either takes a bus or walks from the parking lot to his office on campus, depending on the weather condition. At lunch time, he usually walks to the restaurant area located at the northern part of his university and has a quick lunch there. Areas 3 and 4 are the districts, where several grocery stores are located. He does grocery shopping every Monday afternoon. Several fast food



(a) Sensor samples



(b) PoIs

Figure 4-8: PoIs discovery for Mr. C

restaurants are located in areas 5 and 6. He has dinner in these places sometimes. After dinner, most of time, he drives back to his office on campus and continues his work there till midnight then drives back to his home located in area 1. On weekends, he likes to spend one morning, either hiking on trails in area 7 or jogging around lakes in area 8. In addition, he sometimes visits an electronic store in area 9 for new and cool electronics.

From Fig. 4-8(b), we can see the following places are identified as PoIs of user C: 1) Mr. C's home in area 1; 2) his office, two parking lots and several restaurants where he usually has lunch in area 2; 3) the grocery stores and restaurants in areas 3–6; 4) a park in area 7 and a lake in area 8 where he likes to hike and jog respectively; and 5) an electronic store he occasionally visits in area 9.

We are not able to show the PoI discovery results of all the volunteers due to the space limitation. However, we found similar observations can be made from the experimental results of the other volunteers, i.e., the places he/she usually visits can be successfully discovered as PoIs and samples corresponding to movements between them can be filtered out.

Next, we show the experimental results to justify the effectiveness of the proposed activity prediction algorithm. In the experiments, we aimed to predict activities in the next 1 to 2 hours. To evaluate the activity prediction algorithm, we chose to use the widely used cross-validation method [34]. We split all the training data randomly into 10 disjoint sets. In each test, 9 of these training sets were used for training, and the rest data set was used to test the accuracy of prediction. Hence, a total of 10 tests were performed for each volunteer. We show the results in Fig. 4-9.

From the figure, we can see the activity prediction algorithm works well. Among all the volunteers, it predicts with an average of accuracy of 72%, the lowest confidence at 56% and the highest at 89%. Moreover, we find out the following factors may affect the prediction accuracy: 1) If a mobile user has a very regular schedule, and

his/her daily life follows a regular pattern, e.g., a college student, his/her activities can be predicted with high accuracy. Users D, I, J and N fall into this category. 2) If a mobile user has a quite flexible schedule in his/her daily life, it is hard to make accurate predictions. For example, Mr. M is a senior Ph.D student without any course work, so his schedule is quite flexible and his activities are relatively hard to predict. 3) It is hard to predict the activities of a mobile user who travels often. For example, Mr. L is an engineer, who often travels between cities for technical support. The accuracy of prediction for his activities is not as good as that for those who stay in a single city.

4.5.2 Evaluation of adaptive sampling

The proposed adaptive sampling algorithm was applied in the last three days of field tests. The threshold α was set to a relatively small value, 5%, during experiments. In this way, we can save sensing energy, while still preserving good performance of lifestyle learning. Suppose that the number of samples collected by our adaptive sampling algorithm and by periodical sampling (with the sampling period of 5min) are n' and n respectively. We chose to use the ratio $\frac{n-n'}{n}$ as the performance metric, which we call *energy saving ratio*. The corresponding results are shown in Fig. 4-10. From the figure, we can see that compared to periodic sampling, the proposed adaptive sampling algorithm achieves an energy saving of 52% on average, with the maximum saving at 63% and the minimum at 40%.

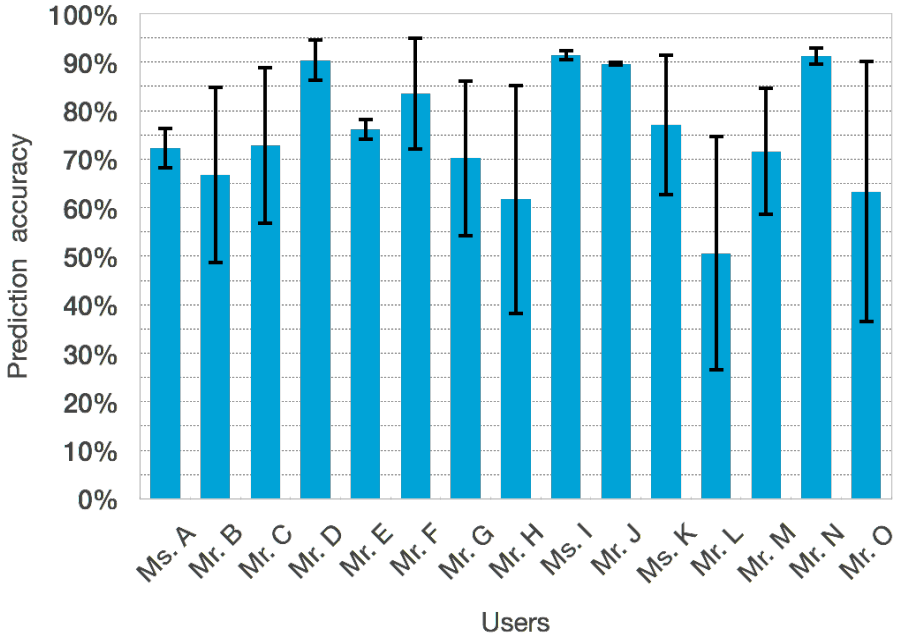


Figure 4-9: Cross-validation for prediction accuracy

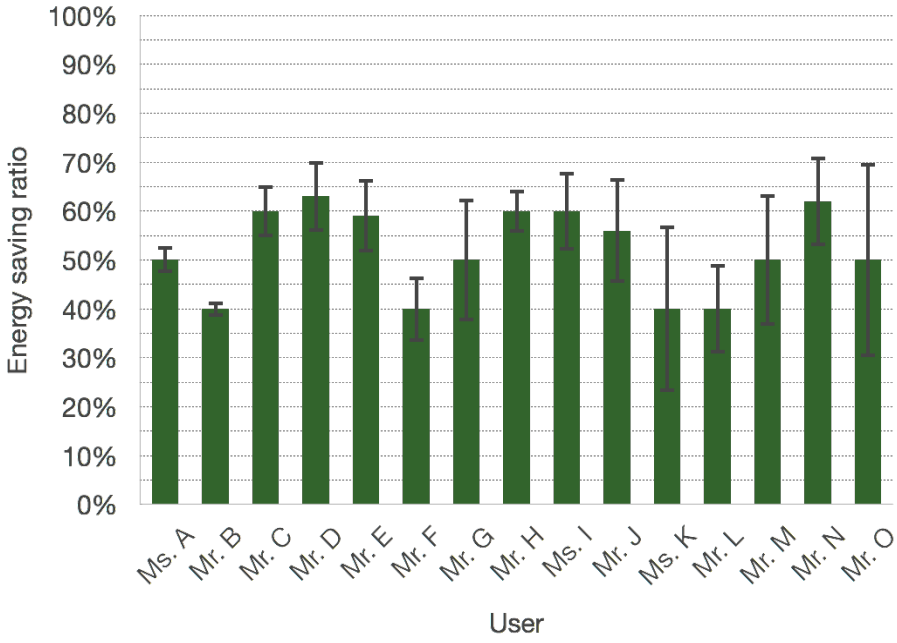


Figure 4-10: Energy savings achieved by adaptive sampling

Chapter 5

Collaborative Sensing

5.1 Overview

As we have analyzed in Section 1.2, even though smartphones' sensing applications are very attractive, performing sensing tasks using a smartphone may consume significant amount of energy. Moreover, most mobile crowd sensing applications are location-dependent, which may require location information to be reported along with sensed data. If energy-hungry GPS is turned on during the whole sensing procedure, the battery may be drained very quickly. Hence, without carefully managing very limited energy resources, a smartphone may end up with running out of its battery after performing a few sensing tasks. There is a large space for energy savings on a mobile phone. In this Chapter, we study how to minimize sensing energy consumption such that smartphones can undertake sensing tasks, and in the meanwhile, they can still fulfill their regular duties, such as phone calls, emails, etc. There is a large space for energy savings on a smartphone. However, fundamental energy-efficient resource management problems have not been carefully studied in the context of mobile crowd sensing.

As presented in Section 1.2, there are two mobile phone sensing paradigms [65]:

Participatory Sensing and *Opportunistic Sensing*. In this chapter, we focus on opportunistic sensing applications and aim to develop general (application-independent) methods to control the sensing procedure with the objective of minimizing sensing energy consumption. A commonly used method is to make every mobile phone sense periodically (every x seconds). This method is obviously not efficient because if this method is used, many redundant data reports may be produced for a target region by a large number of users which happen to show up in that area. Redundancy (i.e., the number of times a user senses) can be reduced and energy-efficiency can be improved by using a coordinator to control sensing activities of users such that those smartphones sense collaboratively to produce just enough data reports for the application. To this end, we propose to use a cloud-assisted collaborative sensing approach. Cloud computing has evolved as an important computing model, which can be leveraged to assist smartphone sensing by using servers in a cloud to not only handle data reports from smartphones but also collect mobility and location information from smartphones, calculate the best sensing schedule and tell them when/where to sense. Those sensors that are not needed for sensing can be turned off or work in a low power mode.

Only few recent works addressed collaborative sensing with smartphones. In [45], the authors presented analytical results on the rate of information reporting by uncontrolled mobile sensors needed to cover a given geographical area, and demonstrated the feasibility of using existing software and standard protocols for information reporting and retrieval to support a large system of uncontrolled mobile sensors using a testbed. In [72], the authors introduced mechanisms for automated mapping of urban areas that provide a virtual sensor abstraction to applications. They also proposed spatial and temporal coverage metrics for measuring the quality of acquired data. In [68], the authors proposed a protocol, Aquiba, that exploits opportunistic collaboration of pedestrians. Its performance was studied via simulations.

Collaborative sensing has been well studied for mobile sensor networks. In [79], Zhou et al. considered how to deploy mobile sensors into an existing sensor network to enhance its connectivity and coverage, and presented a dynamic programming based algorithm under the assumption that each sensor is equipped with GPS. Several distributed algorithms were presented for a sensing coverage problem in [78], which do not need any location and distance information. In [61], Saipulla et al. explored the fundamental limits of sensor mobility on barrier coverage and presented a sensor mobility scheme that constructs the maximum number of barriers with minimum sensor moving distance.

The differences between our work and these related works are: 1) The optimization problems considered in related works [45, 72] are mathematically different from the problems considered here. 2) Closely related works [68, 72] presented heuristic algorithms that cannot provide performance guarantees. We, however, present algorithms to produce optimal solutions, which can be used to justify the use of collaborative sensing in opportunistic sensing applications and show the potential energy savings quantitatively. This is the major contribution of this chapter. 3) The algorithms presented in [61, 78, 79] for mobile sensor networks (in which sensor mobility can be controlled to achieve certain sensing coverage) cannot be applied here because the mobility of smartphones is usually uncontrollable.

In this chapter, we study optimization problems related to energy-efficient collaborative sensing with smartphones. And our contributions are summarized in the following list:

1) Assuming knowing each user's moving trajectory in advance, we present a polynomial-time algorithm to obtain minimum energy sensing schedules. Even though this assumption may not be realistic, the obtained solutions can be used to show potential energy savings that can be brought by using collaborative sensing in smartphone sensing applications, and can also serve as a benchmark for performance evalua-

tion. We also address individual energy consumption and fairness by presenting an algorithm to find fair energy-efficient sensing schedules.

2) We present practical and effective heuristic algorithms to find energy-efficient sensing schedules under realistic assumptions.

3) We present simulation results based on real location (collected from the Google Map) and energy consumption (measured by the Monsosn power monitor [52]) data to show that collaborative sensing significantly reduces energy consumption compared to a traditional approach without collaborations, and the proposed heuristic algorithm performs well in terms of both total energy consumption and fairness.

In the second part of this chapter, we aim to design scheduling algorithms and a protocol for mobile crowd sensing without accurate locations (provided by GPS) with the objective of achieving a defined coverage requirement with limited energy consumption. Our contributions are summarized in the following:

1) We present a probabilistic model for sensing coverage without accurate location information, based on which we formally define the Energy constrained Maximum Coverage Sensing Scheduling (E-MCSS) problem for maximum coverage and the Fair Maximum Coverage Sensing Scheduling (F-MCSS) problem for addressing fairness on individual energy usages.

2) Assuming moving trajectories of mobile users are known in advance, we present a $(1 - \frac{1}{e})$-approximation algorithm and a $\frac{1}{2}$-approximation algorithm to solve the E-MCSS and F-MCSS problems in polynomial time, respectively. Even though this assumption might not be realistic, the solutions given by these theoretically-sound algorithms can serve as benchmarks for performance evaluation.

3) Under realistic assumptions, we present a GPS-less energy-efficient protocol for sensing scheduling based on the proposed approximation algorithms.

4) We developed an Android-based mobile crowd sensing system, on which we implemented the proposed protocol. We present simulation results based on real location data (collected from the Google Map) as well as experimental results from a field test on Syracuse University's campus to validate and justify effectiveness of the proposed algorithms and protocol.

To the best of our knowledge, we are the first to present theoretically well-founded and practically efficient mathematical model, algorithms and protocol for coverage (without accurate locations) and energy-efficient sensing scheduling in the context of mobile crowd sensing.

The rest of this chapter is organized as follows. With the assumption that the user's accurate location is available, We describe the system model and formally define the problems in Section 5.2.1. The proposed algorithms are presented in Section 5.2.2. We present the simulation results in Section 5.2.3. To address the GPS-less collaborative sensing problem, we formulate the problem in Section 5.3.1, propose the probabiliy based sensing model and present the approximate algorithms. The evaluation of GPS-less sensing algorithm is presented in Section 5.3.4.

5.2 Collaborative Sensing

5.2.1 Problem Definition

In this section, we describe the system model, introduce necessary notations and then formally define the problems.

First, we summarize the major notations in Table 5.1.

We consider a smartphone sensing system with multiple mobile users, each of which carries a smartphone equipped with sensors. The mobility of each mobile user cannot be controlled. However, mobile users' movements are highly restricted by roads, i.e., a vehicle or a person can only move along roads and turn at intersections.

Table 5.1: Major notations

$G(V, E)$	The VSG and its vertex and edge sets.
M	The number of roads in the target region.
N	The number of mobile users/phones
S_i	The sensing schedule of user i
T	The deadline of the given sensing task.
w_i	The energy needed to sense once using user i's phone.
Γ_i	The moving trajectory of user i

The movements of a mobile user i can be characterized using a trajectory Γ_i which is a set of 3-tuples (i, t_i, loc_i) and each of them gives the location of user i at time t_i. The more 3-tuples there are in the trajectory, the more accurately it can characterize the movements of mobile user i.

