University of California
Los Angeles

Automating Personalized Battery Management on Smartphones

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Mohammad Hossein Falaki

2012

Abstract of the Dissertation

Automating Personalized Battery Management on Smartphones

by

Mohammad Hossein Falaki

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2012

Professor Deborah Estrin, Chair

The widespread use of smartphones and proliferation of mobile applications are reshaping many other areas ranging from social networking to health care Today's smartphones are much more capable than before, but mobile application are still restricted by limited resources on smartphones. The key hypothesis of this dissertation is that resource management on smartphones can be improved by adapting to usage patterns of users. We extensively studied users in the wild to characterize smartphone usage. We discovered significant diversity in smartphone usage. Along all aspects that we studied, users differ by one or more orders of magnitude. This finding suggests that resource management policies and algorithms on smartphones can become more effective if they learn and adapt to user behavior.

We developed the prototype of a system that adaptively manages battery, one of the most strained resources on smarthpones, and evaluated its performance. PowerLeash is a system that gives users control over their smartphones' battery lifetime when running background applications. With PowerLeash a user who is running power consuming background applications on her smartphone can decide

how long her battery should last. PowerLeash continuously monitors the phone's battery level, the user's interactions with the phone, and progress of background applications. It builds a personalized model to estimate battery consumption based on usage and background applications progress. Using the on-line model and other information, PowerLeash dynamically adjusts the power consumption of background applications to meet the user's desired battery lifetime. We have designed PowerLeash to be easy to deploy, easy to use, and easy to incorporate in background applications. PowerLeash can run on any Android smartphone as a user level application, and relies only on information that is available to user-level processes. We present the design of PowerLeash and a detailed performance evaluation based on user studies. We use the lessons from deploying PowerLeash on volunteers smartphones to inform future iterations.

The dissertation of Mohammad Hossein Falaki is approved.

Mark H. Hansen

Mani B. Srivastava

Lixia Zhang

Deborah Estrin, Committee Chair

University of California, Los Angeles

2012

To my parents, Mehri and Hassan,

and to my lovely wife, Afsoon.

Table of Contents

List of Figures

List of Tables

Acknowledgments

This research would not have been possible without help and support from many individuals and organizations. Words cannot not do justice to them, but I would like to name a few.

First and foremost I would like to thank my adviser, Deborah, whose support was critical in every stage of this work. Deborah helped me with more than technical advice and mentorship. She taught me how to find research problems that can impact people's lives in a meaningful way, and when I was lost in technical details of research she reminded me not to loose sight of the real world — "the bigger picture." I was very fortunate to work with and learn from her.

I had the opportunity to work with three brilliant researchers from Microsoft, Ratul Mahajan, Srikanth Kandula and Dimitrios Lymberopoulo, during the summer of 2009. Since then Ratul continued mentoring me. I owe him for teaching me rigorous attention to details, and critical thinking.

I was fortunate to benefit from the valuable advice of Ramesh Govindan, Mani Srivastava and Lixia Zhang. Knowing that I could go to three of the best minds in the field and ask for advice at any time gave me peace of mind. Mark Hanson's advice on statistical methods and thinking both inside and outside of classrooms helped me a great deal. Srinivasan Keshav, my master's adviser at the University of Waterloo, was the one who prepared me for Ph.D. research. I am indebted to all these professors.

During the Ph.D. years the Center for Embedded Networked Sensing was my second home , and all CENS members were like family. Nithya Ramanathan, Martin Lukac, Eric Graham, and Hongsuda Tangmunarunkit were as kind and

helpful as elder siblings; they never hesitated to help me with their valuable experience.

Every day I enjoyed working with all the talented folks at UCLA. William Wu, John Hicks, Brent Longstaff, John Jenkins, Joshua Selsky, Donnie Kim, Min Mun, Jinha Kang, Dony Goerge, Cheng-Kang Hsieh, Faisal Alquaddoomi, Eric Wang, Cameron Ketcham, Kannan Parameswaran, Andrew Parker and, and Eric Yuan from CENS, and Zainul Charbiwala, Younghun Kim, Thomas Schmid, Supriyo Chakraborty from NESL, and many others enriched my day-to-day academic and professional experience during these years.

I never had to worry about paper work and administrative details because of Betta Dawson, Wesly Uehara, Dennis Urie, Jeff Goldman, Xuan-Mai Vo, Iris Portillo, Terance Tashiro, and Karen Kim with their excellent support as CENS staff members. I specially enjoyed Betta's help and support with human subject studies.

Even outside of school I learned a lot through technical and non-technical conversations with my friends, Earl, Roozbeh, Hamid, and Ardeshir, just to name a few.

I was very lucky for having Hassan Falaki and Mehri Ahardehi as parents. They were my ultimate source of encouragement for persistence in education since the very early days. During the past ten years that I was away from them, they patiently hid their feelings and anxiety to let me grow and flourish.

And most of all, I wish to thank my lovely wife, Afsoon Alishahi, for her patience and encouragement. Without her, my journey as a Ph.D. student would have been lonely and hard.

Vita

2006	B.S. (Compuer Engineering), Sharif University of Technology, Tehran, Iran.
2006–2008	Research Assistant, Tetherless Computing Lab, University of Waterloo, Ontario, Canada
2006–2008	Teaching Assistant, Computer Science, University of Waterloo, Ontario, Canada
2008	M.Math. (Computer Science), University of Waterloo, Ontario, Canada.
2009	Intern, Microsoft Research, Redmond, Washington.
2010	Intern, Cisco Systems, San Jose, California.
2008–2012	Graduate Student Researcher, Center for Embedded Networked Sensing, UCLA, Los Angles, California.

Publications

Hossein Falaki, Ratul Mahajan, and Deborah Estrin, *SystemSens: A Tool for Monitoring Usage in Smartphone Research Deployments,* ACM MobiArch 2011.

Hossein Falaki, Dimitrios Lymberopoulos, Ratul Mahajan, Srikanth Kandula, and Deborah Estrin, *A First Look at Traffic on Smartphones,* ACM IMC, 2010.

Hossein Falaki, Ratul Mahajan, Srikanth Kandula, Dimitrios Lymberopoulos, Govindan Ramesh, and Deborah Estrin, *Diversity in Smartphone Usage,* ACM MobiSys, 2010.

S Guo, M Derakhshani, H Falaki, U Ismail, R Luk, E Oliver, S Rahman, A Seth, M Zaharia, S Keshav *Design and implementation of the kiosknet system,* Computer Networks, 2011.

Hossein Falaki, Ramesh Govindan, and Deborah Estrin, *Smart Screen Management on Mobile Phones,* CENS TechReport, 2009.

S Guo, H Falaki, E Oliver, S Rahman, A Seth, M Zaharia, S Keshav *Very Low-cost Internet Access Using Kiosknet,* ACM SIGCOMM Computer Communication Review, 2007.

Hossein Falaki *The Wi-Fi Roaming Game,* WINE, 2007.

Earl Oliver, Hossein Falaki, *Performance Evaluation and Analysis of Delay Tolerant Networking,* MobiEval, 2007.

CHAPTER 1

Introduction

1.1 Problem Statement

Mobile phones are being adopted at a phenomenal pace. Cell phone penetration has exceeded those of landline phones, personal computers, television sets and FM radio receivers in both the industrialized world and developing countries [Aho10]. Smartphones, mobile phones that can offer "PC-like" functionality, are a growing subset of mobile phones. Smartphones have already passed PCs in sale globally [can12]. According to industry estimates, in the third quarter of 2011, 44% of Americans have smartphones [nie11] and the number is growing especially among the young. In January 2012 smartphone penetration was 66% among the 24-34 age group.

The widespread use of smartphones is reshaping several other areas of computing, ranging from social networking to health care. However, mobile application are still restricted by limited resources on smartphones. Smartphone limitations, when compared to current personal computers, are more significant in *i*) screen and keyboard size, *ii*) computation power, *iii*) size of fast memory, *iv*) communication bandwidth, and *v*) battery capacity. The small screen and keyboard (when there is a physical keyboard) sizes are direct consequences of the small form-factor of smartphones — the very feature that makes them ubiquitous. Smartphones can be equipped with more powerful processors, excess fast memory, and high

bandwidth wireless technologies. In fact, in recent years processing, memory and wireless bandwidth of smartphones have all dramatically improved, but at the cost of much higher power consumption. Therefore, we argue that battery capacity is the most important performance bottleneck on smartphones.

While transistor density doubles approximately every two years [Sch97] and communication bandwidth increases by 50% every year [Nie98], energy density of batteries used in low power electronics has been doubling every 35 years [Pow95]. On mobile phones, the capacity of Li-ion batteries have been increasing only linearly during the past few years [Neu04] and they are unlikely to keep up with the exponential pace of energy demand on smartphones.

The diagram in Figure 1.1 by Yrjo Neuvo [Neu04] qualitatively demonstrates energy supply and demand trends on cell phones before 2006. The evolution of the iPhone serves as a great example for these trends on modern smartphones. The first generation of the iPhone in 2007 had a 620 MHz ARM based processor with a 1400 mAh battery Li-ion battery. Four years later, in 2011, the latest iPhone 4S featured a 1 GHz dual-core ARM based CPU, but the battery capacity had increased only 20 mAh to 1420 mAh.

Figure 1.1 also shows another fundamental limit of power consumption in pocket-size computers, referred to as the *power density limit*. Consuming more than 3 Watts in a small form-factor, like a smartphone, will generate too much heat and increases the temperature to the extent that the user cannot comfortably hold the phone.

Despite the limited resources, service requirements of smartphones are higher than mobile personal computers, such as laptops. Mobile phones are always with their users and are expected to run 24/7. But despite having larger batteries, many users plug their laptops when they use them for an extended time. Also

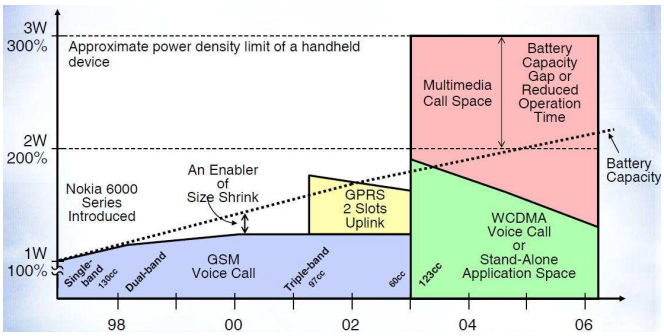


Figure 1.1: Trends of energy supply (battery capacity) and demand on cell phones before 2006 by Neuvo [Neu04] demonstrate a growing gap between the two. This gap is more significant with smartphones.

users do not expect their laptops to continue operation after they close them in their cases.

The growing gap between energy demand and battery capacities on smartphones combined with users' expectations of their phones' battery life time have lead to a dilemma for application developers: having components that continuously run in the background enables new features but renders the users dissatisfied, because it makes the battery life short and unpredictable. For example, mobile health applications and systems can offer much better service if they could continuously monitor users' physical activity on their phones. But because of the adverse impact of continuous background tasks on battery life, mobile health applications with this feature are not popular among the general public.

Unlike fully interactive applications, users do not have direct control over the resource consumption of background tasks. Therefore, the onus is on developers of background applications to select "optimal" duty cycling configurations. We will show in this dissertation that due to user and devise diversity, there is no single optimal configuration. For this reason, most applications with background

components expose one or more parameters to the users to enable them to manually personalize the power consumption configuration. For example, IMAP/POP mobile email clients allow users to change the polling interval. But even if users understood the relation between these configuration parameters and battery consumption rate, they rarely change them because it is cumbersome. [1]

This dissertation presents a solution to the following key problem: **How to enable background applications on smartphones to consume just enough power to meet users' expected battery life time?**

1.2 Approach

Our approach to solving the battery management problem is based on two key assumptions.

Managing only background applications We assume users can effectively control battery consumption of interactive applications. In addition, we try to avoid inhibiting what a user is actively involved with. As an example, we assume that no user would like his/her browsing to slow down because of the intervention of the battery management system. Therefore, we will only manage power consumption of the background components of applications. This makes battery management more challenging, because we have to account for random user interactions.

Personalization In this dissertation we will provide strong evidence that smartphone users are very diverse. In addition, new applications enter smartphone app stores every day which will change usage patterns, and smartphone

[1] In our user studies on two different mobile platforms we found that majority of users never change the default screen timeout value.

hardware specifications change regularly. Therefore, a successful solution to the battery management problem should be able to adapt to a different usage patterns and work on any hardware platform. This requirement has guided many of your design decisions.

1.3 Contributions

Characterizing Smartphone Usage Using detailed traces from 255 users across multiple user studies, we conducted a comprehensive study of smartphone use. We characterized intentional user activities — interactions with the device and the applications used — and the impact of those activities on network and energy usage. We found immense diversity among users. Along all aspects that we studied, users differ by one or more orders of magnitude. For instance, the average number of interactions per day varies from 10 to 200, and the average amount of data received per day varies from 1 to 1000 MB. This level of diversity suggests that mechanisms to manage battery consumption will be more effective if they learn and adapt to user behavior. We found that qualitative similarities exist among users that facilitate the task of learning user behavior. We present these results in Chapter 2.

PowerLeash: A System to Automate Battery Management We built and deployed PowerLeash, a system that gives users control over their phone's battery life time and automates personalization of background applications for developers. PowerLeash monitors the user's interaction with the phone and battery drain, and learns the impact of background applications on battery drain rate. With a simple user interface, PowerLeash receives the user's desired battery life as user input. With this information PowerLeash dynamically changes the

power consumption configuration of background applications to meet the user's battery expectation. In Chapter 3 we present the architecture and design of PowerLeash and results from a user deployment of the system. We closely investigate the shortcomings of our system and propose remedies. In addition, we outline general guidelines that systems similar to PowerLeash can benefit from in Chapter 4.

SystemSens To facilitate smartphone user studies to better characterize smartphone usage and battery consumption we built a comprehensive smartphone monitoring system named SystemSens. We used SystemSens in several user studies to support our research. In addition, we offered the SystemSens source code to the community and more than 20 other researchers have been using it. We present SystemSens in Appendix A.

1.4 Related Work

In this section we present the research and work for each of the three major contributions.

1.4.1 Characterizing Smartphone Usage

In a range of domains, there is a rich history of work that characterizes user workloads. However, because smartphone adoption has gathered pace relatively recently, our work represents one of the few to study how people use smartphones.

Along with other recent works, our findings help complete the picture. Banerjee *et al.* and Rahmati *et al.* report on battery charging behaviors [BRC07, RZ09a]. Like us, they find considerable variation among users. Banerjee *et al.* also propose a predictor that estimates the excess energy of the battery at the

time of charging, using a histogram of past battery usage [BRC07]. Shye *et al.* study the power consumption characteristics of 20 users [SSG09]. They infer properties such as which components in the phone consume most power and explore optimizations based on these characteristics. Rahmati and Zhong study 14 users of a particular demographic to study which applications are popular in that demographic, where the phone is used, and how it is shared among users [RZ09b].

In contrast to these works, we focus on understanding different aspects of smartphone use (e.g., interactions and traffic) and on exposing the diversity of user behaviors, instead of only the average behavior. Our study also entails an order of magnitude more users than previous efforts.

There is a body of work in modeling the aggregate behavior of mobile users. Using traces from a large cellular operator some network related aspects of mobile usage have been modeled. Halepovic *et al.* and Williamson *et al.* report that call arrivals are bursty and present diurnal patterns [WHS05]. Willkomm *et al.* and Brown *et al.* report that mobile users call duration can be approximately modeled by a lognormal distribution [BGM05, WMB08]. We use traces collected on the mobile device itself and focus on modeling the interactions of individual users instead of the aggregate behavior.

Prior to the prevalence of smartphones, Zhong used PDAs to study the energy implications of high user interaction delays relative to the computing hardware and called it the "slow-user problem"

Given the challenge of conducting measurements on mobile devices, existing studies of mobile traffic are based on observations from the infrastructure [SRH06, MSF10, BMV04, Lee06, WPH07, TRK09]. Recently, Maier *et al.* study packet-level traces from residential DSL connections at an aggregation point [MSF10]. Using hints such as HTTP user-agent strings, they identify traffic from mobile

hand-held devices and observe that this traffic is dominated by multimedia content and mobile application downloads. Trestian *et al.* study 3G authentication traces from a provider to measure the correlations between location, time-of-day and application usage [TRK09]. Our approach of monitoring devices is complementary to these studies. It provides a comprehensive view of monitored devices and enables us to study aspects of smartphone traffic such as interaction with radio power management that would otherwise be difficult.

Svoboda *et al.* study the distribution of traffic between access technologies (GPRS and UMTS) and services [SRH06]. Also complementary to our work are studies that conduct active measurements using synthetic workloads [HXM10, CBR04, CCC04, MSZ07]. Such studies can analyze network characteristics in a range of conditions. But they do not provide a view of what users actually experience, which was our focus.

1.4.2 Power and Battery Management

Power-aware computing on mobile devices has been the subject of numerious studies. There is a rich body of literature that views mobile devices as an embedded system with multiple components [Neu04] and aims to reduce the power consumption of one or more components. A commonly used name for this class of techniques is dynamic power management (DPM). In addition to determining when to idle or turn on/off the target component, DPM techniques include voltage and frequency scaling of the CPU [PLS01, PS01, FRM02, YDS02, GCW95, GML00, WWD96], managing brightness and color of the screen [DZ11a, KVM11, ATS11] efficient management of wireless network interfaces [MC11, KK98, SK97], and power-aware memory allocation [DKV01, LFZ00, HPS03].

In contrast to DPM techniques, there are battery drain management (BDM)

System Name	Device	Managed apps	Power model	Development burden	Deployed in field
Odyssey	Laptop	All	Offline high-fidelity	Low	No
ECOSystem	Laptop	All	Offline, high-fidelity	High	No
Cinder	Smartphone	All	Offline, high-fidelity	High	No
Llama	Smartphone, Laptop	Some	Offline, high-fidelity	Low	No
PowerLeash	Smartphone	Background	Online, low-fidelity	Low	Yes

Table 1.1: A high-level comparison of different BDM systems. Unlike previous systems PowerLeash only manages power consumption of background applications and relies on a low-fidelity battery model that can be build on the phone as the system is being used. These two features allowed us to deploy PowerLeash in the wild and evaluate its performance.

techniques. Instead of reducing the power consumed by individual components, BDM allocates power among applications such that the total battery drain rate is compliant with a specified goal. This goal is typically that the battery should last until a certain time (e.g., the next charging opportunity). Prior work shows that DPM and BDM are complementary [FS99].

Table 1.1 lists the main BDM systems that we are aware of and provides a high-level comparison between them and PowerLeash . We first describe these systems and then explain how our work differs.

Odyssey [FS99] was the first BDM system that showed how changing power allocated to applications can help meet user-specified battery deadlines. It estimates the current drain rate and the desired rate for meeting the deadline. Based on the two estimates, it signals applications to increase or decrease their "fidelity," which causes them to consume more or less power.

ECOSystem [ZEL02, ZEL03] and Cinder [RRS11] explicitly allocate the amount of energy consumed by each application. ECOSystem introduces a new unit of energy consumption, called *currentcy*, which represent the right to consume a certain amount of energy within a fixed interval. ECOSystem issues currentcy at the begining of each epoch to applications, based on total desired drain rate (which in turn is based on the battery deadline) and application priority. An application can use a hardware resource (e.g., CPU, NIC) only to the extent to which its currentcy allows.

Cinder generalizes this explicit allocation model. It introduces the abstractions of *reserves* and *taps*, which allow an application to delegate its energy allocation to other applications. They also allow applications to pool their allocations to accomplish tasks that individual applications cannot (e.g., turning of the NIC consumes a lot of power).

Llama [BRC07] is a BDM system with an approach opposite to all the others. Instead of limiting functionality to increase user satisfaction, Llama proposes to increase user satisfaction by taking advantage of the excess energy that is left in the battery before each recharge. This approach by itself is not novel and has been proposed in systems such as SMERT [ZWS06] before. Based on their user study findings, the authors conclude that most recharges happen when the battery has substantial energy left [BRC07]. Llama uses history of battery level information to estimate the energy that is likely be unused (with 90% confidence) the next

time that the phone is plugged to charger. It devotes this energy to increase the functionality of the phone — Llama increases the brightness of the screen and the rate of uploading of a hypothetical ECG application. Llama relies on a high-fidelity model that is built offline, but it builds the histogram of battery levels at charging time online. It estimates the time of next charging event based on past average.

PowerLeash differs from these systems in a few notable ways. First, our goal is to manage the power consumption only of background applications. We posit that this goal better captures users' desires. They do not want, for instance, the browser to be artificially restrained when they are using it actively. They would instead prefer that the power consumed by background applications be reduced if they use the browser heavily. At a technical level, the difference between managing all applications vs. only background ones might appear small at first (e.g., just assign higher priority to interactive applications). But allowing for unrestrained interactive usage means that the total amount of power that can be allocated to managed applications varies (unpredictably) with time, which calls for different allocation mechanisms.

Second, prior systems rely on a high-fidelity power model that is built offline (using specialized equipment) and separately for each hardware type. This makes it hard to extend the system to new devices and applications. To get around this limitation we derive the model online and exclusively from data that is easily available in current commodity smartphones. The resulting model inevitably has lower fidelity—current devices only provide coarse information that makes energy modeling less accurate [SSG09, DZ11b]—but we show that it is nevertheless useful for our purposes.

Third, ECOSystem and Cinder impose a high burden on application develop-

ers. Applications are given a fixed amount of energy resource and developers need to figure out how to best use that resource, which can also be a function of the hardware they are running on. While this gives developers the most flexibility, we believe that it is also a difficult model to implement. In contrast, we provide a much simpler interface to developers. Applications are told how many units of "work" they can do in a given amount of time, where the application itself gets to define what a unit of work is (e.g., it could be polling once for GPS). We consider Odyssey's developer burden to be similarly low; it signals to applications when they should increase or decrease their fidelity.

Finally, PowerLeash is the first BDM system that has been deployed and tested in the wild, under realistic usage and environmental conditions. Prior systems were tested in the lab environment with a handful of chosen applications.

1.4.3 Usage Monitoring

Like us, other researchers have developed logging utilities. MyExperience [FCC07] is one of the earliest tools built to measure device usage and context information *in situ*. It runs on Windows Mobile smartphones and supports active context-triggered experience sampling. SystemSens is designed and implemented for Android smartphones. However, a Windows Mobile port of an early version of SystemSens exists [FLM10]. Unlike MyExperience, SystemSens is a passive logging tool — we chose not to engage with users to minimize impact on usage. SystemSens users are able to tag interesting phenomena regarding their experience on the web interface.

LiveLab [CTZ10] is a similar research tool implemented for the iPhone platform. It measures usage and different aspects of wireless network performance. A key feature of LiveLab is "in-field programmability." The ability to update a

logging tool in the field is critical in any real deployment. We realized this need and implemented a separate tool named *CENS Updater* that can update not only SystemSens but also all other CENS applications in the field. LiveLab is built to run on "jailbroken" iPhones and is capable of collecting a wide range of OS and network related information. We decided to limit the sensing capabilities of SystemSens, but to keep the potential user base as wide as possible by implementing it to run on stock Android smartphones.

Both MyExperience and LiveLab focus on data collection on the phone. SystemSens is an end-to-end system that includes a web-based visualization and authentication service to provide feedback to users while preserving their privacy. In addition, the web interface greatly facilitate browsing and interpreting the data for researchers.

CHAPTER 2

Characterizing Smartphone Usage

Despite the rapid growth of smartphone penetration beyond a few studies that report on users' charging behaviors [BRC07, RQZ07] and relative power consumption of various components (e.g., CPU, screen) [SSG09], many basic facts on smartphone usage are unknown: *i*) how often does a user interact with the phone and how long does an interaction last? *ii*) how many applications does a user run and how is her attention spread across them? *iii*) how much network traffic is generated?

Answering such questions is not just a matter of academic interest; it is key to understanding which mechanisms can effectively improve user experience or reduce energy consumption. For instance, if user interactions are frequent and the sleep-wake overhead is significant, putting the phone to sleep aggressively may be counterproductive [FGE09]. If the user interacts regularly with only a few applications, application response time can be improved by retaining those applications in memory [Esf06]. Similarly, if most transfers are small, bundling multiple transfers [BBV09, SNR09] may reduce per-byte energy cost. Smartphone usage will undoubtedly evolve with time, but understanding current usage is important for informing the next generation of devices.

We analyze detailed usage traces from 255 users of two different smartphone platforms, with 7-28 weeks of data per user. Our traces consist of two datasets. For the first dataset we deploy a custom logging utility on the phones of 33

Android users. Our utility captures a detailed view of user interactions, network traffic, and energy drain. The second dataset is from 222 Windows Mobile users across different demographics and geographic locations. This data was collected by a third party.

We characterize smartphone usage along four key dimensions: *i*) user interactions; *ii*) application use; *iii*) network traffic; and *iv*) energy drain. The first two represent intentional user activities, and the last two represent the impact of user activities on network and energy resources. Instead of only exploring average case behaviors, we are interested in exploring the range seen across users and time. We believe that we are the first to measure and report on many aspects of smartphone usage of a large population of users.

A recurring theme in our findings is the diversity across users. Along all dimensions that we study, users differ by one or more orders of magnitude. For example, the mean number of interactions per day for a user varies from 10 to 200; the mean interaction length varies from 10 to 250 seconds; the number of applications used varies from 10 to 90; and the mean amount of traffic per day varies from 1 to 1000 MB, of which 10 to 90% is exchanged during interactive use. We also find that users are along a continuum between the extremes, rather than being clustered into a small number of groups.

The diversity among users that we find stems from the fact that users use their smartphones for different purposes and with different frequencies. For instance, users that use games and maps applications more often tend to have longer interactions. Our study also shows that demographic information can be an unreliable predictor of user behavior, and usage diversity exists even when the underlying device is identical, as is the case for one of our datasets.

Among the many implications of our findings, an overriding one is that mech-

anisms to improve user experience or energy consumption should not follow a one-size-fits-all mindset. They should instead adapt by learning relevant user behaviors; otherwise, they will likely be only marginally useful or benefit only a small proportion of users.

We show that despite quantitative differences qualitative similarities exist among users, which facilitates the task of learning user behavior. For several key aspects of smartphone usage, the same model can describe all users; different users have different model parameters. For instance, the time between user interactions can be captured using the Weibull distribution. For every user, the shape parameter of this model is less than one, which implies that the longer it has been since the user's last interaction, the less likely it is for the next interaction to start. We also find that the relative popularity of applications for each user follows an exponential distribution, though the parameters of the distribution vary widely across users.

We demonstrate the value of adapting to user behavior in the context of a mechanism to predict future energy drain. Predicting energy drain is an inherently challenging task. Bursty user interactions at short time scales combined with diurnal patterns at longer time scales lead to an energy consumption process that has a very high variance and is seemingly unpredictable. We show, however, that reasonably accurate predictions can be made by learning the user's energy use signature in terms of a "trend table" framework. For predicting the energy use one hour in the future, our predictor's 90th percentile error is under 25%. Without adaptation and basing the predictions on average behavior, the 90th percentile error is 60%.

	#users	Length	Plat.	Demographics	Information logged
DS1	33	7-21 weeks/user	Android	16 high school students, 17 knowledge workers	Screen state, applications used network traffic, battery state
DS2	222	8-28 weeks/user	Windows Mobile	61 SC, 65 LPU, 59 BPU, 37 OP Country: 116 USA, 106 UK	Screen state, applications used
DS3	10	26-84 weeks/user	8 Windows 2 Android	10 Knowledge workers Network: 7 AT&T, 3 T-Mobile	Packet level traces including link layer

Table 2.1: An overview of the datasets in our study. The first is a high-fidelity data set collected by deploying a custom logger on 33 Android smartphones. The second ata set consists of 222 Windows Mobile users. The third dataset contains packet-level traces from 10 smartphone users.

2.1 Data Collection

Our work is based on three sets of data. The first is a high-fidelity data set that we gathered by deploying a custom logger on the phones of 33 Android users. The second data set consists of 222 Windows Mobile users across different demographics. Together, these data sets provide a broad and detailed view of smartphone usage. The thir data set contains packet-level traces from 2 Andriod and 8 Windows Mobile users. We leave for the future the task of studying other smartphone platforms such as iPhone and BlackBerry. The characteristics of our datasets are summarized in Table 2.1.

2.1.1 Dataset1

Our first set of traces is from 33 Android users who ran SystemSens. We introduce SystemSens in Appendix A. These users consisted of 17 knowledge workers and 16 high school students. Knowledge workers were computer science researchers and high school students were interns in a single organization and were recruited by a third person on our behalf. As stated in our study consent form, the users' identities were not revealed to us. The participants were given HTC Dream smartphones with unlimited voice, text and data plans. We encouraged the users to take advantage of all the features and services of the phones. The data was gathered between May and October 2009. There is 7-21 weeks of data per user, with the average being 9 weeks.

2.1.2 Dataset2

Our second data set was collected by an organization that was interested in investigating smartphone usability and application popularity. This organization provided 222 users with Windows Mobile smartphones from different hardware vendors. It also paid for their voice and data plans. For representativeness, the users were drawn from different demographics as shown in Table 2.1. The demographic categories were defined based on what users stated as the primary motivation for using a smartphone. Social Communicators (SC) wanted to "stay connected via voice and text." Life Power Users (LPU) wanted "a multi-function device to help them manage their life." Business Power Users (BPU) wanted "an advanced PC-companion to enhance their business productivity." Organizer Practicals (OP) wanted "a simple device to manage their life." The subjects were asked about their intended use of the phone before the study and were categorized based on their answers. To our knowledge, the results of this study

are not public.

Traces were collected using a logger that recorded start and end time of each application. This information was logged using the ActiveApplication API call of the OS, which reports on the executable that currently has the foreground window (with a callback for changes) Other details that our custom logger in Section 2.1.1 records (e.g., network traffic and battery level) were not logged in this study. Thus, this dataset has lower fidelity than the first one, but it provides a view of smartphone usage across a broader range of users.

The traces were collected between May 2008 and January 2009. There is 8-28 weeks of data per user, with the average being 16 weeks.

2.1.3 Dataset3

Our third dataset is from 8 Windows Mobile (HTC Touch) users and 2 Android (HTC Dream) users. It contains packet-level traces, including link layer headers, for data sent and received by the smartphone. We collected these traces using *Netlog* on Windows Mobile and *tcpdump* on Android. The traces were stored locally and uploaded at regular intervals using the USB connection.

All users are knowledge workers. Each had an unlimited data plan with their carrier (7 AT&T, 3 T-Mobile). The users were resident in two different cities in the USA. Across all users, there is 532 days of data. For individual users, the data varies from 26 to 84 days.

2.1.4 Representativeness of conclusions

An important concern for user studies such as ours is whether the resulting conclusions represent the entire population. There are two potential sources of bias

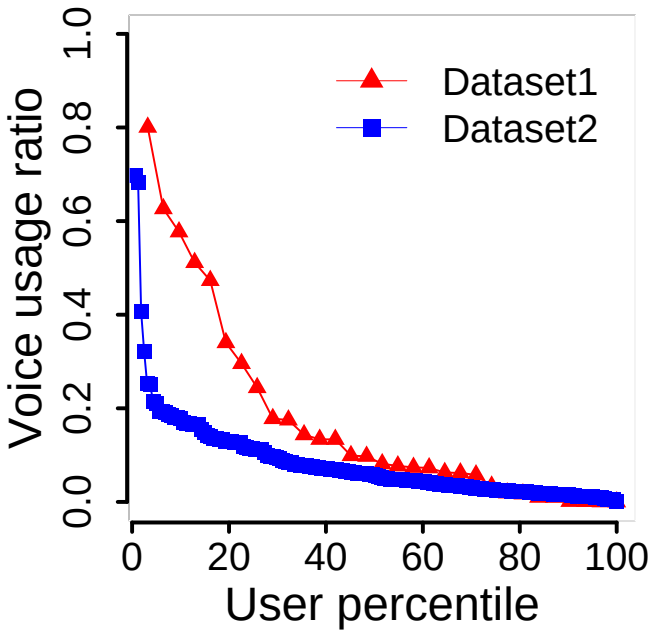


Figure 2.1: Ratio of voice usage to total usage. The x-axis is user percentile and users are sorted in decreasing order of ratio. Voice usage in Dataset1 is higher than that in Dataset2 in which phones were used for personal use.

in our data: *i*) the users are not representative; and *ii*) the measured usage is not representative. We believe that our conclusions are general. The first concern is alleviated by the fact that aside from some quantitative differences, we find remarkable consistency among users in the two datasets. This consistency suggests generality given that the two datasets are gathered independently, on different platforms, and Dataset2 was professionally designed to be representative.

The second concern stems from the possibility that users may not be using the monitored smartphones as their primary devices or that the usage during the monitoring period may not be normal. All users in Dataset2 used the provided smartphones as their primary devices. We do not know this aspect with certainty for Dataset1, but we understand from anecdotal evidence that some users used

these devices as their only phones and others took advantage of the unlimited minutes and text plans. We study voice usage as indicative of the extent to which users relied on the monitored devices. Higher voice usage suggests use as primary phones. Figure 2.1 shows the ratio of time users spent in phone calls to total time spent interacting with the phone (Section 2.2). We see that the overall voice usage in Dataset1 was higher than that in Dataset2 in which all users used the phone as their primary device.

Given that the monitored devices tended to be primary and the long duration of the monitoring interval, we conjecture that our traces predominantly capture normal usage. Earlier work has pointed to the possibility of an initial adoption process during which usage tends to be different than long-term usage [RZ09b]. To show that our traces are not dominated by the initial excitement of users or other special events that cause usage to be appreciably different from the normal usage, Figure 2.2 shows the average interaction time per day (Section 2.2) in the first and second halves of the datasets for each user. We see roughly similar usage in the two halves. Detailed investigation shows that the visible differences in the averages of the two halves, especially in Dataset1, are not statistically significant. Other measures of usage (e.g., network activity) look similar. We do not claim that instances of abnormal usage are absent in the datasets, but the monitored period was long enough for our results to not be impacted by such instances.

2.2 User Interactions

We begin our analysis by studying how users interact with their smartphones, independent of the application used. We characterize application use in the next section, and the impact of user actions on network traffic and energy drain in the following sections.

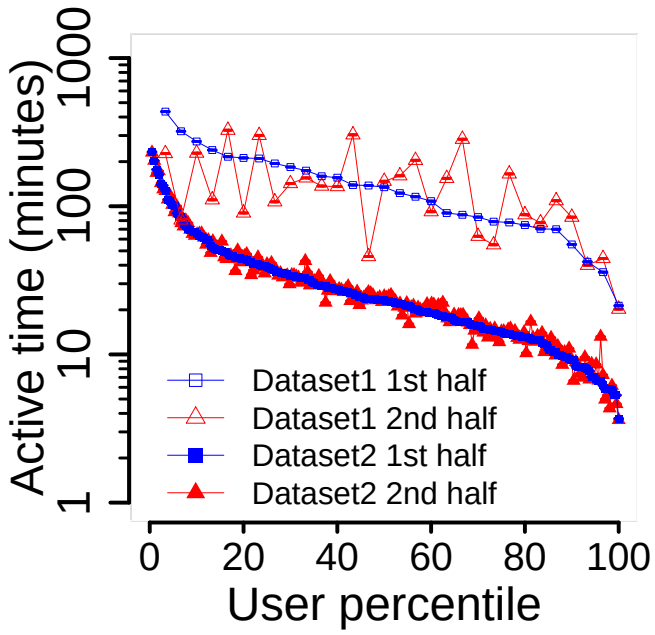
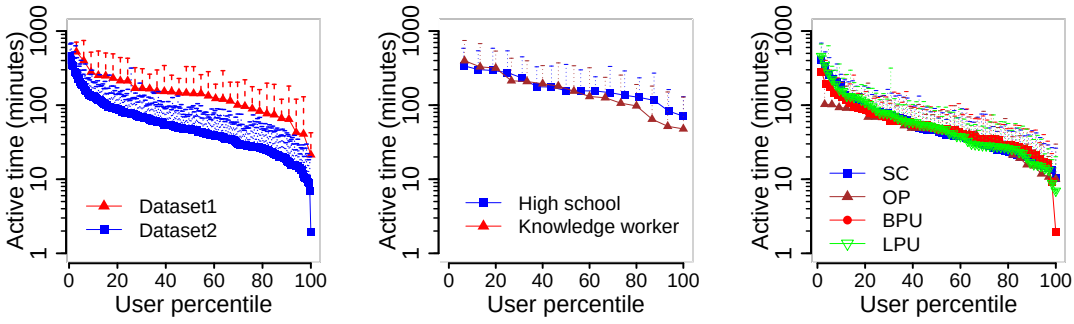


Figure 2.2: Total interaction per day during the first and second halves of study for each user. Users within each dataset are sorted based on the interaction time in the first half. The y-axis is log scale. We see rougly similar usage in the two halves therefore we conclude that usage was not significantly impacted by initial excitement or learning time.

