Crowdsourcing Available Parking and Available Storage Using Smartphones

by

Anandatirtha Nandugudi Sathyaraja

June 5, 2015

A dissertation submitted to the Faculty of the Graduate School
of the University at Buffalo, State University of New York
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science and Engineering

Abstract

Smartphones have emerged as the most significant large-scale mobile platform in computing history. However, the scale of smartphone experimentation has lagged behind. Keeping pace requires new facilities that enable experimentation at a scale large enough to ensure that research discoveries translate to the ever growing network of smartphone devices.

In this dissertation, we introduce PhoneLab, a smartphone testbed that is open for public experimentation. To demonstrate the efficacy of PhoneLab, we present results from three research studies conducted on PhoneLab with the experiments that ran on participants phones.

In the first study, we present results from a usage characterization experiment that ran on 115 phones for 21 days to demonstrate the power of PhoneLab for systems research.

In the second study, we present PocketParker, a crowdsourcing system that uses smartphones to predict parking lot availability. PocketParker does not require explicit user input or additional infrastructure and can run effectively without the phone leaving the user's pocket. We consider PocketParker to be an example of a subset of crowdsourcing that we call *pocketsourcing*. Users interact with PocketParker only when looking for parking spots. PocketParker detects arrival and departure events by leveraging existing activity recognition algorithms. Detected events are used to maintain per-lot availability models allows the PocketParker server to respond to client availability queries. By estimating the number of *hidden drivers*—those not using PocketParker—we can use a small fraction of monitored drivers

to estimate arrival and departure rates and make accurate predictions. Our evaluation uses multiple data sets to determine the accuracy of each PocketParker component and the system as a whole. We show that PocketParker quickly and correctly detects parking events, and that our availability estimator is accurate and robust to the presence of hidden drivers. Finally, we deploy a prototype and use camera monitoring of several parking lots to demonstrate PocketParker's performance in the wild.

In the third and the final study, we present the PocketLocker personal cloud storage system. PocketLocker creates scalable, reliable, and performant personal storage clouds out of available space distributed across multiple personal devices. Designed to store rarely-changed files on both interactive devices with limited storage (such as smartphones) and non-interactive devices with large amounts of storage (such as storage appliances), PocketLocker differs from previous systems in not requiring that each device be able to store all available content or be configured to only view certain files. Instead, a storage orchestrator running as a cloud service distributes erasure-coded file chunks across all available devices to attempt to maximize performance and capacity and minimize energy usage at battery-powered clients while meeting configurable backup requirements. And unlike current cloud storage options, PocketLocker is free and will scale as users add devices to their personal cloud.

We motivate PocketLocker's design by analyzing two months of file access traces taken from 100 smartphones, and evaluate its performance both using trace-based simulations to explore design parameters and measurements of a prototype Android implementation to establish real-world performance. By locating file content close to where it will be accessed by mobile devices, PocketLocker provides low-latency access to large amounts of content. By exploiting mobility and user charging habits, PocketLocker can meet backup requirements without draining the smartphone's battery.

Acknowledgements

I would like to thank Professors Chunming Qiao and Geoffrey Challen for their support during my Ph.D. They have always been patient and ready to take time out of their schedule to discuss ideas and guide me to turn ideas into research studies. In addition to supporting me with my Ph.D., they have been exceptional mentors and role models.

I would also like to thank Prof. Steven Ko, member of my dissertation committee for his support and guidance during my graduate studies.

During my graduate studies I've had the privilege to work with some of the finest computer scientists in the industry through internships. I would like to thank Mo-Han Fong from Intel, Emiliano Milluzzo and Robin Chen from AT&T Labs Research, Vishnu Navda and Venkat Padhmanabhan from Microsoft Research India for providing me the opportunities to work with them.

Most days of my graduate studies were spent in the Blue Systems Research Lab. I would like to thank everyone in the lab for making it a great place to work. I have been fortunate to work along with them and collaborate research studies with them.

I would like to thank all my collaborators Ameya Sanzgiri, Anudipa Maiti, Carl Nuessle, Fatih Bulut, Guru Prasad, Junfei Wang, Mukta Puri, Prof. Murat Demirbas, Scott Haseley, Prof. Shambhu Upadhyaya, Taeyeon Ki and Prof. Tevfik Kosar. It was a pleasure working with them.

Thanks to all my friends Ameya Sanzgiri, Aditya Wagh, Andrew Hughes, Raghuram Sudhakar and Ramanujan Sheshadri for making life outside school fun.

Lastly, I thank my parents for their constant encouragement and support.

Contents

List of Figures

List of Tables

1

Introduction

Mobile devices such as smartphones and tablets have become the preferred device over traditional computing devices such as laptop when carried or when the users are mobile. Gartner predicts that the sales of tablet will surpass the sales of PCs in 2015 Gartner [2014b] and by the year 2018 more than 50 percent of users will use a tablet or smartphone first for all online activities Gartner [2014a]. Although we have seen improvements in hardware to embrace this paradigm shift, mobile system software today still uses and relies on the traditional PC software systems at its core. The full potential of these mobile devices is yet to be realized and has left the user experience of the users of these devices far from satisfactory.

One of the reasons for the lack of rapid advancement of software systems is due to the lack of research facilities available to researchers and software developers to evaluate their experiments in the real world. This is evident from the scale at which mobile systems research and experiments are conducted today. A small survey of recent MobiSys papers reveals that when smartphone evaluations use real devices, they use small numbers of phones—3, 12, or 20 Lee et al. [2012], Qian et al. [2012], Wang et al. [2012]. Other experiments use simulations driven by small, old, or synthesized data sets Isaacman et al. [2012], Nath [2012], Yan et al. [2012a]. In either case, large-scale results from real users would be more compelling.

A number of factors make evaluation of experiments in the real world challenging. Recruiting human subjects to distribute experiments is perhaps the most challenging for researchers. This is because, to distribute experiments to

a large number of mobile phones would require recruiting a large number of human subjects using the phones to which the experiments can be uploaded. This requires careful planning and investment in infrastructure to collect experiment data. In most research projects this is prohibitive both in terms of time and costs to evaluate experiments for a individual research project.

This dissertation solves the challenging problem faced by resarchers today to conduct smartphone experimentation at scale. The first part of the dissertation describes PhoneLab, a large programmable smartphone testbed enabling smartphone research at scales, previously impractical. The second part describes the design and evaluation of two mobile frameworks–PocketParker and PocketLocker.

PocketParker Nandugudi et al. [2014b] was motivated by observing and experiencing the problems faced by everybody who drives to the university. During peak hours, it is time consuming to locate vacant parking spots due to fully occupied parking lots. PocketParker is a system that predicts parking lot occupancy using smartphones. Unlike previous approaches, PocketParker requires no additional infrastructure, no vehicle modifications, and no user interaction, only the installation of a smartphone app. PocketParker runs unattended in the background and uses activity transitions to detect parking lot arrivals and departures. These are forwarded to a central server that incorporates them into per-lot availability models. This allows PocketParker to order lots accurately by the probability that they contain an available spot. In general, we consider our approach to be an example of a subset of crowdsourcing that does not require any manual user input, which we call *pocketsourcing*.

We invited the participants of PhoneLab to thes the end to end efectiveness of PocketParker. The app detected and reported users parking their cars and departing in their cars to a central server. 105 participants used the PocketParker application and generated 10 827 parking events over 45 days. To obtain ground truth, four cameras were deployed to monitor two parking lots over two weeks and hand-coded four days' worth of images to measure their true availability. The results demonstrated that PocketParker can accurately and efficiently detect parking events and use them to make accurate availability predictions. During the field trial PocketParker was able to correctly predict lot availability 94% of the time.

PocketLocker Nandugudi et al. [2014a] is a system enabling scalable, reliable, and performant personal storage clouds (PSC) using personal devices such as smartphones, tablets, laptops, desktops, and dedicated storage appliances. By combining available space on existing personal devices, personal storage clouds can achieve a capacity far greater than offered by free cloud storage services.

PocketLockers' design was motivated by analyzing one month of low-level file access traces from 100 smartphone users to better understand file access patterns on mobile devices. One of the key finding was today's users are generating and accessing far more content than can be stored directly on their mobile device, making distributed file systems which require each client to store a complete replica unusable. However, a survey that was distributed to 47 people indicates that users do have available storage on other personal devices. These results motivate PocketLocker's design.

PocketLocker is designed to store rarely-changed files, such as photos, music, and videos, and to provide access to an entire personal storage cloud from any client device. PocketLocker exploits the locality of devices within the PSC to arrange rapid transfers over local-area networks when possible, and includes several energy-saving features to reduce battery drain on battery-powered mobile clients. While PocketLocker uses direct interaction between clients, it does not attempt to address the difficulties of building a true peer-to-peer distributed storage system. Instead, a cloud service called the *orchestrator* is used to maintain a consistent namespace and ensure that backup and availability requirements are met.

PocketLocker's evaluation confirms that by locating files intelligently, PocketLocker can provide mobile users with energy-efficient low-latency access to far more content than their mobile device can store locally.

1.1 Research Contributions

PhoneLab enables smartphone research at scale that were previously impractical. Both PocketParker and PocketLocker were evaluated on PhoneLab. PocketParker used PhoneLab to collect experiment data by distributing experiments in form of an Android app via the Google Play Store. PocketLocker validated the

ability to distribute experiments requiring platform level changes on the AOSP Android platform.

In addition to demonstration of the power of PhoneLab, both PocketParker and PocketLocker have other contributions in areas of mobile crowdsourcing and personal cloud computing respectively. The research contributions of PocketParker and PocketLocker to their respective fields are listed below.

1.1.1 PocketParker

- Detect parking and leaving events from parking lot in an accurate and energy efficient manner.
- Estimation of the fraction of population participating in a crowdsourcing system.
- Estimate the occupancy of parking lots.

1.1.2 PocketLocker

- Insights into file access patterns on mobile devices.
- Enable scalable, reliable and performant personal storage clouds using personally owned devices.
- Energy efficient method to store personal data.
- Bandwidth efficient method to store personal data.

1.2 Roadmap

This chapter introduced the topics that will be described in detail in the following chapters of this dissertation. Chapter 2 describes PhoneLab testbed in detail. Chapter 3 describes the different components of PocketParker in detail. Chapter 4 describes PocketLocker personal storage cloud system in detail. Chapter 5 discusses the related work to topics described in this dissertation and Chapter 6 concludes this dissertation.

2

A Large Programmable Smartphone Testbed

2.1 Introduction

Smartphones have become the most popular computing platform. Google reports 1.3 M Android device activations per day in September, 2013 Business Insider, while IDC projects that 224 M smartphone units will ship worldwide in 2013 Q4, a 40% increase over 2012 Q4 International Data Corporation. Taken as a whole, the growing network of smartphone devices represents the largest and most pervasive distributed system in history.

Meanwhile, the scale of smartphone experimentation is not keeping pace. A small survey of MobiSys'12 papers reveals that when smartphone evaluations use real devices, they use small numbers of phones—3, 12, or 20 Lee et al. [2012], Qian et al. [2012], Wang et al. [2012]. Other experiments use simulations driven by small, old, or synthesized data sets Isaacman et al. [2012], Nath [2012], Yan et al. [2012a]. In either case, large-scale results from real users would be more compelling. While multiple factors—including recruitment, human subjects compliance, and data collection—make large-scale smartphone experimentation challenging, harnessing the growth of smartphones requires evaluating new ideas at scale.

This chapter presents PhoneLab, a large programmable smartphone testbed enabling smartphone research at scales currently impractical. PhoneLab pro-

vides access to a large and stable set of participants incentivized to participate in smartphone experimentation. By exploiting locality, PhoneLab increases the density and interaction rate between participants, facilitating the evaluation of phone-to-phone protocols and crowd-sourcing algorithms.

By utilizing the Android open-source smartphone platform, PhoneLab enables research above and below the platform interface. Researchers can distribute new interactive applications or non-interactive data loggers, but can also change core Android platform components, allowing PhoneLab to host systems experiments impossible to distribute through the Play Store.

The succeeding sections of this chapter describe the component of PhoneLab in more detail. Section 2.2 describes the testbed design and implementation, introduces our participants, and explains experimental procedures. Section 2.3 describes the Android logging framework and the visibility it provides into the Android platform.

To demonstrate that PhoneLab is powerful and usable. Section 2.4 describes a usage measurement experiment run by 115 PhoneLab participants for 21 days. Section 2.5 presents five results by analyzing the data collected from the usage experiment:

- overall battery usage (Section 2.5.1),
- opportunistic charging (Section 2.5.2),
- 3G to Wifi transitions (Section 2.5.3),
- application usage patterns (Section 2.5.4), and
- location data sharing (Section 2.5.5).

The above results confirms the power of PhoneLab and breadth of research it supports.

2.2 The PhoneLab Testbed

PhoneLab was designed to fill a gap in existing smartphone experimentation capabilities. As mentioned earlier, current experimental approaches are forced to trade off, power, scale and realism. PhoneLab achieves power by utilizing Android open-source devices and a self-signed build which allows us to update any software components; scale by amortizing recruitment overhead, management

CPU	1 GHz ARM Cortex A8
GPU	PowerVR SGX540
RAM	512 MB[1]
Storage	16 GB of NAND Flash, divided into 1 and 15 GB partitions.
Battery	1500 mAh 3.7 V Li-ion.
Display	4" 480 x 800 touch screen.
Networking	1x/3G/4G (WiMax) cellular data, 802.11 b/g/n Wifi, Bluetooth, NFC, and USB.
Sensors	GPS, accelerometer, gyroscope, proximity, magnetometer and light sensor.

[1] 128 MB is reserved for the GPU.

Table 2.1: **The Samsung Nexus S 4G smartphone.**

burden and incentive costs across multiple experiments; and realism by recruiting a diverse set of participants and limiting experimental intrusiveness. We describe the architecture PhoneLab in more detail below.

2.2.1 Overview

PhoneLab currently consists of 191 participants[1] using Sprint Nexus S 4G smartphones nex running Android 4.1.1, Jelly Bean jel. Participants receive discounted voice, data, and messaging, and are instructed to use their PhoneLab phone as their primary device.

PhoneLab experiments are either distributed through the Play Store or as platform over-the-air (OTA) updates. Participants are notified of new experiments and choose whether to participate after reviewing what information will be collected about them. PhoneLab participants are *required* to participate in experimentation but *not required* to participate in any particular experiment. They may remove experiments that they deem too intrusive or that negatively affect their device. Some experiments may run in the foreground like typical applications and require the participant interact with them. Others may run quietly

[1] We refer to people carrying PhoneLab phones and participating in experiments as PhoneLab *participants*, to differentiate them from researchers running PhoneLab experiments which we call *users*.

in the background collecting useful information.

PhoneLab users must provide human subjects review documentation, a list of log tags to capture (which we describe later in this section), and their experimental software—either a link to the Play Store or a patch against the current PhoneLab platform source. Experiments generate data through the standard Android logging interface. Log messages generated by PhoneLab experiments are captured and uploaded to a central server while the device is plugged in and charging. When experimentation completes, the user receives an archive containing every log message matching their tags generated by all participating devices.

2.2.2 Platform and Device

PhoneLab phones run the popular Google Android open-source smartphone platform (AOSP). Using an open-source platform for PhoneLab was an obvious choice for obvious and less-obvious reasons.

The obvious reason is that the AOSP allows PhoneLab users to experiment with any software component, meeting our goal of providing a powerful testbed. Modifications to Android services that provide location, access networks, and manage power can be benchmarked alongside unmodified devices. Of course, power also creates problems: faulty experiments can render phones inoperable and threaten participation. As a result, experimentation at the platform level will require additional pre-deployment testing and interaction with the PhoneLab team when compared with experiments that only distribute novel applications or collect data at the application level.

We have also found that using an open-source platform has other, less obvious benefits. First, the availability of the Android source makes PhoneLab instrumentation easier even when collecting data from the application level because it gives a visibility into hidden APIs. For example, our usage characterization experiment, described in Section 2.4, uses Java reflection to access hidden battery usage APIs.

Second, the AOSP allows us to sign the platform image used by our participants. When the same key is used to sign a software package, that application may run as the system user with root privileges. Using this feature allows us to

Affiliation			
Freshman	64	Masters	5
Sophomore	33	PhD	53
Junior	1	Faculty/Staff	29
Senior	1	None	5
Gender			
Female	51	Male	140
Age			
Under 18	12	30–34	15
18–19	74	35–39	6
20–21	12	40–49	13
22–24	22	50–59	7
25–29	29	60+	1

Table 2.2: **Demographic breakdown of 191 PhoneLab participants.** Date ranges are inclusive.

distribute and update core PhoneLab experimental management software via the Play Store while retaining the privileges necessary to collect logs and perform platform updates.

Finally, we expect that our base PhoneLab platform image will evolve to meet the needs of the research community. While we have found that Android already logs a wealth of information about platform operation, there are places where more information could be exposed or logged in a more experiment-friendly way. Controlling the platform provides the opportunity to supplement existing interfaces or add additional logging to make experimentation and data collection easier.

We have distributed Nexus S 4G smartphones to our first group of participants. The Nexus S 4G was first released by Sprint in May, 2011, and was one of the official AOSP development phones at the time PhoneLab development began. Its features are summarized in Table 2.1. While we expect to receive yearly phone upgrades and will distribute a more up-to-date device to our second group of participants, we anticipate that the prohibitive cost of the newest flagship smartphones will prevent us from ever deploying them on PhoneLab.

2.2.3 Participants

Recruiting a large number of PhoneLab participants requires effective incentives. In their first year of PhoneLab participation, voice, data and messaging are free with funding provided by the National Science Foundation (NSF). This free year of service plays a major role in our recruiting efforts. In subsequent years, participants pay a deeply discounted $45 per month rate for unlimited data and messaging through a deal negotiated with Sprint. Sprint has proved to be an ideal partner for the PhoneLab project, both helpful with testbed logistics and still willing to provide unlimited data plans to subscribers.

Because participants may leave at any time, the front-loaded cost structure of our incentives makes it most efficient to recruit participants who will be able to continue as part of PhoneLab for multiple years. While we anticipate that some of our first group of participants will leave after a single year, interviews with them will help us identify long-term participants during subsequent years. Long-term participants allow us to amortize the first free year and provide a stable group comfortable being a part of PhoneLab experimentation.

When recruiting our first batch of participants, we intended to target freshman and sophomore SUNY Buffalo (UB) students as well as incoming PhD students. The University at Buffalo has a large international graduate student community, and many of these students arrive on campus without phones or phone contracts, making them ideal multi-year PhoneLab participants. After a first round of smartphone distribution in late August and early September 2012, we also began to reach out to the professional population at SUNY Buffalo in an effort to increase the number of potential long-term participants as well as the diversity of our participant pool.

In the end, we believe that we were successful in recruiting potential long-term participants. Table 2.2 describes the demographic breakdown that we achieved. We have handed out our phones to several masters or senior students because they are involved PhoneLab research. The majority consists of potential long-term participants. Roughly half of our participants are first- and second-year undergraduates, a quarter PhD students, and a fifth faculty, staff and other professionals. However, males greatly outnumber females, and the young outnumber

the middle-aged and older, both unrepresentative features we will try and rectify in year two. For management reasons we limited participation to people with a SUNY Buffalo affiliation except for several exceptions: a local reporter, a technology writer, and an international rock star.

2.2.4 Testbed Software

PhoneLab devices are deployed with a small piece of testbed management software embedded in the Android platform image. This heartbeat service uploads periodic reports including information about device location, battery levels, and the installation status of other core PhoneLab components. This information is only used for testbed management and will never be released to researchers.

The heartbeat service is also responsible for starting the primary PhoneLab configuration and data collection software when the phone boots, which allows us to bypass an Android security feature that normally prevents services from running in the background unless started by a foreground application. In order to remain unobtrusive, our experimental management software does not have a foreground component and thus would not normally be able to start.