Again, we focus on the opportunistic sensing scenario in which sensors on each smartphone automatically perform sensing tasks without user involvement. Each sensor is assumed to have a sensing range of r, which basically means if a user sense at a location loc_x and obtain a reading, then there is no need to sense again in any location within the disk with the origin at loc_x and a radius of r since the readings will be similar. Sensing target areas are roads in a given region, which are assumed to be narrow chains. The width of a road is assumed to negligible because r is usually larger or much larger. We consider regular roads which are roughly straight roads. Those irregular roads can be treated as a sequence of regular roads. If a user is on a road, it is assumed to cover a segment (b, c) with a length of $2r$. We can be more or less conservative on coverage by setting the value of r to a smaller or larger value. In the following, we will use *user*, *phone* and *sensor* interchangeably.

Given a sensing application, multiple servers are set up in a cloud to coordinate sensing activities. Servers are assumed to periodically exchange information to have a consistent view of smartphones in the system. Each smartphone can exchange information with one of the servers using its wireless interface. Our approach is to

use servers to gather information from smartphones, determine when/where to sense for each phone, send the sensing schedule to smartphones and then collect sensed data reports from them.

Every 3-tuple in a trajectory can be imagined as a *virtual sensor*. If trajectories of all users are given, then we will have a large network of *virtual sensors* by combining all 3-tuples in trajectories. A sensing schedule S is a collection of virtual sensor sets, i.e., $S = \bigcup_{i=1}^{N} S_i$, where $S_i \subseteq \Gamma_i$. $|S_i|$ gives the number of times user i (smartphone of user i) senses. Note that performing a common sensing task may consume different amount of energy on different phones. For example, energy consumption of a WiFi scan on three popular Andorid-based smartphones can be found in Table 5.2.

Since energy efficiency is the primary design goal of this work, we want to minimize total energy consumed in the whole sensing procedure. So we define the following optimization problem.

Definition 4 (MECSS) *Given a region, M roads in the region, N mobile users, a deadline T and the moving trajectory Γ_i of each user $i \in \{1, \cdots, N\}$ before the deadline, the* **Minimum Energy Collaborative Sensing Scheduling (MECSS)** *problem seeks a sensing schedule $S_i \subseteq \Gamma_i$ for each user i, such that its total energy consumption $\sum_{i=1}^{N} w_i|S_i|$ (where w_i is the energy needed to sense once with the smartphone of user i) is minimized subject to the constraint that the roads in the given region are fully covered before the deadline T.*

However simply minimizing the total energy consumption may lead to unfair utilizations of smartphones, some users' phones are heavily used for sensing and other users' phones are lightly utilized or not utilized at all. A similar issue has been shown by previous works [67] for wireless mesh and sensor networks: simply maximizing network throughput leads to severe unfairness on users' individual throughput. Therefore, we try to improve fairness by only considering those sensing schedules with the min-max number of user sensing times. We call such schedules *min-max fair* sensing

schedules. We also study the *Fair Energy-efficient Collaborative Sensing Scheduling (FECSS)* problem which seeks a min-max fair sensing schedule $S_i \subseteq \Gamma_i$ for each user i, such that its total energy consumption is minimized.

Assuming knowing the trajectory of each mobile user in advance, we present algorithms to solve these sensing scheduling problems optimally, which are presented in Section 5.2.2. It may be argued that these assumptions are not realistic since it is hard to precisely predict how users move in the future and mobile users need to turn on the GPS devices on their smartphones to obtain precise locations, which, however, are energy-hungry [40] (a GPS device usually consumes much more energy than other sensors). However, the optimal solutions can be used to show energy savings that can potentially be achieved by collaborative sensing and they can serve as a benchmark for performance evaluation, i.e., can be used to find out how far a sensing schedule produced by a practical heuristic algorithm is away from the optimal. Therefore, it makes sense to present the optimal algorithms. We also present two practical and simple heuristic algorithms in Section 5.2.2 to find energy-efficient sensing schedules for mobile users, which do not need moving trajectories beforehand or precise locations.

5.2.2 Algorithms

In this section, we present optimal algorithms and practical heuristic algorithms for collaborative sensing.

Optimal Algorithms

In order to solve the MECSS problem defined above, we first introduce a graph model, Virtual Sensor Graph (VSG) $G(V, E)$, to assist computation. As mentioned above, the moving trajectory of each mobile user is assumed to be known and each 3-tuple (i, t_i, loc_i) in a trajectory can be viewed as a *virtual sensor*. The sensing scheduling problem is actually to find a subset of virtual sensors to cover the roads in the target

region. This graph is a multi-layer directed graph and each layer corresponds to a road L in the target region. Every vertex corresponds to a virtual sensor v_j (associated with user i). Let (b_j, c_j) and $(b_{j'}, c_{j'})$ be segments of L covered by virtual sensors v_j and $v_{j'}$ (that are on L), respectively. There is a directed edge from v_j to $v_{j'}$ if $b_j < b_{j'} \leq c_j < c_{j'}$ (i.e., their coverage areas overlap). Its capacity and cost are set to 1 and $w_{i'}$ (virtual sensor $v_{j'}$ is associated with user i'), respectively. For a virtual sensor (vertex) on multiple roads (e.g., virtual sensors at inter-sections are on two or more roads.), we arbitrarily pick a corresponding layer (road) to place it in G. Moreover, for each virtual sensor v (associated with user i) covering segments on multiple roads, we have to create a pair of vertices (v^{in}, v^{out}) to represent it in G, and there is a directed edge from v^{in} to v^{out} whose capacity and cost are set to ∞ and w_i respectively. We call such virtual sensors *cross-road* virtual sensors and those edges *intra-vertex* edges. There is a directed edge from v^{out} to another vertex u or from another vertex u to v^{in} if the aforementioned condition is met. The capacities and costs of all incoming edges associated with a cross-road virtual sensor are set to 1 and 0 respectively. However, the capacity and cost of an outgoing edge (to vertex u associated with user h) associated with a cross-road virtual sensor are set to 1 and w_h respectively. Note that another vertex u here may be in the same layer or in a different layer. Therefore, edges associated with cross-road virtual sensors may cross layers.

In addition, we also add a virtual source s_m for each layer $m \in \{1, \cdots, M\}$ and edges from s_m to all the vertices whose corresponding virtual sensors cover the western/southern boundary of the corresponding road with their capacities and costs set to 1 and the energy cost of the user associated with its destination vertex, as well as a virtual sink z_m and edges from all the vertices whose corresponding virtual sensors cover the eastern/northern boundary of the corresponding road to z_m with their capacities and costs set to 1 and 0 respectively. In addition, there are an ultimate

virtual source s and sink z. There are also edges from s to virtual sources in all layers with their capacities and costs set to 1 and 0, and edges from virtual sinks in all layers to z with their capacities and costs set to 1 and 0 too. A simple example in Fig. 5-1 is used to illustrate the graph construction. In this example, we have 6 virtual sensors and two roads. Virtual sensors $1, 2, 3$ are on road 1 and are assumed to be associated with user 1; and virtual sensors $4, 5, 6$ are on road 2 and they are assumed to be associated with user 2, as illustrated by the first sub-figure. The corresponding 2-layer VSG is given in the second sub-figure, in which the first number associated with each edge is its capacity and the second number is its cost. In this example, vertex v_5 corresponds to a cross-road virtual sensor which can be used to cover both roads 1 and 2. Intuitively, such virtual sensors should be fully leveraged to reduce sensing energy consumption. We present our optimal algorithm for the MECSS problem as follows.

Algorithm 5 The optimal MECSS algorithm

```
1: Construct the VSG G(V, E);
2: Solve the LP relaxation of the ILP-MinE;
3: if No feasible solution then
4:     return "There is no feasible solution!";
5: else
6:     return the corresponding sensing schedule;
7:
8: end if
```

Unknown decision variables:

1) f_e ($e \in E$): the amount of flow on link e.

2) $x_e = \{0, 1\}$ ($e \in E$): If $x_e = 1$, link e in G is selected; $x_e = 0$, otherwise.

ILP-MinE:

$$\min \sum_{e \in E} w_e x_e \tag{5.1}$$

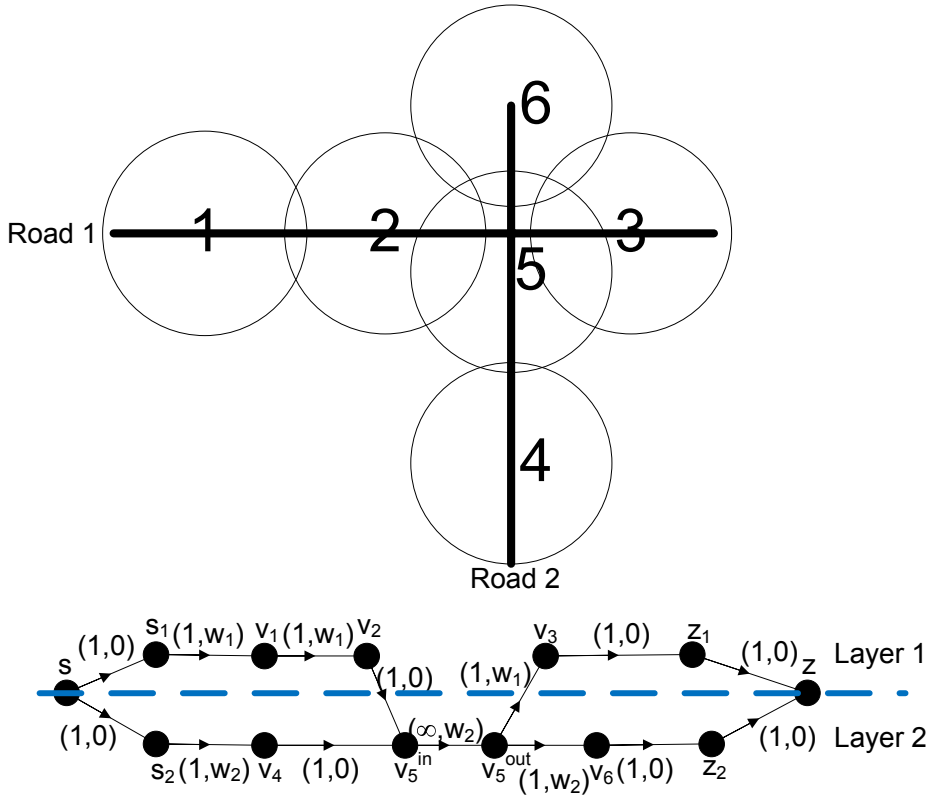


Figure 5-1: Roads, virtual sensors and the corresponding VSG

Subject to:

$$\sum_{e \in E_s^{out}} f_e = M, \tag{5.2}$$

$$\sum_{e \in E_v^{out}} f_e = \sum_{e \in E_v^{in}} f_e, \qquad \forall v \in V \setminus \{s, z\}; \tag{5.3}$$

$$f_e \leq C_e, \qquad \forall e \in E; \tag{5.4}$$

$$x_e \begin{cases} \geq f_{e'}, \forall e \in E^{intra}, \forall e' \in E_e^{in}; \\ = f_e, \forall e \in E - E^{intra}. \end{cases} \tag{5.5}$$

In this algorithm, the LP relaxation of an Integer Linear Programming (ILP), ILP-MinE, needs to be solved to obtain optimal solutions. In the ILP, f_e and x_e ($e \in E$) are both integer variables and E_v^{in}/E_v^{out} is the set of incoming/outgoing edges of vertex v on G. E^{intra} is the set of intra-vertex edges and E_e^{in} is the set of

incoming edges associated with the source vertex of edge e on G. Once we obtain values for x_e by solving the corresponding LP relaxation (which are guaranteed to be 0 or 1), then we can figure out which virtual sensors should be selected for sensing (i.e., which user should sense at when and where). Specifically, if $e = (u, v)$ is a regular edge in G and $x_e = 1$, then the virtual sensor corresponding to its destination vertex v will be selected for sensing. If e is an intra-vertex edge and $x_e = 1$, then obviously the virtual sensor corresponding to e will be selected for sensing. We have the following proposition.

Proposition 2 *Algorithm 5 optimally solve the MECSS problem in polynomial time.*

Proof 1 *The importance of the SVG lies in the fact that any feasible integer $s-z$ flow with a total flow amount of M (the number of roads in the target region) gives a feasible (in terms of coverage) sensing schedule. This is because our graph construction guarantees that every integer $s-z$ flow in a layer (which may include edges from different layers) corresponds to a sensing schedule that can fully cover the road corresponding to that layer. For example, in Fig. 5-1, an integer flow $(s, s_1, v_1, v_2, v_5^{in}, v_5^{out}, v_3, z_1, z)$ corresponds to a sensing schedule with virtual sensors $1, 2, 5$ and 3. Note that we introduce two virtual vertices in each layer that are used to deal with the case where multiple virtual sensors may be able to cover the head of a road. Setting corresponding edges' capacities to 1 ensures that fully covering each road once instead of covering a road more than once but leaving some other roads not fully covered.*

In addition, the costs of most regular edges (except those associated with virtual vertices) are set to the energy cost of users (associated with their destination vertices). Then the selection of an edge $e = (u, v)$ basically means the corresponding virtual sensor (i.e., the virtual sensor corresponding to vertex v) is added to the sensing schedule. For those vertices corresponding to cross-road virtual sensors which may make contributions for covering of multiple roads, the costs of all the corresponding incoming cross-layer edges are set to 0 and the cost of the corresponding intra-vertex

edge is set to the energy cost of the corresponding user. This way of assigning link costs along with Constraints (5.5) ensure that no matter how many roads can benefit from the coverage contributions made by using this vertex (virtual sensor), it is only counted once. The costs of those edges associated with virtual vertices are also assigned properly (e.g., cost$(e) := 1$*, where* $e = (s_i, v)$*; while cost*$(e') := 0$*, where* $e' = (v', z_i)$*) such that by counting the total costs of selected edges, we can find out the total energy consumption. Hence, due to the way how the costs of edges in a VSG is assigned, we can claim that a minimum cost* $s - z$ *flow with a flow amount of* M *actually corresponds to a feasible (coverage-wise) sensing schedule with the minimum energy consumption.*

By replacing variables x_e *in the objective function with Constraints (5.5), we can see that solving the ILP-MinE is equivalent to solving a series of ILP, each of which has Constraints (5.2)–(5.4) and an objective function of* $\min \sum_{e \in E - E^{intra}} w_e f_e + \sum_{e \in E^{intra}} w_e f_{e'}$ *(where* e' *is one of incoming edge of the intra-vertex edge* e*); and then take the maximum of all objective values. Each such an ILP is a minimum-cost-flow-like problem, whose coefficient matrix is totally unimodular [73].*

It is known that solving the LP relaxation of such an ILP problem automatically yields integral optimal solutions [73]. Furthermore, the ILP-MinE obviously includes polynomial numbers of variables and constraints. Therefore, the LP relaxation of the ILP-MinE can be solved by existing algorithms [4] in polynomial time, which can yield integral optimal solutions. This completes the proof.