We define an interaction interval, also referred to as a session , differently for each dataset. In Dataset1, we deem a user to be interacting with the phone whenever the screen is on or a voice call is active. In Dataset2, an interaction is defined as the interval that an application is reported to be on the foreground. This includes voice calls because on Windows Mobile a special program ("cprog.exe") is reported in the foreground during voice calls.



(a) All users (b) Dataset1 (c) Dataset2

Figure 2.3: The mean and the upper end of the standard deviation of interaction minutes per day. (a) All users in each dataset. (b)&(c) Different demographics in the two datasets. The y-axes are log scale. Within each dataset there is an order of magnitude difference among suers.

2.2.1 Interaction Time

Figure 2.3(a) shows a basic measure of user interaction—the number of minutes in a day that a user interacts with the smartphone. The plot shows the mean and the standard deviation of this number for each user. For visual clarity, in such graphs, we plot only the upper end of the standard deviation; plotting both ends occludes the other curves. The interested reader can estimate the lower end since standard deviation is symmetric around the mean.

Dataset1 users tend to have more interaction minutes because, as we show later, they tend to have longer interaction sessions while having a similar number of sessions. Within each dataset, however, there is an order of magnitude difference among users. In Dataset1, the lower end is only 30 minutes in a day. But the high end is 500 minutes, which is roughly eight hours or a third of the day. We are surprised by this extremely high level of usage.

Figure 2.3(a) also shows that users cover the entire range between the two

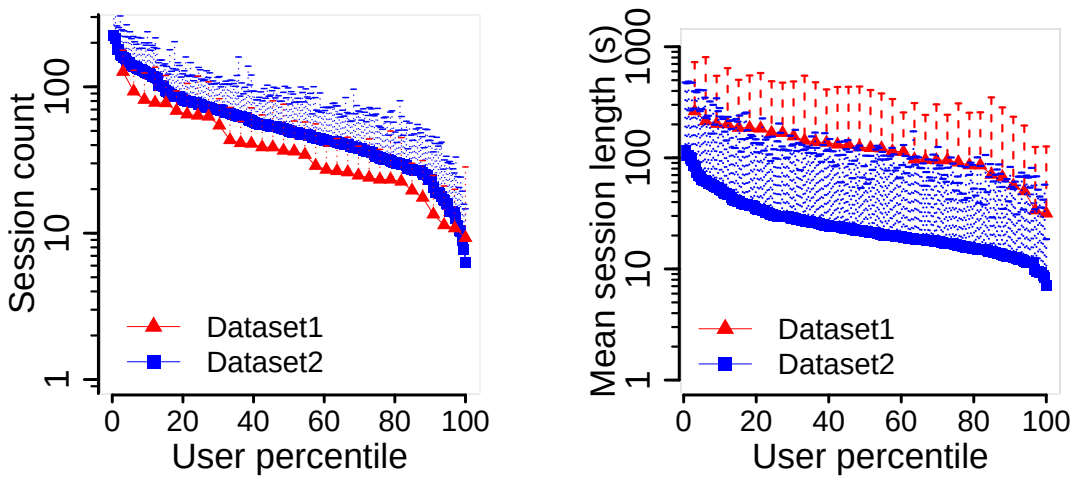
extremes and are not clustered into different regions. The lack of clusters implies that effective personalization will likely need to learn an individual user's behavior rather than mapping a user to one or a few pre-defined categories.

We examine two factors that can potentially explain the extent to which a user interacts with the phone but find that neither is effective. The first is that heavier users use different types of applications (e.g., games) than lighter users. But, we find that the relative popularity of application types is similar across classes of users with different interaction times (Section 2.3.2). The second is user demographic. But, as Figures 2.3(b) and 2.3(c) show, the interaction times are similar across the different demographics in the two datasets. Within each demographic, user interaction times span the entire range. In Section 2.3.2, we show that user demographic does not predict application popularity either.

To understand the reasons behind diversity of user interaction times, we study next how user interaction is spread across individual sessions. This analysis will show that there is immense diversity among users in both the number of interaction sessions per day and the average session length.

2.2.2 Interaction Sessions

Interaction sessions provide a detailed view of how a user interacts with the phone. Their characteristics are important also because energy use depends not only on how long the phone is used in aggregate but also on the usage distribution. Many, short interactions likely drain more energy than few, long interactions due to the overheads of awakening the phone and radio. Even with negligible overheads, battery lifetime depends on how exactly energy is consumed [RVR03]. Bursty drain with high current levels during bursts can lead to a lower lifetime than a more consistent drain rate.



(a) Number of sessions

(b) Session length

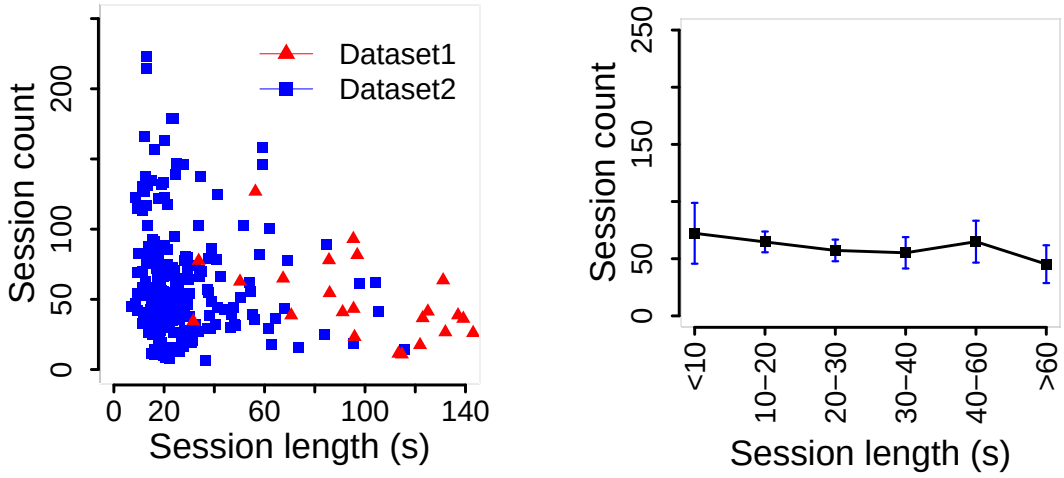
Figure 2.4: The mean and the upper end of the standard deviation for the number of sessions per day and the session length. The y-axes are log scale. We observe a wide range of variation among users and the mean session length varies across users by an order of magnitude.

Figure 2.4(a) shows the number of sessions per day for different users. We again see a wide variation. Individual users interact with their smartphone anywhere between 10 to 200 times a day on average.

Figure 2.4(b) shows the mean and standard deviation of interaction session lengths. Dataset1 users tend to have much longer sessions than Dataset2 users. Given that they have roughly similar number of interactions per day, as seen in Figure 2.4(a), their longer sessions explain their higher interaction time per day, as seen in Figure 2.3(a).

Within each dataset, the mean session length varies across users by an order of magnitude. Across both datasets, the range is 10-250 seconds.

Explaining the diversity in session lengths: Several hypothesis might

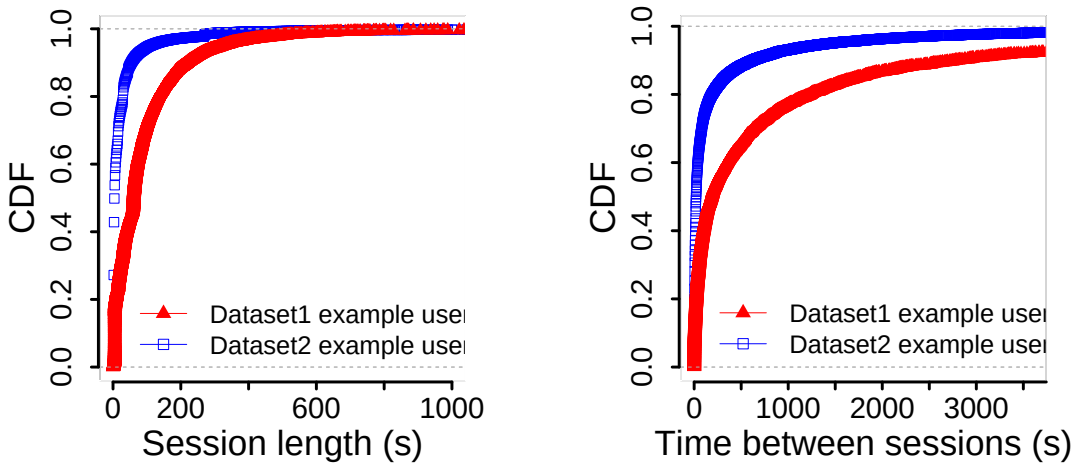


(a) (b)

Figure 2.5: (a) Scatterplot of session count per day and mean session length of various users. (b) The mean and 95% CI of session count per day for users in Dataset2 with different mean session lengths. There is little correlation between users' number of sessions and session length.

explain the differences in different users' session lengths. One hypothesis is that users with longer sessions concentrate their smartphone usage in fewer sessions. Figure 2.5 shows, however, that there is little correlation between users' number of sessions and session length. Figure 2.5(a) shows a scatterplot of session count versus mean length for different users. There is one data point for each user. Figure 2.5(b) shows the dependence of session count on session length by aggregating data across Dataset2 users. It plots the observed mean and 95% confidence interval (CI) for session counts per day for users with different mean session lengths. The differences in the session counts are not statistically significant. In other words, it is not the case that users who have longer sessions have fewer or more sessions.

Our other hypotheses are related to application use. The second hypothesis is



(a) Session lengths

(b) Time between sessions

Figure 2.6: CDFs of session length and time between sessions for two example users. The x-axis ranges in the graphs are different. For any given user, most of the sessions are short but some are very long. The median session length is less than a minute but some are longer than an hour (not shown in the graph)

that users run varying numbers of applications during an interaction, and users that tend to use more applications per session have longer sessions. The third hypothesis is that users run different applications and some applications, such as maps, have longer sessions than others. The fourth one is that even for the same application, users have different session lengths.

Our analysis of application use in §2.3 reveals that the second hypothesis is not explanatory, as users overwhelmingly use only one application per session. It also reveals that the third and fourth hypotheses are likely contributors to diversity in session lengths. Note that the inability of application types to explain interaction time per day, which we mention in the previous section, is different from their ability to explain session lengths.

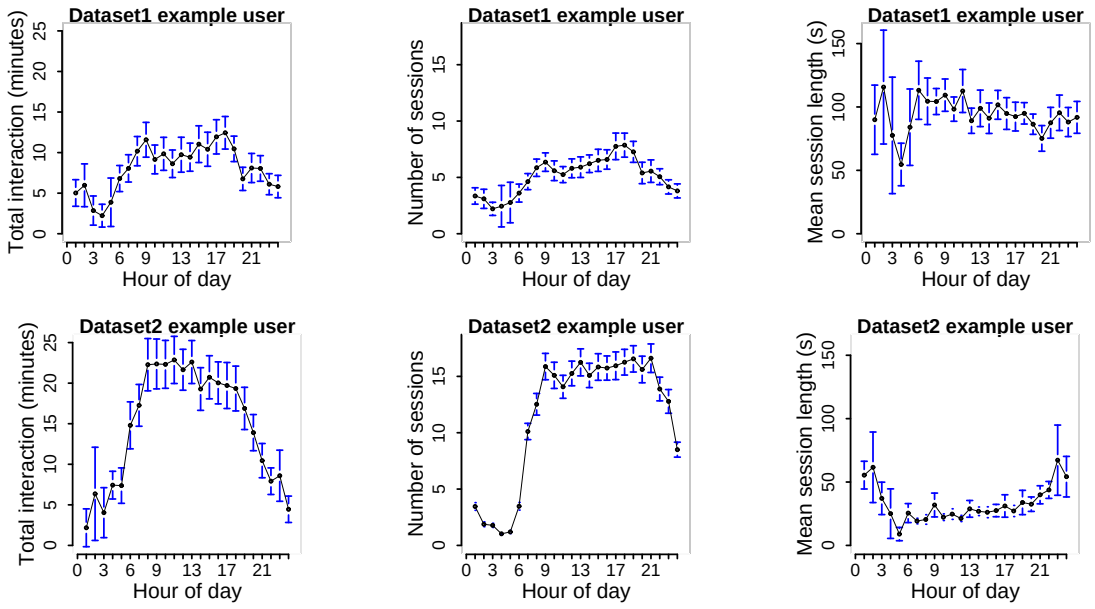
Distribution of a single user's sessions: We find that for any given user, most of the sessions are short but some are very long. Figure 2.6(a) shows the CDF of session lengths for two example users. The median session length is less than a minute but some are longer than an hour (not shown in the graph). A similar skewed distribution can be seen for all users in our datasets, albeit with different median and mean session length values. This highly skewed distribution also explains why the standard deviations in Figure 2.4(b) are high relative to the mean. In Section 2.6.1, we show how session lengths depend on the screen timeout values.

Figure 2.6(b) shows that the time between sessions, when the phone is not used, also has a skewed distribution. Most are short (relative to the mean) but some are very long. We show later that these off periods have the property that the longer a user has been in one of them, the greater the chance that the user will continue in this state.

2.2.3 Diurnal Patterns

We now study diurnal patterns in interaction. The presence of such patterns has several consequences. For instance, the length of time a given level of remaining battery capacity lasts will depend on the time of day.

Figure 2.7 shows for two example users that, as expected, strong diurnal patterns do exist. As a function of the hour of the day, Figure 2.7(a) plots the mean number of interaction minutes per hour. It also plots the 95% confidence interval (CI) around the mean, which can be used to judge if the differences in the means are statistically significant. We see a clear pattern in which daytime use is much higher than nighttime use, though the exact pattern for different users is different.

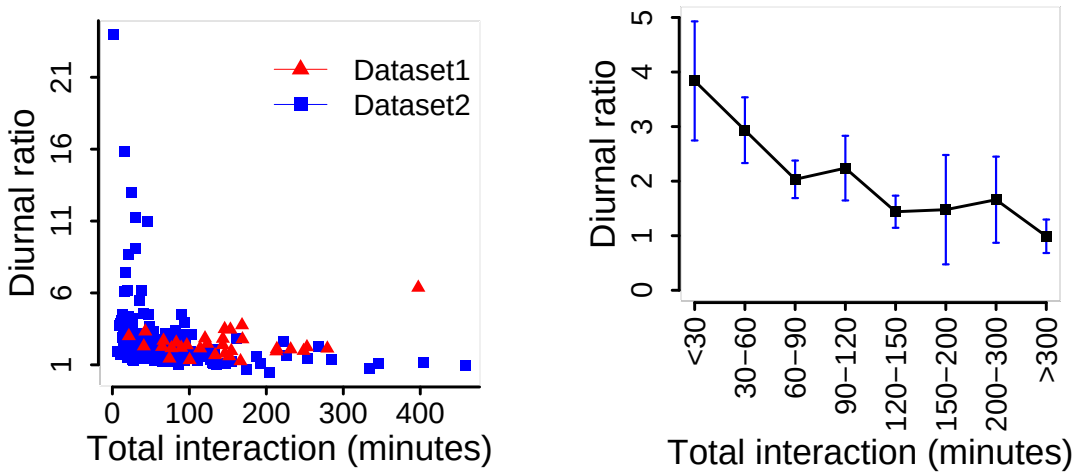


(a) Interaction time (b) Number of sessions (c) Session length

Figure 2.7: The mean and 95% CI of interaction time, number of sessions, and session length during each hour of the day for an example user from each dataset. Daytime use is much higher than nighttime use, and the exact pattern for different users is different.

Figure 2.7(a) also shows that usage at hours in the night is low but not completely zero. We believe that this non-zero usage stems from a combination of irregular sleeping hours and users using their devices (e.g., to check time) when they get up in the middle of the night.

To capture the significance of the diurnal pattern for a user, we define the *diurnal ratio* as the ratio of the mean usage during the peak hour to the mean usage across all hours. A diurnal ratio of one implies no diurnal pattern, and higher values reflect stronger patterns. Figure 2.9(a) plots the diurnal ratio in interaction time for all users. It shows that while diurnal ratios vary across users, roughly 70% of the users in each dataset have a peak hour usage that is more

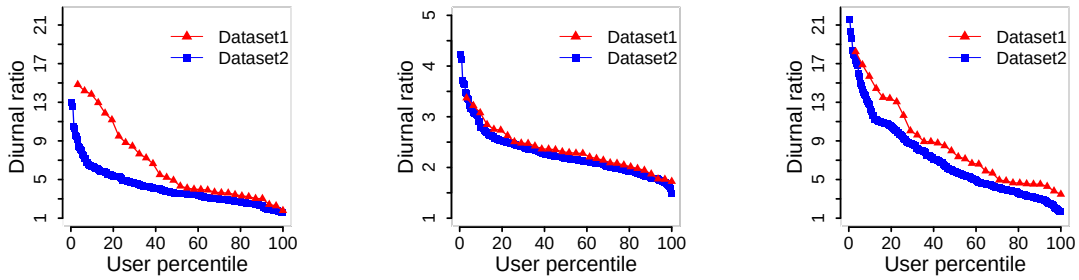


(a) (b)

Figure 2.8: (a) Scatterplot of diurnal ratio of interaction time per hour and interaction minutes per day. (b) The mean and 95% CI of diurnal ratio vs. total interaction time per day. Diurnal ratios vary across users, and roughly 70% of the users in each dataset have a peak hour usage that is more than twice their mean usage.

than twice their mean usage.

Explaining the diversity in diurnal patterns: To help explain the variability among users' diurnal ratios, in Figure 2.8 we study its dependence on interaction time. Figure 2.8(a) shows a scatterplot of the diurnal ratio and the mean interaction time per day. We see that the diurnal ratio tends to be inversely correlated with interaction time. Figure 2.8(b) shows this negative correlation more clearly, by aggregating data across users. It plots the mean and 95% CI of the diurnal ratio of total interaction time per day for users with different total interaction times. The diurnal ratio decreases as interaction time increases. This inverse relationship suggests that heavy users tend to use their phone more con-



(a) Interaction time (b) Number of sessions (c) Session length

Figure 2.9: Diurnal ratio of the interaction time, the number of sessions, and the session length for different users. The y-axis ranges for the number of sessions is different. Users tend to have different number of sessions as well as different session lengths at different hours of the day therefore both variation in number of sessions and session length contribute to variations in interaction time across the day.

sistently during the day whereas light users tend to have concentrated use during certain hours of the day.

Understanding the source of diurnal patterns: The variation in interaction time of a user across the day can result from variation in the number of interaction sessions or the length of individual sessions. We find that both factors contribute. Users tend to have different number of sessions as well as different session lengths at different hours of the day. Figures 2.7(b) and 2.7(c) illustrate this point for two example users. They plot the mean number of sessions and the mean session length for each hour of the day.

Figures 2.9(b) and 2.9(c) show the strength of the diurnal pattern for the number of sessions and session length for all the users. Observe that compared to interaction time and session length, the diurnal ratio of the number of sessions tends to be lower.

2.3 Application Usage

We now study the applications that users run when they interact with their smartphones. Unlike previous attempts to understand mobile application usage [CS08, RZ09b, SLG08] that use diaries and interviews, we rely on the mobile phone's OS to report application usage. We define an application as any executable that the OS reports. On Windows Mobile, we get timestamped records of start and end times of application executions in the foreground. On Android, we log usage counters that are updated by the OS. Every time the OS calls the *onStart*, *onRestart* or *onResume* method of an Android application it starts a timer. The timer stops when the *onPause*, *onStop*, or *onDestroy* method is called. We record periodically the cumulative value of the timer for each installed application. This information on the extent of application use is not as accurate as the equivalent information on Windows Mobile, but it helps us understand relative time spent by the user in each application.

2.3.1 Number of applications

Figure 2.10 shows the number of applications used by users in each dataset over the length of their trace. We see that this number varies significantly, from 10 to 90, across users. The median is roughly 50. We are surprised by this high number given that the iPhone, which is reported to have thousands of applications, is not part of our study. Our results show that avid use of smartphone applications is a trait shared by Android and Windows Mobile users as well.

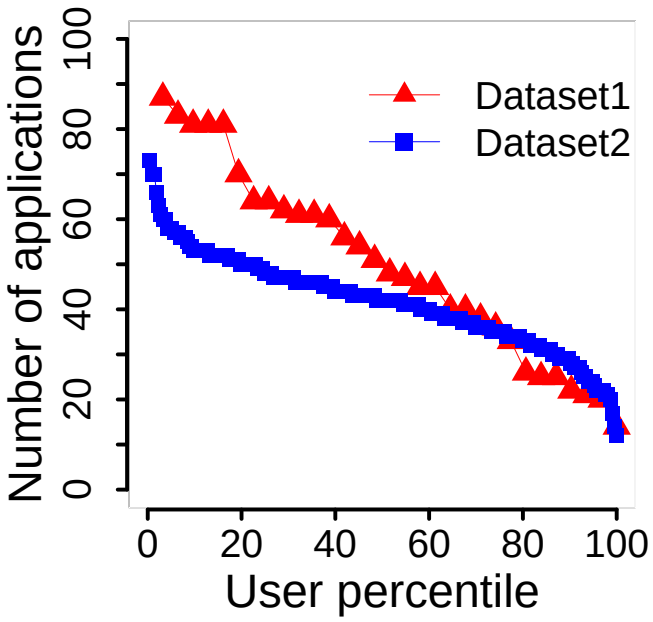


Figure 2.10: Number of applications installed and used by users of each dataset varies significantly across users. In both datasets the median is median number of applications is 50.

2.3.2 Application Popularity

The large number of applications installed by the users does not mean that they use them equally. We find that users devote the bulk of their attention to a subset of applications of their choice. Figure 2.11 illustrates this popularity bias for example users in each dataset. It plots the relative popularity of each application, that is, the ratio of the time spent interacting with the application and the total time spent interacting with the smartphone. The bars show the popularity PDF for the top 20 applications, and the inset shows the semi-log plot for all applications. Because they are binary names, even some popular applications may appear unfamiliar. For instance, in Dataset1, "launcher" is the default home screen application on Android; in Dataset2, "gwes" (Graphical, Windows,

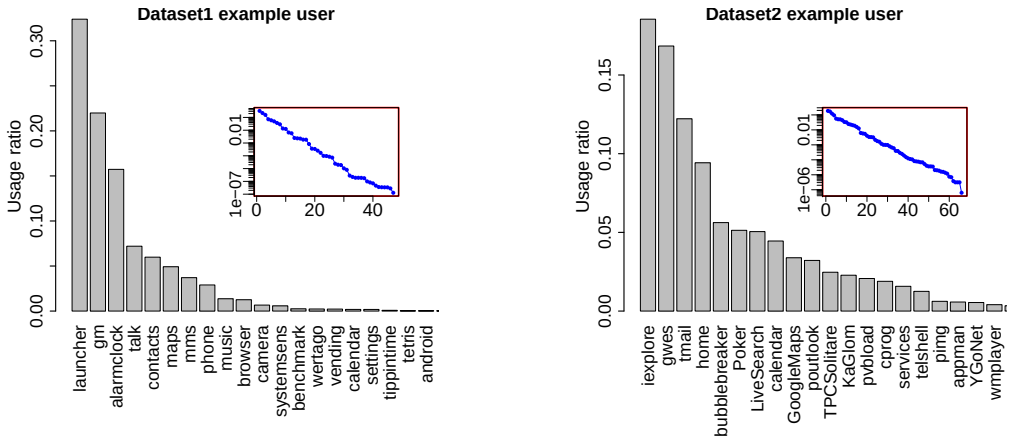


Figure 2.11: Relative time spent running each application for example users in each dataset. Inset is the semi-log plot of application popularity. Clearly users devote the bulk of their attention to a subset of applications of their choice. For each user application popularity can be modeled by an exponential distribution.

and Events Subsystem) is the graphical shell on Window Mobile.

The graphs show that relative application popularity drops quickly for both users. In Section 2.6.3, we show that for all users application popularity can be modeled by an exponential distribution.

Diurnal patterns: Interestingly, application popularity is not stable throughout the day, but has a diurnal pattern like the other aspects of smartphone use. That is, the relative popularity of an application is different for different times of the day. Figure 2.12 illustrates this for an example user in Dataset2. We see, for instance, that tmail.exe, which is a messaging application on Windows Mobile, is more popular during the day than night. Time dependent application popularity was recently reported by Trestian *et al.*, based on an analysis of the network traffic logs from a 3G provider [TRK09]. Our analysis based on direct observation of user behavior confirms this effect.

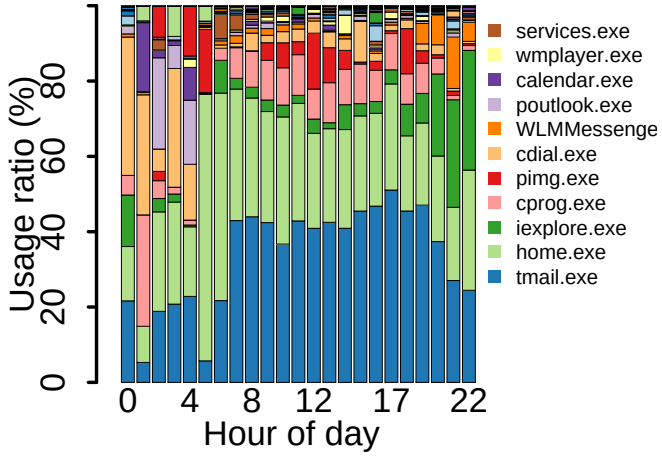


Figure 2.12: Relative time spent with each application during each hour of the day for a sample user and her top applications. We find that relative application popularity is not stable throughout the day and has a diurnal pattern.

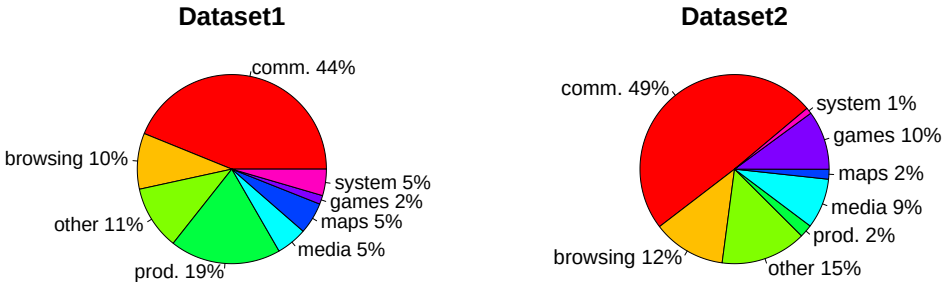


Figure 2.13: Relative popularity of each application category across all users in each dataset. While the results are not identical across the two datasets, they are similar to a first order: Communication dominates in both.

Aggregate view application popularity: To provide an aggregate view of what users use their smartphones for, we categorize applications into eight distinct categories: *i*) *communication* contains applications for exchanging messages (e.g., email, SMS, IM) and voice calls; *ii*) *browsing* contains Web browser,

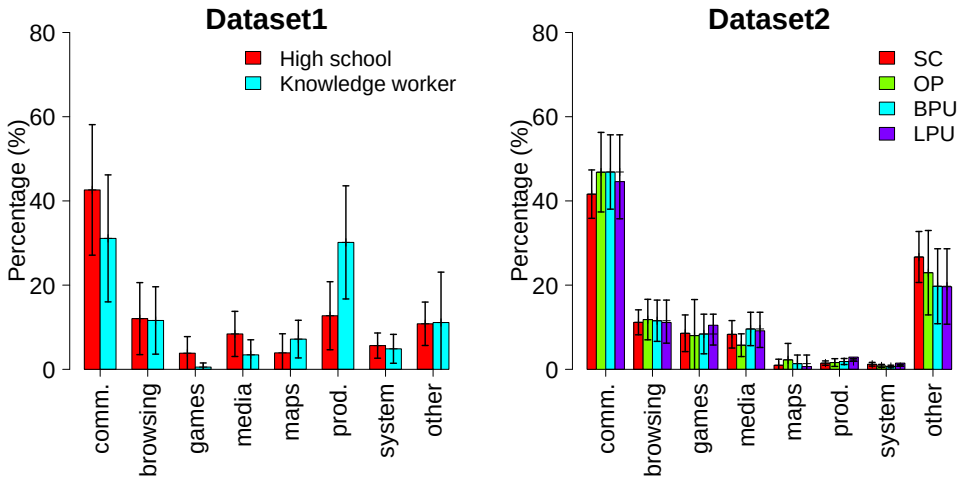


Figure 2.14: The mean and 95% CI of relative popularity of application categories among users of different demographics. User demographics do not seem to impact relative application popularity.

search, and social networking applications; *iii*) *media* contains applications for consuming or creating media content (e.g., pictures, music, videos); *iv*) *productivity* contains applications for calendars, alarms, and for viewing and creating text documents (e.g., Office, PDF reader); *v*) *system* contains applications for changing user preferences and viewing system state (e.g., file explorer); *vi*) games; *vii*) maps; and *viii*) *other* contains applications that we could not include in any of the categories above, e.g., because we did not know their function.

Figure 2.13 shows the mean relative popularity of each application category across all users in each dataset. While the results are not identical across the two datasets, they are similar to a first order. Communication dominates in both. Browsing is another major contributor in both datasets. Maps, media, and games have a comparatively lower but nevertheless substantial share of user attention.

Relationship to user demographic: To understand the extent to which

user demographic determines application popularity, Figure 2.14 shows the mean and 95% CI of relative popularity of each application category for different user demographics. As for interaction time (Section 2.2.1), we see that the impact of user demographics in our datasets is minimal. In Dataset2, the relative popularity of various application types is similar for each of the four demographics. In Dataset1, there are noticeable differences in the mean for communication, games and productivity applications. High school students use communication and games applications more, while knowledge workers use productivity applications more. However, considering the overlap of confidence intervals, these differences in application popularity are not statistically significant.

From this result and the earlier one on the lack of dependence between user demographic and interaction time (Section 2.2.1), we conclude that user demographic, at least as defined in our datasets, cannot reliably predict how a user will use the phone. While demographic information appears to help in some cases (for e.g., the variation in usage of productivity software in Dataset1), such cases are not the norm, and it is hard to guess when demographic information would be useful. Pending development of other ways to classify users such that these classes more predictably explain the variation incorporating factors specific to a user appear necessary. This insensitivity to user demographic has positive as well as negative implications. A negative is that personalization is more complex; we cannot predict a users' behavior by knowing their demographic. A positive implication is that the range of user behaviors along many dimensions of interest can be found in several common demographics. This simplifies the task of uncovering the range because recruiting subjects across multiple demographics tends to be difficult.

Relationship to interaction time: We also study if users that interact

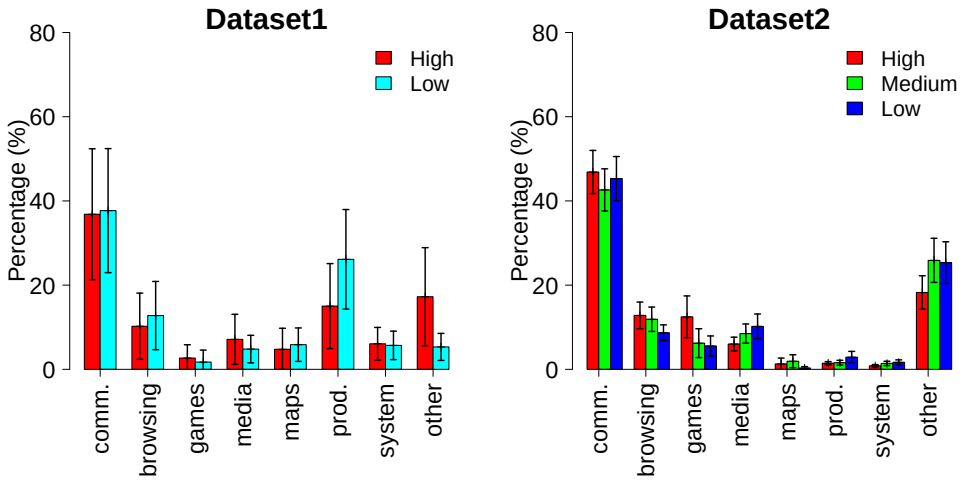


Figure 2.15: The mean and 95% CI of relative popularity of application categories among different classes of users based on interaction time per day. Users in different classes have similar application usage.

more with their phones tend to use different applications. For each dataset, we sort users based on their average interaction time per day and partition them into different classes. For Dataset1, we use two classes, one each for the top and the bottom half of the users. For Dataset2, which has more users, we use three classes for the top, middle, and bottom third of the users. Figure 2.15 shows the mean and 95% CI for relative time spent with each application category by each user class. We see that users in different classes have similar application usage. Thus, we cannot explain why some users use the phone more simply based on the applications that they use.

2.3.3 Application Sessions

We now study the characteristics of application sessions. We conduct this analysis only for Dataset2, based on timestamps for when an application is started and ended; Dataset1 does not contain this information precisely. Because applications

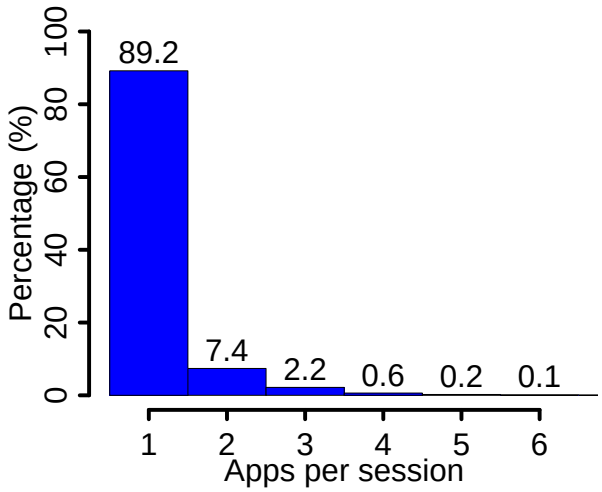


Figure 2.16: Histogram of the number of applications called during each interaction session for all the users in Dataset2. An overwhelming majority of interactions include only one applications therefore users tend to interact with their phone for one task at a time.

can run in the background, start and end refer to the period when the application is in the foreground.

Applications run per interaction: We begin by studying the number of applications that users run in an interaction session. Figure 2.16 shows that an overwhelming majority, close to 90%, of interactions include only one application. This graph aggregates data across all users. We did not find statistically significant differences between users. A large fraction of sessions of all users have only one application.

That interaction sessions very often have only one application session suggests that users tend to interact with their smartphone for one task (e.g., reading email, checking calendar, etc.) at a time, and most of these tasks require the use of only

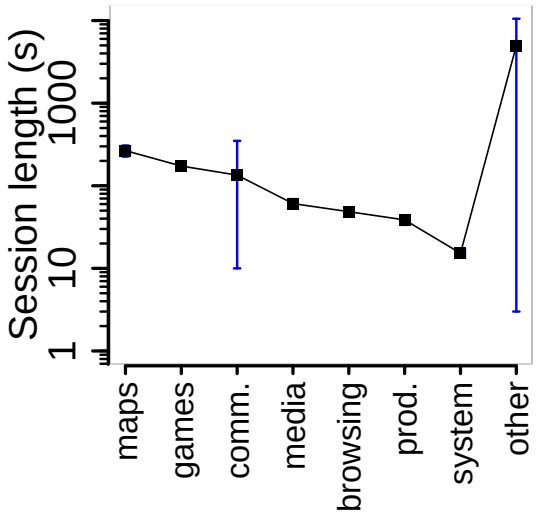


Figure 2.17: The mean and 95% CI of session lengths of different application categories. The y-axis is log scale. Different application types have different session lengths. Maps and games tend to have the longest and those related to productivity and system the shortest.

one application.

Application session lengths: Because interaction sessions are dominated by those with only one application, the overall properties of application sessions, such as their lengths, are similar to those of interaction sessions (§2.2).

