NO LONGER *SOLO*: SIMPLIFYING MOBILE-DEVICE ASSISTED COLLABORATION

by

Nairan Zhang

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Electrical Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2016

Date of the final oral examination: 09/13/16

The dissertation is approved by the following members of the Final Oral Committee:

Parameswaran Ramanathan, Professor, Electrical and Computer Engineering
Kewal K. Saluja, Professor, Electrical and Computer Engineering
Xinyu Zhang, Assistant Professor, Electrical and Computer Engineering
Azadeh Davoodi, Associate Professor, Electrical and Computer Engineering
Suman Banerjee, Professor, Computer Science

To Youth And Failures

ACKNOWLEDGMENTS

First, I would like to thank Professor Ramanathan, my Ph.D advisor, for spending considerable time to guide me to various interesting research topics. He always had great patience when introducing an unfamiliar topic and inspired me to come up with practical solutions.

Other faculty members and researchers in UW-Madison also provided great helps to me over past years. Some of them are Professor Xinyu Zhang, Professor Kewal Saluja, Professor Suman Banerjee, Professor Azadeh Davoodi, Professor Gurindar Sohi, Chin-Ya Huang, Pedro Melgarejo, Aditya Prakash, Xiufeng Xie, Sanjib Sur, Teng Wei and Chi Zhang.

Over my Ph.D, I actively worked with great researchers at other institutions. They showed me their ways of performing research and positively influenced me in many aspects. Some them are Dr. Sujata Banerjee, Dr. Kyu-Han Kim, Dr. Souvik Sen, and Dr. Eduardo Cuervo, all at Hewlett Packard Labs as of 2012-2013; Professor Youngki Lee, Professor Rajash Balan, Professor Archan Misra, and Dr. Kartik Muralidharan, all at Singapore Management University as of 2014-2015. There were some researchers who provided valuable feedback to my research work. Some of them are Professor Lin Zhong, Professor Haibo Chen and Professor Jie Xiong. I also like to thank all anonymous reviewers of my research papers for their useful comments and suggestions.

Finally, without the support of my parents and my wife, this Ph.D would not be possible. Ph.D was not an easy task; they are all co-authors.

ABSTRACT

In recent years, we have witnessed the emergence of mobile applications that attempt to introduce *social awareness* into mobile computing. These applications take advantage of person-oriented contextual information (e.g., location, temporal, proximity, environment, etc.) to build innovative social services. A class of applications are to build social convenience, where a city service can collect individuals' context information from their mobile devices, and aggregate them for large-scale analytics (e.g., weather and urban mobility). Another class of applications are to create social connections, where an individual can employ his/her mobile device to monitor proximity and discover other individuals who share common interests. The goal of this dissertation is to facilitate the acceptance and participation of the above applications.

A challenge for building social convenience is its applications, called *public sourcing* applications in this dissertation, compete with an individual's personal applications for the device's limited battery resource. As a result, neither the energy needs (and thus the quality) of sourcing applications nor the energy needs of personal applications can be assured. We define a novel framework sitting between individual volunteers and sourcing applications, which is used for managing energy resources on mobile devices as well as providing energy guarantees to sourcing applications. We use three key insights to build the framework. First, we can enable an individual to allocate energy into two smaller virtual batteries, one dedicatedly for the personal applications while the other one dedicatedly for the sourcing applications. Second, we can offer energy isolation to ensure that the personal applications own their virtual battery no matter how the sourcing applications use the device, and *vice versa*. Third, the aggregation of energy dedicated to sourcing applications on each device forms a pool of energy, and thus we can offer admission control service to sourcing applications and provide energy guarantee to those admitted sourcing applications.

We further describe a generalized solution to the just-described energy isolation, where an individual may classify mobile applications according to his/her needs and have one virtual battery for each class. Each virtual battery has a semantic meaning to the individual, such as *sourcing* battery, *entertaining* battery or *work-related* battery, and thus he/she can interact with any specific virtual battery (i.e., reading its percentage level) to manage energy usages of the respective application class. To enable the energy isolation, we describe a novel energy accounting system that is not built on hardware-specific power models. The key insight is end users do not typically pay attention to energy activities of fine-grained software entities (e.g., thread or process) at second and/or millisecond levels. This allows an energy accounting approach to estimate energy usages by application classes at a fairly slow rate (e.g., at minute level). We thus employ an adaptive learning paradigm and use CPU time as the sole feature to estimate per-class energy consumption. As a result, the proposed approach offers two salient features: (i) *Highly portable and immediately usable* as it does not need to train hardware-specific power models, and (ii) *Self adaptive* as it sequentially remodels per-class energy consumption in response to varying hardware, system and application states.

For building social connections, we focus on "familiar stranger" social relations in which strangers are repetitively collocated with each other. This type of relation was first observed in 1972 by Stanley Milgram, followed by an increasing number of social network research. With mobile devices that monitor proximity context, discovering familiar strangers becomes easier. In this dissertation, we enable commuters traveling on public buses or trains to enjoy multiplayer gaming with their fellow commuters, called *public gaming*. We define a discovery/matchmaking framework which help people discover familiar peer players on the same train/bus. We perform extensive measurement and evaluation to interpret the possibility of offering *public gaming*, and design an end-to-end system that provides guarantee to high-quality group experience. At the cloud side, a matchmaking service guides neighbor discovery and group formation. Once a group is formed and the group leader is selected, mobile side gaming would be built on p2p communication techniques such as Bluetooth and Wi-Fi Direct. Finally, we invite people to use the system in crowded transport settings and play latency-sensitive games (e.g., First-Person Shooter).

TABLE OF CONTENTS

Page

Appendix

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Appendix
Figure Page

Appendix
Figure Page

Appendix
Figure Page

Chapter 1

Introduction

"I just wondered how things were put together."

- Claude Shannon

Today's mobile devices (e.g., smartphones and tablets) offer a variety of sensing (e.g., microphone, camera, GPS, etc.) and networking (e.g., cellular, Wi-Fi, Bluetooth, etc.) modalities. When combined with the unprecedented computation capacity, these devices have capability to retrieve and infer a wide range of person-oriented contextual information (e.g., location, activity, transportation mode, temperature, in crowd or alone, meeting or working, etc.). When person-oriented contextual information from many individuals are grouped together, there are opportunities of building many innovative services for social convenience and social connections.

For instance, in September 2015, the United States federal government launched a "Smart cities" initiative with over $160 million to help communities improve community services [1]. The initiative is devoted to building research infrastructures (e.g., "the Array of Things" in Chicago [2]), inspiring ideas of next-generation mobile applications (e.g., "Envision America" [3] and "Multi-City Innovation Campaign" [4]), encouraging city-university collaborations (e.g., "the MetroLab Network" [5]), and a series of fundings for applying new sensing and networking techniques in areas such as health, environment, public safety, sanitation, transportation and energy efficiency. Many ongoing projects require engagement of ordinary citizens to collect relevant person-oriented contextual information. Examples include detecting "heat islands" (i.e., an urban area is warmer

than the neighboring countryside), recognizing and locating gunfires, and tracking block-level airborne pollen and pollution level. Such services, henceforth called *public sourcing*, are built on a common social collaboration paradigm between city residents and the city: people collect sensory data while city services aggregate such data for large-scale analytics. In literature, this paradigm is also referred as to *participatory sensing, opportunistic sensing, or crowd-sourcing*. One of the main contributions of this dissertation is a framework for seamless deployment of public sourcing applications.

Besides building social convenience, social connections have likewise been affected by the advances in mobile technologies. Historically, the Chicago school in sociology first stated that because of urban existence the traditional social ties such as family and neighborhood are weakened, and *"the urbanite is bound to exert himself by joining with others of similar interest into organized groups"* [6]. Later on, such interest-based groups were defined more formally by Claude Fischer as *subculture* [7]. To realize more such interest groups, Stanley Milgram described a social relation built on one's proximity context, called "*familiar strangers*". Unlike the acquaintance relation (e.g., family members, friends, co-workers, etc.), Milgram stated somebody can be physically around repetitively due to the common mobility and behavioral patterns in daily life [8]. In 2004, Paulos and Goodman at Intel Research re-validated the experiment conducted by Milgram in 1972, and observed building social connections based on the concept of *familiar stranger* is still promising in today's environments with potential to create a huge social network space. They developed a Bluetooth-based mobile application that helps recognize familiar strangers [9]. In this dissertation, we describe an end-to-end collaboration process between familiar strangers, called *public gaming*: we enable commuters on public buses or trains (a typical setting for familiar strangers introduced by Milgram) to collaboratively enjoy multiplayer games with their fellow commuters. Unlike *public sourcing*, *public gaming* is an example of another type of social collaboration, where all collaborators are peers organized in an ad-hoc manner.

1.1 Research Contributions

Public sourcing is built on people' *willingness*. There are several reasons for people to be less inclined to collaborate with city services. One of the often stated concerns is privacy leak. However, recent real-world deployments indicate that this may not be a major concern. For instance, in [10], the service provider found the energy consumed by sourcing was a bigger concern than privacy. "*When asked about the lack of privacy concerns, some participants replied that privacy was a lower concern as they lived in a densely populated urban city where their activities were already known by numerous people.*" In non-dense places, as revealed by another deployment in rural Scotland [11] which uses GPS locations from bus passengers' cell phones to estimate bus arrival time for other riders, the privacy concern might be oversold as well: "*Because of the low population in the area, it was not difficult to determine who each passenger probably was based on where they got on and off the bus. But when the researchers expressed their privacy concerns to the community, the members said it was worth the access to bus arrival information.*"

Another often stated concern is that devices' limited battery resources may not be adequate for doing both personal applications and public sourcing, and hence there may not be enough of them at a given time to meet the data collection needs of the public sourcing application. This concern can be further detailed to the following two aspects, one is at the volunteer side where one worries about whether private energy demands (e.g., for phone calls, video games, etc.) can be satisfied, while the other one is at the application side where the application worries about whether its data collection can be successfully completed.

In public gaming, the just-described issue of energy availability is often not a major concern, since by definition, the individuals are ready to engage in public gaming. Instead, to have a large acceptance, a key concern is whether one can discover other individuals with similar interests, and have high-quality gaming experience together. This will not be an issue if the players have a pre-defined relationship, such as friends, within physical proximity of each other. But in familiar stranger settings such as trains and buses, the discovery process is not natural; and due to complex environments, group coherence and collaboration experience are not trivially assured.

The research in this dissertation develops mobile services that address the above concerns to facilitate deployment of public sourcing and public gaming applications. Specifically, one of the services developed in this dissertation enables many sourcing applications to be launched by a social service provider. These applications compete with an individual's personal applications for his/her device's limited battery resource. Our solution provides energy isolation between public sourcing applications and private applications on an individual's mobile device. He/she can reserve a certain amount of energy for each class of applications. As a result, an individual is assured that participating in public sourcing will just affect their personal usage in a pre-determined fashion. Further, one can aggregate energy allocations from all participating individuals to provide *a priori* guarantee that adequate energy will be available to meet the needs of a particular data collection. Our service relies on energy contract with each individual and collection contract with each sourcing application, to provide the needed assurances when sourcing applications are launched.

To ensure energy isolation, we provide an energy accounting system. Our accounting solution is not hierarchically built upon fine-grained energy accounting since such accounting requires detailed understanding on hardware power activities. Instead, it estimates energy consumption by a class of applications (e.g., public sourcing class and personal application class) at the granularity of user perception of battery. Working at the user-perceived granularity leads to three desired features. First, there is no need to train hardware-specific power models (thus immediately-usable and largely-deployable). Second, the solution is self-adaptive to various hardware, system and application states (thus accurate). Third, it has very low energy overhead (e.g., milliwatt level) in real time (thus further enabling large acceptance).

In the setting of public transport, a typical familiar stranger setting, this dissertation provides a service that allows commuters to enjoy multiplayer gaming through peer to peer networking. A cloud-assisted matchmaking service eliminates the overhead of discovery, and peer to peer network reduces the need for high latency and expensive cellular data connections. Given the real-time dynamics of such a complex setting, the thesis presents results collected from such a system, with three real game applications, on many different public trains and buses with up to four human players in each game play. Thus, the thesis empirically demonstrates the effectiveness of an

ad-hoc familiar stranger social network, which helps people recognize and construct new social connections via mobile device-assisted proximity discovery.

1.2 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 presents *Sourcing-Contract*, an architecture with contract assurances for public sourcing applications. Chapter 3 discusses *EnergOn*, a novel energy isolation scheme with a user-perceived energy accounting technique. Chapter 4 presents *GameOn*, a system that enables public gaming on public transport. In Chapter 5, we conclude this dissertation.

Chapter 2

Sourcing-Contract : An Architecture with Contract Assurances for Public Sourcing

2.1 Introduction

Since today's mobile devices integrates a variety of sensing and networking modalities, using them collectively for ubiquitous data collection has drawn extensive discussion [12, 13, 14, 15, 16, 17, 18, 19]. For example, in [12], Mohan et al. describe a mobile application that detects potholes, bumps, braking, and honking using accelerometer, microphone, GSM radio, and GPS in a smartphone. Once a target phenomenon is detected, the mobile application signals to the city service along with its real-time location. The city service uses the data aggregated from many individuals to monitor road conditions without the need of a sensor network infrastructure.

To collect sufficient and high quality data, a sourcing application needs a lot of volunteers (i.e., their mobile devices). However, recruiting volunteers is not a trivial task in part because these applications will consume energy, potentially leaving insufficient energy left for important personal applications. On the other hand, even if a mobile user agrees on participating, a sourcing application is not assured that its data collection can be successfully completed by the user since there might be considerable personal applications running on the device at the same time.

We address the two above issues (summarized in 2.1) by defining a contract-based resource management framework, *Sourcing-Contract*, within a sourcing service provider platform. Concretely, it consists of *energy contract* between volunteers and the service provider, and *collection contract* between sourcing applications and the service provider. With an energy contract,

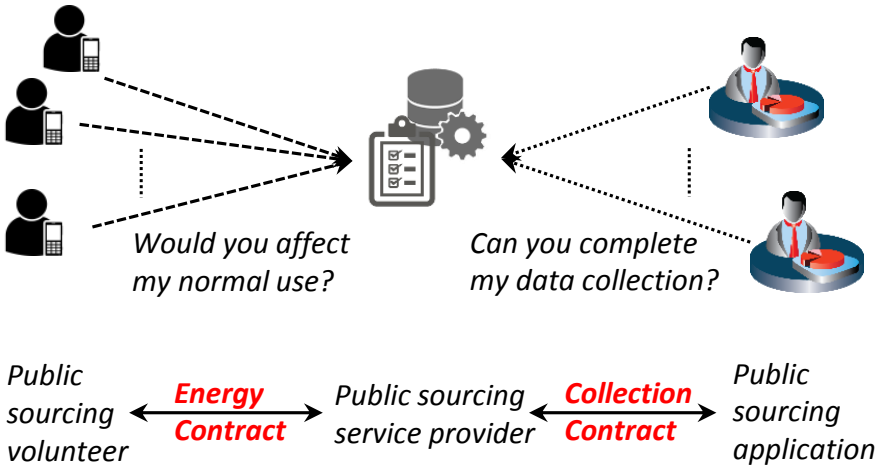


Figure 2.1: The problem space in this chapter.

a volunteer specifies his/her energy dedication to the service provider which ensures that sourcing applications can and only can use the specified amount of energy. By this way, the service provider can assure the volunteer that sourcing applications just affect personal energy usages in a *pre-determined* fashion. With a collection contract, a sourcing application can clarify its energy demand. The service provider then checks current resource (i.e., energy contracts) availability and determines whether to accept or reject the application's collection contract. The core criteria is accepting a new collection contract should not affect the completion of all *already-accepted* collection contracts. By this way, the service provider can assure an application that its collection can be completed as long as its collection contract has been accepted. In real time, *Sourcing-Contract* schedules data collection tasks so that the above two assurances can be provided.

Overall, our contributions in this chapter include:

- We propose a novel solution to runtime resource management for a sourcing service provider, alleviating the energy-related concerns at both volunteer and sourcing application sides (Section 2.2).
- We use the concept of "contract" which clarify energy supply and demand so that the sourcing collaborations are more predictable. We describe key services, protocol and implementation based on the proposed contracts (Section 2.3 and 2.4).

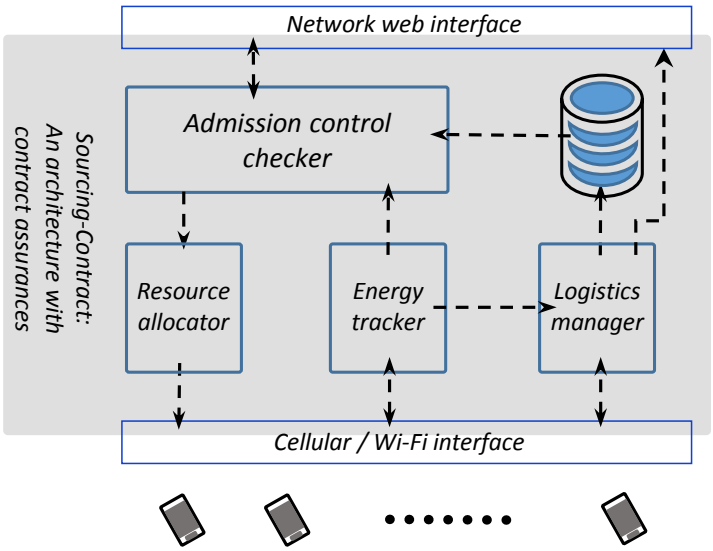


Figure 2.2: The architecture of *Sourcing-Contract*, a solution to runtime resource management.

- We present a prototype to prove its feasibility and ease of implementation (Section 2.5). We validate our solution in simulation (Section 2.6), and the results show both goals are well achieved.

2.2 Solution: An Overview

Figure 2.2 shows the system architecture of *Sourcing-Contract*, which consists of four components: the *energy tracker*, the *resource allocator*, the *admission control checker*, and the *logistics manager*. Collectively, such components constructs a framework that (i) organizes and uses available energy resources, and (ii) provides resource guarantees to various sourcing applications. We now describe the primary responsibilities of four components. In the next two sections, we will describe the way of achieving two assurances using *Sourcing-Contract*.

2.2.1 Energy Tracker

A volunteer starts by downloading a *Sourcing-Contract* -maintained mobile application *agent* on his/her device, and uses the *agent* app to allocate a portion of energy out of the physical battery. Then an energy contract, including the dedicated amount along with a validity period, can be submitted to the service provider, representing the volunteer's interest in participation. The core

responsibility of the *energy tracker* is to track the remaining energy in every energy contract so that the *Sourcing-Contract* can properly make resource allocation decisions.

2.2.2 Resource Allocator

The *resource allocator* assigns data collection tasks to the selected volunteers. From its point of view, the underlying resources are a set of distributed energy contracts. We use General Processor Sharing (GPS) as the allocation policy, and show its benefits when *Sourcing-Contract* performs admission control.

2.2.3 Admission Control Checker

Sourcing applications need to submit a collection contract before using the platform. The contract basically describes its energy demand, duration, a set of qualifier devices, etc. The *admission control checker* is primarily responsible for determining to accept or reject a submitted collection contract. It must make decision as soon as a contract arrivals because once the contract is accepted, the associated data collection tasks should be guaranteed to be completed in this end. I.e., having the incoming contract in the system should not jeopardize the energy guarantee provided to the sourcing applications previously accepted by the *admission control checker*. If the *Sourcing-Contract* denies the contract, it raises a denying code telling the reason. Then the sourcing application can adjust the contrast based on the code, or it might want to try different service providers.

2.2.4 Logistics Manager

The *Logistics Manager* keeps necessary information for every sourcing application and volunteer, such as demographic information, primitive contexts (e.g., in office or at home), and statistics (e.g., a last-seen timestamp and a credit score).

We now describe two key assurance services and our solutions. Table 2.1 lists notations used throughout this chapter.

Symbols	Description
Energy contract (Volunteer denoted as j)	
V_j	represents how late the contract is valid until
B_j	represents the maximum amount of coulomb the PS-SP can use before V_j, which is also known as the public virtual battery in the paper
Collection contract (Application denoted as A_i)	
$\mathcal{D}_i$	represents time length of a collection contract
d_i	represents time interval between two consecutive sampling
$\mathcal{N}_i$	represents a set of volunteers qualified to serve this contract
n_i	represents the number of volunteers needed for each sampling
$\mathcal{S}_i$	represents a set of sensors needed for data collection
s_i	represents the worst-case energy consumption to complete one single sampling, which is used by the admission control checker for feasibility check

Table 2.1: Notations used in the chapter for two types of contract

2.3 Volunteer-Side Energy Assurance

In *Sourcing-Contract*, a volunteer needs to dedicate a portion of energy to sourcing applications. The amount along with its expiration time is called an energy contract. The challenge of providing volunteer-side energy assurance is how to enforce an energy contract.

A key question here is for both a volunteer and the service provider how to perceive the "available energy" at the mobile device. Today's mobile users interact with the battery via a percentage level (empty=0%, full=100%). By regularly checking the level, one can apply his or her own power saving strategies and determine the time of charging. With data collections also running on the device, perceiving the remaining energy at a device becomes not straightforward. For example, suppose in the morning the volunteer has 80% battery, which is typically enough for his/her necessary energy usages over the whole day. Due to the existence of sporadically scheduled data collections, the energy might be consumed much faster than expected (i.e., there was actually less than 80% available in the morning for the volunteer), affecting the volunteer's already-established energy management experiences. Likewise, using the physical battery level can also be a trouble to the service provider: It does not know whether data collections should be scheduled to that device since the remaining battery is not guaranteed.

2.3.1 Solution: Battery Virtualization

Our design is to provide separate battery interfaces, presenting an illusion that both the volunteer and the service provider have their own "battery" on the shared device. As a result, each one can use its dedicated battery irrespective of how the other part behaves. In the background, the energy in the physical battery is partitioned into these two smaller *virtual* batteries, and it is the mobile Operating System that ensures energy isolation between two parts. We will describe the isolation scheme and its energy accounting technique in the next chapter, while in this chapter we assume such isolation exists. The service provider can only use the virtual battery for public sourcing of each device. As Figure 2.3 shows, with such usage model, the physical battery information, such as the remaining percentage level provided by today's mobile devices, is buried in

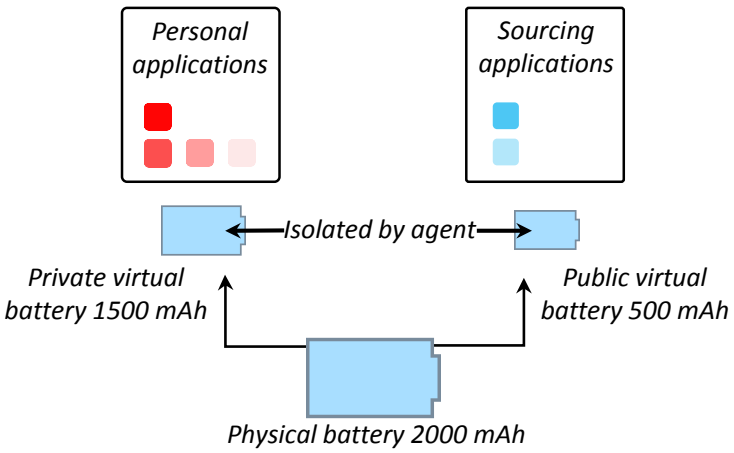


Figure 2.3: The concept of battery virtualization. The electric charge (coulomb) in a physical battery is partitioned into two smaller virtual batteries (each with empty=0%, full=100%). Sourcing applications are encapsulated into a *Sourcing-Contract*-maintained *agent* application.

the background. Instead, the device exposes percentage levels of two virtual batteries, indicating the available energy in respective virtual battery.

When a user registers with the service provider, he or she needs to download an *agent* mobile application which will actually collect sensory data on this particular device in future. Once downloaded *agent* interacts with the OS to enable the capability of battery virtualization. It then allows the user to specify the maximum energy capacity of the public virtual battery out of the physical battery, B_j, which is dedicated to the service provider. Now, a volunteer only needs to interact with the private virtual battery while the service provider would manage the public virtual battery part.

For example, with a battery that has the total amount of electric charge 2000 milli-Ampere hour (mAh) in Figure 2.3, a user may choose allocate 1500 mAh of a fully-charged battery to the personal class while 500 mAh to the sourcing class. Then, even if the user runs a power-hungry sourcing task for long periods time, he/she can be assured that 1500 mAh of the battery resources will always be available to personal applications. Likewise, there is 500 mAh charge guaranteed to the sourcing, irrespective of how the user uses personal applications.

A volunteer also specifies a validity period V_j by when any sourcing should be terminated. However, if the B_j limit is reached before the validity period expires, any sourcing should not be

allowed unless a new contract by the volunteer is submitted. Once the contract starts, the volunteer do not worry about in what way the energy B_j would be used (e.g., which sourcing applications are scheduled on his/her device, which sensors are being sampled).

Overall, since sharing energy is unavoidable while running sourcing applications, our solution guarantees that sourcing applications would just affect a volunteer's normal energy usages in a pre-determined fashion by enforcing energy contract. As we will describe in the next section, energy contract plus battery virtualization also enable easier resource management for the service provider.

2.3.2 Volunteer-side protocol

The *energy tracker* component in *Sourcing-Contract* responds to following three types of events from volunteers: V_Joining, V_Contract_Completion, V_Contract_Interruption.

V_Joining. A volunteer becomes active when he/she submits an energy contract to the service provider via *agent*. From this moment on, if he/she is qualified for a data collection task, *Sourcing-Contract* would possibly schedule the task on the device. The *energy tracker* monitors the energy usages by sourcing, preventing it from draining more than B_j coulomb. *agent* at the mobile side guarantees that personal applications do not use the pre-allocated energy promised for the sourcing.

V_Contract_Completion. When an energy contract depletes or expires, the PS-V ends up with sending a completion signal to the *energy tracker*. The *energy tracker* sends to the *logistics manager* a success flag along with the volunteer profile. The times of success and failure would be added into the history data used by the reputation and rewarding recorded in the *logistics manager*.

V_Contract_Interruption. Similarly, if the *energy tracker* finds a volunteer becomes unavailable during a data collection task (e.g., the device is turned off), it logs a fail code to the *logistics manager* and marks the already-collected data at an earlier time invalid if necessary.

2.4 Application-Side Energy Assurance

A primary goal of the service provider is to guarantee an application's data collection can be gracefully finished. However, there are two reasons that make providing such guarantees a challenge. First of all, each application has its own energy need (e.g., applications that need to continuously sense microphone would consume more energy than those with just sporadic microphone polling) and qualifying criteria (e.g., volunteers who are in their offices would not be qualifiers for detecting road condition). Thus, to offer energy guarantee, the service provider should reserve a dedicated portion of energy on qualified volunteers. Second, the available energy resources are not unlimited. From the service provider's perspective, these resources are a distributed set of public virtual batteries. The service provider needs to calculate whether the remaining energy in each virtual battery is sufficient to accommodate one or multiple energy reservations.

Our solution is using admission control to guarantee completions of all already-admitted applications. In the meanwhile, we seek for proper resource allocation policies so that the possibility of accepting a newly-arrived application is maximized. We start by introducing protocol and collection contract before describing our solution.

2.4.1 Application-Side Protocol

Figure 2.4 illustrates the communication protocol used between a sourcing application and *Sourcing-Contract*. There are three types of events to which the *admission control checker* responds (labeled in red): *A_Qualifier_Inquiry, A_Admission_Control_Check, A_Execution*.

A_Qualifier_Inquiry. This event (#1) is a query on whether there are volunteers in the specified contexts. The asked contexts are generic and *primitive*, such as location and activity, which are tracked by the *logistics manager*. The *admission control checker* searches over its database (#2) and replies to the sourcing application with all qualifiers (#3). Next, if the application has more stringent requirements on the qualifier context (e.g., riding bus around the national stadium), it may select a subset of them (#4) by combining primitive contexts provided by the *Sourcing-Contract*.