The FECSS problem can be solved by an algorithm similar to Algorithm 5. Instead of solving the ILP-MinE, if we solve two following ILPs sequentially, then we can obtain a fair energy-efficient sensing schedule. Specifically, we first solve the ILP-Maxmin and obtain the min-max number of sensing times β. Because of Constraints (5.7) and the objective function, we can guarantee that for any feasible solution given by solving the ILP-Maxmin is min-max fair (according to our definition

above). Next, we feed β to the ILP-FECSS as a parameter, which has the objective function of minimizing the total energy consumption and Constraints (5.7). Here, E_i is the set of edges associated with user i. Note that for a user with cross-road virtual sensors, only the corresponding intra-vertex edges are counted. Therefore, solving the ILP-Maxmin and ILP-FECSS in sequence can provide an optimal solution for the FECSS problem. We also used solutions generated by this algorithm as a benchmark for comparison in our simulation.

ILP-Maxmin:

$$\min \beta \tag{5.6}$$

Subject to: Constraints (5.2)–(5.5)

$$\sum_{e \in E_i} x_e \leq \beta, \quad \forall i \in \{1, \cdots, N\}; \tag{5.7}$$

ILP-FECSS(β):

$$\min \sum_{e \in E} w_e x_e$$

Subject to: Constraints (5.2)–(5.5) and (5.7)

Practical Heuristic Algorithms

In this section, we present two practical heuristic algorithms. First, we do not assume the moving trajectory of each user is known; second, we do not assume that users use their energy-hungry GPS devices all the time. However, without knowing anything about mobile users, the only thing we can do is probably to let them sense

periodically. Therefore, we do assume that users' moving directions and speeds can be detected and measured using some sensors (such as accelerometer and digital compass) on smartphones and a method such as that introduced in [10]. Furthermore, the GPS device is assumed to be turned on right after a user initiates a sensing task to provide one or multiple reference locations for mobility prediction. It can certainly be (automatically or manually) turned off after necessary information is collected. In this way, a mobile user can keep track of where he/she is during the sensing procedure. In addition, every time a mobile user enters a new road segment (which can be detected by the smartphone by measuring the distance travelled and detecting the direction change using an accelerometer and a digital compass), a short report message will be automatically sent to a server by his/her smartphone. Note that in this section, a road segment is defined by two intersections, which may be different from the "road" (which may include a set of consecutive road segments) considered in the last section.

Both heuristic algorithms are used by a server to calculate a sensing schedule which will then be broadcast to mobile users. The first algorithm can be viewed as a realistic way to apply the optimal algorithms presented above. The basic idea is to sequentially use an algorithm presented above with partial trajectories that can be predicted to find out how to sense for the next certain period of time. We call this algorithm the *prediction-based* algorithm, which is presented as follows.

Algorithm 6 The prediction-based algorithm

1: Predict users' moving trajectories (until the earliest time a user will enter a new road segment) according to their current locations and mobility information;
2: Generate virtual sensors according to the predicted partial trajectories;
3: Based on these virtual sensors, construct a connected VSG and apply the optimal MECSS or FECSS algorithm to calculate a new sensing schedule and broadcast it to mobile users;
4: Update the target region by removing the road segments that have been covered;
5: Update the number of times each user has already sensed;

In this algorithm, we do NOT predict how a user will do in an intersection (make a turn, go straight, or even u-turn), which is hard. Instead, we apply an algorithm to predict how the user will move towards the intersection he is facing, from which we can obtain a partial trajectory (for each mobile user) that characterizes his movement from current location to wherever he will reach at the earliest time a user (himself or another one) will reach an intersection. Note that any prediction algorithm (e.g., an application-specific prediction algorithm) can be applied here. In the simulation, we used a simple but practical method, which assumes that the user will move towards the intersection he is facing without changing his direction or speed. The VSG constructed based on partial trajectories may not be connected. We simply connect disconnected components (if there are any) by connecting vertices on the edge to produce a connected graph. In Step 2, every time the same algorithm, the optimal MECSS or FECSS algorithm, is applied, however, the input changes over time because the algorithm needs to take account of the portions of roads that have been covered as well as the number of times each user has already sensed. In the simulation, we used the optimal FECSS algorithm. Obviously, the prediction-based algorithm does not always yield optimal solutions, however, we show that it works fine on average cases via simulations.

This second algorithm uses a function of a couple of sensing-related factors to make sensing decisions for mobile users. Hence, we call it the *function-based* algorithm, which is formally presented as Algorithm 7.

First, this algorithm generates virtual sensors on the roads in the target region to make sure all the roads are fully covered and tries to find a sensing schedule with a minimum subset of virtual sensors (i.e., a minimum number of sensing times) to cover all the roads in the target region. This can be easily done by applying the optimal MECSS algorithm described above with $w_i (i \in \{1, \cdots, N\})$ set to 1.

Again, we assume that when a user enters a new road segment L, it will notify the

Algorithm 7 The function-based algorithm

```
1: Generate virtual sensors according to the roads in the target region to make sure
   all the roads are fully covered;
2: Select a minimum subset of virtual sensors that can cover all the roads in the
   target region using the MECSS algorithm and store them in V_S;
3: while True do
4:     if Receive a report about a new road segment L then
5:         if V_S = ∅ OR time is up then return ;
6:         else
7:             Use a function to determine the number J of virtual sensors in V_S^L that
   need to be used;
8:             Notify the user to use the first J virtual sensors and remove them from
   V_S^L;
9:         end if
10:    end if
11: end while
```

server with a report message. Then the server needs to determine how to make this user sense in this new segment. In the algorithm, $V_S^L \subseteq V_S$ is the subset of virtual sensors on L that are selected in the second step. The V_S^L is updated every time after some virtual sensors are used (i.e., some users used their sensors to sense at times and locations specific by these virtual sensors).

Our function-based algorithm can rather be considered a general optimization framework that uses a sensing-related function to determine how many times a user should sense in the road segment he/she enters. Any function can be used here, however, it may not lead to good performance. Here are some guidelines for designing a "good" function: 1) The function value should decrease with the number of times this user has already sensed for fairness purpose. 2) It should increase with the decrease of time left to perform the sensing task. 3) If no (or almost no) time left, $J = |V_L^S|$, where J is the value returned by the function. We suggest to use the following exponential function in the algorithm:

$$f(t, T, q, \overline{q}, c) = \lceil e^{-c\frac{q}{\overline{q}}\frac{t}{T}} |V_S^L| \rceil, \tag{5.8}$$

where t and T are the time left to complete the sensing task and the deadline respectively; q and $\overline{q}$ are the number of times this user has already sensed and $\overline{q}$ is the average number of sensing times among all users. c is a tunable parameter. This function certainly satisfies the three requirements mentioned above and its values fall in the range $(0, |V_S^L|]$. Note that $|V_S^L|$ gives the number of virtual sensors left for use. Furthermore, the value of c can be set in a certain way to achieve a good tradeoff between coverage and fairness. Specifically, a smaller c can ensure coverage but may lead to unfair sensing schedule (some users' phones may be abused!); however, a larger c leads to fair sensing but may result in loss of coverage. We performed simulations to evaluate the performance of this function and to investigate what is the best value for the constant c, which will be discussed in details later. Certainly, some other application-specific factors may be brought into the function to (hopefully) improve the performance further. But we try to design a general approach here, and we found that this algorithm with the function in (5.8) performed very well from simulation results.

5.2.3 Evaluation

In this section, we present and discuss simulation results to show the performance of the proposed algorithms.

In the simulation, WiFi signal sensing was considered to be our application. We selected three popular Android-based smartphones, Google Nexus S [55], Samsung I9000 and S5830 [62], as our sensing devices. The energy consumed by these phones for conducting a WiFi scan was measured by the Monsoon power monitor [52] (particularly designed for smartphones), which are summarized in the following table. In the simulation, the smartphone of a user was randomly selected from these three kinds of Android phones. The sensing range r was set to 7 meters. Since most of current smartphone sensing projects were conducted in urban areas, we picked a typ-

ical urban area to carry out simulation runs. As shown in Fig. 5-2 obtained from the Google Map, the target region is located at Manhattan, NY, which spans 4 blocks from west to east with a total length of 1.135km and 4 blocks from south to north with a total width of 0.319 km, and includes the 6th,7th,8th Avenues and the 45th, 46th,47th Streets.

Table 5.2: Energy consumption of a WiFi scan

Phone Models	Energy Consumption(μAh)
Google Nexus S	30.99
Samsung S5830	16.25
Samsung I9000	54.08

We used a mobility model similar to the well-known Manhattan model [3] to generate mobile users' moving trajectories. Specifically, each user was assumed to enter the target region from a road at a random time, randomly pick a speed from $\{2, 4\}$ meters per second, move towards an intersection, and then move straight ahead with a probability of 50% and turn left or right with equal probabilities (i.e., 25%). The trajectory of each user was constructed with evenly paced sample points (6 meters between two consecutive ones) and points on critical locations (such as intersections and road heads.) The location data were collected from the Google Map using its API.

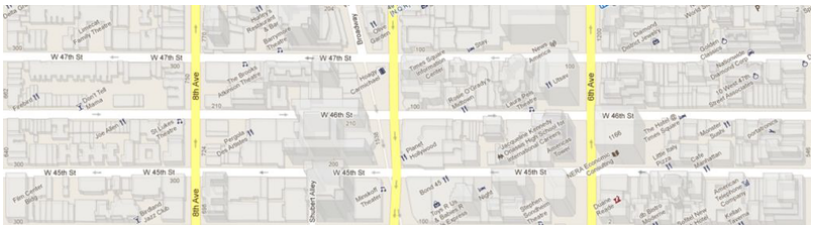


Figure 5-2: The target region

We compared our algorithms, the prediction-based algorithm (labeled as "Prediction-Based"), the function-based algorithm (labeled as "Function-Based"), the optimal MECSS algorithm (labeled as "MinTotalEnergy"), the optimal FECSS algorithm

(labeled as "FECSS") against a baseline approach in which every user performs a WiFi scan every 3 seconds. The total energy consumption, the variance of the number of sensing times, the maximum number of sensing times were used as performance metrics to show the energy consumption as well as fairness. In the simulation scenario, we increased the number of users from 25 to 50 with 5 as the step size. The simulation results are presented in Figs. 5-3–5-6.

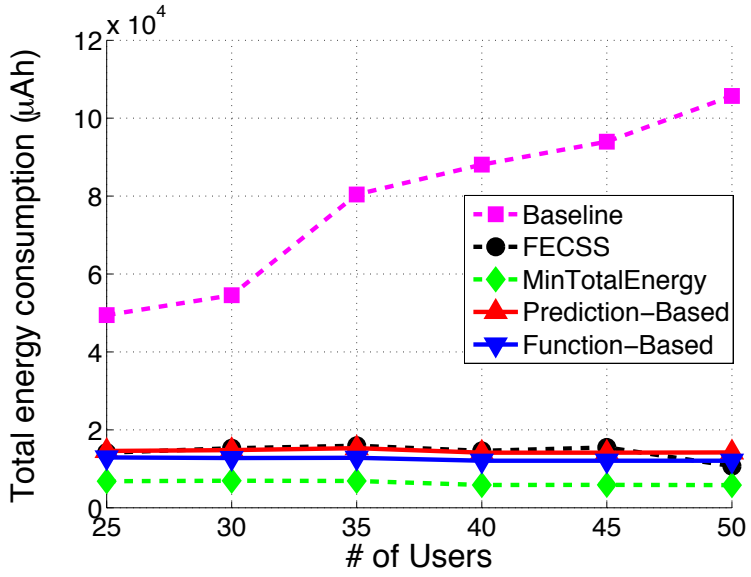


Figure 5-3: The total energy consumption

From these simulation results, we can make the following observations.

1) From Fig. 5-3, we can see that in terms of total energy consumption, all the proposed algorithms perform very well. Specifically, compared to the baseline approach, the optimal MECSS algorithm, the optimal FECSS algorithm, the prediction-based algorithm and the function-based algorithm significantly reduce energy consumption by 91%, 80%, 80% and 82% respectively, on average. The optimal MECSS algorithm is certainly the best in terms of this metric. The optimal FECSS algorithm tries to minimize total energy consumption under the constraint of achieving the min-max number of sensing times. Therefore, the total energy consumption given by this algorithm is larger than the minimum value. In addition, the function-based

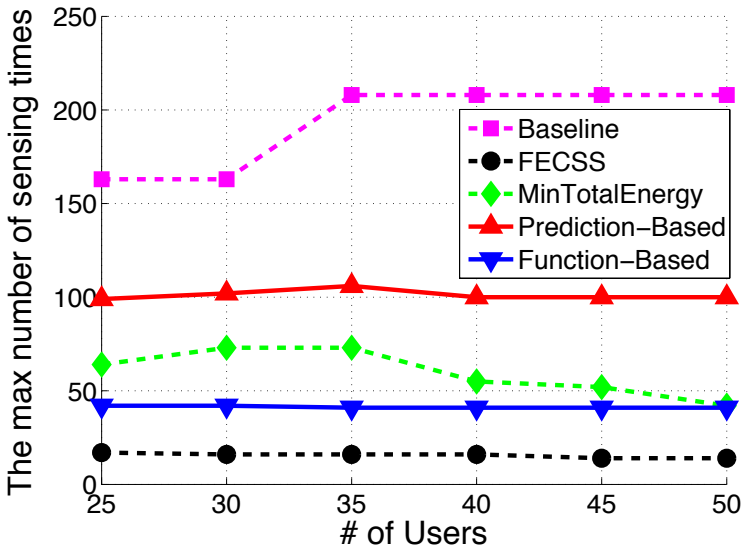


Figure 5-4: The maximum number of sensing times

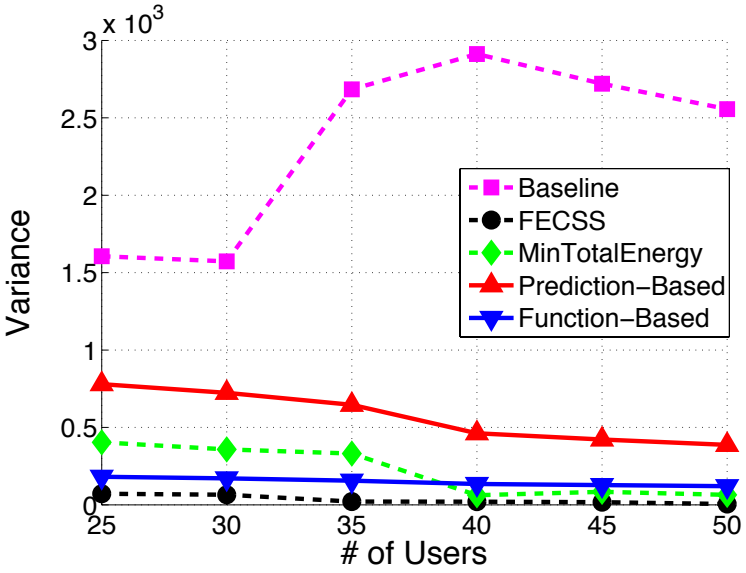


Figure 5-5: The variance of the number of sensing times

algorithm perform very well: close to the optimal MECSS algorithm and better than the prediction-based algorithm and the optimal FECSS algorithm (in terms of total energy consumption).