However, studying the session lengths of applications separately reveals interesting insights. Different application types have different session lengths, as shown in Figure 2.17, for the categories defined earlier. Interactions with maps and games tend to be the longest and those related to productivity and system tend to be the shortest.

Further, given an application, different users run them for different times. Figure 2.18 shows this effect for a messaging application, tmail.exe, and a brows-

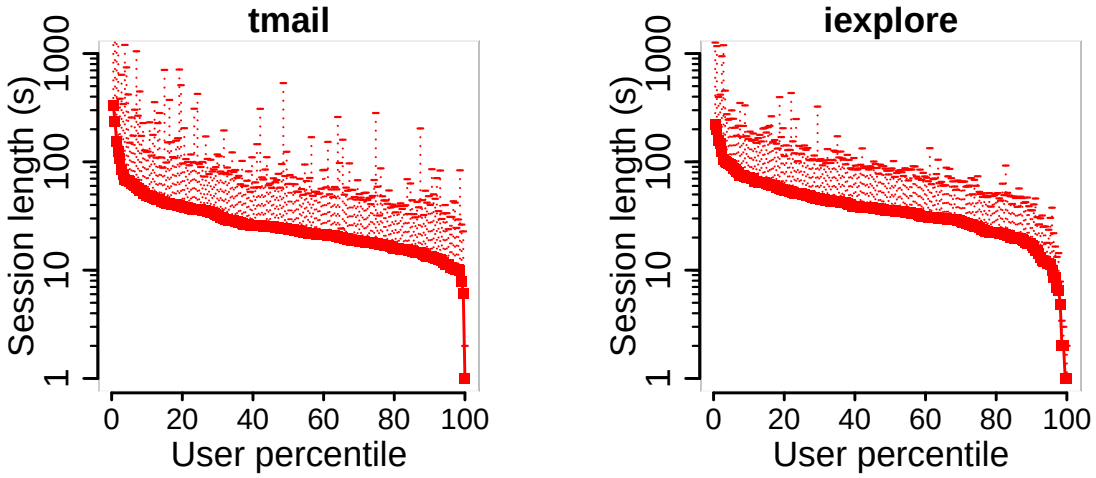


Figure 2.18: The mean and the upper end of the standard deviation of session lengths of two applications. For each application the mean session lengths of users differ by more than two orders of magnitude.

ing application, iexplore.exe. For each application, the mean session lengths of users differ by more than two orders of magnitude.

These observations help explain why users have different session lengths (Section 2.2.2). They prefer different applications and those application sessions tend to have different lengths. Further analysis confirms this phenomenon. For instance, if we divide users into two classes based on their mean session lengths, the popularity of games is twice as high in the class with high session lengths.

2.4 Traffic

In this section, we investigate traffic generated by smartphones. Unlike interaction events and application use, network traffic is not an intentional user action but a side-effect of those actions. Most users are likely oblivious to how much traffic they generate. We show that the diversity and diurnal patterns of this

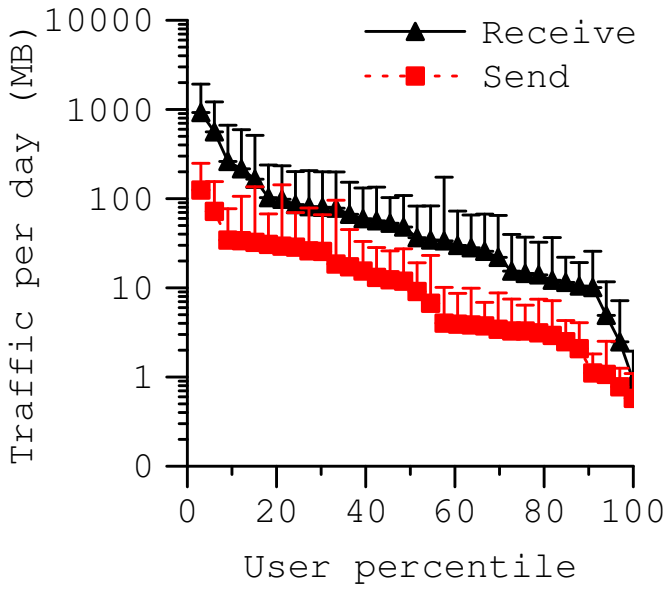


Figure 2.19: The mean and the upper end of the standard deviation of the traffic sent and received per day by users in Dataset1. The amount of sent and received traffic per day varies across users by about three orders of magnitude.

side-effect match those of user actions themselves.

The analysis in this section is based on Dataset1 and Dataset3; we do not have traffic information for Dataset2. In Dataset1, we record all of the data sent (or received) by the phone except for that exchanged over the USB link, i.e., the traffic herein includes data over the 3G radio and the 802.11 wireless link. Dataset3 consists of packet level traces from 10 smartphone users across two different platforms.

2.4.1 Traffic per day

Figure 2.19 shows that the amount of traffic sent and received per day differs across users by almost three orders of magnitude. The traffic received ranges from 1 to 1000 MB, and the traffic sent ranges from 0.3 to 100 MB. The median

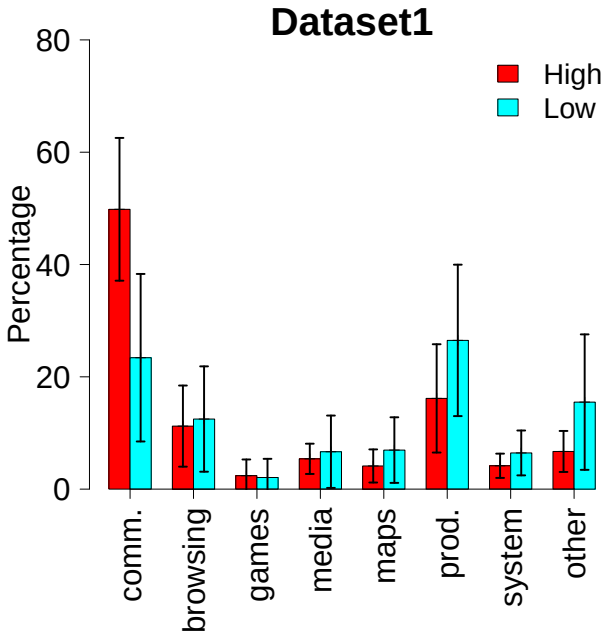


Figure 2.20: The mean and 95% CI of relative popularity of each application category among high and low traffic consumers. Communication applications are more popular among users that consume more traffic.

values are 30 MB sent and 5 MB received.

Our results indicate that traffic generated in a day by smartphone users is comparable to traffic generated by PCs a few years ago. A study of a campus WiFi network revealed that on average users were generating 27 MB of traffic in 2001 and 71 MB in 2003 [KE05]. A study of Japanese residential broadband users in 2006 revealed that on average users generate 1000 MB of traffic per day [FCE05]. This high level of traffic has major implications for the provisioning of wireless carrier networks as smartphone adoption increases.

Relationship to application types: To investigate if certain types of applications are favored more by users that generate more traffice, we divide the users into two equal classes based on their sum of sent and received traffic

per day. Figure 2.20 shows mean and 95% CI of relative popularity of each application category for each user class. Expectedly, it shows that communication applications are more popular among users that consume more traffic.

2.4.2 "Interactive" traffic

We next estimate what fraction of the total traffic is "interactive," that is, has timeliness constraints. Approaches to reduce power consumption of data transfers often advocate rescheduling network transfers, for instance, by delaying some transfers so that multiple of them may be bundled [BBV09, SNR09]. Such policies are difficult to implement for interactive traffic without hurting user response time, and thus are likely to be less valuable if the bulk of the traffic is interactive.

We classify traffic as interactive if it was generated when the screen is on. This classification method might classify some background traffic as interactive. We expect the error to be low because the screen is on for a small fraction of the time for most users. Because certain user interactions with the phone begin right after traffic exchange (e.g., a new email is received), we also consider traffic received in a small time window (1 minute) before the screen is turned on as having timeliness constraints. The results are robust to the exact choice of time window. Indeed some of the traffic that we classify as interactive might be delay tolerant, e.g., a new email to be sent might be deferred for a little while. However, the volume of traffic received by the phone, which users would rather see immediately, dominates by one order of magnitude the volume that is sent and delay-tolerant messaging applications such as email contribute only a small chunk of all traffic.

Figure 2.21 shows the fraction of interactive traffic for each user. We see that for about 90% of the users, over 50% of the traffic is interactive but for the rest almost none of their traffic is interactive. Stated differently, for differ-

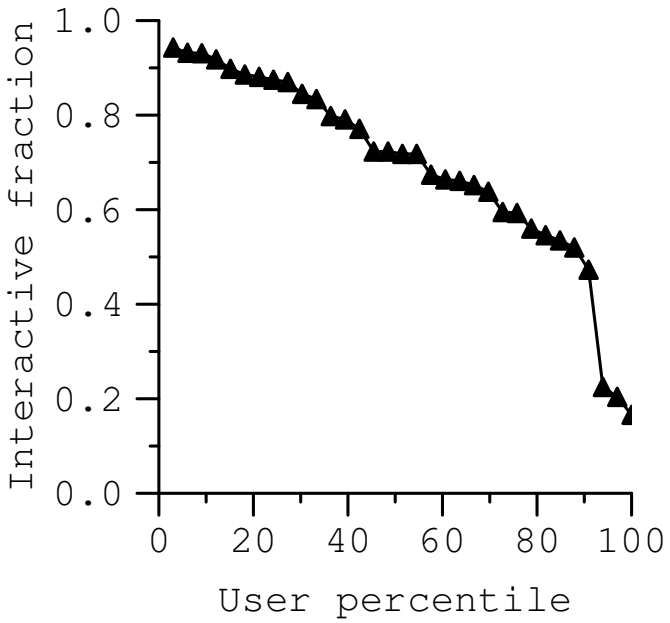
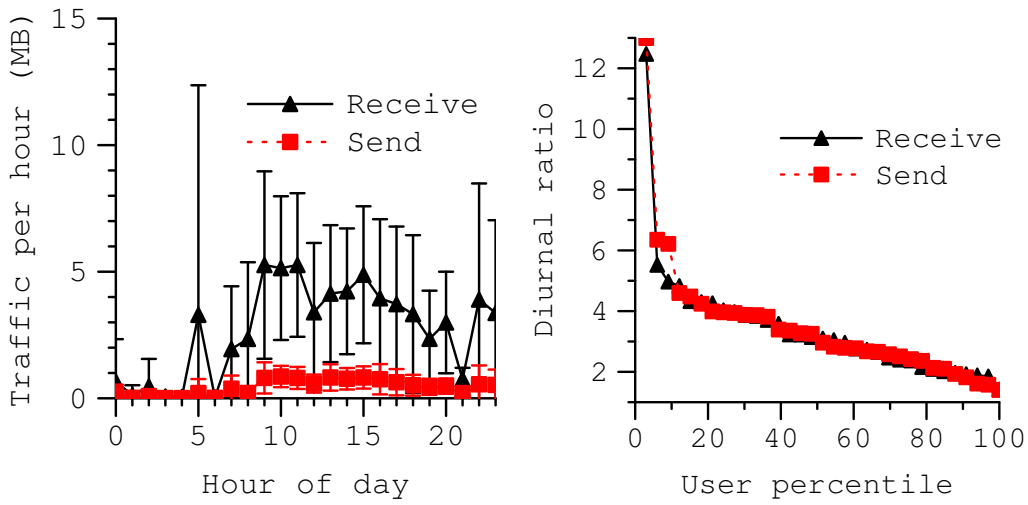


Figure 2.21: The fraction of interactive traffic. For about 90% of the users of 50% of the traffic is interactive. We also observe significant diversity among users in terms of interactive traffic.



(a) (b)

Figure 2.22: (a) The mean and 95% CI for traffic generated per hour by an example user. (b) The diurnal ratio of traffic per hour for all users. We find that diurnal ratio varies across users but most users have a strong diurnal behavior.

ent smartphone users, almost all to almost none of the traffic is generated by applications in the background. The extremes represent disparate ways in which people use smartphones and which applications on the smartphone generate most traffic. Our results imply that the energy savings that can be had by rescheduling network activity will vary across users.

2.4.3 Diurnal patterns

Figure 2.22(a) shows the diurnal pattern for an example user, with a sharp decline at night. Figure 2.22(b) shows the strength of the diurnal pattern for individual users by plotting the diurnal ratio of traffic. As defined in §2.2.3, the diurnal ratio reflects how much higher the mean during the peak hour is to the overall mean.

We see that the diurnal ratio varies across users but most have a strong diurnal

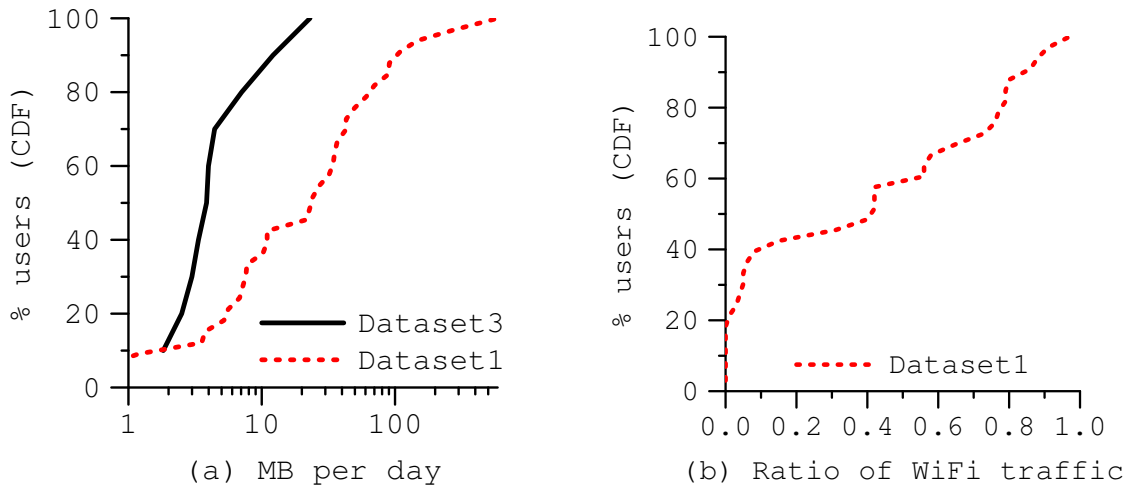


Figure 2.23: (a) Smartphone traffic per day is one order of magnitude smaller than residential broadband traffic. (b) Ratio of traffic sent on the WiFi interface varies widely across users. The median is almost 0.5.

behavior. 80% of them generate over twice their average amount of traffic in their peak hour. This behavior is likely a direct result of the high proportion of interactive traffic for most users and that user interactions themselves have a diurnal pattern.

2.4.4 Traffic composition

In this section, we study the basic makeup of smartphone traffic, starting with volume per user. Figure 2.23(a) shows how much traffic is exchanged per day by users. This amount is 2-20 MB in Dataset3 and 1-500 MB in Dataset1. Compared to residential broadband traffic this is roughly one order of magnitude smaller [cho09].

Two factors may explain the differences in the two datasets. One is that Dataset3 is dominated by Windows Mobile, while Dataset1 is exclusively An-

droid. It is likely that the Android OS and users generate more traffic. The heaviest user in Dataset3 is in fact an Android user. In earlier work, we found that Android users interact with their devices more heavily than Windows Mobile users [FMK10]. For instance, the median session length of Android users is more than twice that of Windows Mobile users.

The second factor, not unrelated to the first, is that many users in Dataset1 use WiFi heavily. Dataset3 does not provide direct information on the interface (WiFi or cellular) used by individual packets. But by observing interface addresses and path delays—cellular delays are much higher—we conclude that WiFi usage was minimal among those users. In Dataset1, we can reliably identify the share of WiFi traffic using information about interface state.

Figure 2.23(b) shows the ratio of WiFi traffic in Dataset1. We see that the median ratio is almost 0.5 but it varies widely across users. While the bottom 20% do not use WiFi at all, the top 20% use it for more than 80% of their traffic. These results also imply that depending on the user population smartphone studies based on only cellular traffic [TRK09] or only WiFi traffic [MSF10] can miss a significant fraction of device traffic.

We now study the composition of smartphone traffic from other perspectives.

Downlink vs. uplink Figure 2.24 shows the ratio of downlink (from the network to the smartphone) to uplink traffic. There is a wide variation among users, caused likely by diversity in application usage, from downlink traffic equaling uplink traffic to it being over 10 times the uplink traffic. The average across all users for downlink to uplink ratio is 6:1. This high asymmetry, indicating a strong bias towards downloads, has implications for provisioning access technologies for smartphones. It is comparable to asymmetry in residential broadband

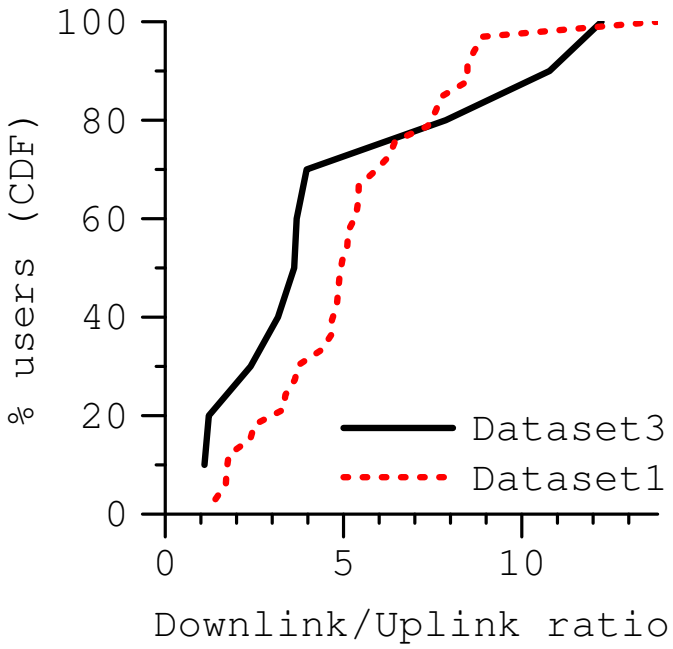


Figure 2.24: Ratio of downlink to uplink traffic. There is a wide variation among users, caused by diversity in application usage. The average across all users for downlink to uplink traffic is 6:1.

traffic in Europe [MFP09] and Japan [FCE05] but is well above the subset of "peer-type heavy-hitters"[FCE05] whose ratio is close to 1:1.

Despite differences in total traffic exchanged the downlink to uplink ratios in the two datasets are similar. This suggests that the mix of activities that generate network traffic may not be disparate for the two cases. We study these next.

Common ports [Dataset3] Ports provide insight into user activities on smartphones. Identifying applications using ports may be inaccurate in some cases (e.g., peer-to-peer), but it is a simple indicator that works well overall [MFP09].

Table 2.2 shows for Dataset3 all ports that carry over 0.1% of the bytes. We

Table 2.2: Ports (in parenthesis) used by IP packets in Dataset1 that carry over 0.1% of the bytes. HTTPS, HTTP and IMAP4S are the dominant ports suggesting that main traffic generators on smartphones are email, browsing and other web-based applications.

see that the dominant ports correspond to HTTPS, HTTP, and IMAP4S. These results suggest that the main traffic generators on smartphones of these users are browsing and email. HTTPS is used by secure Web sites and email servers (including Exchange). HTTP of course is used heavily as part of browsing and for downloading data of various kinds. IMAP4S is the secure version of the IMAP protocol for email. While IMAP4S has the most packets, it is third with respect to the number of bytes. This implies a bias towards small packets, likely generated as part of frequently polling for (often non-existent) new email. Many Dataset3 users had two configured email accounts—a push-based work account and a polling-based personal account. Increased adoption of push-based email, by which clients are notified when new email arrives, will change the nature of email traffic.

The 37% share of HTTP traffic that we find is roughly half of that observed for mobile handheld devices in homes [MSF10] and 50% less than that in residential broadband traffic [MFP09].

The large volume of traffic that uses HTTP or HTTPS perhaps indicates the trend among smartphone applications to tunnel their data through these protocols, even when such data would not normally be considered HTTP payload (e.g.,

Table 2.3: Traffic generated by applications in Dataset1. Browsing dominates smartphone traffic and media and maps are other major contributors in addition to messaging.

music, video and, social apps).

Applications [Dataset1] Our second dataset lets us directly observe what applications are generating smartphone traffic. We partition applications into several categories that are shown in Table 2.3. "System" includes applications that are part of the OS (e.g., package manager, backup), and "Productivity" includes applications for calendars, alarms, and document handling (e.g., Office, PDF reader). The meanings of the other application categories are what their names suggest.

We see in the table that browsing dominates smartphone traffic. As in Dataset3, messaging is also a significant contributor. Media and maps are other major contributors. These applications tend to use HTTP and HTTPS for transport, but the application-level view lets us quantify their contribution independently.

2.4.5 Transfer sizes

In this section, we study the sizes of individual data transfers, which impact throughput as well as power consumption [BBV09]. We identify individual transfers using TCP flows. A TCP flow is identified using IP addresses and ports.

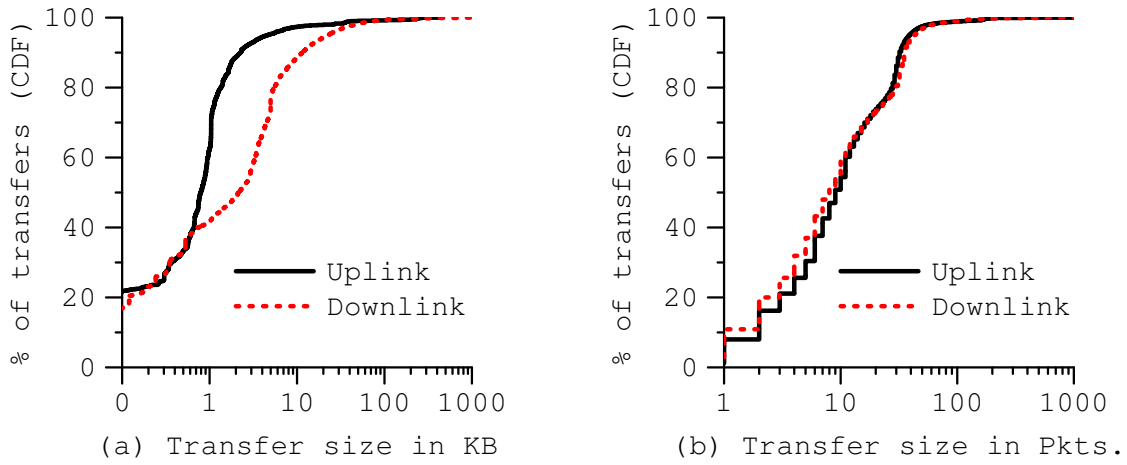


Figure 2.25: Transfer sizes in Dataset3. The x- axes are log scale. While the mean transfer size is 273 KB sent and 57 KB received, most transfers are extremely small and 30% of transferes contain fewer than 1K bytes and 10 packets.

Packets of a flow without an extended idle period (1 minute), are considered as part of one transfer; flows with long idle periods are considered separate transfers [Cla94]. Long idle periods can arise within a TCP flow if the client (e.g., email application) maintains an open connection to the server, to avoid the TCP connection setup overhead for each transfer.

Figure 2.25 shows the CDF of transfer sizes in bytes and packets across all users in Dataset3. The size in bytes includes the bytes contributed by TCP and IP headers. While the mean transfer size is 273 KB sent and 57 KB received, most transfers are extremely small. When considering both directions cumulatively, 30% of them have fewer than 1K bytes and 10 packets. These results are consistent with those of Maier *et al.* for HTTP traffic from handheld devices [MSF10].

Figure 2.26 shows that Dataset1 is dominated by small transfers as well. We define a transfer size differently in this case. Dataset1 has bytes sent and received

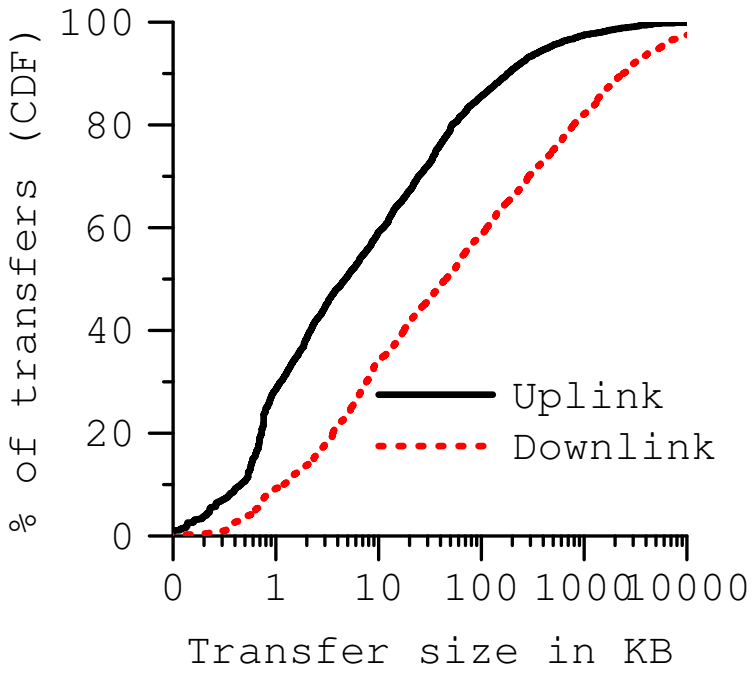


Figure 2.26: Transfer sizes in Dataset1. The x-axis is log scale. Transfer sizes are dominated by small transfers as well.

by individual applications in 2-minute long intervals. For each application, we combine contiguous intervals with non-zero data exchanged as one transfer. If an application exchanges data over multiple TCP connections, this definition aggregates data across those connections into one transfer. Despite this aggregation, the graph shows that most transfers are small.

The small transfer sizes that we observe have many implications. Given the high amount of energy consumed by the 3G radio to go from sleep to ready state and from idle to sleep state, there can be a high energy overhead associated with them. Another implication is that a scheme like Catnap [DS10] is unlikely to reduce radio power consumption. Catnap puts the radio to sleep during transfers but is effective only for long transfers.

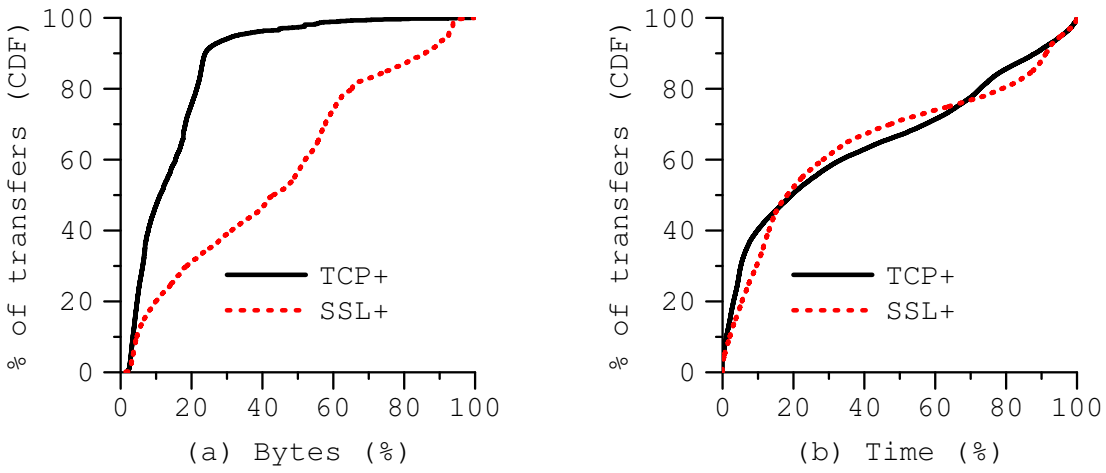


Figure 2.27: The overhead of layers below TCP and SSL (inclusive) in Dataset3. "TCP+" captures overhead of TCP and all layers bellow it. The median TCP+ overhad at byte-level is 12%. "SSL+" captures overhead of SSL and all layers bellow it for SSL-based transfers. The median SSL+ overhad is 40%.

Yet another implication of small transfers is that the already high overhead of lower-layer protocols can dominate. This overhead manifests as extra bytes that must be transmitted as headers as well as extra time that it takes to complete handshakes.

We quantify these overheads at the transport (TCP) and transport security (SSL) layers. According to our analysis 96% of smartphone traffic is TCP-based and more than half uses SSL (through HTTPS and IMAP4S). In Figure 2.27, "TCP+" captures overhead of TCP and all layers below it. "SSL+" captures overhead of SSL and all layers below it, and it is computed only for SSL-based transfers.

Figure 2.27(a) shows that the median TCP+ overhead at byte-level is 12%, i.e., more than one in ten bytes is devoted to TCP or lower layer headers. SSL further increases overhead. The median SSL+ overhead is 40%, and 20% of the

transfers have an overhead that is twice that amount.

Figure 2.27(b) shows the time overhead. TCP+ is measured as the time between the first SYN and the first packet that contains non-TCP bytes. If the radio is asleep when the SYN is sent, this measure will include the time to wake up the radio. In the next section, we quantify the radio waking overhead separately. SSL+ is measured as the time between the first TCP SYN and the first packet that contains non-TCP, non-SSL bytes. Thus, in addition to the TCP handshake, it includes any time needed for SSL key exchange.

We see that the time overhead too is significant. The median overhead of TCP is 20%, i.e., a fifth of the total transfer time is spent waiting for TCP handshake to complete.

Surprisingly, SSL does not add much to the time overhead beyond TCP, which points to the effectiveness of SSL session key caching for smartphone workloads. Smartphones frequently talk using SSL to the same server (e.g., email server). Cached session keys enable quick connection establishment without the overhead of full key exchange. Most of the additional overhead due to SSL appears to be due to their larger headers. In Figure 2.27(b), the SSL+ overhead appears slightly lower than TCP+ in some cases because the two curves are computed over different sets of transfers.

In summary, we find that most smartphone transfers are small. Such transfers have a high energy cost and amplify the overhead of lower-layer protocols. One way to avoid this overhead, which we will investigate in the future, is to aggregate transfers across applications and across time. Using a proxy in the cloud can facilitate such aggregation.

2.4.6 Performance

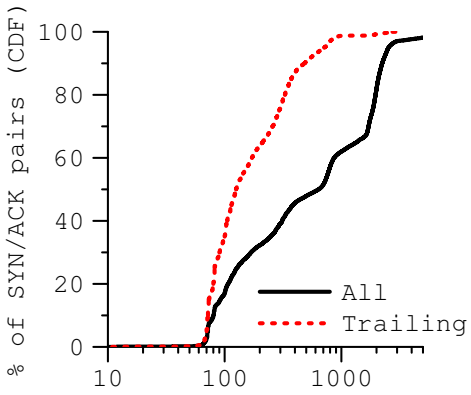
We now investigate the performance of TCP transfers. We study observed round trip time, throughput, and retransmission rate as well as estimate what limits the transfer throughput. We use only Dataset3 because Dataset1 does not have the granularity of information needed for this analysis. As almost all of Dataset3 traffic is 3G-based (§2.4.4), our analysis sheds light on the performance that is seen by smartphones when using the 3G interface in real operating conditions.

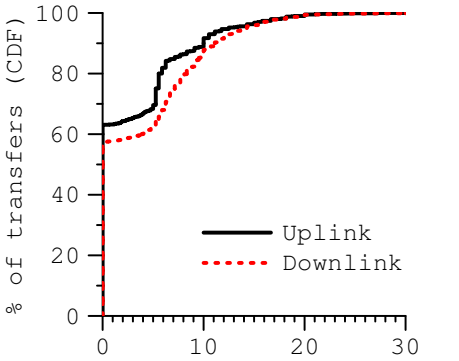
2.4.6.1 Round trip time

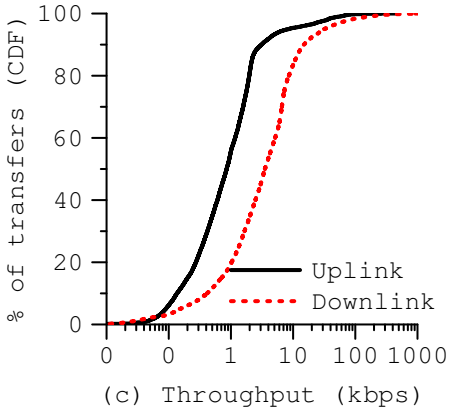
We estimate the RTT of a transfer as the difference between the SYN and SYN-ACK packets. Accurate inference of RTT using data and acknowledgment packets is complicated by delayed acknowledgments. If multiple SYN packets are transmitted for a transfer, we use the last SYN packet. In some cases, the SYN-ACK packet may be sent by a proxy in carrier network instead of the contacted server. Even in these cases, we get a good estimate if the dominant component of the RTT is the wireless delay [HXM10].

We explicitly correct for a source of error that would otherwise significantly overestimate network RTT. If the radio is asleep when the transfer is initiated, it includes the times it takes for the radio to wake up and synchronize with the tower. To weed out this impact, we focus on transfers that are initiated when the radio is in full power mode. We identify such transfers as those that are initiated within 3 seconds of the previous transmission or reception. The idle period after which the radios go into a lower power mode is well above this threshold.

The "Trailing" curve in Figure 2.28(a) shows the RTTs observed by such transfers. We see that the median is 125 ms but 10% of the transfers observe an







Limit	Uplink (%)	Downlink (%)
Packet loss	81.0	61.5
Sender window	4.1	27.4
Receiver window	6.8	3.4
Bandwidth	5.1	2.9
Transport	0.7	0.0
Application	0.0	0.0
Unknown	2.4	4.8

(d) Throughput limits

Figure 2.28: Performance of TCP transfers in Dataset3. (a) median RTT is 125 ms for transfers that happen when the radio is already awake and 10% of transfers observe an RTT of more than 0.5 seconds. (b) Retransmission rate for transfers that transfer send more than 10 data packets in a given direction. 60% of connections observe no retransmissions but 25% of them retransmit 5% of the packets. (c) Throughput of TCP transfers with at least 10 packets in a given direction. Most transfers have very low throughput — the median is 0.8 Kbps for uplink and 3.5 Kbps for downlink. (d) Performance bottleneck analysis based on [ZBP02].

RTT of over 0.5 seconds. Such high variance in RTT is consistent with controlled experiments [HXM10]. A TCP flow that experiences high variance in RTT will suffer from delayed response to congestion among other things. Such variance

can stem from a host of factors including link layer retransmissions (that are not visible to us), network congestion, and overloaded equipment inside the carrier network.

The "All" curve in the graph represents RTTs computed for all transfers, not just those initiated when the radio is awake. The difference in the two curves quantifies the overhead of radio wake-ups. The difference is 400 ms at the median and 1.7 seconds at 90th percentile. The variation stems from the variable amount of time the radio takes to fully synchronize with the tower and a wake-up may already be in progress because of another transfer.

2.4.6.2 Retransmission rate

We now study how frequently packets are retransmitted in TCP transfers. Retransmissions are identified using sequence numbers and provide a good estimate of path loss rate. Their rate can differ slightly from loss rate due to TCP dynamics such as spurious timeouts.

Across all transfers, the uplink retransmission rate is 3.7% and the downlink rate is 3.3%. These loss rates are much higher than those for wired paths. The median average loss rate seen from the SLAC laboratory during 2008 was less than 0.1% for North America and less than 1% for most of the world [CK09]. However, our observed wireless loss rate is similar to those inferred using controlled experiments [HXM10]. We show below that packet loss is the main bottleneck for TCP throughput.

Figure 2.28(b) shows the retransmission rates for individual transfers. This graph is based only on connections that send more than 10 data packets in a given direction. We see that roughly 60% of the connections experience no retransmissions. But 25% of them retransmit 5% of the packets and 10% of them

retransmit more than 10% of the packets.

2.4.6.3 Throughput

As a final measure of smartphone traffic performance, we focus on the throughput observed by TCP connections in our data. Connection throughput is a function of not only path RTT and loss rate but also of application-level factors such as the amount of data.

Figure 2.28(c) shows the throughput of TCP transfers with at least 10 data packets in a given direction. We see that most transfers have very low throughput. The median is 0.8 Kbps for uplink and 3.5 Kbps for downlink. The 90th percentile values are 3 and 15 Kbps respectively.