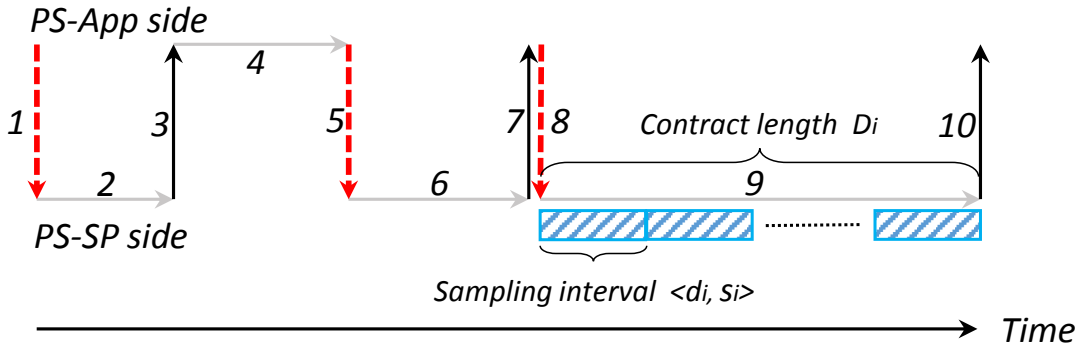


Figure 2.4: The lifecycle of a *collection contract*.

In addition, the sourcing application may also have its own user selection criteria such as those proposed in [20] to be considered.

A_Admission_Control_Check. The event (#5) occurs when a sourcing application has selected a set of qualifiers and now submits a collection contract. The *admission control checker* reviews the contract (#6) and determines whether to accept the contract (#7). Collection contract and detailed admission control algorithm will be presented in the following subsections.

A_Execution. If the collection contract is accepted, this event (#8) just starts it; otherwise the sourcing application has to adjust the contract or change to a different service provider. Other protocol design is also possible, such as *Sourcing-Contract* directly starts data collection if the contract is accepted. During the execution (#9), *Sourcing-Contract* communicates with volunteers on behalf of the sourcing application, leaving the application agnostic to runtime resource availability. While in Figure 2.4 the *Sourcing-Contract* only responds the completion notification at the end (#10), an alternative design for runtime communication can also be possible (such as sending notification after every sampling). When the contract expires, *logistics manager* logs basic information in the database as the history associated with the application.

2.4.2 Collection Contract

A *collection contract* has two classes of parameters: a coarser one in the contract scope, and a finer one in the sampling scope. As shown in Figure 2.4 (#9), the contract scope consists of multiple sampling scopes. An example collection contract described by XML language is shown

in List 2.4.2. In this example, the collection duration is 360 minutes while each sample takes 5 minutes; there are 4 qualifier devices and each sample needs only 2 out of them; each sample will engage the gps positioning at every 15 seconds and the accelerometer sampling at every 1 second, spending energy at most 10 coulomb.

More specifically, the contract scope parameters include $\mathcal{D}_i$, $\mathcal{N}_i$, $\mathcal{S}_i$. (i) $\mathcal{D}_i$ is typically determined by the physical meaning of the sourcing. For example, a sourcing application that monitors the real-time movement of spectators after a big sporting event can claim the duration from 9:30 to 11:00pm, while a sourcing application that periodically (rather than sporadically) detects pollution levels in a region may specify from 5:00 to 6:00am and resubmit its contract every hour. (ii) $\mathcal{N}_i$, as mentioned above, are selected from the return of A_Qualifier_Inquiry. When determining this set of volunteers, the sourcing application designer should be sufficiently confident that the selected qualifiers will not leave the desired context over $\mathcal{D}_i$. *Sourcing-Contract* is difficult to maintain such specific things and can only notify the sourcing application if any leaving takes place. The QualifierID field is a hash value on the device's MAC address, which is not able to be reversed by the sourcing application. (iii) $\mathcal{S}_i$ is provided by the application designer, specifying concrete sensors the sourcing application is interested in.

The sampling scope parameter include d_i, n_i, s_i. (i) d_i is determined by the sourcing application designer, which is usually a domain expert. For a real-time monitoring application, the sampling rate can be at the *Nyquist* frequency; while for a behavior observation, it is a reasonable time duration to detect the observed phenomenon. When one sampling is completed, the *resource allocator* may allocate a different group of volunteers for the next sampling. (ii) For each sampling, n_i is typically smaller than the number of qualifiers ($\mathcal{N}_i$), giving the room for the *resource allocator* to efficiently make use of the underlying energy resources. (iii) s_i is needed by the admission control, which is estimated by the sourcing application designer by in-lab profiling. There is also a crowd-sourcing solution [21] to help determine this quantity. Note that the sampling details within one sampling interval is application-specific and out of the scope of this paper.

Listing 2.1: An example contract

```
<?xml version="1.0" encoding="UTF-8"?>
<Contract xmlns="http://thesis_contract/crowd_control">
    <Duration>180</Duration>
    <Qualifiers>
        <Qualifier QualifierID="6cff5fc....c162c8d1ac"></Qualifier>
        <Qualifier QualifierID="6257fd4....d7c27fd2dc"></Qualifier>
        <Qualifier QualifierID="4cdc8c5....fcd77cec63"></Qualifier>
        <Qualifier QualifierID="332eaed....sc687b0a3d"></Qualifier>
    </Qualifiers>
    <Sensors>
        <Sensor>
            <name>GPS</name>
            <rate>15</rate>
        </Sensor>
        <Sensor>
            <name>Accelerometer</name>
            <rate>1</rate>
        </Sensor>
    </Sensors>
    <SampleInterval>5</SampleInterval>
    <MaxEnergy>10</MaxEnergy>
    <NeededVolunteers>2</NeededVolunteers>
</Contract>
```

2.4.3 Resource Allocation

We consider Generalized Processor Sharing (GPS) [22] an ideal model and design our allocation strategy to be close to it. In the GPS model, a data collection can be split into infinitesimal collection tasks. Then the allocator strategically distributes such tasks to qualifier devices in a way that every qualifier device consumes identical amount of energy. Using the foregoing notations (Table 2.1), the overall worst-case energy consumption by a *collection contract* A_i is $s_i * \left\lceil \frac{\mathcal{D}_i}{d_i} \right\rceil * n_i$.

In GPS, for application A_i over time duration t, the energy consumed on each qualifier device in the worst-case is:

$$E_j^i = \frac{n_i}{|\mathcal{N}_i|} * (s_i * \left\lceil \frac{t}{d_i} \right\rceil) \tag{2.1}$$

where $|\cdot|$ is the cardinality of the qualifier set, i.e., the number of qualifiers for this particular application. We call E_j^i as the GPS target with regard to the application i and time duration t.

In real time, it is possible that the above allocation strategy may have deviated from the GPS target in part because a selected worker device consumed less energy than the worst-case s_i. We thus use an approach that resembles Weighted Fair Queuing (WFQ) [23]: the allocation strategy is to allocate the next sampling to devices that are farther from the GPS target so that all devices in $\mathcal{N}_i$ move towards the GPS target. For example, if qualifier devices a, b and c have already consumed 10, 20, 30 coulombs for an application, and at this moment the GPS target is 40 coulombs for each. Suppose the application needs two workers for each sampling, then in the next allocation the *resource allocator* will pick a and b.

2.4.4 Admission Control Process

An admission control process can be interpreted as a "simulation" of a corresponding resource allocation strategy. The end goal of the process is to evaluate whether accepting an application at the current moment would disable all already-accepted applications to be successfully completed. In our solution, the *admission control checker* uses GPS to determine energy consumption on volunteer devices. There are many other possible strategies. As we will show in the evaluation, GPS-based admission control has higher possibility to accept a newly-arrived collection contract. Moreover, to the best of our knowledge, the non-GPS admission control during the simulation will require evaluating the energy consumption at each sampling time of every application until all applications complete. In contract, in GPS, the admission control requires the energy evaluation only when the GPS target needs to be updated, described as below.

The admission control checker is triggered when a new collection contract arrives. Let t_0 be the current time, and at this moment suppose there exists $n-1$ collection contracts labeled $\{A_i\}, i =$

$1,2,...,n-1$ in the system, and a new one A_n arrives. The objective of the admission control is to check, from time t_0 on, whether all n collection contracts can be successfully completed by their respective finish time. Naturally, the checker needs to look ahead till t_{end} which is the latest finish time among n collection contracts, i.e, $t_{end} = \max_{i \in n} D_i$. We call the period from t_0 to t_{end} a *simulation period* with respect to t_0. Once the checker finds any A_i cannot be satisfied before the simulation ends, it terminates the simulation process and rejects A_n; otherwise, it accepts A_n into the system.

The process involves computations of the remaining coulomb on each device from t_0 to t_{end}. Fortunately, as mentioned above, since GPS assumes every volunteer will continuously get a bit of tasks to do, we do not need to frequently check the remaining coulomb. Instead, we can compute only when significant events occur which will update the GPS target. Hence, the *simulation period* is partitioned into back-to-back *event-free* windows, and the remaining coulomb are computed only at each window end. We consider following three types of significant event, which will lead to updates of the GPS targets: *collection-contract-completed*, *energy-contract-completed*, and *energy-contract-depleted*. Since the first two types can be obtained directly from collection and energy contract, we only need to capture the *energy-contract-depleted* events on which a device has already depleted its public virtual battery.

The admission control process is described in Algorithm **??**. (i) Combine the *energy-contract-completed* events T_{cc} and *energy-contract-depleted* T_{ec} events, and sort the union of two sets. (ii) Find the earliest one in the union, $firstT$, plug $firstT - t_{current}$ into Equation 2.2 (replace t), and then for each device j determine whether the left hand side of the equation (i.e., the remaining coulomb) is equal or smaller than zero. If so, (iii) further determine an earlier time, rather than $firstT$, at which the right-hand side becomes zero, which is the moment an *energy-contract-depleted* event occurs.

$$B_j(t_{current} + t) = B_j(t_{current}) - \sum_{i \in I_j} E_j^i \tag{2.2}$$

Algorithm 1: Algorithm for the admission control

Input: Existing collection contracts $A_i, i = 1, 2, ..., n-1$, and the incoming collection contract A_n

Input: m engaged device (energy contracts)

Input: Completion times $\{T_{cc}\}$ of n collection-contract and Completion times $\{T_{ec}\}$ of m energy-contract

Output: Boolean result: accepte or reject A_n

$t_{current}$ = getCurrentTime(); t_{end} = getEndSimulationPeriod(); stack = initStack(); hash = loadDevices();

Sort $T_c = T_{cc} \cup T_{ec}$ in descending order and push all into the stack;

while *true* **do**

    firstT = stack.pop();

    if *firstT* $\geq t_{end}$ **then**

        Break while loop;

    maxHeap = initPriorityQueue();

    for *each device j in hash* **do**

        Compute t so that the left-hand side of Equation 2.2 is 0;

        if $t_{current} + t \leq$ *firstT* **then**

            Offer $t_{current} + t$ to maxHeap;

    if *maxHeap is not empty* **then**

        stack.push(firstT);

        while *maxHeap is not empty* **do**

            tempT = maxHeap.poll();

            stack.push(tempT);

        firstT = stack.pop(); Update GPS targets to time firstT;

        for *each device j in hash* **do**

            if *the public virtual battery level of j* ≤ 0 **then**

                hash.remove(j);

        for *each collection contract in* A_n **do**

            if *the number of qualifiers not enough* **then**

                return REJECT;

    else

        Update GPS targets to time firstT;

    $t_{current}$ = firstT;

return ACCEPT;

where I_j is a set applications that j is qualified to serve. It should be noted that the checking occurs at t_0, and hence the remaining energy after this time point are estimated values based on the worst-case energy consumption s_i.

Whenever any of three types of the significant events occurs, the *admission control checker* need to update the GPS targets. It then sums up the updated GPS targets, and determines whether every single device still has energy in its public virtual battery if such amount of energy will be consumed. If depleted, the device will be removed. If any sourcing application cannot be satisfied, i.e., the available qualifier devices are not enough for its next sampling, the *admission control checker* rejects A_n; otherwise, it moves on to the next window end.

2.5 Implementation

We now describe how the proposed architecture can be easily built on a modern web service framework. Our current prototype consists of a cloud-side service layer, and a mobile-side *agent* software. The former implementation is based on the Play framework and the latter is based on Android. However, the architecture is agnostic to specific Play and Android features, and can be realized on other platforms with similar engineering efforts.

The service provider prototype is built on the Play Framework [24] with version 2.5.4, a web service framework supporting Java and Scala applications. The communication between an application owner and the service provider is via Apache HTTP interfaces. Specifically, the collection contract is submitted through the POST method that carries an XML format contract. The internal modules in the service provider are implemented within the Play framework, and volunteer profile and history data are stored in a MySQL database. The resource allocation module in the service provider uses WebSocket persistent connection (with Json data format) to communicate with volunteers. We deploy the architecture in a lab computer with Intel i5-3470 4-core Processor.

The mobile-side prototype with battery virtualization support is implemented in Android OS version 2.3 on Google Nexus S smartphone. Our implementation modifies the Android framework so that *agent* can terminate and prevent sourcing applications from starting when its virtual battery

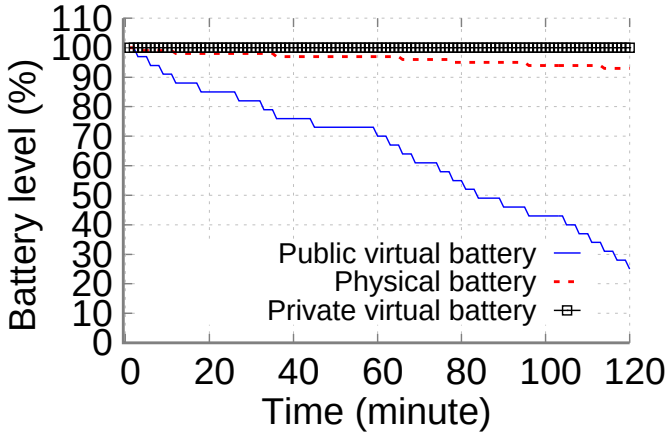


Figure 2.5: Percentage level changes of physical battery and two virtual battery.

level goes zero. The modification was using the source code downloaded from the Android Open Source Project [25].

2.6 Evaluation

Our volunteer side evaluation was performed on the prototype implementation with a focus on illustrating the effectiveness of energy isolation. Our application side evaluation, in simulation, shows the benefit and cost of GPS-based admission control compared against Largest Battery First (LBF) resource allocation policy.

2.6.1 Results of Battery Virtualization Interface

We used our prototype implementation to illustrate how a volunteer would interpret the energy usages of public sourcing. In the experiment, we run the PS-SP with 10 devices, 9 virtual devices which were implemented by software (a.k.a. *bot*), and 1 real physical device. The service provider allocates tasks to 10 devices according to our allocation strategy. The real device has a battery with 1650 mAh total capacity, and offers a public virtual battery that has 150 mAh total capacity. Thus, the rest 1500 mAh is for the private virtual battery. Each of two virtual battery presents a percentage level from 0% to 100%, indicating the remaining energy. Figure 2.5 shows the percentage levels of both changed over the 2-hour evaluation period. The public virtual battery scheduled to execute

sensing tasks dropped quickly. However, since there was not any personal applications activated, the private virtual battery level kept 100%, not affected by the public sourcing. It also should be noted that *Sourcing-Contract* will only look at the public virtual battery to make allocation and admission control decision.

We also plotted the physical battery level changes over time. Unlike today's common battery interface, however, its percentage may not be shown on the device screen in the real use. The volunteer only focuses on the private battery. To accurately compute the virtual battery level the device needs an energy model. We will describe our energy accounting solution in the next chapter.

2.6.2 Results of GPS-Based Admission Control

We seek for an allocation policy that has higher possibility to accept more collection contracts. To study the effectiveness on this measure of various policies, we randomly created many configurations, each including a fixed set of energy contracts and a sequence of collection contracts. For each configuration, we added its collection contracts into the pool of energy contracts one after another one, and for each newly-added collection contract, we performed admission control process to determine whether it should be accepted. The sequence number of the first rejected collection contract with regard to a particular configuration might be different when using different allocation policies: the later the first rejection occurs, the more collection contracts the policy is able to accommodate.

Over many configurations, we can statistically compare the first rejection between GPS-based admission control with other policies. In this dissertation, we used another simple policy called Largest Battery First (LBF) policy that schedules collection tasks to qualifiers who have more energy in their public virtual batteries. The simulation were carried out for a wide range of parameters. The parameters include the maximum public virtual battery capacities of energy contracts, the durations of collection contracts, the sample intervals of collection contracts, the worst-case energy consumption values of collection contracts, the ratio of the number of volunteer devices (i.e., energy contracts) to the number of applications (i.e., collection contracts), the ratio of the number of qualifier devices to the number of volunteer devices, and the ratio of the number of participants

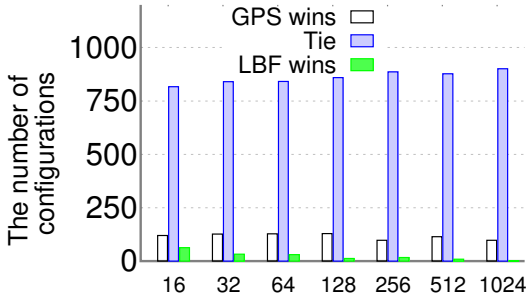


Figure 2.6: Possibility density of first rejection as the number of volunteers increases.

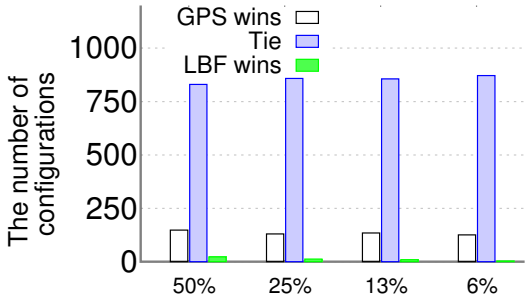


Figure 2.7: Possibility density of first rejection as ratio of #qualifiers to #volunteers goes down.

to the number of qualifier devices. Effect of each one of these parameters were analyzed from these simulations. Due to the large number of parameters, only significant and/or typical results from these simulations are presented here.

Figure 2.6 shows the variance of possibility density of the first rejection. We fixed the number of applications as ten, and increased the number of volunteers a configuration includes. Note that as the number of volunteers increases, the number of qualifiers and the number of participants increases accordingly. For each setting, we generated one thousand configurations. Observe that in most configurations, both GPS-based and LBF-based admission control have the same first rejection (i.e., "tie" in the plot). However, as there are more volunteers available in a configuration, LBF-based admission control is less likely to "win", i.e., have a later first rejection that GPS-based admission control. In the rightmost setting, the ratio of the number of applications to the number of volunteers is around one tenth.

Figure 2.7 shows the variance of possibility density when we tuned the ratio of the number of qualifiers to the number of volunteers. Likewise, in most configurations both policies reject collection contract at the same time. As the ratio decreases, GPS-based admission control starts to perform better significantly.

2.6.3 Results of Resource Allocation

The goal of this evaluation is to illustrate how *Sourcing-Contract*, using a battery-centric resource allocation described in this paper, would perform compared against a battery-agnostic *random* allocation scheme. Note that we use the *random* allocation because it also uses the same admission control equations. Other policies such as the LBF-based admission control, as explained in earlier sections, use different equations.

The following results are based on the a simple *proof-of-concept* configuration as below. It includes two applications and four volunteers. All volunteers are qualified for the application 1 while only volunteer #3 and #4 are qualified to serve application 2. The application 1 needs two workers for each sampling and the application needs one single device. Two applications are also different in the values of other parameters which are described in earlier sections.

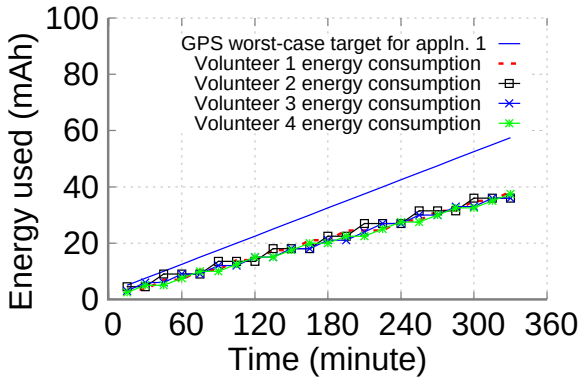


Figure 2.8: Actual energy consumption of 4 qualifiers compared against the GPS target of application 1.

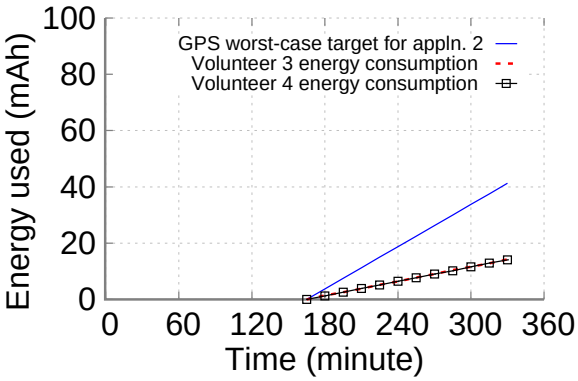


Figure 2.9: Actual energy consumption of 2 qualifiers compared against the GPS target of application 2.

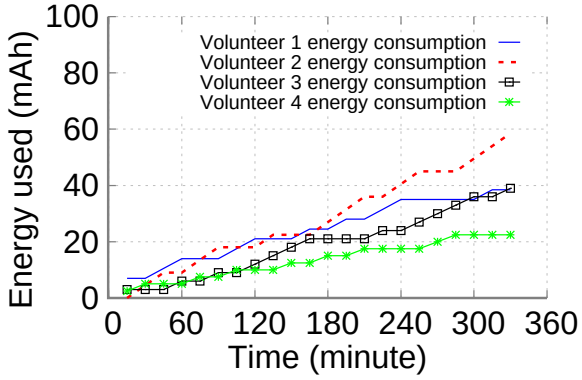


Figure 2.10: Actual energy consumption of 4 qualifiers by running application 1 using *random* allocation.

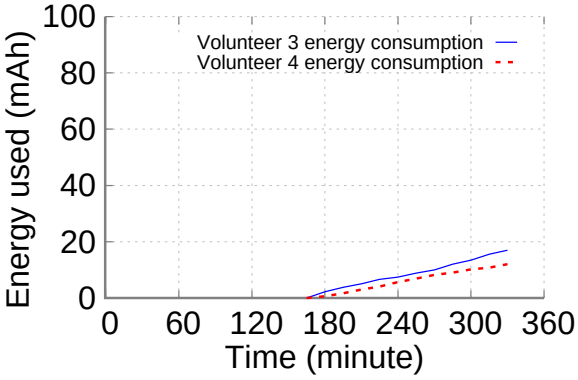


Figure 2.11: Actual energy consumption of 2 qualifiers by running application 2 using *random* allocation.

Figure 2.8 and 2.9 show the energy consumption over a 6-hour period. Observe that the GPS allocation has an important feature that for every single application, all qualifiers consume battery at almost the same speed. Also, per-application energy consumption on each device is deterministically smaller than the corresponding GPS target. These two salient features make the resource allocation highly predictable. On the other hand, Figure 2.10 and 2.11 show how the *random* allocation made decisions under the same setting. Clearly, energy consumption for each qualifier are diverged due to the battery-agnostic allocation.

2.7 Related Work

2.7.1 Admission Control And Scheduling

Admission control has existed long before the idea of public sourcing with mobile devices. It plays a critical role in many resource-constraint computer and communication systems such as supercomputing [26], networking [27], and cloud computing [28].

In contrast, little attention has been drawn in the domain of mobile device-based public sourcing. A unique challenge here is ordinary people are engaged, making resource supply a very dynamic and unpredictable factor. In this circumstances, we propose an admission control protocol along with a resource allocator to guarantee the sourcing quality. A similar work to deal with human factors is Medusa [29]. However, authors did not address energy concerns of participants and applications.

2.7.2 Energy-Aware Resource Management for Public Sourcing

PRISM [30] focused on the ease of developing and deploying sourcing applications. Also, their system monitors the energy consumption of public sourcing applications, and enforces limits on their energy use. In contract, *Sourcing-Contract* focuses on a complete both-end energy management solution.

PCS [31] reduces the energy consumption of sourcing applications by piggybacking their executions with the user's personal applications. Micro-blog [32] recognized the location service

is commonly important for sourcing applications, and traded the energy used by sourcing applications with the location accuracy. The most similar work as *Sourcing-Contract* was proposed by Zhao et al [33]. Authors considered the energy-aware resource allocation problem. Again, their solution trades energy efficiency with fairness for the resource allocation problem while the *Sourcing-Contract* manages the battery to guarantee the quality of data collection

2.7.3 User Selection

There are also discussions on selecting proper participants out of a pool of volunteers so as to meet specific data collection requirements. Current best user selection practices differ from each other in the information used to make such decision. Among them are reputation-aware [20] solutions that consider the quality and validity of one's history commissions, pricing-based [34] solutions that balance bids of volunteers with the application budget, and churn-prevention [35] solutions that are to prevent existing participants from leaving.

However, these work do not consider the battery availability as a primary concern. In contrast, *Sourcing-Contract* considers it a first class concern for a sourcing platform, especially when the energy resources is fairly limited (e.g., the number of registered people on the platform is low). Neglecting the energy availability would possibly result in a decreasing number of potential devices.

2.8 Conclusion

While public sourcing is a promising technique, it has not been widely adopted. Alleviating energy concerns described in this chapter is one of the most critical challenges. In this paper, we proposed our solution approach to providing energy assurances to both sourcing volunteers and sourcing applications. The core idea is to clarify energy supply and demand via contracts, and *Sourcing-Contract* offers the energy management service and admission control service that make the contracted strictly followed. We implemented and simulated the proposed approach, and the results show it is feasible and effective. In future, we plan to solicit sourcing applications and volunteers to use our system.

Chapter 3

EnergOn : Ensuring Energy Isolation via Unsupervised Energy Accounting

3.1 Introduction

In the last chapter, we have summarized the concept of battery virtualization, which can be a solution for public sourcing volunteers to managing their battery that caters to sourcing applications and personal applications at the same time. A more general situation is to enable mobile users to classify their applications and manage energy of application classes according to their own preferences.

In fact, today's mobile users use personal mobile devices to navigate their work life, personal life, disparate social circles, etc. and mobile applications correspondingly are purposed in different settings. For example, public sourcing applications enable users to be actively involved in building social convenience, while some work-related applications installed by enterprise IT people enable employees to access corporate data and system [36]. As a result, applications on a device are naturally separated into a few classes, each having a *semantic* meaning to the user, such as *Personal*, *Sourcing*, and *Work*. While earlier work [37, 38] have discussed the methods of creating sandboxes for such classes to guarantee the isolation of functions and data, a less discussed yet critical point is the energy resource, battery, is also shared by such classes.

In this chapter, we present our design, *EnergOn*, to ensure the energy isolation between such application classes: once a mobile user specifies the maximum amount energy for each class, *EnergOn* strives to guarantee that any class can and only can use the energy in its own battery. In its core, *EnergOn* has an energy accounting system to measure remaining energy in respective

virtual batteries at any given time. Several successful techniques in this area target on fine-grained energy measurement. For example, Eprof [39] traces energy consumption at thread and system call levels, while Cinder [40] controls energy usage at application and process levels. In contrast, the energy accounting granularity that *EnergOn* pursues is as large as a class of mobile applications.

An important reason why *EnergOn* is not hierarchically built upon fine-grained energy measurement is their solutions usually rely on hardware-specific power models [41, 42, 43, 44] trained at an offline time. Unfortunately, such energy models are tied to a single point of time, i.e., the training time, and hence when new hardwares are integrated and/or software workloads are different, the models have to be re-trained [45]. Unfortunately, time-sensitive applications such as collecting data for public sourcing applications might not allow a re-training process. Moreover, some solutions [40, 46] closely work with the process management primitives in the Operating System kernel, making such solutions not flexibly ported on various mobile platforms.

Unlikely, *EnergOn* does not rely on any hardware-specific power models. It iteratively learns energy consumption models with regard to softwares on an application classes basis. This solution leads to an *unsupervised* approach which can adapt the energy accounting to various hardware, system and application states. We take advantage of the fact that mobile users do not typically care about fine-grained energy activities, and thus we reduce the frequency of energy measurement to a human-perceived timescale, i.e., the minute level. Correspondingly, we use simple features, e.g., the CPU time, to scale per-class charge consumption over such long intervals.

We hence trade the capability of performing fine-grained energy control with several important properties, namely, *the maximum possible portability*, *adaptivity to various hardware, system and application states*, and *negligible energy overhead*. Empirical evaluation shows our approach achieves comparable accuracy with existing energy accounting solutions in scenarios of pursuing accuracy at coarse-grained timescales.

Overall, our primary contributions include:

- We make a case for user-perceived battery management across multiple application classes. We further design an energy isolation model to provide energy usage assurance to the classes (Section 3.2).