2) In Fig. 5-4, we show the fairness of the sensing schedules given by each algorithm in terms of the maximum number of sensing times. Since one of the constraints of the

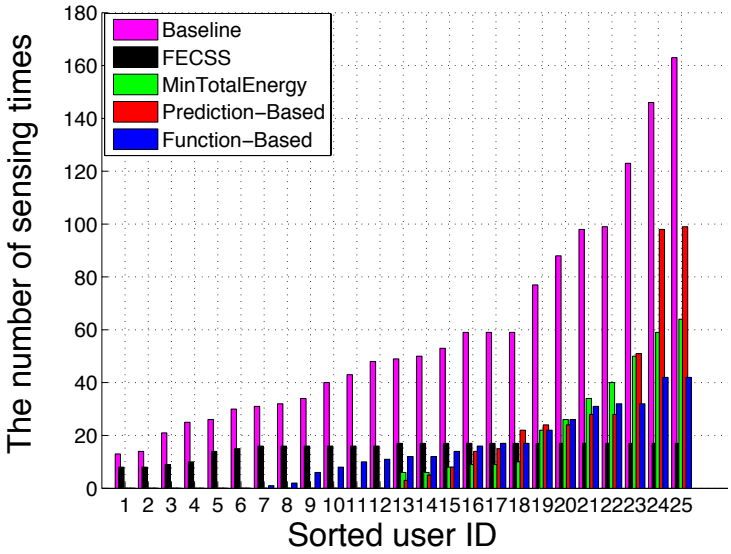


Figure 5-6: The number of sensing times VS. the sorted user ID

FECSS problem is to achieve the min-max number of sensing times. Therefore, we first used the maximum number of sensing times as a metric to evaluate the fairness performance. Clearly, the optimal FECSS algorithm is the best in terms of this metric since it is guaranteed to produce a solution in which the maximum number of sensing times is minimum among all possible solutions. This was verified by the results in Fig. 5-4. None of the other four algorithms can provide any guarantee for this metric. Not surprisingly, the baseline approach still performs worst. This is because the number of sensing times given by the baseline approach depends on how long a user stays in the target region, which can be arbitrarily large. An interesting observation is that the function-based algorithm outperforms both the optimal MECSS algorithm and the prediction-based algorithm.

3) In Fig. 5-5, we also used the variance of the number of sensing times as a metric to evaluate the fairness performance of algorithms. For all algorithms presented here, their performance in terms of variance matches that in terms of the maximum number of sensing times. Specifically, the optimal FECSS algorithm performs best as expected and the baseline approach is still the worst one. The function-based

algorithm performs well too. On average, the optimal MECSS algorithm, the optimal FECSS algorithm, the prediction-based algorithm and the function-based algorithm outperform the baseline approach by 89%, 98%, 72% and 93% respectively.

In addition, we also present a bar graph in Fig. 5-6 to show how the number of sensing times is distributed over 25 users. In this figure, x-axis is the sorted user ID. As can be clearly seen, the number of user sensing times is distributed quite evenly over all the users if the optimal FECSS algorithm is used to determine the sensing schedule. If the baseline approach is used, every user needs to sense a few times, however, the distribution is not even at all.

4) From Fig. 5-3, we can also see that the total energy consumption given by a proposed algorithm does not increase (decreases slowly in most cases) with the number of users. As long as a sensing task is undertaken by mobile users in the target region collaboratively, more users usually offer more flexibility for sensing scheduling, which should better performance on energy consumption. As expected, the total energy consumption given by the baseline approach increases sharply with the number of users. This is because with this algorithm, each user senses individually without any collaborations. Hence, increasing the number of users does not necessarily bring any benefits. This observation well justifies the advantage of using collaborative sensing. Similar observations can be made for the other two metrics from Figs. 5-4–5-5.

In short, we can make two conclusions from the discussions above: 1) Compared to traditional smartphone sensing without collaborations, collaborative sensing significantly reduces energy consumption. 2) The proposed function-based algorithm perform well in terms of both total energy consumption and fairness.

Since the function-based algorithm seems a promising method for collaborative sensing, we decided to study it further via simulation by investigating how the value of c (the tunable parameter in the exponential function) affects fairness. In this scenario, $N = 50$ and we used the same input as before. From the results in Figs. 5-

7–5-8, we can see that the maximum number of sensing times and the variance given by the algorithm decrease with the value of c as expected. However, we found that if we increased it to a value larger than 2, we lost full coverage of some roads in the target region in some cases, which is not acceptable.

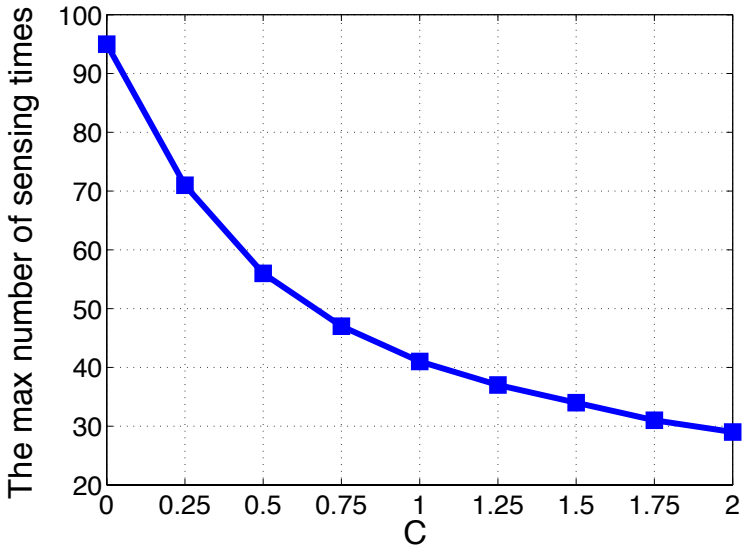


Figure 5-7: The value of c VS. the maximum number of sensing times

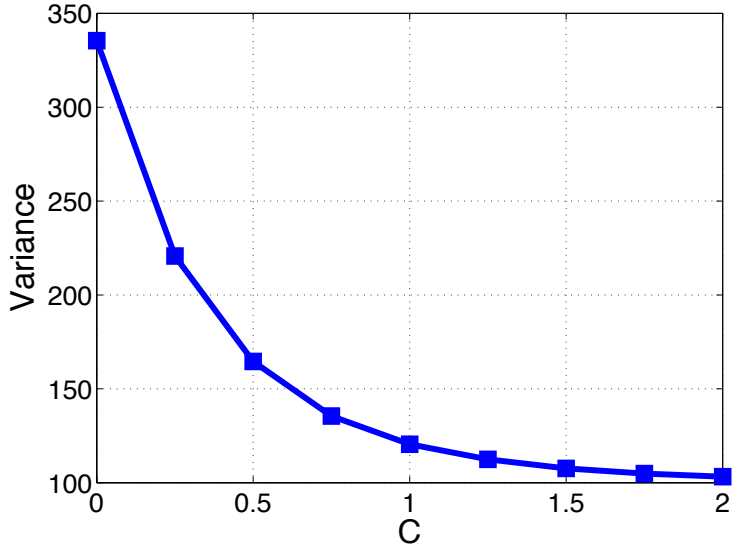


Figure 5-8: The value of c VS. variance

5.3 GPS-less Collaborative Sensing

5.3.1 Problem Formulation

As shown in previous sections, cloud coordinated collaborative sensing could efficiently save the smartphones' power by remove the redundant data collection. Moreover, since most mobile crowd sensing applications are location-dependent and GPS is well known as a energy-hungry device, there is still a large space for energy savings. From this section, we aim to design scheduling algorithms and a protocol for mobile crowd sensing without accurate locations (provided by GPS) with the objective of achieving a defined coverage requirement with limited energy consumption.

Sensing scheduling is only considered for opportunistic sensing applications since mobile users usually control sensing activities manually in participatory sensing applications.

The movement of a mobile user k can be characterized using a trajectory Γ_k that is a set of 3-tuples (k, t_k, loc_k). Each tuple gives the location of user k at time t_k. The more 3-tuples there are in the trajectory, the more accurately it can characterize user's movement. Usually a mobile user only carries one phone, therefore, we use "smartphone" and "user" interchangeably.

Ideally, a smartphone should send sensor readings along with the corresponding locations obtained from its GPS to a sensing server. However, it is well-known that GPS is very energy-hungry [40] (a GPS device usually consumes much more energy than other sensors). Keeping GPS on during the whole sensing procedure is not feasible since it may drain the battery quickly. Other approaches, such as Google's location application, can also be used to obtain location information from a remote Google server by calling an Android system API, which usually consumes much less power [6], but provides less accuracy compared to GPS. We consider a *GPS-less* system in which each smartphone uses certain location application (such as Google)

to obtain location information without turning on its GPS.

Every 3-tuple (k, t_k, loc_k) in a trajectory can be imagined as a *virtual sensor*. If trajectories of all users are given, then we can have a large network of *virtual sensors* by combining all 3-tuples in trajectories. A sensing schedule S is a collection of virtual sensor sets, i.e., $S = \bigcup_{k=1}^{K} S_k$, where $S_k \subseteq \Gamma_k$. Selecting a virtual sensor (k, t_k, loc_k) into the schedule essentially means that mobile phone k is scheduled to sense at the location loc_k and time t_k. $|S_k| = |S \bigcap \Gamma_k|$ gives the number of times smartphone of user k senses.

Sensing targets are assumed to be a set V of points. If the accurate location of each virtual sensor is known, then it is easy to figure out whether a target point $v_j \in V$ can be covered by a virtual sensor s_i. However since a GPS-less location approach usually cannot provide precise locations, we present a probabilistic model to estimate the probability that a virtual sensor s_i covers a target point v_j with location errors (P_{ij}) and the probability that a target point v_j is covered by a given sensing schedule (P_j), which will be discussed in the next section. Since coverage quality and energy efficiency are the primary design goals, we consider a problem of maximizing sensing coverage subject to sensing energy consumption constraints. We formally define the optimization problem in the following.

Definition 5 (E-MCSS) *Given a set V of J target points, K smartphone users, a sensing deadline T and the moving trajectory Γ_k of each user $1 \leq k \leq K$ before the deadline, the* **Energy constrained Maximum Coverage Sensing Scheduling (E-MCSS)** *problem seeks a sensing schedule $S = \bigcup_{k=1}^{K} S_k$, where $S_k \subseteq \Gamma_k$, $1 \leq k \leq K$, such that the total coverage probability $\sum_{j=1}^{J} P_j$ is maximized subject to the constraint that the total number of sensing times $\sum_{k=1}^{K} |S_k| \leq B$ before the deadline T, where B is a given threshold (which we call the* sensing budget*).*

However, bounding the total number of sensing times only may lead to unfair utilizations of smartphones: some users' phones are heavily used for sensing and

other users' phones are lightly utilized or not utilized at all. Therefore, we try to improve fairness by considering those sensing schedules in which each user's sensing time is bounded. We call such schedules *fair* sensing schedules. We also study the *Fair Maximum Coverage Sensing Scheduling (F-MCSS)* problem which seeks a fair sensing schedule $S = \bigcup_{k=1}^{K} S_k$, where $S_k \subseteq \Gamma_k$ and $|S_k| \leq B_k$, $1 \leq k \leq K$ (B_k is the sensing budget of mobile user k). We will discuss how to set B_k and B in practice in Section 5.3.4.

A probabilistic model for sensing coverage with location errors is used to in our problem formulation.

The Google's location application provides a rough estimation for the location of a smartphone, which is given by a 3-tuple, (x, y, R), where x and y are latitude and longitude of the estimated location respectively and R is the accuracy radius which can be obtained from the MaxMind Accuracy Radius database [48]. With 50% probability, the phone is located inside the *location disk* with origin at (x, y) and a radius of R. The actual location is assumed to follow a two dimensional normal distribution with correlation $\rho_{xy} = 0$.