Given that half the transfers in the analysis above have over 25 data packets, the lack of application data or slow TCP dynamics alone cannot explain the low throughputs that we observe. To understand the bottlenecks, we conduct the analysis of Zhang *et al.* [ZBP02]. This analysis estimates the factor that limits the throughput of a given TCP transfer, based on the timing and sequence of packets. We refer the reader to the original paper for details. The accuracy of this analysis has been evaluated in the wired case but not in a wireless setting. Manual inspection of several cases shows that it provides accurate answers for our data. This gives us confidence that it can yield an accurate aggregate characterization of the type that we present below. We will conduct a more rigorous evaluation in the future.

Figure 2.28(d) shows the results for transfers that have more than 100 data packets in the given direction. We find that with this threshold the analysis yields reliable, consistent estimates. Similar results are obtained with a threshold of 50.

We see that packet loss is the primary limiting factor in both directions,

and the large transfers that we focus on are rarely bottlenecked by transport or application dynamics.

Interestingly, sender window limits over a quarter of the downlink transfers. This suggests that increasing the size of this window (which holds unacknowledged data) at servers will increase the throughput of downlink TCP transfers to smartphones. It is likely that the current buffer sizes are tuned to wired clients which tend to have much lower path delays. In future work, we plan to investigate this issue in detail.

2.5 Energy Consumption

The final aspect of smartphone usage that we investigate is energy consumption. Energy drain depends on two factors: *i*) user interactions and applications; and *ii*) platform hardware and software. If the second factor dominates, the energy drain of various users with the identical smartphone will be similar. Otherwise, the energy drain will be as diverse as user behaviors.

We estimate the amount of energy drain based on the remaining battery indicator which varies between 0 and 100%. If the battery indicator has gone down by X% in a time period for a battery with capacity Y mAh, we compute the energy drain in that period to be $X \cdot Y$ mAh [1]. Given that batteries are complex electro-chemical devices [LDP02, RVR03], this computation is approximate. It assumes that the battery level indicator is linear with respect to energy drain.

Controlled experiments suggest that the linearity assumption holds to a first order. We run a benchmark load that drains the battery at a fixed rate in room

[1] mAh is technically a unit of charge, yet is commonly used to indicate energy drain because battery voltage during normal operations is typically constant. For phones in our dataset, this is 4V, so multiplying a mAh reading by 4 would yield an accurate energy reading in milli-watt-hours.

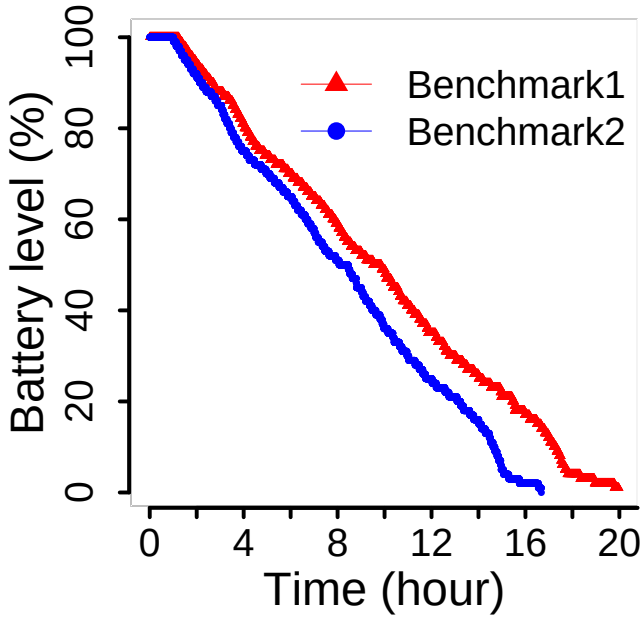


Figure 2.29: Timelapse of the remaining battery level indicator in controlled experiments with two different workloads at room temperature. Benchmark1 turns the screen on and off periodically. Benchmark2 computes and idles periodically. This graph suggests that the level indicator can be used to estimate energy drain.

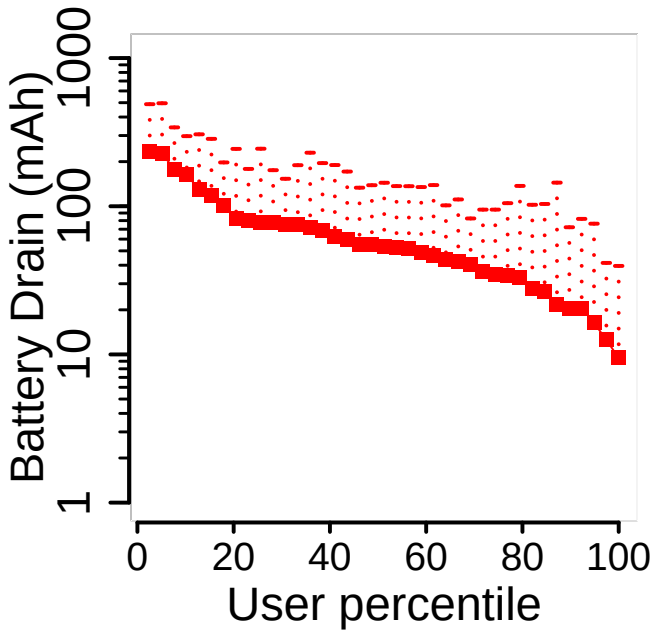
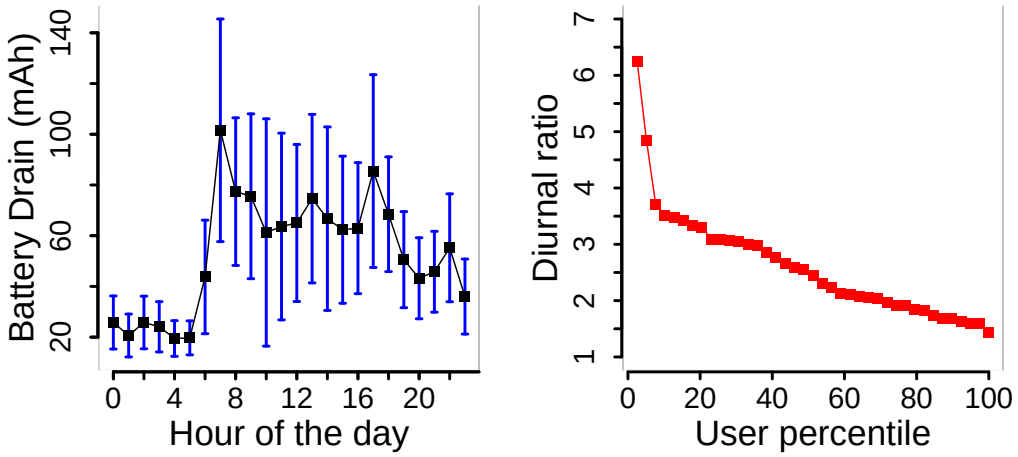


Figure 2.30: The mean and the upper end of the standard deviation of one hour energy drain for Dataset1 users during discharge periods. Battery level indicator decreases roughly linearly for two different benchmarks. We conclude that the level indicator can be used to estimate energy drain.

temperature. Under this benchmark, if the battery level indicator decreases linearly with time, it must be linear with respect to energy drain. Figure 2.29 shows that the level indicator decreases roughly linearly for two different benchmarks. Benchmark1 turns the screen on and off periodically. Benchmark2 computes and idles periodically. We conclude thus that the level indicator can be used to estimate energy drain.

Figure 2.30 shows the mean and standard deviation of energy that users drain in an hour. This graph is computed using only periods in which the battery is not charging because energy drain in those periods are of primary interest. We see a two orders of magnitude difference among users. While heaviest users drain



(a) (b)

Figure 2.31: (a) The mean and 95% CI of energy drain of an example Dataset1 user. (b) Diurnal ratio of all users in Dataset1. We find two orders of magnitude difference among users. While heaviest users drain 250 mAh, the lightest of the users drain only 10 mAh.

close to 250 mAh the lightest of users drain only 10 mAh. If the battery capacity is 1200 mAh, this leads to a lifetime variation from about 4 to 120 hours.

Figure 2.31(a) shows for an example user that the drain is not the same throughout the day but has diurnal variations in which more energy is consumed during the day than during the night. For this user, the level of energy consumed changes by roughly a factor of five. Figure 2.31(b) plots the diurnal ratio of energy use for all users. It shows that diurnal variations occur, with different strengths, for all users.

Our results show that user activities contribute heavily towards energy drain; users in Dataset1, who have identical smartphones, drain energy at different rates, and energy drain has diurnal patterns. In the future, we will develop methods to accurately quantify the energy consumption of the platform from that due to

user-induced workload.

We uncover a surprising level of diversity among smartphone users. For almost every aspect of usage that we study, we find one or more orders of magnitude difference between users. Our findings strongly motivate the need for customizing smartphones to their users. We believe that this need is greater than that for customizing ordinary cellphones or laptops. Ordinary cellphones do not have as rich an application environment. Laptops are not as portable and are more resource rich. For example, many users plug-in their laptops while using them.

Customization can help at all levels. Consider something as low-level as the battery. Suppose we want batteries to be both lightweight and last for at least a day with a high probability. Meeting the latter goal for all users of a given platform will require catering to the heaviest users. But that will lead to unnecessarily heavy batteries for many users. (Higher capacity batteries are heavier.) Offering multiple types of batteries with different lifetime-weight tradeoffs provides a way out of this bind.

At levels where intelligent mechanisms to improve user experience or reduce energy consumption reside, user diversity motivates adapting to the smartphone user. Driving these mechanisms based on average case behaviors may not be effective for a large fraction of the users.

The ease and utility of customization depends on two properties of user behavior. First, despite quantitative differences, there must be qualitative similarities among users. For instance, we should be able to describe the behavior of all users with the same model. Different users may have different parameters of this model, which will then lead to quantitative differences among them. The presence of qualitative similarities imply that users are not arbitrary points in space, and it significantly simplifies the task of learning user behavior. Second, user be-

havior in the past must also be predictive of the future. Otherwise, customization based on past behaviors will be of little value in the future.

In the next two sections, we present evidence that these properties hold for several key aspects of smartphone usage. In Section 2.6, we show that user sessions and relative application popularity can be described using simple models.

2.6 Smartphone Usage Models

In this section, we develop simple models that describe three aspects of smartphone usage – session lengths, inter-arrival time between sessions, and application popularity. The models are common across users but have different parameters for different users. While they do not completely describe user behavior, they capture first order factors and represent a first step towards more complete modeling of smartphone usage. More importantly, along with the results of the next section, they show that qualitative similarities do exist among users.

2.6.1 Session Lengths

We first consider the statistical properties of session length distributions of users. We find that session length values tend to stationary. With the KPSS test for level stationarity [KPS92], 90% of the users have a p-value greater than 0.1. The presence of stationarity is appealing because it suggests that past behavior is capable of predicting the future.

We also find that session lengths are independent, that is, the current value does not have a strong correlation with the values seen in the recent past. With the Ljung-Box test for independence [LB78], 96% of the users have a p-value that is greater than 0.1.

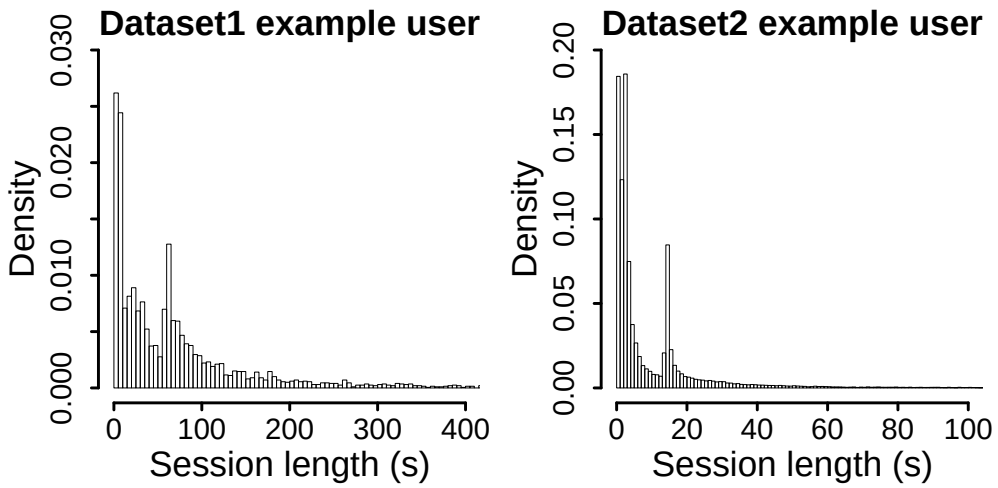


Figure 2.32: The histogram of session length for sample users of each dataset. Most interaction sessions are very short and the frequency drops as the length increases. However, inconsistent with exponential behavior there are some very long sessions. In addition, there is a spike in frequency of session length for each user.

Stationarity and independence, considered together, suggest that session length values can be modeled as i.i.d samples from a distribution. Choosing an appropriate distribution, however, is complicated by the nature of the session lengths. Most sessions are very short and the frequency drops exponentially as the length increases. However, inconsistent with exponential behavior, there are some very long sessions in the tail for each user.

We find that a mixture of exponential and Pareto distributions can model both ends of the spectrum. The former captures short sessions and the latter captures long sessions. That is, session lengths can be described by the following mixture model:

$$r \cdot Exp(\lambda) + (1 - r) \cdot Pareto(x_m, \alpha) \tag{2.1}$$

In this equation, r is the relative mix of the two distributions, λ is the rate of the

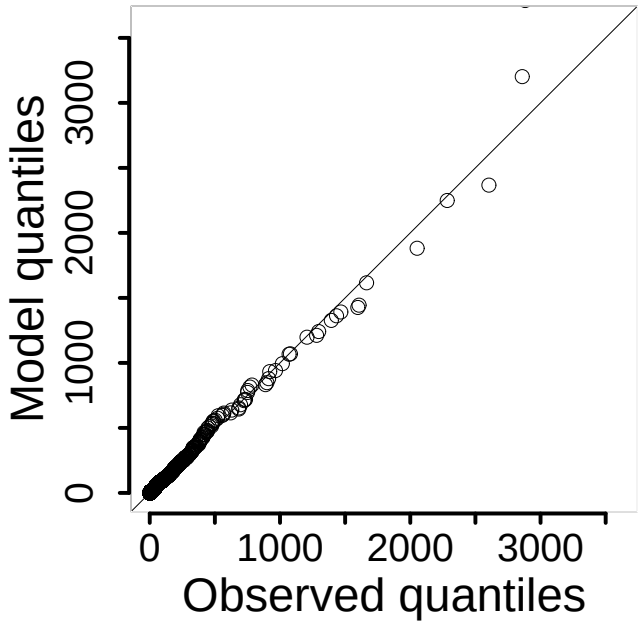


Figure 2.33: QQ plot of session lengths model for a sample user. The linearity of the fitted line graphically indicates that the mixture model is a good fit for session length values.

exponential, and x_m and α are the location and shape parameters of the Pareto distribution.

The location for a Pareto distribution represents the minimum possible value of the random variable. The location value that offers the best fit is the screen timeout value of the user, because the session length PDF has a spike at this value. The spike corresponds to short sessions that are ended by the timeout (when the user forgets to switch the screen off); we confirm this using controlled experiments with different timeout values. Figure 2.32 shows this spike, at 60 and 15 seconds, for example users from each dataset. The timeout provides a natural division between the two component distributions. We automatically infer its approximate value using a simple spike detection algorithm.

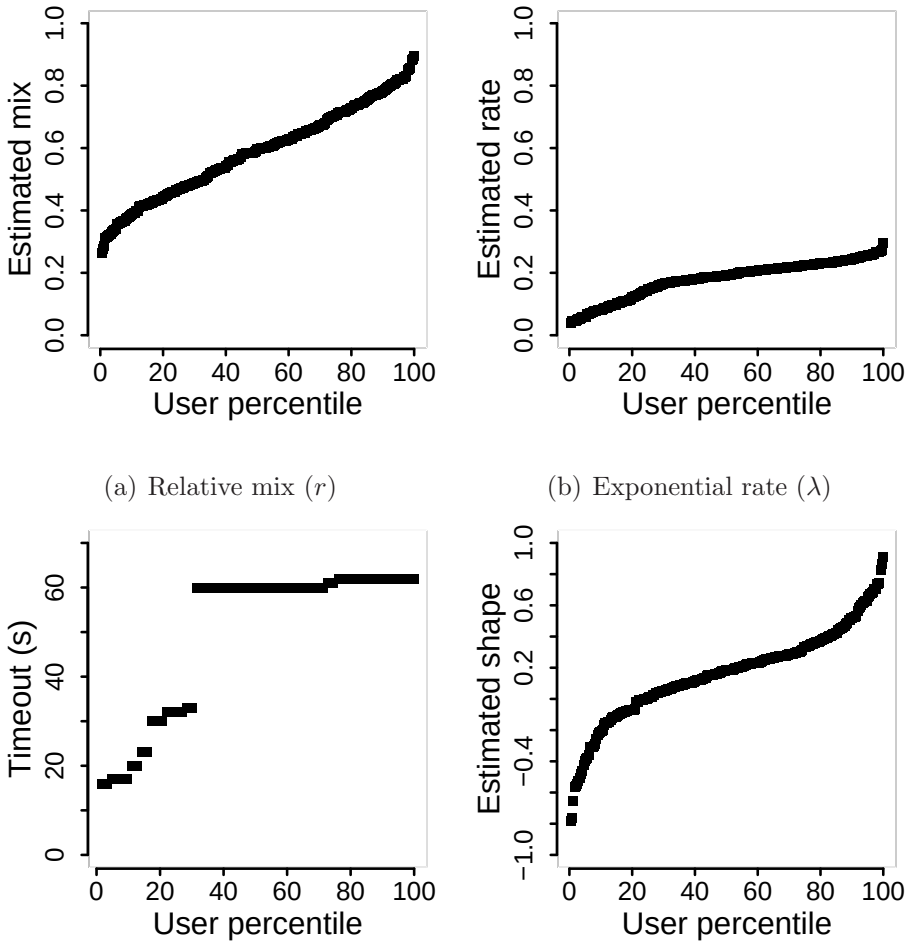
We use the EM algorithm to infer the maximum likelihood estimation (MLE) of the remaining three parameters [DLR77]. Figure 2.33 shows the quality of this fit for an example user using the QQ plot [BCW88]. Almost all quantiles are along the $y = x$ line, indicating a good fit.

Figure 2.34 shows the four inferred parameters for various users. While users can be modeled using the same mixture model, the parameters of this model vary widely across users. Because of the way we construct our model, the distribution of the parameter r and x_m also provide insight into how frequently users' screen is switched off by the timeout and the relative popularity of different timeout values. 60 seconds is the most popular value, likely because it is the most common default timeout and many users never change the default.

2.6.2 Time between Sessions

We find that the Weibull distribution can explain the screen off times. This distribution has two parameters referred to as its scale and shape. We find the MLE for these parameters for each user. From the QQ-plot in Figure 2.35, we notice that the model predicts a greater probability of seeing some very large offtimes than are observed in the datasets. However, the probability of seeing these large offtimes is small; there are 2.7% data points that have a y-value greater than 8000 in that graph. Hence, we believe that Weibull provides a good fit for the length of intervals between interactions.

Figure 2.36 shows the distribution of the estimated shape and scale of the fitted Weibull distributions. Interestingly, the shape is consistently less than one. Weibull shape values less than one indicate that the longer the screen has been off, the less likely it is for it to be turned on by the user. This behavior has interesting implications for power saving policies. For instance, periodic activities such as



(a) Relative mix (r)

(b) Exponential rate (λ)

(c) Pareto location (x_m)

(d) Pareto shape (α)

Figure 2.34: Distribution of inferred model parameters that describe session length values of users in both datasets. While the users can be modeled using the same mixture model, the parameters of this model vary widely across users. The distribution of the Pareto location parameter indicates that most users never change the default timeout of the screen.

checking for email when the screen has been off for a while may be deferred or rescheduled if needed without hurting user experience.

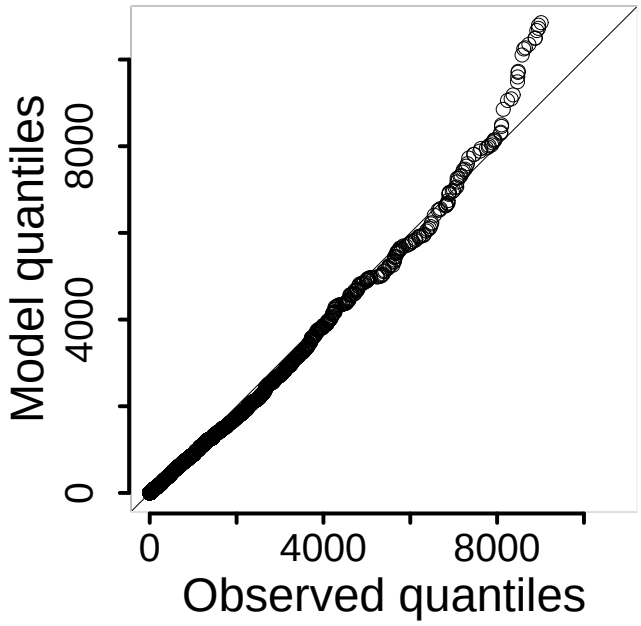
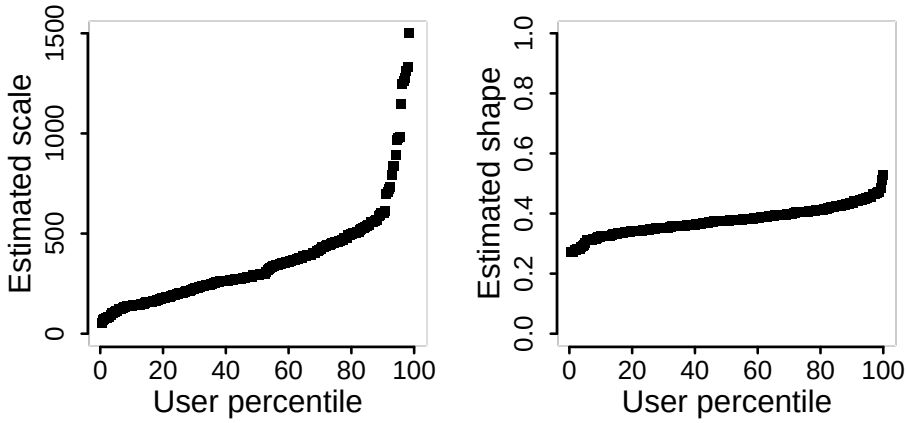


Figure 2.35: QQ plot of session offtime model for a sample user. The linear relation between model quantiles and observed quantiles graphically suggests that the model fits the data well.

2.6.3 Application Popularity

We find that for each user the relative popularity of applications can be well described by a simple exponential distribution. This qualitative invariant is useful, for instance, to predict how many applications account for a given fraction of user attention. For the example users in Figure 2.11, this facet can be seen in the inset plots; the semi-log of the popularity distribution is very close to a straight line.

Figure 2.37(a) shows that this exponential drop in application popularity is true for almost all users; the mean square error between modeled exponential and actual popularity distribution is less than 5% for 95% of the users.



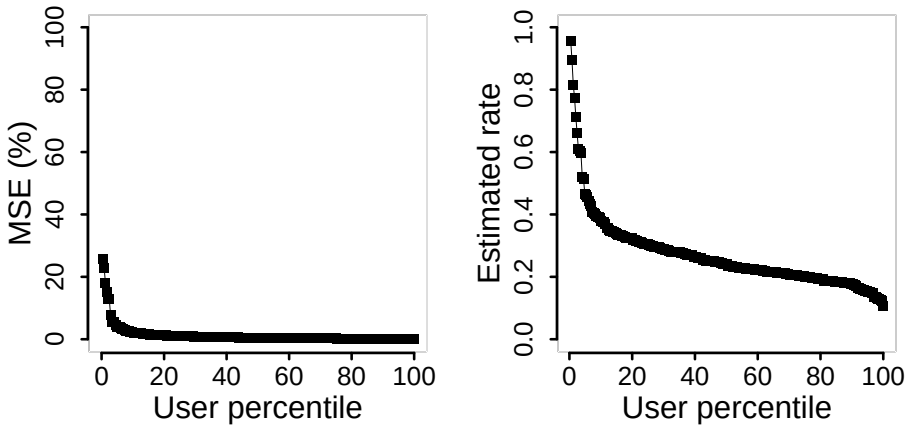
(a) Weibull scale (b) Weibull shape

Figure 2.36: Distribution of inferred model parameters that describe the distribution of time between sessions for users in both datasets. The shape is consistently less than one which indicates that the longer the screen has been off, the less likely it is to be turned on again by the user.

Figure 2.37(b) shows the inferred rate parameter of the application popularity distribution for various users. We see that the rate varies by an order of magnitude, from 0.1 to almost 1. The value of the rate essentially captures the pace of the drop in application popularity. Lower values describe users that use more applications on a regular basis. One implication of the wide range is that it may be feasible to retain all popular applications in memory for some users and not for others.

2.7 Summary

By studying 255 users of two different smartphone platforms, we comprehensively characterized user activities and their impact on network and battery. We quantify many hitherto unknown aspects of smartphone usage. User diversity



(a) MSE (b) Rate parameter

Figure 2.37: (a) The mean square error (MSE) when application popularity distribution is modeled using an exponential. MSE is less than 5% for 95% of the suers which indicates that the exponential drop in application popularity is true for almost all users. (b) The inferred rate parameter of the exponential distribution for different users. The rate varies by an order of magnitude among users.

is an overarching theme in our findings. For instance, different users interact with their phones 10-200 times a day on average; the mean interaction length of different users is 10-250 seconds; and users receive 1-1000 MB of data per day, where 10-90% is received as part of active use.

This extent of user diversity implies that mechanisms that work for the average case may be ineffective for a large fraction of the users. Instead, learning and adapting to user behaviors is likely to be more effective, as demonstrated by our personalized energy drain predictor. We show that qualitative similarities exist among users to facilitate the development of such mechanisms. For instance, the longer the user has not interacted with the phone, the less likely it is for her to start interacting with it; and application popularity for a user follows a simple exponential distribution.

Our study points to ways in which smartphone platforms should be enhanced. Effective adaptation will require future platforms to support light-weight tools that monitor and learn user behaviors *in situ*. It will also require them to expose appropriate knobs to control the behavior of lower-level components (e.g., CPU or radio).

CHAPTER 3

Automating Battery Management

Our user studies revealed that users run a wide range of applications in addition to voice calling on modern smartphones. We also learned that significant diversity in usage habits combined with the diversity in hardware and battery specifications, makes battery life of smartphones unpredictable. If batteries lasted long enough, unpredictability would not have been a serious concern. However, the linear improvements in battery capacities as discussed in Chapter 1 are no match for the rate of new power consuming features and applications that are offered on smartphones. As a result, the average battery life time gets shorter with the introduction of each new generation of smartphones to the market.

Many pervasive computing applications and increasing number of other applications, such as email clients and cloud sync services, have components that continuously run in the background. Such applications are particularly power consuming [1]. Unlike traditional fully interactive applications, users do not have direct control over the resource consumption of background tasks. Therefore, the onus is on developers of such applications to select "optimal" duty cycling configurations. Due to significant diversity among users and variable usage patterns of individuals, as presented in Chapter 2, no single static configuration can be optimal for all users. As a result, there are always some users whom application

[1] Modern smartphones rely on long user idle times to switch the phone hardware off and save battery power. Background processes invalidate this assumption.

developers cannot satisfy when it comes to battery life time.

Personalization can be the solution to these problems. But so far with no systematic way to do it, personalization has been yet another challenge for app developers. Most developers expose one or more parameters to the users to enable them to manually change the power consumption configuration. For example, a mobility tracking application developed at the Center for Embedded Networked Sensing allows users to change GPS and Accelerometer sampling intervals. Even if users understood the relation between these configuration parameters and battery consumption rate, they would rarely change them because it is cumbersome. For instance, in our user studies on different platforms we found that majority of users never change the default screen timeout value of their smartphone.

In this chapter we introduce PowerLeash, a system that gives users control over their phone's battery life time and automates personalization of background applications for developers. PowerLeash monitors the user's interaction with the phone and battery drain. It learns the impact of background applications on battery drain rate. With a simple user interface, PowerLeash receives the user's desired battery life in terms of a single metric that users can understand — battery lifetime goal. With this information PowerLeash dynamically changes the power consumption configuration of background applications to meet the user's battery expectation each day.

3.1 System Design

The design of PowerLeash can be best explained in the context of the typical use case. Consider a smartphone user, Alice, who wants to run an application, Actigraphy, that estimates her mobility and calorie expenditure by continuously

recording GPS and accelerometer information in the background. But she is reluctant to do so because when Actigraphy runs, her battery lasts only 12 hours on average, which means that on many days it will run out before she gets an opportunity to charge her phone in the evening. With existing smartphones, she thus faces the undesirable trade-off between not running Actigraphy or actively managing its power use.

We want to give Alice the following ability: she tells her phone how long she would like the battery to last every day or for all days through her calendar, and with this information the power consumed by Actigraphy and other background applications is automatically managed to meet the deadline, adapting as needed to her use of other (interactive) applications.

If PowerLeash detects that the battery goal is unlikely to be met (e.g., because of abnormally high interactive usage), it sends an early warning to Alice. This warning signal can help her take appropriate actions such as avoiding lower priority usage [RSH08] or planing to charge her phone at a convenient time.

In principle, Actigraphy itself can be written in a way to meet user-defined battery goals. But we advocate a system-wide solution. It is harder for an application to engineer this functionality because it does not have a global view (e.g., other background applications and their future power consumption).

Further, automatic, adaptive control that is enabled by PowerLeash is preferable to static configuration options (e.g., sampling interval) that some applications provide to users today. Users have little understanding how these options translate to power use, and any static configuration can be too aggressive on some days and too conservative on others.

3.1.1 Design principles

The design of PowerLeash is guided by three principles. Here, we introduce these principles and our justifications.

1. Easy to deploy We want to build a system that can be easily deployed. This desire has two implications. First, it requires us to leverage information sources that are already available on commodity smartphones. This will enable us to immediately start gaining field experience from Battery Drain Management (BDM) systems towards refining their design, rather than waiting for future generations of hardware or software that may be able to provide more accurate or granular information.

Second, it requires us to handle the high degree of diversity that exists for devices, users, or background applications. This in turn means that we should not require any offline customization or parametrization for individual devices or users. Any personalization of system operation should occur online and in-situ.

2. Easy to use for application developers The developers of background applications will be more likely to adopt our proposed APIs if they are easy to use. For this reason, PowerLeash expects applications to implement only two simple functions–one that enables it to poll how much work they have done since start and another that enables it to communicate the maximum amount of work they can do in the next interval. Further, the units of work are application-level (e.g., number of times location was polled), rather than absolute level of energy which is harder to use and varies with the platform.

3. Easy to use for users Finally, to have any chance of acceptance, the system should be easy to use for users themselves. For this reason, the only

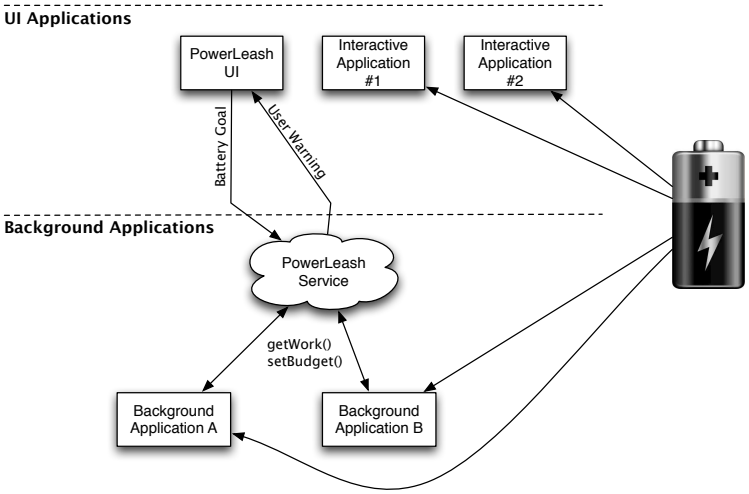


Figure 3.1: General architecture of PowerLeash. PowerLeash consists of a user interface, application interfaces, and a background service.

input we ask of users is to specify a battery lifetime goal. We believe that users are capable of specifying this goal because they know when the next charging opportunity will arise (e.g., at night, after the flight).[2]

3.1.2 Design Overview

Figure 3.1 shows an overview of PowerLeash. PowerLeash consists of a user interface, application interfaces, and a background service.

User interface Users configure their desired battery goal using a simple interface (Figure 3.2). This interface appears automatically when the phone is unplugged from the charger, to remind the user to set the goal. It can also be recalled later to change the goal.

[2]In future work, we will explore the extent to which we can free users from providing even this input by learning their daily patterns.

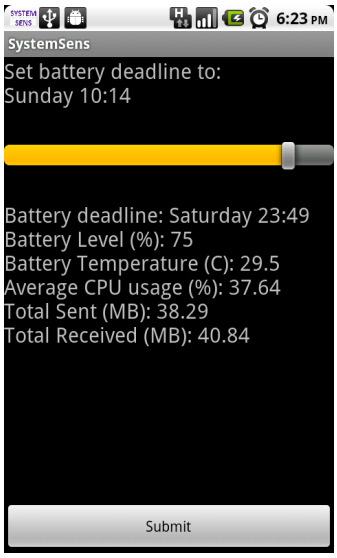


Figure 3.2: User interface of PowerLeash that prompts user to set a battery goal after every charging instance.

Application interface Background applications register with PowerLeash along with a desired planning interval that we refer to as the *horizon*. A short horizon is for applications that can modulate their energy use at short time frames (e.g., by changing the location sampling interval). Longer horizons is for applications with uneven energy usage. For instance, an application that needs to periodically upload data to a server consumes a burst of energy when the data is uploaded. This application is not in a position to promise that it will not consume over x units of energy every minute, but it can promise to conusme less than $30x$ units over 30 minutes.

Background applications implement an interface with two functions. The nature of this interface is inspired by MS Manners [DB00]. The two functions are:

- **getWork()**: This function should return a vector indicating progress or

amount of work done since the application started. The application counts its main energy consuming operations (e.g., location polling, data uploading) and returns the cumulative values in a vector. PowerLeash does not need to know what the individual entries in the vector represent. It only expects that their order is consistent. The counts can be fractional values, e.g., (126.0, 1763.5)

- **setBudget()**: This function is called by PowerLeash to signal the amount of work that the application can perform within its next horizon. It is called with two vectors. The first vectors contains the amount of work of each type and the second contains their relative impact on power consumption. The second vector enables the the application to do more of one type of work and less of another, while keeping power footprint the same if it decides to and is possible.

Background service PowerLeash constantly monitors resource usage (e.g., CPU, screen), the work done by background applications along with the battery level. This information is used to build and update a model that estimates the impact of resource usage and background work on power consumption. It is also used to learn statistical measures of resources consumed by interactive usage. Based on the power model and this measure, PowerLeash periodically decides how much energy can be allocated to background applications such that the battery will likely not run out before the specified deadline. It then divides this energy among individual background applications. An application's budget is communicated to it, after translating it to its units of works using the power model.

Battery level	[1..100]
Time the screen is ON	seconds
CPU utilization	[1..100]
Memory usage	bytes
Cellular data usage	bytes
WiFi data usage	bytes
Voice call duration	seconds
# of disconnections	count
Signal strength	[1..5]
Work by background apps	app-specific

Table 3.1: Quantities monitored by PowerLeash to build the power profile and their units.

3.2 Power Consumption Model

PowerLeash learns the power profile of the smartphone while it is being used, therefore it does not require any off-line power profiling or any external measurement setup. We use the power profile to compute the impact of *a)* user interaction with the phone, and *b)* work done by background applications on battery drain.

We use a modified version of SystemSens (introduced in Appendix A) to track resource consumption and other events of interest that have an impact on power (e.g., network disconnections). Table 3.1 lists the quantities that we currently track. Each of these are easy to monitor on Android phones with minimal impact on phone battery as discussed in Chapter A. Some of these quantities are event-driven (e.g., disconnections); those that need polling (e.g., CPU usage) are polled every two minutes. When the phone is plugged to the charger, data since the last upload is uploaded to the PowerLeash server.