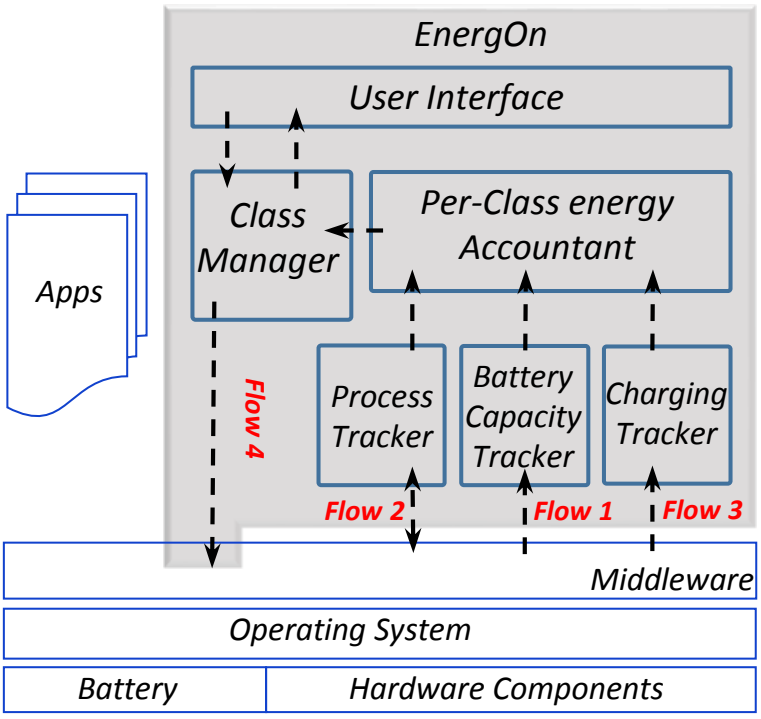


Figure 3.1: *EnergOn* architecture.

- We present an unsupervised energy accounting approach that does not rely on the hardware-based system energy model. The approach also incorporates the charging phase, making it a complete solution for end users to manage class-level energy usages (Section 3.3).

- We present an implementation of the energy isolation as a battery management tool for end users (Section 3.4), and by performing an extensive evaluation, we show our solution can produce accurate results compared with other approaches, while without using detailed model training process (Section 3.5).

3.2 Proposed Solution

3.2.1 Architecture

EnergOn runs on top of an existing mobile OS kernel (e.g., Android, iOS, webOS, etc.) across middleware and application layers. Figure 3.1 shows the diagram of the *EnergOn* architecture. Initially, the user needs to configure the *EnergOn* system via the user interface. This includes

grouping applications into classes, allocating energy resource to the classes and specifying the preferred charging policy. These configurations can be adjusted afterwards. Once the *EnergOn* system is installed, a core component, called *per-class energy accountant*, starts to estimate the remaining energy for each class over time.

Specifically, it listens to the updates of physical battery information regularly reported by the underlying system (*flow 1* in Figure 3.1), which in turn triggers a model update. In the current implementation, the battery information is the changes of the physical battery percentage level over time. For one single estimation, *EnergOn* reads per-class runtime statistics via *flow 2*, and performs linear regression to estimate the remaining energy in each virtual battery. At the very beginning of using *EnergOn*, the *per-class energy accountant* cannot produce estimations since it does not collect enough observations to do regression. After this *bootstrapping latency*, it can generate estimates sequentially. The virtual battery information are maintained by the *class manager* that controls the application executions (*flow 4*) as well as updates the user interface. When a charging status is detected via *flow 3*, the *per-class energy accountant* distributes new energy into virtual batteries according to the user's charging policy.

3.2.2 Isolation Model Used by Class Manager

Suppose that a mobile phone has battery with a maximum capacity of $\bar{B}$ mAh. Further suppose that the user applications in a mobile phone are partitioned into M classes denoted by $A_1, A_2, \ldots, A_M$. To meet the energy usage of each class, the user has also pre-partitioned the physical battery into *M virtual batteries*.

Let $B(t)$ denote the energy remaining in the physical battery at time t. Let VB_i denote the virtual battery of class A_i, and let $B_i(t)$ denote the energy remaining in VB_i at time t. Then, an algorithm to isolate the physical battery is correct if the following five conditions are satisfied.

1. *Conservation of Charge Condition*: At all times t,
   $\sum_{i=1}^{M} B_i(t) = B(t)$.

2. *Fractional Allocation Condition*: Suppose that the user allocated a fraction ϕ_i of the physical battery to virtual battery VB_i, $1 \leq i \leq M$. Then, for all $1 \leq i \leq M$ and for all times t, $B_i(t) \leq \phi_i \bar{B}$.

3. *Energy Consumption Condition*: If VB_i has no remaining energy, then an application from A_i will not be permitted to run. Note that, this condition is imposed even if the physical battery still has some remaining energy.

4. *Charging Allocation Condition*: Let $C(t_0,t)$ denote the amount of new electric charge into the battery from the charger in the interval $[t_0,t)$. This energy has to be allocated to virtual batteries using a suitable allocation policy. Let $C_i(t_0,t)$ denote the VB_i's share of $C(t_0,t)$. Then, $\sum_{i=1}^{M} C_i(t_0,t) = C(t_0,t)$.

5. *Consistency Condition*: Let $E_i(t_0,t)$ denote the energy drawn (without charging) from the physical battery by applications in A_i in the time interval $[t_0,t)$. Then, $\sum_{i=1}^{M} E_i(t_0,t) = E(t_0,t)$. Integrating the charging phase, for all $1 \leq i \leq M$ and for all times t, $B_i(t) = B_i(t_0) + C_i(t_0,t) - E_i(t_0,t)$.

3.2.3 Desired Features

Realizing the above isolation model needs an energy accounting system that estimates available coulomb in each virtual battery. In our design, we focus on the following features. Table 3.1 summarizes the features and our design choices.

Zero training burden. Without the training burden and the kernel-space mechanism, *EnergOn* can be immediately usable to most mobile devices and achieve the maximum possible adoption.

Self-adaptive energy model. By iteratively updating model, *EnergOn* does not require a re-training process. This is desired in public sourcing situation, where the assigned tasks are heterogeneous, and enter/exit sporadically, which does not allow re-training.

Energy friendly in the wild. To be a practical and largely-adopted energy management tool, energy overhead of the proposed system should be negligible.

Features of EnergOn	*Design choices used in EnergOn*
Be independent of hardware and also general to various systems	Eliminate hardware-specific power model construction; reduce the coupling of the implementation with the underlying system.
Be adaptive to various hardware, system and application states	Employ self-adaptive online learning paradigm, and update the charge consumption model iteratively.
Have negligible energy overhead while the user is using the tool	Reduce the frequency of energy measuring down to the minute-level, and use simple model features to estimate per-class energy consumption.
Trade-offs	
Not for fine-grained energy management, be suitable only for scenarios of pursuing accuracy at coarse granularity.	

Table 3.1: Desired Features, design choices and trade-offs of the *EnergOn* design.

An important tradeoff involves *EnergOn's* capability to manage battery usage in fine-grained manners. For example, it cannot estimate the energy consumption of applications at second and millisecond levels, and hence is not suitable for system designers. The paper shows that such a design is suitable only for scenarios of pursuing accuracy at the speed of human perception.

3.2.4 Compared With Prior Work

Prior work on energy budgeting. This body of research focus on *system-level* and *fine-grained* approaches. Authors in Cinder [40] describe a new OS which has explicit primitives for fine-grained energy management. The thread scheduler in the OS supports two novel primitives called *Reserves* and *Taps* through which a thread can allocate and bound the energy consumption rate of each of its spawned threads. By utilizing these primitives, one can easily bound the energy consumption rate of an application irrespective of the behavior of the other applications, i.e., achieving isolation. Not limited to the mobile devices, Zeng et al [46] present an OS-level energy management solution that assists the OS to fairly allocate energy usages to applications.

These work are closely built on specific system kernel such as HiStar to Cinder and Linux to EcoSystem. They are for the OS to manage system energy, while not a direct solution for end users to manage mobile device battery. There are several work on battery interface [47, 48], which talks about user-level energy management; but they do not provide energy isolation.

In contrast, *EnergOn* targets on the maximum portability. It eliminates the hardware-based modeling process, and has a general interface with the underlying system, which can be easily ported on many mobile platforms. Also, by decoupling energy accounting logic from the hardware and kernel, *EnergOn* achieves self adaptability when facing various software workloads and hardware states.

Prior work on energy accounting. PowerTutor [41] is one of the first work that targets on a model generation system for Android-based Smartphones. It employs a self-modeling paradigm to train power models for key hardware components such as display, networking and CPU. Next, Dong et al [45] present Sesame that allows a mobile system to construct a high-rate system power model based on the current sensor. Given that the current sensor is not commonly available on mobile platforms. Xu et al [43] present V-edge that only relies on the voltage sensor available on most mobile platforms. There are also other successful work [42, 49] that address different aspects of power modeling.

There are other energy accounting work that does not rely on a power modeling. Dong et al [50] challenge the accuracy of power model-based approach and present a solution that is based on cooperative game theory. However, the solution also requires fine-grained measurement.

Unlike the above work which often require energy measurement for thread or process and at a second or millisecond level, *EnergOn* explicitly takes advantage of the relatively slow speed of the human perception of battery (i.e., at the minute level), and is only suitable for managing mobile battery on an application-class level.

3.3 Unsupervised Energy Accounting

We next present our energy accounting technique in detail. We describe the discharging phase followed by the charging phase, and formulate the entire algorithm as an optimization problem. Without loss of generality, we use a two-class scenario as a running example.

3.3.1 An Online Learning Paradigm

The *Consistency Condition* (without charging) shows energy consumption over a time window $[u,v)$ satisfies:

$$E(u,v) = E_1(u,v) + E_2(u,v) \tag{3.1}$$

Then, we assume that the energy consumption by the applications in class A_i in $[u,v)$ is modeled by

$$E_i(u,v) \quad = \quad \beta_i(v)\delta_{\text{CPU},i}(u,v) \tag{3.2}$$

where, $\delta_{\text{CPU},i}$ is the CPU time used by application class i.

Whenever the system informs a change on B, *EnergOn* can obtain a new observation of Equation 3.1. Then over an extended period w with multiple instances of equation, *EnergOn* is able to compute model coefficients using the *least square* regression.

Example: Consider a mobile phone with two application classes. For simplicity, assume that each class contains only one application; one class contains YouTube and the other contains Navigation. Assume that the physical battery is fully charged, i.e., at 100% level, at time t_0. Due to the charge drawn by YouTube and Navigation, assume that battery level drops to 99% at time t_1, to 98% at time t_2, 97% at time t_3 and 96% at time t_4. Then, based on the energy consumption model of Equation 3.2, we can write the following four equations, one for each of the intervals $[t_0,t_1)$, $[t_1,t_2)$, $[t_2,t_3)$, and $[t_3,t_4)$, with the assumption that $\beta_1(t_k)$, $k = 1,2,3,and4$ are approximately equal and let $\beta_1 = \frac{1}{4}\sum_{k=1}^{t}\beta_1(t_k)$. Similarly, let $\beta_2 = \frac{1}{4}\sum_{k=1}^{t}\beta_2(t_k)$:

$$\beta_1 * \delta_{\text{CPU},1}(t_0,t_1) + \beta_2 * \delta_{\text{CPU},2}(t_0,t_1) = 0.01 * \bar{B} + \varepsilon(t_1)$$
$$\beta_1 * \delta_{\text{CPU},1}(t_1,t_2) + \beta_2 * \delta_{\text{CPU},2}(t_1,t_2) = 0.01 * \bar{B} + \varepsilon(t_2)$$
$$\beta_1 * \delta_{\text{CPU},1}(t_2,t_3) + \beta_2 * \delta_{\text{CPU},2}(t_2,t_3) = 0.01 * \bar{B} + \varepsilon(t_2)$$
$$\beta_1 * \delta_{\text{CPU},1}(t_3,t_4) + \beta_2 * \delta_{\text{CPU},2}(t_3,t_4) = 0.01 * \bar{B} + \varepsilon(t_4)$$

We can estimate β_1 and β_2 by minimizing the square error, $\varepsilon(t_1)^2 + \varepsilon(t_2)^2 + \varepsilon(t_3)^2 + \varepsilon(t_4)^2$, subject to the above four equations. Then, the energy consumption from the battery by YouTube and

Navigation in the interval $[t_3, t_4)$ can be estimated as:

$$E_1(t_3, t_4) = \beta_1(t_4) * \delta_{\text{CPU},1}(t_3, t_4)$$
$$E_2(t_3, t_4) = \beta_2(t_4) * \delta_{\text{CPU},2}(t_3, t_4)$$

Thus, we can estimate per-class energy consumption over any (t_i, t_{i+1}). We now discuss two important parameters in the above accounting process.

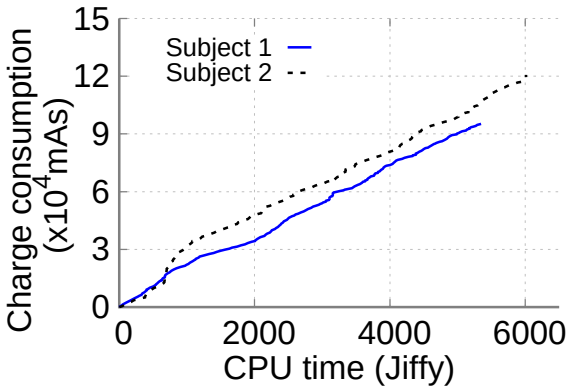
The estimation interval, $[u, v)$, denoted as $\mathbb{T}$ throughout the paper. The above example uses 1% as estimation interval. However, this is not a must for the algorithm. If there is other source to provide more precise $E(u, v)$ than physical battery level change, it can be easily integrated into the algorithm. However, in *EnergOn's* usage scenarios, $[u, v)$ is relative long for the energy consumption model to be accurate.

The number of *Least Square* equations, $\mathbb{N}$. It represents an observation window during which the model coefficients are unchanged. It is sufficient to say that the *least square* computation becomes more accurate as more equations are added; while as the window size increases, the assumption that coefficients are constant (not adaptive) throughout the larger window may degrade the estimation accuracy.

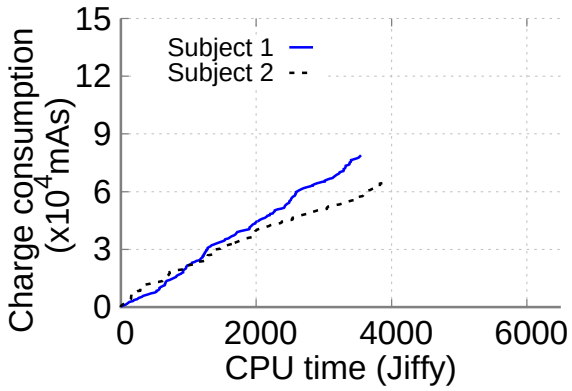
3.3.2 Per-Class Energy Consumption Model

As described above, the time window $[u, v]$ is relatively as long as the minute level. In the current implementation, we use the time elapsed for 1% physical battery level change. Correspondingly, $E(u, v)$ represents the amount of $1\% * \bar{B}$ coulomb that has been consumed since the last 1% drop. We then model the energy consumption by an application class as a linear function of the CPU time of its applications. Over a relatively long interval $[u, v)$, we observe the energy consumption from the battery can be well-characterized using a linear model as a function of CPU time.

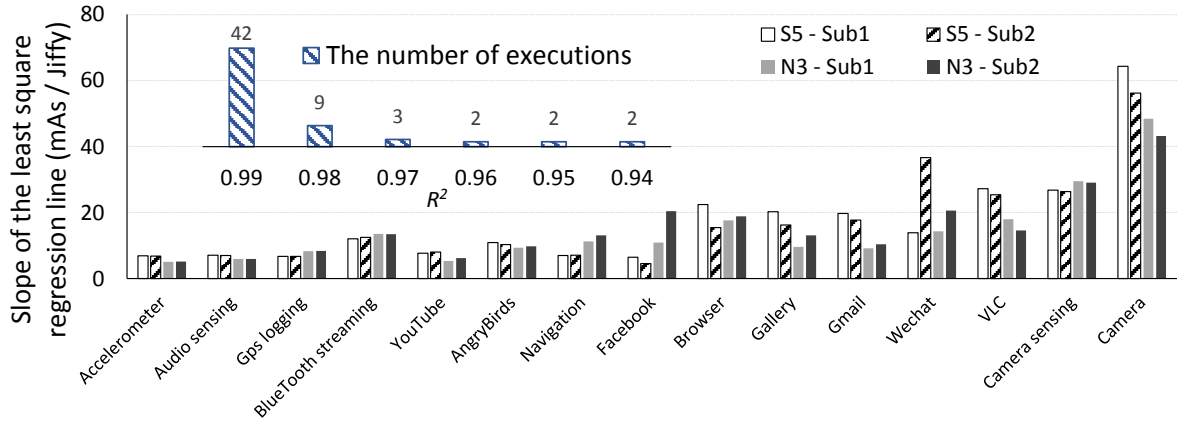
Figure 3.2(a) and 3.2(b) visualize the dependency of the device's energy consumption on the application's CPU time. We invited two subjects to play the *Browser* applications on two different



(a) Running *Browser* On Note 3



(b) Running *Browser* On Galaxy S5



(c) Statistics of *least square* parameters (slope and R^2) across 60 executions, each lasting for 200 seconds.

Figure 3.2: Linear correlation between per-application CPU time and the device charge consumption. Experiments are done by two subjects playing 15 apps on two phones. Only *Browser* is highlighted while results for other apps are visually similar. Jiffy is a system-defined time unit, used for the CPU time.

Application	Background	Customized	Interactive
YouTube	No	No	No
AngryBirds	No	No	Yes
Navigation	No	No	No
Facebook	No	No	Yes
Browser	No	No	Yes
Gallery	No	No	Yes
Gmail	No	No	Yes
WeChat	No	No	Yes
VLC	Yes	No	No
Camera	No	No	Yes
Accelerometer sensing	Yes	No	No
Audio sensing	Yes	Yes	No
Gps logging	Yes	Yes	No
BlueTooth streaming	Yes	Yes	No
Camera sensing	Yes	No	No

Table 3.2: Fifteen benchmark applications for evaluation.

mobile devices. When a device was used, it was connected to a MonSoon [51] power monitor to measure the energy consumption of the device. We also logged the CPU time of the *Browser* at every second. Observe that over a short time period, there is considerable variation in the energy consumption. These variances are caused by the uses of different hardware resources. However, over a relatively long interval, e.g., 200 seconds in this experiment, the CPU time is an adequate linear proxy of the energy consumption.

We further use 15 Android applications to validate the linear model (see Table 3.2), including 12 commercial applications and 3 customized applications. This set of applications covers most types of workloads such as sensing, interactive, computation and network. Figure **??** summarizes the statistical results of the best fitting straight line from all 15 applications. Each execution lasts for 200 seconds. The small figure shows all executions have R^2 equal or greater than 0.94. Overall, although considerable variation exists over fine-grained time intervals, the linear model based solely on an application's CPU time can effectively explains the charge consumption over a relative long period; in fact, power draw variances caused by software logics, user actions, and context changes would be largely smoothed out over a long interval, e.g, 1% physical battery level change, even though the variances are considerable and evident at fine-grained timescales such as seconds and milliseconds. Consider a user is swiping pictures through the OLED display. Each picture having a unique color distribution drains the battery at its own rate, while each one spends similar amount of the CPU time as other pictures. That is, the CPU time would not be a good indicator for showing individual pictures. *However, over minutes when the user has browsed tens of hundreds of pictures, the "overall" color distribution may be uniform, and arguably, the CPU time as the feature can reflect this uniform distribution.* The CPU time has been used in several work (e.g. Magpie [52]) as the proxy of long-term software workloads. We use it here to reflect long-term energy consumption.

3.3.3 Concurrent Execution

In our algorithm, applications in different classes can be executed sequentially, or they may also be in parallel if the system and hardware support concurrent execution. However, the ground

truth of concurrent execution is not readily available. As a result, existing work either use various policies to allocate energy usages or "provide" a ground truth for energy accounting. Our isolation model described in Section 3.2.2 is a conjecture about the ground truth. System processes (including hardware housekeeping energy) are shared by classes. They are not visible to end users and hence will not be allocated into any class. Thus, we consider that they consume energy on user applications's behalf.

3.3.4 Integrating Charging

Battery charging circuits in most mobile devices operate in two modes, *constant current* mode and *constant voltage* mode. The constant current mode is used when the battery level is below a chosen threshold often around 90%. In this mode, the battery charges rapidly. Once the battery level exceeds the threshold, the charger switches to the constant voltage mode in which the output of the charger is maintained at constant voltage. This causes the charging current to decrease with a commensurate decrease in charging rate. This is done so that the charging process changes to almost a trickle when the battery is fully-charged but the device is connected to an external power supply.

To calculate the amount of new charge, we need to know the present charging rate. In *EnergOn*, we assume the above transition point is at 90%, and acquire the charging rate while the user connects the device to a charger: when *EnergOn* detects a charging behavior, it evaluates the amount of present workload by checking the overall CPU time. If the load is light (i.e., almost idle), *EnergOn* determines the charging rate using the 1% charge capacity over the time for increasing 1%. Once *EnergOn* obtains the charging rate, the amount of new charge over any time window equals to the charging rate times the window size.

With the charging phases, we now describe the algorithm in more general way. Let $t_0, t_1, \ldots$ denote the times at which *EnergOn* receives a callback from battery manager of underlying mobile platform or OS. Typically, these occur when the physical battery level changes by 1%. Assume that the current time is t_k and we are using a window of w to estimate the values of β. Then, the optimization problem used to estimate these values can be written as follows.

Minimize $\sum_{l=0}^{w-1} \varepsilon_{k-l}^2$

Subject to:

$$\sum_{i=1}^{M} \beta_i * \delta_{\text{CPU},i}(t_{k-l-1}, t_{k-l})$$
$$= B(t_{k-l-1}) - B(t_{k-l}) + C(t_{k-l-1}, t_{k-1}) + \varepsilon_{k-l}$$
$$\text{for each } l = 0, \ldots, w-1.$$

$\beta_i \geq 0$, for $1 \leq i \leq M$

where $\beta_i = \frac{1}{w} \sum_{l=0}^{w-1} \beta_i(t_{k-l})$.

A slightly trickier issue is the charge delivered to the battery during the charging process is tied to the whole device. Hence, we need to allocate the new charge a particular virtual battery. There are some possible user-specified policies to do so. For example, in the *Least Percent First Policy*, all the incoming charge is allocated to the virtual battery that has the smallest percentage level, while in the *Proportional Allocation Policy*, the incoming charge is allocated to the non-full virtual batteries in proportional to their partition portions. We now describe four charging policies.

Let $\frac{dC(t)}{dt}$ denote the rate at which the charge is being supplied to the battery from the charger at time t. Let $\mathcal{N}(t)$ denote the set of virtual batteries which are not full at time t, i.e., if $B_i(t) < \phi_i \bar{C}$, then $i \in \mathcal{N}(t)$.

Least Percent First Policy. In this policy, all the incoming charge is allocated to the virtual battery that is most depleted, i.e., the one with the smallest percentage level. If there are multiple virtual batteries that are at the same and most depleted level, then divide the incoming charge between them so that they are all charged at the same rate.

More formally, let

$$L(t) = \left\{ i : \frac{B_i(t)}{\phi_i \max} \leq \frac{B_j(t)}{\phi_j \bar{C}}, i \neq j \right\}$$

$$\frac{dB_i(t)}{dt} = \begin{cases} \frac{\phi_i}{\sum_{j \in L(t)} \phi_j} \frac{dC(t)}{dt} & \text{if } i \in L(t) \\ 0 & \textit{otherwise.} \end{cases}$$

This policy violates the isolation property of virtualization. Suppose that a virtual battery has the least level at 10% when the phone is connected to a power supply. Further suppose its application are actively running, while applications associated with other virtual batteries are not running. As a result, all the incoming charge is allocated to the least one, although it remains at 10%. This behavior prevents the other two virtual batteries from getting charged. And it is contrary to the normally observed behavior where the user expects a battery to pick up charge when the device is charging and applications are not running.

Strict Priority Allocation: As part of the user configuration, the virtual batteries are prioritized in some total order. Allocate all the incoming charge to the highest priority virtual battery which is not already full. That is, let h be the virtual battery in $\mathcal{N}(t)$ with the highest priority.

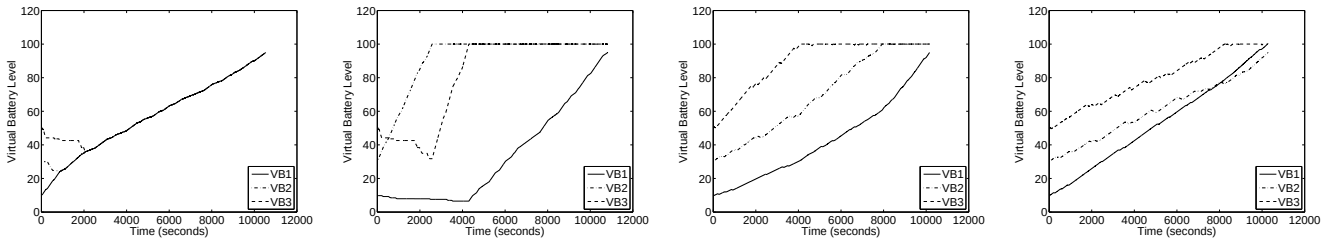
$$\frac{dB_i(t)}{dt} = \begin{cases} \frac{dC(t)}{dt} & \text{if } i = h \\ 0 & otherwise. \end{cases}$$

This policy is based on the viewpoint that each user can prioritize the charging process depending on his/her need or context. Unfortunately, this policy may also result in violation of virtualization's isolation property. As in the case of Least Percent First policy, the behavior of applications in one party can adversely influence the charging of other virtual batteries. For instance, the applications in a high priority party can starve the applications in a lower priority party by continuing to consume charge at a rate which keeps the virtual battery level of the high priority class below the virtual battery level of the low priority class.

Equal Allocation Policy: At each t, allocate the incoming charge equally to all virtual batteries in $\mathcal{N}(t)$. That is,

$$\frac{dB_i(t)}{dt} = \begin{cases} \frac{1}{|\mathcal{N}(t)|}\frac{dC(t)}{dt} & \text{if } i \in \mathcal{N}(t) \\ 0 & otherwise. \end{cases}$$

The primary rationale for this policy is that it does not give preference to any virtual battery. As a result, the ratio of the charging rate of physical battery and the charging rate of any single virtual battery is bounded above by the total number of virtual batteries. Thus, the behavior of applications in other classes cannot prevent a given virtual battery from being charged, in fact, a minimum charging rate can be guaranteed to each virtual battery. This is very desirable feature.



(a) Least Percent First (b) Strict Priority Allocation (c) Equal Allocation (d) Proportional Allocation

Figure 3.3: Illustrating the key characteristics of the charging policies.

In this policy, equal amount of charge is allocated to each virtual battery during charging. Since virtual batteries have different capacities, smaller batteries reach full capacity faster than larger batteries if they both start with the same remaining capacity.

Proportional Allocation Policy: Allocate the incoming charge to the non-full virtual batteries in proportion to their weights.

$$\frac{dB_i(t)}{dt} = \begin{cases} \frac{\phi_i}{\sum_{j \in \mathcal{N}(t)} \phi_j} \frac{dC(t)}{dt} & \text{if } i \in \mathcal{N}(t) \\ 0 & otherwise. \end{cases}$$

In this policy, the incoming charge is distributed to all the non-full virtual batteries in rates proportional to their maximum capacities. This policy also has the desirable feature that each virtual battery can be guaranteed certain minimum charging rate irrespective of the charge consumption of applications in other classes. However, unlike the Equal Allocation Policy the percent levels of all virtual batteries increase at the same rate in this policy.

Clearly, each of these policies have their own pros and cons. The minimum guaranteed charging rate in the latter two policies is clearly a very desirable. However, a user may prefer the first two policies depending on his/her context. A battery virtualization system should probably support all four of the above policies and let user choose a suitable policy based on his/her context.

Figure 3.3 shows the virtual battery levels of three classes when the smartphone is being charged. During the charging process, the smartphone is being used for some applications and hence, consuming some charge. Nevertheless, the key characteristics of the charging policies are clear in the figure. For example, the Least Percent First policy starts by charging the most depleted

battery. After its level equals that of the second battery, both of them get charged proportionally and their levels increase at the same rate. Finally, once all three batteries reach the same level, all three of them are charged proportionally so that their levels increase at an equal rate.