Suppose that a virtual sensor s_i has a location of (x_i, y_i, R_i) given by the Google's location application. Note that this location may not be accurate and it is called *reported location* in the following. We present a method to calculate P_{ij}, i.e., the probability that a particular target point $v_j \in V$ can be covered by this virtual sensor (i.e, if a smartphone senses at this location, what is the probability that this target point can be covered). First, it is easy to calculate the probability that a mobile user actually shows up at a specific location. If the sensing range is R', the probability that a target v_i can be covered equals the probability that a virtual sensor is located inside the corresponding *target disk* with origin at v_i and a radius of R', which is given by the following equation:

$$P_{ij} = \int_{\phi_1}^{\phi_2} \int_{r_1}^{r_2} P(r) r \mathrm{d}r \mathrm{d}\phi. \tag{5.9}$$

Since the distance between the actual location and the reported location, r, follows a 2-dimensional normal distribution with $\sigma_x = \sigma_y = \sigma$, the probability that the actual location is r away from the reported location can be given by the following equation:

$$P(r) = \frac{1}{2\pi\sigma^2} \exp\{-\frac{r^2}{2\sigma^2}\}. \tag{5.10}$$

As shown in Fig. 5-9, ϕ is the angle between the line with the reported and actual locations and the line with the reported location and the target point. $[r_1, r_2]$ and $[\phi_1, \phi_2]$ are the integration intervals for r and ϕ respectively. As mentioned above, the probability that a virtual sensor s_i is actually inside the location disk with origin at the reported location and a radius of R_i is 50%, therefore, we can obtain σ:

$$\sigma \approx 1.5 R_i. \tag{5.11}$$

As shown in Equation (5.10), $P(r)$ is independent of ϕ, so Equation (5.9) can be simplified to:

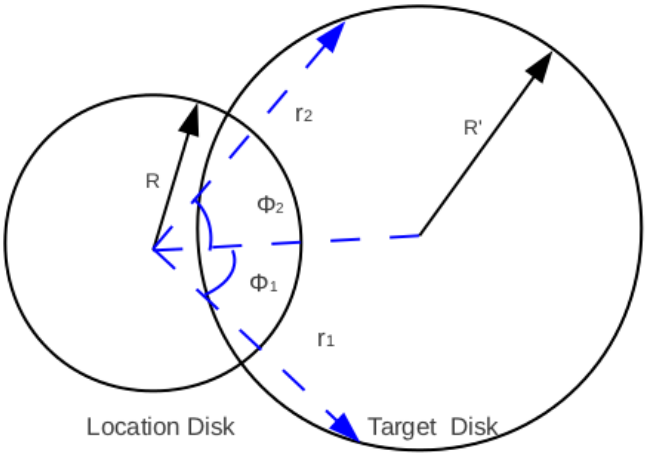
$$P_{ij} = \int_{r_1}^{r_2} P(r)(\phi_2(r) - \phi_1(r)) r \mathrm{d}r. \tag{5.12}$$

We need to consider two cases for the distance d_{ij} between virtual sensor s_i and the target point v_j: 1) $d_{ij} \geq R'$ and 2) $d_{ij} < R'$, which are shown in Fig. 5-9.

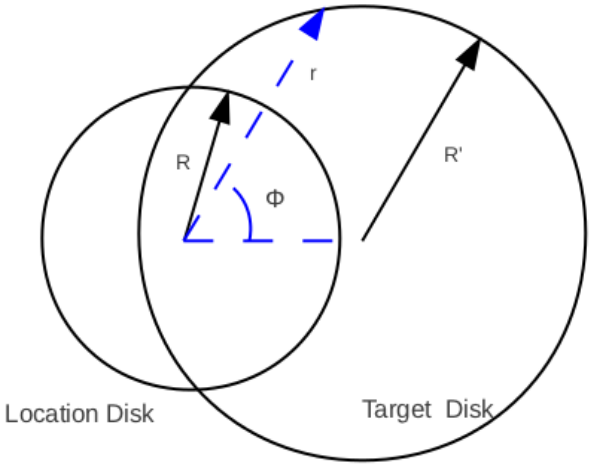
For case 1: $d_{ij} \geq R'$, by using the law of cosine, we can have:

$$\phi_2(r) - \phi_1(r) = 2\cos^{-1}(\frac{r^2 + d_{ij}^2 - R'^2}{2 r d_{ij}}). \tag{5.13}$$

By replacing $P(r)$ and $(\phi_2(r) - \phi_1(r))$ in Equation (5.12) with Equations (5.10) and (5.13), we have:



(a) Case 1: $d_{ij} \geq R'$



(b) Case 2: $d_{ij} < R'$

Figure 5-9: The sensing coverage model

$$P_{ij} = \int_{d_{ij}-R'}^{d_{ij}+R'} \frac{1}{\pi\sigma^2} \exp\{-\frac{r^2}{2\sigma^2}\} \cos^{-1}(\frac{r^2 + d_{ij}^2 - R'^2}{2rd_{ij}}) r\mathrm{d}r. \tag{5.14}$$

For case 2: $d_{ij} < R'$, we have:

$$\phi_2(r) - \phi_1(r) = \begin{cases} 2\pi, & \text{if } 0 \le r \le R' - d_{ij}; \\ 2\cos^{-1}(\frac{r^2+d_{ij}^2-R'^2}{2rd_{ij}}), & \text{if } r > R' - d_{ij}. \end{cases} \tag{5.15}$$

Similarly, we have:

$$\begin{aligned} P_{ij} &= \int_0^{R'-d_{ij}} \frac{1}{\sigma^2} \exp\{-\frac{r^2}{2\sigma^2}\} r \mathrm{d}r \\ &+ \int_{R'-d_{ij}}^{d_{ij}+R'} \frac{1}{\pi\sigma^2} \exp\{-\frac{r^2}{2\sigma^2}\} \cos^{-1}(\frac{r^2+d_{ij}^2-R'^2}{2rd_{ij}}) r \mathrm{d}r \end{aligned} \tag{5.16}$$

Because there does not exist the closed form integration for the functions in Equations (5.14) and (5.16), we use the numerical method to obtain approximate solutions. Note that the number of integration steps were set to $H = 100$ in the simulation and experiment.

For case 1: $d_{ij} \ge R'$, based on Equation (5.14), we can have:

$$\delta = \frac{2R'}{H}; \tag{5.17}$$

$$r = (d_{ij} - R' + h\delta); \tag{5.18}$$

$$P_{ij} = \sum_{h=0}^{h=H} (\frac{r\delta}{\pi\sigma^2} \exp\{-\frac{r^2}{2\sigma^2}\} \cos^{-1}(\frac{d_{ij}^2+r^2-R'^2}{2d_{ij}r})). \tag{5.19}$$

For case 2: $d_{ij} < R'$, based on Equation (5.16), we can have:

$$\delta = \frac{d_{ij}+R'}{H}; \tag{5.20}$$

$$r = h\delta; \tag{5.21}$$

$$P_{ij} = \sum_{h=0}^{h=\lfloor \frac{R'-d_{ij}}{\delta} \rfloor} \frac{r\delta}{\sigma^2} \exp\{-\frac{r^2}{2\sigma^2}\} \tag{5.22}$$

$$+ \sum_{h=\lceil \frac{R'-d_{ij}}{\delta} \rceil}^{h=H} \frac{r\delta}{\pi\sigma^2} \exp\{-\frac{r^2}{2\sigma^2}\} \cos^{-1}(\frac{d_{ij}^2+r^2-R'^2}{2d_{ij}r}). \tag{5.23}$$

So given a set of virtual sensor S, the probability that a target point j can be

covered (by some virtual sensor(s) in S) is given as follows:

$$P_j = (1 - \prod_{s_i \in S} (1 - P_{ij})). \tag{5.24}$$

5.3.2 Approximation Algorithms

Assuming knowing the trajectory of each mobile user in advance, we present constant factor approximation algorithms to solve the E-MCSS problem and the F-MCSS problem, respectively. It may be argued that this assumption is not realistic since it is hard to precisely predict how users will move in the future. However, the provably-good solutions given by these algorithms can be used to show sensing coverage that can potentially be achieved by collaborative sensing and to serve as benchmarks for performance evaluation.

The E-MCSS problem can be formally formulated as follows:

$$\max \sum_{j=1}^{J} (1 - \prod_{s_i \in S} (1 - P_{ij})) \tag{5.25}$$

Subject to:

$$|S| \leq B. \tag{5.26}$$

Basically, the problem is to maximize the total sensing coverage probability by selecting a subset S of virtual sensors with a cardinality no more than the given budget B. Mathematically, the E-MCSS problem has the same objective function as that of the Budget Server Problem with a Uniform Cost (BSP-UC) studied in [74]. It has been shown in [74] that this objective function is a non-negative, monotone and submodular function. In a well-known work [54], Nemhauser et al. proved that,

for such a problem, a simple greedy algorithm guarantees a solution with value at least $(1-\frac{1}{e}) > 0.63$ of the optimal. We present the greedy algorithm in the following, which is a $(1-\frac{1}{e})$-approximation algorithm for the E-MCSS problem.

Algorithm 8 The greedy algorithm for the E-MCSS problem

1: $S_0 = \emptyset$;
2: **for** $h = 1 \cdots B$ **do**
3: $\quad s := \text{argmax}_{s' \in \Gamma \setminus S_{h-1}} f(S_{h-1} \bigcup \{s'\}) - f(S_{h-1})$;
4:
5: $\quad S_h := S_{h-1} \bigcup \{s\}$;
6:
7: **end for**
8: **return** S_B;

In this algorithm, $f(\cdot)$ is the objective function, i.e., $f(S) = \sum_{j=1}^{J} (1 - \prod_{s_i \in S}(1 - P_{ij}))$. This algorithm starts with an empty set and adds a virtual sensor that maximizes the incremental objective value in each iteration. The running time of this algorithm is $O(B|\Gamma|)$, where $\Gamma = \bigcup_{k=1}^{K} \Gamma_k$ is the ground set of virtual sensors.

Next, we consider the F-MCSS problem and present the problem formulation in the following.

$$\max \sum_{j=1}^{J} (1 - \prod_{s_i \in S} (1 - P_{ij}))$$

Subject to:

$$|S \bigcap \Gamma_k| \leq B_k, 1 \leq k \leq K. \tag{5.27}$$

We construct $\Omega = \{S_l : S_l \subseteq \Gamma, |S_l \bigcap \Gamma_k| \leq B_k, 1 \leq k \leq K\}$, which is a collection of subsets of the ground set Γ. We assume $\emptyset \in \Omega$. Next, we show that (Γ, Ω) is a

matroid.

Definition 6 (Matroid [21]) *A pair $(U, \mathcal{Z})$ consisting of a ground set U and a collection $\mathcal{Z}$ of subsets of U is a matroid if:*

1) If $X \in \mathcal{Z}$ and $Y \subset X$, then $Y \in \mathcal{Z}$;

2) for all $X, Y \in \mathcal{Z}$, if $|X| > |Y|$ then there exists some $x \in X \setminus Y$ such that $Y \bigcup \{x\} \in \mathcal{Z}$.

Lemma 1 (Γ, Ω) *is a matroid.*

Suppose that $S_{l_1} \in \Omega$. According to the definition of Ω, S_{l_1} satisfies the constraint $|S_{l_1} \bigcap \Gamma_k| \leq B_k, 1 \leq k \leq K$. And if $S_{l_2} \subset S_{l_1}$ then we have $|S_{l_2} \bigcap \Gamma_k| \leq |S_{l_1} \bigcap \Gamma_k| \leq B_k, 1 \leq k \leq K$. So $S_{l_2} \in \Omega$.

We prove that condition 2) is also satisfied by contradiction. Suppose that $S_{l_1} \in \Omega, S_{l_2} \in \Omega$, and $|S_{l_1}| > |S_{l_2}|$, but there does not even exist any element x such that $x \in S_{l_1} \setminus S_{l_2}$ and $S_{l_2} \bigcup \{x\} \in \Omega$. If this statement is not true, then $S_{l_2} \bigcup \{x_k\} > B_k, \forall x_k \in \{S_{l_1} \setminus S_{l_2}\} \bigcap \Gamma_k, 1 \leq k \leq K$. This means for any k, if $\{S_{l_1} \setminus S_{l_2}\} \bigcap \Gamma_k \neq \emptyset$, then $|S_{l_2} \bigcap \Gamma_k| = B_k$. So $|S_{l_1} \bigcap \Gamma_k| \leq |S_{l_2} \bigcap \Gamma_k|, \forall \{S_{l_1} \setminus S_{l_2}\} \bigcap \Gamma_k \neq \emptyset, 1 \leq k \leq K$. And since $\{S_{l_1} \setminus S_{l_2}\} \subset S_{l_1}$, and all the elements in $S_{l_1} \bigcap S_{l_2}$ are shared by both S_{l_1} and S_{l_2}, $|S_{l_1}| \leq |S_{l_2}|$, which is in contradiction to our assumption. This completes our proof.

The F-MCSS problem can be re-formulated as follows:

$$\max_{S \in \Omega} \sum_{j=1}^{J} (1 - \prod_{s_i \in S} (1 - P_{ij})). \tag{5.28}$$

Therefore the F-MCSS problem belongs to a class of problems of maximizing a submodular set function over a matroid [21]. We present a greedy algorithm to solve it and we have the following theorem.

Algorithm 9 The greedy algorithm for the F-MCSS problem

1: $S_0 := \emptyset$; $h := 1$;
2: **while do**$\exists s \in \Gamma \setminus S_{h-1}$ s.t. $S_{h-1} \bigcup \{s\} \in \Omega$
3:
4: $s := \text{argmax}_{s' \in S \setminus S_{h-1}} f(S_{h-1} \bigcup \{s'\}) - f(S_{h-1})$;
5:
6: $S_h := S_{h-1} \bigcup \{s\}$;
7:
8: $h := h + 1$;
9:
10: **end while**
11: **return** S_h;

Theorem 3 *Algorithm 9 is a $\frac{1}{2}$-approximation algorithm for the F-MCSS problem and has a time complexity of $O(N^2)$.*

Proof 2 *In [21], it is shown that a simple greedy algorithm gives a $1/2$ approximation for an optimization problem of maximizing a non-decreasing submodular set function over a matroid. The idea of Algorithm 9 is the same as the greedy algorithm presented in [21], therefore, according to Lemma 1, Algorithm 9 is a $\frac{1}{2}$-approximation algorithm for the F-MCSS problem. The running time is $O(|\Gamma|^2 \cdot p(|\Gamma|))$, where $p(|\Gamma|)$ is the running time for testing whether $S_{h-1} \bigcup \{s\} \in \Omega$ or not. In our case, this testing can be easily done in constant time by maintaining a counter for each user and checking if its value exceeds the given budget.*

5.3.3 Scheduling Protocol

In this subsection, we present the protocol of GPS-less sensing scheduling, which leverages the algorithms described above for sensing scheduling. As mentioned above, both algorithms need to know the trajectories of all mobile phones in the whole sensing period beforehand, which is not quite possible in practice. Therefore, we need to find a practical way to apply the proposed sensing scheduling algorithms.