To build the power profile, we assume that power consumption has a linear relationship with battery level and a linear, additive relationship with resources used. In other words, the change in battery level in a given time period is:

$$\Delta BatteryLevel = \beta_0 + \Sigma_i(\beta_i \times r_i) + \Sigma_{j \in BgApps}\Sigma_k(\alpha_{jk} \times w_{jk}) \qquad (3.1)$$

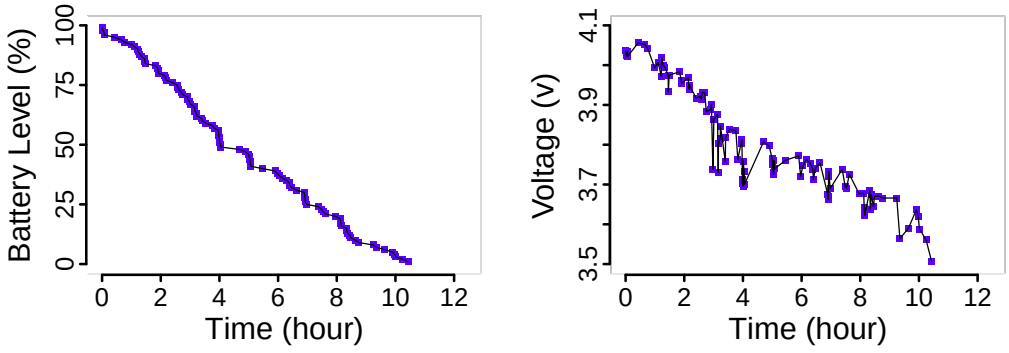
where r_i represents the phone-level quantities (the first group of quantities in Table 3.1), $BgApps$ is the set of background applications, and w_{jk} is the k-th element of the work vector reported by Application j. β_i and α_{jk} represent the impact on power of their corresponding factor and the model parameters that we need to learn. β_0 represents baseline power consumption. While accounting for phone resources, we subtract what is used by background applications, that is, r_{CPU} is measured total CPU usage minus what is used by background applications. (Android provides resource usage information for individual applications.) Resource use of background applications is accounted for as part of their work.

To learn model parameters, we construct one such row for each 10-minute interval (we will refer to the resulting matrix as *usage matrix*) and then run robust regression [RLW87] on the usage matrix.

We run the model building process at the server once per day and separately for each user. That is, the model is personalized to the user and is updated daily. Each time a maximum of the last two weeks of data is used for the user. New models are communicated to the phone when it connects to the server next.

3.2.1 Discussion

The model as built above is low-fidelity due to several reasons. First, battery level is a coarse measure of power consumption. Figure 3.3(a) shows the sampled battery level of an example user. We can see that it varies unevenly, instead of



(a) (b)

Figure 3.3: Sampled battery level (a) and voltage (b) for an example user. Battery information on Android is broadcast event based, therefore the sampling interval varies. Both battery level and voltage vary unevenly, but battery voltage is much more noisy. Therefore we decided to use battery level.

a smooth monotonic decrease. Such behavior is part of the reason we use a 10-minute interval to build the power profile, so that local effects can be smoothed out. Using current instead of battery level would permit more accurate estimate of power draw [DZ11b] but only a few commodity phones provide that information today. We consider using battery voltage as well (which like battery level is easily obtained) but found it to be even more noisy than battery level due to recovery and rate effects [RVR03]. Figure 3.3(b) shows an example.

Second, the measures we use for resource consumption correspond only approximately to power consumption. For instance, because of DVFS (dynamic voltage and frequency scaling) [PS01] the power consumed by CPU depends on its operating mode in addition to how much it was used. Similarly, power consumed by the radio depends not only on the total amount of data transferred but also on the sizes of individual transfers [BBV09]. We work instead with approximate quantities because they can be obtained easily, universally and with

minimal overhead.

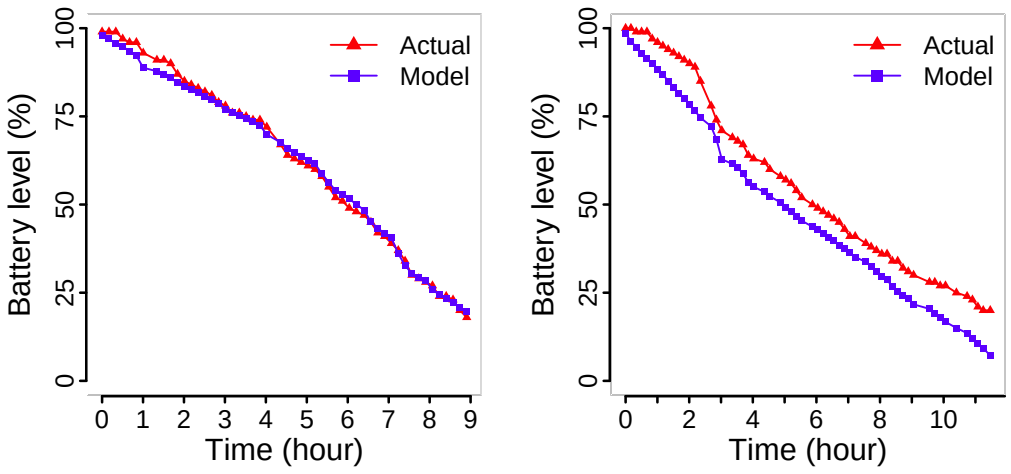
Finally, recent work shows that total power consumed may not be a linear, additive function of power consumed by individual resources [PHZ11]. We note that prior works on power modeling [DZ11b], even those that build high-fidelity models, make this linearity assumption as well because linear models are simpler to build and use. Despite these sources of errors, we find that our model is good enough for our purposes of managing background applications. In that sense, what we have a wrong but useful model [Geo79].

3.2.2 Evaluation

In this section we use data from 20 users that were running PowerLeash without any adaptation for 5-8 weeks (see Section 2.1.1 for more details) to evaluate the performance of our modeling.

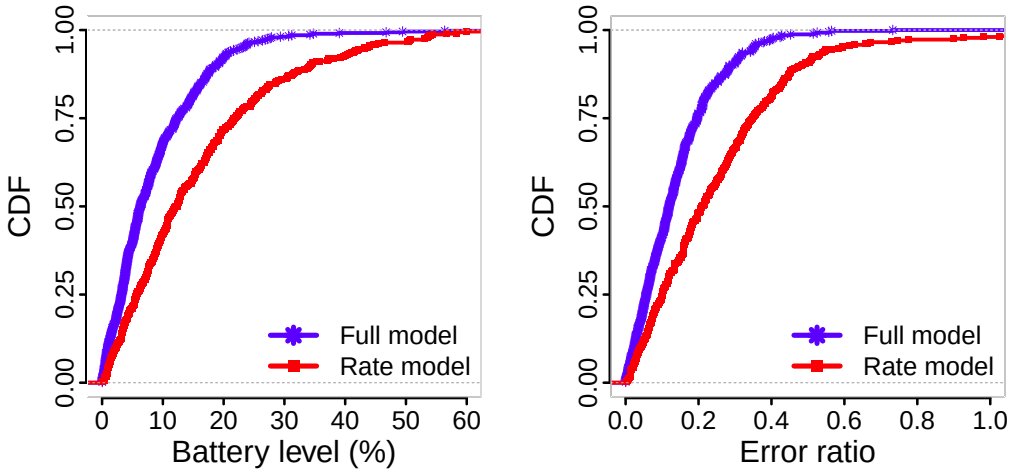
Figure 3.4 shows actual and predicted battery level for two example discharge cycles. A discharge cycle begins when the phone is disconnected from a charging source and ends when it is charged again or runs out of battery. Lengths of discharge cycles vary, and they do not necessarily start when the battery is completely charged. Figure 3.4(a) is a case where the model very accurately predicts changes in battery level based on resource consumption. In contrast, the model fails to accurately predict battery level changes in Figure 3.4(b).

Figure 3.5 quantifies the error of our model when used to predict battery level change across a complete discharge cycle. Figure 3.5(a) shows the CDF of absolute error in predicting battery level at the end of a discharge cycle. It plots the absolute difference between the predicted and actual levels. Figure 3.5(b) shows the CDF of relative error ratio, that is, absolute error divided by the actual level. We see that the error, while noticeable, is low. The median of



(a) (b)

Figure 3.4: Actual and predicted battery level for two example discharge cycles of a user. (a) A case where the model very accurately predicts changes in battery level based on resource consumption. (b) Example of a case where the model fails to accurately predict battery level.



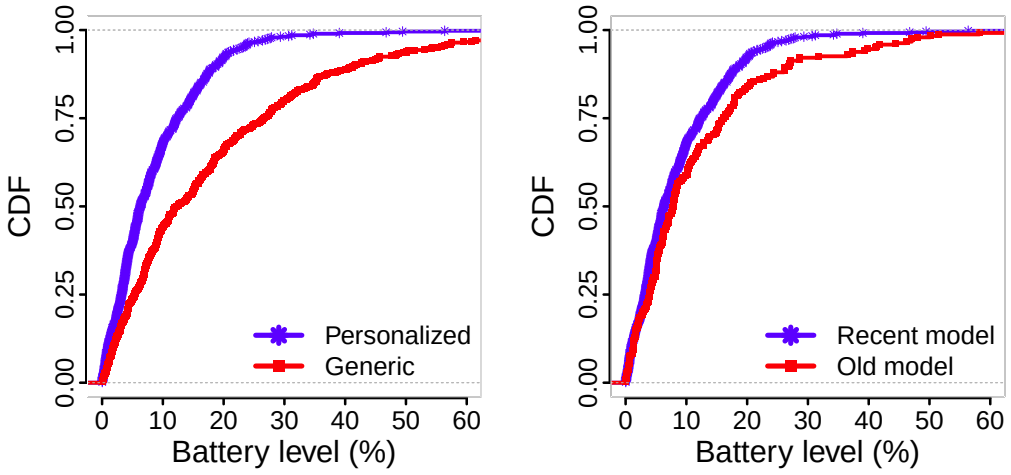
(a) (b)

Figure 3.5: Absolute and relative error of two models when predicting battery level at the end of a discharge cycle. The error, while noticeable, is low. The median of relative error is 0.11 and the mean is 0.14.

relative error is 0.11 and the mean is 0.14. Though not shown in the figure, we find that the model is unbiased, that under prediction is just as likely as over prediction. Predictions for a given day are made based on the model built the previous day using at most two weeks of data.

To put the accuracy of our model in context, we compare it against another low-fidelity model built using easily obtained information. Oliver *et al.* use drain rate for each user and clusters of similar users to predict battery level [OK11]. Figure 3.5 shows that the error of this model is higher. The median of relative error is 0.22 and the mean is 0.30— about twice that of our model. The higher accuracy of our model stems from separating out the power consumed by individual resources.

We also find that personalizing of the model to each user and updating the model every day helps increase accuracy. Technically, the model should not

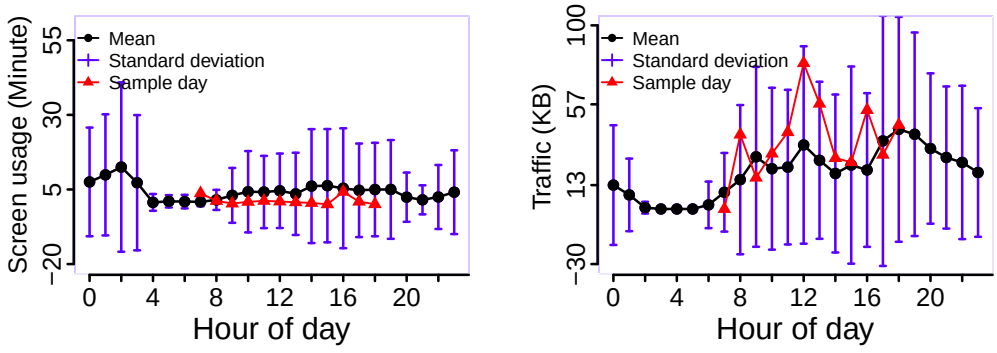


(a) (b)

Figure 3.6: Absolute error of a (a) generic battery model and (b) old battery model compared to personalized recent models for each user when predicting battery level at the end of a discharge cycle. We conclude that model accuracy is lower when an old or stale model is used or when the model is not personalized.

change with time and should be identical for users that use the same smartphone. (High-fidelity models assume that this is the case.) But this does not hold for our low-fidelity model. Figure 3.6 shows that the model accuracy is lower when an old model is used to predict the battery level for the current day or when the model is built using information from all users.

We conclude that personalization and recency help combat some of error sources mentioned above due to low-fidelity information. Our results also suggest that building a universal (across users and time) model with low fidelity information is a more challenging task than building a personalized model that is updated periodically. Dong *et al.* [DZ11b] also pointed out the value of personalization.



(a) (b)

Figure 3.7: Mean and standard deviation and sample of one day of screen and traffic usage for an example user as a function of time of day. Variations in mean values are much smaller than the standard deviation across each mean. Throughout the day each parameter is on average statistically similar, that is, the error bars are overlapping.

3.3 Estimating Interactive Usage

To accurately estimate energy that can be safely assigned to adaptive background applications, PowerLeash needs to estimate expected future interactive usage. This estimation is challenging because we find that interactive usage is highly variable. But we show that reasonable accuracy can be achieved by basing the estimate on recent history.

3.3.1 Highly Variable Usage

By analyzing interactive usage of data collected from real users, we found that the usage of every resource (e.g., CPU, screen, radio) is highly variable. For an example user, Figure 3.7 shows for every hour of the day the mean and standard deviation of two interactive usage parameters. Figure 3.7(a) shows mean and

standard deviation of the duration for which the screen was on, and Figure 3.7(b) is the same graph for cellular network traffic. These graphs are based on 83 days worth of data. The graphs show that throughout the day each parameter is on average statistically similar (that is, the error bars are overlapping).

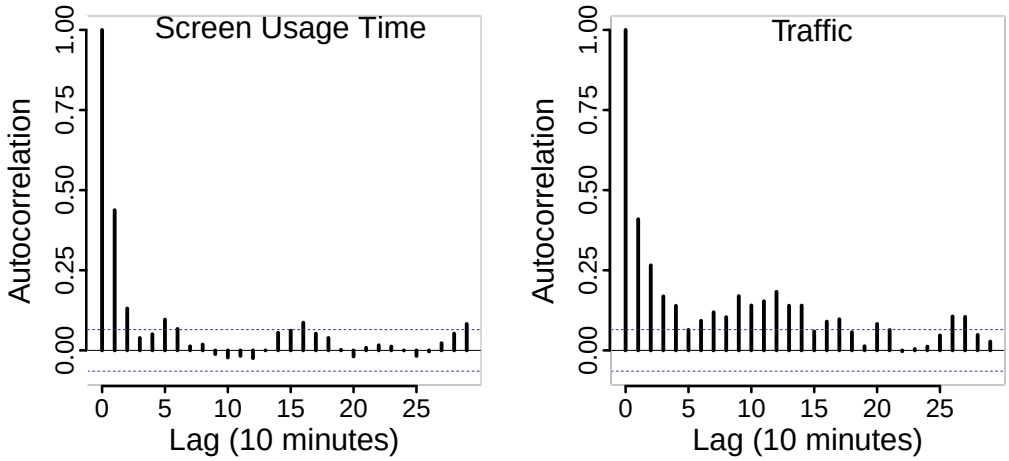
More importantly, we see that during the day time when the phone is being discharged (and controlling background applications is most needed), the variance both parameters are significant, roughly 2-4 times their mean. This property holds for other resources as well. Because of this high variance, we find that estimates of interactive usage based on previous days' data have poor accuracy.

Our analysis also revealed that the variance of interactive usage *within* a day (as opposed to across multiple days) is much lower. Figure 3.7 also includes the resource usage within a single discharge cycle. We see that the variance is much lower. If a user is having a low-usage day, she is likely to continue to have a low-usage day. On the other hand, when her usage is high, it is likely to continue above average.

3.3.2 Short-term Memory in Usage

We find that every usage parameter exhibits significant autocorrelation in relatively short time lags. For example, the total time that a user keeps the screen on during a 10 minute interval is strongly correlated with that parameter during the previous 10 minutes. Figure 3.8 shows the autocorrelation coefficient of two example parameters during 10 minute intervals at different time lags for a sample user. Ignoring $Lag = 0$, that corresponds to self correlation, $Lag = 1$ shows the highest correlation.

These observations imply that smartphone interaction parameters have short-term memory. To test this hypothesis we use the *k-means* algorithm [Mac67]



(a) (b)

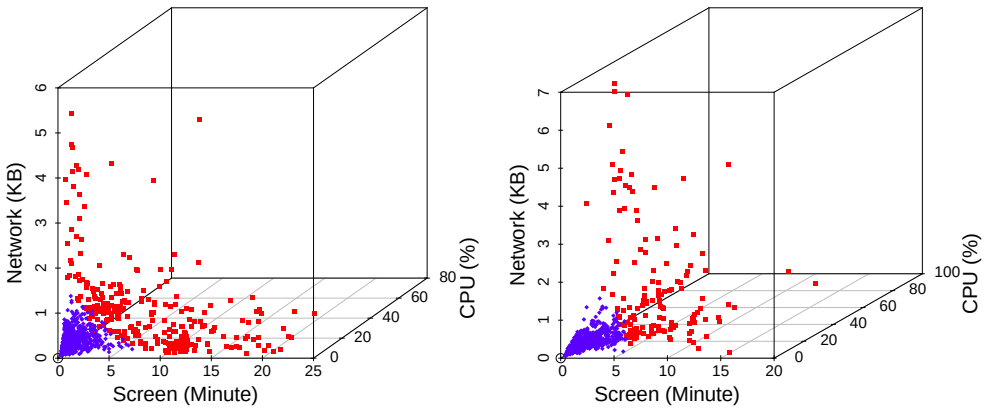
Figure 3.8: Autocorrelation of screen time (a) and cellular traffic (b) across 10 minute intervals at different time lags. There is significant correlation at lag = 1 implying that smartphone interaction parameters have short-term memory.

to partition the usage matrix to two clusters. Considering the highly skewed distribution of usage parameters as presented in Chapter 2 we expect one cluster to represent inactive times and the other cluster active intervals. We later explore partitioning the usage matrix to more than two clusters.

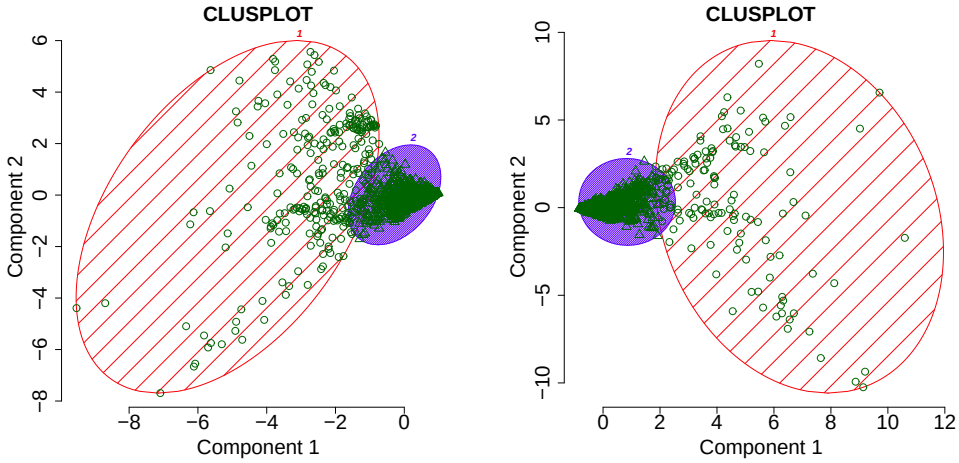
Figure 3.9(a) shows a 3D scatter plot of the usage matrix values when considering screen usage, CPU usage and network usage [3]. Similarly Figure 3.9(b) shows a bivariate CLUSPLOT of the usage matrix based on the two principal components for the same two users. We can see that the shapes of active and inactive clusters are different for these two users, but in both cases they are distinct from each other. We observed the same pattern on all other users.

Assuming that transitions between these two states are based on a first or-

[3]We find these three parameters to be the most important usage parameters both in terms of their coefficients in the power model and in terms of their impact on clustering



(a)



(b)

	inactive	active
inactive	0.89	0.11
active	0.65	0.35

	inactive	active
inactive	0.96	0.04
active	0.62	0.38

(c)

Figure 3.9: (a) 3-dimensional scatter plots of screen, CPU and network usage for two example users. Blue points belong to the inactive cluster and red points belong to the active cluster. (b) bivariate clusplot of usage matrix. The cluster labeled as number 2 represents inactive times. (c) transition probabilities assuming a first-order Markov chain. We see that both of these example users are very likely to stay in the inactive state. This observation matches with the skewed distribution of usage parameters.

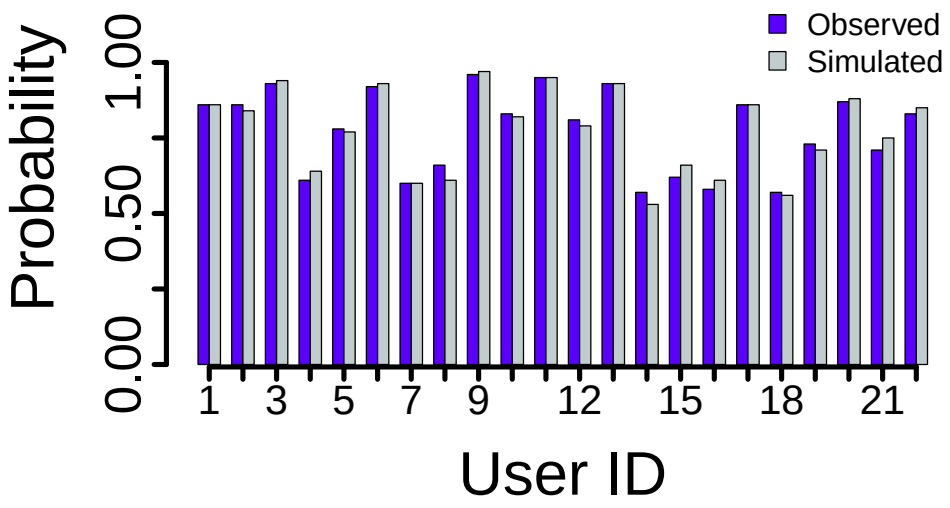


Figure 3.10: Probability of being in inactive state computed from real traces and simulated state sequence based on a first order Markov Chain. The close match between simulated and observed marginal probabilities for all users confirms that smartphone interactive usage behaves based on a first-order Makrov Chain with two states.

der Markov Chain model, meaning that the probability of being at each state only depends on one past state, Figure 3.9(c) shows the transition matrix of such a hypothetical Makrov Chain for the two example users. We see that both users, are very likely to stay in the inactive state which matches with the skewed distributions of usage parameters.

To verify the Markov Chain process assumption, we build the transition matrix for all the users and use it to simulate a large sequence of states. We then compare the marginal probability of being in the inactive (or active) state using both observed and simulated data. Figure 3.10 compares these two values for 22

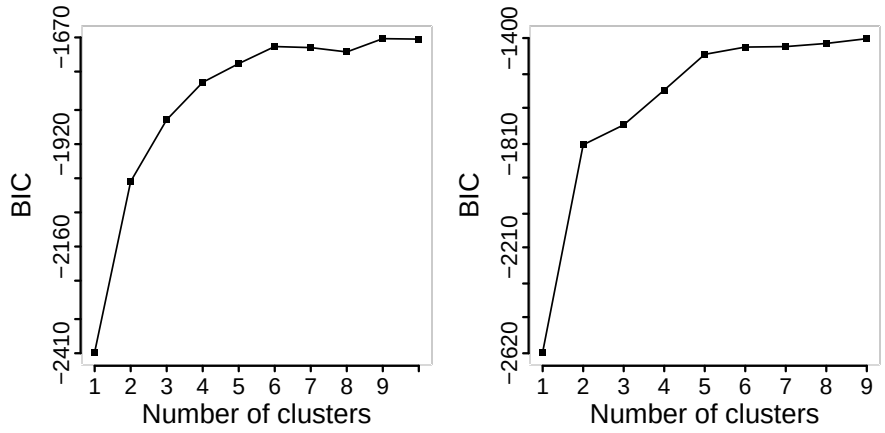


Figure 3.11: BIC of different number of clusters for two sample users. We find that for all other users, similar to these two example users, BIC increases with the number of clusters and the rate of its growth is much faster for 1-5 clusters.

users from a real user study. The close match between simulated and observed probabilities for all users confirm that smartphone interactive usage behaves like a first order Markov Chain consisting of two states.

Optimal number of clusters One method to find the optimal number of clusters in unsupervised learning is using the Bayesian Information Criteria (BIC) [PB04]. When clustering, if BIC is maximized at any specific number of clusters that number can be considered to be the optimal model. We evaluated BIC for models consisting of 1 to 10 different clusters for all the users. Figure 3.11 shows the BIC graph for two sample users. We find that for all users, BIC increases with number of clusters. The rate of its growth is much faster for 1-5 clusters.

A different method to identify the optimal number of clusters is considering within-groups sum of squares (WSS). WSS monotonically decreases as the number clusters increase but if there is an optimal number of clusters, the rate of decrease of WSS flattens. Statistical "folklore" has it that the location of such a

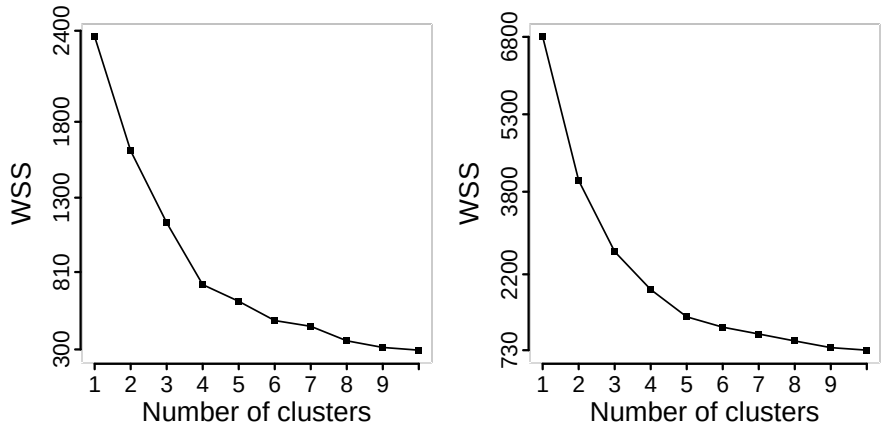


Figure 3.12: WSS of different number of clusters for the two sample users. WSS monotonically decreases as the number of clusters increases but if there is an optimal number of clusters, the rate of WSS decrease would flatten beyond that point.

"elbow." Figure 3.12 shows the values of WSS at different number of clusters for the same two sample users.

Tibshirani *et al.*[TWH01] formalized this heuristic for identifying the optimal number of clusters by introducing the GAP statistic. This method standardizes the graph of WSS (in logarithm scale) by comparing it with its expectation under an null reference distribution. We also evaluated the gap statistic [TWH01] on all the users. Figure 3.13 shows the gap statistic for the two sample users. For these two users $K = 2$ maximizes the GAP statistic. We come to the same conclusion when looking at the GAP statistic for most other users. This finding matches our anecdotal understanding of smartphone usage.

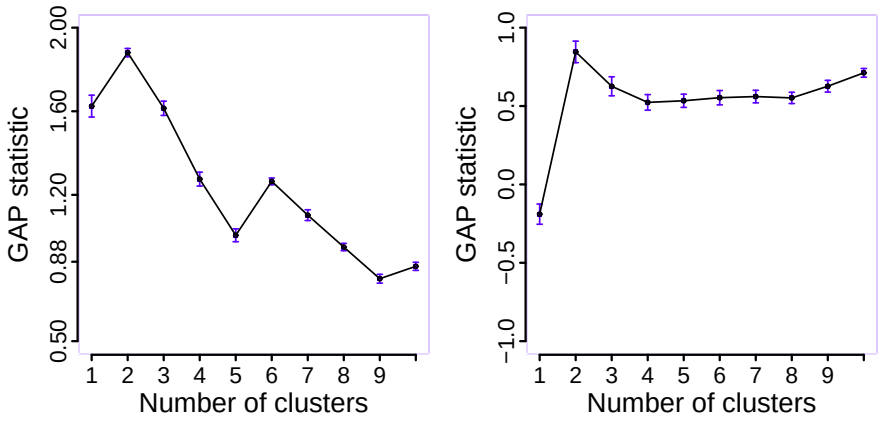


Figure 3.13: The GAP statistic for the two sample users. For these two users $K = 2$ maximizes the GAP statistic.

3.3.3 Estimation Algorithm

Building on this insight, we can estimate new future interactive usage based on recent past usage. Figure 3.14 shows the CDF of error when estimating traffic and screen interaction time for all users. Recent mean is an estimation based on the method above, and past mean uses the mean of the parameter at the same time of day during past days. Table 3.2 compares the mean absolute error for these two algorithms.

We use the recent estimator to estimate the usage of each resource. To estimate the total energy needed for interactive usage, we aggregate results across resources using the coefficients of the power model.

3.4 Adaptation Policy

In this section we describe how PowerLeash allocates energy to adaptive background applications such that the battery lifetime goal is met and interactive

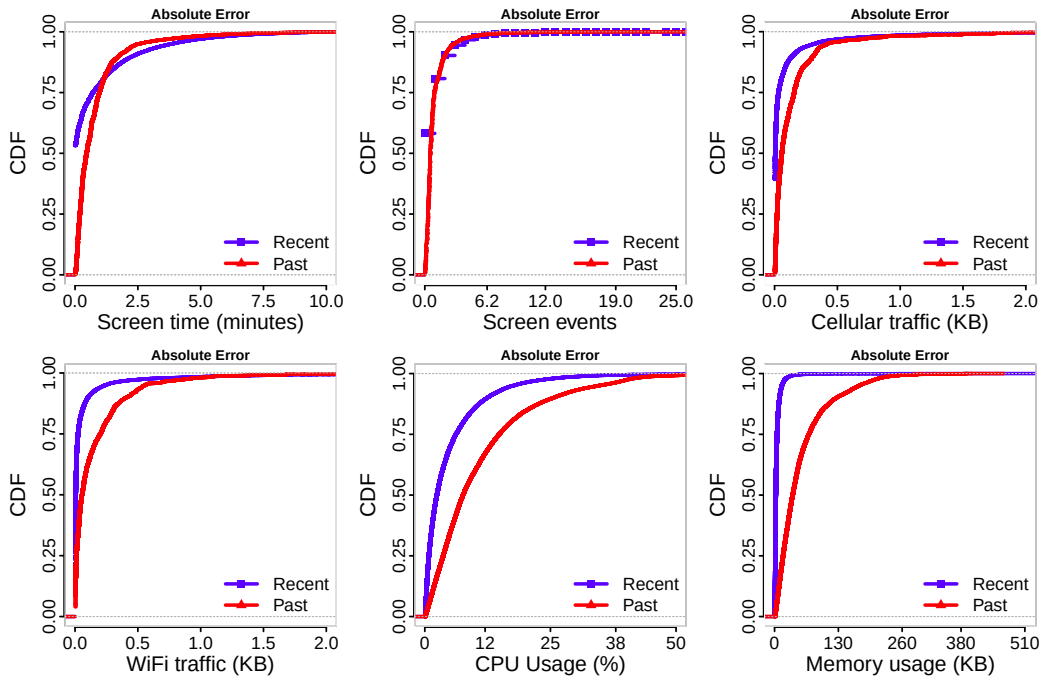


Figure 3.14: CDF of the error when estimating usage parameters for next 10 minutes using past and recent usage for all users. For all usage parameters recent mean is a better estimator of future compared to passed mean. The difference in error of these two approaches is more significant for traffic, CPU and memory usage.

	Recent	Past
Screen	0.69 Min	0.78 Min
# Events	0.87	0.9
Cell. Traffic	0.08 KB	0.13 KB
WiFi Traffic	0.07 KB	0.17 KB
CPU	4.7%	11%
Memory	3.9 KB	52.7 KB
# Disconnections	0.84	0.98
Call	5.57 Sec	5.67 Sec

Table 3.2: Mean of absolute error of estimating usage parameters for next 10 minutes for Recent and Past algorithms.

usage is not curtailed. To motivate our eventual policy, we first describe two policies that represent extremes with respect to how energy to background applications can be controlled. We call these policies ShortLeash and LongLeash.

3.4.1 ShortLeash Policy

As the name suggests, the ShortLeash policy maintains a tight control over the energy used by background applications. We assume that the desirable battery drain rate from the beginning of the discharge cycle to the end (lifetime goal) is linear. This policy can be easily modified to follow a different drain rate curves if needed.

Given the desired drain rate curve, we can compute if the current battery level is higher or lower than what is desirable. If it is higher, we can increase the energy allocation for background applications. Otherwise, we should decrease it.

$\delta \leftarrow IdealLevel - Level$
$\sigma \leftarrow \sum_{ji \in BgApps} \alpha_{jk}$
for $j \in BgApps$ **do**
  for $k \in BgApp_j$ **do**
    if $\delta > 1$ **then**
      $t_{jk} \leftarrow \frac{w_{jk}(i-1)}{\delta}$
    else if $\delta < -1$ **then**
      $\gamma \leftarrow \frac{\alpha_{jk}}{\sigma}$
      $t_{jk} \leftarrow w_{jk}(i-1) + \gamma$
    else
      $t_{jk} \leftarrow w_{jk}(i-1)$
    end if
    $b_{jk}(i) \leftarrow 2 \times \left(t_{jk} \times horizon_j\right) - B_{jk}$
  end for
end for

Figure 3.15: Pseudocode of ShortLeash policy. ShortLeash maintains a tight control over the energy used by background applications. To avoid rapid changes in assigned budgets ShortLeash uses additive increases and multiplicative decrease (AIMD).

This policy is not identical to but is inspired by Odyssey [FS99]. Figure 3.15 shows the pseudocode for the ShortLeash policy. It uses additive increase and multiplicative decrease (AIMD), which leads to a slow increase in budget but a rapid decrease along with quickly reaching fairness [Jac88]. The control algorithm in Figure 3.15 is run every two minutes, and new allocations are computed and communicated to the background applications.

In all the algorithms in this section B_{jk} is the total amount kth element of reported work vector by the jth adaptive application during the past horizon interval and b_{jk} is the amount of budget for the kth resource of the jth application within its specified horizon, $horizon_j$. Assigned budgets have memory within one past horizon, meaning that if an application does not consume all its assigned budget, its budget within the next horizon will include the unused budget and vise versa.

3.4.2 LongLeash Policy

In this policy, applications are given a lot more freedom. Their budget is decided not on the basis of the current battery level alone but based on what they can safely consume until the battery deadline. Given the current battery level and an estimate of how much energy will be consumed through interactive usage until the deadline, the LongLeash policy computes how much energy can be safely allocated for background work. Thus, unlike the ShortLeash policy that allocates energy with the aim of aligning the battery level to the desirable drain rate curve at all times, it simply aims to meet the deadline from this point onwards.

Figure 3.16 shows the pseudocode for the LongLeash policy. Periodically, it evaluates the total budget available to all adaptive applications by subtracting the expected power needed by interactive usage from available battery capacity.

$Left \leftarrow Deadline - CurrentTime$
$Interactive \leftarrow \sum_{i \in Resources} E_i(Deadline) \times \beta_i$
$Available \leftarrow Level - Interactive$
for $j \in BgApps$ **do**
  for $k \in BgApp_j$ **do**
    if $Available < 0$ **then**
      $b_{jk}(i) \leftarrow 0$
    else
      $\rho_{jk} \leftarrow \frac{Available}{\alpha_{jk} \times Left}$
      $b_{jk}(i) \leftarrow 2 \times \rho_{jk} \times horizon_j - B_{jk}$
    end if
  end for
end for

Figure 3.16: Pseudocode for LongLeash policy. LongLeash gives background applications significant freedom because the budget is not decided based on current level alone.

In this algorithm $E_i(t)$ gives the estimated usage of the ith resource between now and a future time t.

3.4.3 PowerLeash Policy

While we present detailed empirical results in the next section, the nature of interactive usage limits the effectiveness of the ShortLeash and LongLeash policies. Because interactive workload is highly bursty, at any given time battery level can be significantly higher or lower than what is desirable. The ShortLeash policy reacts to these short-term variations, as a result of which assigned application budgets become highly variable, which creates a hostile environment for the applications. On the other hand, interactive workloads are highly unpredictable as well. This property means that the LongLeash policy can significantly underestimate or overestimate interactive workloads, which can lead to battery underflows

$window \leftarrow CurrentTime + W$
$Interactive \leftarrow \sum_{i \in Resources} E_i(window) \times \beta_i$
$Available \leftarrow Level - IL(window) - Interactive$
for $j \in BgApps$ **do**
  for $k \in BgApp_j$ **do**
    if $Available < 0$ **then**
      $b_{jk}(i) \leftarrow 0$
    else
      $\rho_{jk} \leftarrow \frac{Available}{\alpha_{jk} \times W}$
      $b_{jk}(i) \leftarrow 2 \times \rho_{jk} \times horizon_j - B_{jk}$
    end if
  end for
end for

Figure 3.17: Pseudocode for PowerLeash policy. PowerLeash can be controlled by adjusting the size of planning window (W) to emulate ShortLeash or PowerLeash.

and overflows.