3.4 Implementation

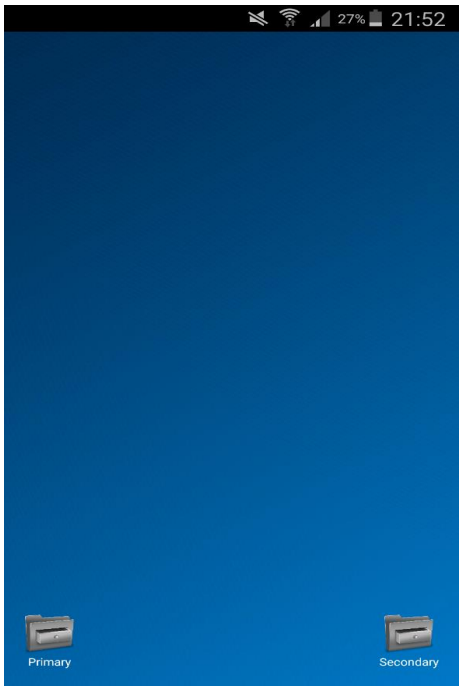
Our first version of *EnergOn* works with the Android framework to be able to terminate a running application It has been used in the evaluation of *Sourcing-Contract*. However, this version is not suitable for a small-scale qualitative user study since it requires installing custom ROM.

We then implemented a version in which *EnergOn* is an launcher application. Users can group applications into folders on the phone screen. Each folder is a class powered by a dedicated virtual battery, and cannot be opened when its battery (the virtual one) is dry. This version cannot kill a class if an application in it runs in the background. Instead, *EnergOn* pops up a dialog to notify the user. Interestingly, the feedback from our user experience shows they usually do not even know such activities exist, and they kill the applications right away when seeing the pop-up message. Snapshots at four important stages are shown in Figure 3.4.

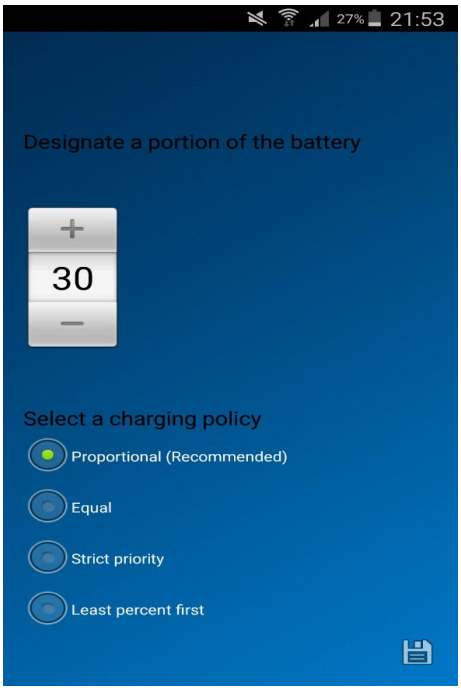
Overall, the current implementation has around 2000 lines of Java code plus the least square implementation by Apache math library (*apache.commons.math*). In particular, *User Interface* takes 1261 lines primarily about Android fragment and view management. *Class Manager* (327 lines) is responsible for maintaining and updating real-time data and user configuration. Finally, *Charging Tracker* (74 lines), *Process Tracker* (121 lines) *Battery Capacity Tracker* (58 lines) and *Per-class Energy Accountant* (152 lines) implements *EnergOn* components as described in Section 3.2.

3.5 Evaluation

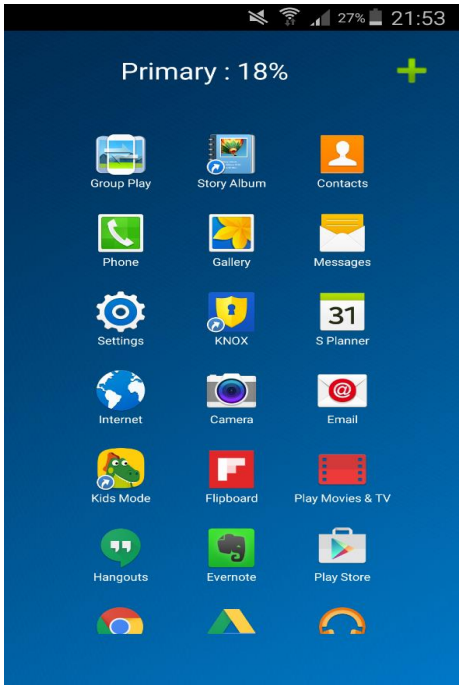
In this section, we present the evaluation results of *EnergOn's* accuracy, adaptability and overhead. The accuracy part consists of evaluation results of sequential and concurrent executions. The evaluation on adaptability studies how *EnergOn* reacts to the variances of application states and environments. Finally, we present the computation and energy overhead of the *EnergOn* system.



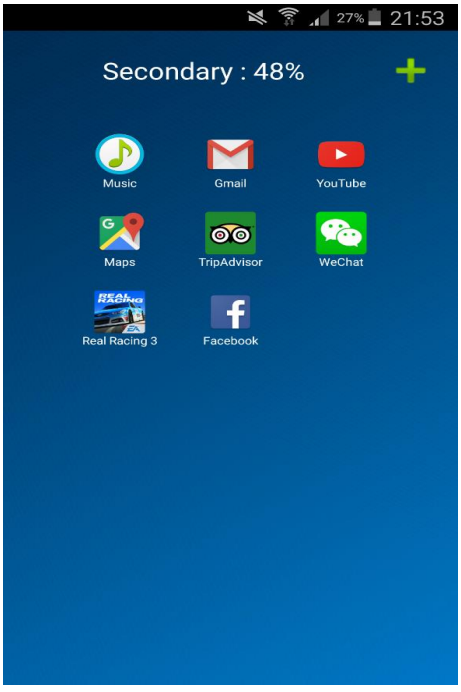
(a) Home page



(b) Configuration

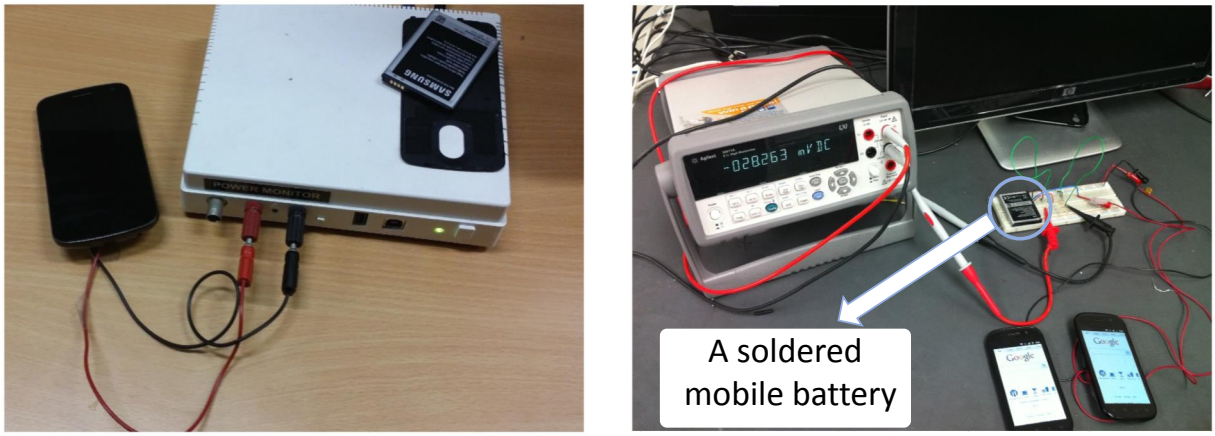


(c) Personal class



(d) Sourcing class

Figure 3.4: *EnergOn* snapshots.



(a) Power monitor setup (b) Digital multimeter setup

Figure 3.5: Two experimental setups.

3.5.1 Methodology

3.5.1.1 Ground truth collection

We use following two setups 3.5 to collect the ground truth measurement of energy consumption used by the evaluation.

- **Power monitor setup (Monsoon [51]).** Because this setup does not have a real battery attached, it can be used only for evaluating the accuracy of the discharging phase. However, it does not need to solder test phones and is very flexible to use. We used Galaxy S5 (S5) and Note 3 (N3) as test phones.
- **Digital multimeter setup (Agilent 34411A digital multimeter (DMM) [53]).** To evaluate the solution in both discharging and charging phases, we soldered two batteries: a Nexus S (NS) phone that has a single core CPU, and a Galaxy S2 (GS2) that is armed with a dual-core CPU. As shown in Figure 3.5(b), the multimeter measured the voltage across a small resistor in series with the battery. These voltage measurements directly correspond to the electric charge drawn or supplied to the battery. When we collected *discharging traces*, the device was operating on the attached battery; and when we collected *charging traces*, the device was connected to a battery charger while the applications were running.

3.5.1.2 Benchmark applications

Our trace collection, either discharging or charging, needs both energy measurements from either of the above two setups and the test application's CPU time at every second (available in the */proc/*[pid of the application]*/tasks* folder). It should be clear that we do not log background services' CPU time since they are shared by classes, i.e., they are not per-class statistics. We then combined both CPU trace and energy trace. Unless otherwise stated the "trace" used throughout the rest of this chapter is referred as to the combination of both. To calibrate the current readings by the logger program, we also deduct a fixed amount of current consumed, which is measured beforehand with the screen turned off. We created the following two sets of benchmarks.

- **Evaluating accuracy of sequential execution.** We use the 15 mobile applications described in Section 3.3.2 and extend the trace collection period of each application from 200 seconds to 60 minutes on average. Any 60-minute trace is a composite that may include multiple sub-traces that are collected at different days. Collectively, we build our database including near 120-hour traces collected from various settings. We then create synthetic energy profiles by mixing smaller trace segments extracted from per-application traces. We will describe this process later.
- **Evaluating accuracy of concurrent execution**: We create 16 benchmark applications for the evaluation of concurrent executions. They are combinations of one out of four background sensing/computing applications (*Audio sensing*, *Camera sensing*, *Gps logging* and *Bluetooth streaming*) and the other one out of four foreground applications (*Web browser*, *WeChat*, *YouTube* and *Navigation*). We compare our evaluation results against other energy accounting solutions.

3.5.2 Accuracy of Sequential Execution

The first task of evaluation is to create synthetic energy profiles each of which consists of many small trace segments extracted from the application traces. For each synthetic profile, we first tagged applications with different class Ids, then started an automated process of picking an application, getting a segment from its trace, followed by appending this segment to the profile with the specified class Id. We also collected the energy consumption when a test device was in

the idle state (the screen was off without running processes in the foreground and background), and considered it as a separate application that can be picked by the above automated process. Other parameters such as the length of each segment were determined from the statistical analysis in the user study work [54].

Overall, each synthetic energy profile mixes a fair amount number of segments. Recall that the segments from the same application might be collected in different settings. Therefore, we believe our synthetic profiles can represent the everyday experience of battery use.

3.5.2.1 Overall accuracy

We first randomly grouped all applications into two classes. For each class size (in our experiments they are 1, 3, or 5), we created 1000 synthetic profiles from the application traces. Figure 3.6 shows the relative error varies with the different group sizes. When there is only one application in each group, the accuracy of estimations is 8.9% at N3 phone and 7.5% at S5. When the group size increases to five (i.e, the diversity within one class becomes larger), the errors become 16.5% and 14.9%.

We now quantify the influence caused by the diversity. We sorted 15 applications by their model coefficients (i.e., the slope in Figure 3.2) obtained when running individually. We then group them using following four allocation policies. (a) *Homogeneity (Homo.)*: allocating the first seven applications in the sorted order into one class, while the rest into the other one; (b) *Heterogeneity(Hetero.)*: allocating odd-number applications in the sorted list into one class, while even-number applications into the other one; (c) *Random:* randomly allocating them into two groups, as we did in the last evaluation; and (d) *Sensing-aware(Sensing)*: allocating 5 sensing-intensive applications into one class, representing the public sourcing class. Figure 3.7 shows the accuracy of four policies when the group size is 5. As shown in the figure, the *sensing-aware* policy performs close to the *Homogeneity* policy.

Last but not the least, we evaluate the accuracy when integrating the charging phase. Each trace is sufficient long consisting of many discharging, charging, and idle segments. Figure 3.8 shows

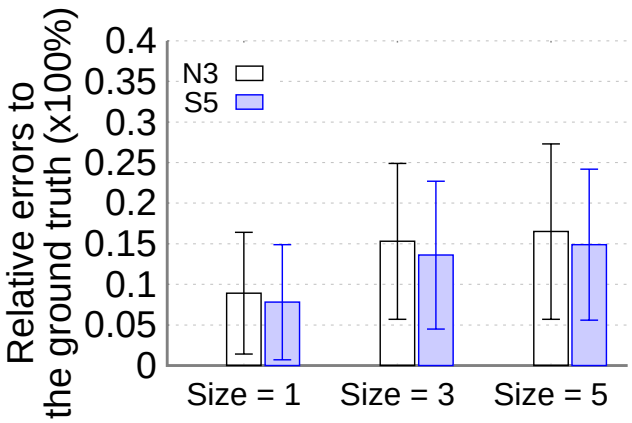


Figure 3.6: Accuracy of the discharging phase.

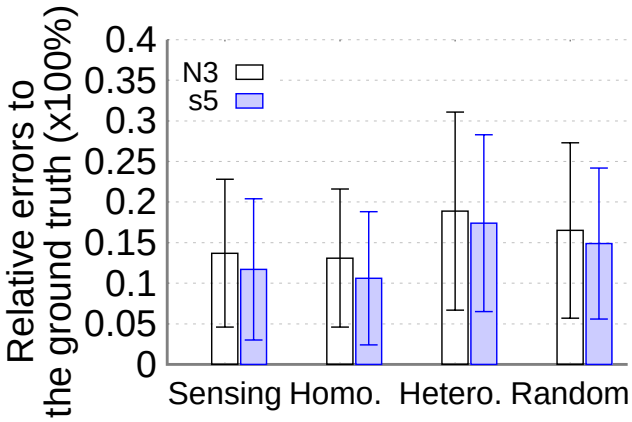


Figure 3.7: Accuracy with different grouping policies.

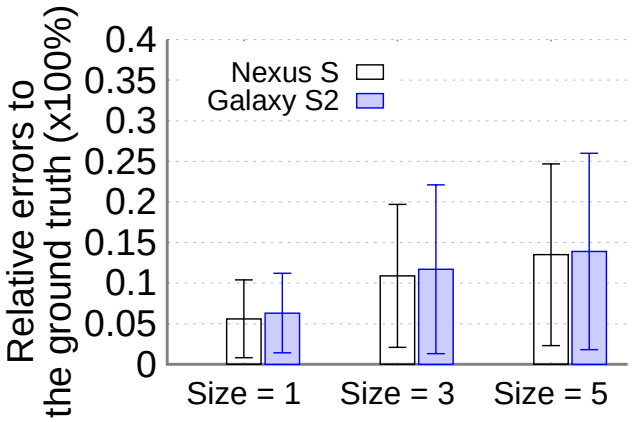


Figure 3.8: Accuracy when having the charging.

the estimation errors over two classes on NS and GS2. The overall results show the estimation of remaining energy is reasonably accurate.

3.5.2.2 Accuracy affected by parameters

We now study the error variances when tuning two important parameters described in Section 3.3: the estimation interval, $\mathbb{T}$, and the number of *Least Square* equations, $\mathbb{N}$. We also examine whether the session length, i.e., the average execution duration, would affect the accuracy.

The estimation interval. This parameter corresponds to the frequency of performing algorithm computation. In the current implementation, we use 1% physical battery level change to trigger the algorithm computation. Typically, for modern smartphone batteries, 1% capacity of physical batteries is near 60000-70000mAs. However, when a trigger can produce triggers at different intervals, this parameter would affect the accuracy. Figure 3.9 shows the accuracy varies from a fast rate (trigger the algorithm every 10000mAs used, about 1-2 minutes when the device is being used) to a slow rate (every 90000mAs used). This demonstrates the *EnergOn* design is suitable only for longer estimation intervals (at least 40000mAs for the current evaluation).

The number of equations. Figure 3.10 shows two opposite trends. When the group size is one or three, the homogeneity inside a group dominates. In this case, more equations bring us more stable and accurate *least square* outputs. However, when group members are more heterogeneous (i.e, group size == 5), a longer period, i.e., more equations integrated, the accuracy decreases.

Average session length. The shorter this parameter is, within one single estimation interval the more trace segments (i.e., activations) are included. Figure 3.11 shows the errors variances when the average session length increases from 30 to 150 seconds. An evident trend is when the group size is one, estimations are more accurate when the session length increases. The trend is not significant as the other two cases.

3.5.3 Accuracy of Concurrent Execution

There are some policies and solutions that work in fine-grained accounting. Albeit targeting on different usage scenarios, we compare against following two approaches.

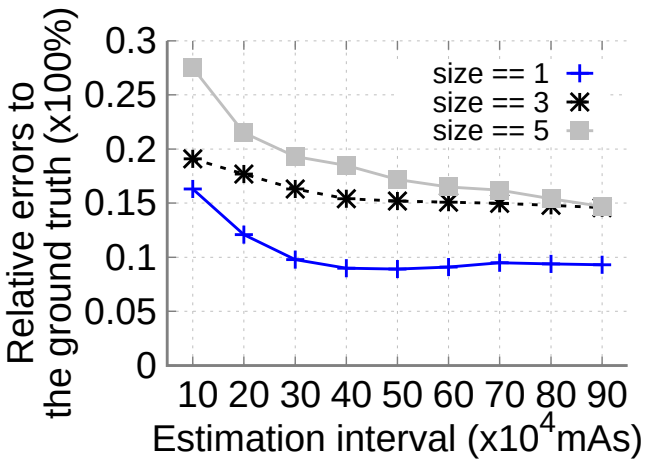


Figure 3.9: Influenced by estimation interval.

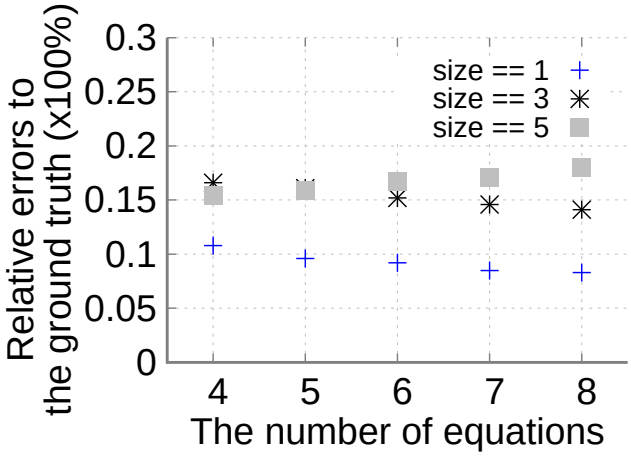


Figure 3.10: Influenced by the number of equations.

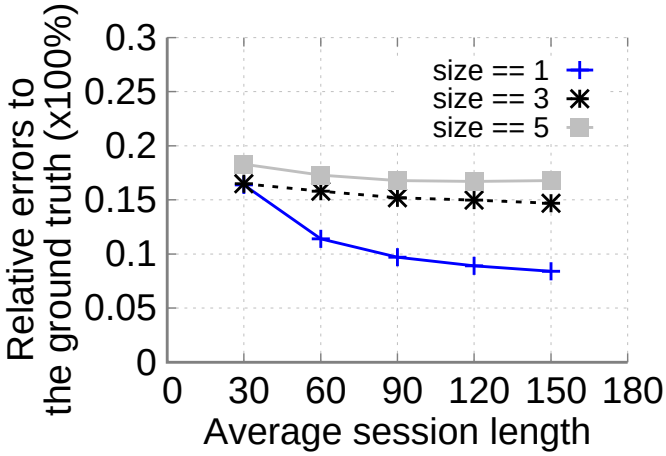


Figure 3.11: Influenced by session length.

1. **Proportional allocation policy**: This is a policy that are usually considered "good enough": portioning the overall energy into two applications in such a way that the portion is the ratio of energy consumption during their standalone executions. Zhong et al [55] uses a similar policy.
2. **Shapley value-based accounting**: This policy looks at the energy accounting from a microeconomic point of view. Dong et al [50] describe the policy based on cooperating game theory, which can be considered as a general form of Nash Bargaining Solution [56].

3.5.3.1 Energy consumption of a concurrent execution

Prior study [31] has shown energy consumption of a concurrent execution is not simply equal to the sum of their individual executions. Our measurements using the Monsoon setup over 16 concurrent benchmarks show consistent results with this observation. Some concurrent executions consume roughly equal (*equal* case) or lower (*lower* case) amount of current than the sum of individuals, while some may consume even a higher amount (*higher* case). In the meanwhile, we also observe for a large number of combinations it is difficult to reproduce the relation across multiple executions due to different hardware and system states, which restates the need of a self-adaptive solution.

We pick 3 out of the 16 combinations, which can evidently and reproducibly demonstrate each of the above three cases: *Browser* + *Audio sensing*, *YouTube* + *Camera sensing*, and *Navigation* + *Gps logging*. For each case, we started with two phases for two individual executions, followed by a third phase that is a concurrent execution. In the *equal* case shown in Figure 3.12, albeit considerable noises due to user actions, the *Browser* consumes 400mA on average; the *Audio* consumes 500mA in the background with the screen turned off; and the mixture of two is on average 900mA. In the *higher* case shown in Figure 3.13, the mixture phase consumes more current because two applications compete for the shared hardware and software resources, leading to inefficient usages. In the *lower* case shown in Figure 3.14, the mixture phase consumes less current because two applications share the GPS polling results without duplicate requests.

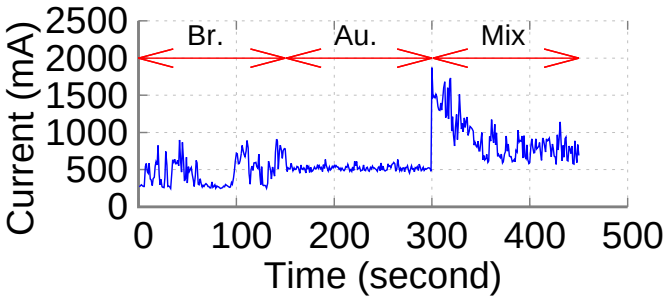


Figure 3.12: Browser + Audio sensing, *equal* case.

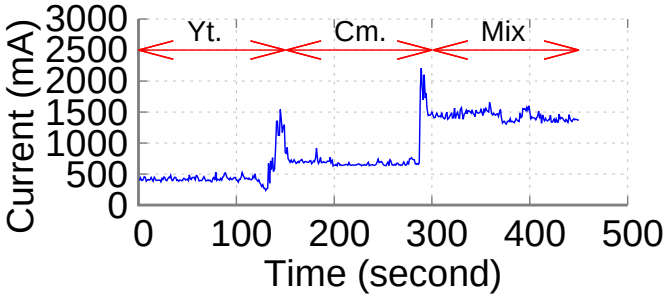


Figure 3.13: YouTube + Camera sensing, *higher* case.

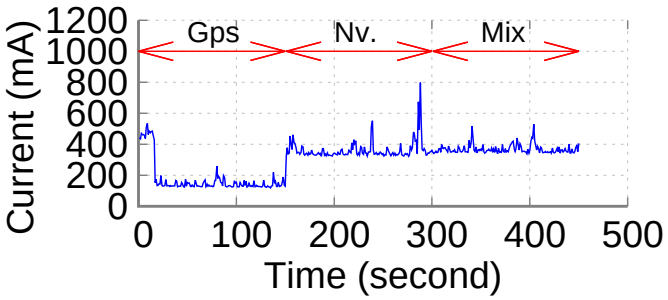


Figure 3.14: Navigation + Gps logging, *lower* case.

3.5.3.2 Result comparison over three approaches

We run the above 3 combinations each for 15 minutes. Figure 3.15, 3.16, 3.17 show the results of energy accounting. Each column in the figure stands for the normalized total charge consumption over the 15-minute execution. Each column consists of two parts, and the percentages are results generated from three accounting approaches. Overall, three approaches produce similar results. Note again that we do not have the ground truth for such evaluation. The purpose of a comparison is not to conclude which policy is more accurate. However, the results show *EnergOn* can achieve a similar level of accuracy without complex algorithm and fine-grained measurements required by other approaches.

Figure 3.18 shows the comparison results across all 16 combinations. We plot both *Shapley* and *EnergOn* against the results the proportional approach. Overall, the difference of accounting results across three approaches is small.

3.5.4 Adaptability

The above evaluation reveals, for individual executions, the short-term variances albeit considerable do not greatly affect the long-term linear trend. If there is long-term variances, however, the linear model needs to be sequentially updated. For example, when a user turns off the screen for a long time, the slope of the *least square* fitting changes significantly. Figure 3.19 shows the slope of an *Audio sensing* application decreases after turning off the phone screen. Similarly, long-term variances also include moving from an indoor office (with Wi-Fi connection) to an outdoor place (with cellular connection), phone getting hot after a long time usage, etc.

In fact, there would be a huge number of application states within one application, causing the model coefficients to vary with time. Figure 3.20 shows a simple example where the current measurement varies when the *Audio* application is altered with different sampling frequencies from 8K to 44.1K as well as different computation intensity related to respective sampling frequencies. We observe that the 44.1K case can consume 130mA on average more than the 8K case. Both *Proportional* and *Shapley* approach needs to know the charge consumption in the mixture scenario and individual scenarios. The existence of various application states makes them not practical to

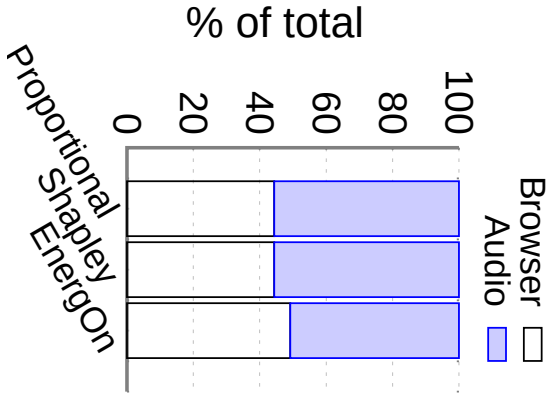


Figure 3.15: *equal* case. Energy accounting results generated by three approaches.

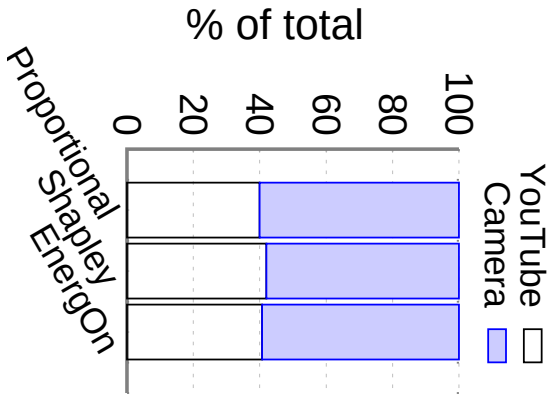


Figure 3.16: *higher* case. Energy accounting results generated by three approaches.

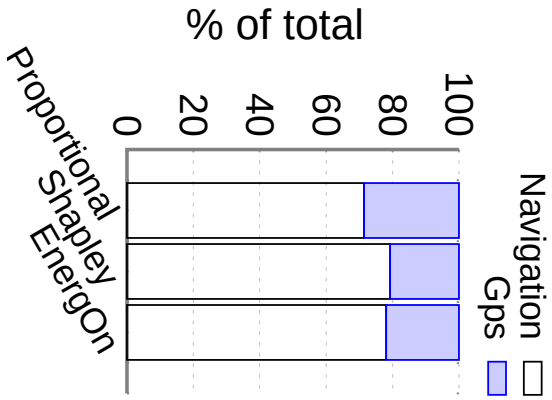


Figure 3.17: *lower* case. Energy accounting results generated by three approaches.