Fig. 5-10 illustrates how the proposed protocol works. Each smartphone peri-

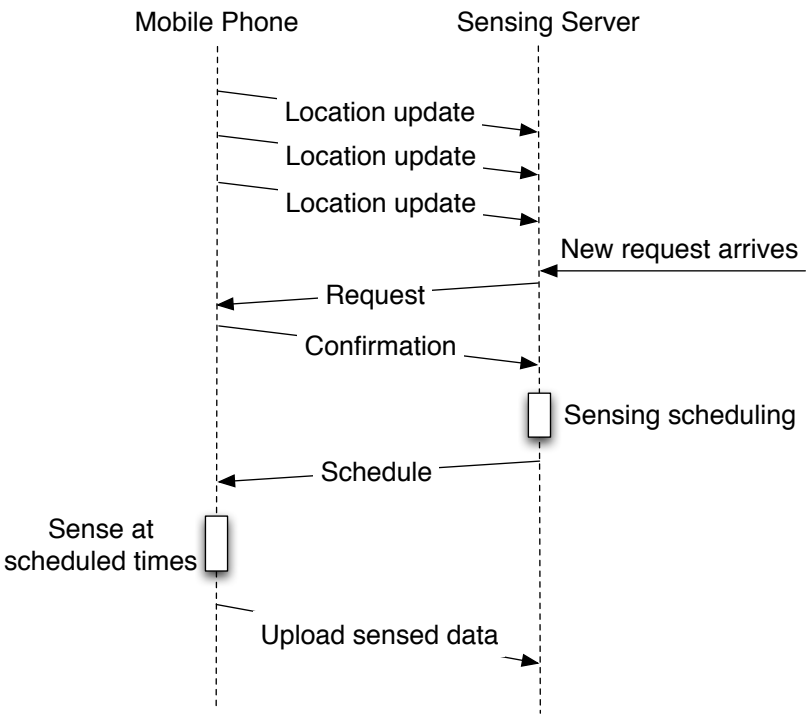


Figure 5-10: The proposed sensing scheduling protocol

odically reports its location (obtained from the Google's location application) to a sensing server. Note that the location updating period was set to 20 seconds in our field test. The sensing server keeps track of recent locations for each smartphone (currently, our system keeps 10 most recent locations for each phone). When a new sensing request (that specifies what to sense and area of interest) arrives, the sensing server will push the request to smartphones that happen to be in the area of interest defined by the request and wait for confirmations from them. Then the sensing server will apply a sensing scheduling algorithm to schedule sensing activities of a set of smartphones that confirm their willingness to participate, and instruct them to sense by sending them a schedule that specifies when to sense. Each phone will then use proper sensors to sense, encapsulate sensed data in an HTTP message and forward it to the sensing server. Any algorithm can be applied here for sensing scheduling. Of course, we employ the two algorithms presented above in our system. However, in order to use them, we need to predict the moving trajectory for each phone in the

sensing period defined by the request, which is hard if the period is too long. In our protocol, we divide the whole sensing period into timeslots with the same duration (which was set to 200sec in the simulation and field test) and schedule sensing activities at the beginning of each timeslot. Specifically, during each timeslot, the sensing server performs the following actions:

1) Uses a mobility prediction algorithm to predict the moving trajectory of each smartphone.

2) Feeds the predicted trajectories to a scheduling algorithm to obtain a schedule and informs each smartphone.

3) Collects sensed data from smartphones.

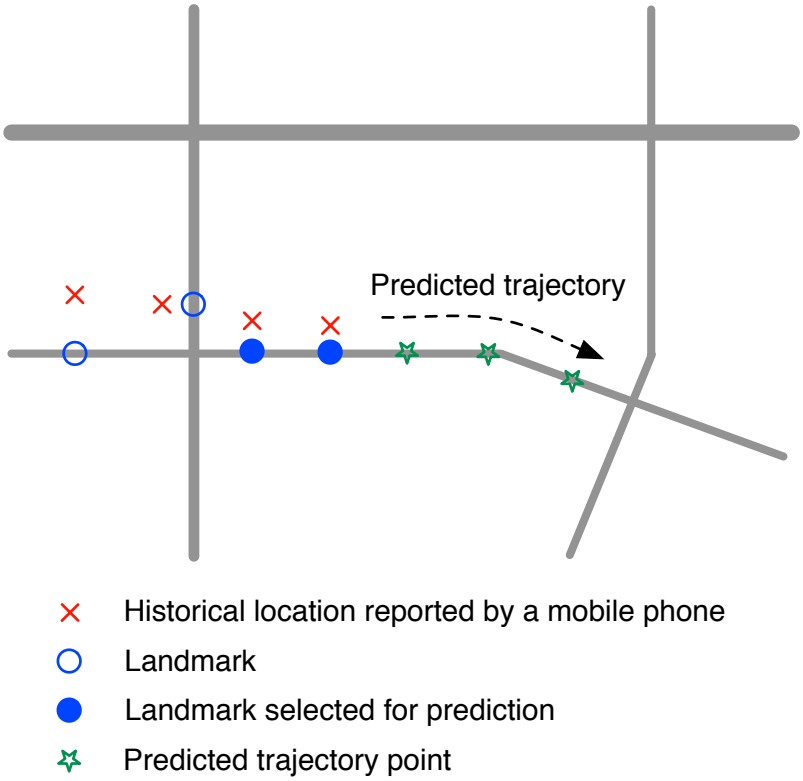


Figure 5-11: The mobility prediction algorithm

Essentially, any mobility prediction algorithm can be applied here. We design a simple and effective algorithm for prediction and use it in our system. For prediction,

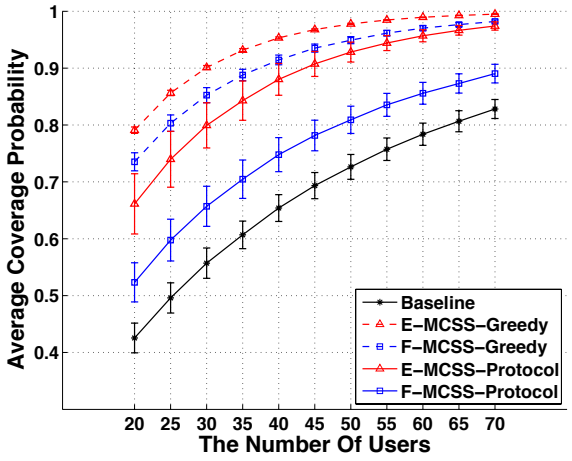
we assume that each mobile user only moves along a roadway and only turns in an intersection. In our prediction algorithm, we use a set of evenly distributed landmarks to characterize a roadway. Currently, the distance between a pair of consecutive landmarks was set to one meter. The historical locations reported by smartphones (which might not be on roadways because they are not accurate) are mapped to the closest landmarks on the roadways. Then the prediction algorithm checks all the landmarks corresponding to recent reported locations and selects most recent two which happen to belong to the same roadway. In Fig. 5-11, the two solid circles in the center are the two selected landmarks. Using these two landmarks, the algorithm calculates the speed and direction for this mobile user. The future locations can thus be predicted by assuming the mobile user will not change his speed and direction within this timeslot. Based on the prediction, the algorithm will then generate the future moving trajectory (as shown by stars in the figure), which stops if a predefined number (which was set to 10 in our simulation and field test), or an intersection is reached. The algorithm makes no attempt to predict the trajectory after an intersection, since it cannot know to which direction the user will head.

5.3.4 Validation And Performance Evaluation

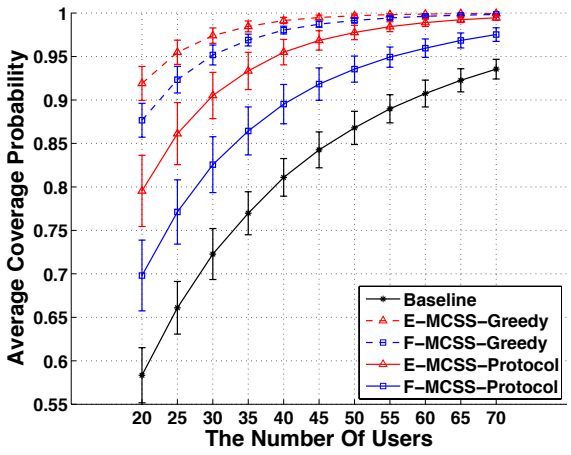
In this subsection, we present simulation results as well as experimental results to validate and justify effectiveness of the proposed scheduling algorithms and protocol.

Simulation Results

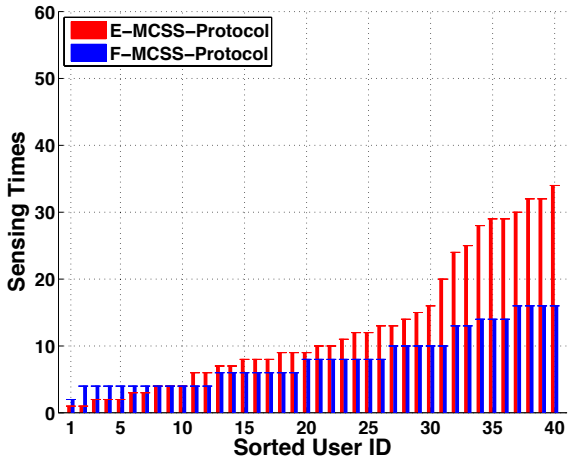
In the simulation, the target area we chose is an area located in the center of Manhattan, NY, which spans 3 blocks from west to east with a total length of 1600m and 3 blocks from south to north with a total length of 800m, and includes the 5th, 6th, 7th, 8th Avenues and the 44th, 45th, 46th, 47th Streets. We used a mobility model similar to the well-known Manhattan model [3] to generate mobile users' moving tra-



(a) Average coverage probability VS. the number of users ($b = 2$)



(b) Average coverage probability VS. the number of users ($b = 4$)



(c) Sensing times vs sorted user ID ($b = 2$)

Figure 5-12: Performance of the proposed algorithms and protocol

jectories. Specifically, each user is assumed to be at a random location in the target region at the very beginning. Then it randomly selects a direction and moves with a speed randomly selected from $\{2, 4\}$ meters per second towards an intersection. Then the user moves straight ahead with a probability of 50%, turns left or right with equal probabilities (i.e., 25%). The trajectory of each user was constructed with points on critical locations (such as intersections and road heads) and evenly spaced sample points (1 meter between two consecutive ones) between them. All the location data were collected from the Google Maps using its APIs. The deadline (the duration

of the whole sensing period) was set to $T = 1800$s. The sensing range was set to $R' = 20$m. The target points were evenly distributed on the roadways in the target area with a distance of 40m between a pair of consecutive points. Random errors were introduced to simulate the inaccuracy of reported locations.

In the simulation, we evaluated the performance of the greedy algorithm for E-MCSS (E-MCSS-Greedy), the greedy algorithm for F-MCSS (F-MCSS-Greedy), the protocol using E-MCSS-Greedy (E-MCSS-Protocol) and the protocol using F-MCSS-Greedy (F-MCSS-Protocol). We used the average coverage probability (i.e., the total coverage probability divided by the number of target points) as the performance metric and changed the number of mobile users from 20 to 70, with a step size of 5. In the protocol, the whole sensing period was divided into multiple timeslots with the same duration, which was set to 200s in the simulation. $T = 1800$s, but the first timeslot was used only for collecting user's mobility information for prediction. Therefore, there are 8 timeslots for sensing. For F-MCSS-Protocol, we set each user's sensing budget in each timeslot to $b = 2$ and $b = 4$ respectively in two scenarios. To ensure fair comparisons, for F-MCSS-Greedy, each user's total sensing budget was set to $B_i = b * 8$, i.e., 16 and 32 respectively. Accordingly, the total sensing budget was set to $B = B_i * K$ for E-MCSS-Greedy and the total sensing budget in each timeslot for E-MCSS-Protocol was set to $b * K$, where K is the number of users. Moreover, we used the periodic sensing method (i.e., every smartphone periodically senses without collaborations) as the baseline for comparison, in which each smartphone senses every 100s. Note that during the sensing period, some mobile users might leave the target area at certain times, which were not used for sensing any longer after their departure in the simulation.

We conducted simulation runs on 40 sets of randomly generated trajectories. The simulation results are shown in Figs. 5-12(a)– 5-12(c). We can make the following observations:

1) As expected, all the proposed approaches outperform the baseline method and E-MCSS-Greedy performs best in terms of coverage probability. On average (based on Figs. 5-12(a) and 5-12(b)), in terms of the coverage probability, E-MCSS-Protocol and F-MCSS-Protocol outperform the baseline method (for the case where $b = 2$) by 33.3% and 15.5% respectively. We can also see that the performance of these two methods is comparable. In addition, we can view their performance from another perspective. In order to achieve an average coverage probability of 80%, according to Fig. 5-12(a), E-MCSS-Protocol needs about 30 users and F-MCSS-Protocol needs about 50 users, while the baseline method needs about 65 users.

2) No matter which method is used, the average coverage probability always increases with the number of users. In addition, by comparing Figs 5-12(a) and 5-12(b), we can see that the higher the sensing budget (which leads to more energy consumption), the better the coverage.

3) In Fig. 5-12(c), we show fairness of E-MCSS-Protocol and F-MCSS-Protocol by plotting the number of sensing times of each user (the total number of users is 40) using a set of randomly generated trajectories. We can see that compared to E-MCSS-Protocol, F-MCSS-Protocol offers more even distribution of the number of sensing times among mobile users, which means better fairness. Note that the total number of sensing times given by E-MCSS-Protocol is larger than that given by F-MCSS-Protocol due to the early departure of some mobile users.