Experimental configuration, log collection, data upload and platform updates are performed by the PhoneLab experimental harness, which is installed and updated through the Google Play Store. By signing it to match the platform build key it runs with root privileges, necessary to collect logs from all applications and perform platform updates. Periodically, the experimental harness retrieves an XML configuration from a central PhoneLab server. The configuration specifies what background experiments to start or stop, what data to collect, which server the phone should upload data to and the policy for when to perform uploads. The PhoneLab harness also uploads status information to the server during the configuration exchange, including what versions of various harness components are installed, what experiments are running and how much data is waiting to be uploaded.

PhoneLab logging and data collection must be unintrusive. If it is not, either our participants will leave or their usage patterns will be affected. We believe that we have achieved this goal. First, measured battery usage of PhoneLab is

low. A conservative overhead estimate that includes all of the applications that run as the shared system user comes to a per-participant average of 2.4%. This should be considered a strict maximum. Our policy of only uploading while the device is plugged and charging eliminates the overhead of the most power-hungry task.

Second, we have received no major complaints about our the final version of our PhoneLab experimental harness after we instructed participants to install it. Given that participants we allowed to use their phone without our software for several months, we believe that any significant changes in phone behavior caused by our experimental harness would have been noticed.

2.2.5 Safety and Privacy

PhoneLab is different from many other computer systems testbeds, such as Emulab emu, White et al. [2002], PlanetLab pla, Peterson et al. [2003], MoteLab Werner-Allen et al. [2005], or OpenCirrus ope, Avetisyan et al. [2010]: our experiments involve real people. There are two core requirements regarding our participants. First, they should use their phone as they normally would, which motivated the design of unintrusive testbed management software. Second, and more importantly, they must feel safe and in control while part of PhoneLab.

To accomplish this, when possible, we leverage several existing safety mechanisms. First, we require an Institutional Review Board (IRB) to review each PhoneLab experiment for human subjects compliance. IRB approval or an official waiver is required before any PhoneLab any experiment can begin.

Second, we distribute experimental applications to a group of developers prior to broader release, allowing us to identify any significant problems before they reach our participants. This step is particularly important for platform experiments, which must be established as stable before being distributed.

Finally, we utilize Android's existing safety and privacy mechanisms. Participants are presented with the typical Android privacy dialog during experiment installation. Rather than building an alternate distribution channel or privacy mechanism, we felt it was sufficient and probably better to use a process participants are familiar with. After installation, if a participant discovers that an

Tag Name	Tag Count	%
ActivityManager	96 251 731	13.7
dalvikvm	92 565 828	13.1
ConnectivityService	19 195 475	2.7
ActivityThread	17 447 815	2.5
PhoneStatusBar	13 823 998	2.0
SizeAdaptiveLayout	9 857 534	1.4
wpa_supplicant	9 279 597	1.3
System.err	8 141 399	1.2
SAN_SERVICE	7 530 577	1.1
LocationManagerService	6 640 001	0.9
DexLibLoader	5 438 086	0.8
SecCamera	5 436 968	0.8
HeartbeatService	4 871 085	0.7
Beautiful Widgets(4120000)	4 692 578	0.7
AudioCache	4 447 544	0.6
k9	4 330 848	0.6
SensorActivatorService	4 177 370	0.6
ThrottleService	4 121 301	0.6
VoldCmdListener	4 014 302	0.6
WindowManager	3 948 168	0.6
AudioHardware	3 913 724	0.6

Table 2.3: **Top 20 log tags generated by Android.** PhoneLab has collected 704 216 410 log messages from 7556 different tags. Tags generated by PhoneLab tools and our usage experiment are ommitted.

experiment malfunctions or wastes power, they can uninstall it. If we notice patterns of experimental removal, we will flag the experiment and notify the researcher.

2.2.6 Bootstrapping and Management

We began advertising PhoneLab on campus via posters, flyers, Facebook, and mass emails in late July, 2012. As mentioned previously, PhoneLab phone distribution began on August 24, 2012. Most phones were distributed between August 24, 2012 and August 31, 2012. Our initial plan was to distribute 200 phones during that period, but we ran into an unexpected shortage of supplies for Nexus S 4G. Due to this reason, the last device was handed out on October

Tag Name	Tag Count	%	Description
PhoneLabSystemAnalysis-Snapshot	4 507 143	71.8	Collects battery breakdown across components and applications. Polled every 15 minutes.
ActivityManager	1 078 872	17.2	Logs application management actions.
PhoneLabSystemAnalysis-Telephony	240 882	3.8	Records phone call state and radio signal strength.
PhoneLabSystemAnalysis-BatteryChange	212 929	3.4	Logs every change to the battery level.
PhoneLabSystemAnalysis-Wifi	144 163	2.3	Logs connection state, scan information and signal strength.
LocationManagerService	45 478	0.7	Records when GPS is enabled and disabled.
PhoneLabSystemAnalysis-Location	26 588	0.4	Passively logs all location updates.
PhoneLabSystemAnalysis-Misc	20 960	0.3	Logs when the screen turns on and off.
SmsReceiverService	2686	0.0	Used to count text messages sent and received.
PhoneLabSystemAnalysis-Packages	112	0.0	Records when applications are installed and removed.

Table 2.4: **Log tag statistics for one day during our experiment.** 6 279 813 total log tags were collected.

29, 2012.

We delayed the release of our experimental harness for two months until November 8, 2012. This was done for several reasons. First, we wanted to complete the distribution of phones. Second, we wanted to complete development and testing of the harness and backend infrastructure. Third, we wanted to receive training in human subjects experimentation and prepare the materials for our first experiment. Finally, the delay allowed our participants to develop normal usage patterns before experimentation began. For this last reason, we may repeat a shorter version of this delay with subsequent new groups of PhoneLab participants.

2.2.7 Experimental Procedures

To conclude, we review PhoneLab experimentation from a researcher's perspective.

First, develop your application locally. Any information logged through the standard Android logging library can be recorded. In addition, the platform may already be logging useful information for you. Keep track of all the log tags you want PhoneLab to capture. Approach your local IRB and receive experimental approval and upload your application to the Play Store.

Second, upload your list of log tags, IRB letter, and link to your application on the Play Store through the PhoneLab website. We will contact you when we begin beta testing and again once your experiment is ready for the testbed. During beta testing you will be provided with PhoneLab log output to ensure that your experiment is running properly.

Finally, your experiment will be scheduled. Our goal is to maintain a medium-sized list of active experiments for our participants: large enough to make good use of the testbed, but small enough to ensure that each experiment is picked up by many participants. When your experiment completes, you will receive a archive with messages matching the tags you selected.

2.3 Android Instrumentation

Android's logging mechanism provides a surprisingly powerful view of the internals of platform operation. Despite instructions that state to disable logging before release, many developers either forget or ignore this advice. We have seen logs generated on PhoneLab with 7556 different log tags, indicating that information about application behavior is also available.

Here we present several useful Android logging techniques suitable for use on PhoneLab. While we do plan on extending our data collection interface to support arbitrary data generated by experiments, we believe that the logging interface will prove a popular way to recover data. Particularly because support for Android logging in Eclipse provides a seamless transition from local experiment development to PhoneLab distribution.

2.3.1 Intent Monitoring

Inter-process communication on Android occurs via *intents*, which are Android message objects. Because the Android platform uses broadcast intents to distribute useful information to applications, they provide ideal logging hooks. An experimental application can subscribe to intents that it is interested in, and log information when they arrive.

Broadcast intents are sent when applications are installed or uninstalled, the radio of Wifi signal strength changes, the phone rings and is answered, and the battery level changes. Our usage experiment described in Section 2.4 subscribes to many of these useful intents and uses them to monitor device behavior.

2.3.2 Log Snooping

Experiment-driven logging of information obtained through intents may not be sufficient to reveal all events of interest. In certain cases, however, information that cannot be obtained and logged by an experiment is already ending up in the Android logs via messages sent by another component. We noticed during testing that many system components logged useful information, and so received IRB approval to collect all logs tags generated by participants phones—not only the ones generated by our usage experiment. This also served as a useful stress test on the PhoneLab infrastructure infrastructure.

Table 2.3 lists the top 20 tags from the over 700 million log messages in our database. As the table demonstrates, many core Android services already dump data, much of it useful, to the system log. Our usage experiment also uses several of these tags to uncover information that would normally be accessible, such as the screen state transitions. Table 2.4 has more details.

To make log snooping more feasible and useful we are exploring the option of improving logging coverage within the Android platform. Our experience with our first experiment has indicated that some information is more difficult to obtain than we would prefer, and other pieces of critical information are entirely missing. For example, while the `ActivityManager` tags indicate when applications are started, use of the back button by the participant is not logged. This makes it

impossible to determine what application is currently in the foreground at a fine granularity.

With access to the platform source, we can improve the visibility of important usage information. Another benefit of this approach is that experimenters will not have to incorporate common code for logging standard Android information into their experimental applications. Instead, they will simply request the appropriate tag be added to their log archive.

2.3.3 Java Reflection

The third monitoring technique is to use Java reflection to access hidden APIs of Android. Typically, this is not a recommended practice because hidden interfaces can change any time and break the application that uses them. However, if used carefully, it can be a useful tool to collect information not available otherwise. Battery statistics is a good example because there is no public Android interface that gives access to the information. However, this information is still available through hidden APIs that Android's Settings application uses to display battery statistics. Using Java reflection, any application can access these APIs. In fact, our experiment uses this method to access battery statistics as we describe in Section 2.4.

2.4 Experiment Case Study

As a case study in PhoneLab usage, we have developed a measurement student, deployed it on 115 phones and collected data for 21 days. While less exciting than the potential experiments we discuss later, we felt that a measurement study was an ideal place to begin. It demonstrates the scale, realism, and power of PhoneLab, and it generates a broadly-useful dataset that will serve as a starting point for future experiments.[1]

[1]Access to data generated by our measurement study will be allowed with IRB approval.

2.4.1 Usage Measurement

Our experiment attempts to collect information about all salient features of smartphone usage: networking, mobility, power consumption, and application usage. Table 2.4 describes each log tag used by our experiment and what data it collects. Notice that we use a mixture of active log generation and log snooping.

2.4.2 Logging Tool

Our experiment records usage information in two ways. The first way is to take a snapshot every 15 minutes. This snapshot is intended to capture the overall state of the phone periodically. The information we capture includes the amount of battery consumed, the amount of data sent and received over 3G or Wifi, storage use, and other salient features. We have chosen the 15-minute interval in order to reduce the battery consumption of our experiment. Whenever we can, we also log broadcasts intents that we receive representing per-event information such screen lock transitions, Wifi scan results, call status, and power state changes.

Most of the information we collect is available either through the standard Android interfaces or by subscribing to system intents. The only exception is the information related to battery since there is no interface or intent that provides the information. Due to this reason, we use Java reflection to introspect the internal battery APIs. PowerTutor Zhang et al. [2010] takes a similar approach to analyze battery usage.

2.4.3 Approval, Distribution and Deployment

IRB approval was quick. We had two revisions due to our misunderstanding of the instructions, but the turnaround time for each was about a week.

We uploaded our experiment to the Play Store on November 14, 2012 and announced its available to our participants via a mass email. Within a day, 82 participants installed our experiment. After 5 days, the number grew to 115. We have only sent out the email announcement once, and this may be the reason that not every participant has joined our experiment. However, the fact that 115

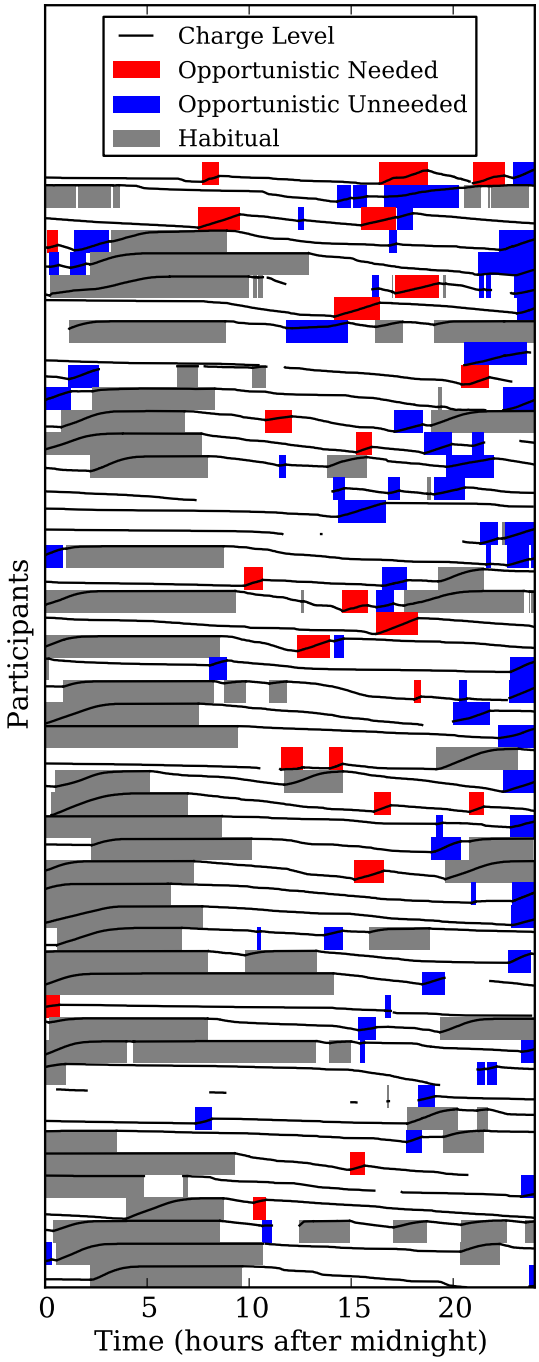


Figure 2.1: **Patterns of opportunistic charging.** Many users perform opportunistic charging multiple times during the day.

participants did elect to participate within a week after only one email indicates that our participants understand PhoneLab expectations.

Interestingly, one participant has expressed a concern about the permissions our experiment requested citing the lack of accurate information about Android permissions. The participant was particularly concerned about two permissions, "Hardware controls" that we request for collecting camera usage and "Phone calls" that we request for telephony usage. Since Android's default descriptions provide vague descriptions for these permissions such as "(Hardware control permission) record(s) audio" and "(Phone call permission) determine(s) ... the remote number connected by a call," the participant believed our experiment was doing those things. A survey study has reported a similar problem that the user comprehension level for Android permissions is remarkably low Felt et al. [2012].

We leveraged our data logging and collection mechanism by labeling different types of information with different tags. For example, we use `PhoneLabSystemAnalysis-Snapshot` for snapshots and `PhoneLabSystemAnalysis-Location` for location usage. Table 2.4 describes most of the tags that was used.

2.5 Usage Examples

This section presents five PhoneLab usage examples. Each vignette begins by highlighting an interesting or important aspect of the usage data we have collected. Our goal, however, is not to conduct an exhaustive analysis. Instead, each example continues by discussing how the presented results would guide the design of future PhoneLab experiments.

2.5.1 Overall Battery Usage

Smartphones are constrained by power, and a large amount of research on smartphone systems is motivated by energy conservation Banerjee et al. [2007], Nath [2012], Shye et al. [2009]. While power is a definite concern, evaluating the potential impact of energy saving approaches requires an accurate breakdown of where

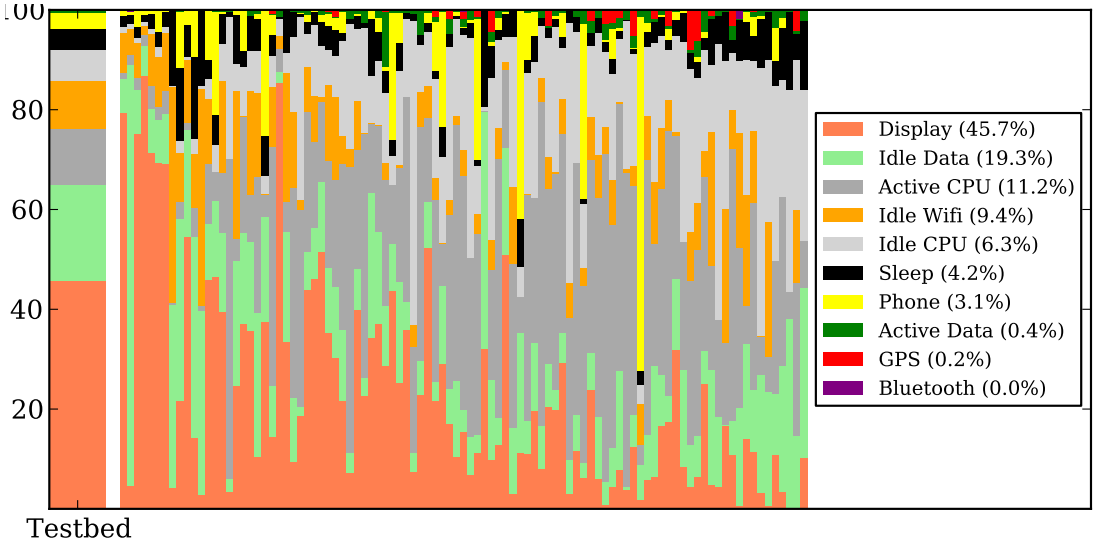


Figure 2.2: **Power usage by component.** The large bar at left shows an aggregated breakdown for the entire testbed. The participant bars are scaled against the participant with the most energy usage.

energy is used by real phones. Only then can we be sure we are addressing actual energy bottlenecks and put relative energy savings into context.

2.5.1.1 Energy Breakdown

A single-day component-by-component breakdown for the entire testbed and per-participant is shown in Figure 2.2. Our results are similar to those reported by other studies, and indicate that mobile data (labeled as "Idle data" and "Active data" depending on the state), the screen, and CPU usage are the main sources of smartphone power consumption. The per-participant bars also show a great deal of variation, with differences in both the amount and the breakdown of energy consumed by each participant.

One supposedly power-hungry component that has less of an impact than we had expected is the GPS. This is particularly surprising given the large amount of location-monitoring work motivated by GPS power consumption. One of several factors may be at work. First, the Android platform estimates the GPS chipset current consumption at 50 mA. This number is used by the standard "Fuel Gauge" battery monitor and by our calculations. However, it is lower than

the data sheet for the Broadcom 4751 GPS receiver bcm and may represent a best-case average. Still, even if the GPS current consumption is off by as much as a factor of five, it does not represent a significant contribution. Other hypotheses are that Android network location is providing location with sufficient accuracy for many applications, eliminating the need for GPS, or participants and applications may simply be conscious of GPS power consumption and taking steps to control it.

2.5.1.2 Future Experiments

While previous smaller studies on earlier Android models Shye et al. [2009] have presented similar taxonomies, the process of identifying energy bottlenecks must be repeated regularly as hardware and user behavior changes. PhoneLab provides an ideal environment for repeating energy usage experiments. Access to a stable set of participants allows us to identify changes due to participant behavior, as participants develop an awareness of the power consumption properties of their phones and how to control them. And as we bring new hardware onto the testbed, we can repeat power usage experiments to determine differences between smartphone hardware generations.

2.5.2 Opportunistic Charging

One way that users work around the battery limitations of their smartphone devices is by finding new times and places to charge their phones: plugging in at their desk at work, in the car during their commute, or at home before a long night out. We refer to these charging sessions as *opportunistic* to distinguish them from *habitual* nightly charging. Assuming that many smartphone users encounter plug points throughout the day, engaging in opportunistic charging becomes an additional sign of energy awareness, and understanding opportunistic charging becomes necessary to improving energy management on mobile devices.