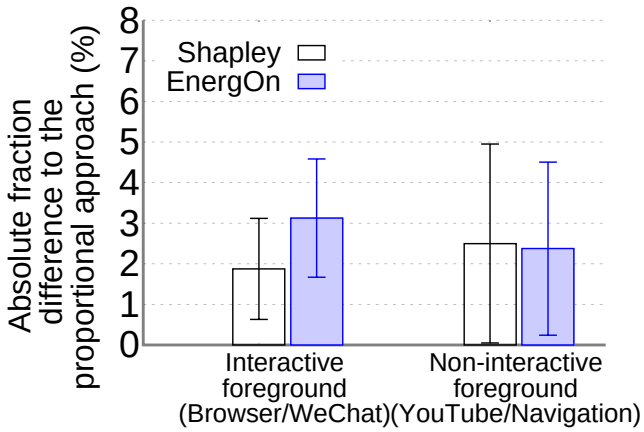


Figure 3.18: Overall comparison of 16 combinations of concurrent executions. The plot only includes the fraction comparison of the foreground application. We separate combinations that have interactive foreground applications from those with non-interactive foreground application.

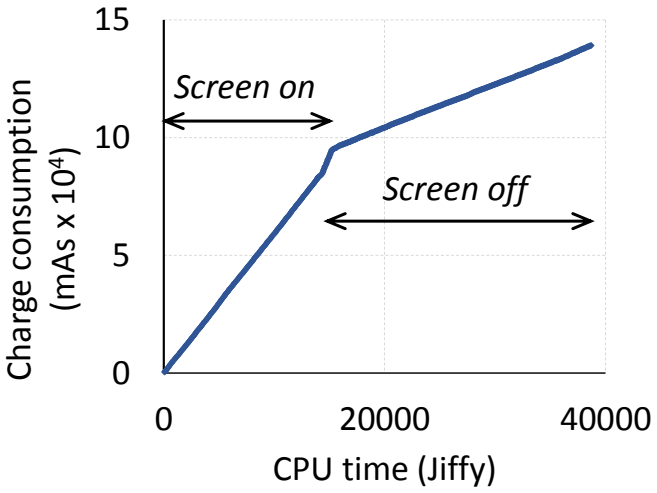


Figure 3.19: The *least square* slope of the audio sensing app changes after the user turns off the phone screen.

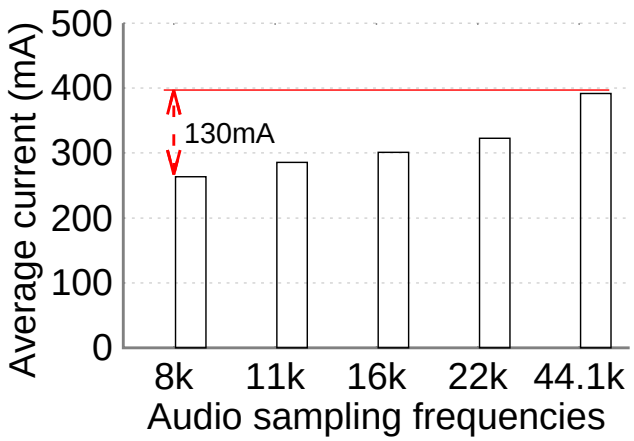


Figure 3.20: Application states of the *Audio* app affect the current measurement.

measure current in all cases. In contrast, Figure 3.21 shows *EnergOn* sequentially updates model coefficients to react to the application state change. In this experiment, both *Audio* and *Browser* are running concurrently. When the 44.1k sampling frequency is used, *EnergOn* increases the fraction accounted for the *Audio* by around 5%.

We observe the *least square* regression is affected during the switch from a session of individual execution to another session of concurrent execution, and *vise versa*. In fact, when there is a second application added, the relation of CPU time of two classes significantly changes: a part of them is obsolete but still involved in the *least square* computation. They cannot represent the present energy usage relation across two classes. This "phase transition" leads to a time window in which the regression results in not stable. This can also be observed in Figure 3.21 where the change of application states make the regression output fluctuating. A naive solution is wait for several more observations and compute the results using only the CPU observations valid in the current phase. However, we examine whether the history CPU observations, by the same two applications, can be reused. For this purpose, we conduct an experiment over three days. On each day, we collect the CPU time observations when the *Audio* and *Browser* are running concurrently for two times with one hour interval without turning off the device in between. Finally, we mix such history data with the present data, and generate fraction estimations. Concretely, for each model update, we use 3 obsolete and 3 fresh observations to make an estimation. This experiment reveals how

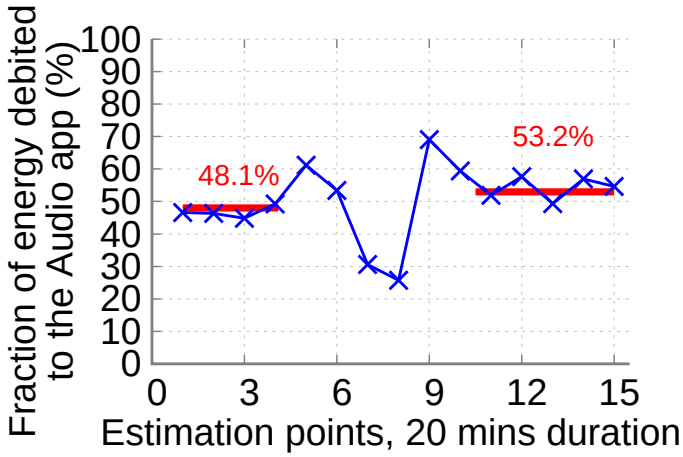


Figure 3.21: Model coefficients (as well as the fractions) are adaptive to varying application states.

EnergOn would perform when there are not sufficient number of equations to run regression. Over a 15-minute time window, we compare such results with that computed by 6 fresh observations (Thursday, left bar) that is normalized. Figure 3.22 shows that the observations collected from Tuesday and Wednesday should not be used on Thursday, while the observations collected one hour ago, when mixed with fresh observations, can be used as an approximation.

We understand that this is our initial findings given that we have not collected enough CPU observations for more comprehensive evaluation. In the future work, we will study on further stabilizing the *least square* outputs.

3.5.5 System Overhead

Figure 3.23 shows the current waveform of energy spendings of doing two consecutive model computations. The peak current is near 330mA and the latency of performing estimation once is 600ms. Overall, both the energy overhead and computation latency are negligible. It would be expected when we shorten the estimation interval $\mathbb{T}$, the energy overhead and latency increases. However, taking the average over an extended time window, *EnergOn's* energy cost is negligible.

3.5.6 Training Cost Vs. Bootstrapping Latency

EnergOn does not employ an offline training process, and thus there is no training cost and time required for power-model generation. However, it has a bootstrapping latency when the user starts

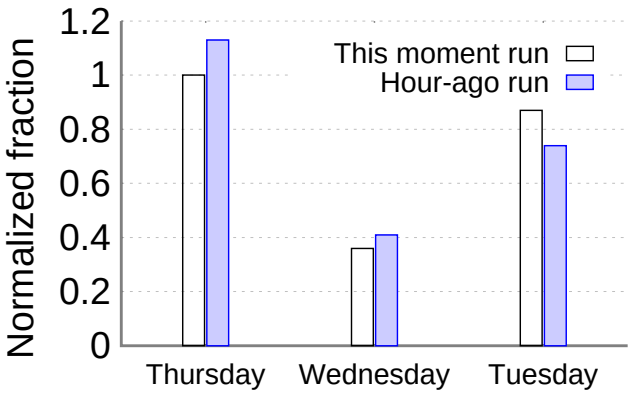


Figure 3.22: Effectiveness and results of using history CPU time observations.

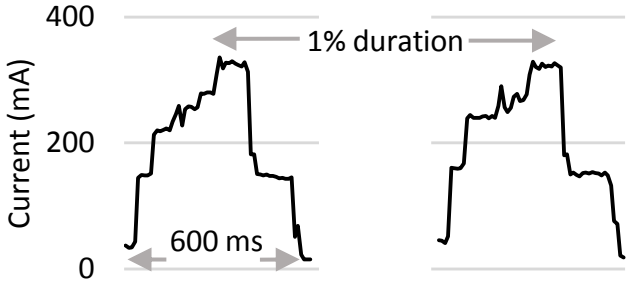


Figure 3.23: Energy overhead of performing two consecutive model computations over a 1% window.

to use it. The latency is to wait for enough CPU time observations to do *least square* regression. This is needed just when the user starts to use it in the first time. The latency would be $\mathbb{T} * \mathbb{N}$. Once model coefficients are built by the initial execution, it can continue to generate new coefficients.

3.5.7 Qualitative Study

We invited 8 Android users to use *EnergOn* for 7 days. The users' devices are all recent Android devices. Seven devices have quad-core CPU and one has octa-core. Their systems are at least 4.0 version of Android (Galaxy S3), and the battery capacity is from 2100mA (the smallest) to 3200mA (the largest). Since *EnergOn* is implemented as an application, it was installed on these devices without any modification.

Before installing the application on our users' devices, we explained the goal and key features of the *EnergOn* application. At the end, we asked each of *EnergOn* users to answer self-reported questions on whether they felt the *EnergOn* application successfully implements the baseline features. We summarize the results and raised issues as follows.

1. All users agree *EnergOn* can control energy usages of a class of applications, and successfully isolate the energy share of individual application classes.
2. All users agree the virtual battery level reports appear to be reasonably accurate.
3. Over the week, none of users report over-heat situations. When there is not any application running, the physical battery level drops at a reasonable pace.

Our users also provide simple comments and interesting hints based on their preliminary experience with *EnergOn*. Six users report a borrow-return policy is needed across classes: "*The isolation is a good idea to prevent applications from consuming more battery than they are allowed to. But sometimes I might need an application to work in case of emergency. Could we probably just allow a class to suck battery from the other after its battery is dead?*"

One of our users suggests that the access control can also be based on time: "*The launcher app (i.e., the* EnergOn *tool) should have the option to set the active time, like I set a group of apps that can only access from 8 AM to 5 PM.*"

The first comment is about *battery partition policy* when using *EnergOn*, while the second comment is about *access control policy*. Both comments are valuable for the next-step design of *EnergOn*.

3.6 Related Work

In Section 3.2.4, we have differentiated *EnergOn's* design goal and usage scenarios from that in existing fine-grained energy accounting work and hardware-based power modeling work. In this section, we discuss other related work in terms of following two aspects.

OS/machine-level energy management. In a virtualized system, each system user (e.g., an Operating System) may have its own needs and policies for managing its energy consumption. Some of their management mechanisms require privileged accesses to real hardware power states. The system-level energy management solution thus needs the "hypervisor" (i.e., Virtual Machine Monitor) below the system users to limit the direct executions, and coordinate various power management policies. Nathuji and Schwan propose a solution [57] for the server domain. Similarly, Cao et al [58] aims at supporting power management in a sensor network.

In contrast to system users in the above scenarios, mobile applications do not have sophisticated energy control strategies (e.g., kill itself to save energy for the entire device). Consequently, it is the end user that coordinates energy usages across applications over time. *EnergOn* is an attempt to simplify such user-managed approach.

Some research work trade the application performance and user experiences for energy saving [59, 60], while JouleGuard [61] maximizes the performance under a given energy budget. e-Doctor [62] detects abnormal energy usages of mobile applications. Badam et al [63] propose software-defined batteries, which is for system designers to integrate heterogeneous batteries. *EnergOn* is for end users to manages multiple "virtual" batteries with the same chemistries.

Design for energy efficiency. Energy-aware design occurs at all layers of the software stack. There has been an extensive body of research on developing energy-efficient design for mobile systems

and applications [64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 31, 75]. These work are orthogonal to *design for energy assurance.* We now describe two examples.

Many energy efficiency design save energy consumption by utilizing the hardware power state information. In PCS [31], for example, the authors target on energy-efficient data collection on mobile platforms. They observe it is more energy efficient for a data collection task to execute concurrently with a user-activated application, saving the wake-up energy of hardware components. Therefore, they strive to figure out when personal applications would be started by the user. We do observe similar behaviors in the scenarios of concurrent execution. However, *EnergOn aims at accurately accounting for energy consumption by concurrent applications, and can be complementary to the PCS technology.* Lentz et al propose that waking up all hardware and software components for one task is a waste. They create a drowsy state for the OS level power management [76], in which only a necessary set of components would be activated.

Other energy measurement approaches. NEAT [77] and PowerBlade [78] are recent work on high-accuracy hardware power management solutions. In the cloud computing context, a linear energy consumption model for power metering of virtual machines in cloud computer is proposed in [79]. Specifically, the energy consumption of three key resources (CPU, memory, and disk) are each modeled as linear function of the time utilization of that resource by the corresponding virtual machine.

3.7 Conclusion

In this paper, we presented design, implementation and evaluation of the *EnergOn* system. *EnergOn* enables different application classes to share the battery resource efficiently and predictably. The growing trend is that mobile users today use the same device to navigate their work life, personal life, disparate social circles, etc. and use multiple diverse applications to do so. These applications intermingle with each other in random patterns as the mobile user furiously multitasks across all of her mobile personas. As this trend accelerates, we believe that the capability to share battery charge predictably across multiple application groups will become crucial. Our

user study though preliminary, indicates that mobile professionals are positively inclined towards *EnergOn*.

We first sketched out several new requirements for a user-managed energy management tool. We then formalized an online learning algorithm that can dynamically learn energy usages at application class basis. Further, by reducing the estimation interval to a human-perceived speed, *EnergOn* can accurately estimate per-class energy consumption with milliwatt-level energy overhead. The most important advantage of *EnergOn* is it does not rely on a hardware-based power model training. This enables *EnergOn* to be highly flexible and portable. Evaluation results show *EnergOn* achieves a comparable level of estimation accuracy with the state-of-the-art energy accounting solutions in user-perceived management granularity. We evaluated the operation of *EnergOn* over a few days to understand the preferred usage models as well as use the feedback to design a better system.

The main limitation of the current *EnergOn* prototype is, the implementation that can kill a background application needs to work with the middleware code, which makes a complete, large-scale usability study not possible. We are limited to a workaround implementation in order to perform the preliminary user study. Also, albeit 15 applications, most of them are well-designed commercial applications used by millions of users. These applications have already optimized device resource usages in great details. In the real world, however, mobile application codes may not be resource-efficient. In the future work, we will investigate in more depth on evaluating the system in the wild. We evaluated three class scenarios that have similar accuracy as two-class cases. However, an interesting future work would be to study if abundant battery information increases the management burden.

Chapter 4

GameOn :
Towards Building Familiar Stranger Social Network

4.1 Introduction

In this chapter, we shift the focus from the public sourcing type of collaboration to the public gaming type of collaboration. Unlike public sourcing where individuals' willingness to participate is a concern, public gaming is a type of collaboration that people seek for to build familiar stranger social network. A typical familiar stranger setting is on public transport. In many dense crowded urban cities (which are common in Asia and Europe, and include some US cities like New York City and San Francisco), the cost of driving tends to be quite high in terms of traffic, time taken, aggravation, and parking availability, etc. As such, a significant fraction of the population in these cities take public transport for their daily commute.

This commute period is a natural "down time" where the commuter can be engaged. Currently, many commuters spend the time by sleeping, reading something, or using their phones to check email, browse the web, chat with friends, watch videos, listen to music, or play games. We also observed that an increasingly large fraction has access to smartphones that have the performance and networking capabilities required for mobile game playing. We thus hypothesize that these commuters could benefit from playing spontaneous multiplayer games, to ease the commute boredom and potentially build more social connections.

Thus, we present *GameOn*, a system for allowing public transport commuters to engage in multiplayer games with fellow commuters traveling on the same bus or train. The key technical challenges we overcame were:

High 3G/LTE latencies. This causes serious lag and playability issues in multiplayer games (especially in the near real-time games like shooting or racing games). We overcame this by using p2p networking solutions for the actual game plays.

Identifying the appropriate p2p networking mechanisms. As we show in Section 4.4.4, Bluetooth [80] does not work well for this use case. Instead, we used Wi-Fi Direct [81], a relatively new Wi-Fi mode optimized for p2p communications, which is now a standard feature on new smartphones, as our communication medium. We showed performance characteristics of both Bluetooth and Wi-Fi Direct in various p2p game scenarios.

Matching game players in an efficient way. A drawback of pure p2p solutions is that neighbor discovery can take a long time. We overcame this by using the insight that all the passengers still have Internet connectivity via cellular connections — albeit with high latency and low bandwidth. We leveraged this and used a central server to perform the matchmaking of players. This also allowed us to match players based on various pre-collected player information such as skill levels, travel times, and other preferences.

Working with minimal modifications to existing systems. For *GameOn* to be successful, it should be as backward compatible as possible. As such, we designed it to run as a normal application (root access is not needed) with minimal changes needed to existing applications. For instance, we retained the existing client-server models used by most existing games to minimize code changes. Thus, one smartphone will have to serve as both the master server as well as a client — we show that the energy overhead for the server phone is still quite acceptable. We also intentionally kept *GameOn* as simple as possible to make it easier to deploy, debug, and explain to end users.

Overall, we make the following contributions:

- A detailed analysis of the public transport travel times. In particular, we show how long passengers are co-located on the same train or bus (which is the shared time when they can play a game together).

- A detailed comparison of the efficacy of Wi-Fi Direct and Bluetooth as communication mediums for playing multiplayer games. We show results from both in-lab synthetic experiments as well as real-world experiments conducted by playing games on actual public trains at various times of the day.

- A detailed description of the design and implementation of *GameOn*. This includes a discussion of how the *GameOn* matchmaker can be extended to support many more metrics (such as co-location times and connection stability) to enable spontaneous p2p games, beyond ping times commonly used by existing matchmakers.

- An in-depth evaluation of *GameOn* that comprises of both micro benchmarks involving synthetic evaluations of various system components as well as real-world tests involving actual game play, using three different popular games, on a public train (at various times of the day). The games chosen were *OpenArena* [82], *Racer* [83], and *2048* [84], which represent the shooter, car racing, and casual game genres, respectively.

4.2 Motivating Scenarios

Jill is heading to school and her regular commute involves a 25 minute train ride. She boards the train and settles in for the somewhat long journey. She starts using her smartphone to do her regular routine — check emails, browse news articles, facebook posts, and videos tagged by friends. However, she quickly finishes all of these and realizes that she is still 20 minutes away from her station and she is getting bored.

Fortunately, she remembers about that new application, called GameOn*, that her friend asked her to install. She starts* GameOn *and sees that 3 people around her are interested in playing Quake III multiplayer (which is setup to require at least 4 people). She expresses her interest in playing the game. Within seconds,* GameOn *starts a server on one of the 4 phones, and automatically connects all the 4 game players (using their anonymous in-game IDs) to the server using Wi-Fi Direct and the game starts. 10 minutes later, the game concludes as some of the participants get*

off the train. Jill is happy with her performance and wonders who she was playing with (that info is not revealed).

She realizes that she still has ≈ 10 minutes left and she decides to see if a quick round of 2048 (a puzzle game) is possible. She re-starts GameOn *and specifies that she is looking for one other 2048 player who will alight at the same station. Within seconds, she is connected with another anonymous player (on the same train) and the game starts. Jill finds the system just assigned a player who played with herself yesterday. The game continues until Jill reaches her train stop at which point she ends the game, gets off, and goes to her classes happily.*

The above scenario motivates the entire design of *GameOn*, as the first step to build familiar stranger social network. In Section 4.3, we first show that passengers spend sufficient shared time on public buses and trains. We then present the design, implementation, and evaluation of *GameOn* in the remaining sections.

4.3 Is GAMEON even Practical?

To support the above scenarios, we require two pre-conditions to be true as follows: First and most importantly, commuters must be on the same train or bus long enough for a shared game session to be feasible. Prior work [85] has published the minimum game length at about 10 minutes. Accounting for the overheads of settling onto the bus/train and allowing for time to finish reading emails, news sites, etc., we pessimistically need commuters to be co-located with a large number of other commuters on the same train or bus for at least 20 to 25 minutes for a *GameOn*-like system to be plausible.

Second, commuters must have the interest to play games while on buses and trains with others, probably random commuters. Fortunately, statistics [86] show that games are the most popular applications downloaded from any app store and this popularity increases as the population gets younger. In addition, multiplayer games tend to be the most engaging of all game types. Thus, we believe that the desire to play multiplayer games is present in a large fraction of the commuting population.

4.3.1 Concern 1: Commute Times

We pick Singapore as an experimental place for *GameOn*. Singapore is a small country of about 715 square kilometers (≈60% the size of New York City) with about 5.3 million inhabitants. It has a modern integrated public transportation network of trains, buses, and taxis (not considered for this analysis). The bus network uses about about 360 bus routes to serve over 4,800 bus stops while the train network comprises of over 120 stations across 5 main lines. In total, the buses and trains handle over 6 million trips per day [87].

Singapore uses a NFC-based store value card system to pay for bus and train rides that requires every commuter to tap their NFC cards at both entry and exit before the actual fare is computed based on the distance traveled. This is different from fixed rate systems used elsewhere, such as the New York City and Paris subways, which only require a tap on entry. This requirement to tap in and out makes it possible for data analysts to know exactly when a particular NFC card has entered or exited a bus or train station (even though the owner of the card is unknown).

In this section, we present rigorous analysis of its transportation system data (Section 4.3.1) along with summary data from other countries (Section 4.3.1.3), and show that a sufficient long commute times are very achievable in practice.

4.3.1.1 Singapore Transportation Data Set

We used three months of bus and train entry and exit data (from November 2011 to January 2012) obtained from the Land Transport Authority of Singapore [88]. For every public bus, we had the time and location (bus stop number) where every passenger boarded and alighted. For trains, we have, for every train station, the exact time when a commuter entered and left that train station. With these two sets of data along with the publicly available train/bus timings and route maps, we can quantitatively determine the average commute time needed to reach any destination. Table 4.1 summarizes the data used for this analysis.

For the purpose of this analysis, we used the Singapore Management University campus as the final destination and calculate the commute time statistics needed to reach the campus from any location in Singapore. Because the university is located downtown, it is very well connected and

Bus Data			
	Nov. 2011	Dec. 2011	Jan. 2012
Total # of Records	100,521,633	100,732,193	105,449,970
Unique Bus Routes	353	353	353
Unique Bus Stops	4873	4873	4873
Unique Commuters	3,910,636	4,364,309	4,202,792

Train Data			
	Nov. 2011	Dec. 2011	Jan. 2012
Total # of Records	62,272,880	63,655,069	63,092,608
Unique Train Stations	127	127	127
Unique Commuters	4,210,625	4,051,357	4,384,240

Table 4.1: Summary of Public Transportation Data

served by 3 different train stations and 43 different bus routes across 7 different bus stops. Also, when performing our analysis, we only considered the most direct routes to the campus that did not require switching between trains to buses and *vice versa*.

4.3.1.2 Quantitative Analysis Results

Table 4.2 shows the results of our data-driven analysis for both trains and buses across all 5 weekdays for both peak hours (7.30 a.m. - 9.30 a.m.) and off-peak hours (9.31 a.m. - 5.59 p.m.). Note that we only consider the morning peak period as the evening peak period will not have too many people coming to campus.

The data shows that the average time spent on a bus is about 17 minutes with a fairly high standard deviation (numbers in parenthesis). For trains, the average time is about 26 minutes with a reasonably large standard deviation as well. This matches well with reported data [87] that states that trains are the preferred option for longer routes. However, even though these numbers look low, many commuters experience higher commute times as they need to take more indirect routes that involve multiple trains/buses for their commute. We show this through a survey in the

	Commute Time (mins)			
	Bus		Train	
	Peak	Off-Peak	Peak	Off-Peak
Mon	17.4 (9.2)	18.5 (10.6)	24.1 (11.1)	23.3 (13.4)
Tue	16.5 (11.8)	17.1 (9.8)	27.5 (12.9)	21.1 (13.7)
Wed	17.0 (10.4)	17.9 (10.1)	25.6 (11.2)	20.5 (13.3)
Thu	16.9 (11.0)	17.1 (10.3)	27.9 (12.9)	20.9 (13.6)
Fri	17.1 (10.7)	17.4 (10.2)	25.6 (11.2)	21.1 (13.6)
All	16.9 (10.8)	17.6 (10.2)	26.5 (12.1)	21.4 (13.5)

Table 4.2: Average Commute Times for Buses and Trains. Numbers in parenthesis are the standard deviations

following section where the majority of respondents reported high commute times with more than one transfer.

4.3.1.3 Commute Times in Other Major Cities

We additionally describe analysis of the commute times observed in other urban cities (from prior work and online sources). Table 4.3 shows our findings. What we observe is that commute times in other cities are higher, and thus *GameOn* might also prove to be useful in other cities.

4.3.2 Concern 2: Willingness

In addition to the data driven analysis presented above, which is completely game agnostic, we also surveyed students and working professionals to obtained their willingness to play multiplayer games while commuting. Out of 118 participants, 90 participants (76%) stated that they played mobile games with 67 (57%) saying that they played mobile games while commuting.

64 participants (54%) answered yes to the question “Are you interested in playing multiplayer games with other commuters traveling in the same bus/train/car?”. When asked why they wanted to play these games, the answers provided were “Ease the boredom during the commute” - 44 (37%),

City	Average One-Way Commute Time (minutes)
London	39.5 [89]
New York	40.0 [90]
Montreal	38.0 [91]
Toronto	39.5 [91]
Tokyo	66.0 [92]
Seoul	53.0 [93]
Hong Kong	46.0 [94]
Taipei	37.5 [95]
Beijing	97.0 [96]
Delhi	42.3 [97]
Mumbai	47.3 [97]

Table 4.3: Average Commute Times for Other Cities

"Potential to meet more people who share similar interests" - 34 (28%), "Thrill of competitive challenge inherent in multiplayer gaming" - 33 (28%), "Other" - 2 (2%).

For the 54 participants (46%) who were against the idea, the most common reason offered (via a free form text box) was the unwillingness to pay for 3G/LTE bandwidth just to play a game on the train – "Dataplan consumption and slow/connectivity issue when in train". They also felt that the 3G/LTE speeds were not good enough for gaming – "You need a solid connection when playing such games during commuting.". Another strong opinion raised was the fear that playing with nearby strangers would impact their real world comfort levels – "Do you really think we are that open to play with strangers standing right next to us?". Finally, some participants feared that the game experience would be bad due to poor player quality or players leaving abruptly.

Overall, the survey results indicate that there is potential for *GameOn* to be successful. However, to become even more accepted, *GameOn* must reduce the use of 3G/LTE bandwidth that (i) may have high usage charges in some countries, and (ii) might have connectivity issues in certain parts of the transport network. *GameOn* overcomes this by using completely local bandwidth provided by Wi-Fi Direct to support the various games. Thus, it does not incur any charges and is much less likely to have connectivity issues. In addition, the survey shows that the matchmaking component also needs to take into account the physical proximity of people, historical collocation

records, the expected trip length for each person to avoid game interruptions caused by people leaving.

4.4 Design Goals & Assumptions

In this section, we present the design goals for *GameOn* along with our assumptions.

4.4.1 Design Goals

The main design goals for *GameOn* were:

- **Provide a smooth gameplay experience**: This is the most important design goal and it permeates all the other goals below. In a nutshell, *GameOn* should add as little overhead as possible to both game players and game developers.
- **Low latency networking with sufficient bandwidth**: A key cause of discontent in multiplayer games is lag caused by network issues. Thus, *GameOn* should not introduce any user noticeable lag or bandwidth artefacts when games are being played. We compared the client to server latencies and energy consumption of LTE, Bluetooth, and Wi-Fi Direct (results shown in Figure 4.1) and found Wi-Fi Direct to have the lowest latencies and the lowest energy consumption. *GameOn* thus uses Wi-Fi Direct for the actual game plays while using the cellular Internet connectivity only for the matchmaking process (a low bandwidth latency tolerant task that requires history tracking)
- **Easy and effective matchmaking**: Commuters should be able to easily express their game interests and also easily find games that they can join. The matchmaker should also ensure that the players in the game do not leave abruptly and that any skill, demographics, or other factors are also factored in, where necessary, when performing the matchmaking. For example, even though *GameOn* enables playing multiplayer games with fellow passengers in close proximity, some players may not want to be matched with players located next to them on the bus/train as they may not want their physical identities to be easily discovered. To support this, we use a centralised matchmaking service, that can track historical performance etc., located in the cloud.