Experimental Results

The field test was performed on Syracuse University's campus, in which 8 volunteers participated. The application is to scan WiFi signal strengths on roadways. Popular Android smartphones, including Motorola Droid, Motorola Razr, Google Nexus S, Samsung Galaxy I9000 and Galaxy S2 were used as our sensing devices. Each smartphone used Google's location application to obtain its locations and report them to

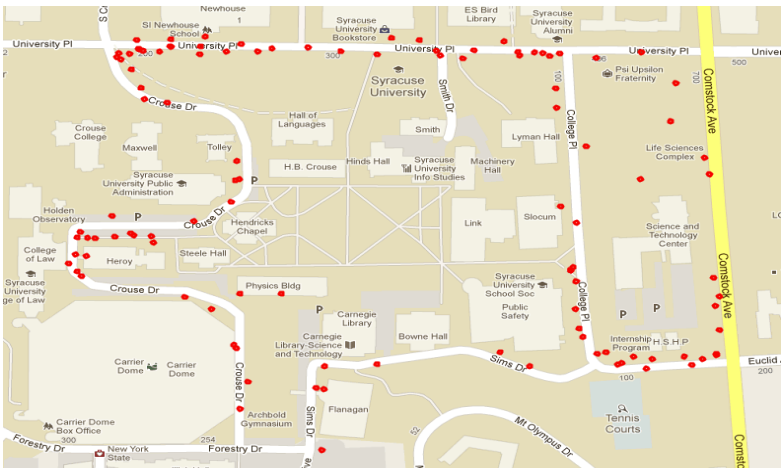
Table 5.3: Experimental settings

Variables	Description
$K = 8$	The number of users
$T = 1800$s	The duration of the whole sensing period
$T' = 200$s	The duration of a timeslot
$T_{loc} = 20$s	Location reporting period
$b = 2$	The sensing budget for each user in a timeslot
$T_p = 100$s	the time interval for the baseline method

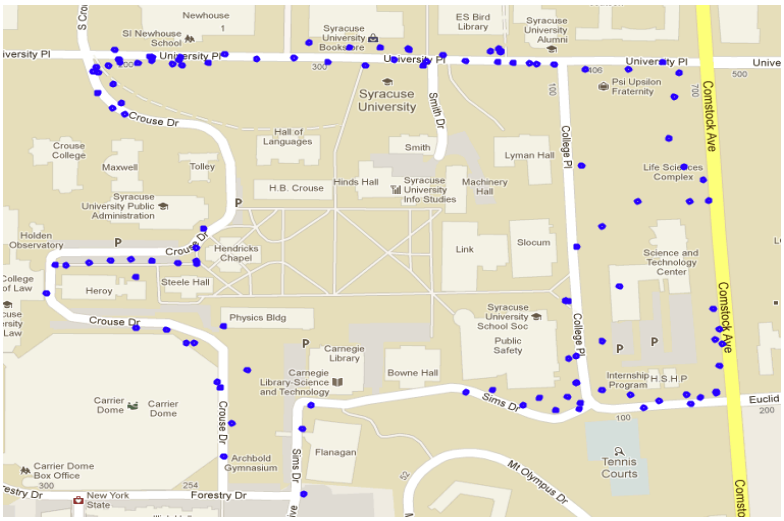
the sensing server every $T_{loc} = 20$s. *For the testing purpose*, GPS was also turned on to collect the *actual* locations where smartphones performed WiFi scans.

At the very beginning of our field test, each user was randomly located on a roadway in the target area. Then he/she started to walk along the roadway in a random direction at his/her regular walking speed and turned randomly in an intersection. The settings of the field experiment are summarized in Table 5.3.

We tested the proposed protocol and system in the field experiment. The sensing budgets were set in the same way as simulations. Note that in the first timeslot, each smartphone simply reported its locations obtained from the Google's location application to the sensing server for mobility prediction without performing any WiFi scans. We plot the users' actual sensing locations (obtained from GPS) given by the proposed protocol in Fig. 5-13. From the figure, we can see that the locations that smartphones sense are widely distributed over the roadways. In this way, the target points in the area are well covered, which verifies that the proposed system and protocol work well in a real environment with random mobility and unreliable wireless links.



(a) E-MCSS-Protocol



(b) F-MCSS-Protocol

Figure 5-13: Actual sensing locations given by the proposed protocol

Chapter 6

Conclusions

6.1 Conclusions

In this thesis, we studied the topic of smartphone sensing and identified challenges of building a unified sensing platform. First, this thesis designs and implements a unified sensing platform for smartphone sensing.

In chapter 2, we overview the system design need of a unified and green platform for smartphone sensing, which can support various sensing applications and employs energy-efficient algorithms for sensing scheduling. The technical details of the design are presented. By integrating the script interpreter into the unified platform, platform can interpret and fulfill the sensing tasks at runtime. Modular design makes the backend easy to be configured and adjusted. The design and implementation details of both mobile frontend and cloud backend are presented. The proposed sensing platform is general enough to support various sensing applications (requires different set of sensors). Also the platform is easy to be extended to support newly developed sensor technologies: only a provider module is needed to be developed and registered in the platform. Multiple sensing applications could run on the sensing platform concurrently. And sensors' data could be shared among these applications to save

energy.

Then, two sensing applications are built on top of the sensing platform: **SOR**(Sensing base Objective Ranking application) and **LIPS**(LIfestyle learning via smartPhone Sensing): **1)** In chapter 3, we presented design, implementation and evaluation of SOR. SOR is easy to use and its architecture is so scalable that various sensors can be easily integrated into it. We presented an online scheduling algorithm for coverage maximization, which has a constant approximation ratio of 1/2; moreover, we presented a personalizable ranking algorithm, which ranks target places based on various sensor readings and user preferences. Both of them have been used in SOR for scheduling and ranking respectively. We validated and evaluated SOR via both field tests (using real hiking trails and coffee shops in Syracuse as target places) and simulation. Field-testing results showed that data collected and processed by SOR can well capture characteristics of target places, and personalizable rankings produced by SOR can well match user preferences. In addition, simulation results showed that the proposed scheduling algorithm outperforms a baseline algorithm by 65% in terms of average coverage probability. **2)** In chapter 4, we presented design, implementation and evaluation of LIPS application. First, we presented the workflow and architecture of LIPS application. We proposed a hybrid scheme for lifestyle learning, which consists of two parts: characterization and prediction. Specifically, we presented a two-stage algorithm to characterize the lifestyle of a mobile user using PoIs, which leverages the DBSCAN and MeanShift algorithms for coarse-grained and fine-grained clustering in the first and second stages respectively. Based on discovered PoIs, we developed a method to build a model to predict his/her future activities using SVM. In addition, we presented a lifestyle-aware adaptive sampling algorithm for improving energy efficiency. We implemented the proposed system and algorithms based on the Android platform. We have validated and evaluated LIPS via extensive field tests carried out in 6 major cities of USA. The experimental results showed that LIPS can

1) well discovers PoIs of mobile users, 2) precisely predict their future activities with an average accuracy of 72%, and 3) achieve a significant energy saving of 52% on average (compared to periodic sampling).

This thesis also studies how to reduce sensing energy consumption for sensing application. In chapter 5, we proposed to leverage cloud-assisted collaborative sensing to reduce energy consumption for smartphone sensing applications. First, By assuming the moving trajectory of each mobile user is known in advance, we presented a polynomial-time algorithm to obtain minimum energy sensing schedules, which can be used to show potential energy savings that can be brought by collaborative sensing and can serve as a benchmark for performance evaluation. We also presented an algorithm to achieve a good tradeoff between total energy consumption and fairness. Under realistic assumptions, we presented two practical and effective heuristic algorithms: the prediction-based algorithm and the function-based algorithm. It has been shown by simulation results based on real energy consumption and location data that compared to traditional sensing without collaborations, collaborative sensing significantly reduces energy consumption, and the proposed function-based algorithm performs well in terms of both total energy consumption and fairness. In the second part of chapter 5, we presented a probabilistic model for sensing coverage without accurate location information, based on which we formally define the E-MCSS problem and the F-MCSS problem. Under a strong assumption that the moving trajectories of mobile users are known in advance, we presented a $(1 - \frac{1}{e})$-approximation algorithm and a $\frac{1}{2}$-approximation algorithm to solve the E-MCSS and F-MCSS problems in polynomial time respectively. Under realistic assumptions, we presented a GPS-less and energy-efficient protocol for sensing scheduling based on the proposed approximation algorithms. It has been shown by simulation results that the proposed protocol significantly outperforms a baseline method in terms of coverage probability and F-MCSS-Protocol (the protocol using the greedy algorithm

for F-MCSS) and achieve a good tradeoff between coverage and fairness (on individual smartphone usages). Experimental results from a field test were also presented to validate the proposed protocol.

6.2 Future Research Directions

In this section, we point out future research directions:

Sensing Task Scheduling on a Smartphone: A sensing task will be assigned to multiple smartphones. Correspondingly, a smartphone may be used to process multiple sensing tasks. Hence, sensing task scheduling algorithms are also needed to schedule multiple sensing tasks on a smartphone. The following optimization problem needs to be addressed: given a set of sensing tasks (on a smartphone), each with certain temporal requirement (i.e., must be completed at a particular time or during a certain period), spatial requirement (i.e., must be performed at a particular location or in a certain area), or both, find a schedule with minimum energy consumption for performing these tasks such that the given requirements are met. To the best of our knowledge, this problem has not been studied yet. One trivial solution is to treat each sensing task as an independent task and handle them one by one. However, this may not be energy-efficient because multiple sensing tasks may share one or multiple sensing actions (e.g., request location information from GPS). The best way may be to group multiple correlated tasks together by exploiting the temporal-spatial correlations between them, schedule sensing actions associated with them and determine when to conduct common sensing actions based on user mobility status with the objective of minimizing energy consumption and satisfying the temporal and spatial requirements.

Privacy-preserving Incentive Mechanisms: Incentive and privacy have been addressed separately in the context of mobile sensing but has not been considered

simultaneously. For example, in the incentive mechanisms designed in [75], privacy issues were not considered at all. It is important to design incentive mechanisms that can enhance user privacy. Having privacy protection will encourage more mobile users to participate in sensing activities. Initial work was done in a very recent paper [39]. In that paper, the authors proposed two privacy-aware incentive schemes, which allow each mobile user to earn credits by contributing data without leaking which data it has contributed, and in the meanwhile ensure that dishonest users cannot abuse the system to earn unlimited amount of credits. The first scheme considers scenarios where a Trusted Third Party (TTP) is available. It relies on the TTP to protect user privacy, and thus has very low computation and storage cost at each mobile user. The second scheme removes the assumption of TTP and applies blind signature and commitment techniques to protect user privacy. We believe this line of research can make a significant impact on smartphone sensing.

A Reputation System of Mobile Users: It is very important to establish a common reputation system of mobile users for various smartphone sensing applications. In the current incentive mechanisms [75], mobile users are selected purely based on their bids. It would be interesting to study user selection schemes based on the reputation of individual users. Moreover, users' reputations can be used to enhance the reliability of sensed data provided by mobile users. For example, mobile users' reputations can be used as weights for generating the final sensing results. A mobile user in an cloud may be involved in various applications. A unified and fair approach needs to be developed to adjust the reputation score(s) of a mobile user based on his/her performance in different applications in terms of various metrics (efficiency, quality of sensed data, etc), which is very challenging but has not yet been studied.

Smartphone Sensing based Social Networking: Social networks have been making a significant impact on people's life. Marrying smartphone sensing with social networking can benefit both systems. On one hand, a popular social networking

system, such as Facebook, Twitter and Weibo, serves as a perfect platform for sharing data collected via smartphone sensing; on the other hand, mobile phone sensing can substantially enrich social networking activities by providing various context information of mobile users, such as location, moving states, etc. The CenceMe [50] represents the first system that combines the inference of the presence of individuals (e.g., walking, in conversation, at the gym) via smartphone sensing with sharing of this information through social networking systems such as Facebook and MySpace. In [13], the authors designed and implemented a crowd-sourced sensing and collaboration system over Twitter, and demonstrated their system using two applications: a crowd-sourced weather radar, and a participatory noise-mapping application. In [58], a smartphone sensing based platform, SociableSense, was developed to capture user behavior in office environments, while providing users with a quantitative measure of their sociability and that of colleagues. It will be very interesting to develop new social networking applications based on smartphone sensing.

Bibliography

[1] Recognizing the user's current activity. `http://developer.android.com/training/location/activity-recognition.html`. accessed: 2014-08-20.

[2] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. *Network flows: theory, algorithms, and applications*. Prentice hall, 1993.

[3] Bai, F., Sadagopan, N., and Helmy, A. The important framework for analyzing the impact of mobility on performance of routing protocols for adhoc networks. *Ad Hoc Networks 1*, 4 (2003), 383–403.

[4] Bazaraa, M. S., Jarvis, J. J., and Sherali, H. D. *Linear programming and network flows*. John Wiley & Sons, 2011.

[5] Bing maps api. `http://www.microsoft.com/maps/choose-your-bing-maps-API.aspx`. accessed: 2014-08-20.

[6] Carroll, A., and Heiser, G. An analysis of power consumption in a smartphone. In *USENIX annual technical conference* (2010), pp. 271–285.

[7] Clark reservation. `http://nysparks.com/parks/126/details.aspx`. accessed: 2014-08-20.

[8] Comaniciu, D., and Meer, P. Mean shift: A robust approach toward feature space analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 24*, 5 (2002), 603–619.

[9] Consolvo, S., McDonald, D. W., Toscos, T., Chen, M. Y., Froehlich, J., Harrison, B., Klasnja, P., LaMarca, A., LeGrand, L., Libby, R., et al. Activity sensing in the wild: a field trial of ubifit garden. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2008), ACM, pp. 1797–1806.

[10] Constandache, I., Choudhury, R. R., and Rhee, I. Towards mobile phone localization without war-driving. In *Proceedings of IEEE International Conference on Computer Communication (INFOCOM'2010)* (2010), pp. 1–9.

[11] Cornelius, C., Kapadia, A., Kotz, D., Peebles, D., Shin, M., and Triandopoulos, N. Anonysense: privacy-aware people-centric sensing. In *Proceedings of the 6th international conference on Mobile systems, applications, and services* (2008), ACM, pp. 211–224.

[12] Das, T., Mohan, P., Padmanabhan, V. N., Ramjee, R., and Sharma, A. Prism: platform for remote sensing using smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services* (2010), ACM, pp. 63–76.

[13] Demirbas, M., Bayir, M. A., Akcora, C. G., Yilmaz, Y. S., and Ferhatosmanoglu, H. Crowd-sourced sensing and collaboration using twitter. In *World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a* (2010), IEEE, pp. 1–9.

[14] Denning, T., Andrew, A., Chaudhri, R., Hartung, C., Lester, J., Borriello, G., and Duncan, G. Balance: towards a usable pervasive wellness application with accurate activity inference. In *Proceedings of the 10th workshop on Mobile Computing Systems and Applications* (2009), ACM, p. 5.

[15] Diaconis, P., and Graham, R. L. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society. Series B (Methodological)* (1977), 262–268.

[16] Django web framework. `https://www.djangoproject.com/`. accessed: 2014-08-20.

[17] Dwork, C., Kumar, R., Naor, M., and Sivakumar, D. Rank aggregation methods for the web. In *Proceedings of the 10th international conference on World Wide Web* (2001), ACM, pp. 613–622.