2.5.2.1 Seizing Energy Opportunities

Figure 2.1 shows that many users engage in opportunistic charging. We define a charging session as opportunistic if is long enough to not be spurious (over

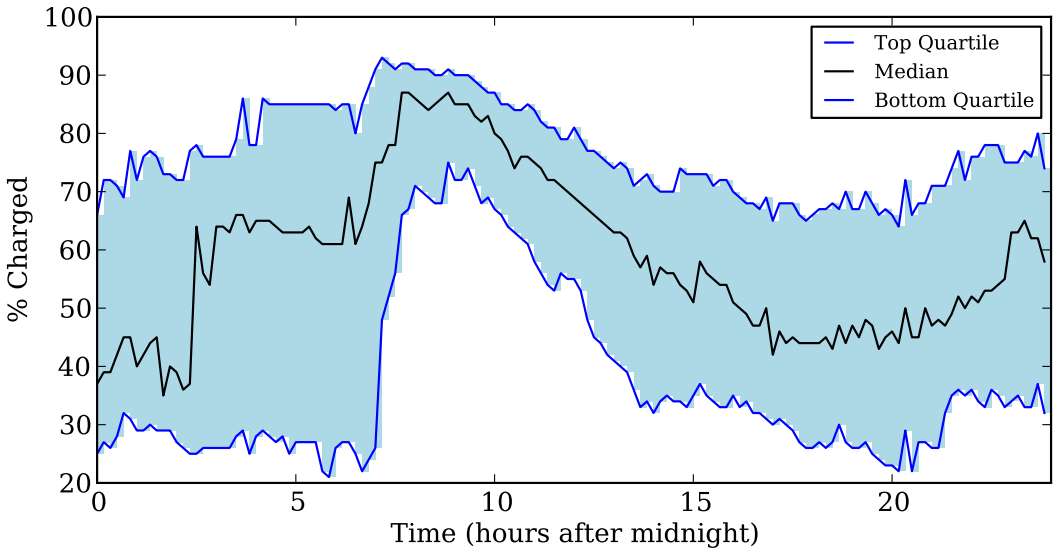


Figure 2.3: **Charge difference between participants during one day.** The graph plots the top and bottom quartiles as well as the median. A significant spread is present on the testbed throughout the day.

10 minutes) but does not bring the battery to a fully-charged state, indicating that the user disconnected the device before charging could finish. For a representative day during our experiment, of the 245 charging sessions we observed that day, 96 (39%) were opportunistic by this definition. 50 of 95 active participants engaged in opportunistic charging at some point during our experiment an average of once per day.

Opportunistic charging may be a response to an anticipated need for more smartphone battery power: the student who plugs her smartphone in for a brief charge before a night out. Our data also allowed us to examine how many of these opportunistic charging sessions were necessary to bridge the gap to the next full charge. We found that 24 of the 96 (25%) of the opportunistic charges we observed were necessary. We believe that this indicates that participants have responded to their smartphones' battery limitations by engaging in conservative charging behavior, grabbing power whenever possible even if they do not anticipate needing it later.

Combining opportunistic charging combined with the varied rhythms of our participants creates a second interesting effect: at any given point on PhoneLab

there is a wide disparity in the amount of power available on different phones. Figure 2.3 displays the top, bottom, and middle (median) quartiles for a single day on PhoneLab. Only phones that are discharging are shown, which explains the sharp increase between 6 and 10AM as participants end nightly charging cycles. As the graph indicates, there is a high chance that two smartphones that meet have very different battery levels.

2.5.2.2 Future Experiments

We are not the first to note opportunistic charging patterns Banerjee et al. [2007], Rahmati et al. [2007], but we believe PhoneLab can be used to address several interesting questions raised by opportunistic charging. First, why do users engage in this practice? By monitoring charging patterns an experiment could prompt users to indicate why they were charging their phone after opportunistic charging sessions. This would help shed more light as to the motivations of smartphone users and their evolving relationship with power.

Second, new protocols might seek to use the increased charging differentials as a result of opportunistic charging to establish a distributed energy market. Phones with spare power and users that engage in opportunistic charge frequently may agree to help another phone with less power conserve its battery level—by switching Wifi access points, providing GPS coordinates or acting as a real for cellular data—in exchange for assistance when the tables are turned. Platform experiments on PhoneLab could evaluate the effectiveness of collaborative mobile energy management on a dense testbed.

2.5.3 Mobile Network Transitions

Mobile devices like smartphones move through a complex network environment. Providing the illusion of seamless connectivity requires negotiating hand-offs both between Wifi access points and between Wifi and 3G radios. Further exacerbating the situation, users are typically very aware of the device's mistakes. It is clear to them that the phone should give up on the Wifi signal since they are halfway across the parking lot. Unfortunately, it is not so obvious to Android.

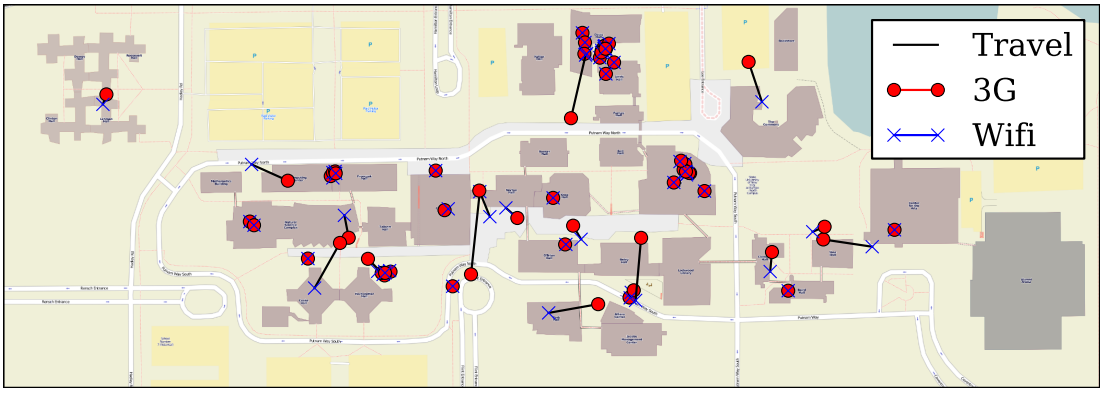


Figure 2.4: **3G to Wifi transition locations.** The map indicates that there are several common areas where network hand-offs occur.

2.5.3.1 Stuck in the Middle

We were interested in observing hand-offs between 3G and Wifi and found many in the dataset collected by our usage experiment. Since the Android `ConnectivityService` frequently switches network interfaces for exploration purposes, we have defined a transition as two one-minute or longer sessions on different interfaces separated by less than one minute. We further limit ourselves to cases where we received a location update during the transition.

Figure 2.4 plots the location of transitions that occurred on or near SUNY North Campus. We notice that many cluster in expected locations: near the entrance and exits of buildings where participants are likely to be moving from campus Wifi to Sprint 3G.

2.5.3.2 Future Experiments

Mapping and remembering where network transitions take place may allow the Android platform to conduct the transitions more smoothly. When it observes a participant heading towards a transition area, it may decide to be more aggressive about abandoning the current interface and less inclined to hang on to a weakening connection. Platform experiments using transition data generated by PhoneLab participants would use this crowd-sourced transition map to adjust the policies of the `ConnectivityService` component. Improvements in hand-offs could be benchmarked against unmodified PhoneLab phones.

Device	First App	Second App	%
7d552e	Skyvi	GO SMS	91
a991b7	Dolphin Browser Engine	Dolphin Browser	89
028e2e	Phone	Google Voice	81
2a0185	Phone	Google Voice	73
933eca	Genie Widget	Browser	68
c16dc2	Sina Weibo	MIcons Project	60
11edfb	Genie Widget	Browser	58
3cd6b1	Google Search	Browser	57
b9f595	Google Search	Browser	53
00b5ae	Phone	Google Voice	53
a991b7	GO Contacts	Phone	53
2a0185	Contacts	Google Voice	52
3ddb92	Phone	Google Voice	50

Table 2.5: **Application transitions.** The table shows the percentage starts of Second app when First app was already started on a user device.

2.5.4 Application Transitions

In any computing system, studying its workload is a key to improve the overall performance. By analyzing how applications behave, we can observe common usage patterns that arise and optimize relevant components to exploit those patterns. This is even more crucial for smartphones since they are resource constrained and still expected to run resource-heavy applications like games.

Joint use of multiple applications is potentially one interesting usage pattern. This might arise in scenarios such as checking social networking applications in series and playing different games in one sitting. If there is a group of applications with a high correlation in usage, then a phone OS might treat them together and apply the same scheduling policy, i.e., the applications can be loaded into the memory together to reduce the latency of context switch. A previous study has looked at a similar usage pattern by analyzing a network traffic trace Xu et al. [2011]. Our study described below is a more direct analysis based on recorded application usage events, hence finer in granularity.

2.5.4.1 Jointly-Used Applications

In order to understand which applications are used together, we calculate a *transition* probability from an application a to another application b within a session. A session is defined as the time between a screen unlock event and the subsequent screen lock event. To calculate a transition probability, we count the number of start events for a, i.e., *start(a)*, and the number of start events for b that occur after an a's start event within the same session, i.e., *start(a → b)*. Our transition probability is $\frac{start(a \to b)}{start(a)}$.

Table 2.5 shows some example pairs of applications with high correlations of being started together. We only show application pairs that are used more then 20 times together by a single user. The table demonstrates that for some applications there is indeed a significant correlation. For example, many users use relevant applications together such as the Phone app and Google Voice or Contacts, Google Search and Browser, etc. The fact that such high correlations exist points to the possibility of jointly scheduling these applications as a unit.

2.5.4.2 Future Experiments

We expect that PhoneLab will enable many kinds of optimization techniques that adapt to different usage patterns. As our study shows and previous studies have pointed out Falaki et al. [2010], Shye et al. [2009], there is significant diversity in smartphone usage across different users. Thus, future optimization techniques will need to adapt to each user's behavior rather than relying on commonalities.

As with energy usage experiments, application usage studies need to be revalidated; as new software and hardware become available, user behavior will inevitably change over time. PhoneLab gives an opportunity to study such changes.

2.5.5 Location Sharing

Location tracking has been the focus of many recent mobile systems research efforts Kim et al. [2010], Kjaergaard et al. [2011], Paek et al. [2010]. Given PhoneLab's density it provides an ideal proving ground for new location techniques.

Gap	GPS		Network	
(min)	Hits	%	Hits	%
5	4742	1.4	71 444	10.3
10	5486	1.6	79 877	20.5
15	6064	1.8	85 091	21.9
20	6450	1.9	88 990	22.9
Total	340 084		388 800	

Table 2.6: **Coordinate sharing counts.** We discovered few opportunities to reduce GPS usage through coordinate sharing.

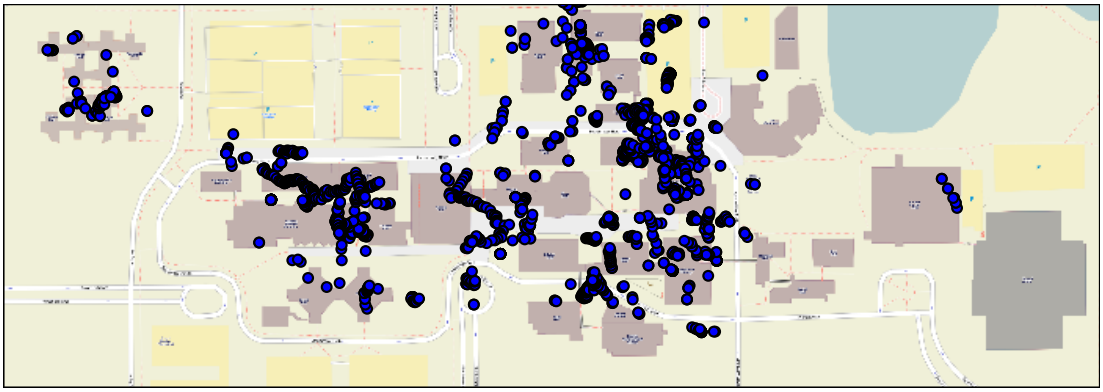


Figure 2.5: **Location of GPS sharing opportunities.**

One promising idea is to reduce GPS usage through local coordinate sharing. Before taking a reading, a smartphone will use a local communication protocol to determine whether a device nearby has recently obtained a GPS reading. If so, and if that reading is sufficiently recent and accurate, the device may not have to turn on its GPS at all, reducing latency and saving power. As a more complex variant, multiple readings from different nearby devices at different times could be combined to produce a new, more accurate reading.

Previous work Paek et al. [2010] has included the more limited form of GPS coordinate sharing into their location management system. However, given that the evaluation was done using five phones traveling in a backpack together, it is likely that their experiment evaluated nearly the best case for this approach. Given that PhoneLab provides access to a dense set of participants, it would seem a good fit for determining whether GPS coordinate sharing has merit.

2.5.5.1 Can Smartphones Share?

To answer this question we process the data from our usage experiment. In order for GPS coordinate sharing to take place, several conditions must be true. First, the two phones must be nearby in time and space. We use the location updates logged by our experiment to determine this. Note that we assume that the participant remained at the place where they acquired the location information for the time necessary to participate in sharing. While this assumption clearly does not hold in any case, mobility would not necessarily bias our result in either direction. For every false positive, a participant that did not remain where they obtained the last location fix, there may be a corresponding false negative: a participant that moved and ended up nearby another participant without our knowledge.

The second requirement is that the first phone acquire a location with sufficient accuracy to satisfy the second device. We use the accuracy estimation provided by the Android location manager to determine this.

Table 2.6 summarizes our negative result: we find few opportunities for GPS sharing on our testbed. The table is interpreted as follows: of the 3 400 084 total GPS updates performed during the three week study, only 4742 could have been satisfied with a less than five minute old coordinate from a nearby device of equal accuracy. Figure 2.5 shows the location of coordinate sharing opportunities on campus. Many cluster around areas where students travel.

Obviously the effectiveness of collaborative protocols increases as more phones participate in them, meaning that this result may be interpreted as indicating that GPS coordinate sharing would require higher densities to be effective. However, as a point of comparison we present numbers for network coordinate sharing in Table 2.6. The interpretation of the data is the same except in this case the phone is using a network provider instead of GPS. Given that the power overhead for obtaining a network location is likely to be similar to retrieving one from a nearby phone, this is not necessarily a positive result, but it does help put the low GPS sharing numbers in context.

2.6 Summary

This chapter described PhoneLab, a new large-scale programmable smartphone testbed hosted by SUNY Buffalo. PhoneLab provides a mixture of features designed to enable the next generation of mobile systems research. The five experiments based on data collected by a preliminary usage study demonstrates that PhoneLab is useful and powerful. The next chapter presents PocketParker, the first research experiment deployed on PhoneLab.

3

PocketParker: Pocketsourcing Parking Lot Availability

3.1 Introduction

Parking lots present a difficult search problem. Drivers lack the visibility to determine where spots are available, and may spend a non-trivial amount of time searching for a spot. The problem is difficult enough that WikiHow includes directions wik [2012], and the Wall Street Journal has published an online article wsj [2011] with tips on spot stalking for shoppers during the holidays. Searching not only generates frustration but also wastes energy and produces harmful carbon emissions.

Online smartphone application stores such as Google Play and the App Store are teeming with apps claiming to help you find a parking spot. Although some drivers may find these applications useful, they either do not provide real-time parking lot availability or simply display publicly-available information. Several research projects have attempted to address these limitations Caliskan et al. [2007], Chen et al. [2012], Delot et al. [2009], Lu et al. [2009], Mathur et al. [2010], but include requirements rendering them impractical, such as additional infrastructure Lu et al. [2009], on-vehicle equipment Mathur et al. [2010] or vehicular networking Delot et al. [2009], Mathur et al. [2010], or onerous manual user input Chen et al. [2012]. In contrast, we believe the solution is already in your pocket.

3. POCKETPARKER: POCKETSOURCING PARKING LOT AVAILABILITY

We present *PocketParker*, a system that predicts parking lot availability using smartphones. Unlike previous approaches, PocketParker requires no additional infrastructure, no vehicle modifications, and no user input, only installation on a small percentage of the 100 million smartphones already in use in the US sma [2011]. PocketParker runs unattended in the background and uses the accelerometer to detect parking lot arrivals and departures. These are forwarded to a central server, which incorporates them into per-lot availability models. This allows PocketParker to order lots accurately by the probability that they contain an available spot. In general, we consider our approach to be an example of a subset of crowdsourcing that does not require any manual user input, which we call *pocketsourcing*.

Providing parking availability predictions requires efficiently and accurately detecting parking-related events, and incorporating the effect of *hidden drivers*—those not using PocketParker—into our availability model. We address the first challenge by designing a simple yet effective event detector which uses the smartphone accelerometer to efficiently detect arrival and departure events, triggering energy-hungry GPS acquisition only when necessary. We address the second challenge by designing an availability estimator that maintains a probability model for each lot continuously incorporating data from PocketParker clients. We use detected events both to estimate arrival and departure rates and to make changes in real time. Part of the key to our approach is the observation that even with limited information, there are moments when PocketParker can be certain about the availability of a parking spot in a given lot, and this certainty allows PocketParker to assist users.

We perform a careful evaluation of PocketParker using a variety of methods tailored to each system component. We evaluate our parking event detector in a controlled environment with eight volunteers participating in ten parking scenarios. We design a simulator to evaluate our parking availability estimator, which gives us the flexibility to experiment with a variety of parameters and parking lot types. Finally, we evaluate the overall effectiveness of PocketParker by deploying it with 105 smartphones used by our participants over forty five days. To obtain ground truth, we deploy four cameras that monitor two parking lots over two weeks. We inspect and hand-code four days' worth of images of

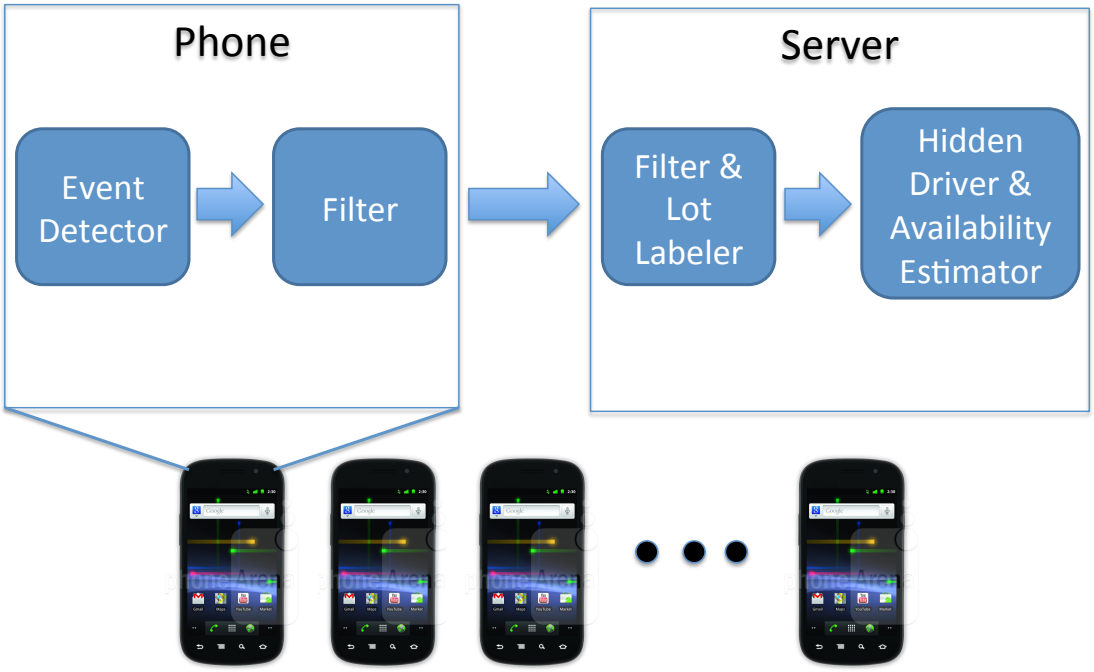


Figure 3.1: **The PocketParker architecture.** Events generated by an activity detector running in the background quietly on each smartphone are processed by a central server and used to estimate parking lot availability.

these lots to measure their true availability. Altogether, our results show the efficiency and accuracy of PocketParker.