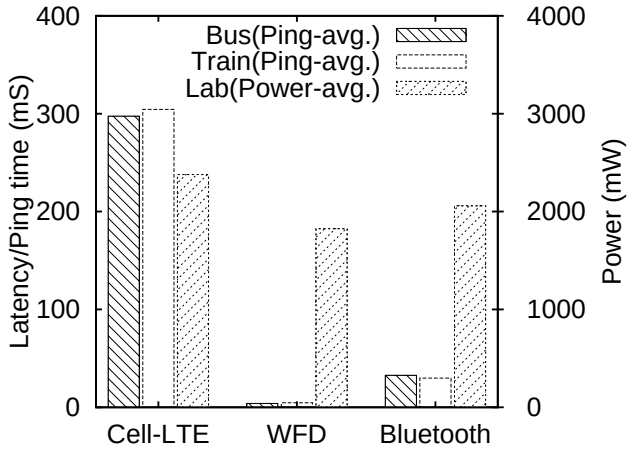


Figure 4.1: Latency & Power Consumption Comparison. All experiments were conducted with a Galaxy S3 & S5 Android smartphones. The latencies were computed by pinging a common in-lab server. The power consumption was measured in our lab using a Monsoon power monitor [51]. WFD == Wi-Fi Direct.

- **Use simple user-space mechanisms**: For *GameOn* to be easily deployable, it has to be a user space component (i.e., no rooting of the phone is required) and it should be as simple as possible (making it easier to explain to end users and more robust overall). In addition, we retain the existing client-server models used by almost all multiplayer games. However, this requires us to dynamically host the server on one of the smartphones of the commuters playing that game. The game and player statistics are then uploaded to the matchmaking service after the game ends.
- **Low energy usage**: *GameOn* should not add any significant energy cost beyond the cost of playing the game itself. In particular, the smartphone that has to host the game should not see a large increase in energy usage.

4.4.2 Assumptions

The assumptions we made when devising a solution that addressed our design goals were the following: (i) We assume that every commuter had access to a smartphone with cellular Internet connectivity. The smartphone was necessary for gameplay while the connectivity was necessary to

use a central matchmaker. (ii) Some changes to the game interfaces may be needed for *GameOn* to be fully operational. In particular, the game will have to report game statistics (in game scores etc.) to *GameOn* so that it can be used during matchmaking and have to use the *GameOn* APIs to send data to/from other p2p clients. Indeed, to demonstrate how easy our APIs are to use, for our evaluation, we converted, with minimal effort, an open source *single player version* of a popular game, *2048*, to work as a multiplayer game using *GameOn*. Finally, (iii) we assume that the multiplayer games will only be played by a small number of players – 2 to 6 players at most. This system is not designed by larger games that involve 10s or 100s simultaneous players. However, there can be multiple games being played simultaneously in the same area.

4.4.3 Overall Architecture

To satisfy the design requirements stated in Section 4.4, *GameOn* was designed to use a hybrid p2p architecture composed of *GameOn* clients interacting with each other using local networking capabilities coupled with a matchmaking service located in the cloud. Figure 4.2 shows the architecture overview of *GameOn*. We focus our discussion on only a few core modules (the shaded blocks in Figure 4.2). Overall, *GameOn* comprises of two components:

1. *GameOn* clients: A *GameOn* client supports various multi-player games that can be played by peers co-located on a train or bus. It has a UI component that allows players to login, specify grouping preferences, and discover co-located peers. When a user starts *GameOn*, peer discovery is started and any discovered peers (along with their performance metrics) is passed to the matchmaker. Upon request, the matchmaker provides the *GameOn* client with the list of playable games and corresponding game hosts. When a peer is already hosting a user's desirable game, the *GameOn* client makes a new game client connection to the peer. Otherwise, it serves as a game host for the user's specified game and waits for other players to join.

The game play is automatically initiated when the required number of players join. During game play, *GameOn* clients form a star topology by default, and all the game packets are relayed through the host device; *GameOn* also supports a multi-hop topology when a host cannot be connected to a client directly due to distance (located on the other side of the train for example). The

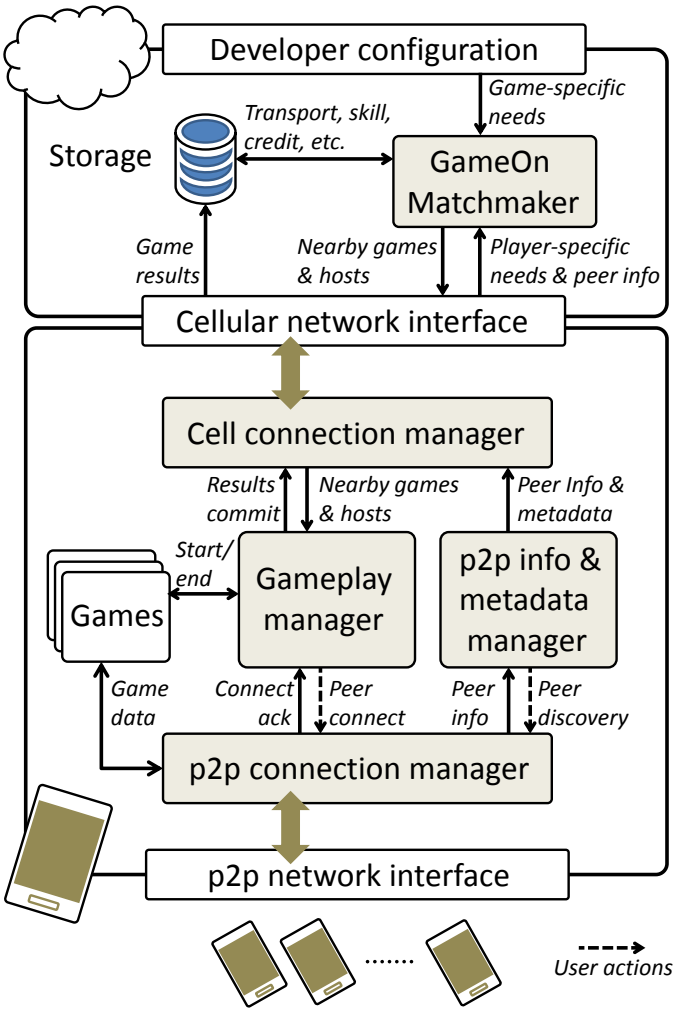


Figure 4.2: *GameOn* Architecture

Gameplay Manager component configures the *p2p connection manager* component by specifying the role of the player in a group and the topology of collocated peers. Note: We do not require any changes to the existing game logic to support this type of p2p game play. *GameOn* wraps around the original networking APIs (Section 4.4.5) used by the multiplayer games and automatically re-routes packets to p2p hosts using either Wi-Fi Direct or Bluetooth. When the game ends, the game results and performance data are reported to the matchmaking server to update its records.

2. The *GameOn* matchmaking server: This server allows *GameOn* clients to find a set of players that are collocated and who will stay on the same bus/train long enough for a satisfying game session. It collects various information required for p2p matchmaking from *GameOn* clients such as the observed signal strength and ping times between peers, as well as the mobility patterns (how long they spend on a specific train etc.) and skill levels (how well they did in previous sessions of a game etc.) of each user (as represented by their mobile phones). We show how the matchmaker can use all available data (mobility history, user preferences, game-specific skill levels, and performance measures) to match the best set of people together for any game request.

4.4.4 Which p2p Protocol is Best on Trains?

The success of *GameOn* depends on having reliable p2p networking connectivity between peers on a bus or train. However, these are particularly challenging environments due to their movement, layouts, and frequent passenger movements. In this subsection, we present detailed performance results to understand the performance of wireless protocols in these environments.

The first key question we addressed was the choice of network protocol. The two main options were Bluetooth and Wi-Fi Direct. Eventually, we chose Wi-Fi Direct as its overall performance, beyond just the better latencies and power consumption (Figure 4.1), was better as explained below.

4.4.4.1 Experiment Setup

To understand how the protocols behave in realistic environments, we conducted experiments using Galaxy S3 and S5 smartphones on a train during three time periods – when the train was extremely full (6 p.m.), normal load (8 p.m.), and empty (midnight). We used the Galaxy S3

(running Android 4.3) as the stationary peer and moved the S5 (running Android 4.4.2) to different adjacent train carriages (up to 3 carriages away) and measured (on the S3), using both Bluetooth and Wi-Fi Direct, the RSSI signal strengths of the S5 and the ping times to the S5. Each train consisted of 3 carriages [98]. Each carriage was filled with numerous metallic objects (seats, hand rails, guard rails etc.) and was 23 meters in length, 3.2 meters in width, and 2.1 meters in height with a very small (negligible) inter-carriage gap. We repeated each experiment multiple times over different days. We do not report any results for buses as the public buses are shorter in length (each bus is about 12 meters long [99]) than a train carriage. Thus, the train is a more demanding environment.

4.4.4.2 Peer Discovery, Connectivity & Density

The first step in connecting phones together in a p2p fashion is to discover them. In our preliminary measurements, we also discovered that just because a device can be discovered does not mean that a successful connection can be made to it. A typical Wi-Fi Direct connection starts with scanning, then group owner negotiation, then provisioning, and finally DHCP. When peers are side-by-side, these steps can be done quickly without packet loss. However, as peers are further away and/or in "noisy" environments, these steps can become harder to complete.

To include the effect of people density in this experiment, we performed it during normal hours (when the train was normally crowded). With this level of crowd, we can assume that the density of people increases linearly as we move further away from the discovery node. To perform this experiment, we used one device as the stationary node and moved another device further and further away (in increments of half a train carriage every time). Both devices then tried to discover the other device. In addition to discovering the device, we also tried to connect to the device after it was discovered. We found that even though both devices could eventually discover each other (taking about 10 seconds) even at a two train carriage distance (about 46 meters), they were unable to actually connect to each other. However, at shorter distances, the two devices could discover and connect to each other. Our experiment results are shown in Figure 4.3.

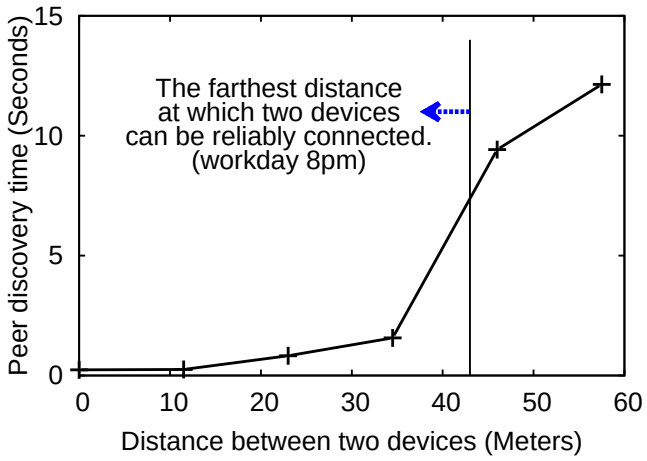


Figure 4.3: Discovery Time vs. Distance (Wi-Fi Direct). We found that discovery times $>$ 10 seconds, corresponding to a peer distance of about 46 meters (2 carriages away) was a good indicator of a peer that could not be reliably connected to. We then shortened the inter-device distance by a meter at a time (roughly) and were able to reliably connect two devices at a 38 to 43 meter range. We found that even very far devices (70 meters away) could be eventually discovered (taking 108 seconds). We omit these long tail numbers from the plot.

The *GameOn* matchmaker has to make decisions about peering without being able to actually check the connectivity between those hosts – at best it knows something about inter-peer ping times. As such, a naive host assignment might pair hosts together who can discover each other but cannot actually connect (because one of the steps involved (probably DHCP) fails). What we discovered, for Wi-Fi Direct, was that the discovery time turned out to be a good predictor of connectivity. In particular, as shown in Figure 4.3, peers that could be discovered within 10 seconds (i.e., before the discovery time shoots up) can be successfully connected to.

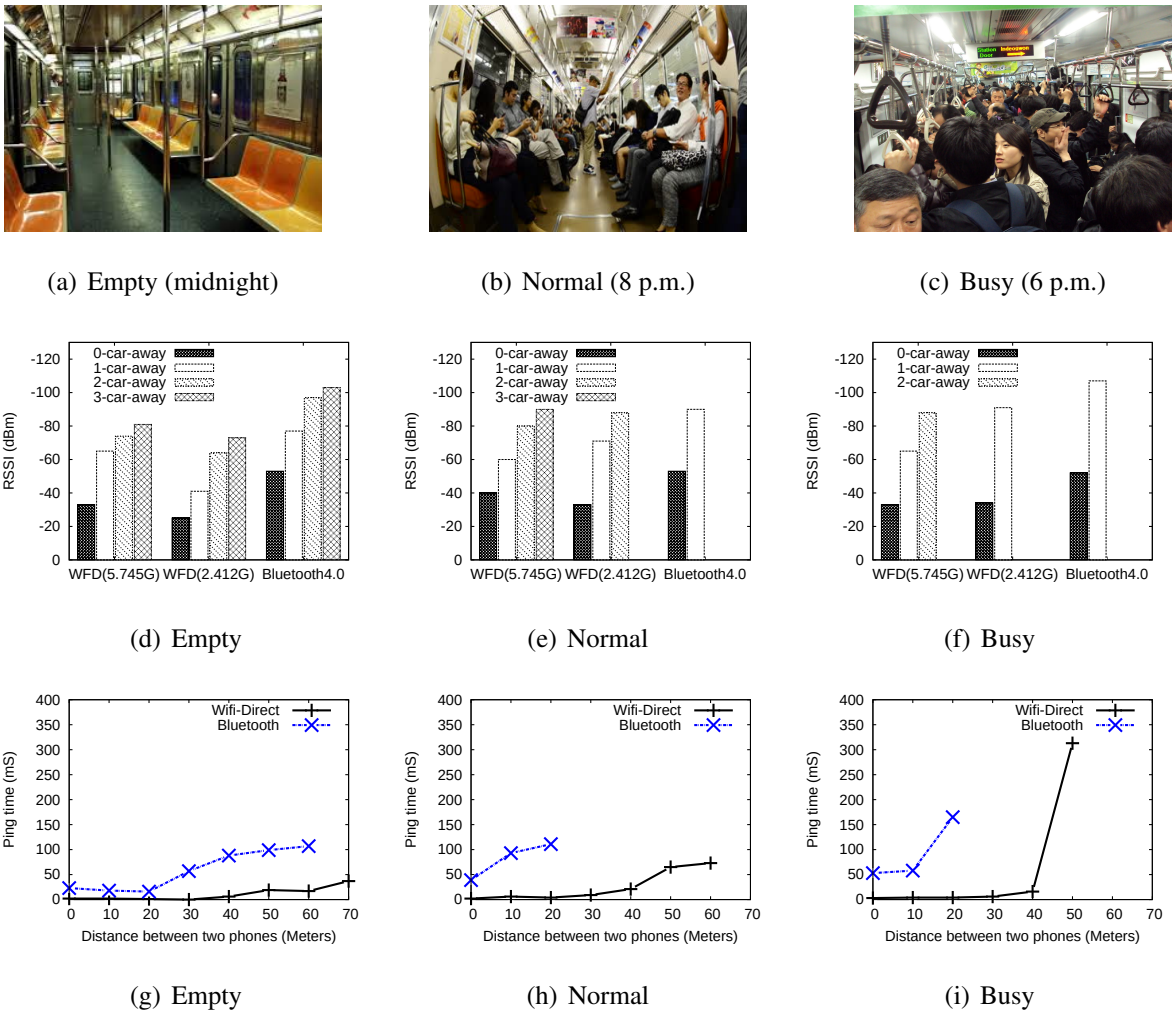
However, even a 10 second discovery time can be too long as every scan is costly in terms of battery usage. Thus we reduced the scan time to 5 seconds to strike a balance between power consumption and finding enough nearby connectible peers. Each peer performs a scan every time it requests a peer match list from the matchmaker. This allows the matchmaker to gradually build a client map for a bus / train without needing aggressive client scanning.

The discussion above is solely for Wi-Fi Direct. We also repeated this discovery and connectivity tests for Bluetooth and achieved very disappointing results. We found that Bluetooth was unreliable beyond 20 to 25 meters. We show the difference between Bluetooth and Wi-Fi Direct in terms of RSSI and ping times in Figure 4.4.

4.4.4.3 Effect of Density on Network Latency

We now investigate the effect of people density on wireless performance – in particular the latency of the connection. This is important as games require low latency network connections. To do this, we picked three different times of the day (corresponding to light, normal, and heavy train/bus use) and four different inter-client distances. We measured the inter-client ping times and also measured the RSSI values. Note: the ping times changed when we repeated this experiment across different days. In the rest of this section, we present the ping times for the worst day.

Figure 4.4 shows how the signal strength and ping times changed as the distance to the peer phone varied. We observe that in all cases, Wi-Fi Direct performs better than Bluetooth. In particular, the second row of results shows the RSSI observed when the stationary phone connects to the



(a) Empty (midnight) (b) Normal (8 p.m.) (c) Busy (6 p.m.)

(d) Empty (e) Normal (f) Busy

(g) Empty (h) Normal (i) Busy

Figure 4.4: Comparison of Observed RSSI & Ping Times at Different Times on a Public Train. The top row shows the state of the train (empty, normal, busy) at the time of the measurement, the middle row shows the observed RSSI values of a peer device by a stationary phone (after connecting to that peer) when the peer device was placed further and further away (by up to 3 train carriages). The bottom row shows the observed ping times on one device (to the other) as the other device moved further away. Missing data in the figures indicates that the other phone was not connectible to or pingable at that distance using that protocol. Each result was repeated multiple times over different days and the averages are shown. We omit the error bars to improve readability as these results are presenting trends (actual values are not important).

moving peer using various protocols. The actual RSSI values are not important (as they fluctuate due to noise etc.). What matters is the pattern and trends.

For Wi-Fi Direct, we could connect using both the 5Ghz and 2.4Ghz spectrums with no clear winner in spectrum choice emerging. We found that, when the train was normally occupied, the maximum distance that a peer could be connected to was 2 carriages away using Wi-Fi Direct. For Bluetooth, the range was just 1 carriage away. When the train was busy, the range of Wi-Fi Direct decreased to just the same train carriage while Bluetooth could only usefully connect to clients very close by (distances greater than 20 meters had very high ping times).

The last row shows the ping times achievable to the connected peer using Wi-Fi Direct and Bluetooth. In all cases, the ping times for Wi-Fi Direct are much lower than Bluetooth. In addition, Bluetooth stops working (the line for the ping graphs stops) at much lower distances than Wi-Fi Direct. For example, on a normal occupancy train, Bluetooth stops receiving pings at about 20 meters while Wi-Fi Direct continues until about 60 meters.

4.4.4.4 Connectivity Issues at Train Stations

Unfortunately, even with Wi-Fi Direct, we found that if peers were 2 or more carriages apart, on entering a station, the process of opening the doors to let passengers embark and disembark resulted in high latency spikes. Figure 4.5 shows this where a peer (located 1 carriage away) experiences constant good ping times while another peer (located 2 carriages away) experiences consistent latency spikes which corresponded directly with the train entering a station, stopping, opening its door, and then leaving (the high latency goes away at this point). We have no current solution other than adding a matchmaking heuristic to not match peers more than one carriage away for games that cannot handle brief latency spikes.

4.4.5 Modifying Games to Work with GameOn

In this section, we describe how *GameOn* support can be added to existing games by making two different modification; 1) support local client-server multiplayer, and b) interface with *GameOn's* networking, matchmaking, and reporting APIs.

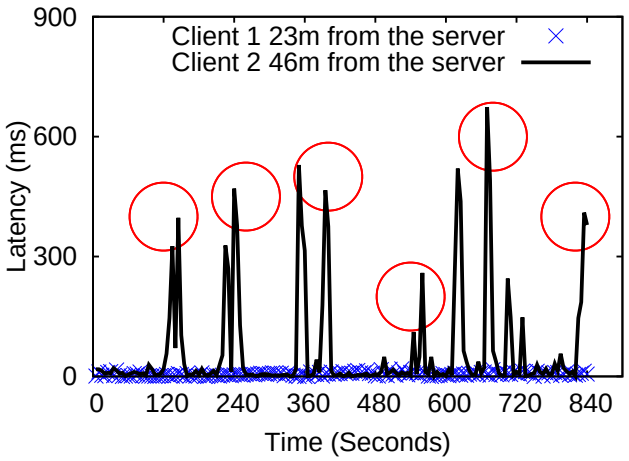


Figure 4.5: Latency Spikes at Stations. During a 15-minute experiment, the train stopped at 6 stations (times at each station are circled). We conducted this experiment during peak hours.

Games	Multiplayer already?	Language	Lines of code added
OpenArena [82]	Y	C / C++	8
Racer [83]	N	Java	86
2048 [84]	N	Java / JavaScript	14

Table 4.4: Three Games Modified to Use *GameOn*

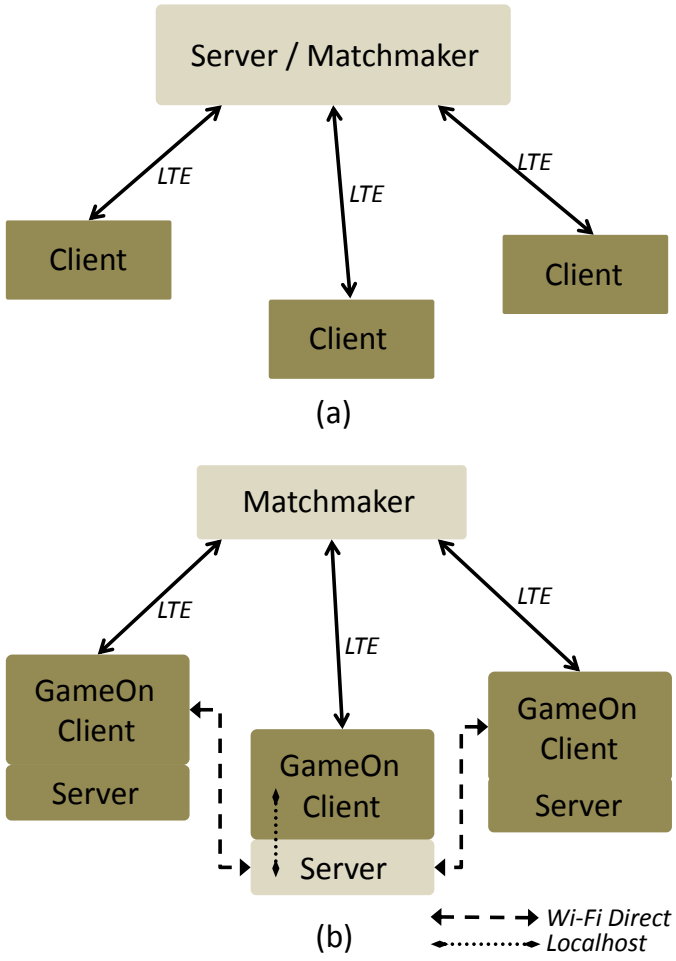


Figure 4.6: Traditional Approach (a) vs. *GameOn* Approach (b). Unlike traditional approaches, in *GameOn*, each peer can serve as a client and also as a server. *GameOn* selects only one peer to serve as the game server. Game traffic is exchanged with peers using p2p connections (usually Wi-Fi Direct).

4.4.5.1 Games Used for Evaluation

The first requirement for any game to work with *GameOn* is for the game to support client-server multiplayer. To make a game multiplayer compatible, it requires creating a server component for the game along with changing the UI, where necessary, to display any multiplayer-specific information. In this work, we decided to use both existing multiplayer games as well as support single-player games to understand the complexity inherent in making different types of games work with *GameOn*.

The three games we used are described in Table 4.4. *OpenArena* was the only game that already multiplayer-enabled with separate client and server components. Even in this case, as shown in Figure 4.6, we need to modify the game to use a local server (that is running on a peer phone and accessed via Wi-Fi Direct) instead of a server sitting in the cloud that is accessed via a cellular link.

Unlike OpenArena, *Racer* and *2048* were single player games that had no server component. For both games, we created a simple server that basically stored and forwarded packets to other clients. To help developers to extend existing singleplayer games to communicate with a game server, we provide two functions to share game state: *sendCommand(String jsonObjectInString)* is used to send client moves periodically, and *updateSnapshot()* is used to receive global game states from the game server.

In all cases, the amount of additional code we had to write was minimal (86 lines for *Racer* and 14 for *2048*). For both games, we did not modify the UI component and just leveraged the existing game code that could already display the output of a secondary player. Currently *GameOn* does not provide any UI modules as these components are very game specific. Instead, *GameOn* focuses on the networking components and provides enough infrastructure (and APIs) to allow game developers to concentrate on the UI and gameplay portions of the game and let *GameOn* handle all the networking bits.

4.4.5.2 Using GameOn Libraries

Next, we had to modify all three games to use the *GameOn* APIs. This required 1) using the *GameOn* matchmaking service, 2) using the *GameOn* networking libraries, and 3) using the *GameOn* game statistics reporting libraries.

The matchmaking service is initiated by the player (in our prototype, the player presses a UI button). This is a single API call in *GameOn* and it sends a request to the matchmaker, using JSON objects, along with the performance measurements from the current client (neighbours discovered, ping times to neighbours etc.). The matchmaker responds with a list of games and hosts. The developer can then use *GameOn*p2p APIs to initiate a game request with discovered clients. Once the game starts, the state sharing APIs described earlier are used to play the game. Finally, the developer has to use the *GameOn* reporting libraries to commit the game results back to the matchmaker (for use in global statistics and future matchmaking sessions). Note: games don't communicate with the matchmaker directly. That functionality is handled transparently by *GameOn*.

The *GameOn* networking libraries handle most networking requests. Internally, the *GameOn* networking logic uses two layers: a physical Wi-Fi Direct group, and a logical game group. In our current implementation, the physical group is built using legacy Android APIs (for backward compatibility), while the logical group is built using TCP/UDP sockets. All status sharing information is exchanged via the TCP/UDP sockets. For simplicity reasons, when a player initiates a new game, our current implementation makes him or her the game server and owner of the logical group. All subsequent players are clients in the group. This logic can be changed, if necessary, to share the server load among other all players.

Overall, all these changes were easy to implement. A single grad student, with no game development experience, managed to modify all three games in less than 2 days each. Most of the time was spent understanding how each of the games maintained its game state (to find the right places to insert the networking and statistics reporting APIs). As shown in Table 4.4, the amount of code that needed to be created was minimal. *OpenArena*, in particular, needed very little code as it already had discrete client-server components.

The goal of the *GameOn* matchmaker is to find groups of commuters on the same bus / train who can play a game together. In such dynamic environments, these formed groups should be chosen so that they are stable – i.e., members don't abruptly leave. For example, a group is not considered to be stable if the elected game host alights (thus ending the game) very soon after a game session is started.

4.4.6 Data Used For Matchmaking

To make these matchmaking decisions, the matchmaker can use data from three information sources as shown in Table 4.5.

4.4.6.1 Performance Data

The first is performance data such as RSSI values and ping times of various nodes (as observed by other nodes). The *GameOn* client periodically updates its discovery results to the matchmaker. With this data, the matchmaker can create a logical map of where each player is situated relative to other players. It can then use the heuristics shown earlier (peers more than 1 carriage apart can experience variable ping times etc.) to match clients together.

In addition to network measurements, we can also use historical predictions about how long a particular client will remain on the train/bus as a key input. These values can be computed using historical data (using techniques similar to Balan et al [100]). We show in Section 4.6.3.4 how using predicted trip times can improve the matchmaking performance.

4.4.6.2 Game-Specific Data

The next category of matchmaking data is game specific data. This is data that categorizes players into different buckets – based on their skill levels, probability of cheating, and other game-specific data. In addition, games can specify minimum and maximum game player numbers to ensure a high game experience. Grouping players according to the skill level is a well-known matchmaking metric. There are a few algorithms that have been employed by commercial video gaming platforms. For example, Xbox Live [101] uses the TrueSkill ranking system [102] that

Data	Reason To Use It
Performance	
Detection time	Hint for a robust connection
RSSI	Hint for a robust connection
Ping time	Hint for distance and crowdedness
Pred. Trip time	Games don't end abruptly
Game-specific	
Player Level	Ensure a fair/engaging game
Player Credibility	Ensure no cheating
Min. Player No.	Ensure game is interesting
User-specified	
Only with friends	Guarantee game experience
Nobody close by	Reduce real-world detection probability
Similar interests	Find future friends

Table 4.5: Data that can be used by the Matchmaker

computes the skills of gamers. Unfortunately, our current prototype does not use any of this data or these algorithms as we do not have the game-specific player information to generate this data historical data. However, adding this data into the matchmaking decision process, when the information does become available, is fairly straightforward.

4.4.6.3 User-Specific Data

The last category of data that can help the matchmaker is user-specific data. This is data that encapsulates a specific user's preferences and interests. For example, a player may not want to play games with nearby people as they are afraid it might lead to a confrontation. On the other hand, another player might want to meet nearby game players – but only if their interests match. Unfortunately, similar to game-specific data, our current prototype does not use this type of data as we have no historical or player records to generate the data from. However, once the data is available, integrating it into the matchmaker is easy.