[18] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd* (1996), vol. 96, pp. 226–231.

[19] Fitbit products. `https://www.fitbit.com`. accessed: 2014-07-30.

[20] Gaonkar, S., Li, J., Choudhury, R. R., Cox, L., and Schmidt, A. Micro-blog: sharing and querying content through mobile phones and social participation. In *Proceedings of the 6th international conference on Mobile systems, applications, and services* (2008), ACM, pp. 174–186.

[21] Gargano, L., and Hammar, M. A note on submodular set cover on matroids. *Discrete Mathematics 309*, 18 (2009), 5739–5744.

[22] Giannotti, F., Nanni, M., Pinelli, F., and Pedreschi, D. Trajectory pattern mining. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining* (2007), ACM, pp. 330–339.

[23] Google glass. `http://www.google.com/glass/start/`. accessed: 2014-08-20.

[24] Google map api. `https://developers.google.com/places/`. accessed: 2014-08-20.

[25] Google places api. https://developers.google.com/places/documentation/. accessed: 2014-08-20.

[26] Green lake state park. http://nysparks.com/parks/172. accessed: 2014-08-20.

[27] Han, J., and Kamber, M. *Data Mining, Southeast Asia Edition: Concepts and Techniques*. Morgan kaufmann, 2006.

[28] Here maps api. https://developer.here.com/. accessed: 2014-08-20.

[29] Hull, B., Bychkovsky, V., Zhang, Y., Chen, K., Goraczko, M., Miu, A., Shih, E., Balakrishnan, H., and Madden, S. Cartel: a distributed mobile sensor computing system. In *Proceedings of the 4th international conference on Embedded networked sensor systems* (2006), ACM, pp. 125–138.

[30] Apple iphone. https://www.apple.com/iphone/. accessed: 2014-07-30.

[31] Jawbone up. https://jawbone.com/up. accessed: 2014-07-30.

[32] Kemeny, J. G. Mathematics without numbers. *Daedalus 88*, 4 (1959), 577–591.

[33] Kemeny, J. G., and Snell, J. L. *Mathematical models in the social sciences*, vol. 9. Ginn Boston, 1962.

[34] Kohavi, R., et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. *IJCAI 14*, 2 (1995), 1137–1145.

[35] Lane, N., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T., and Campbell, A. A survey of mobile phone sensing. *IEEE Communications Magazine 48* (2010), 140–150.

[36] Lee, H. J., Lee, S. H., Ha, K.-S., Jang, H. C., Chung, W.-Y., Kim, J. Y., Chang, Y.-S., and Yoo, D. H. Ubiquitous healthcare service using

zigbee and mobile phone for elderly patients. *International journal of medical informatics 78*, 3 (2009), 193–198.

[37] Lee, J.-G., Han, J., Li, X., and Gonzalez, H. Traclass: trajectory classification using hierarchical region-based and trajectory-based clustering. *Proceedings of the VLDB Endowment 1*, 1 (2008), 1081–1094.

[38] Leijdekkers, P., and Gay, V. A self-test to detect a heart attack using a mobile phone and wearable sensors. In *Computer-Based Medical Systems, 2008. CBMS'08. 21st IEEE International Symposium on* (2008), IEEE, pp. 93–98.

[39] Li, L., Zhao, X., and Xue, G. Unobservable re-authentication for smartphones. In *NDSS* (2013).

[40] Lin, K., Kansal, A., Lymberopoulos, D., and Zhao, F. Energy-accuracy trade-off for continuous mobile device location. In *ACM MobiSys* (2010), pp. 285–297.

[41] Lu, H., Lane, N. D., Eisenman, S. B., and Campbell, A. T. Bubble-sensing: Binding sensing tasks to the physical world. *Pervasive and Mobile Computing 6*, 1 (2010), 58–71.

[42] Lu, H., Pan, W., Lane, N. D., Choudhury, T., and Campbell, A. T. Soundsense: scalable sound sensing for people-centric applications on mobile phones. In *Proceedings of ACM international conference on Mobile systems, applications, and services (MobiSys'2009)* (2009), pp. 165–178.

[43] Lu, H., Yang, J., Liu, Z., Lane, N. D., Choudhury, T., and Campbell, A. T. The jigsaw continuous sensing engine for mobile phone applications. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems* (2010), ACM, pp. 71–84.

[44] Lua. http://www.lua.org. accessed: 2014-07-30.

[45] Madhani, S., Tauil, M., and Zhang, T. Collaborative sensing using uncontrolled mobile devices. In *Collaborative Computing: Networking, Applications and Worksharing, 2005 International Conference on* (2005), IEEE, pp. 8–pp.

[46] Maisonneuve, N., Stevens, M., Niessen, M. E., and Steels, L. Noisetube: Measuring and mapping noise pollution with mobile phones. In *Information Technologies in Environmental Engineering*. Springer, 2009, pp. 215–228.

[47] Mapquest apis. http://developer.mapquest.com/. accessed: 2014-08-20.

[48] Maxmind. http://www.maxmind.com/app/geolite_city_accuracy. accessed: 2014-08-20.

[49] Miluzzo, E., Lane, N., Fodor, K., Peterson, R., Lu, H., Musolesi, M., Eisenman, S., Zheng, X., and Campbell, A. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In *Proceedings of ACM Conference on Embedded Network Sensor Systems (SenSys'2008)* (2008), pp. 337–350.

[50] Miluzzo, E., Lane, N. D., Fodor, K., Peterson, R., Lu, H., Musolesi, M., Eisenman, S. B., Zheng, X., and Campbell, A. T. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In *Proceedings of the 6th ACM conference on Embedded network sensor systems* (2008), ACM, pp. 337–350.

[51] Mohan, P., Padmanabhan, V. N., and Ramjee, R. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM conference on Embedded network sensor systems* (2008), ACM, pp. 323–336.

[52] Monsoon inc. http://www.msoon.com/LabEquipment/PowerMonitor/.

[53] Mun, M., Reddy, S., Shilton, K., Yau, N., Burke, J., Estrin, D., Hansen, M., Howard, E., West, R., and Boda, P. PIER, the personal environmental impact report, as a platform for participatory sensing systems research. In *Proceedings of ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'2009)* (2009), pp. 55–68.

[54] Nemhauser, G. L., Wolsey, L. A., and Fisher, M. L. An analysis of approximations for maximizing submodular set functionsi. *Mathematical Programming 14*, 1 (1978), 265–294.

[55] Google nexus. `http://www.google.com/intl/all/nexus/`.

[56] Postgresql database. `http://www.postgresql.org/`. accessed: 2014-08-20.

[57] Python programming language. `https://www.python.org/`. accessed: 2014-08-20.

[58] Rachuri, K. K., Mascolo, C., Musolesi, M., and Rentfrow, P. J. Sociablesense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing. In *Proceedings of the 17th annual international conference on Mobile computing and networking* (2011), ACM, pp. 73–84.

[59] Rana, R. K., Chou, C. T., Kanhere, S. S., Bulusu, N., and Hu, W. Ear-phone: an end-to-end participatory urban noise mapping system. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks* (2010), ACM, pp. 105–116.

[60] Reddy, S., Parker, A., Hyman, J., Burke, J., Estrin, D., and Hansen, M. Image browsing, processing, and clustering for participatory sensing: lessons

from a dietsense prototype. In *Proceedings of the 4th workshop on Embedded networked sensors* (2007), ACM, pp. 13–17.

[61] Saipulla, A., Liu, B., Xing, G., Fu, X., and Wang, J. Barrier coverage with sensors of limited mobility. In *Proceedings of ACM international symposium on Mobile ad hoc networking and computing (Mobihoc'2010)* (2010), pp. 201–210.

[62] Samsung android phones. `http://www.samsung.com/global/microsite/galaxys/`.

[63] Sensordrone: Tricorder bluetooth sensor for gas light humidity and more. `http://sensordrone.com/`. accessed: 2014-08-21.

[64] Sheng, X., Tang, J., Xiao, X., and Xue, G. Sensing as a service: Challenges, solutions and future directions. *Sensors Journal, IEEE 13*, 10 (2013), 3733–3741.

[65] Sheng, X., Tang, J., and Zhang, W. Energy-efficient collaborative sensing with mobile phones. In *INFOCOM, 2012 Proceedings IEEE* (2012), IEEE, pp. 1916–1924.

[66] Stuntebeck, E. P., Davis II, J. S., Abowd, G. D., and Blount, M. Healthsense: classification of health-related sensor data through user-assisted machine learning. In *Proceedings of the 9th workshop on Mobile computing systems and applications* (2008), ACM, pp. 1–5.

[67] Tang, J., Xue, G., and Zhang, W. Maximum throughput and fair bandwidth allocation in multi-channel wireless mesh networks. In *INFOCOM* (2006), pp. 1–10.

[68] Thepvilojanapong, N., Konomi, S., Tobe, Y., Ohta, Y., Iwai, M., and Sezaki, K. Opportunistic collaboration in participatory sensing environments.

In *Proceedings of ACM international workshop on Mobility in the evolving internet architecture (MobiArch'2010)* (2010), pp. 39–44.

[69] Thiagarajan, A., Ravindranath, L., LaCurts, K., Madden, S., Balakrishnan, H., Toledo, S., and Eriksson, J. Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems* (2009), ACM, pp. 85–98.

[70] Urbanspoon. `http://www.urbanspoon.com/`. accessed: 2014-09-04.

[71] Weather channel api. `http://www.wunderground.com/weather/api/?ref=twc`. accessed: 2014-08-20.

[72] Weinschrott, H., Durr, F., and Rothermel, K. Streamshaper: Coordination algorithms for participatory mobile urban sensing. In *Mobile Adhoc and Sensor Systems (MASS), 2010 IEEE 7th International Conference on* (2010), IEEE, pp. 195–204.

[73] Wolsey, L. A. *Integer Programming*. John Wiley & Sons Inc., 1998.

[74] Yang, D., Fang, X., and Xue, G. Espn: Efficient server placement in probabilistic networks with budget constraint. In *INFOCOM, 2011 Proceedings IEEE* (2011), IEEE, pp. 1269–1277.

[75] Yang, D., Xue, G., Fang, X., and Tang, J. Crowdsourcing to smartphones: incentive mechanism design for mobile phone sensing. In *Proceedings of the 18th annual international conference on Mobile computing and networking* (2012), ACM, pp. 173–184.

[76] Ye, Y., Zheng, Y., Chen, Y., Feng, J., and Xie, X. Mining individual life pattern based on location history. In *Mobile Data Management: Systems,*

Services and Middleware, 2009. MDM'09. Tenth International Conference on (2009), IEEE, pp. 1–10.

[77] Yelp. `http://www.yelp.com/`. accessed: 2014-09-04.

[78] Yu, C. W., Wang, C.-H., Hsu, L. C., and Cheng, K. J. Coverage algorithms in gps-less wireless mobile sensor networks. In *Proceedings of ACM International Conference on Mobile Technology, Applications, and Systems (Mobility'2008)* (2008), pp. 1–7.

[79] Zhou, S., Wu, M. Y., and Shu, W. Finding optimal placements for mobile sensors: wireless sensor network topology adjustment. In *Proceedings of the IEEE 6th Circuits and Systems Symposium on Emerging Technologies: Frontiers of Mobile and Wireless Communication, 2004* (2004), pp. 529–532.

Xiang Sheng

+1-(315)308-1895
xsheng.syr@icloud.com

Education

08/2008 - 08/2014 **PH.D. of Electrical & Computer Engineering**, *Syracuse University*.

08/2008 - 05/2010 **M.S. of Electrical Engineering**, *Syracuse University*.

09/2004 - 07/2008 **B.E. of Electrical Engineering**, *Nanjing University of Science and Technology*.

Experience

09/2014 - current **Senior Software Engineer**, *Ebay Inc.*, Bellevue, WA.

05/2013 - 08/2013 **Software Development Engineer intern**, *Microsoft Corporation*, Bellevue, WA.

08/2010 - 08/2014 **Research Assistant**, *NetLab, Syracuse University*.

Featured Publications

Sensing as a Service: a cloud computing system for mobile phone sensing

Authors X.Sheng, X. Xiao, J. Tang and G. Xue

Description IEEE Sensors Conference, 2012, Invited Paper

Sensing as a Service: Challenges, Solutions and Future Directions

Authors X. Sheng, C. Gao, J. Tang and W. Zhang

Description IEEE Sensors Journal, Vol.13, Iss.10, pp. 3733-3741, 2013

SOR: A sensing based objective ranking system

Authors X.Sheng, J. Tang and G. Xue

Description IEEE ICDCS, 2014, pp. 114–123

Energy-efficient collaborative sensing with mobile phones

Authors X.Sheng, J. Tang and W. Zhang

Description IEEE INFOCOM, 2012

Leveraging GPS-less Sensing Scheduling for Green Mobile Crowd Sensing

Authors X.Sheng, X. Xiao, J. Tang and G. Xue

Description IEEE IoT Journal, pp. 328 – 336

Leveraging Load Migration and Basestaion Consolidation for Green Communications in Virtualized Cognitive Radio Networks

Authors X.Sheng, J. Tang, C. Gao, W. Zhang and C. Wang

Description IEEE INFOCOM, 2013

Signal-aware green wireless relay network design

Authors C. Gao, J. Tang, X. Sheng, W. Zhang and C. Wang

Description IEEE ICDCS, 2013

Energy efficient Algorithms for Electric Vehicle Charging with Intermittent Renewable Energy Sources

Authors C. Jin, X. Sheng and P. Ghosh

Description IEEE Power & Energy Society General Meeting, 2013

Joint Mode Selection, Channel Allocation and Power Assignment for Green Device-to-Device Communications

Authors C. Gao, X. Sheng, J. Tang and W. Zhang

Description IEEE International Conference on Communication 2014

Best Paper Award

Optimized Electric Vehicle Charging with Intermittent Renewable Energy Sources

Authors C. Jin, X. Sheng and P. Ghosh

Description IEEE Tran. on Signal Procesing, accepted

Honors

2014/06 Best paper award of IEEE Internation Conference on Communication 2014

2013/12 One of the Top 25 downloaded papers among thousands papers in IEEE Sensors Area, in Aug/Sep/Oct/Nov

2012/03 Student travel grant for INFOCOM 2012

2008/07 Graduated with honors (top 5%)

2004/09 Scholarship for outstanding new students (top 0.5%)