As depicted in Figure 3.1, PocketParker has several components distributed across participating smartphones and a backend server. The rest of the chapter describes each component in detail. In Sections 3.2 and 3.3 we describe two major components of PocketParker: our parking event detector and availability model. We base our evaluation in Section 3.4 on simulations and controlled exeriments stemming from two real-world deployments. Finally, we discuss limitations and future work in Section 3.5 before concluding in Section 3.6.

3.2 Event Detector

The inputs to PocketParker's availability estimation algorithm are arrival and departure events generated by an activity detector running unattended on users' mobile devices. While considerable previous research has explored activity detec-

tion using mobile sensing Constandache et al. [2010], Keally et al. [2011], Reddy et al. [2010], Wang et al. [2009], Yang et al. [2011], we design a custom parking event detector tailored to the goals of PocketParker. In this section, we briefly describe this detector and other portions of our system that run on the smartphone itself.

3.2.1 Parking Events

PocketParker assumes that transitions between walking and driving that occur in and adjacent to locations known to be parking lots constitute either arrival (driving to walking) or departure (walking to driving) events. We thus must be able to discern between walking and driving states of the user, and to do so fast enough to fix the the location of the parking lot in which the event took place. Detecting these states could be achieved using continuously-sampled GPS data would consume too much energy for an effective pocketsourcing solution. Rather, we rely on duty-cycled accelerometer data to classify the user behavior into one of three states: walking, driving, or idle.

The initial inference of user states yielded by accelerometer sensing is subsequently refined with GPS and WiFi sense data to yield the desired goal: detection of arrival and departure events. The mobile device reports these events, along with their locations, to the server. Before recording the event, the server verifies its location against a pre-compiled list of known parking lot locations. This final step eliminates events that are either incorrect (e.g., a user parking in a field) or unwanted (e.g., a user genuinely parking but in a loading area rather than in a parking lot).

Subsequent to the conclusion of our research, Google released an update to its closed Android binaries that contained user activity detection functionality. As this was similar to that developed for PocketParker, substituting this code would not have materially affected the detection of arrival and departure events. Both algorithms treat user state detection in terms of relative likelihoods. rec [2013] False detection could be further minimized with additional sensing, particularly GPS data, or with more computationally intense detection algorithms. We deliberately shied away from such an approach however, as the inherent nature of

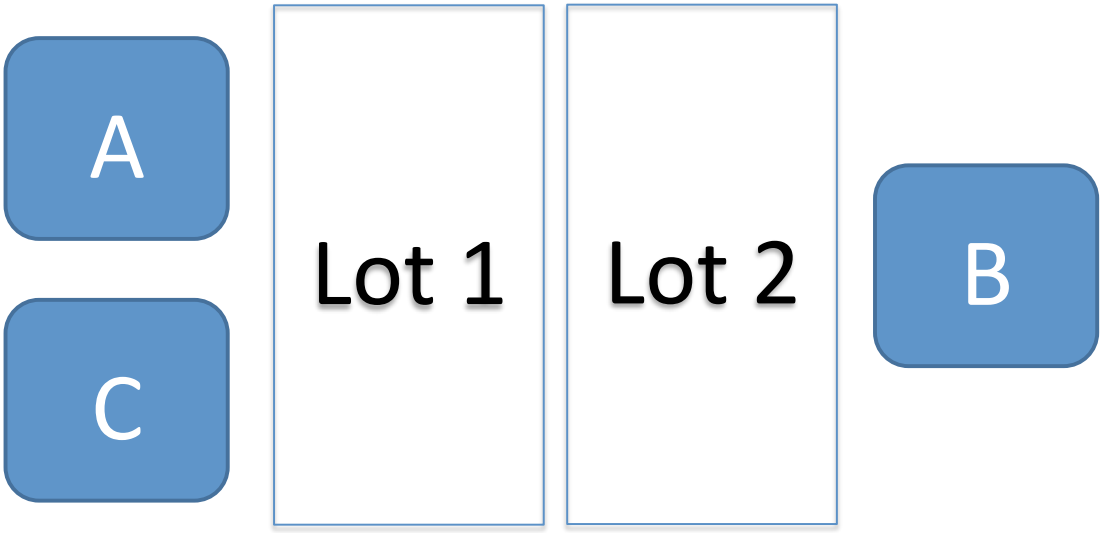


Figure 3.2: **Example parking lot setup.** Two lots and three destinations are shown.

our application dictates that the bulk of testing and filtering takes place on an energy-constrained mobile device.

3.3 Availability Estimation

In order for parking events to be useful, they must be incorporated into a model allowing us to predict parking lot availability. Our goal is to respond to queries with the probability that a given parking has a space available, information that can be used in several ways to determine what lots to search and in what order. PocketParker's estimator uses the events produced by our parking event detector both to estimate the rates at which drivers are searching and departing from the lot and to adjust the availability probability directly. In this section, we present both the design of the PocketParker client parking lot availability estimator and portions of the backend server for our system.

3.3.1 Overview

Figure 3.2 shows an example setup with two parking lots and two destinations that are used throughout this section. For each lot PocketParker maintains a

time-varying probability that the lot has n free spots $P(t, n)$. While we are mainly interested in the probability that the lot has a space available $P_{free} = \sum_{n>0} P(t, n)$, we maintain separate probabilities for each number of free spots so that we can manipulate individual probabilities in response to events and queries as described below. We bound the count probability distribution to lie between 0 and the capacity of the parking lot. Section 3.3.2 describes how PocketParker estimates lot capacity.

PocketParker's estimator receives two types of events: arrivals and departures. However, for each arrival in a given lot, a number of additional lots may have been searched unsuccessfully, information critical to the accuracy of our availability model. Section 3.3.3 describes how PocketParker determines relationships between parking lots, and Section 3.3.4 describes how we combine that information with arrivals to estimate implicit search behavior.

Between events we want to maintain our availability model by estimating the rate at which departures and searches are taking place. PocketParker must use the events it can detect to estimate the rate at which events are taking place in the lot, which includes the effect of drivers not using PocketParker, which we call *hidden drivers*. Accomplishing this requires that we estimate the ratio between monitored and hidden drivers. We describe an approach to doing so in Section 3.3.5. With an estimate of the hidden driver ratio, we can scale the search and departure rates accordingly, described in Section 3.3.6. Finally, Section 3.3.7 describes how we integrate all of this information to update our availability estimate as arrival and departure events are received.

3.3.2 Estimating Lot Capacity

PocketParker requires an estimate of lot capacity C in several places. First, we use this estimate to bound $P(t)$ such that $P(t, n > C) = 0 \; \forall \; t$. Second, we use the capacity to determine the number of hidden drivers, as detailed in Section 3.3.5. To calculate a lot capacity, we use the location of the parking lot obtained from the OpenStreetMap database ope [2013]. We derive the lot size from its location and then divide the total size by that of a typical standard parking spot lot design par [2012]. In comparing this estimate with manually counted capacities

for the five lots monitored by our deployment, capacity estimates were in all cases within 6% of their true capacities.

Errors in the capacity can result if the size of parking spots in the lot differ from our estimate, or if the parking lot is not efficiently packed with spots. Given the incentive of parking lot designers to maximize capacity, we believe that the second case will be unlikely. Parking spot sizes, however, may vary significantly from lot to lot or based on the lot's location. To improve our estimate, we may need to incorporate location-specific parking spot size estimates. Alternatively, mapping databases may be directly annotated with the number of spots per lot.

3.3.3 Lot Relationships

PocketParker's detector identifies only arrivals and departures. However, understanding and incorporating search behavior is critical to our model. For example, if we observe the arrival rate fall at a given lot, it may be because the lot is full, or it may be simply because fewer drivers are arriving and the lot still has many spaces available.

In order to estimate search behavior, we need to understand the relationships between parking lots. This requires two additional pieces of data about each lot: one or multiple destinations, and a desirability index. The destination represents the place the user is going when they park in a given lot, and note that some lots may be associated with multiple destinations. In Figure 3.2, lot 1 may be associated with destinations A, B and C; while lot 2 is only linked to B.

The desirability index produces an ordering of lots associated with a given point-of-interest based on how preferable they are compared with other lots. We assume that most users will park in desirable lots if they are available, and may have searched in more desirable lots before parking in a lot desirable lot. In Figure 3.2, if Lot 2 is associated with destination A it will probably receive a lower desirability score than Lot 1 because it is further away.

While this information is not currently part of open mapping databases, we believe that it is straightforward to collect. Parking lot operators and business owners can annotate the mapping database with destinations for each lot. In addition, data from navigation tools may be able to automatically link destinations

with lots by noting where users park after requesting directions to a particular place. The desirability index may also be determined by navigation tools observing what lots are searched by users on their way to a particular destination. Lacking these traces, simple proximity to the destination may determine the desirability index directly. As example of this automatic annotation, in Figure 3.2 if both lot 1 and 2 are associated A, we consider lot 2 less desirable because lot 1 lies between it and the destination.

3.3.4 Implicit Searches

With an understanding of lot relationships we can use observed arrivals to model implicit—or unobserved—searches. When a user parks in a given lot, we use the desirability index of the lot to add unsuccessful searches in more desirable lots associated with the some destination. There are two challenges to this approach. First, as described above, lots may be associated with multiple destinations. Second, the user may not have actually performed the search. After discussing both of these issues below, Section 3.3.7 describe below how PocketParker incorporates the information from implicit searches in a way sensitive to these uncertainties.

3.3.4.1 Determining the destination

If a lot is associated with multiple destinations, we cannot uniquely determine the destination of the user. However, this only becomes important if the two destinations would produce different desirability rankings for affected lots. For example, in Figure 3.2, if lots 1 and 2 are both associated with destinations A and C, but not with B, then an arrival with an unknown destination into lot 2 can always be used to generate an implicit search in lot 1, since the desirability ranking for the two lots are unchanged if the destination is either A or C. However, if both lots 1 and 2 are associated with all three destinations, then an arrival detected in lot 2 becomes more ambiguous. If the user was trying to go to destination A, it may mean that lot 1 was searched and is full; however, if they were trying to go to destination B, it may not indicate anything about lot 1.

If lot destination annotations are generated by mapping software, we can use this data to estimate the probability that a user is going to each of the destinations

associated with a particular lot. Instead of generating a single implicit search in one lot, we generate multiple implicit searches in each of the lots weighted by the destination probabilities. Lacking these probabilities, we simply generate implicit searches in each destination associated with a given lot.

3.3.4.2 Speculative searches

If we do not directly observe a user searching a lot before we detect an arrival, we cannot be certain that they performed the search. If the unsearched but preferable lot was available, they may not have searched it because they preferred to choose the first available spot or enjoyed the exercise of walking farther to their destination. However, these are not the type of users we believe would benefit from or use our PocketParker application, since finding a non-optimal parking spot is fairly simple in most cases.

A more interesting case is where a user has not performed a search before parking in a less-desirable lot because they *believe* the more desirable lot to be full. Users that park regularly at the same destination usually have their own mental models for the availability of spots in certain lots, causing them to discard those lots without searching them if they believe the probability of finding a spot in the desirable lot is low. While this behavior can cause users to miss available spots, these speculative searches are useful inputs since they reflect lots users think are full.

A final corner case that PocketParker does not handle is if all of the lots for a given destination are full, and many undetected unsuccessful searches are taking place. On one hand, if all lots are full then the availability of spots is determined by departures, not arrivals, and so search data is useless anyway. On the other hand, we would still like to identify this situation for users that would prefer to avoid destinations where it is difficult to park. While discussing future work in Section 3.5, we point out how integrating PocketParker into existing navigation applications could address this problem by making searches explicit, rather than implicit.

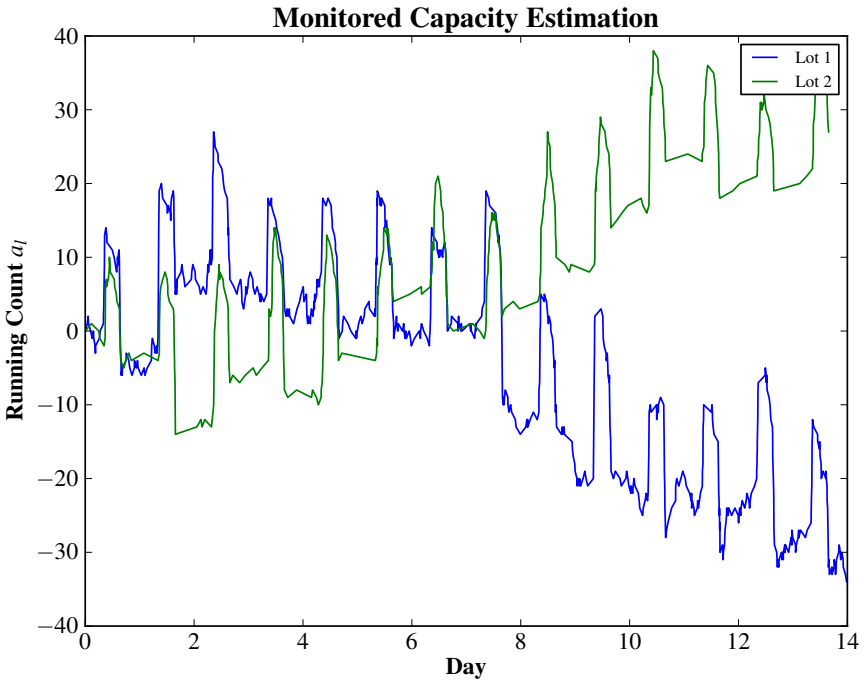


Figure 3.3: **Example of capacity estimation.** Running counts for two lots are shown.

3.3.5 Hidden Driver Estimation

Monitored PocketParker users compete for parking spaces with unmonitored users, which we call *hidden drivers*. While we assume that PocketParker users are generally representative of the entire driving population, we do not assume that all or even a large fraction of drivers will download and install PocketParker. We want our system still to provide accurate predictions with the limited information caused by hidden drivers. To accomplish this, PocketParker needs to estimate the percentage of drivers that are monitored, which we call the *monitored fraction* f_m. A low monitored fraction indicates that few users are using PocketParker, whereas a high monitored fraction means most are. Put another way, the amount of uncertainty PocketParker faces when predicting availability is inversely-proportional to the monitored fraction.

3.3.5.1 Importance of monitored fraction estimation

Two examples will illustrate why we need this information and how it is used. First, when a monitored driver leaves a parking lot, the monitored fraction determines how long PocketParker will predict that a spot in that lot is available. As the monitored fraction increases, the probability of PocketParker seeing the arrival into the lot that occupies that spot increases, and we can increase the amount of time that we estimate a spot is available. On the other hand, as the monitored fraction decreases we see fewer arrivals and are faced with more uncertainty. Hence, PocketParker reduces the amount of time it predicts the spot is available. In Section 3.3.7 we describe how hidden drivers influence the changes to the availability model made when arrivals and departures are detected.

Second, PocketParker uses the arrival and departure rates of monitored drivers to estimate changes to parking lot availability over time. Here we must scale the observed number of events to the actual number of events, which requires an estimate of the monitored fraction. Section 3.3.6 describes how the monitored rate is used for rate estimation and scaling.

3.3.5.2 Estimating the monitored fraction

PocketParker estimates the monitored fraction by first determining the monitored capacity—the capacity of the lot measured by monitored drivers—and then using our estimate of the lot capacity discussed in Section 3.3.2. Specifically, given a lot with capacity C, the monitored fraction can be estimated as $f_m = \frac{C_m}{C}$. Our task then becomes estimating the monitored capacity C_m. To estimate the monitored capacity we maintain a running count a for each lot, decremented when drivers arrive and incremented when they leave. We can consider a as a estimate of the number of spots available in the lot scaled by f_m, although we do not bound a to be below the lot capacity or greater than zero.

Figure 3.3 shows an example of the running count for two related lots over seven days using data generated by our lot simulator described in Section 3.4.2. Both lots have capacity 200 and the actual monitored fraction is 0.1. As the data shows, the running count experiences long-period (greater than one day) fluctuations due to events missed by our event detector and the randomness

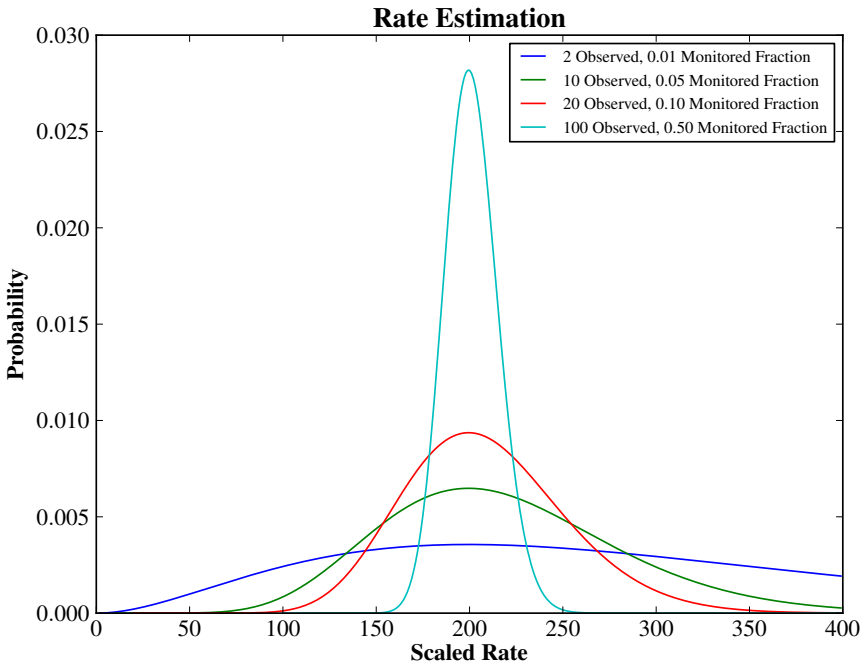


Figure 3.4: **Example of rate estimation.** Spread of each distribution shows the effect of the monitored fraction on rate certainty.

associated with the small percentage of drivers being monitored. However, the data also contains short-period (less than one day) fluctuations caused by the dynamics of the lot being monitored, and these fluctuations are roughly the size of the monitored capacity C_m, which in this case is 20 spots.

This observation motivates the design of our monitored capacity estimator. First, we bin the data into 24 hour intervals. Next, we identify the largest availability swing over each window. Finally, we average multiple swings together for a period of days to determine the final estimate. This simple approach works well on lots that fill on a regular basis. For the example in Figure 3.3, our estimator estimates the monitored capacity of lots 1 and 2 as 21.01 and 21.08, respectively, within 10% of the true value in both cases.We perform a further evaluation of our capacity estimator using multiple lot simulations in Section 4.5.

For lots that do not fill, or do not fill regularly, we may need to produce a weighted sum where larger swings are weighted more heavily given our assumption that they more accurately measure the true monitored capacity of the lot.

Another approach is to use the monitored fraction estimated at desirable lots for a given destination, which are more likely to fill completely and often, to estimate the monitored fraction for other less-desirable lots. Here we are making the reasonable assumption that lots connected to the same destination share similar fractions of PocketParker users. Finally, PocketParker's monitored fraction estimator runs periodically to incorporate changes in the monitored fraction caused by increasing use of PocketParker.

3.3.6 Rate Estimation

When PocketParker receives arrival and departure event information, it knows something concrete about the state of the lot. However, to predict availability at other times we need to adjust our estimation based on recently-observed events, which we call rate estimation. To estimate the rate of events in the entire population including hidden drivers, PocketParker must scale its rate of parking events by monitored drivers appropriately. Next, we use these scaled estimates to adjust the probability that a given lot has a certain number of spots and has spots available.