4.4.7 Matchmaking Algorithm

In this work, we do not propose any new matchmaking algorithms. Instead, we leverage existing techniques to build a reasonable matchmaking solver. Our current prototype uses a weighted sum of components to determine the final match score of each player relative to every other player. The matchmaker then clusters these matched scores together to group players together who have similar scores. Currently, we use equally weighted normalized forms of collocation time, detection time, and ping time as the data sources for the match. In future work, we plan to investigate more sophisticated algorithms (including dynamic matchers that change their match goals (i.e., weights) based on the current situation) as well as add more data sources to the matching process.

4.5 Implementation

The *GameOn* Android client was implemented using Android 14 APIs (Android 4.0) as a user space application. It implements two background services that do the following; 1) *Cell*

connection manager (225 lines of code) that uses *WebSocket*s to communicate with the cloud-based matchmaker using JSON objects, and 2) *p2p connection manager* (1,027 lines of code) that implements the functions required to support multiple communication mediums (Wi-Fi Direct, Bluetooth, and etc.) as well as support multiple roles (client, server, relay node, and etc.). The *GameOn* client also provides a simple UI (201 lines of code) for the player to sign in, configure which games are available, configure their in-game names (handle), and to select games to play, and accept game requests. All the components are wrapped around a central control core (539 lines of code) that runs in separate threads.

We implemented the matchmaker in Java (188 lines of code) using the Play Framework [24] version 2.3.7. The matchmaker uses *WebSocket* and multiple threads to support multiple *GameOn* clients. All client generated data is stored in a MySQL database. The server also has a web interface for game developers to configure their game requirements and access game and credit records. The code size is small as the matchmaker currently uses "Performance" data only to make its decisions. However, as discussed earlier, the matchmaking logic can be easily modified to support use other data sources as and when they become available.

4.6 Evaluation

In this section, we present performance evaluation of *GameOn*. We first experimented its performance under various real-world use cases. In addition, we present detailed results from micro-benchmark experiments conducted under controlled settings, including overheads of matchmaking and hosting games as a server, performance impacts by various underlying network topologies (star topology vs. multi-hop topology), and impact of co-location time to game plays.

4.6.1 Experimental Setup

We performed all the experiments using Samsung Galaxy S3 (running Android 4.3) and S5 (running Android 4.4.2) phone. We used the three benchmark games described in Section 4.4.5 for all our real-world usage results as well as some of our micro-benchmarks. The matchmaker

was run on an Ubuntu server with a 3.4GHz 4-core CPU with 32 GB of memory. All power consumption values were measured using a Monsoon power monitor [51].

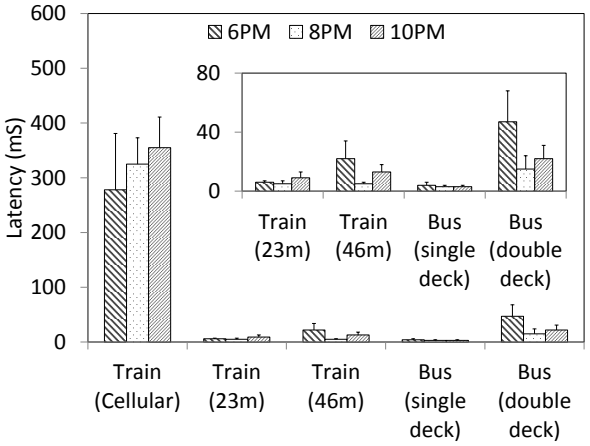
4.6.2 GameOn Working in Real Environments

We evaluated the end-to-end real-world performance of *GameOn* by playing three games on real public transports. The main goal was to compare *GameOn's* performance with that of the game played with *GameOn*. Each experiment was a 10 minute game session conducted by a four person group. After each gameplay session, all group members were asked to report their current phone battery level (which was compared to the reading just before the session started).

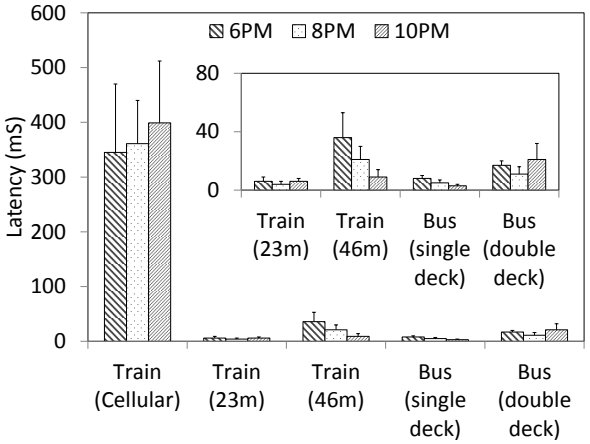
Figure 4.7 shows the latencies observed when playing the three games under five scenarios across three different times. In the first scenario, the games were hosted on an Internet server that was accessed using a cellular LTE connection. All four players in this scenario played solely as clients. The remaining four scenarios use *GameOn* where one peer device is selected to be the server with all the other peers connecting to it via Wi-Fi Direct. The four scenarios were "All players in the same train carriage, but spread throughout the carriage" (Train-23m), "All players spread across two train carriages (Train-46m), "All players spread across the same single deck bus" (Bus-Single-Deck), and "All players spread across the same double deck bus with the server on the lower deck" (Bus-Double-Deck). During game play, we periodically logged the ping latencies to the server on each client phone.

From the figure, we observe that LTE latencies are about 10 times longer than *GameOn* and that *GameOn* has very low latencies even across different types of transport and at different times (peak hours, normal etc.)

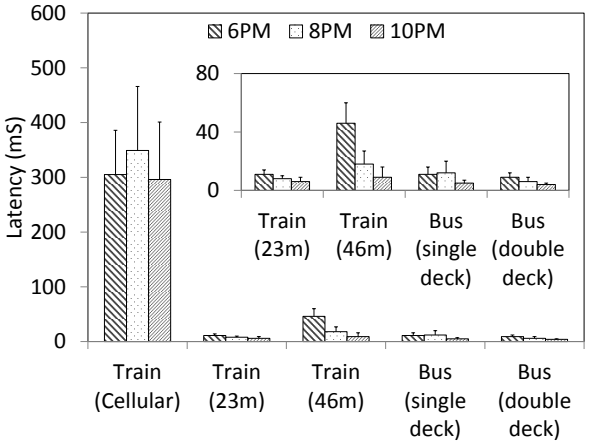
Figure 4.8 shows the battery usage of the phone when playing those three games. Since these experiments were done on buses and trains, we could not connect a hardware power monitor to the phones. Instead, we just used the Android battery levels as a gauge. The "Host" values is the power consumption of the phone that was chosen to host the server while "Client" values are the power consumption of the other client-only phones. Note: the "Host" phone serves as both a server *and* a client.



(a) Quake 3 (OpenArena)

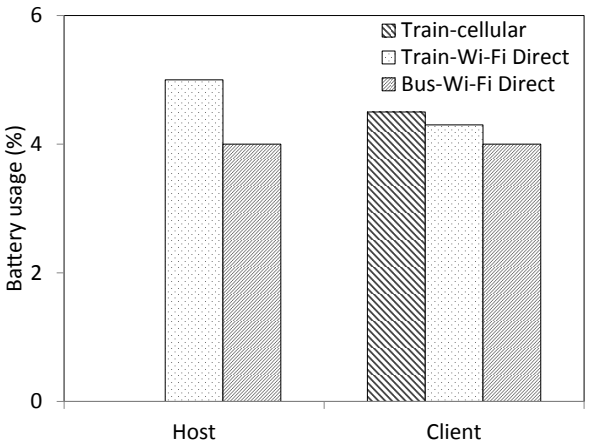


(b) Racer

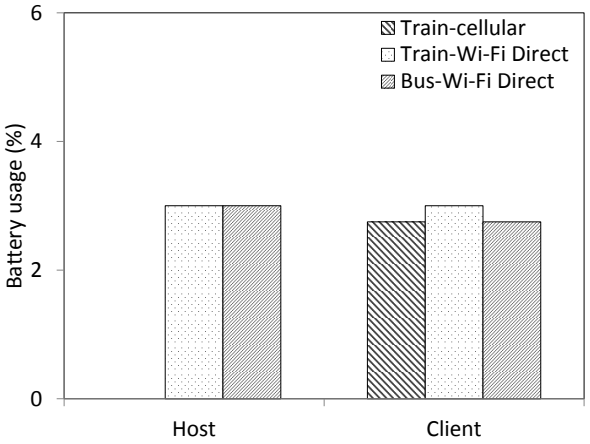


(c) 2048

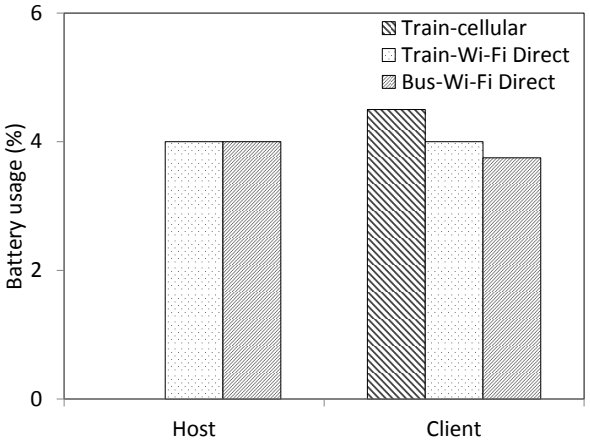
Figure 4.7: Game Latencies across 5 Scenarios and 3 Test Times



(a) Quake 3 (OpenArena)



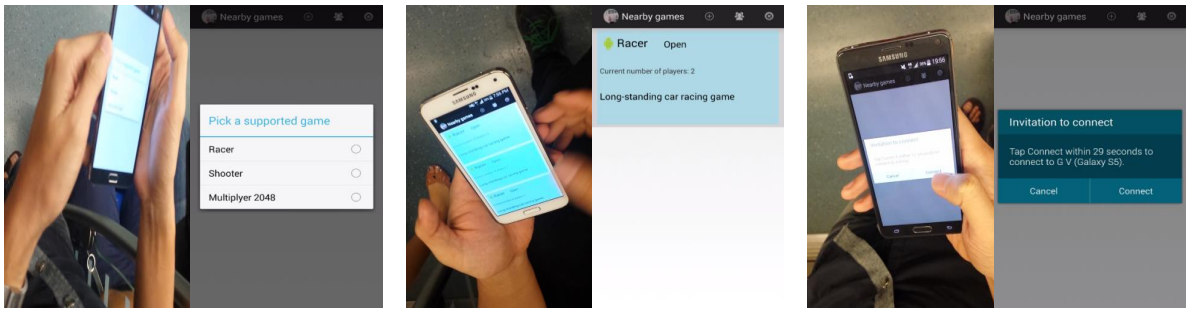
(b) Racer



(c) 2048

Figure 4.8: Battery Usage after 10 Minutes of Game Play

Figure 4.9: *GameOn* Being Used on a Real Public Train



(a) Player 1 Starts a New Game (b) Player 2 Searches for a Game to Play (c) Player 2 Joins Player 1's Game

Figure 4.10: The Demonstration of the Bootstrap of a *GameOn* Game Play

From the figure, we observe that hosting a server is not that expensive – power wise. Indeed, the power consumption for Hosts and Clients are quite similar and within the margin of error. Across the protocols, the power consumption is also somewhat similar.

The measured latency and energy values show that *GameOn* is capable of providing good local multiplayer game experience even in different types of train and bus environments. However, does it impact the user experience in some subtle way? To verify this, we asked each of the 4 game players to answer two self-reported questions on whether they felt the game was playable. To calibrate each member, they were asked, before doing the experiment, to play each game in a lab setting with no *GameOn* modifications to understand what the unmodified game felt like under perfect conditions. The two self reported questions were 1) "The game experience is the same as the one in the lab" and 2) "The phone feels hotter than it did in the lab". For both questions, the members had to answer using a 5-point Likert scale (1 – Strongly Agree to 5 – Strongly Disagree).

The final score was that all 4 game players strongly agreed that the modified game had the same experience as the in-lab unmodified version. In addition, all 4 players also strongly disagreed that the phone felt hotter than it did in the lab. However, they also mentioned that one of the games, *2048*, was not the easiest to play in a multiplayer fashion due to some UI limitations. However, this bug was not introduced by *GameOn* and was beyond our ability to fix.

Figure 4.9 demonstrates two players playing a game on the same train using *GameOn*. In this use case, the two players are a half-carriage away from each other – one sitting and one standing. Figure 4.10 shows the matchmaking process to start the game session. At step (a), the player 1 starts a new game session using the *GameOn* UI, by selecting a game to host. At step (b), player 2 starts *GameOn* and queries the *GameOn* matchmaker to find the available games in their vicinity. Player 2 then picks one of the available games through the *GameOn* UI – he can only host a game if there are no suitable games available. Finally, at step (c), player 1 accepts the join request from player 2, and the game starts.

4.6.3 Micro-benchmarks

We now present micro-benchmarks results:

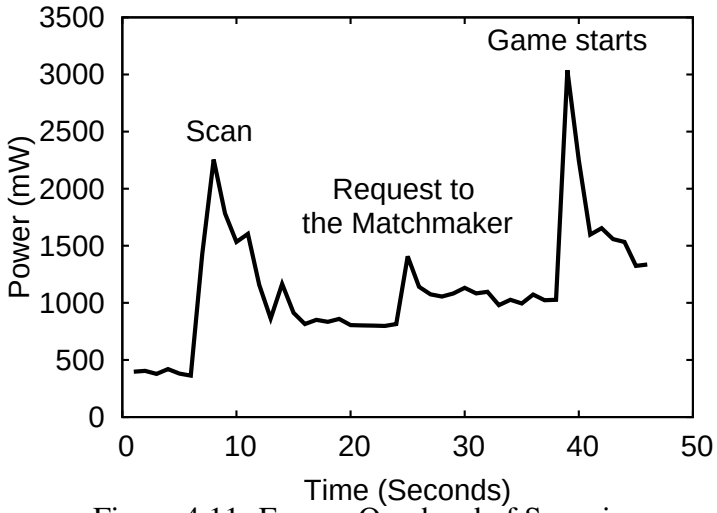


Figure 4.11: Energy Overhead of Scanning

4.6.3.1 GameOn Overheads

We evaluated the overheads of two key operations: peer discovery and requesting a suitable game group from the matchmaker. Figure 4.11 shows the energy cost of scanning for nearby players via Wi-Fi Direct and that of sending a request for a game group. By themselves, both actions cause reasonably high spikes in the power consumption. However, compared with the power spike when the game itself starts, the scanning and requesting costs are acceptable.

4.6.3.2 Resource Usage and Group Scalability

GameOn selects a player to host the game server and all the other players will connect to this server. As shown earlier, this does not increase the energy cost of the server device. However, what about the scalability of the device? Can it support multiple game clients without any performance degradation?

To understand this, we scheduled 8 clients to join a particular server one after the other at fixed intervals over a 10-minute period. On the server device, we logged its resource usage, including CPU utilization, heap usage, and network traffic using Wi-Fi Direct, every 10 seconds. Figure 4.12 shows the CPU and heap usage plots. Each plot starts from a single client case where the server phone is connected to itself with new clients (up to a max of 8) periodically connecting. The heap

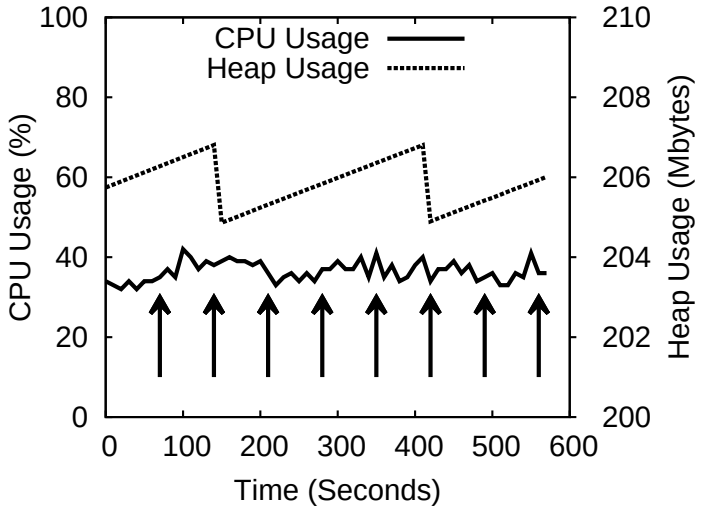


Figure 4.12: CPU & Memory Overhead. Up to 8 clients join the group gradually. The arrows represent the time at which one more client joins. The saw tooth decrease in heap size is when the Java garbage collector activates.

usage shows a zigzag curve due to memory being reclaimed by the Android garbage collector at regular intervals. From the figure, we observe that the CPU and heap usage do not substantially increase even when the server device is hosting all 8 client players.

We next investigate how large p2p groups can become before performance (in terms of server ping times) and server energy consumption become factors. Figure 4.13 shows the ping times and power consumption when the group size scales up. When connected to 8 players, the power consumption of the server increases 23.7% compared to hosting just 1 player. Thus, hosting a game does not add a very large overhead to the phone's energy usage. However, we found that the ping latencies increase quite fast as more and more clients are added. In particular, we observed a large latency rise when the 7th client was connected. Thus, we find that a current modern smartphone can comfortably serve as the server for up to 6 clients. After this point, the ping latencies start to increase significantly which could result in gameplay issues.

Figure 4.14 shows the network usage of the server in terms of the number of bytes exchanged over the Wi-Fi Direct link. We observe that both the received and transmitted traffic grows quadratically as the group size increases. A transmitted packet from the server usually includes a snapshot of the whole group state, while a received packet usually includes only a single client command

or update. Thus, on the server, the amount of data received is usually much lower than the amount sent.

4.6.3.3 Support for Other Topologies

In all previous experiments, we have used a star topology where every client is connected directly to the server. However, in some cases, a client may not be able to connect directly to the server (when the client is at the other end of a crowded train for example). For example, Figure 4.15 shows a scenario where the four players are spread out linearly so that the rightmost player does not have a reliable connection with the server client. In these cases, is it possible to leverage intermediate clients as relay nodes to form a multi-hop *linked-list* topology where a node is connected to an intermediate node that connects it to the server? *GameOn* supports multi-hop networks but with some limitations. In particular, joining two Wi-Fi Direct groups (to create a multi-hop network) at the same time is not allowed, even in the latest version of Android. This joining of groups feature is an optional feature in the Wi-Fi Direct standard that has not been implemented in Android. Thus, to create a relay node for a multi-hop environment, we have to use two different networking technologies / radios. In this case, we will have to use Bluetooth together with Wi-Fi Direct with one side of the linked-list using Bluetooth and the other side using Wi-Fi Direct. However, as stated earlier, Bluetooth is not the best protocol for the scenarios *GameOn* is tackling. We re-visit these claims using a three-node scenario as shown in Figure 4.15.

Figure 4.16 shows the energy consumption when using three nodes with a star and a linked-list topology. We instrumented the *Racer* game so that it automatically looped the same track to create a repeatable trace. For each experiment, we turned off all background processes and measured the power consumption using the Monsoon power monitor. It should be noted that the absolute numbers are not that interesting as they are device-specific. Instead we focus on the difference between the two topologies.

We observed that in the star topology, the server node (the one sending most of the data) consumes the most power followed by the other two nodes (client 2 and client 1). When using the linked-list topology, the host (server) and sink (client 1) nodes consume similar power to the star

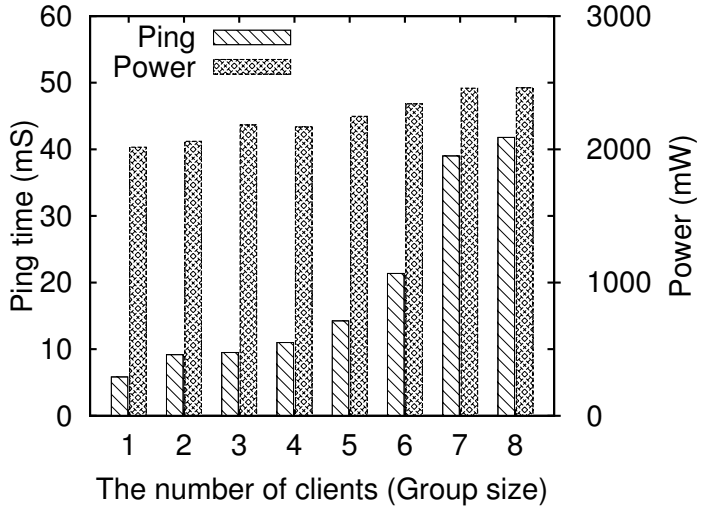


Figure 4.13: Power & Latency Overhead. The ping latencies and power consumption of the server device when hosting a game. The ping time at the client side increases quickly while the power consumption at the host side rises linearly. The result proves a modern smartphone can modestly support a group of 1-8 players.

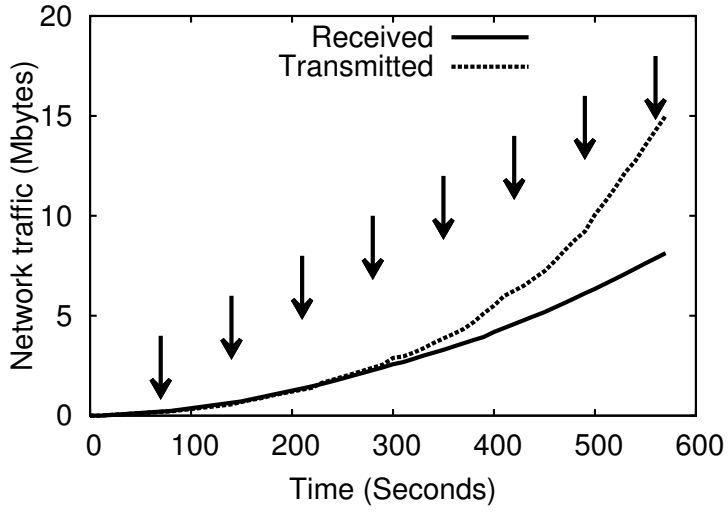


Figure 4.14: Network Overhead. Up to 8 clients join the group gradually. The arrows represent the time at which one more client joins. As the group size becomes larger, both the transmitted and received traffic increases quadratically.

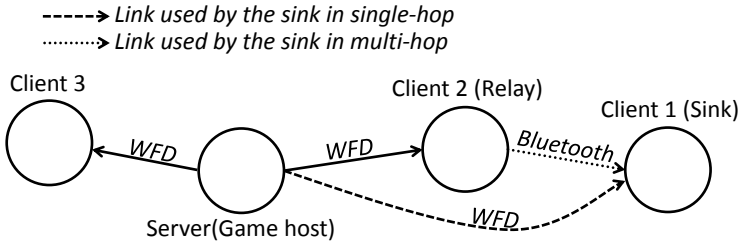


Figure 4.15: Single-hop vs. Multi-hop Topologies. The arrows indicate the server information flow direction. Dotted Wi-Fi Direct link indicates poor connectivity.

topology. However, the relay node (client 2) uses 13% more power in the linked-list case. This is because it has to use two radios (Wi-Fi Direct and Bluetooth) simultaneously to bridge the two sides of the relay.

We now evaluate the effectiveness of the linked-list topology at reducing latency spikes caused by nodes being too far away from each other. To do this, we placed two node (a source and a sink) two train carriages apart from each other on a public train (that was moving and picking up passengers etc.). We then placed a relay node in between the two nodes (i.e., the relay node was 1 carriage away from both the source and the sink). The source and the sink were then connected to each other using Wi-Fi Direct. The source was also connected to the relay node via Wi-Fi Direct while the sink connected to the relay node via Bluetooth. The sink then started pinging the source across both the direct Wi-Fi Direct connection as well as the multi-hop (via the relay) Bluetooth connection.

Figure 4.17 shows the latency results. We observe that the direct link between the source and the sink (i.e., the star topology) showed variable ping times as the distance was far and the link quality was thus affected by passenger movements etc. However, the link via the relay node showed much more predictable and stable performance. However, the ping times for the Bluetooth link are still high (yet stable) as Bluetooth is not the best protocol (as shown earlier) for this type of environment.

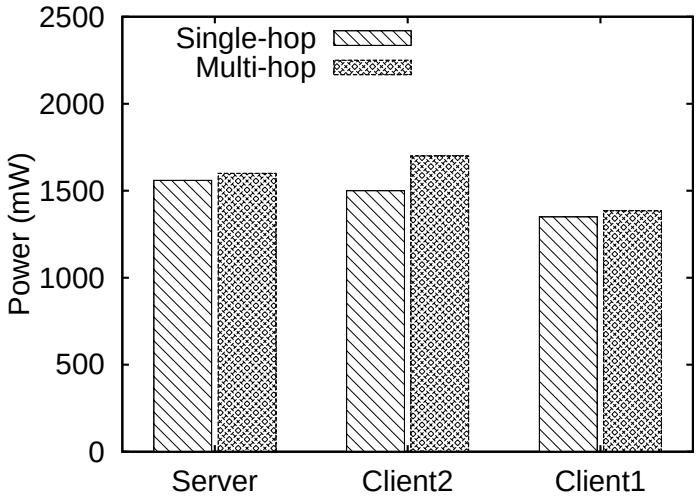


Figure 4.16: Power Consumption under Various Topologies. Power consumption of three nodes using different topologies. In the linked-list topology, the relay node (client 2) connects to the source node via a Wi-Fi Direct link, and is connected by the sink node via a Bluetooth link.

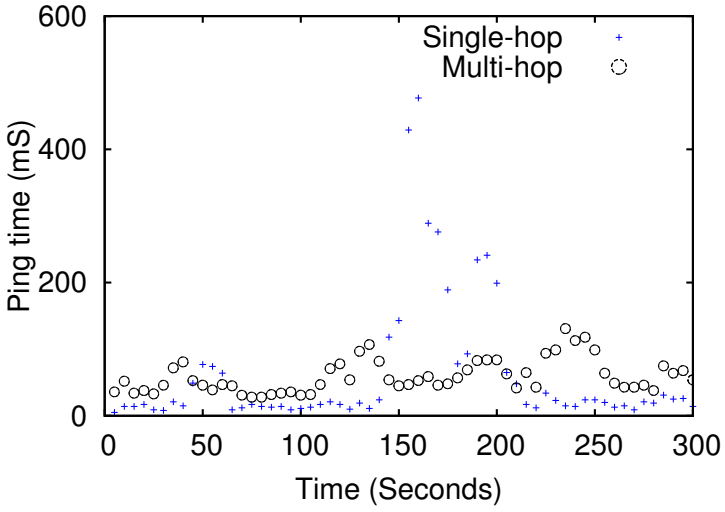


Figure 4.17: Latency of Various Topologies. Client-side latencies under two topologies. A client (sink) is connected to the group owner via two different routes at the same time. One is a direct connection via a Wi-Fi Direct link. And the other one is a connection to the relay node via a Bluetooth link.

4.6.3.4 Improving Matchmaking With Time Predictions

In this final test, we show that using predictions of how long a particular trip will last for a given individual can have big positive effects on the matchmaking performance. To perform this test, we selected only the trips that started from one of the starting train stations (called k) to a specific train station at our university. The starting train station was chosen because the trains start off empty there (so everyone on that train when it leaves got on at station k) and it was not an interchange station. I.e., everyone had to swipe their NFC cards at that station itself to get into it. It was not possible for them to enter at some other station and take another train to this station. 2) We picked a specific time (8 a.m.) and day (1st Monday of Nov.) and extracted all the commuters who entered station k at that time and day. 3) We then exhaustively created all possible 2, 3, 4, 5, 6 person groups that could be created from the set of people who entered that station. 4) We then computed how long each of these groups actually stayed together (i.e., the minimum co-location time until someone in that group left the train). This result represents a naive matchmaker that just selects people randomly and hopes that they will be together long enough.

Figure 4.18 shows the results of this test. We observe that the time the entire group was together is quite low and with a very high standard deviation (indicating that some groups were together for much less time). In addition, as the group size increased, the time spent together decreased significantly as the probability of any one person in the group leaving increased. This result shows that just randomly grouping people together can lead to bad outcomes.