We wanted a policy that combines the stable control of LongLeash and the accuracy of ShortLeash. Towards that goal, we use a hybrid policy in PowerLeash. In this policy, we use a planning window of size W and aim to align the battery level to the desirable curve at the end of that window. ShortLeash and LongLeash can be seen as special cases of this policy, with W=0 for ShortLeash and W=time until the battery goal for LongLeash. In our experiments with PowerLeash, we use W=30 minutes. It builds on the insight from the previous section that the interactive workload for the near future can be estimated with some accuracy.

Figure 3.17 shows the PowerLeash policy. In this algorithm $IL(t)$ gives the ideal battery level in given future time, t and $E_i(t)$ gives the estimated usage of the ith resource between now and a given future time, t.

3.5 Simulation

To guide the design of our system and evaluate its performance we conducted two separate user studies. We used the traces from the first study to compare the performance of different policies in simulation. We then used our findings to implement and deploy PowerLeash in the field to evalute its performance on real users. In this section we present the simulation results.

During the summer of 2011 we deployed SystemSens on 20 users along with a simple location tracking application, called LocationTracker. This application turned on the GPS every minute and recorded the current location of the user. The users were given Samsung Galaxy S smartphones with unlimitted data minutes and text plans. The deployment lasted between 5 and 8 weeks. We used this data as input to a trace-based simulation of the PowerLeash system to compare the performance of the policies introduced in Section 3.4.

3.5.1 Simulation Procedure

Table 3.3 lists the parameters used in our simulation and the corresponding definition. For each user the simulator builds the battery model using the algorithm in Section 3.2. Using this model it calculates the battery level at the end of each discharge cycle if the adaptive application did not run (L_b). Battery goal for simulation (L_g) is defined as $L_g = \frac{L_b + L_a}{2}$.

The simulator replays the traces in each discharge cycle and uses an adaptation policy to assign budget to the background application. At the end of each cycle battery *deficit* is calculated as $L_g - L_s$. A positive deficit indicates that the system did not meet the battery goal and a negative value for deficit indicates that the algorithm was too conservative in assigning budget to the adaptive

Parameter	Definition
L_a	Battery level at the end of discharge cycle
L_b	Calcualted battery level with no background app
L_g	Battery goal for simulation ($\frac{L_a+L_b}{2}$)
L_s	Simulated battery level at the end of discharge cycle
$deficit$	$L_g - L_s$

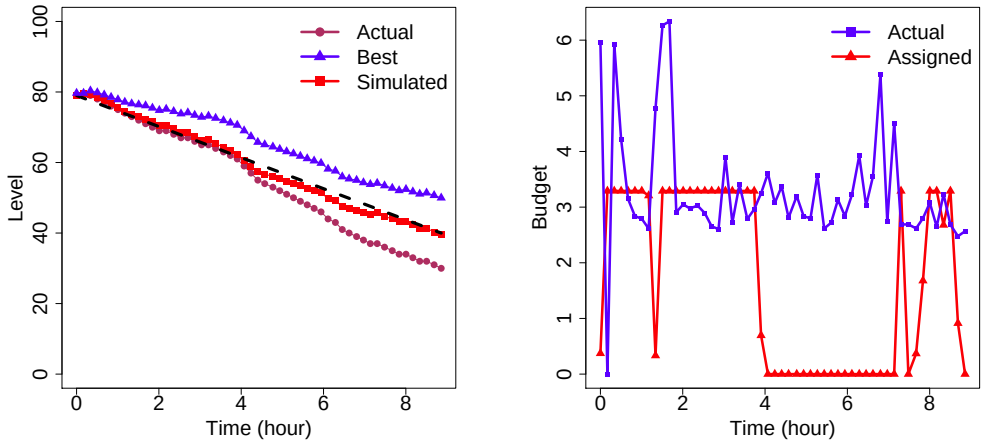
Table 3.3: Definition of simulation parameters. Each parameter is computed at the end of a discharge cycle. We used deficit as the performance metric of the simulation.

background application.

Figure 3.18(a) shows the actual and simulated battery level during a sample discharge cycle. The figure also includes the best possible battery level, if the background application did not run. Figure 3.18(b) shows the actual background work and the budget assigned by the adaptation policy for the same discharge cycle.

3.5.2 Comparing Policies

We designed the trace-based simulation procedure to compare the three policies of Section 3.4 Figure 3.19 is CDF of battery deficit of ShortLeash, with LongLeash and PowerLeash. It also includes an Oracle policy. The Oracle policy is the LongLeash, but it knows future battery drain. From this plot we can see that The deficit of ShortLeash and PowerLeash policies are distributed between -5% and +5% with with PowerLeash being slightly closer to Oracle. The deficit of LongLeash policy is not as good as the other two and LongLeash misses the deadline in 75% of the cases. This result confirms that PowerLeash, as designed,



(a) Battery Level

(b) Assigned Budget

Figure 3.18: (a) Actual, best possible, and simulated battery level during a sample discharge cycle. In this example the simulated battery level ends the discharge cycle very close to the battery goal. (b) Actual and assigned budget for the same discharge cycle. PowerLeash is capable of meeting the battery goal by effectively turning off the background application between 4 and 7 hours into the experiment.

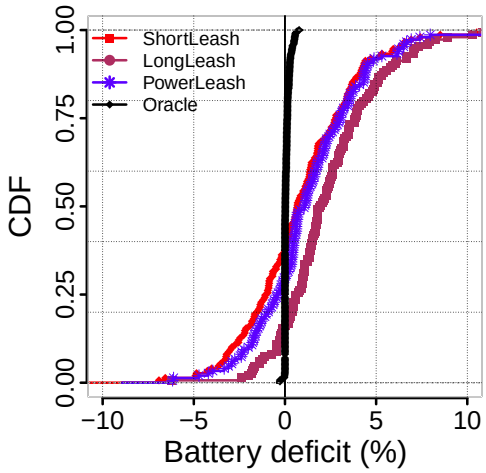


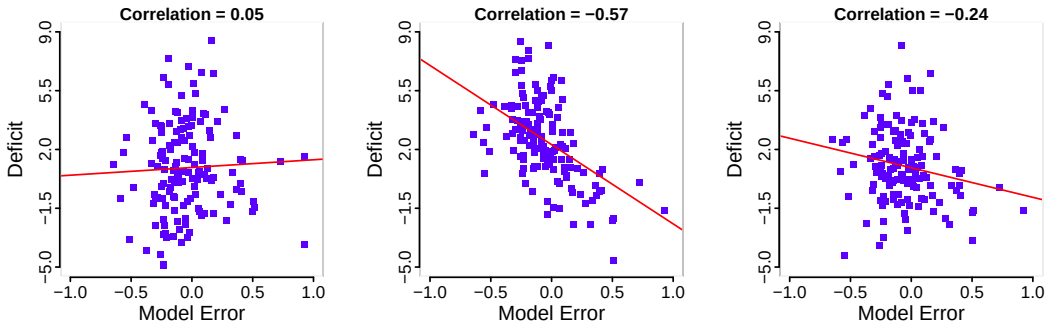
Figure 3.19: CDF of battery deficit of ShortLeash, LongLeash, PowerLeash and an Oracle policy that knows the future. The deficit of ShortLeash and PowerLeash policies are distributed between -5% and +5% with PowerLeash being slightly closer to Oracle. The performance of LongLeash is not as good as the other too and it misses the battery goal more often.

performs better than both LongLeash and ShortLeash.

To investigate why PowerLeash performs better we look at the sensitivity of each policy to sources or error. We identified three contributers to battery deficit.

Model Error Both LongLeash and PowerLeash use the battery drain model to convert expected future interactive usage to changes in battery level. They also rely on the model to translate available battery capacity to background applications' budget. Therefore, we expect correlation between model error and deficit for these two policies. The ShortLeash policy, on the other hand, does not use the model in either stages. Therefore, we do not expect the battery deficit of the ShortLeash policy to have any correlation with model error.

Figure 3.20 shows the scatter plots of battery deficit vs. model error (actual



(a) ShortLeash (b) LongLeash (c) PowerLeash

Figure 3.20: Scatter plot of battery deficit vs. model error during each discharge cycle for ShortLeash, LongLeash, and PowerLeash policies and the least square fit. ShortLeash battery deficit is not significantly correlated with model error. LongLeash deficit is strongly correlated and PowerLeash stands in between.

- estimated) for the three policies. Each plot also includes the least square fitted line and the correlation coefficient between battery deficit and model error. As we expected ShortLeash battery deficit is not significantly correlated with model error. LongLeash battery deficit is strongly correlated with model error. PowerLeash stands in between and is not as strongly affected by model error as LongLeash.

Non-Linear Discharge The natural question to ask is that if ShrotLeash is not affected by model error, what is the source of its battery deficit? The answer lies in the core assumption behind the design of the ShortLeash policy. ShortLeash assumes that the battery discharge rate is linear. We hypothesize that when battery level values during a discharge cycle deviate from a straight line ShortLeash cannot perform well.

To test our hypothesis we define a measure of nonlinear battery discharge for each discharge cycle as follows:

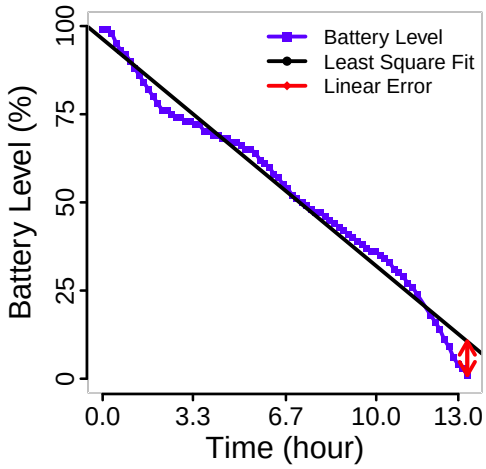


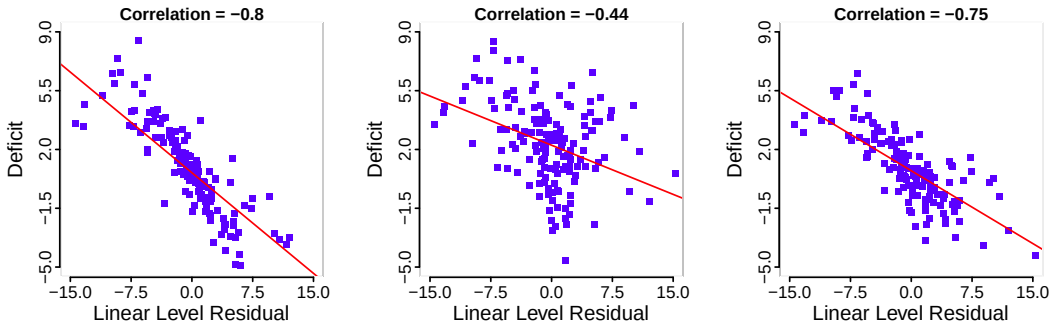
Figure 3.21: Battery levels and Least square fitted line during an example discharge cycle. The red arrow shows the Linear Error during this cycle.

Definition Linear Error is defined as the last residual of the least square line fitted to battery level values during each discharge cycle.

Figure 3.21 shows the Linear Error for a sample discharge cycle. In this example, battery level values deviate from a straight line, therefore the Linear Error is relatively large. The advantage of defining the Linear Error the way we did is that it can distinguish between cases where battery levels are below or above the straight line.

Figure 3.22 shows the scatter plots of battery deficit vs. Linear Error for the three policies. Each plot also includes the least squared fitted line and the correlation coefficient between battery deficit and Linear Error. We see significant correlations for ShortLeash and PowerLeash. The correlation coefficient is not as significant for LongLeash, but it is not negligible.

We see that LongLeash is not significantly affected by usage estimation error. We hypothesis that the PowerLeash deficit, even when future usage is known



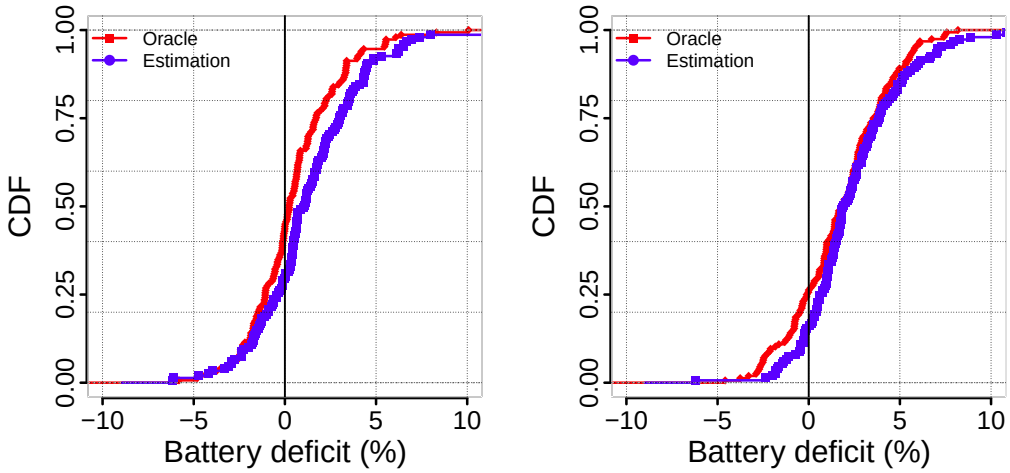
(a) ShortLeash (b) LongLeash (c) PowerLeash

Figure 3.22: Scatter plot of battery deficit vs. linear error during each discharge cycle for ShortLeash, LongLeash, and PowerLeash policies and least square fitted line. Battery deficit of ShortLeash and PowerLeash have significant correlation with linear error.

is so high that the additional error from usage estimation does not make a big difference. On the other hand, PowerLeash performs better when it knows future usage.

Usage Estimation Both PowerLeash and LongLeash estimate future interactive usage based on recent past usage. LongLeash needs to estimate usage for long intervals into the future, but PowerLeash relies on short-term estimations. To evaluate the impact of usage estimation on these two algorithms, we run the simulations again but instead of estimating usage, read future values from the traces.

To evaluate the impact of error of future usage estimation on PowerLeash we run the simulation and instead of estimating future usage, we read future values from traces. We call this the Oracle version of PowerLeash. Figure 3.23(a) shows the CDF of battery deficit for PowerLeash and Oracle PowerLeash. We repeat this experiment with LongLeash and Figure 3.23(b) compares the CDF of battery



(a) PowerLeash

(b) LongLeash

Figure 3.23: CDF of battery deficit of PowerLeash and LongLeash when compared to the Oracle versions of those policies. The Oracle algorithms know the future. The slight difference between the two version quantify the error caused by usage estimation error.

deficit of LongLeash and Oracle LongLeash.

You can see that as expected PowerLeash performs best compared to the both ShortLeash and LongLeash.

3.6 User Deployment

Based on the simulation results we decided to use the PowerLeash policy as the most effective algorithm for our system and used the recent past usage to estimate future interactive usage. During November and December of 2011 we recruited 22 users who owned Android smartphones. These users ran PowerLeash along with an adaptive background application (Mobility or Acoustic) for two weeks without setting battery goals. Then we enabled them to set battery goals for

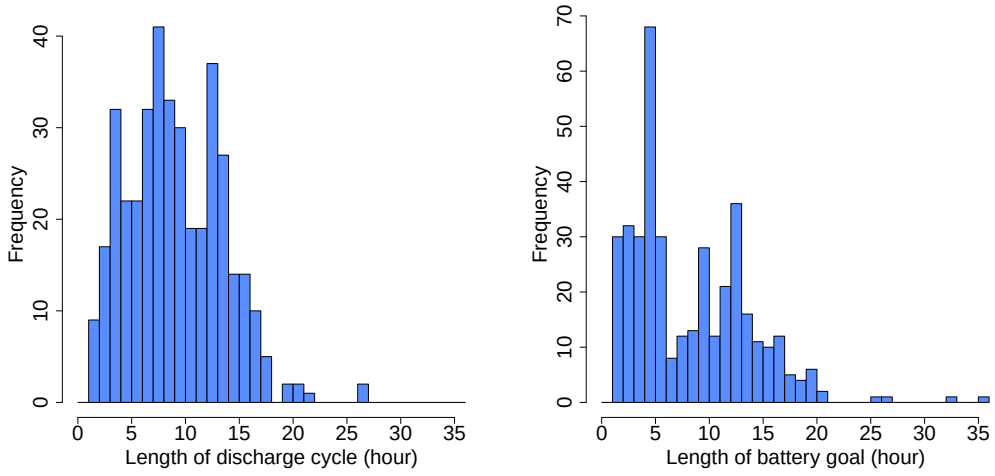
another two weeks (some users continued this phase for more than two weeks). 11 users ran a mobility tracking application (Mobility) and 6 others ran an audio recording application (Acoustic) and 4 users ran both. We will provide more details about these applications bellow.

Mobility Application The Center for Embedded Networked Sensing developed a mobility classification application to help health researchers and practitioners [RLR09]. A person's range and type of mobility have numerous health related implications. For example, after some types of treatment a patient's level of ambulation is a direct indicator of recovery [HRF11]. Our mobility classification application, referred to as *Mobility*, uses GPS speed and one-second windows of accelerometer data sampled at 20-30 Hz every minute [4] to classify the user's state as one of *still*, *walking*, *running*, *biking*, and *driving*.

To conserve battery power the Mobility uses the list of visible WiFi access poitns to detect when the user is not mobile and turns off the the GPS receiver. It reports both GPS usage and Accelerometer usage to PowerLeash.

Acoustic Application Several physiological, social and health related metrics can be assessed based on continuous audio samples [RAC11, RMM11, WBC08]. At the Center for Embedded Networked Sensing we developed an application, called *Acoustic* that samples a few seconds of audio every minute and extracts features from the recorded sound. Length of recording and sampling interval of the application are configurable. Computed features are then uploaded to a server for visualization and processing when the phone is plugged for charging. This application reports the total time that it keeps the microphone on to PowerLeash.

[4] The interval is configurable



(a) (b)

Figure 3.24: Histogram of (a) battery discharge cycles and (b) the length of battery goals submitted by users with bin size of one hour.

3.6.1 Evaluation

During the deployment the battery goals that each user selected were uploaded to the PowerLeash server in addition to all the SystemSens traces. After the deployment we analyzed users' data automatically and manually using the SystemSens visualization web interface. We identified the discharge cycles during which the user had selected a battery goal. Figure 3.24(a) shows the histogram of the length of discharge cycles for all users.

Although users were asked to specify a battery goal every day, some of them did not select goals on some days. In addition, in some cases a user would choose a battery goal and submit it, but shortly after would change it and submit again. In these cases we only consider the second battery goal, which guided the behavior of PowerLeash for most of the discharge cycle. Figure 3.24(b) shows the histogram of the length of battery goals submitted by the users.

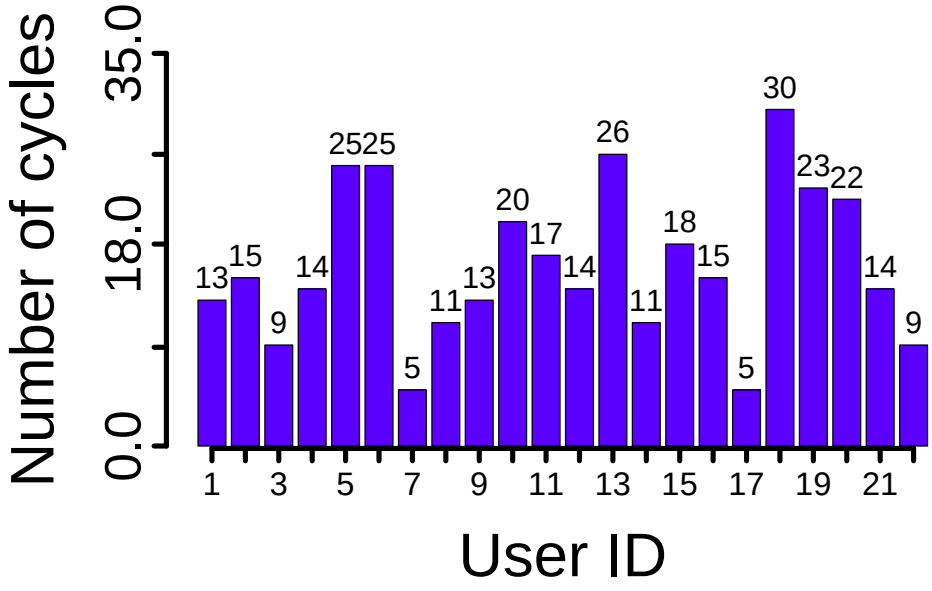


Figure 3.25: Number of discharge cycles that included battery goals for all the deployment users. On average each user submitted 16 effective battery goals. A few users (users 5, 6, 10, 13, 18, 19, 20) continued using PowerLeash for a few days after the end of the user study.

Figure 3.25 shows the number of discharge cycles that were accompanied by a battery goal by each of the users. On average each user submitted 16 effective battery goals. A few users (users 5, 6, 10, 13, 18, 19, 20) continued using PowerLeash for a few days after the end of the user study.

To evaluate the performance of PowerLeash we categorized discharge cycles to the following types:

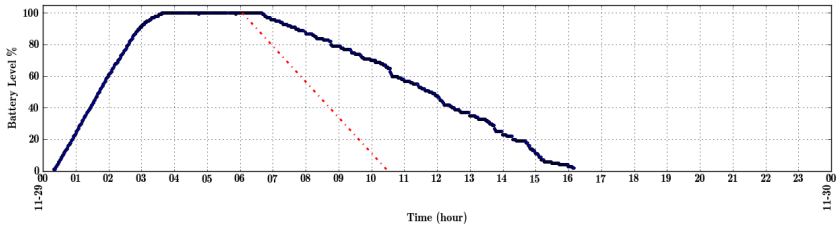
1. **Trivial:** if the selected battery goal was so short that PowerLeash did not need to throttle the background application. Figure 3.26(a) is an example.

2. **Success:** if PowerLeash successfully meets the battery goal within 30 minutes, such as the example in Figure 3.26(b).

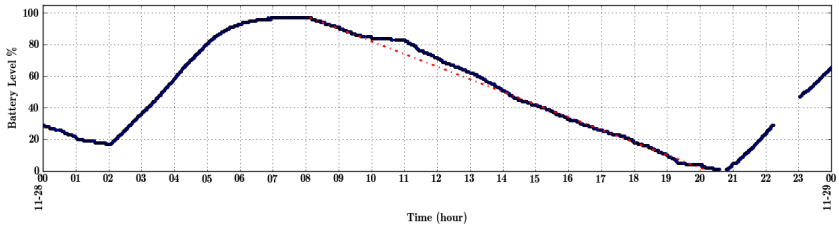
3. **Failure:** if PowerLeash fails to meet the battery goal through throttling the background application. Figure 3.26(c) shows an example.

Figure 3.27 shows the number of discharge cycles of each type for each of the users. For two users, 1 and 5, all the discharge cycles are trivial. For most users, majority of the recorded discharge cycles are trivial. This means that in most cases our users chose a battery goal that did not trigger the PowerLeash budget scheduling policy to take any action. In all these case we find that PowerLeash detects that the user's battery would last longer than the selected goal and does not throttle the background application. We asked some of the users about this and found that the term *battery goal* has been confusing to some of them. Although we trained the users at the beginning of the study, and explained what this term means, it seems some users did not have a clear understanding of the battery goal concept. We revisit this problem in Chapter 4 in further details.

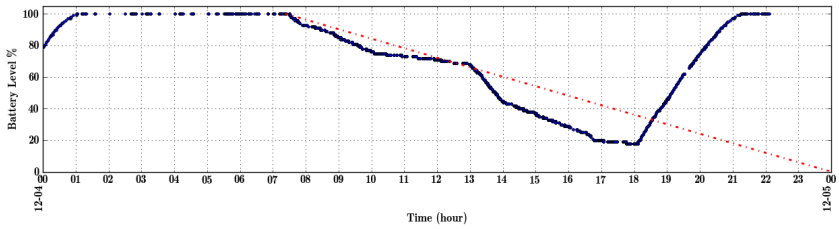
Figure 3.28(a) shows the break-down of discharge cycles across all users. We see that more than half of the total cases are trivial. When ignoring the trivial



(a) Trivial



(b) Success



(c) Failure

Figure 3.26: Examples of different battery discharge cycles from the deployment. In each case the red dotted line is a straight line that connects the beginning and end of the battery goal. These graphs are generated by the PowerLeash server visualization. (a) Selected battery goal is too short. (b) PowerLeash successfully meets battery goal. (c) PowerLeash fails to meet the battery goal.

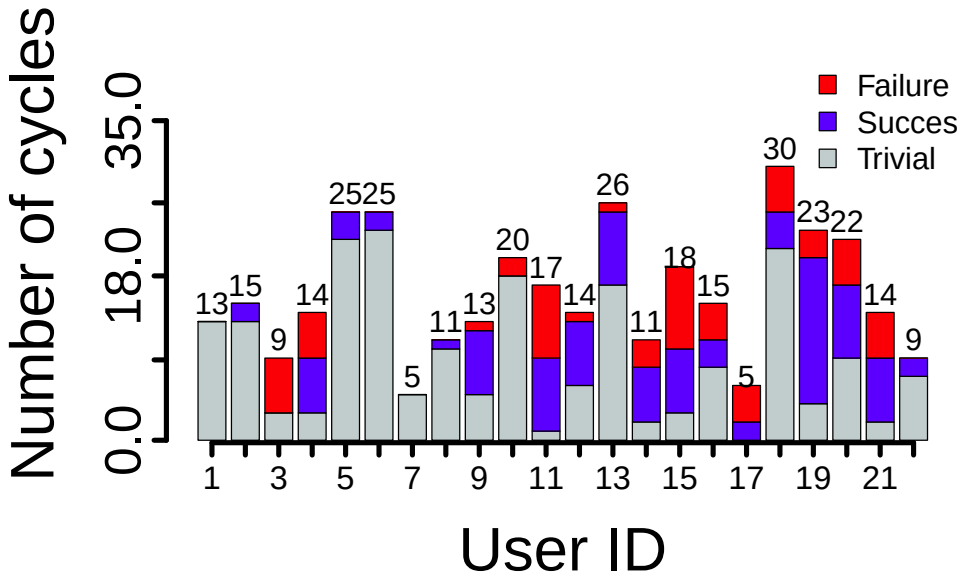
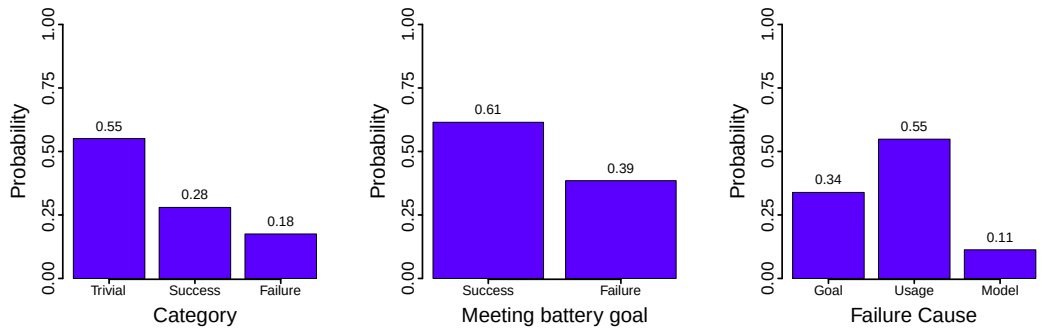


Figure 3.27: Number of discharge cycles of each type for all the deployment users. For two users, 1 and 5, all the discharge cycles are trivial. For most users, majority of the recorded discharge cycles are trivial. This means that in most cases our users chose a battery goal that did not trigger the PowerLeash budget scheduling policy to take any action.



(a) (b) (c)

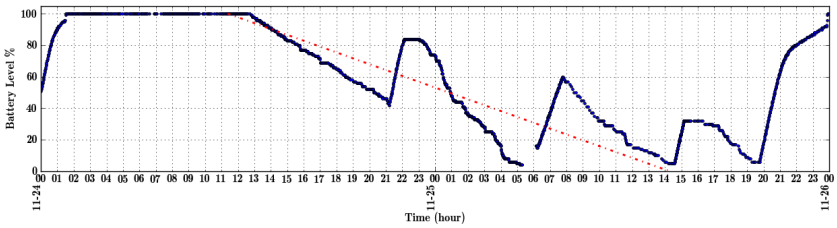
Figure 3.28: (a) Break-down of total battery discharge cycles (b) success vs. failure probability, and (c) different failure types. More than half of the total cases are trivial. When ignoring the trivial cases, in about 60% of the remaining cases PowerLeash succeeds in throttling background applications just enough to meet the battery goal.

cases, we see that in about 60% of the remaining cases PowerLeash succeeds in throttling background applications just enough to meet the battery goal as shown in Figure 3.28(b).

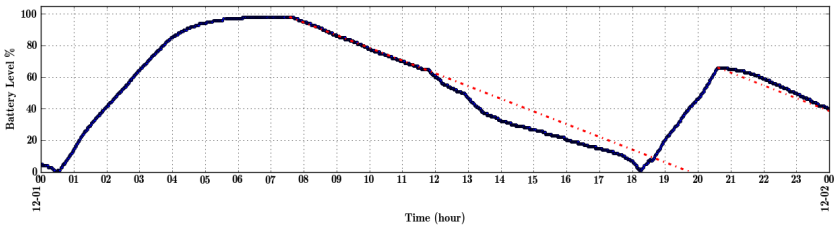
To better diagnose the failure cases that happened in 40% of the cases we inspected the traces and categorized the root cause of the failure cases. We found the following causes:

1. **Goal**: The battery goal was set too long. In many cases users set battery goals that spanned several normal discharge cycles such as the one in Figure 3.29(a).

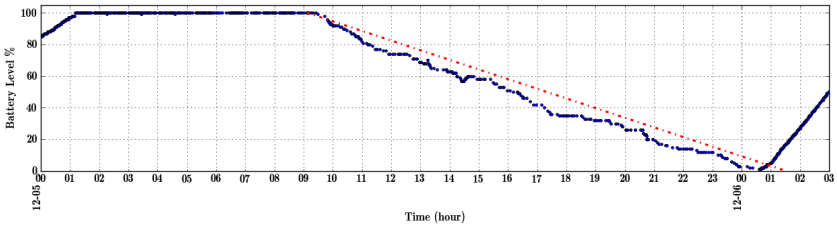
2. **Usage**: Battery drain rate is on track, until an unexpected usage spike happens and PowerLeash ends up missing the goal. Figure 3.29(b) shows one such example.



(a) Goal



(b) Usage



(c) Model

Figure 3.29: Examples of different failure cases from deployment. In each case the red dotted line is a straight line that connects the beginning and end of the battery goal. (a) The battery goal was set too long. (b) PowerLeash misses the goal because of unexpected interactive usage. (c) Due to model error PowerLeash assigns too much budget to background applications.

3. **Model**: Due to model error, PowerLeash assigns too much budget and ends up missing the goal. Figure 3.29(c) is one such example.

We recognize that distinguishing between failure cases that were caused by unexpected usage and wrong model is challenging. When inspecting the traces we gave usage the benefit of doubt, meaning that we did not categorize a case as a model induced problem unless we were 100% confident.

Figure 3.28(c) shows the break-down of the failure cases. We see that about 35% of the failures result from the same problem that rendered 55% of the cycles trivial. But the most important cause of missing battery goal is unexpected interactive usage (55%). And inaccurate model caused failure cases which contributed 11% of the total cases.

3.6.2 Discussion

We find that in most cases our users specified trivial battery goals or goals that were not feasible for PowerLeash. When ignoring both too long and too short battery goals, we find that PowerLeash meets the goals in 64% of the cases. We argue that the performance of PowerLeash based on the user deployment findings complies with the simulation results in Section 3.5.

Our simulation setup automatically selected feasible goals. The simulation performance metric is battery deficit which is the number of percent points of battery capacity that PowerLeash is missing at the end of each discharge cycle. When considering any non-zero battery deficit a failure, the probability of success for PowerLeash would be less than 2%. When considering any non-negative battery deficit a failure, then the probability of success for PowerLeash would be 50%. But when considering the granularity of the PowerLeash user interface,

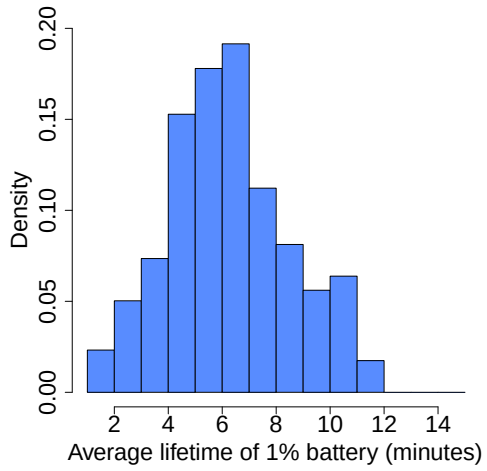


Figure 3.30: Histogram of expected lifetime of 1% battery capacity in minutes. For each discharge cycle, we get this value by dividing the length of the cycle in minutes by the change in battery level from the beginning to the end of the cycle. The man value for all users is about 6 minutes.

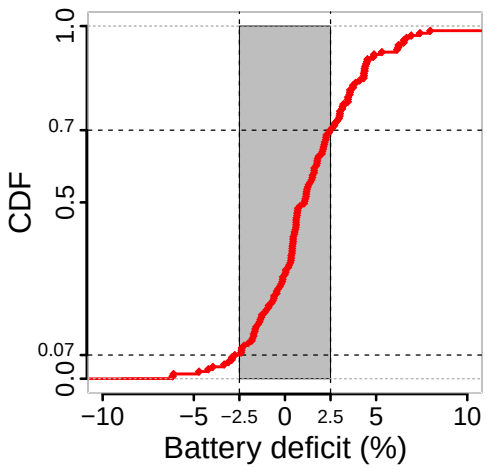


Figure 3.31: CDF of battery deficit of PowerLeash based on simulations. The probability of the deficit being inside the ±2.5 band is 63%.

which is 15 minutes, we argue that any deficit less than +2.5% and greater than −2.5% can be classified as success.

Figure 3.30 is the histogram of the average lifetime of one percent battery capacity. For each discharge cycle, we get this value by dividing the length of the cycle in minutes by the change in battery level from the beginning to the end of the cycle. The man value for all users is about 6 minutes. To count discharge cycles that end within 15 minutes before or after their corresponding battery goals as success, we should consider any battery deficit less than +2.5 and more than −2.5 success. Figure 3.31 shows the CDF of the deficit of PowerLeash from Section 3.5 and highlights the ±2.5 deficit range. We see that the probability of the PowerLeash deficit being in this range is 63% which matches the deployment findings. This gives us confidence regarding the validity of our simulation-based inference when studying PowerLeash.

3.7 Conclusion

We designed PowerLeash to give users control over their battery lifetime when running background applications. We optimized the building blocks of PowerLeash — power model, usage model, and scheduling policy — using traces from real users. After integrating the system and developing the battery adaptation interface into two existing health applications of the Center for Embedded Networked Sensing, we deployed the end-to-end system on the smartphone of 22 volunteers for one month.

Our deployment results show that PowerLeash effectively enables users to control battery consumption of background applications in a way that is much easier for them to understand than traditional ways. When users selected feasible battery goals, the system could meet the battery goal in 64% of the cases. But we found that our user interface did not effectively engage the users. Our findings from the user deployment identified the user interface of PowerLeash as a critical bottle-neck for the end-to-end performance of our system. In the next chapter we discuss the shortcomings of PowerLeash and our solutions to improve them.

CHAPTER 4

Lessons

We learned several valuable lessons from our deployment of PowerLeash on real users' smartphones. In this chapter we present these lessons. Several of these findings can help us improve future iterations of PowerLeash and design and implementation of similar systems. We use traces from the deployment to evaluate the impacts of potential improvements. We also try to generalize our findings into guidelines that can be used by other researchers.