During a time interval t_0 to t_1, PocketParker will observe some number of searches $s_{obs}(t_0, t_1)$ or departures $d_{obs}(t_0, t_1)$ in any given lot[1]. Note that the search count includes both arrivals—successful searches—and implicit unsuccessful searches derived from arrivals at related lots as explained above. However, depending on the monitored fraction f_m the true count $s_{true}(t_0, t_1)$ is likely to be much larger. Rather than simply scaling the count by $\frac{1}{f_m}$, we want to determine the probability distribution over all possible true counts given the rate we observed and the estimated monitored fraction as derived in Section 3.3.5.2. One reason we do not simply scale by $\frac{1}{f_m}$ is that our uncertainty about the true count should be affected by f_m. If all drivers use PocketParker, we know the true count exactly; if few do, we should be uncertain.

[1]Without loss of generality our examples of scaling and estimating rates use notation for the search rate.

To compute the probability distribution we treat s_{obs} as the output of a binomial distribution with probability f_m and vary the number of trials. Specifically:

$$P(s_{true}|s_{obs}) = C \cdot \binom{s_{obs}}{s_{true}} f_m^{(s_{obs})} \cdot (1 - f_m)^{(s_{true} - s_{obs})} \tag{3.1}$$

where C is a renormalization constant equal to $\sum_{s_{true}} P$. Figure 3.4 shows the resulting distribution for three different values of f_m with $s_{true} = 200$. While for all four values of f_m, 200 is the most likely true count, as we desired the spread of the distribution increases with decreasing f_m.

3.3.6.1 Updating the count probabilities

Given the probability that a lot has n free spots at time t_0, $P(t_0, n)$, we want to estimate the probabilities $P(t_1, n)$ at a later time t_1. PocketParker uses recently-observed arrivals, implicit searches and departures to estimate the search s_{est} and departure d_{est} rates the lot experienced between t_0 and t_1. Currently, we use arrival and departures over a fixed-size window of time I before t_0, $s_{obs}(t_0 - I, t_0)$ scaled to the length of the interval t_0 to t_1:

$$s_{est}(t_0, t_1) = s_{obs}(t_0 - I, t_0) \cdot \frac{(t_1 - t_0)}{I}$$

The value of $s_{est}(t_0, t_1)$ is then scaled as described above to determine the distribution of s_{true}. PocketParker assumes the rates experienced over the last I time interval will continue. It may be possible to perform better rate estimation by using historical information, but this is left as future work.

The distribution of search rates $s_{true}(t_0, t_1)$ represents the probabilities that the number of available spots in the lot will decline, whereas the departure rate $d_{true}(t_0, t_1)$ represents the probability the number of spots will increase due to departures. The convolution of $-1 \cdot s_{true}$ and d_{true}, $\Delta(t_0, t_1)$, represents the change in the number of spots produced by the specific combination of arrival and departure rates. A further convolution of $\Delta(t_0, t_1)$ with $P(t_0, n)$ produces $P(t_1, n)$, the desired probability at t_1:

$$P(t_1, n) = P(t_0, n) * (-1 \cdot s_{true}(t_0, t_1) * d_{true}(t_0, t_1))$$

where $*$ represents the discrete convolution.

Note that the convolution of P with Δ can cause non-zero probabilities in P that violate our boundary conditions, namely that $P(n < 0) = 0$ and $P(n > C) = 0$ where C is the estimated capacity of the lot. To correct this, we simply set $P(n = 0) = \sum_{n<0} P(n)$ and $P(n = C) = \sum_{n>C} P(n)$, assigning all the probability that the lot has less that zero free spots to the zero state and all probability that it has more than the capacity of the lot of free spots to the empty state.

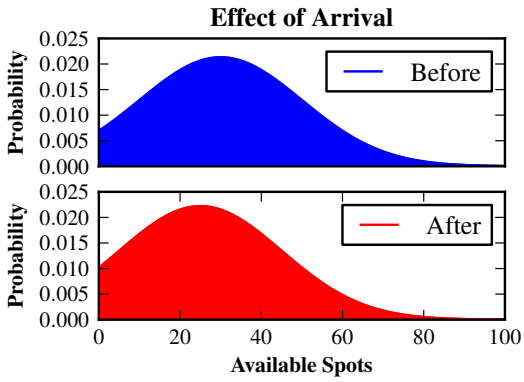
3.3.6.2 Rateless spreading

If the departure rate exceeds the arrival rate, the probability mass of Δ will lie primarily to the positive side and it will shift P in the positive direction, producing higher probabilities that more spots are available in the lot and lowering the probability that the lot is full. The opposite is true when the search rate exceeds the arrival rate.

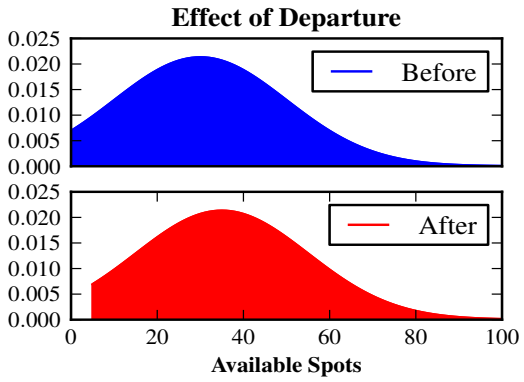
An important case is intervals during which PocketParker has observed neither arrivals nor departures in a given lot. In this case, Δ will be centered around 0 but have a spread determined by the monitored fraction. Its effect on P will be to redistribute the probability mass more evenly across the entire interval from 0 to C. Taken over many intervals, the probability of the lot having any number of spots available will equalize, which is what we would expect: after a long period without any information, all states become equally likely and we cannot make an accurate prediction of the state of the lot. Note also that the speed at which the probabilities are redistributed through rateless spreading is determined again by the monitored fraction. The fewer drivers we monitor, the more quickly we lose all memory of the state of the lot.

3.3.7 Online Updates

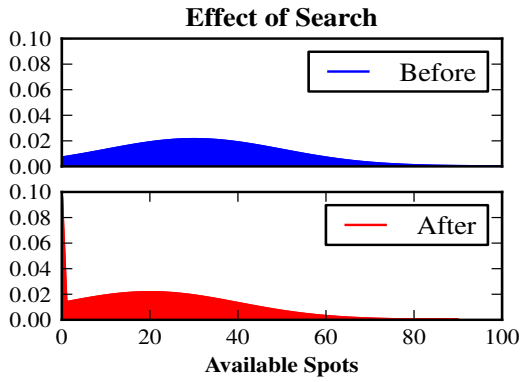
Finally, we conclude by describing how PocketParker uses arrival to adjust its availability model instantaneously at runtime. Each arrival and departure received at time t represent strong positive information—moments when PocketParker knows either that a spot just existed (arrival) or now exists (departure).



(a) Effect of arrival.



(b) Effect of departure.



(c) Effect of search.

Figure 3.5: **Effect of different types of events on the lot availability distribution.** Arrivals, departures, and implicit searches each have a different instantaneous effect on PocketParker's availability distribution.

PocketParker uses these events to adjust the probability distribution and incorporate this new information. Figure 3.5 displays an example of the effects of departures, arrivals, and searches discussed below.

Arrivals provide two somewhat conflicting pieces of information. First, PocketParker knows that at the time of the arrival there was a spot free, so in this way arrivals indicate that the lot is not full. However, PocketParker also knows that immediately after an arrival the lot has one fewer available spots. So we incorporate arrivals in two steps. First, we set $P(t, 0) = 0$ indicating the availability of a spot and renormalize the distribution. Second, we shift the entire distribution downward by one spot, $P(t, n) = P(t, n - 1)$, reflecting the loss of a parking space due to the arrival. Figure 3.5a shows an example of the effect of an arrival, including an increase in the probability that the lot has no spots.

Departures produce a straightforward change to the probability distribution. When a user departs, we know at that moment that there is a free spot in the lot, so we can set $P(t, 0) = 0$ and renormalize the distribution. Note that, since the probability that the lot is free is $P_{free} = \sum_{n>0} P(t, n)$, at the exact time of each departure the probability that a spot is free is equal to 1. Figure 3.5b shows the hole in the distribution at zero created by a departure event.

Unsuccessful implicit searches, in contrast, represent weaker negative information, both because they were not observed by PocketParker and so may not have actually taken place, or because they may not have been thorough. What we want is to increase the probability that the lot is full while reflecting our current estimate of the lot. We do this by shifting the availability distribution towards full by some amount s, which we refer to as the *search shift parameter*. So, after an implicit unsuccessful search, we set $P(t, n) = P(t, n - s)$, with $P(t, 0) = \sum_0^s P(t, n)$. Figure 3.5c shows how the distribution shifts towards zero and probability accumulates in the full state after an unsuccessful search. The search shift parameter determines how aggressively PocketParker will use information provided by implicit searches.

3.3.7.1 Weighted arrivals and departures

Shifting the distribution one space on arrivals and departures is the most conservative approach representing what we definitely know: that one spot is available.

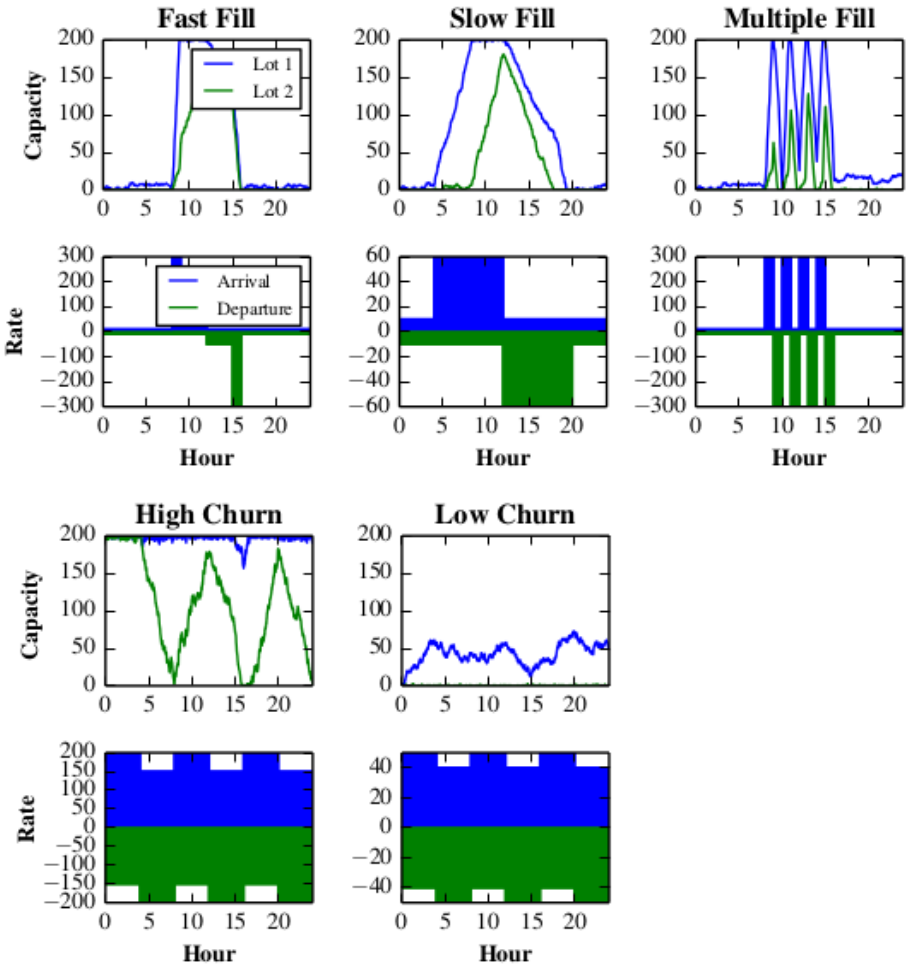


Figure 3.6: **Description of each type of lot simulated.** Five different lots with different behaviors were used during simulations.

However, if we assume that our monitored drivers are representative of some larger number of hidden drivers, we may set $P_l(t, n < X) = 0$ for some X larger than 1 and scaling with $\frac{1}{f_m}$. For our experiments we choose the conservative approach and set $X = 1$. We discuss in Section 3.5 how users may customize the behavior of PocketParker to be more or less aggressive in locating parking spots, trading off time for a better spot.

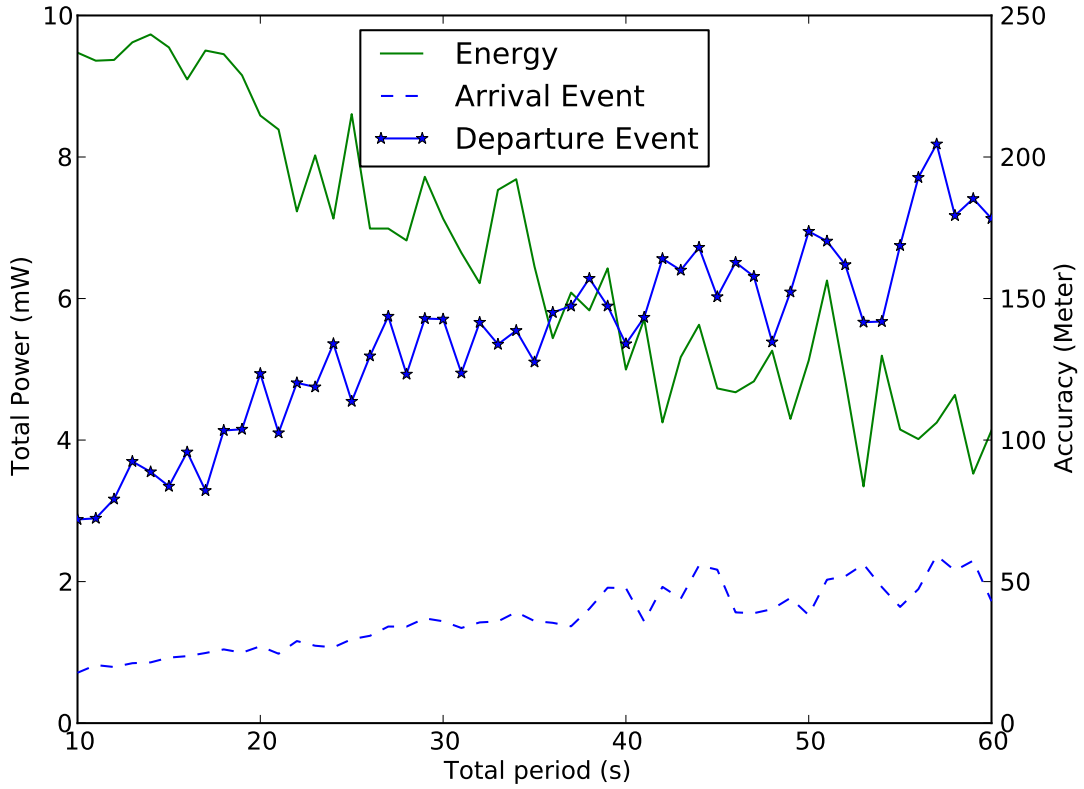


Figure 3.7: **Power usage vs. detector accuracy.** Energy usage by PocketParker is low at all duty cycles, so we chose a high duty cycle in order to improve detection accuracy.

3.4 Evaluation

We evaluated PocketParker in three ways. First, we conducted a controlled experiment to determine the best parameter settings for our event detector. Second, we implemented a parking lot simulator to experiment with various kinds of lots under differing monitored fractions. Finally, we performed a large-scale deployment of PocketParker on our campus and used it to monitor two lots. Camera monitoring was used to ground truth the predictions from our deployment dataset. Our evaluations confirm that PocketParker is efficient and accurate.

Carry Location	Count
In hand	18
Side bag	10
Back pack	10
In hand talking	7
Front pocket	14
Jacket pocket	14
Back pocket	7
Car Location	**Count**
Cup holder	16
Car seat	9
Side bag	10
Back pack	9
Front pocket	14
Jacket pocket	14
Back pocket	8

Table 3.1: **Carry and Car Location for Controlled Detector Experiment.** Eight participants generated 80 runs, carrying the phone and placing the phone in their car in many ways.

3.4.1 Detector Experiment

To determine the right parameter settings for our transition detector, we conducted a controlled experiment. During this experiment, accelerometer and GPS data was collected and stored continuously on each device, and participants were asked to manually label each transition into and out of the car. Afterwards, data was processed by a Python simulator implementing the identical algorithm used by the PocketParker application, allowing us measure accuracy and energy consumption as a function of the detector duty cycle.

Eight volunteers participated, including seven men and one woman. Seven were right-handed and one was left-handed. Each was asked to conduct the same experiment ten times: (1) carrying the instrumented phone, walk to their car; (2) label departure; (3) drive around campus briefly; (4) park and label arrival; (5) return inside. Since the way the phone is carried while walking and placed in the car while driving affects the accelerometer readings, care was taken to generate a good mix of carry and car location styles. Table 3.1 shows the breakdown. The experiment permitted us to obtain sensing data from a cross

section of individuals possessing different body morphologies, habits of driving cars, and ways of handling mobile devices.

Figure 3.7 displays the tradeoff between energy usage and detection accuracy as a function of the PocketParker duty cycle. Here we combine an active period of 5s with a inactive period of variable length, between 5 and 55s, for an overall duty cycle between 0.5 and 0.06. Our simulator uses energy numbers from the Android Fuel Gauge application o estimate average power consumption. This graph measures the accuracy of detected events in terms of distance from the actual location of the event labeled by the participant.

As we expect, longer duty cycles consume less energy but produce longer detection latencies which translate into higher distances from the event location. Note also that the departure events have higher location error than the parking events, because departing users are driving and therefore traveling more rapidly. Overall power usage by PocketParker is low, under 10 mW at all duty cycles. Because PocketParker's ability to map parking events into lots is affected by the detection distance accuracy, we chose a low total period of 15 s for a 0.25 duty cycle. This allows PocketParker to determine location to within 25 m for arrival events and 80 m for departures. Power consumption at this duty cycle is 8 mW, representing 4.2% of the capacity of a 1500 mAh battery over 24 hours of usage.

Using the same data we also examine the false positive and negative rates for arrivals and departures. This is important since, without explicit user input, it would be impossible to determine this information while PocketParker is in use. Figure 3.8 shows PocketParker can detect 80% of arrival and departure events correctly at the 0.25 duty cycle we use. False positive rates are already quite low, and this is before we apply our GPS availability filter and lot location filters. False positives decline as the duty cycle decreases because PocketParker has fewer opportunities to detect user activity.

Figure 3.9 shows that PocketParker detects parking events faithfully: 80% of users missed less than 25% parking events. The main reason is because PocketParker has transition time threshold of 5 minutes. Therefore, PocketParker requires a minimum of 5 minutes to detect an user transition event (from walking to driving or driving to walking).