However, we found that using predicted individual trip times can result in better estimates. First, we created historical buckets for each user that was station, day, and time specific. We then used this history to calculate, for each user, a predicted trip start time (with stdev.) for any station at any time and day. This prediction lets us increase the minimum co-location time for all group sizes as we can cluster passengers by their predicted trip times. For example, our standard deviation for any given trip time and group size dropped to a few % compared to 60-70% with the naive approach. However, this approach can lead to data sparsity issues. For example, only 3,500 of the 15,948 passengers (22%) used to generate Figure 4.18 had multiple trips from that station from

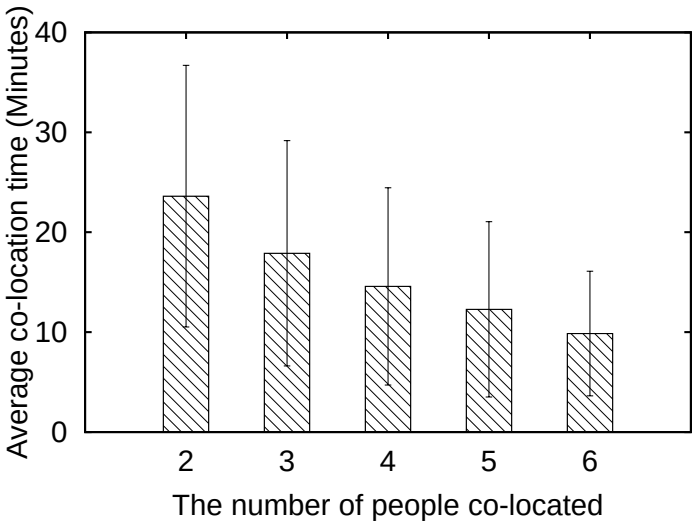


Figure 4.18: Group Size versus Collocation Time

which we could calculate a history. We plan to look at techniques to improve this yield in future work.

4.6.4 Summary

In this section, we showed that *GameOn* works well in the real world with good latencies and end-user experiences with up to 4 players in a variety of bus and train environments. We then showed that the energy, CPU, ping latencies, and memory overheads of hosting a game server are minimal if the number of game clients is kept low (under 8). Next, we should that we could support multi-hop p2p methodologies in addition to he base star topology. Finally, we showed that naively predicting the co-location time can result in very sub-optimal matches.

4.7 Discussion

The main limitations of the current *GameOn* prototype are: 1) The matchmaker, while supporting many attributes well (as far as we can tell), cannot be completely validated as we do not have data for many of the skill and player-centric attributes. 2) We have built *GameOn* to handle only the system aspects of multiplayer games. Unfortunately, we have no control over the game itself which has a larger say on user satisfaction. For example, *GameOn* can handle cases where users

join and leave a game in the middle (as long as the player hosting the server does not leave) and *GameOn* can handle alternate game modes such as "spectator mode" if those modes use standard networking APIs. However, *GameOn*, by itself, can do very little to make a game "fun" which is ultimately the most important criteria. And 3), the multi-hop support needs to use Bluetooth and Wi-Fi Direct for multi-hop settings (thus lowering its range to what Bluetooth supports) as Android does not currently allow multiple Wi-Fi Direct connections. Finally, the survey presented in Section 4.3.2 was conducted mostly with undergraduate students and thus may not be generalisable. In addition, our performance experiments were conducted using only two different models of phones. Thus results may vary with other phone types.

4.8 Related Work

p2p game matchmaking: Switchboard [103] proposed techniques to predict latencies during a game play using quick pre-game measurements. Htrae [104] predicts inter-player latencies using geo-location data. Ly et al. [105] developed an approach to select the best detour route for game packets. However, these systems and techniques were targeting game consoles or devices connected to the Internet. Our goal is to use p2p networking to connect players on public transport.

Mobile p2p applications: Collaborative smartphone applications have emerged in diverse application domains such as media sharing [106] and context sensing [107][108]. Like *GameOn*, they propose several core techniques to enable in-situ collaboration among co-located smartphones. McNamara et al. [106] devised a scheme to predict remaining co-location duration for stable exchange of multimedia files. CoMon [107] proposed a resource planning mechanism to maximize benefit while achieving fairness. However, building a system for collaborative mobile gaming imposes a set of unique challenges due to the strict gaming latency and power requirements. To address these, we developed a new end-to-end system, *GameOn*, with careful attention to various system components such as network protocols, peer discovery, matchmaking, and low latency game play. Some airlines offer multi-player games among passengers during long-haul flights [109].

However, only a few limited games can be supported through wired entertainment systems embedded in passenger seats whereas *GameOn* can support commodity mobile games on smartphones without any infrastructure support in buses or trains.

Mobile p2p framework: There have been efforts to develop generic platforms to facilitate development of various mobile p2p applications [110]. For example, the well-known open source project Alljoyn [111], aims to provide a set of APIs and runtime to easily build network connections among multiple mobile devices. *GameOn* opens a new application domain of multiplayer gaming by supporting game-specific requirements that Alljoyn does not support. It will be an interesting to test if Alljoyn can work with *GameOn*. There have also been prior work to re-write binaries without source code access. RetroSkeleton [112] presents an app rewriting framework that allows developers to integrate new features into existing apps. *GameOn* did not use re-writing methods initially as we wanted to understand the challenges required to port existing games to use *GameOn*.

4.9 Conclusion

In this paper, we presented *GameOn*, a system for allowing commuters on public transportation to play multiplayer games with each other using Wi-Fi Direct as a p2p communication medium. We motivated the reasons why *GameOn* is useful (long commute times) and then described the various components of *GameOn*. We plan to extend *GameOn* to allow users who share similar interests (that are discovered through specific types of games) to meet up with each other in the physical world. We are also considering system-level support for spectator-mode; where commuters can join existing games as passive observers instead of active players. Both of these planned extensions should increase the adoption rate of *GameOn* .

Chapter 5

Conclusion

5.1 Lessons Learned

- The gap between *a well-established technique* and *its wholesale adoption*.
  This dissertation draws inspiration from many existing systems. Such systems usually have been published in good academia venues, and some does have their mobile users. Can such systems be directly plugged into our design? If not, what are they missing? In fact, there have been some research papers that talk about selecting proper participants for public sourcing, but they do not consider whether the selected ones can afford; there have been many energy accounting systems available in literature, but they are not able to be immediately applied to most of today's mobile devices; there have been industrial tutorials and platforms that assist discovering mobile devices and building mobile p2p networks, but they do not enable one to find the "wanted" devices. One may argue that it is technology creators that should fill such gaps; while my argument is it is ordinary people's true needs that drive us to find out such gaps. The three components of this dissertation is under the guidance of such philosophy.

- The scope in which we discuss the accuracy (or performance) of a proposed technique.
  Given a target goal and a target group of users, a designer can quickly sketch a rough design space including core goals and a set of design choices. When we include ordinary people in the loop, the design space might be enlarged. For example, an energy accounting technique used by computer system designers basically would target on improving accuracy and reducing energy overhead. However, when we design an energy accounting technique for ordinary people, our

goal now include good user experience and high portability. Sufficient tradeoff analysis within the design space is definitely needed.

- The value of performing empirical research in the field of mobile computing.
  While my dissertation attempts to technically eliminating people' concerns to participating in mobile device-assisted collaboration, it is just the initial step to finally achieving large participation. We performed preliminary user study and qualitative research, but it was hard to field test it in the real world. However, by comparing different design choices and results, we clarify the desired properties in a concrete system. A networked computer system can be as complicated as a biological and physical system. Without prototyping and testing end-to-end systems, it is not easy to get insights out of it and then make a system effective and efficient.

5.2 Open Questions

- In the first work, we observed while the desired contexts by applications can be various, the underlying sensors (GPS, microphone, accelerometer, camera etc.) and general contexts (location, activity, transportation mode etc.) are not unlimited. This offers an opportunity of combining multiple data collection instances of different applications so as to greatly reduce overall energy consumption. In fact, when the resource management as an separate layer emerges, other management facilities, such as data management and privacy management, can be added into this layer, and all together can make up an "Operating System" of "Smart cities".

- In the second work, we opened the possibilities of turning personal device into a multi-user usage model. However, the idea also raises interesting usability issues. For example, *how much energy a user may want to allocate into a class?* Relevantly, *should the system implement a borrow-return mechanism across classes, as our users suggest, or encourage users to proactively adapt to new usage model and battery interface?* Our user also suggests adding access control scheme based on time. All these problems surely need to be investigated in more detail.

- In the third work, we have not investigated the downsides of enabling familiar stranger social network. *What happens if that person gets angry and starts looking for the person(s) they lost*

to? and *Would it be a form of "game-rage" (similar to road rage)?* Identifying the people you are playing with will be hard in crowded trains where everyone is awake and using their phone. But what about on longer train journeys where a majority of commuters are sleeping? These types of social phenomena and implications need to be investigated in more details. In addition, *whether can the same setup be applied to other venues such as schools and conferences?*

LIST OF REFERENCES

[1] The White House, Office of the Press Secretary. Accessed on 13th August 2016.

[2] National Science Foundation, the Chicago Innovation Fund, and Argonne National Laboratory. Accessed on 13th August 2016.

[3] Envision America. Accessed on 13th August 2016.

[4] Multi-City Innovation Campaign. Accessed on 13th August 2016.

[5] MetroLab Network: A CITY-UNIVERSITY COLLABORATIVE FOR URBAN INNOVATION. Accessed on 13th August 2016.

[6] L. Wirth, "Urbanism as a way of life," *The American Journal of Sociology*, vol. XLIV, no. 1, pp. 1–24, July 1938.

[7] C. S. Fischer, *To Dwell among Friends: Personal Networks in Town and City*. University Of Chicago Press, 1982.

[8] S. Milgram, *The individual in a social world*. Addison-Wesley, 1977.

[9] E. Paulos and E. Goodman, "The familiar stranger: Anxiety, comfort, and play in public places," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, (New York, NY, USA), pp. 223–230, ACM, 2004.

[10] R. K. Balan, A. Misra, and Y. Lee, "Livelabs: Building an in-situ real-time mobile experimentation testbed," in *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*, HotMobile '14, (New York, NY, USA), pp. 14:1–14:6, ACM, 2014.

[11] Computation Institute, University of Chicago. Accessed on 13th August 2016.

[12] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: Rich monitoring of road and traffic conditions using mobile smartphones," in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, (New York, NY, USA), pp. 323–336, ACM, 2008.

[13] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson, "Cooperative transit tracking using smart-phones," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, (New York, NY, USA), pp. 85–98, ACM, 2010.

[14] Z. Yang, C. Wu, and Y. Liu, "Locating in fingerprint space: Wireless indoor localization with little human intervention," in *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, (New York, NY, USA), pp. 269–280, ACM, 2012.

[15] D. Hasenfratz, O. Saukh, S. Sturzenegger, and L. Thiele, "Participatory air pollution monitoring using smartphones," *Mobile Sensing*, pp. 1–5, 2012.

[16] P. Jain, J. Manweiler, A. Acharya, and K. Beaty, "Focus: Clustering crowdsourced videos by line-of-sight," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, SenSys '13, (New York, NY, USA), pp. 8:1–8:14, ACM, 2013.

[17] K. K. Rachuri, M. Musolesi, C. Mascolo, P. J. Rentfrow, C. Longworth, and A. Aucinas, "Emotionsense: A mobile phones based adaptive platform for experimental social psychology research," in *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, UbiComp '10, (New York, NY, USA), pp. 281–290, ACM, 2010.

[18] S. Sen, D. Chakraborty, V. Subbaraju, D. Banerjee, A. Misra, N. Banerjee, and S. Mittal, "Accommodating user diversity for in-store shopping behavior recognition," in *Proceedings of the 2014 ACM International Symposium on Wearable Computers*, ISWC '14, (New York, NY, USA), pp. 11–14, ACM, 2014.

[19] R. Wang, F. Chen, Z. Chen, T. Li, G. Harari, S. Tignor, X. Zhou, D. Ben-Zeev, and A. T. Campbell, "Studentlife: Assessing mental health, academic performance and behavioral trends of college students using smartphones," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14, (New York, NY, USA), pp. 3–14, ACM, 2014.

[20] S. Reddy, D. Estrin, and M. Srivastava, "Recruitment framework for participatory sensing data collections," in *Proceedings of the 8th International Conference on Pervasive Computing*, Pervasive'10, (Berlin, Heidelberg), Springer-Verlag, 2010.

[21] A. J. Oliner, A. P. Iyer, I. Stoica, E. Lagerspetz, and S. Tarkoma, "Carat: Collaborative energy diagnosis for mobile devices," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, SenSys '13, (New York, NY, USA), pp. 10:1–10:14, ACM, 2013.

[22] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Transactions on Networking (ToN)*, vol. 1, no. 3, pp. 344–357, 1993.

[23] J. C. R. Bennett and H. Zhang, "Wf2q: Worst-case fair weighted fair queueing," in *Proceedings of the Fifteenth Annual Joint Conference of the IEEE Computer and Communications Societies Conference on The Conference on Computer Communications - Volume 1*, INFOCOM'96, (Washington, DC, USA), pp. 120–128, IEEE Computer Society, 1996.

[24] Play Framework https://www.playframework.com/. Accessed on 23th July 2016.

[25] Android Open Source Project https://source.android.com/. Accessed on 23th July 2016.

[26] B. Awerbuch, R. Gawlick, T. Leighton, and Y. Rabani, "On-line admission control and circuit routing for high performance computing and communication," in *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pp. 412–423, IEEE, 1994.

[27] A. I. Elwalid and D. Mitra, "Effective bandwidth of general markovian traffic sources and admission control of high speed networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 1, no. 3, pp. 329–343, 1993.

[28] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, (New York, NY, USA), pp. 18:1–18:17, ACM, 2015.

[29] M.-R. Ra, B. Liu, T. F. La Porta, and R. Govindan, "Medusa: A programming framework for crowd-sensing applications," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, (New York, NY, USA), pp. 337–350, ACM, 2012.

[30] T. Das, P. Mohan, V. N. Padmanabhan, R. Ramjee, and A. Sharma, "Prism: Platform for remote sensing using smartphones," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, (New York, NY, USA), pp. 63–76, ACM, 2010.

[31] N. D. Lane, Y. Chon, L. Zhou, Y. Zhang, F. Li, D. Kim, G. Ding, F. Zhao, and H. Cha, "Piggyback crowdsensing (pcs): Energy efficient crowdsourcing of mobile sensor data by exploiting smartphone app opportunities," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, SenSys '13, (New York, NY, USA), pp. 7:1–7:14, ACM, 2013.

[32] S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt, "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*, MobiSys '08, (New York, NY, USA), pp. 174–186, ACM, 2008.

[33] Q. Zhao, Y. Zhu, H. Zhu, J. Cao, G. Xue, and B. Li, "Fair energy-efficient sensing task allocation in participatory sensing with smartphones," in *2014 IEEE Conference on Computer Communications, INFOCOM 2014, Toronto, Canada, April 27 - May 2, 2014*, pp. 1366–1374, 2014.

[34] D. Yang, G. Xue, X. Fang, and J. Tang, "Crowdsourcing to smartphones: Incentive mechanism design for mobile phone sensing," in *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, (New York, NY, USA), pp. 173–184, ACM, 2012.

[35] L. Gao, F. Hou, and J. Huang, "Providing long-term participation incentive in participatory sensing," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 2803–2811, IEEE, 2015.

[36] IBM, *BYOD: Bring your own device, Why and how you should adopt BYOD*. `http://www.ibm.com/mobilefirst/us/en/bring-your-own-device/byod.html`.

[37] J. Andrus, C. Dall, A. V. andO. Laadan, and J. Nieh, "Cells: A virtual mobile smartphone architecture," in *Proceedings of Symposium on Operating Systems Principles (SOSP)*, 2011.

[38] T. Oluwafemi, E. Fernandes, O. Riva, F. Roesner, S. Nath, and T. Kohno, "Per-app profiles with appfork: The security of two phones with the convenience of one," Tech. Rep. MSR-TR-2014-153, December 2014.

[39] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof," in *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, (New York, NY, USA), pp. 29–42, ACM, 2012.

[40] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazieres, and N. Zeldovich, "Energy management in mobile devices with Cinder operating systems," in *Proceedings of European Conference on Computer Systems (EuroSys 2011)*, 2011.

[41] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES/ISSS)*, 2010.

[42] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, (New York, NY, USA), pp. 153–168, ACM, 2011.

[43] F. Xu, Y. Liu, Q. Li, and Y. Zhang, "V-edge: Fast self-constructive power modeling of smartphones based on battery voltage dynamics," in *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.

[44] X. Chen, N. Ding, A. Jindal, Y. C. Hu, M. Gupta, and R. Vannithamby, "Smartphone energy drain in the wild: Analysis and implications," in *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '15, (New York, NY, USA), pp. 151–164, ACM, 2015.

[45] M. Dong and L. Zhong, "Self-Constructive High-Rate System Energy Modeling for Battery-Powered Mobile Systems," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2011.

[46] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "Ecosystem: Managing energy as a first class operating system resource," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS X, (New York, NY, USA), pp. 123–132, ACM, 2002.

[47] A. Rahmati, A. Qian, and L. Zhong, "Understanding human-battery interaction on mobile phones," in *Proceedings of the 9th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '07, (New York, NY, USA), pp. 265–272, ACM, 2007.

[48] W. Jung, Y. Chon, D. Kim, and H. Cha, "Powerlet: An active battery interface for smartphones," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14, (New York, NY, USA), pp. 45–56, ACM, 2014.

[49] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "Appscope: Application energy metering framework for android smartphone using kernel activity monitoring," in *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, (Boston, MA), pp. 387–400, USENIX, 2012.

[50] M. Dong and L. Zhong, "Rethink Energy Accounting with Cooperative Game Theory," in *Proceedings of ACM 20th International Conference on Mobile Computing and Networking (MobiCom)*, 2014.

[51] Monsoon Solutions Inc., *Monsoon Power Moniter*. `http://www.msoon.com/LabEquipment/PowerMonitor/`.

[52] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using magpie for request extraction and workload modelling," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, (Berkeley, CA, USA), pp. 18–18, USENIX Association, 2004.

[53] Agilent Technologies, *Agilent 34411A digital multimeter*. `http://cp.literature.agilent.com/litweb/pdf/5989-3738EN.pdf`.

[54] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, "Diversity in smartphone usage," in *Proceedings of International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2010.

[55] L. Zhong and N. K. Jha, "Graphical user interface energy characterization for handheld computers," in *Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '03, (New York, NY, USA), pp. 232–242, ACM, 2003.

[56] J. F. Nash Jr, "The bargaining problem," *Econometrica: Journal of the Econometric Society*, pp. 155–162, 1950.

[57] R. Nathuji and K. Schwan, "Virtualpower: Coordinated power management in virtualized enterprise systems," in *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, (New York, NY, USA), pp. 265–278, ACM, 2007.

[58] Q. Cao, D. Fesehaye, N. Pham, Y. Sarwar, and T. Abdelzaher, "Virtual battery: An energy reserve abstraction for embedded sensor networks," in *Proceedings of the 2008 Real-Time Systems Symposium*, RTSS '08, (Washington, DC, USA), pp. 123–133, IEEE Computer Society, 2008.

[59] J. Flinn and M. Satyanarayanan, "Energy-aware adaptation for mobile applications," in *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, SOSP '99, (New York, NY, USA), pp. 48–63, ACM, 1999.

[60] R. LiKamWa, B. Priyantha, M. Philipose, L. Zhong, and P. Bahl, "Energy characterization and optimization of image sensing toward continuous mobile vision," in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, (New York, NY, USA), pp. 69–82, ACM, 2013.

[61] H. Hoffmann, "Jouleguard: Energy guarantees for approximate applications," in *Proceedings of The 25th ACM Symposium on Operating Systems Principles*, SOSP '15, 2015.

[62] X. Ma, P. Huang, X. Jin, P. Wang, S. Park, D. Shen, Y. Zhou, L. K. Saul, and G. M. Voelker, "edoctor: Automatically diagnosing abnormal battery drain issues on smartphones.," in *NSDI*, vol. 13, pp. 57–70, 2013.

[63] A. Badam, R. Chandra, J. Dutra, A. Ferrese, S. Hodges, P. Hu, J. Meinershagen, T. Moscibroda, B. Priyantha, and E. Skiani, "Software defined batteries," in *Symposium on Operating Systems Principles (SOSP'15)*, ACM Association for Computing Machinery, October 2015.

[64] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson, "Vtrack: Accurate, energy-aware road traffic delay estimation using mobile phones," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, (New York, NY, USA), pp. 85–98, ACM, 2009.

[65] D. H. Kim, Y. Kim, D. Estrin, and M. B. Srivastava, "Sensloc: Sensing everyday places and paths using less energy," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, (New York, NY, USA), pp. 43–56, ACM, 2010.

[66] R. Jurdak, P. Corke, D. Dharman, and G. Salagnac, "Adaptive gps duty cycling and radio ranging for energy-efficient localization," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, (New York, NY, USA), pp. 57–70, ACM, 2010.

[67] D. Chu, N. D. Lane, T. T.-T. Lai, C. Pang, X. Meng, Q. Guo, F. Li, and F. Zhao, "Balancing energy, latency and accuracy for mobile sensor data classification," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, (New York, NY, USA), pp. 54–67, ACM, 2011.

[68] Y. Chon, E. Talipov, H. Shin, and H. Cha, "Mobility prediction-based smartphone energy optimization for everyday location monitoring," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, (New York, NY, USA), pp. 82–95, ACM, 2011.

[69] T. Park, J. Lee, I. Hwang, C. Yoo, L. Nachman, and J. Song, "E-gesture: A collaborative architecture for energy-efficient gesture recognition with hand-worn sensor and mobile devices," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, (New York, NY, USA), pp. 260–273, ACM, 2011.

[70] J. Liu, B. Priyantha, T. Hart, H. S. Ramos, A. A. F. Loureiro, and Q. Wang, "Energy efficient gps sensing with cloud offloading," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, (New York, NY, USA), pp. 85–98, ACM, 2012.

[71] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell, "The jigsaw continuous sensing engine for mobile phone applications," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, (New York, NY, USA), pp. 71–84, ACM, 2010.

[72] Y. Ju, Y. Lee, J. Yu, C. Min, I. Shin, and J. Song, "Symphoney: A coordinated sensing flow execution engine for concurrent mobile sensing applications," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, (New York, NY, USA), pp. 211–224, ACM, 2012.

[73] H. Han, J. Yu, H. Zhu, Y. Chen, J. Yang, G. Xue, Y. Zhu, and M. Li, "E3: Energy-efficient engine for frame rate adaptation on smartphones," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, SenSys '13, (New York, NY, USA), pp. 15:1–15:14, ACM, 2013.

[74] M.-R. Ra, B. Priyantha, A. Kansal, and J. Liu, "Improving energy efficiency of personal sensing applications with heterogeneous multi-processors," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, (New York, NY, USA), pp. 1–10, ACM, 2012.

[75] P. Georgiev, N. D. Lane, K. K. Rachuri, and C. Mascolo, "Dsp.ear: Leveraging co-processor support for continuous audio sensing on smartphones," in *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, SenSys '14, (New York, NY, USA), pp. 295–309, ACM, 2014.

[76] M. Lantz, J. Litton, and B. Bhattacharjee, "Drowsy power management," in *Proceedings of The 25th ACM Symposium on Operating Systems Principles*, SOSP '15, 2015.

[77] N. Brouwers, M. Zuniga, and K. Langendoen, "Neat: A novel energy analysis toolkit for free-roaming smartphones," in *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, SenSys '14, (New York, NY, USA), pp. 16–30, ACM, 2014.

[78] S. DeBruin, B. Ghena, Y.-S. Kuo, and P. Dutta, "Powerblade: A low-profile, true-power, plug-through energy meter," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, SenSys '15, (New York, NY, USA), pp. 17–29, ACM, 2015.

[79] H. Lim, A. Kansal, and J. Liu, "Power budgeting for virtualized data centers," in *2011 USENIX Annual Technical Conference (USENIX ATC '11)*, USENIX, June 2011.

[80] Bluetooth SIG, *Learn the Bluetooth(R) technology basics*, 2014. Accessed on 25th March 2015.

[81] Wi-Fi Alliance, *Portable Wi-Fi(R) that goes with you anywhere*, 2014. Accessed on 25th March 2015.

[82] Sergii Pylypenko, *Openarena Android port site*. Accessed on 3rd December 2014.

[83] Nicolas Gramlich, *AndEngine game engine*. Accessed on 3rd December 2014.

[84] Paul Sarbinowski, *2048 Android port*. Accessed on 3rd December 2014.

[85] A. Patro, S. Rayanchu, M. Griepentrog, Y. Ma, and S. Banerjee, "The anatomy of a large mobile massively multiplayer online game," in *Proceedings of the First ACM International Workshop on Mobile Gaming (MobiGames)*, 2012.

[86] S. Khalaf, *Flurry Five-Year Report: Its an App World. The Web Just Lives in It*, 2013. Accessed on 6th December 2014.

[87] Land Transport Authority of Singapore, *Singapore Land Transport: Statistics In Brief*. Accessed on 25th March 2015.

[88] Land Transport Authority of Singapore. Accessed on 25th March 2015.

[89] Trade Union Congress (TUC), UK, *Commute times increase as the UK's transport system gets more crowded*, 2014. Accessed on 25th March 2015.

[90] Aaron M. Renn, New Geography, *Commuting In New York City, 2000-2010*, 2012. Accessed on 25th March 2015.

[91] National Post, *Travel times prove transit's a non-starter*. Accessed on 25th March 2015.

[92] N. Bakki, *Commuting time of workers in Japan, 2011*, 2014. Accessed on 25th March 2015.

[93] Seoul Metropolitan Government, *Seoul Public Transportation*, 2014. Accessed on 25th March 2015.

[94] W. Cox, *Hong Kong's Decentralizing Commuting Patterns*, 2012. Accessed on 25th March 2015.

[95] National Statistics, Republic of China (Taiwan), *2000 Taiwan Social Development Trends Survey Results*, 2000. Accessed on 25th March 2015.

[96] chinadaily.com.cn, *Commuting time in capital averages 97 minutes*, 2014. Accessed on 25th March 2015.

[97] Vinita Dawra Nangia, The Times of India, *Travel times prove transit's a non-starter*, 2008. Accessed on 25th March 2015.

[98] Land Transport Authority of Singapore, *Fast Facts*. Accessed on 25th March 2015.

[99] SBS Transit, *Bus Memory Lane*. Accessed on 25th March 2015.

[100] R. K. Balan, K. X. Nguyen, and L. Jiang, "Real-time trip information service for a large taxi fleet," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2011.

[101] Microsoft, *Xbox Live*.

[102] R. Herbrich, T. Minka, and T. Graepel, "Trueskill(tm): A bayesian skill rating system," in *Advances in Neural Information Processing Systems 20 (NIPS)*, 2007.

[103] J. Manweiler, S. Agarwal, M. Zhang, R. R. Choudhury, and P. Bahl, "Switchboard: a matchmaking system for multiplayer mobile games," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2011.

[104] S. Agarwal and J. R. Lorch, "Matchmaking for online games and other latency-sensitive p2p systems," in *Proceedings of the ACM SIGCOMM conference on Data communication (SIGCOMM)*, 2009.

[105] C. Ly, C.-H. Hsu, and M. Hefeeda, "Improving online gaming quality using detour paths.," in *Proceedings of the international conference on Multimedia (MM)*, 2010.

[106] C. M. L. McNamara and L. Capra, "Media sharing based on colocation prediction in urban transport," in *Proceedings of ACM 14th International Conference on Mobile Computing and Networking (MobiCom)*, 2008.

[107] Y. Lee, Y. Ju, C. Min, S. Kang, I. Hwang, and J. Song, "Comon: cooperative ambience monitoring platform with continuity and benefit awareness.," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2012.

[108] Y. Lee, C. Min, C. Hwang, J. Lee, I. Hwang, Y. Ju, C. Yoo, M. Moon, U. Lee, and J. Song, "Sociophone: Everyday face-to-face interaction monitoring platform using multi-phone sensor fusion," in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2013.

[109] Singapore Airlines, *Flying with us - Games and applications*. Accessed on 25th March 2015.

[110] F. H. P. Fitzek and H. Charaf, *Mobile Peer to Peer (P2P): A Tutorial Guide*. Wiley, 2009.

[111] The Allseen Alliance. Accessed on 25th March 2015.

[112] B. Davis and H. Chen, "Retroskeleton: Retrofitting android apps," in *Proceeding of the 11th annual international conference on Mobile systems, applications, and services (MobiSys)*, 2013.