We have categorized these lessons and guidelines based on subsystems of PowerLeash. In Section 4.1 we present the future user interface of PowerLeash, in Section 4.2 improvements to battery model, and in Section 4.3 improvements to the adaptation policy.

4.1 User Interface

When evaluating our users submitted battery goals we found that in about 55% of the cases users selected battery goals that were trivial. In other words, these goals were significantly shorter than the expected battery lifetime of the users' smartphones, even when the background application is working at its highest fidelity. Therefore, more than half of the discharge cycles of our users could not be used to evaluate the performance of PowerLeash.

One might argue that the short battery goals were users' genuine desired

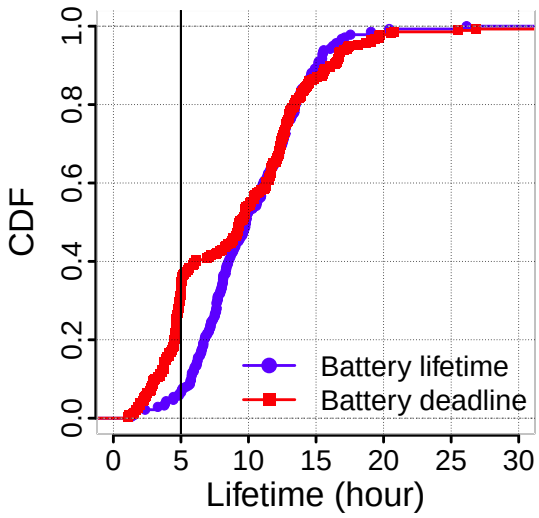


Figure 4.1: CDF of length of battery goals and actual battery lifetime of discharge cycles that start with more than 90%. The probability of lasting less than 5 hours is less than 5%, but about 40% of the submitted battery goals are less than five hours.

battery lifetime. We argue otherwise. Figure 4.1 shows the CDF of the length discharge cycles that start with more than 90% and end with less than 20% of battery capacity. We have also included the CDF of the length of submitted battery goals for these cycles. We see that the probability of lasting less than 5 hours is less than 5%, but about 40% of the submitted battery goals are less than five hours. The high proportion of submitted goals of five hours is likely due to the fact that mid way through the study we found that sometimes users submit very short battery goals. Therefore we pushed an update of the PowerLeash package that had the minimum possible goal set to five hours.

Based on this, and other anecdotal evidence from our users, we believe these very short battery goals were the product of our ineffective user interface and con-

fusion of our users. Therefore, we expect that improved user interface combined with direct feedback to users can dramatically improve the users' understanding of and interaction with PowerLeash. More details of these modifications to PowerLeash follow.

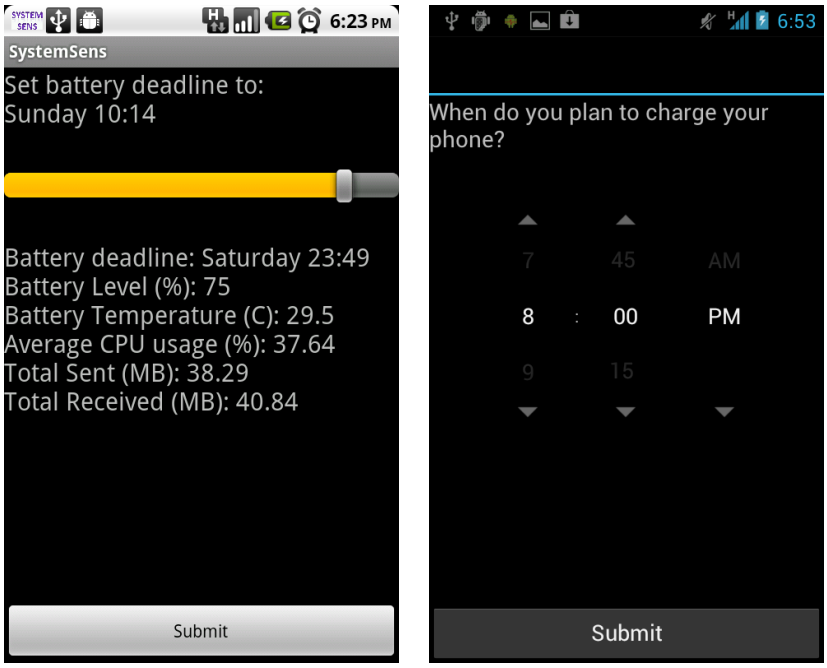
4.1.1 Battery Goal Setting UI

Figure 4.2(a) shows our current user interface. Based on our findings we are going to change this interfaces and how it works in several ways. Figure 4.2(b) will be the new user interface for PowerLeash. This new UI is different from the previous one in several ways.

Wording The current PowerLeash user interface asks the user to "specify a battery deadline." Based on interviews with a few users we know that most users found this question confusing — the term *battery deadline* was not clear for non-tech savvy users. The next generation of the PowerLeash user interface will ask a more understandable question with fewer technical terms: "When do you plan to charge your phone?"

In addition, we assumed that users are interested to know more details about the current state of their smartphone, therefore we included a summery of battery and system resource usage on the UI. We have confirmed that few uses understand this information. In fact, instead of giving users a better understanding of their phone, the additional system-level information further confused the users. Therefore, the new user interface will not include any such information and there is smaller chance of confusing the user and the UI is less cluttered and cleaner.

Input method The current PowerLeash user interface presents the user with a scrolling bar to select the desired battery goal. Scrolling the bar to the



(a) Current (b) Future

Figure 4.2: Current and Future user interfaces of PowerLeash to get the user's desired battery goal. The new UI is different from the old one in three major ways. It asks a different question, has a different input method, and always presents the user with reasonable default value.

right increases the selected value and at any time the selected value is displayed to the user. The granularity of the movement of the scrolling bar was set 15 minutes to ease the selection process. We found that this setup encouraged users to randomly select a point and click on submit. Our future user interface will require the user to select the hour and minute for when they plan to charge their battery (battery goal).

Default value In order to encourage the users to select a battery gaol, every time they finish charging their phone the current UI resets the selection to zero.

Therefore, when the PowerLeash UI is presented to the user its default selection is always zero. This default selection changed to five hours when we increased the minimum possible battery goal to five hours. We found that this approach did not work as we intended. In fact, instead of encouraging the users to select valid values, our UI encouraged selecting very short battery gaols and hence we ended up with many trivial battery goals in our dataset. Therefore, we have decided to use reasonable default values for the battery goal.

As a first step we will start by using the previously selected value as the default when the user is presented with the PowerLeash UI. We understand that more effort needs to be spent on developing algorithms to effectively infer user's desired battery goals. We believe that in this case "the best UI is no UI," and if inference of desired battery lifetime becomes accurate enough, the PowerLeash interface does not need to ask the user every time.

4.1.2 Feedback to Users

When categorizing failure cases of PowerLeash based on their cause we found several cases where the users selected very long battery goals. Anecdotally we know that users want their smartphone batteries to last as long as possible. Despite this desire, some users select to run power consuming background applications, because they value the utility or benefit of such applications more than they desire long battery life. Through PowerLeash we give these users control over their battery lifetime. But users should be given clear feedback about the implications of using PowerLeash.

Knowing that users desire longest possible battery lifetimes, some might be inclined to always select the longest possible battery goal not understanding that this will effectively stop the background application from doing what it is

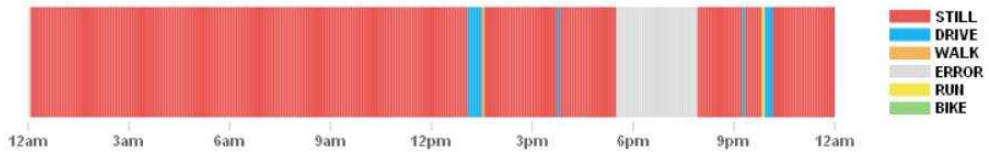


Figure 4.3: Temporal summary of Mobility classification data. The gap between 6pm and 9pm marked by the error color is caused by the Mobility application stopping as a result of running out of budget.

expected to do. We believe the solution is closing the feedback loop to the user. For example, if user is interested in tracking her mobility, based on her specified battery goal the Mobility application may run out of budget and stop working for extended intervals during the day. If this user gets to see a visual implication of the additional error that is incurred as a result of her choice of battery goal, she will have a better understanding of the implications of her choices. Figure 4.3 shows an example of visualizing the output from the Mobility application. [1]

During our deployment of PowerLeash this feedback mechanism was not ready to be used. Therefore, our users could not see the utility of the background applications and the impact of the battery goals they set on the background applications. We understand that merely offering such a feedback, does not mean all users will engage with it, unless they are truly interested in the utility of the background applications.

4.2 Battery Lifetime Estimation

The user interface that was deployed along with PowerLeash, limited the maximum possible values of the battery goal. We used the power consumption model

[1]This visualization is developed as part of the Ohmage platform.

Get model building interval: I

Get current battery level: L

for $i \in Resources$ **do**

$L \leftarrow L - \tilde{r_i} \times \beta_i$

end for

$N = \frac{L}{\beta_0}$

Return $N \times I$

Figure 4.4: Initial algorithm for estimating maximum feasible battery lifetime. $\tilde{r_i}$ is the estimated usage of the ith resource during each day — median of daily usage within past two weeks. This algorithm sets all adaptive applications work vectors to zero and estimates the needed to consume the remaining battery capacity.

to estimate this maximum value. One week into the deployment we found that our model-based estimation of maximum feasible battery lifetime was not accurate. In this section we present the problem, a diagnosis of the root cause of the estimation error and our solution for future systems.

4.2.1 Problem statement

When a user is presented with the PowerLeash UI, he/she can only select a battery goal that is less than the maximum feasible value. The deployed version of PowerLeash estimated the maximum possible value for battery goals based on the simple algorithm in Figure 4.4. In this algorithm $\tilde{r_i}$ is the estimated usage of the ith resource during each day. We used median of daily usage within past two weeks as $\tilde{r_i}$. This algorithm essentially sets all adaptive applications work vectors to zero and estimates the amount of time that it takes to consume the remaining battery capacity.

We expected that this algorithm would overestimate the maximum possible battery lifetime, because it ignores long sessions of interactive usage. Since we assumed users will diligently select their desired battery goals, and to make sure that PowerLeash does not limit users' input range, we multiplied the output of the algorithm by a factor of 2.

About one week through the study we found that for some users the maximum battery range was too small. As a quick fix we replaced this estimation with a fixed 36 hour maximum for a 100% battery. Later we performed controlled experiments in the lab to diagnose this problem. Our experiments confirmed that the algorithm in Figure 4.4 significantly underestimates the true maximum battery lifetime for most users.

Based on the user deployment results we know that in many cases users who desire long battery lifetime would select the maximum possible value as their battery goal. Therefore, having an accurate maximum feasible battery goal estimation algorithm is critical. This implies that we need to improve our estimation of maximum feasible battery lifetime to avoid giving users false expectations, or excessively limiting their choices when using PowerLeash.

4.2.2 Diagnosis

When looking at traces from several users we observed that β_0 overestimates base battery drain rate. In other words, although our model is unbiased when estimating battery drain, the model intercept is not an unbiased estimator of base power consumption.

We performed a small controlled experiment on two users to confirm our hypothesis. We asked two volunteers to run PowerLeash without any adaptive background applications. After one week we asked them to install the Mobility

	W/ Adaptive App	W/O Adaptive App
User A	1.03	0.51
User B	0.61	0.10

Table 4.1: Intercepts for battery drain model of two users with and without any background applications. the intercept of the model built while Mobility was running is significantly larger than the model with no background application confirming that the intercept is not an unbiased estimator of base power consumption.

application and continue running PowerLeash for another week. We then used the data from each of these two weeks independently to build a battery drain model as presented in Section 3.2. When comparing the intercept of these two models for each user we see that the intercept of the model built while Mobility was running is significantly larger than the model with no background application. Table 4.1 shows the estimated intercept values of the models in each case for these two users. This confirms our hypothesis that the model intercept is not an unbiased estimator for base battery consumption rate.

4.2.3 Solution

Our solution to improve the estimation algorithm is building a separate model for when the user is not running background applications. This model will be used solely to estimate the maximum feasible battery goal. We will refer to this model as the *base model*. Since the base model is not used in the adaptation algorithm, it does not need to be updated over time. This reduces the additional cost of our proposed change, because it means we need to build the base model for each user only once.

This modification of PowerLeash usage scenario is practical and does not incur additional cost. Based on our experiments with model building from Section 3.2 we know that the linear model converges after two days. Therefore, the new usage scenario of PowerLeash would be installing PowerLeash first and running it for three days first. Then installing adaptive background applications.

4.3 Improving Adaptation Policy

We evaluated PowerLeash using both simulation and user deployment and found that simulation results agree with the results from the deployment. This finding gives us extra confidence in validity of our simulation procedure and enables us to more confidently use trace-based simulation to investigate PowerLeash.

We dedicated the first two weeks of our user deployment to data collection. During these two weeks users were running a version of PowerLeash that did not support battery goal setting, and did not throttle background applications. In this section we use the traces from from the 22 deployment users during the first two weeks to propose improvements to the PowerLeash adaptation policy.

Figure 4.5 shows the CDF of PowerLeash battery deficit when using deployment traces for simulation. Considering the same 30 minutes margin for success we see that the probability of is 60% which closely matches deployment results. In Section 3.5 we found that PowerLeash error can be partially explained by both model error and *linear error*. Linear error is a metric that we developed to measure nonlinearity of battery drain rate. Analysis of variance (ANOVA) [IN87] on user deployment results in Table 4.2 formally confirms that both factors play a role in explaining battery deficit.

We experimented with using this information to improve the performance of

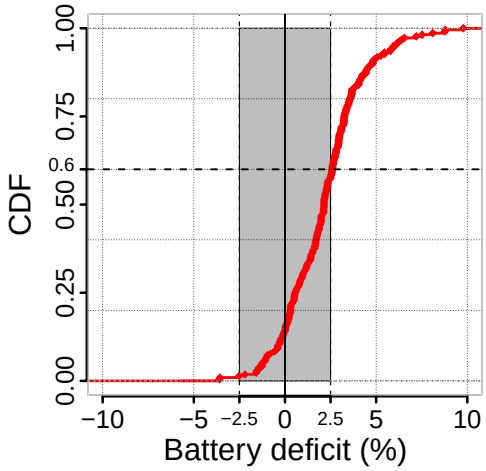


Figure 4.5: CDF of battery deficit of PowerLeash based on simulation results using deployment data. Considering the same 30 minutes margin for success we see that the probability of is 60% which closely matches deployment results.

	Df	**Sum Sq**	**Mean Sq**	**F value**	**Pr(>F)**
Model Error	1	43.31	43.31	17.652	3.887e-05
Linear Error	1	447.63	447.63	182.442	$<$ 2.2e-16
Residuals	215	527.52	2.45		

Table 4.2: Analysis of variance indicates that PowerLeash battery deficit can be explained by both model error and linear error.

$window \leftarrow CurrentTime + W$
$LineFit \leftarrow LE(Levelsvs.Time)$
$LinearError \leftarrow Last(Resid(LineFit))$
$Interactive \leftarrow \sum_{i \in Resources} E_i(window) \times \beta_i$
$Available \leftarrow Level - IL(window) - Interactive - LinearError$
for $j \in BgApps$ **do**
  for $k \in BgApp_j$ **do**
    if $Available < 0$ **then**
      $b_{jk}(i) \leftarrow 0$
    else
      $\rho_{jk} \leftarrow \frac{Available}{\alpha_{jk} \times W}$
      $b_{jk}(i) \leftarrow 2 \times \rho_{jk} \times horizon_j - B_{jk}$
    end if
  end for
end for

Figure 4.6: Pseudocode for new PowerLeash policy. LinearError is last residual of the least square fitted line to battery level readings. This algorithm subtracts the LinearError from available battery capacity to get the new available battery capacity that can be assigned to background applications.

PowerLeash. Linear Error can be calculated on-line, that is at any time during a discharge cycle we know the value of linear error so far. Simulation experiments show when the value of the Linear Error is large throughout a discharge cycle, PowerLeash battery deficit can be reduced by making it more conservative and vise versa. We implemented this idea by subtracting the running value of Linear Error from the current battery level during each interval as presented in the algorithm in Figure 4.6.

Figure 4.7 shows the CDF of battery deficit when using this technique. We see that the probability of success increases by 10% when accounting for Linear Error based on the new algorithm.

We performed similar experiments to account for model error. But we could

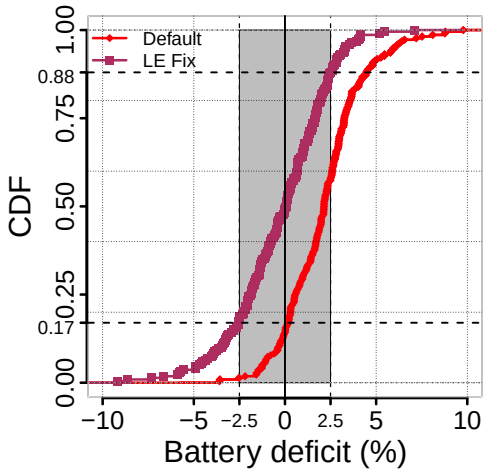


Figure 4.7: CDF of battery deficit of PowerLeash with and without the linear error offset. When offsetting linear error success probability increases by 10%.

not use model error in a similar way for two reasons. First, no accurate online measure of model error could be found. Second, when using heuristics or estimates of model error in a similar algorithm as the one in Figure 4.6 we observe that probability of success does not increase.

4.4 Summary

A closer look at results and traces from the PowerLeash user deployment revealed three areas where performance of PowerLeash can be improved.

First, we can improve users interactions with PowerLeash by simplifying the user interface. In addition to removing unnecessary information from the UI, changing the input method, and including reasonable default values can reduce user confusion. We also found that offering feedback on the utility of background applications to users can help users make better choices.

Second, we found that we can reduce the errors of the maximum battery lifetime estimation algorithm through building a separate battery model. We build this model when PowerLeash is running without any other background applications, therefore it can ore accurately estimate base power consumption of the smartphone.

Finally, by incorporating the linear error into the adaptation algorithm, the probability of meeting user battery goals will increase.

CHAPTER 5

Conclusions

5.1 Summary of the Thesis

The key hypothesis of this dissertation is whether resource management on smartphones can be improved by adapting to usage patterns of individuals. We tested this theory in two ways. First, we built SystemSens, a robust tool for measuring smartphone usage in the field. We used SystemSens in several user studies to study and characterize smartphone usage. We discovered significant diversity in smartphone usage. Along all aspects that we studied, users differ by one or more orders of magnitude. This finding suggests that resource management policies and algorithms on smartphones can become more effective if they learn and adapt to user behavior.

Second, based on our findings regarding smartphone usage, we built PowerLeash, a prototype system that adapts to interactive usage to manage battery consumption of background applications. Therefore, PowerLeash gives users control over their battery lifetime when they run background applications. PowerLeash continuously monitors the phone's battery level, the user's interactions with the phone, and progress of background applications. It builds a personalized model to estimate battery consumption based on usage and background applications progress. Using this model and other information, PowerLeash dynamically adjusts the power consumption of background applications to meet the user's

desired battery lifetime. We evaluated the performance of PowerLeash with real users. In this section we summarize the contributions of this thesis.

5.1.1 Charachterizing Smartphone Usage

We analyzed detailed usage traces from 255 users of two different smartphone platforms, with 7-28 weeks of data per user. Our traces consisted of two datasets. For the first dataset we deployed SystemSens on the phones of 33 Android users. SystemSen, and the second dataset was from 222 Windows Mobile users across different demographics and geographic locations. This data was collected by a third party. We characterized smartphone usage along four key dimensions: *i*) user interactions; *ii*) application use; *iii*) network traffic; and *iv*) energy drain. The first two represent intentional user activities, and the last two represent the impact of user activities on network and battery consumption. Instead of only exploring average case behaviors, explored the range of usage across users and over time. A recurring theme in our findings was the diversity across users. Along all dimensions that we studied, users varied by one or more orders of magnitude. We also found that users are along a continuum between the extremes, rather than being clustered into a small number of groups.

The diversity among users that we found stems from the fact that users use their smartphones for different purposes and with different frequencies. For instance, users that use games and maps applications more often tend to have longer interactions. Our study also showed that demographic information can be an unreliable predictor of user behavior, and usage diversity exists even when the underlying device is identical, as is the case for one of our datasets.

Among the many implications of our findings, an overriding one was that mechanisms to improve resource management should not follow a one-size-fits-all

mindset. They should instead adapt by learning relevant user behaviors; otherwise, they would likely be only marginally useful or benefit only a small proportion of users. We showed that despite quantitative differences qualitative similarities exist among users, which can facilitate the task of learning user behavior. For several key aspects of smartphone usage, the same model can describe all users; different users have different model parameters.

5.1.2 Managing Battery Lifetime

As an example of a system that automatically adapts to usage patterns of individuals to improve resource management, we built and evaluated PowerLeash. PowerLeash is a system that gives users control over their phone's battery lifetime and automates personalization of background applications for developers. It monitors the user's interactions with the phone and battery drain, and learns the impact of background applications on battery drain rate. With a simple user interface, PowerLeash receives the user's desired battery life as input. With this information PowerLeash dynamically changes the power consumption configuration of background applications to meet the user's battery expectation. In addition to trying to evaluate the effectiveness of learning and adapting to user patterns, we designed PowerLeash to have the following key features:

1. **Easy to deploy:** Designing PowerLeash to be easily deployable had two implications. First, we could only leverage information sources that were already available on commodity smartphones. This enabled us to immediately start gaining field experience from Battery Drain Management systems towards refining their design, rather than waiting for future generations of hardware or software that may be able to provide more accurate or granular information. Second, it required us to handle the high de-

gree of diversity that exists among devices, users, or background applications. As a result, PowerLeash does not require any off-line customization or parametrization for individual devices or users. All personalization occurs on-line and in-situ.

2. **Easy to use for application developers:** The developers of background applications are more likely to adopt our proposed APIs if they are easy to use. For this reason, we designed PowerLeash to expect applications to implement only two simple functions that enable it to poll how much work they have done since start and communicate the maximum amount of work they can do in the next interval. Further, the units of work are application-level (e.g., number of times location was polled), rather than absolute level of energy which is harder to use and varies with the platform.

3. **Easy to use for users:** The only input we ask of users is to specify a battery lifetime goal. We believe that users are capable of specifying this goal because they know when the next charging opportunity will arise.

We evaluated PowerLeash using both simulation (based on real usage traces) and user deployment. We found that PowerLeash enables users to control battery consumption of background applications in a way that is much easier for them to understand, but PowerLeash's control over battery lifetime is not perfect. When users selected feasible battery goals, the system could meet the battery goal in 64% of the cases. In addition, our findings from the user deployment identified the user interface of PowerLeash as a critical bottleneck for the end-to-end performance of our system. We investigated several ways of alleviating this bottleneck and improving the interfaces of PowerLeash and systems like PowerLeash. We also improved the adaptation policy of PowerLeash to reduce its error.

5.2 Comments on Design of PowerLeash

We built PowerLeash as a prototype system to evaluate the effectiveness of a new idea in battery management systems. In this section we first summarize the principals behind the design of PowerLeash. We then highlight why we implemented those ideas the way we did.

5.2.1 Design Choices

Several battery drain management systems have been proposed in the past that control and manage power consumption of all applications. Such systems have great control over the battery and can effectively meet user or systems specified battery goals. But they have not been adopted yet due to their strict requirements. Considering expanding diversity and fast rate of change in mobile hardware platforms, building fine-grained power consumption models is challenging. In addition, neither developers nor platform owners are willing to compromise users' interaction experience for battery, therefore the idea of limiting interactive applications has not gained popularity. In addition to relaxing requirements, we paid special attention to factors that can accelerate adoption of PowerLeash.

When designing PowerLeash we decided to relax both these requirements. First, we do not require off-line and fine-grained power consumption models for each hardware model. PowerLeash will learn a coarse-grained model and update it over time as it is being used. Second, we do not limit interactive applications. This makes meeting user specified battery goals more challenging, because PowerLeash has to deal with highly variable user interaction.

Based on our experience with building several power consuming background applications we believe that application developers are in the best place to de-

cide how to trade off fidelity and power consumption. Therefore, we designed PowerLeash to give application developers control over how and when they consume their assigned budget. When applications register with PowerLeash they can decide their budget planning interval (horizon). A long horizon allows the application to consume its budget in bursts. Also, PowerLeash allows application developers to define the unit of their budget. This relieves them from having to deal with units of energy or power that are difficult to measure and track.

Finally, PowerLeash is an application itself and interacts with users. We designed it to be simple. The PowerLeash user interface is launched automatically right after a user unplugs the phone from a charging source to remind the user to set desired battery goal. The user needs to input only one value and it will be used for managing all adaptive background applications. If PowerLeash detects that the user's desired battery goal will not be met, it shows an early warning to the user. The user can either change the battery goal or avoid unnecessary use.

5.2.2 Implementation Choices

When building the PowerLeash prototype we made three key decisions. We consider these as implementation decisions that were made based on the context of the project and other limitations. We highlight these and the reason behind them. Future incarnations of PowerLeash or similar systems may be implemented differently.

We implemented PowerLeash on the Android platform as a user-level application. An alternative decision would have been modifying the Android platform and including PowerLeash inside the platform. This would have given PowerLeash much more control over background applications. We chose not to modify the Android platform to increase the potential user base of PowerLeash. Re-

cruiting volunteers and deploying PowerLeash on their smartphones would have been much more difficult, if we had to flash their phones with a custom ROM. In essence, we prioritized broader user base and user studies to the advantages of being part of the platform or kernel.

PowerLeash was a first prototype of our proposed design. For this first prototype we chose to use simple and understandable algorithms. This had two advantages. First, it allowed us to easily diagnose performance problems. Second, it allowed us to deploy and experiment with the end-to-end system, identify the bottlenecks and spending more effort in tuning components and algorithms that significantly affected the end-to-end performance.

Several parts of PowerLeash that require inference and model building were implemented inside the PowerLeash back-end server. For example, the PowerLeash client uploads all battery and usage traces to the server where the power consumption model is built and updated. This client-server design has several privacy implications. But we decided to rely on a server because it allowed us to retain all usage traces. use them to evaluate the performance of PowerLeash offline, and experiment with new algorithms and models. Future implementations of PowerLeash or similar systems may choose to implement the entire functionality on the phone. Based on our experience this is possible and may be necessary for commercial or very large deployments — we expect that the server can become a scalability bottleneck in large deployments.

5.3 Future Work

We highlight potential future work in two directions. First, ways to improve PowerLeash and personalized battery management systems. Second, extending

the idea of automated personalization to management of other resources such as network or memory traffic.

5.3.1 Improving PowerLeash

One area for future research would be exploring the extent to which PowerLeash, and battery drain management systems in general, can relieve users from providing battery goals every day through learning their daily patterns. We did not explore this, and instead focused on building the mechanisms that are needed to meet a battery goal once it is given to the system. However, based on related research and anecdotal evidence from the usage traces collected during our studies, we believe there is significant opportunity for inferring users' intentions. Several context hints can be used by such algorithms including time, location, and recent usage patterns.

A second important area that needs further investigation is considering multiple adaptive applications. We focused on the case when there is only one primary background application. We did, however, include users with two adaptive applications in our deployment of PowerLeash and found that PowerLeash can manage more than one applications. But we overlooked several key questions that would arise when there are multiple background applications including prioritizing applications based on user's preference or other metrics. Also, we expect additional modeling considerations must be made when two or more background applications contend on a single resource (such as GPS).

To extend PowerLeash's reach and facilitate experimentation with our prototype we decided to implement PowerLeash entirely in user space. One consequence of this choice was that PowerLeash can manage only those applications that implement its public interfaces and cannot control other applications. Also

PowerLeash is not able to enforce the assigned budgets on the background applications — it relies on cooperation of background applications. Therefore, a malicious application can cause PowerLeash to miss the battery goals or to starve other applications. A future implementation of PowerLeash may choose to reside inside the mobile platform, instead of user space. If so, applications would not need to directly report their work to PowerLeash and PowerLeash can enforce the budget assignments. Since the set of distinct types of resources on a smartphone are limited, the mobile platform can accurately track their usage by each application.

Finally, we made several key decisions regarding the design and implementation of PowerLeash based on its usage context inside a research environment. A commercial implementation of such a system may need to be designed differently. For example, we used a client-server architecture that keeps all usage and battery consumption traces on the server, because this enabled us to explore different algorithms and evaluate their performance off-line. Commercial systems can be built entirely on the smartphone without relying on any servers.

Finally, for privacy considerations PowerLeash does not take advantage of two important sources of context information: location and contents of applications such as calendar and email, to predict future interactive. It is possible that including these sources of information, can improve the performance of PowerLeash. The privacy concerns can be alleviated if future systems avoid transferring data to a central server for model building, and instead build usage and battery models on the phone.

5.3.2 Extending Automated Personalization

After identifying the potential for effectiveness of personalization in improving resource management on smartphones, we focused on building a system to improve battery management by adapting to usage patterns. But this approach can be applied to several other aspects of smartphones. Effective management of other resources on smartphones such as memory, CPU, and network interfaces will improve both usability and power consumption.

We observed recurring patterns in application usage of individuals. While each user has her own favorite set of applications and diurnal patterns in application execution, no two users are identical in these terms. This provides a unique opportunity to improve memory management on smartphones. By observing past application usage, smartphone operating systems can predict the application that is most likely to be called in the near future. Based on this prediction, the operating system can allocate memory to that application in advance. Therefore, when the user invokes an application he/she will not have to wait for it to load. Similar systems have been tested on desktops [Esf06], but because memory is much more limited on mobile phones the resulting gain is likely to be higher on smartphones.

Current smartphones are equipped with several network interfaces, including: cellular, Wi-Fi, and bluetooth. These network interface have complementary features. Cellular networks are available more ubiquitously but they have lower bit rates and higher power consumption per byte. On the other hand, Wi-Fi networks offer higher bit rate and consume less energy per byte. Therefore, opportunistic communication over Wi-Fi access points is favorable. Strong repeating patterns in human mobility imply that the Wi-Fi management subsystem of a smartphone can improve its performance by adapting to the individuals usage patterns and

habits.

It has also been shown that delaying network transmissions can result in significant energy saving [BBV09, SNR09]. But in our traffic characterization of smartphone users we found that for different users the ratio of interactive to total traffic is widely different among users. Therefore, interface management policies that are based on delaying transmission may be personalized to avoid significantly inhibiting user experience.

APPENDIX A

Measuring Smartphone Usage with SystemSens

To capture usage context and battery information in the wild we have developed a tool called *SystemSens*. It collects and logs smartphone usage parameters in the wild in an unobtrusive, and expandable way. SystemSens consists of an Android logging client and a visualization web service. This tool has been used in several deployments, where it has helped us better understand users' interactions with research applications [BEH06, HRK10]. In this chapter we describe its design and our experience with it. We will present the findings from data collected by SystemSens in next chapters.

During the past three years SystemSens evolved through three phases. Initially (v0) it was a simple battery and screen status logging tool that kept the traces on the SD card. The next major version of SystemSens (v1) recorded rich operating system and network information such as packet headers, and uploaded data in XML format to SensorBase [CYH06]. This version used low-level Android libraries and could thus run only on developer phones, which proved to be a significant limitation because recruiting users became harder. In the next version of SystemSens (v2), we dropped some of the low-level logging capabilities to be able to run on stock Android phones. From this version we also started uploading data in JSON format to a collection and visualization server. The current version of SystemSens (v3) allows other (third-party) applications to log additional sensors through SystemSens, thus offering an extensible platform for

monitoring smartphones. It also allows third party applications on the phone that are interested in context information to receive real-time information from SystemSens.

SystemSens is designed to be unobtrusive—it has no user interface to minimize impact on usage, and it has a small footprint in terms of memory, CPU, and energy consumption. When having to choose between getting rich, low-level information and portability, we chose portability to run on any Android smartphone. Based on our experiments, we find that event-based data logging is more efficient than periodic polling. We also find that the primary energy cost of logging relates to how often the device is woken up, and the marginal cost of reading and storing additional sensory information is not significant. Thus, the designer of tools like SystemSens should optimize for maximizing sleeping and worry less about the amount of information being collected and logged.

A.1 Architecture & Design

In this section we introduce the architecture of SystemSens. We highlight our design decisions and the reasons behind them. The principal goal that derives most of our design and implementation decisions of SystemSens is to keep a low profile in terms of resource consumption on the smartphone. Therefore we have designed SystemSens to minimize the amount of work on the client and delegate complexity to the server and (offline) analysis. We will refer to this choice as the *thin client principle*.

A second goal behind many design choices of SystemSens is to keep the user-base as broad as possible. Most importantly we abandoned some of the features of version 1 to be able to to run SystemSens on stock Android smartphones.

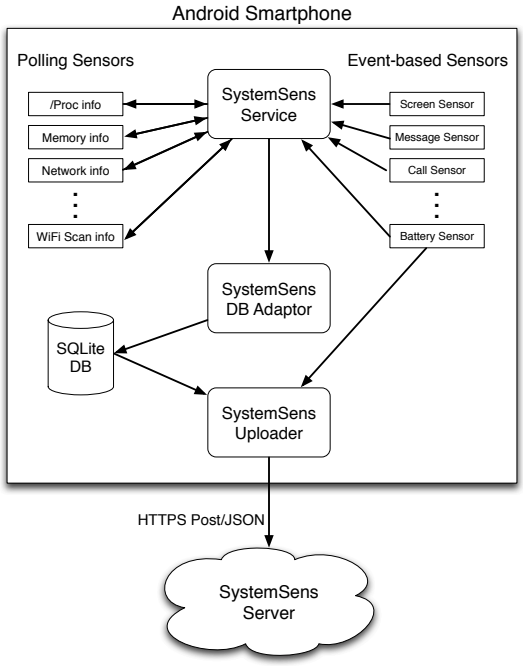


Figure A.1: The architecture of the SystemSens client application. Event-based sensors generate a log record whenever the corresponding state changes. Polling sensors record the corresponding information at regular intervals. The main thread is responsible for recording both event-based and polling sensors.

A.1.1 SystemSens Client

Figure A.1 shows the architecture of the SystemSens client application. The SystemSens client continuously runs as a background Android service. We chose not to implement any user interfaces to minimize impact on usage. The client records and uploads a range of operating system events. Each group of related OS information is recorded by a virtual "sensor." SystemSens supports two types of sensors. *Event-based* sensors generate a log record whenever the corresponding state changes. For example, the screen sensor records the state of the screen every time it turns on or off. *Polling* sensors record the corresponding information at

regular intervals. For example, every two minutes the average CPU and memory usage are recorded. Table A.1 is the list of all the sensors currently supported by SystemSens. To minimize energy consumption of SystemSens we prefer an event-based sensor to a polling sensor to record the same information.

The main SystemSens thread is responsible for recording both event-based and polling sensors. All polling sensors are queried at fixed intervals (currently set to two minutes — we found this value good enough). In previous versions we polled the sensors more aggressively when the user was interacting with the phone. We found that variable intervals make visualization and analysis of traces more difficult. To improve the performance of the recording operation, all generated records are kept in a memory buffer that is flushed to an SQLite database table on the phone regularly in a separate thread.

The most energy and resource consuming task of the SystemSens client is uploading the records to the server because of high energy consumption associated with network transmission. Since none of our research studies require real-time data analysis, we designed the upload mechanism to upload only when the phone is being charged. Most smartphone users charge their phones overnight, which offers SystemSens enough opportunity to upload all the data. In addition, with this policy upload happens at a time when users tend to have zero interaction with their phones. This scheme would fail if a user charges her phone while it is turned off, or at a location with no network connectivity, or for users with multiple batteries who charge them outside the phone.