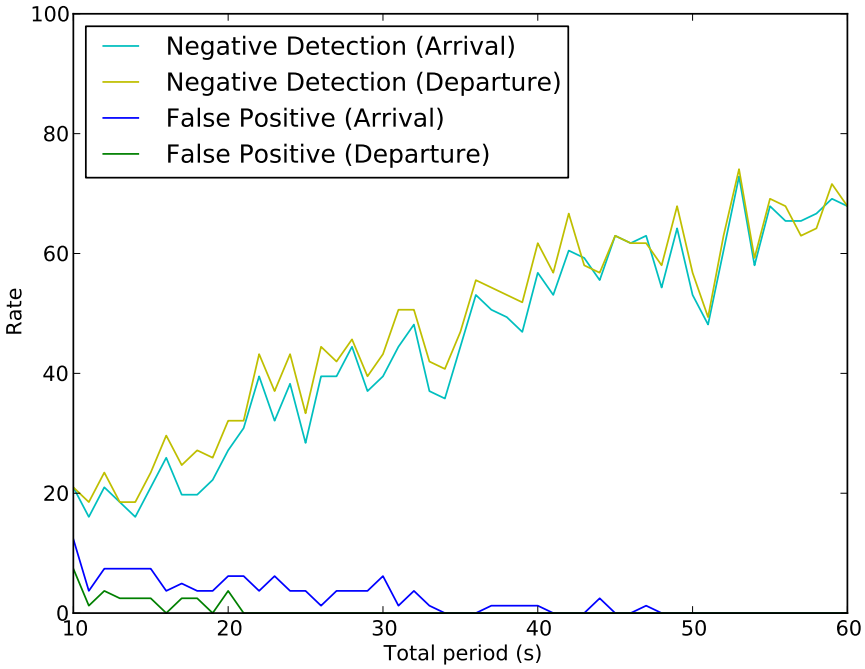


Figure 3.8: **False positive and negative rates as a function of detector duty cycle.**

3.4.2 Simulation Results

To experiment with PocketParker in a more controlled setting, we implemented a parking lot simulator in Python. Our simulator allows us to simulate any number of parking lots associated with any number of points of interest with varying desirability levels. For simplicity during our evaluation, we simulate two lots 1 and 2 with lot 1 filling before lot 2, although lot choice by simulated drivers is randomly weighted. Particularly for evaluating our monitored fraction estimation, we use five types of lots that fill and empty differently:

- **Fast Fill** and **Slow Fill** fill once per day quickly or slowly, like a lot associated with a place of work.
- **Multiple Fill** represents a lot that rapidly fills and empties repeatedly during each day, like a campus lot or movie theater.
- **High Churn** starts with lot 1 full and experiences continuously high arrival and departures rates, like an airport parking lot.

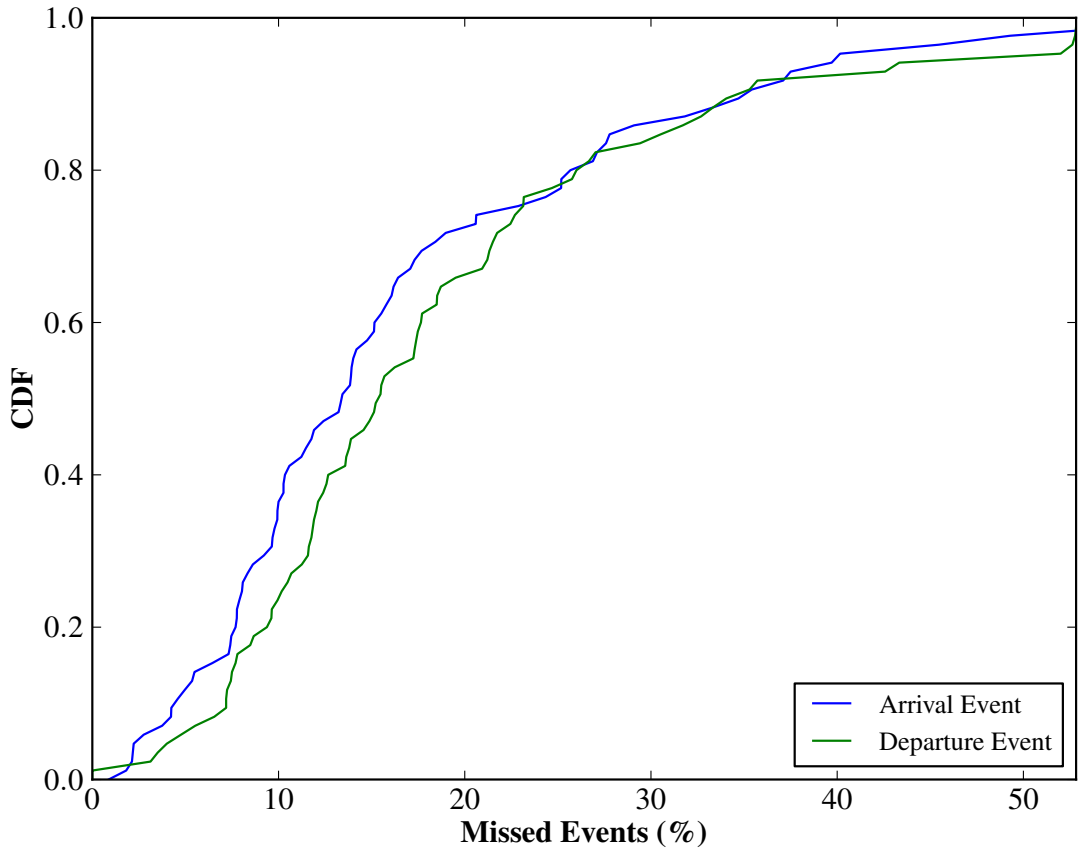


Figure 3.9: **The percentage of missed parking Events.**

- **Low Churn** represents underutilized lots that never completely fill, with lot 2 almost completely unused.

Figure 3.6 shows the arrival and departure rates for each of the types of lot as well as the resulting per-lot capacity.

3.4.2.1 Monitored fraction estimation

In Section 3.3.5.2 we describe our approach to estimated the monitored fraction, a parameter important to the operation of the PocketParker availability estimator. Figure 3.10 shows the results of 10 random simulations for each lot type. In each case, the monitored fraction estimator uses a weeks worth of data and proceeds as described previously. The error in the monitored fraction estimate is shown as a function of the actual monitored fraction for the simulation used.

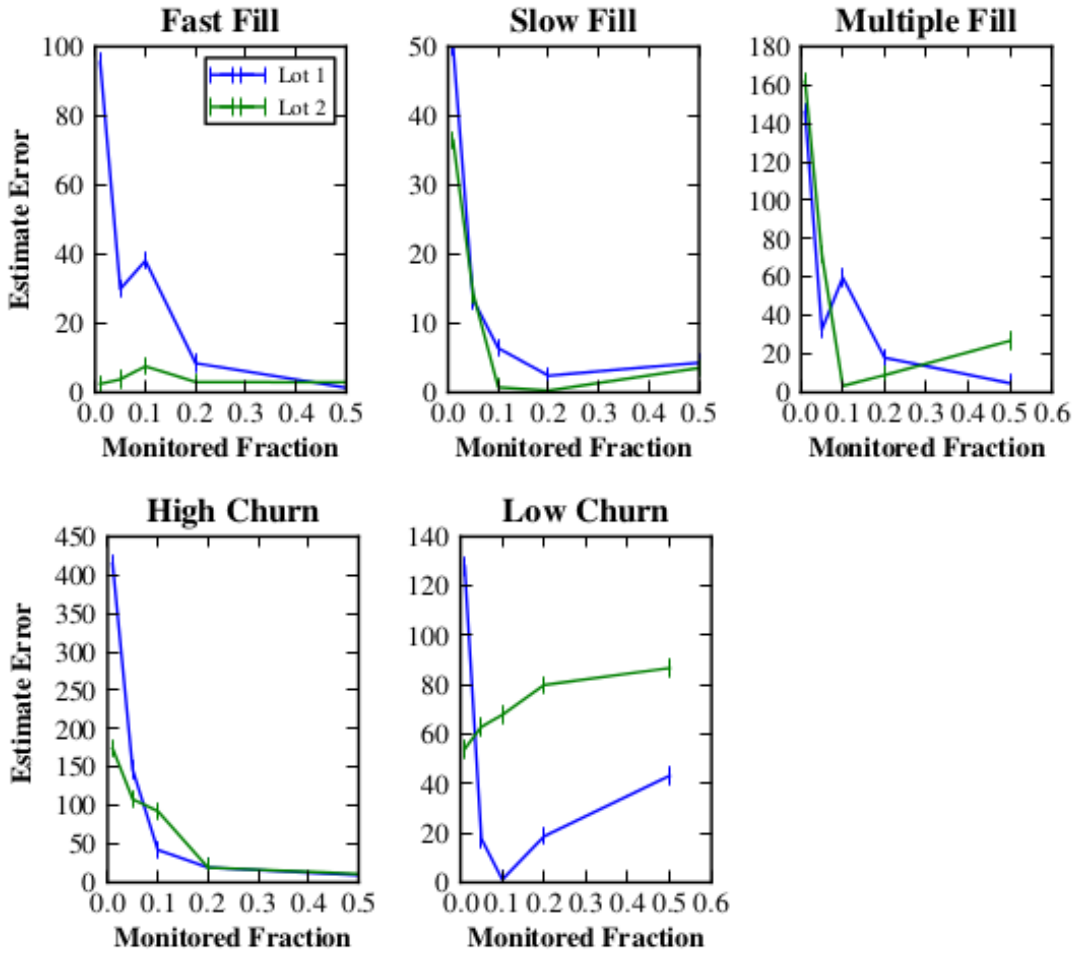


Figure 3.10: **Errors in monitored fraction estimation.** Currently PocketParker is better at estimating the monitored fraction when lots fill and empty regularly.

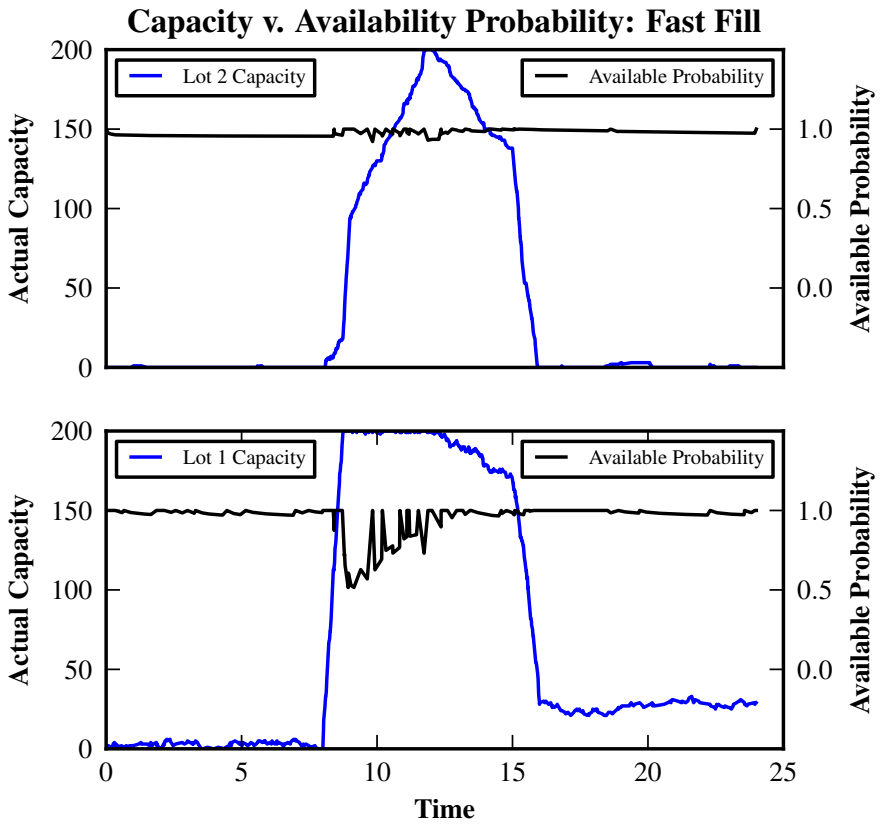


Figure 3.11: **Availability probabilities tracking lot capacity.** Dips in the availability probability correspond to times when PocketParker believes the lot is full. Discontinuities are caused by departures, which set the instantaneous probability that the lot is available to 1.0.

For the five types of lots, we would expect PocketParker to do better monitored fraction estimation when lots fill regularly—Fast Fill, Slow Fill, and Multiple Fill—and poorly when they do fill erratically or not at all—High and Low Churn. The results in Figure 3.10 generally follow this pattern. Errors for High Churn are quite high, and Low Churn errors persist even at high monitored driver fractions. This is natural, as the Low Churn lot never fills. By contrast, the accuracy rate for the Fast, Slow and Multiple Fill models improve with an increasing fraction of monitored drivers.

Type	f_m	Correct	Missed	Waste
Campus	0.07	56.1 %	43.9 %	0.0 %
	0.13	80.9 %	1.9 %	17.2 %
	0.17	72.4 %	11.0 %	16.6 %
	0.20	94.2 %	5.8 %	0.0 %

Table 3.2: **Accuracy of PocketParker predictions for various fraction of monitored drivers.**

3.4.2.2 Probability and availability

At this point we take a closer look at the way that PocketParker adjusts lot availability probabilities. Keep in mind that the absolute value of the availability probability for each lot may not meaningful. Instead, PocketParker uses the probabilities to order available lots in response to queries. We examine its accuracy at performing this essential task next. However, it is illustrative to examine the probabilities PocketParker maintains and observe how they vary as the number of available spots in the lot changes.

Figure 3.11 shows a 24 hour simulation of a Fast Fill parking lot with a monitored fraction of 0.1 and a 10% error in the estimation of the monitored fraction. The ground truth capacity of the lot as simulated is plotted next to the PocketParker probability that the lot has an available spot. At the beginning of the simulation, both lots are marked as free. When lot 1 fills and lot 2 begins to fill, generating implicit searches in lot 1, the availability probability of lot 1 drops. It spikes upward repeatedly due to departures from lot 1—which reset the short-term probability of an available spot back to 1—but does not equal the probability for lot 2 again until the point when the departure rate for lot 1 climbs.

3.4.2.3 Prediction accuracy

PocketParker exists to help drivers choose parking lots efficiently. Here, we examine the accuracy of the predicitions generated by our system. To do so, we have PocketParker rank two model lots in order of preference at regular timesteps. We compare these results with the ground truth available through a simulator and then categorize them as be in a correct prediction, a missed opportunity, or a waste of time. A missed opportunity represents a case where a more desirable

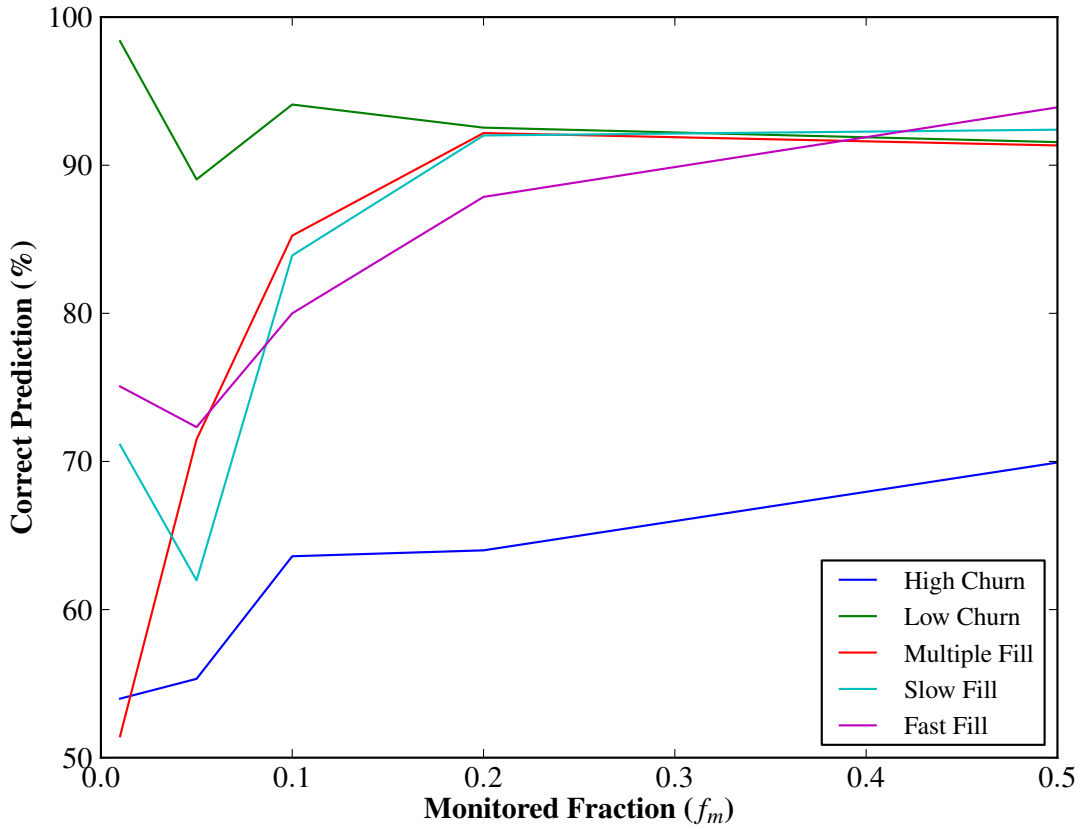


Figure 3.12: **Accuracy predictions for various kind of lots and parameters.**

lot was available than the one that PocketParker recommended. A waste of time indicates that PocketParker sent the user to a lot that did not actually have an available spot. Table 3.2 shows data results from simulations run using varying monitored fractions f_m of drivers.

Also, Figure 3.12 shows that several trends can be observed in the results. First, overall PocketParker does well on most lot types. The High Churn lot presents the greatest difficulty, which we would expect since its large number of incoming and outgoing drivers make prediction difficult. We are also concerned that the High Churn errors are largely waste of time errors, indicating that PocketParker is frequently sending drivers to the wrong lot. This is likely because it is predicting that spots are available longer than they actually are. Clearly more work is needed to determine the right approach for High Churn lots.

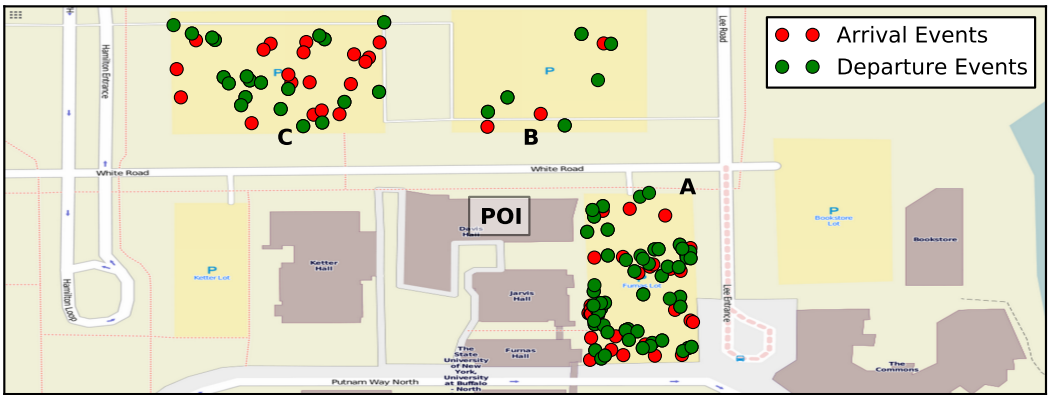


Figure 3.13: **Map showing 217 parking events detected by PocketParker during our forty-five-day deployment in three key lots.** These were generated by 26 participants. Lot A is considered the most desirable of the three lots, a fact reflected in the higher event density of this lot. Lots A and B were monitored by cameras to establish ground truth

Excluding the High Churn lot, the lot with the lowest correct percentage with a $f_m > 0.1$ is 80% for the Slow Fill lot. Accuracy for all lots above this f_m is consistently good for all lots save the High Churn model. The Low Churn lot does have a small number of errors but this is because both lots are usually empty.

One unavoidable lower bound to the accuracy of PocketParker is imposed by the frequency of parking events. This is because PocketParker has the most information about lot availability during active periods of arrival and departure events. Once it stops receiving event information, prediction uncertainty grows. Thus, to the degree that PocketParker queries follow at least pattern of arrivals and departures, we will have fresh data and do well.

3.4.3 Deployment

Finally, to establish the accuracy of PocketParker we performed a pair of deployments on our university campus. The system structure was the same in both cases: The only infrastructure required was the PocketParker server for receiving events and generating availability estimates. For the first rollout, we installed the PocketParker client on the Android version 4.1 system and the Samsung Nexus S 4G smartphone. In this experiment, we configured PocketParker

to operate in the background and to require no user interaction. We recruited five participants who generated 372 events over ten days—202 arrivals and 170 departures.

In our second rollout, we deployed PocketParker on the Android version 4.2 system and Galaxy Nexus smartphone. PocketParker, rather than running in the background, displayed to users a campus map showing recent parking events. The userbase involved 105 total participants, 102 of whom were members of PhoneLab, an existing campus mobile phone network. Over 45 days of monitoring, they generated 10,827 events – 5916 arrivals and 4911 departures – for an average of 241 per day. Our main and medical campuses produced 3645 and 846 total events respectively, with non-campus locales contributing the remaining 6336 events.

Figure 3.13 shows all of the events that occurred in three key lots that we monitored in the second experiment. Our computer science building is labeled as the point of interest (POI). To determine ground truth availability, we positioned four cameras at locations within the building to monitor lots A and B in Figure 3.13. Despite the fact that many parking events took place in lot C, we were unable to locate a suitable unobstructed vantage point to gather camera data for that lot. Nexus S 4G smartphones equipped with fish-eye lenses served as our cameras. Each took time lapse images at 1 Hz, time-stamped them using NTP and uploaded them to a central server. A total of 34,138 images were collected for the two monitored lots over two weeks.

To measure capacity, we hand-coded four days' worth of images for two lots on a ten-point scale at ten-minute intervals. We were particularly interested in the transition between empty and full states, so we were careful to ensure that the lot was never marked completely full if there was a single available spot visible. We generate 4 different dataset using parking events in camera-monitored lots A and B and then feed the events into the PocketParker estimation engine.

Table 3.2 also includes numbers for our campus deployment labeled as "Campus". Overall the accuracy of PocketParker is excellent, achieving 94.2% accuracy at a monitored driver fraction of 0.2.

3.5 Limitations and Future Work

The pocketsourcing approach taken by PocketParker makes it easy to integrate into existing mapping applications, such as Google Maps. Doing so would benefit PocketParker in two ways. First, Google Maps and other navigation tools are in extremely wide deployment, with the Play Store estimating millions of installs for Google Maps. If we can increase the monitored fraction significantly, much of the work PocketParker does to perform estimation and work around low monitored fractions will be unnecessary.

PocketParker will also benefit from the increased amount of location context available through integration into mapping software. We imagine a "Help Me Park" button which engages PocketParker. This small piece of natural user input allows PocketParker to identify *explicit* searches and use them to build up a lot desirability model with requiring annotations. However, once PocketParker begins guiding users to available parking spaces we will have to incorporate the effects of our guidance on natural user behavior. However, we believe that many users will only query PocketParker when parking in unfamiliar locations, while still providing data about lots they use regularly and know well.

PocketParker presently bases its parking predictions on a fifteen-minute limited rolling window of recent parking events. We do not presently tap the benefit of daily and weekly patterns that would otherwise enhance predictive accuracy, but hope to do so in the future. Maintaining a database of previously collected historical data from our own application will increase the sample size and hence statistical accuracy of our parking predictions. This is another area where integration with a mapping application would help, providing PocketParker with access to much more data.

Access to historical data will also address a present fundamental limitation: the situation of a lot that fills abruptly, a typical occurrence at universities during class changes. Basing a negative recommendation about a particular lot's availability solely upon recently acquired unsuccessful searches implies that a time lag necessarily exists between when users start discovering on their own that a lot is full and when we have collected enough data to conclude—somewhat belatedly—that a lot is an unwise option. Having historical data on hand will dissolve this

limitation immediately: using previous trends, we will be able to time parking advisories for particular lots before they hit capacity.

Finally, we believe that once users begin interacting with PocketParker we will see different preferences emerge. Some user will want PocketParker to help them aggressively hunt for spots, and be willing to wait for drivers to leave. Others may be more interested in simply finding a spot quickly even if it is farther away. PocketParker has several parameters that can control its predictions, and we will need to determine which are the most intuitive to users.

3.6 Summary

This chapter presented PocketParker, a pocketsourcing solution for predicting parking lot availability. PocketParker requires no explicit user input and can provide parking lot predictions without being removed from a user's pocket. PocketParker's accuracy derives from combining a simple and energy-efficient parking event detector with a sophisticated parking lot availability model that incorporates the effect of hidden drivers that compete with PocketParker users for parking spots. Our evaluation has demonstrated that PocketParker can provide accurate predictions across a variety of parking lot types and patterns, and that a fielded deployment of PocketParker performed extremely well. The next chapter describes PocketLocker, a system enabling scalable, reliable, and performant personal storage clouds from multiple personal devices.

4

The PocketLocker Personal Cloud Storage System

4.1 Introduction

As smartphones become ubiquitous, it is natural to expect to have access to all of your personal content from these powerful devices—to view all of your photos and videos; play any track from your music collection; and browse through all of your previously sent chats, texts, and emails. All of these use cases require smartphones to efficiently access far more data than they can store locally, and yet both existing cloud storage solutions Dropbox [2014], Google [2014] and recent research filesystem designs Mashtizadeh et al. [2013], Peek and Flinn [2006] require each client store a complete replica. As users assemble clouds of personal devices—including smartphones, tablets, laptops, and desktops—that collectively contain large amounts of storage, their storage capacity should not be bottlenecked by the most storage-constrained device. Particularly not if that device is their smartphone, which may have several orders of magnitude less storage than other devices. Given the cost of flash storage on mobile devices and its energy consumption, we do not anticipate mobile device storage capacity to increase at the same pace as other devices, e.g., network attached storage and laptops.

To address this personal storage bottleneck we propose to allow users to apply the same techniques used to build reliable cloud storage to create *personal storage*

clouds using their existing personal devices. By combining available space on existing personal devices, personal storage clouds can achieve a capacity far greater than offered by free cloud storage tiers. By utilizing nearby personal devices, personal storage clouds can provide better performance than cloud storage. And by applying modern approaches to reliability and availability, personal storage clouds can cope with failures and disconnections inherent to personal devices.

We present the design and implementation of PocketLocker, a system enabling scalable, reliable, and performant personal storage clouds (PSC). PocketLocker is designed to store rarely-changed files, such as photos, music, and videos, and to provide access to an entire personal storage cloud from any client device. PocketLocker exploits the locality of devices within the PSC to arrange rapid transfers over local-area networks when possible, and includes several energy-saving features to reduce battery drain on battery-powered mobile clients. While PocketLocker uses direct interaction between clients, it does not attempt to address the difficulties of building a true peer-to-peer distributed storage system. Instead, a cloud service called the *orchestrator* is used to maintain a consistent namespace and ensure that backup and availability requirements are met. Clients apply local data caching policies that reflect their usage patterns and their interaction with other clients. PocketLocker simplifies locating data within the personal storage cloud by utilizing Reed-Solomon erasure coding Reed and Solomon [1960], allowing clients to reconstruct files as long as they can acquire any set of mutually-redundant chunks.

This study makes the following contributions. First, we examine one month of low-level file access traces from 100 smartphone users to better understand file access patterns on these popular mobile devices. We conclude that today's users are generating and accessing far more content than can be stored directly on their mobile device, making distributed file systems which require each client to store a complete replica unusable. However, a survey that we distributed to 47 people indicates that users do have available storage on other personal devices. These results motivate PocketLocker's design.

Second, we present the design of PocketLocker and describe how it uses multiple personal devices to build scalable, performant, and energy-efficient personal storage clouds to store rarely-changed files such as music, videos, and photos.

PocketLocker uses erasure coding to divide files into multiple chunks which are distributed across all the users' participating devices—which can include smartphones, tablets, laptops, desktops, and dedicated storage appliances. PocketLockers orchestrator, which runs as a cloud service, distributes chunks across the users personal devices to maintain file availability, maximize performance, and meet configurable backup requirements.

Finally, we perform a detailed evaluation of PocketLocker that proceeds along two lines. First, we utilize our file access traces to examine the impact of several key PocketLocker design parameters and estimate the performance of file access on PocketLocker. Second, we measure the energy consumption and performance of an Android prototype as it accesses files under a variety of real-world conditions. Our results confirm that by locating files intelligently, PocketLocker can provide mobile users with energy-efficient low-latency access to far more content than their mobile device can store locally.

This chapter is structured as follows. Section 4.2 presents several results that motivate and inform PocketLocker's design, which we present in detail in Section 4.3. Section 4.4 briefly describes the implementation of our current PocketLocker prototype for Android smartphones, which we evaluate in Section 4.5. Section 4.6 discusses future work and concludes the chapter.

4.2 Motivation

To better understand storage usage on smartphones and the potential to expand capacity by creating personal storage clouds, we performed several measurement studies. We were interested in answering the following questions:

1. How much storage do users have available on their smartphones?
2. How frequently are media files created, modified, and accessed on smartphones?
3. How is available storage distributed across multiple personal devices?

Next we discuss our findings and how they influence PocketLocker's design.

By Gender			
Female	127	Male	122
By Age			
< 18	0	35–39	28
18–20	12	40–49	52
21–24	21	50–59	34
25–29	34	> 60	9
30–34	35		

Table 4.1: **PhoneLab demographic breakdown.**

4.2.1 Rate of Smartphone Storage Decline

To determine how rapidly users are creating content on their mobile devices, we ran an IRB-approved experiment on PhoneLab, a public smartphone testbed located at the University at Buffalo Nandugudi et al. [2013]. 288 students, faculty, and staff carry instrumented Android Samsung Galaxy Nexus smartphones and received subsidized service in return for willingness to participate in experiments. PhoneLab provides access to a representative group of smartphone users balanced between genders and drawn from a wide variety of age brackets, making our results representative of the broader smartphone user community. Table 4.1 shows a demographic breakdown of the testbed based on a survey completed by 249 of the 288 PhoneLab users.

We distributed a simple experiment that periodically logged the storage available on each smartphone which 105 PhoneLab joined for eight months, beginning shortly after PhoneLab users received new smartphones in August, 2013. Our log messages show users available storage declining by roughly 30 MB per day, approximately the size of 30 photos or a half-dozen music files. Aggregate capacity reflects both the rate at which users are creating content, but also the rate at which they may be moving content such as music on to their device. However, if this rate of decline continues it will only take the average PhoneLab participant three years to generate more content than they can fit onto their Samsung Galaxy Nexus Wikipedia with its 32 GB of Flash. And this estimate shows users coping with the existing storage limitations of their personal devices.

We expect that many already have far more photos, music, and video than will fit onto their mobile device.

4.2.2 Media Access Patterns

To obtain a more detailed picture of file access patterns for the media files that we expect users to want to access on many devices—pictures, movies, and music—we distributed a second IRB-approved experiment on PhoneLab to collect more detailed file access patterns. By instrumenting the `bionic` C library used by Android applications, we were able to log every file open performed on all participating devices. We also logged the file size each time a file was opened. Our changes were distributed as a platform over-the-air update which PhoneLab participants downloaded in November, 2013. To indicate consent, participants were also required to download and run a separate app. We collected one month of data from December, 2013, for this experiment from 100 users.

We filtered the dataset by extension to only include media files[1], which still left 1 780 617 opens of 147 756 distinct files by 100 users during the month. Figure 4.1 shows CDFs of the total number and size of the files opened by each PhoneLab user during one month, demonstrating the smartphone users access a large amount of media content from their mobile device.

We marked 151 904 media files as created if they were empty when opened, and only 11 612 files as modified by comparing their sizes reported by successive open calls. This limited us to files that were opened multiple times during the trace, but we were still able to determine modifications for 89% of the file accesses we observed. Figure 4.2 shows modifications rates for photos, video, and audio files, demonstrating that these files are rarely modified on mobile devices. PocketLocker incorporates this assumption into its design.

4.2.3 Available Storage Distribution

Finally, to investigate the potential to utilize other nearby personal devices as part of a personal storage cloud, we distributed an IRB-approved survey to un-

[1]We marked files with the follow extensions as media: `flv`, `mp4`, `3gp`, `wmv`, `avi`, `mov`, `mpg`, `mpeg`, `aac`, `jpg`, `jpeg`, `png`, `gif`, `pdf`, `bmp`, `m4a`, `mp3`, `m4v`, `3g2`, `asf`, `asx`, `swp`, `swf`, and `tif`.

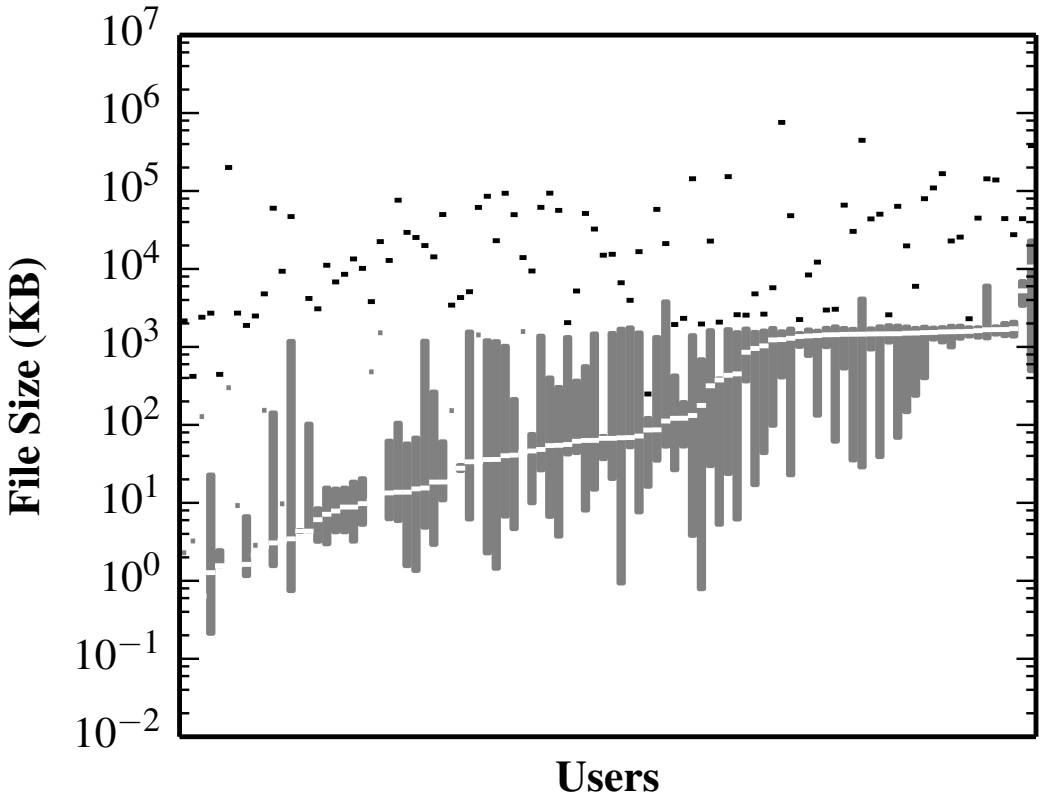


Figure 4.1: **File Sizes.** Per-user distributions are shown for all media files accessed by PhoneLab users during the one month experiment. Most files are between 10 KB and 1 MB, but some are up to 100 MB.

dergraduates at our university. For each device they owned, respondents were asked to indicate how much storage capacity it had available and, for immobile devices, where they used it most: at work or school, or at home. Table 4.2 shows results collected from 47 volunteers. Results indicate that users have large storage capacity from other devices, such as laptops and desktops, available to them at multiple locations, while the free storage available on their smartphones was one order-of-magnitude smaller than the storage available on their other devices. By utilizing storage on other personal devices, PocketLocker can address the mobile storage bottleneck.

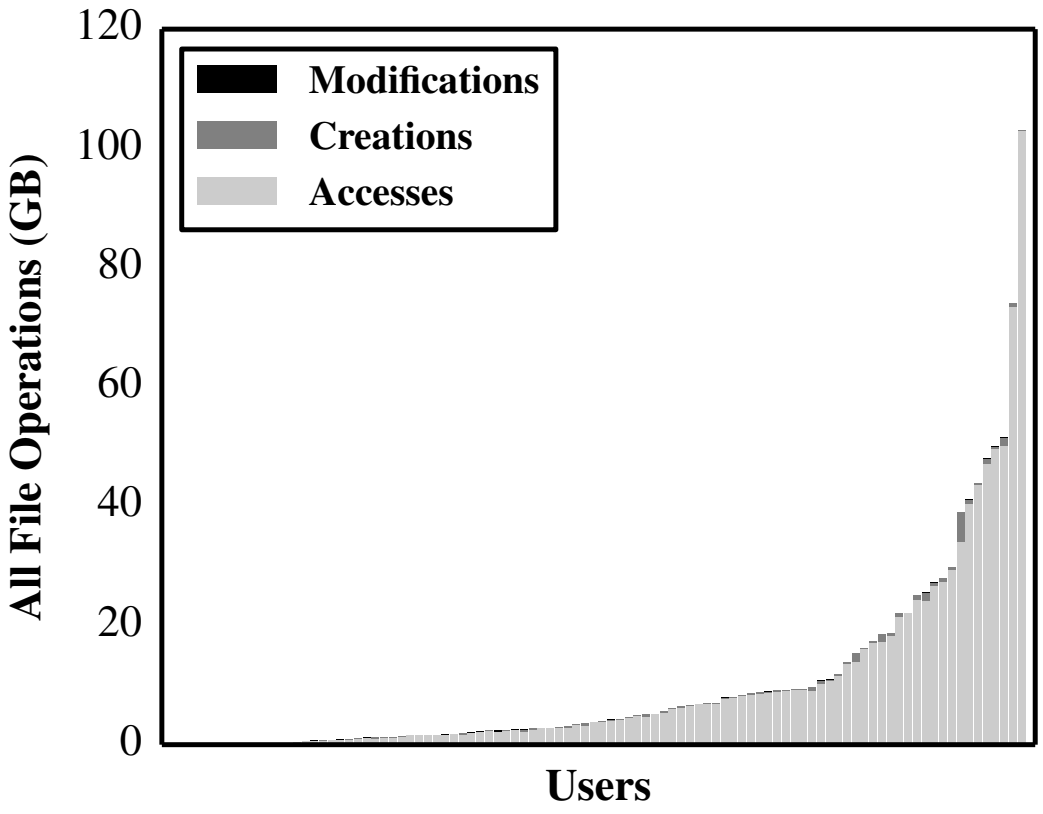


Figure 4.2: **Media files are rarely modified.** Most file operations are accesses.

4.3 Design

A PocketLocker personal storage cloud (PSC) consists of a set of clients—including smartphones, tablets, laptops, desktops, and dedicated storage appliances—and a cloud service called the *orchestrator*. Like most filesystems, PocketLocker uses a namespace to map paths to a set of n uniquely-identified *chunks* containing file data. Because chunks are the output of erasure coding, the number of distinct chunks required for reconstruction k is less than n, and k is stored in the namespace for each path.

The orchestrator maintains the authoritative namespace by using a monotonically-increasing counter to version all namespace modifications, including opens, renames, updates, and deletes. It also fixes the location of certain

Location	Min (GB)	Mean (GB)	Max (GB)
Home	8	308	3000
Work	7	97	600
Mobile	0.5	10	42
Total	15	415	3806

Table 4.2: **Storage space available at different locations.** Results from a survey of 47 people. Users have an order-of-magnitude less space available on mobile devices compared with their other personal devices.

chunks to meet backup requirements and coordinates transfers between firewalled clients during open. While the orchestrator must track the location of some chunks to meet backup requirements, it does not maintain the location of all chunks. The orchestrator only requires a small amount of storage to facilitate transfers between firewalled PSC clients.