When the upload thread starts, it reads the local database in batches of 200 records, URL encodes them and posts them to the SystemSens server over HTTPS. If the post request succeeds, the records are deleted from the database with a single range query. We decided not to authenticate clients when uploading

Name	Type	Information
battery	Event	Battery level, voltage, temperature, health and charging status
callstate	Event	State of dialer application
dataconnection	Event	State of connection to data networks
servicestate	Event	Operator information
gpsstate	Event	State of Android GPS provider
callforwarding	Event	State of call forwarding
netlocation	Event	Course (network based) location
systemsens	Event	Start time of the SystemSens application
message	Event	State of pending unread text messages
screen	Event	State of the screen
celllocation	Event	ID of the connected cell tower
call	Event	Voice call state
servicelog	Polling	Start and end time of background services
activitylog	Polling	Start and end time of applications
cpu	Polling	Contents of /proc/cpuinfo
meminfo	Polling	Contents of /proc/meminfo
memory	Polling	Android reported memory information
netlog	Polling	Contents of /proc/net/dev
network	Polling	Traffic statistics per interface
wifiscan	Polling	Signal strength of visible WiFi APs
appresource	Polling	Memory and CPU usage of running apps
netiflog	Polling	Traffic statistics per applications

Table A.1: List of default SystemSens virtual sensors, their type, and meaning.

to be able to support users who are not registered on the server and keep the potential user base broad.

If for any reason, such as network disconnection, the server does not confirm receiving a batch of data, they will not be deleted. This approach further simplifies the logic of the client, but may result in duplicate records in the database. However, filtering duplicate records during analysis is trivial.

A.1.2 Data format

We chose the JSON format [jso] for SystemSens data both on the server and the client. In both client and server databases JSON objects are stored as strings. This gives us the flexibility to add new types of sensors and data record types without the need to change database schema. In addition, handling JSON objects on Android is slightly more efficient than XML [Sha09].

Each record contains the IMEI[1] or ESN[2] of the phone to identify the user, the timestamp in milliseconds in UTC, the human readable date in phone local time zone, and the version number of SystemSens. Each record contains a record type field. The type value is used to parse the contents of the data field which is itself a JSON object. Figure A.2 is an example of the JSON object generated by the screen sensor.

A.1.3 SystemSens Server

Our server configuration consists of a MySQL database on a Linux machine running Apache. All server functionality is implemented in Python.

When the server receives a post request from a client it parses the JSON

[1]International Mobile Equipment Identity
[2]Electronic Serial Number — equivalent to IMEI for CDMA phones.

```
{'date':       '2011-2-22 0:42:56',
 'time_stamp': 1298364176416,
 'type':       'screen',
 'user':       '355060041008892',
 'ver':        '3.2',
 'data':       {'status':'off'}}
```

Figure A.2: Example of a SystemSens data record. Every SystemSens record contains time stamp, local time, user ID, version number and type name. The content of the data field is another JSON object and its structure depends on the type field.

object and inserts the type, user, and date records along with the original JSON string in a database table. This table is indexed on time, user and date fields to facilitate fast search queries on the records.

To preserve privacy all server functionalities that read the data require authentication. Each login is paired with an IMEI or ESN, therefore each user can only access her data. The data is presented as a number of time graphs such as those in Figures A.6 and A.8. Each graph has time as the X axis and the user can control the range of the X axis using time controls. Graphs can be generated for each data type or combinations. Adding a new graph to the visualization service is as simple as adding a new function.

A.1.4 External Sensors

Other applications can act as virtual sensors for SystemSens, and log information through SystemSens by implementing a simple AIDL [3] interface. SystemSens will

[3] Android Interface Definition Language

treat the application as another virtual sensor and poll its values.

Several research applications developed at CENS use this feature to closely monitor their resource consumption. Other research applications can use the SystemSens framework to upload and monitor their data. For example, a sleep survey application that uses accelerometer to detect when the user wakes up and asks questions regarding quality of sleep logs its accelerometer usage through SystemSens.

A.2 Evaluation

In this section we present basic evaluation of the performance of the SystemSens client. We summarize statistics regarding the amount of data that SystemSens generates. Next we evaluate the impact of running SystemSens on the battery life of Android phones. CPU and memory usage of SystemSens is insignificant. On a Samsung Galaxy S smartphone SystemSens on average consumes about 3% and 2.5% of CPU time in user and kernel mode, respectively. It occupies about 4% and 3% of memory pages in private and shared mode, respectively.

A.2.1 Data Size

The amount of data that SystemSens generates for each user varies widely depending on usage and version of Android. When there is more interaction, more event based records are generated and on older versions of Android some of the virtual sensors are not accessible. Figure A.3 is the CDF of the number of records per hour for two example users both with high-end phones running Android 2.2. The median for the first user is 408 and for the second user is 445.

The length of records primarily depends on their type. Table A.2 contains the

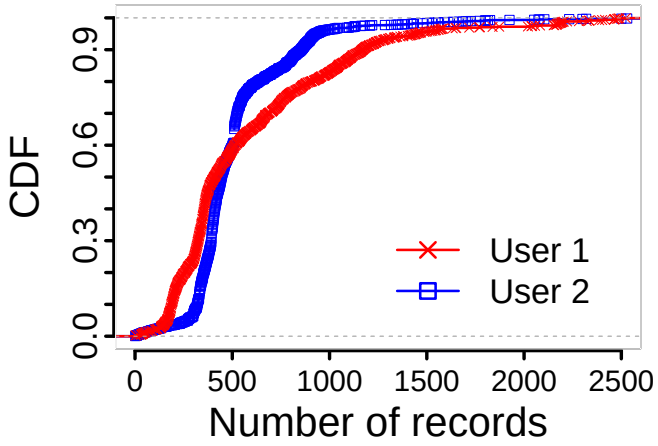


Figure A.3: CDF of the number of records generated per hour for two example users. The median for the first user is 408 and for the second user is 445.

median length of records of each type in the SystemSens database. The median length of all records is 159 characters and the mean is 362.

Based on these statistics and further inspection of a subset (more than 10 million records) of the records in the database , we find that SystemSens on average generates about 2.5 MB of data for each user per day. On the high end this value is 5.5 MB and on the low end 0.75 MB.

A.2.2 Energy Consumption

To evaluate the energy consumption of SystemSens we report the results of two types of experiments. First, we directly measure power consumption of a phone when running SystemSens. Second, we measure the reduction in battery life time of a phone when running our tool. Both of these tests are common in the mobile computing community.

We placed a high frequency digital voltmeter in parallel and a current meter

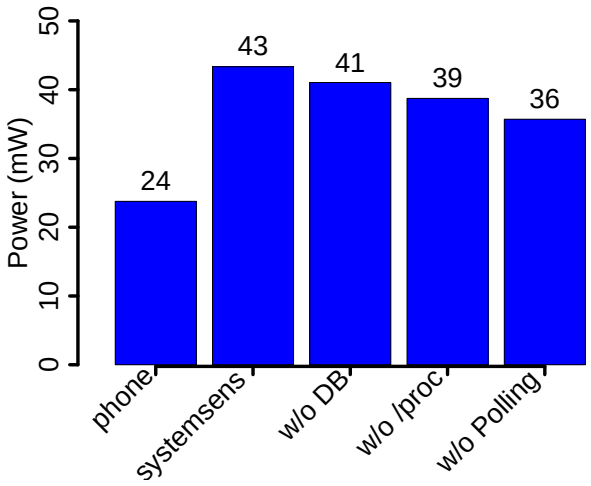


Figure A.4: Average power consumption of a Galaxy S smartphone with different versions of SystemSens. When the phone is woken up the marginal cost of polling additional sensors is insignificant. In addition, writing data into the persistent storage is not expensive in terms of power. Therefore, the most effective way of reducing energy consumption of SystemSens is increasing the polling interval.

Type	Length	Type	Length
activitylog	133	battery	224
callforwarding	153	call	145
celllocation	159	callstate	150
dataconnection	181	cpu	324
gpsstate	192	meminfo	459
message	146	memory	128
netiflog	314	netlog	2871
servicelog	449	screen	143
servicestate	235	network	149
appresource	4132	netlocation	223
systemsens	190	wifiscan	154

Table A.2: Median length of different record types of SystemSens. The median length of all records is 159 characters and the mean is 362.

in serial with the battery of a new Samsung Galaxy S Android smartphone. We measured the current and voltage across the battery with a frequency of 50 Hz. For each experiment we measured power consumption for 10 minutes and report the mean power consumption in Figure A.4. No other third-party application was running on the phone during these tests. We start measuring power a few minutes after the screen turns off.

The average power consumption of the phone when no background application is running is about 24 mW. When SystemSens is running the phone consumes about 43 mW. To put these numbers in perspective note that this device consumes about 500 mW when the screen is on, and about 1500 mW when it rings to an incoming call.

When there is no interaction with the phone, SystemSens consumes about 19 mW just for recording the polling sensors. Each polling cycle consists of waking

the phone up, reading three files from /proc, querying some Android libraries for other polling sensors, and finally writing the results, along with event-based sensor data received during the past polling interval, in the local database. To measure the power consumption associated with each of these steps we performed three additional experiments. The results are summarized in Figure A.4. Reading three files from /proc consumes about 4 mW. Reading the other polling sensors consumes an additional 3 mW, and writing the data into the database consumes about 2 mW on average. These results suggest that when the phone is woken up the marginal cost of polling additional sensors is insignificant. In addition, writing data into the persistent storage is not expensive in terms of power. Therefore, the most effective way of reducing energy consumption of SystemSens is increasing the polling interval.

Calculating battery lifetime using the declared battery capacity and measured power is inaccurate because it cannot account for many external factors such as variations in battery voltage [RVR03] and user interactions. To get a realistic estimation of impact on battery life we performed our second experiment. We implemented a simple application that continuously records battery levels. We installed this application on a Nexus One with a full battery. We left the phone on a shelf until its battery died. We repeated this experiment with SystemSens and placed the phone in the exact same location. Figure A.5 shows the recorded battery levels and the least squared fitted lines for both cases. It shows that SystemSens reduced the battery lifetime of this phone by about two hours when there is no user interaction. Note that the impact is different on different hardware platforms. We chose the used Nexus One rather than the new Galaxy S, because its shorter battery life made our experiments shorter.

Both these experiments capture worst case scenarios, because when a phone is

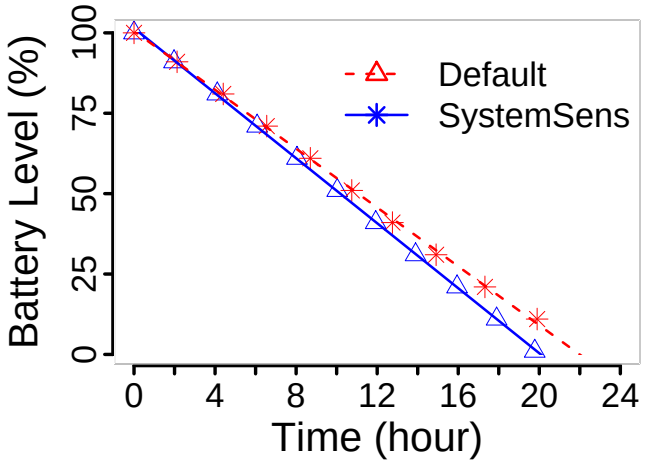


Figure A.5: Lines fitted to battery level readings show the impact of running SystemSens on battery life of an old Nexus One phone. SystemSens reduced the battery lifetime of this phone by about two hours.

actively used some of the polling events occur when the phone is already powered up by usage. By investigating SystemSens logs from many users, we find that when the phone is not being charged, between 8% to 20% of polling events happen when the screen is on or within 30 seconds (sleep timeout) after the screen is turned off. We found the distribution of this ratio robust to the exact choice of sleep timeout value. Furthermore, based on anecdotal evidence from our deployments, the impact of SystemSens on battery life of actively used smartphones does not disturb normal day to day usage of the phone.

A.3 Monitoring Smartphone Research Deployments

We built SystemSens to log usage and battery data from real users' Android smartphones. But SystemSens found applications in other research projects at the Center for Embedded Networked Sensing. In this section we outline its appli-

cation as a smartphone monitoring tool for other projects that use smartphones.

Using smartphones as a research platform is challenging [KCC07]. Several engineering barriers such as closed and fragmented platforms [Oli09] make it hard to develop software that works robustly on a range of devices. In addition, the diversity of usage patterns across users [FMK10], makes it difficult to debug unexpected behavior, interpret results, and draw general conclusions.

Through deploying and supporting several research applications on smartphones, we found that capturing the broader usage context greatly simplifies some of the challenges. That is, research applications should not only capture information of direct interest (e.g., location for an application interested in user mobility) but also other information on how the smartphone is being used by the user (e.g., CPU, memory, battery, etc.) By doing so, researchers can better understand the environment their application operates in and interpret and qualify the results more accurately. For instance, if an application interested in location information also captures CPU and screen activity, it can better distinguish between users that are sedentary versus those that leave their smartphones on the desk for long periods.

In this section, we motivate the need for capturing broader usage context when deploying research applications. During the past two years, SystemSens was used in several deployments. During the summers of 2009 and 2010 it was used by about 30 high school students who ran a range of participatory sensing applications [BEH06]. Within the past year, three pilot deployments of the Andwellness project [HRK10] had SystemSens running on the smartphones of their users. The Mobilize project [mob12] is using SystemSens to gauge high school students' engagement with their smartphone data collection tool. These were in addition to several internal studies. In all these cases we found that

data from SystemSens provided valuable insight into the usage context, helped us optimize our applications to consume less energy, and pointed us to external factors that were affecting our studies, but we were not aware of. We present a few examples.

A.3.1 Unexpected User Behavior

When using smartphones as a research platform, it is convenient to assume that the research application is the only one on the subjects' smartphones, because it is the only application that researchers can control. This assumption works most of the time, but when it does not, it is difficult to identify the culprit.

For example, on a memory constrained device, seemingly irrelevant issues may cause problems for a research application. In Android, if a new application is launched when there is not enough free memory, the system automatically reclaims memory from other applications, even if that leads to stopping some of the background services. In one incident, we had a location tracking application that was supposed to continuously reside in memory and run. When testing it in the lab, it presented the right behavior, and worked as expected for most users. We experienced problems with a few of our users, which we could not explain initially. But after looking at memory and interaction traces from their SystemSens logs, we found that those were active users with high memory usage. In another deployment, some of our users reported repeated phone reboots. Inspection of memory usage information from SystemSens showed that these were heavy phone users, who continuously invoked different applications, and therefore Android was under memory pressure on those phones, and that version of Android on that hardware platform resorts to rebooting the phone when memory is scarce.

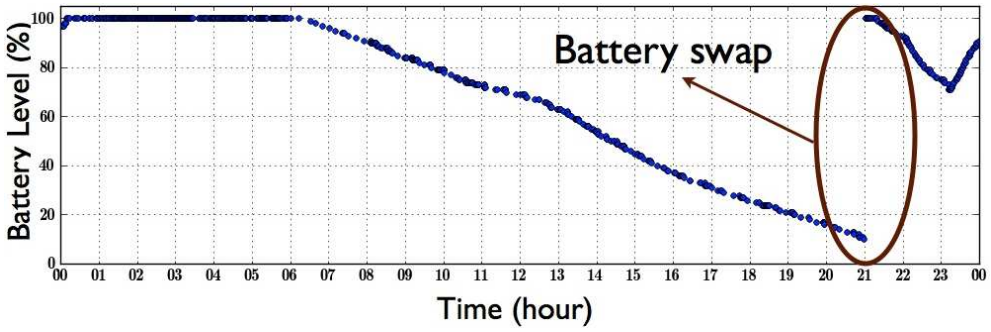


Figure A.6: Snapshot of SystemSens battery graph of a user who used a backup battery.

In every deployment of research software we have encountered a few users with other forms of unexpected behavior that impacts the research applications. As an example, consider the graph in Figure A.6. It is a snapshot of the SystemSens battery level graph over 24 hours for a user who used two batteries. The sudden jump to 100% at about 9pm, indicates switching batteries. We never considered this behavior when developing and testing our log upload mechanism that kicks in only when the phone is being charged. If battery swap behavior were dominant, no data would get uploaded (and we wouldn't be able to distinguish between the application not working and upload failures). Fortunately, this user sometimes also charged her phone, which led only to delayed uploads and not no uploads.

A.3.2 Debugging Battery Consumption

Using SystemSens in our deployments helped us better understand the impact of our research applications on the battery experience of our users. To do this, just monitoring battery information is not enough. Very often when trying to explain short battery life of the users we identified problems in our software and fixed it, but in many cases other information from SystemSens helped us identify

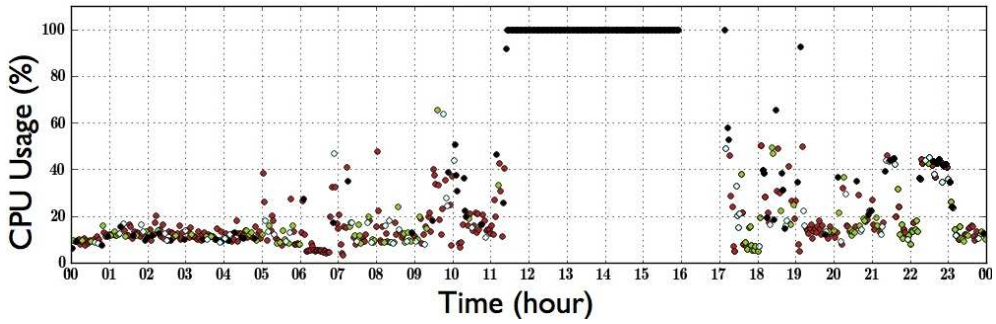


Figure A.7: Snapshot of a SystemSens graph showing average CPU usage during one day for a user. Colors represent CPU frequency.

external factors that were contributing to high battery drain.

For a few users, when inspecting SystemSens graphs we encountered unusual CPU activity as shown in Figure A.7. We traced the problem to a bug in the GPS driver of that phone model. Under specific conditions, the GPS driver consumed 100% of CPU time at the highest frequency until no juice was left in the battery and the phone died.

For some other users we found an unusually high number of network disconnection events. Figure A.8 is a snapshot of the SystemSens graph for the number of cellular disconnections for one such user who worked in a building with very poor wireless reception. SystemSens reported repeated disconnection events during day hours. When this happened the cellular interface consumed significantly more power, which resulted in much shorter battery life time.

In some other cases, we discovered that some of our users had ruptured batteries. Their SystemSens battery graphs showed that the batteries could not hold charge more than two hours — an unusually short life for that phone model. When we inspected the phones we found the batteries had been ruptured.

In each of these instances, without access to the information from SystemSens,

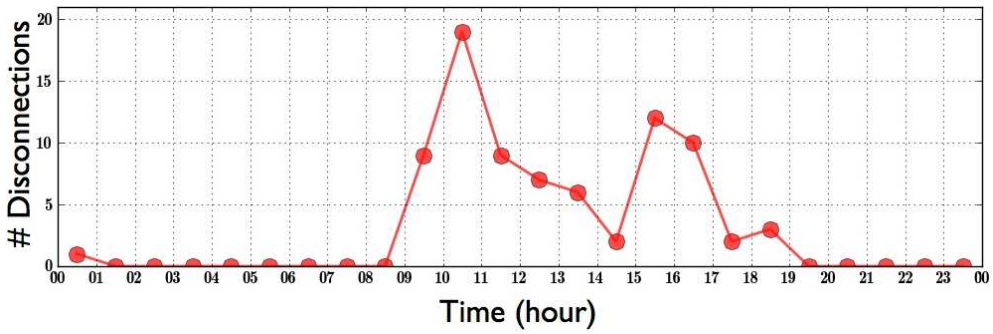


Figure A.8: Snapshot of a SystemSens graph showing the number of cellular disconnection events per hour during one day for a user with poor connectivity at work.

we could not have identified the real culprits. That would have led to unsatisfied users who would have blamed our application for the drain and likely uninstalled it.

A.4 Summary

We have been developing and using SystemSens as a tool to capture usage and context related parameters in our research for the past two years. SystemSens has become a robust logging tool with a portable visualization server implementation. It is now an integral part of our research deployments. We released the source code [4] for other interested researchers.

In future we will continue improving SystemSens. Specifically we will address the potential upload problems by including checks to trigger opportunistic uploads when the default scheme fails repeatedly. We will also include mechanisms for third-party applications to log information in an event driven manner.

[4] Available at *http://systemsens.cens.ucla.edu/*

References

[Aho10] Tomi Ahonen. *Mobile Telecoms Industry Review*. www.tomiahonen.com, 2010.

[ATS11] B. Anand, K. Thirugnanam, J. Sebastian, P.G. Kannan, A.L. Ananda, M.C. Chan, and R.K. Balan. "Adaptive display power management for mobile games." In *ACM MobiSys*, 2011.

[BBV09] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. "Energy consumption in mobile phones: A measurement study and implications for network applications." In *IMC*, 2009.

[BCW88] R.A. Becker, J.M. Chambers, and A.R. Wilks. *The new S language*. Chapman & Hall/CRC, 1988.

[BEH06] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M.B. Srivastava. "Participatory sensing." In *World Sensor Web Workshop*, 2006.

[BGM05] L. Brown, N. Gans, A. Mandelbaum, A. Sakov, H. Shen, S. Zeltyn, and L. Zhao. "Statistical Analysis of a Telephone Call Center." *Journal of the American Statistical Association*, **100**(469), 2005.

[BMV04] P. Benko, G. Malicsko, and A. Veres. "A large-scale, passive analysis of end-to-end TCP performance over GPRS." In *INFOCOM*, 2004.

[BRC07] N. Banerjee, A. Rahmati, M.D. Corner, S. Rollins, and L. Zhong. "Users and batteries: Interactions and adaptive energy management in mobile systems." *LNCS*, **4717**:217, 2007.

[can12] "Smart phones overtake client PCs in 2011." `http://www.canalys.com/newsroom/smart-phones-overtake-client-pcs-2011`, 2012.

[CBR04] R. Chakravorty, S. Banerjee, P. Rodriguez, J. Chesterfield, and I. Pratt. "Performance optimizations for wireless wide-area networks: Comparative study and experimental evaluation." In *MobiCom*, 2004.

[CCC04] J. Chesterfield, R. Chakravorty, J. Crowcroft, P. Rodriguez, and S. Banerjee. "Experiences with multimedia streaming over 2.5 G and 3G Networks." In *BORADNETS*, 2004.

[cho09] "Trends in Japanese Residential Traffic." `http://www.isoc.org/isoc/conferences/bwpanel/docs/20091111_bandwidth_cho.pdf`, 2009.

[CK09] L. Cottrell and S. Khan. "ICFA SCIC network monitoring report." `http://www.slac.stanford.edu/xorg/icfa/icfa-net-paper-jan09/report-jan09.doc`, 2009.

[Cla94] K.C. Claffy. *Internet traffic characterization*. PhD thesis, University of California at San Diego, 1994.

[CS08] K. Church and B. Smyth. "Understanding mobile information needs." In *MobileHCI*, 2008.

[CTZ10] AR Clayton Shepard, C. Tossell, L. Zhong, and P.K. LiveLab. "Measuring Wireless Networks and Smartphone Users in the Field." *HotMetrics*, 2010.

[CYH06] K. Chang, N. Yau, M. Hansen, and D. Estrin. "SensorBase.org – A Centralized Repository to Slog Sensor Network Data." In *Technical Report*. Center for Embedded Network Sensing, 2006.

[DB00] John R. Douceur and William J. Bolosky. "Process-based regulation of low-importance processes." *SIGOPS Operating Systems Review*, **34**:26–27, April 2000.

[DKV01] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M.J. Irwin. "DRAM energy management using sofware and hardware directed power mode control." In *hpca*, 2001.

[DLR77] A.P. Dempster, N.M. Laird, and D.B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm." *Journal of the Royal Statistical Society. Series B (Methodological)*, **39**(1), 1977.

[DS10] F.R. Dogar and P. Steenkiste. "Catnap: Exploiting High Bandwidth Wireless Interfaces to Save Energy for Mobile Devices." In *MobiSys*, 2010.

[DZ11a] M. Dong and L. Zhong. "Chameleon: a color-adaptive web browser for mobile OLED displays." In *ACM MobiSys*, 2011.

[DZ11b] M. Dong and L. Zhong. "Self-constructive high-rate system energy modeling for battery-powered mobile systems." In *ACM MobiSys*, 2011.

[Esf06] B. Esfahbod. *"Preload: An Adaptive Prefetching Daemon."*. Master's thesis, University of Toronto, 2006.

[FCC07] J. Froehlich, M.Y. Chen, S. Consolvo, B. Harrison, and J.A. Landay. "MyExperience: a system for in situ tracing and capturing of user feedback on mobile phones." In *MobiSys*, 2007.

[FCE05] K. Fukuda, K. Cho, and H. Esaki. "The impact of residential broadband traffic on Japanese ISP backbones." *SIGCOMM CCR*, **35**(1), 2005.

[FGE09] H. Falaki, R. Govindan, and D. Estrin. "Smart Screen Management on Mobile Phones." *Center for Embedded Network Sensing Technical Reports 74*, 2009.

[FLM10] Hossein Falaki, Dimitrios Lymberopoulos, Ratul Mahajan, Srikanth Kandula, and Deborah Estrin. "A First Look at Traffic on Smartphone." In *IMC*, 2010.

[FMK10] Hossein Falaki, Ratul Mahajan, Srikanth Kandula, Dimitrios Lymberopoulos, Ramesh Govindan, and Deborah Estrin. "Diversity in Smartphone Usage." In *ACM MobisSys*, 2010.

[FRM02] K. Flautner, S. Reinhardt, and T. Mudge. "Automatic performance setting for dynamic voltage scaling." *Wireless networks*, **8**(5):507–520, 2002.

[FS99] Jason Flinn and M. Satyanarayanan. "Energy-aware adaptation for mobile applications." In *SOSP*, 1999.

[GCW95] K. Govil, E. Chan, and H. Wasserman. "Comparing algorithm for dynamic speed-setting of a low-power CPU." In *ACM MobiCom*, 1995.

[Geo79] B. George. "Robustness in the strategy of scientific model building." *Robustness in Statistics. Academic Press, New York*, 1979.

[GML00] D. Grunwald, C.B. Morrey III, P. Levis, M. Neufeld, and K.I. Farkas. "Policies for dynamic clock scheduling." In *SOSP*, 2000.

[HPS03] H. Huang, P. Pillai, and K.G. Shin. "Design and implementation of power-aware virtual memory." In *USENIX Annual Technical Conference*, 2003.

[HRF11] J. Hicks, N. Ramanathan, H. Falaki, B. Longstaff, K. Parameswaran, M. Monibi, D.H. Kim, J. Selsky, J. Jenkins, H. Tangmunarunkit, et al. "ohmage: An Open Mobile System for Activity and Experience Sampling." *CENS Technical Reports No. 100*, 2011.

[HRK10] J. Hicks, N. Ramanathan, D. Kim, M. Monibi, J. Selsky, M. Hansen, and D. Estrin. "Andwellness: An open mobile system for activity and experience sampling." In *Wireless Health 2010*, 2010.

[HXM10] J. Huang, Q. Xu, Z.M. Mao, M. Zhang, and P. Bahl. "Anatomizing Application Performance Differences on Smartphones." In *MobiSys*, 2010.

[IN87] G.R. Iversen and H. Norpoth. *Analysis of variance*, volume 1. Sage Publications, Inc, 1987.

[Jac88] V. Jacobson. "Congestion avoidance and control." In *ACM SIGCOMM CCR*, volume 18, pp. 314–329, 1988.

[jso] "JavaScript Object Notation." *http://www.json.org/*.

[KCC07] S. Keshav, Y. Chawathe, M. Chen, Y. Zhang, and A. Wolman. "Cell phones as a research platform." In *MobiSys Panel*, 2007.

[KE05] David Kotz and Kobby Essien. "Analysis of a Campus-wide Wireless Network." *Wireless Networks*, **11**(1–2), 2005.

[KK98] R. Kravets and P. Krishnan. "Power management techniques for mobile communication." In *ACM MobiCom*, 1998.

[KPS92] D. Kwiatkowski, P.C.B. Phillips, P. Schmidt, and Y. Shin. "Testing the null hypothesis of stationarity against the alternative of a unit root." *Journal of Econometrics*, **54**(1-3), 1992.

[KVM11] M. Kennedy, H. Venkataraman, and G.M. Muntean. "Dynamic stream control for energy efficient video streaming." In *BMSB*, 2011.

[LB78] GM Ljung and GEP Box. "On a measure of lack of fit in time series models." *Biometrika*, **65**(2), 1978.

[LDP02] K. Lahiri, S. Dey, D. Panigrahi, and A. Raghunathan. "Battery-driven system design: A new frontier in low power design." In *Asia South Pacific design automation/VLSI Design*, 2002.

[Lee06] Y. Lee. "Measured TCP Performance in CDMA 1xEV-DO Network." In *PAM*, 2006.

[LFZ00] A.R. Lebeck, X. Fan, H. Zeng, and C. Ellis. "Power aware page allocation." *ACM SIGOPS Operating Systems Review*, **34**(5):105–116, 2000.

[Mac67] J. MacQueen et al. "Some methods for classification and analysis of multivariate observations." In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, p. 14. California, USA, 1967.

[MC11] J. Manweiler and R.R. Choudhury. "Avoiding the rush hours: WiFi energy management via traffic isolation." In *ACM MobiSys*, 2011.

[MFP09] G. Maier, A. Feldmann, V. Paxson, and M. Allman. "On dominant characteristics of residential broadband internet traffic." In *IMC*, 2009.

[mob12] "Mobilize: Computational Thinking, Data for Social Awareness & Civic Engagement." `http://www.mobilizingcs.org/`, 2012.

[MSF10] G. Maier, F. Schneider, and A. Feldmann. "A First Look at Mobile Hand-Held Device Traffic." In *PAM*, 2010.

[MSZ07] K. Mattar, A. Sridharan, H. Zang, I. Matta, and A. Bestavros. "TCP over CDMA2000 networks: A cross-layer measurement study." In *PAM*, 2007.

[Neu04] Y. Neuvo. "Cellular phones as embedded systems." In *ISSCC*, pp. 32–37, 2004.

[Nie98] J. Nielsen. "Nielsens law of internet bandwidth." *Online at http://www. useit. com/alertbox/980405. html*, 1998.

[nie11] "State of the Media: Mobile Media Report Q3 2011." `http://www.nielsen.com/us/en/insights/reports-downloads/2011/state-of-the-media--mobile-media-report-q3-2011.html`, 2011.

[OK11] E.A. Oliver and S. Keshav. "An empirical approach to smartphone energy level prediction." In *Ubicomp*, 2011.

[Oli09] E. Oliver. "A survey of platforms for mobile networks research." *ACM MCCR*, **12**(4), 2009.

[PB04] D. Posada and T.R. Buckley. "Model selection and model averaging in phylogenetics: advantages of Akaike information criterion and Bayesian approaches over likelihood ratio tests." *Systematic Biology*, **53**(5):793–808, 2004.

[PHZ11] A. Pathak, Y.C. Hu, M. Zhang, P. Bahl, and Y.M. Wang. "Fine-grained power modeling for smartphones using system call tracing." In *EuroSys*, 2011.

[PLS01] J. Pouwelse, K. Langendoen, and H. Sips. "Dynamic voltage scaling on a low-power microprocessor." In *SOSP*, 2001.

[Pow95] RA Powers. "Batteries for low power electronics." *Proceedings of the IEEE*, **83**(4):687–693, 1995.

[PS01] P. Pillai and K.G. Shin. "Real-time dynamic voltage scaling for low-power embedded operating systems." In *SOSP*, 2001.

[RAC11] M. Rabbi, S. Ali, T. Choudhury, and E. Berke. "Passive and In-situ Assessment of Mental and Physical Well-being using Mobile Sensors." In *Ubicomp*, 2011.

[RLR09] J. Ryder, B. Longstaff, S. Reddy, and D. Estrin. "Ambulation: A tool for monitoring mobility patterns over time using mobile phones." In *Computational Science and Engineering, 2009. CSE'09. International Conference on*, volume 4, pp. 927–931. IEEE, 2009.

[RLW87] P.J. Rousseeuw, A.M. Leroy, and J. Wiley. *Robust regression and outlier detection*, volume 3. Wiley Online Library, 1987.

[RMM11] K.K. Rachuri, C. Mascolo, M. Musolesi, and P.J. Rentfrow. "SociableSense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing." In *ACM MobiCom*, 2011.

[RQZ07] Ahmad Rahmati, Angela Qian, and Lin Zhong. "Understanding human-battery interaction on mobile phones." In *MobileHCI*, 2007.

[RRS11] A. Roy, S.M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich. "Energy management in mobile devices with the Cinder operating system." In *EuroSys*, 2011.

[RSH08] N. Ravi, J. Scott, L. Han, and L. Iftode. "Context-aware Battery Management for Mobile Phones." In *PerCom*, 2008.

[RVR03] R. Rao, S. Vrudhula, and DN Rakhmatov. "Battery modeling for energy aware system design." *Computer*, **36**(12):77–87, 2003.

[RZ09a] A. Rahmati and L. Zhong. "Human–battery interaction on mobile phones." *Pervasive and Mobile Computing*, **5**(5), 2009.

[RZ09b] Ahmad Rahmati and Lin Zhong. "A longitudinal study of non-voice mobile phone usage by teens from an underserved urban community." Technical Report 0515-09, Rice University, 2009.

[Sch97] R.R. Schaller. "Moore's law: past, present and future." *Spectrum, IEEE*, **34**(6):52–59, 1997.

[Sha09] J. Sharkey. "Coding for Life – Battery Life, That Is." *Google IO Developer Conference*, 2009.

[SK97] M. Stemm and R.H. Katz. "Measuring and reducing energy consumption of network interfaces in hand-held devices." *IEICE Transactions on Communications*, **80**(8):1125–1131, 1997.

[SLG08] T. Sohn, K.A. Li, W.G. Griswold, and J.D. Hollan. "A diary study of mobile information needs." In *SIGCHI*, 2008.

[SNR09] A. Sharma, V. Navda, R. Ramjee, V.N. Padmanabhan, and E.M. Belding. "Cool-Tether: Energy efficient on-the-fly WiFi hot-spots using mobile phones." In *CoNEXT*, 2009.

[SRH06] P. Svoboda, F. Ricciato, E. Hasenleithner, and R. Pilz. "Composition of GPRS/UMTS traffic: Snapshots from a live network." *IPS MoMe 2006*, **4**, 2006.

[SSG09] A. Shye, B. Sholbrock, and Memic. G. "Into the Wild: Studying Real User Activity Patterns to Guide Power Optimization for Mobile Architectures." In *IEEE Micro*, 2009.

[TRK09] Ionut Trestian, Supranamaya Ranjan, Aleksandar Kuzmanovic, and Antonio Nucci. "Measuring serendipity: Connecting people, locations and interests in a mobile 3G network." In *IMC*, 2009.

[TWH01] R. Tibshirani, G. Walther, and T. Hastie. "Estimating the number of clusters in a data set via the gap statistic." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **63**(2):411–423, 2001.

[WBC08] D. Wyatt, J. Bilmes, T. Choudhury, and J.A. Kitts. "Towards the automated social analysis of situated speech data." In *Ubicomp*, 2008.

[WHS05] Carey Williamson, Emir Halepovic, Hongxia Sun, and Yujing Wu. "Characterization of CDMA2000 Cellular Data Network Traffic." In *Local Computer Networks*, 2005.

[WMB08] D. Willkomm, S. Machiraju, J. Bolot, and A. Wolisz. "Primary users in cellular networks: A large-scale measurement study." In *DySPAN*, 2008.

[WPH07] Y. Won, B.C. Park, S.C. Hong, K. Jung, H.T. Ju, and J. Hong. "Measurement Analysis of Mobile Data Networks." *PAM*, 2007.

[WWD96] M. Weiser, B. Welch, A. Demers, and S. Shenker. "Scheduling for Reduced CPU Energy." *Kluwer International Series in Engineering and Computer Science*, pp. 449–472, 1996.

[YDS02] F. Yao, A. Demers, and S. Shenker. "A scheduling model for reduced CPU energy." In *FOCS*, pp. 374–382. IEEE, 2002.

[ZBP02] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. "On the characteristics and origins of Internet flow rates." In *SIGCOMM*, 2002.

[ZEL02] H. Zeng, C.S. Ellis, A.R. Lebeck, and A. Vahdat. "ECOSystem: managing energy as a first class operating system resource." *ACM SIGPLAN Notices*, **37**(10):123–132, 2002.

[ZEL03] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. "Currentcy: A Unifying Abstraction for Expressing Energy." In *Usenix Annual Technical Conference*, 2003.

[ZWS06] L. Zhong, B. Wei, and M.J. Sinclair. "SMERT: energy-efficient design of a multimedia messaging system for mobile devices." In *DAC*, 2006.