Clients contribute storage to the PSC which PocketLocker divides into a *file cache*, used to store reconstructed files, and a *chunk store*, used to store chunks. By applying updates from the orchestrator clients maintain a local cache of the namespace to efficiently perform path lookups. Clients also track what chunks for each path they have in their local chunk store. PocketLocker users configure several attributes when attaching clients:

1. **Capacity.** The storage a client contributes to the PSC
2. **Backup.** Whether the client should be used to meet PSC backup requirements. If so, it's failure may lead to data loss. PocketLocker uses this attribute when determining where to backup chunks.
3. **Availability.** Whether files stored on the PSC can be unavailable when the client is unreachable. PocketLocker uses this attribute when determining where to store chunks so that files remain available.
4. **Interactivity.** Whether files are created or accessed on this client. PocketLocker uses this attribute when reclaiming storage.
5. **Power.** Whether the client is battery- or wall-powered. PocketLocker uses this attribute when acquiring chunks during open.

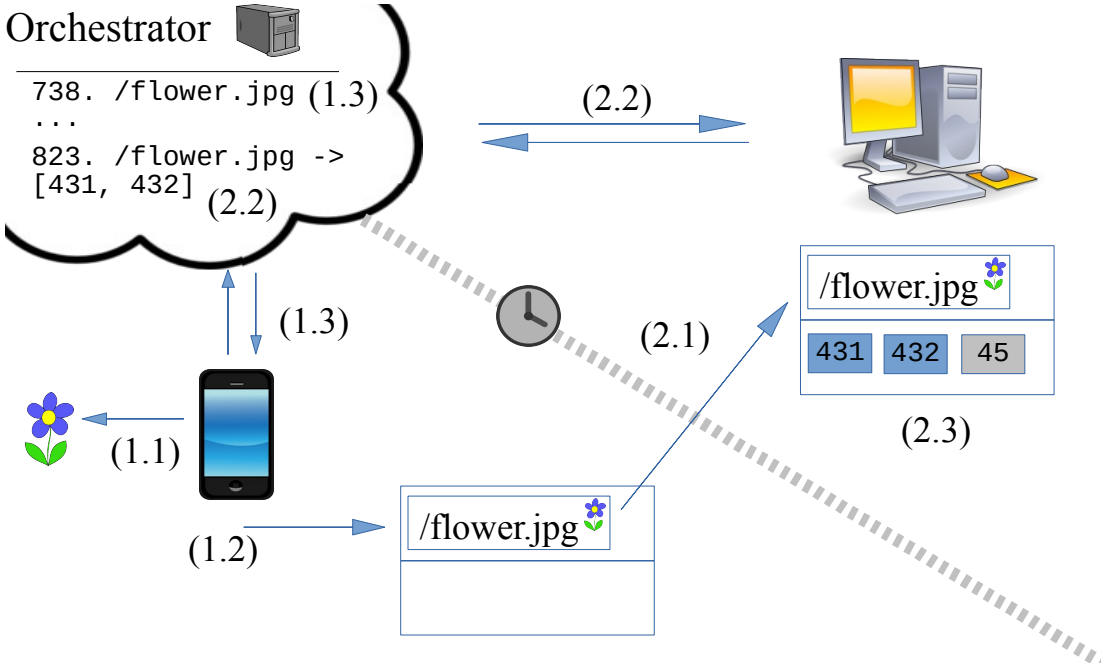


Figure 4.3: **Creation.** This illustrates (1) path registration, performed immediately by a battery-powered client; and (2) erasure coding and chunk registration, performed later by a wall-powered client.

PocketLocker provides example sets of configuration options for common types of devices. A NAS appliance would be used for backup and availability, non-interactive and wall-powered. A laptop would be used for backup but not for availability, interactive and battery-powered. A smartphone would not be used for backup or availability, interactive and battery-powered. A desktop would be used for backup, interactive and wall-powered, and could or could not be used for availability depending on whether it was regularly shut off and whether the user cared if they were able to access their PSC when it was.

4.3.1 Creating, Modifying, and Deleting Files

To reduce energy usage on battery-powered clients, PocketLocker separates creation into two steps which can be performed on different clients: (1) path registration, a lightweight operation; and (2) erasure coding and chunk registration, a heavyweight operation. Figure 4.3 illustrates the process. When a file is created

the creator moves the file into its file cache (1.1) and immediately registers the path with the orchestrator (1.2), which immediately publishes it to other clients to avoid path collisions (1.3). During the second part of creation, once the file is erasure coded (2.1) n new chunks will be created and registered with the in setting up our experiments and providing information about the testbed. orchestrator, which assign them unique IDs and associates the set of IDs with the path (2.2). The client then moves the chunks into its chunk store (2.3). Battery-power clients will wait to transfer the file for a period of time configured as part of the backup process, described later in Section 4.3.4.

Modifications to existing PocketLocker files create a new version of a file. They require an additional round of erasure coding and distribution of new chunks. Because updating files is a heavyweight operation, PocketLocker is designed for files that are rarely or never changed, such as the media files in our traces of which were almost never altered. Renames simply alter the path associated with existing chunks, and deletes removes the path from the namespace.

Both updates and deletions invalidate chunks which clients add to a reclamation list, but chunks are not removed until storage is needed. Because clients do not coordinate chunk removal with the orchestrator, PocketLocker provides no guarantees about the existence of old version or deleted files. However, because the reclamation list is processed in FIFO order, modifications are generally removed first. Providing stronger semantics would require more coordination between clients and the orchestrator which we have chosen to avoid.

4.3.2 Opening Files

To open a file, the opener first verifies that the path is valid. If the file is already in the file cache, the open completes immediately. Otherwise, the client maps the path to the n chunk IDs and locates k as follows:

1. **Local chunk store.** If the opener has k chunks of the file in its chunk store it reconstructs the file and adds it to its file cache.

2. **LAN transfer.** If the opener is on a LAN with other PSC clients it will broadcast a *chunk request* identifying the path and chunks it needs

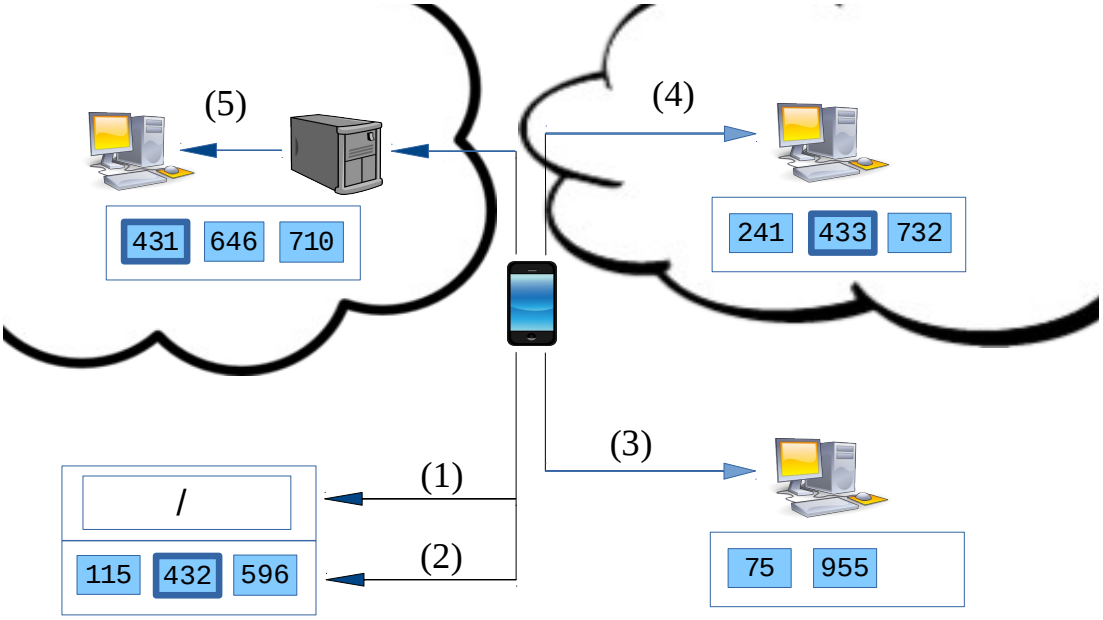


Figure 4.4: **Open.** The figure illustrates a case where the request is satisfied by locating k=3 chunks: one in the client's local chunk store, and two on PSC devices in the WAN.

to its LAN PSC neighbors which will each report which requested chunks they store. PocketLocker clients discover neighbors using a simple UDP broadcast. Based on the replies the opener will acquire any needed chunks and add them to its chunk store. If it has k chunks, then the open continues as in Step 1.

As an optimization, if an opener requests k chunks for a path and a PSC neighbor has the reconstructed file in its file cache, it will offer to transfer the file instead of chunks. This optimization is also applied in Steps 3 and 4.

3. **WAN transfer.** If the opener has not acquired k chunks after Step 2, it forwards the remaining request to other reachable PSC clients and processes replies as in Step 2.

4. **Orchestrated transfer.** If at the end of Step 3 the opener still does not have k chunks, it forwards the remaining request to the orchestrator. At

this point the orchestrator may be able to facilitate transfers with clients not publicly reachable in Step 3, or the open may fail.

Figure 4.4 highlights the above process. The client needs chunks 431-433 to reconstruct a file. The client first checks its local cache (1) and is unable to locate the file. It is, however, able to locate chunk 432 in its own chunk store (2). A search of LAN devices does not turn up any of the desired chunks (3), but the client finds chunk 433 on a WAN device (4). Finally, the Orchestrator is able to locate the last chunk, 431, on another WAN device (5).

PocketLocker reduces energy consumption at battery-powered clients in two ways. First, it allows them to delegate opens to a wall-powered client which receives a delegated chunk request and then proceeds as in Step 2: issuing any additional chunk requests on the battery-powered client's behalf and transferring all chunks to the opener when the open completes. Second, all clients will prefer to transfer chunks from wall-powered clients in Steps 2 and 3.

Depending on where required chunks are located, opening a file can take a variable amount of time. We allow apps to request a notification if an open may require more than a configurable amount of time, allowing them to notify the user or move themselves temporarily into the background until the required transfers complete.

Finally, PSC clients can request files as soon as they receive the path creation notification, meaning that this can occur before the file has been erasure coded and the chunks registered. In this case the open request only specifies the path, and clients reply if they have the file in their file cache. Normally the file creator will be the only PSC client with the file and required to transfer it to the opener. If the opener is wall-powered, it then performs the erasure coding and creation continues as described previously. We expect that in most cases when files are requested before they have been erasure coded, the user has moved the creator onto the same LAN with the opener—such as when a user tries to open a photo taken on their smartphone on their laptop. If so, the time and energy required to transfer the file to the opener should be minimal.

4.3.3 Performance

PocketLocker attempts to use all available client storage to enable reliability, availability, and performance by intelligently locating chunks within the PSC. PocketLocker reduces the latency of file accesses in two ways. First, because many file accesses occur soon after the file is created, PSC clients opportunistically acquire k chunks of newly created files after receiving creation notifications from the orchestrator. On wall-powered clients this is done immediately; battery-powered clients wait until their next charging session. To evenly distribute chunks between clients to help meet later backup requirements, these chunk requests identify the path but not the chunk IDs, allowing the client receiving the request to provide distinct subsets of k chunks of the n available to different clients. Initial chunk requests also disable the optimization described previously to prevent the creator from transferring the reconstructed file rather than file chunks.

Second, PSC clients track local file access patterns to intelligently manage their local chunk store when reclaiming space. When under storage pressure, after emptying their reclamation list, clients can either (1) remove reconstructed files from their file cache or (2) remove chunks from their chunk store. Because erasure decoding is more efficient than erasure encoding, clients first remove files in their file cache such that they have enough chunks in their chunk store to reconstruct.

At this point removing either files or chunks allows the client to make latency tradeoffs between different files. For example, keeping one chunk of many files reduces the access latency of all but provides low-latency access to none[1]; at the other extreme, keeping complete files—either in the file store or as k chunks—provides low-latency access to a smaller set of files but higher latency for the rest. Because file chunk size varies, removing one chunk of a large file can create more space than several chunks of smaller files, but removing chunks of more files increases the probability that a chunk will be required during open. We compare several algorithms for reclaiming storage in Section 4.5.1 evaluating their performance on our PhoneLab file access traces.

[1]Note that wall-powered PocketLocker clients can also repeat the erasure coding process to reconstruct missing chunks for files in their file store, but do to the overhead of erasure coding battery-powered clients will not.

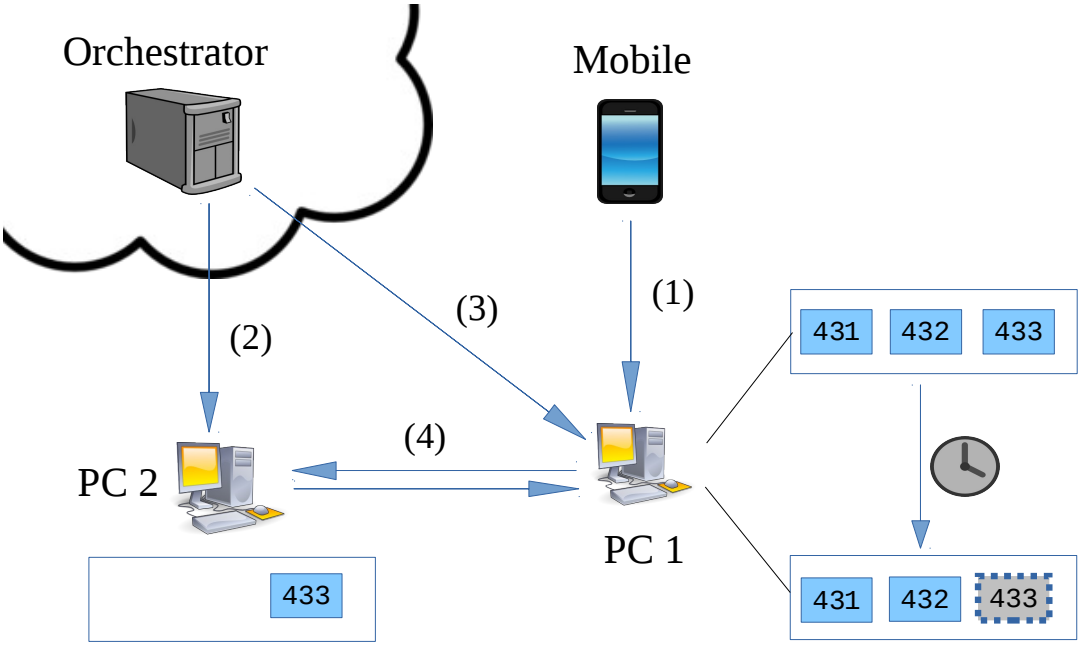


Figure 4.5: **Backup.** A file is received and chunked by a powered device. Under the direction of the Orchestrator, pinned chunks are distributed among different devices.

Interactive and non-interactive clients reclaim storage differently. Interactive clients keep statistics on their own chunk access patterns and utilize them during reclamation, but do not track chunks transferred to other devices in response to chunk requests. Because non-interactive clients do not access files locally, they only keep statistics on chunks accessed to respond to chunk requests. As a result, interactive clients optimize for their own behavior, while non-interactive clients optimize for the clients they interact with.

4.3.4 Backup and Availability

The orchestrator both meets backup requirements and ensures availability by *pinning* chunks at clients to ensure that k chunks will always be available—as long as the client configured as available are reachable—and survive client failures. Pinning prevents clients from removing chunks during reclamation. PocketLocker allows users to configure their PSC to not lose any files older than a certain time interval (the backup window) if up to a certain number of clients fail (the backup

threshold).

Figure 4.5 shows the most common steps in the backup process. A powered device receives and chunks a file originally created by a mobile device (3.1). All chunks are initially pinned by default. Next, the Orchestrator orders PC 2 to request chunk 433 from PC 1 (3.2), and PC 1 to unpin chunk 433 after the chunk has been copied to PC 2 (3.3). The clients then carry out these instructions: PC 1 fetches and pins chunk 433 from PC 2, and PC 1 unpins chunk 433 (3.4).

Backup and availability requirements can reduce the usable size of the PSC depending on the distribution of storage contributed by clients and how they are configured. For example, a single small client can limit the size of the entire PSC if its storage must be used for backup. Or, a single large client may find its storage underutilized if it is not marked as available. PocketLocker estimates the capacity of the PSC at configuration time as the lesser of (1) the sum of all the capacity contributed by clients marked as available and (2) the sum of the storage contributed by the smallest backup clients required to meet the backup threshold. The tradeoff between client attributes and the PSC capacity is presented to the users when they configure clients and choose their backup threshold. Remaining PSC space is not unused: PocketLocker uses it to improve performance by caching chunks and reconstructed files, and to allow users to recover deleted files and old file versions.

When the orchestrator is unable to meet the backup or availability requirements the PSC is full and new files cannot be created. The user is warned when the PSC is nearing capacity and requested to add storage or remove files. To allow file access, interactive clients reserve a portion of their storage for the file cache; to allow chunk transfers, all clients reserve a portion of their chunk store for unpinned chunks.

The backup window allows PocketLocker to reduce energy usage on battery-powered clients by not forcing them to immediately transfer created files to other PSC clients or receive pinned chunks required for backup. When new files are created on battery-powered client, the client begins attempting to offload the file to a wall-powered client, which will perform the second part of the file creation process, including erasure coding and distributing chunks to other clients. Our current algorithm waits a configurable portion of the backup window for the

device to be plugged in, and if that time window expires it transfers the file as soon as it reaches an energy-efficient network such as a wired or Wifi connection. When the backup window is about to expire, any available connection—including mobile data networks—is used as a last resort. Users are warned that short backup windows will produce high energy consumption when configuring their backup window.

Users can report client failures to PocketLocker manually or configure PocketLocker to consider a backup client as failed if the orchestrator cannot reach it for a period of time. Once a new client has been attached to the PSC after a failure, the orchestrator will immediately rerun the pinning algorithm described in Section 4.3.6 which will cause the new client to request chunks needed to meet the backup requirement. In certain cases erasure coding may need to be repeated for some files to recover the full set of n chunks, but this can proceed using any k chunks that are available.

4.3.5 Erasure Coding Parameters

The erasure coding parameters affect the design of the PocketLocker PSC in two ways. First, if n is smaller than the number of backup clients then the orchestrator may need to move a chunk from one client to another to rebalance storage usage while meeting backup requirements. Since this is undesirable, we choose n to be equal to the number of devices initially configured for backup.

Second, k determines both the chunk size—which is equal to the file size divided by k—and the overhead of erasure decoding, which increases with k. Using larger values of k and creating larger numbers of smaller chunks allows more even storage distribution over clients, and allows clients to make finer-granularity tradeoffs between storage and access latency by caching between 1 and k chunks of the file in their chunk store. However, due to PocketLocker's focus on supporting battery-powered clients, we set $k = 2$ to minimize the energy overhead of erasure decoding.

4.3.6 Chunk Pinning Algorithm

Periodically the orchestrator collects a list of chunks they are storing from all PSC clients and runs a *chunk pinning algorithm* to determines where to pin chunks to meet the user's backup and availability requirements. Our current placement algorithm uses a greedy approach that meets the backup requirements and availability requirements in separate passes. If the size of the PSC is constrained by the backup requirement, the availability pass proceeds first in order to avoid reducing capacity on clients needed for backup. If the size of the PSC is constrained by the availability requirement, the order of the two passes is reversed.

In each pass, for each file the orchestrator begins with the client with the most capacity and begins pinning chunks until the requirement is met. The backup pass stripes chunks across backup clients to meet the backup requirement, while the availability pass stacks chunks onto available devices to meet the availability requirement. The algorithm attempts to avoid chunk transfers when possible by considering what chunks each client already has pinned or available in their chunk store.

When clients receive a list of pinned chunks from the orchestrator, they retrieve any chunks they are missing using the chunk request process described previously. To ensure that chunks for newly-created files are not evicted before they can be pinned by the orchestrator, chunks for new files and file updates are initially pinned after creation at all clients. The next time the backup algorithm runs, many of these chunks will be unpinned.

4.3.7 Offline Operation

PocketLocker assumes clients are connected most of the time, but can support periods of disconnected operation. Disconnected clients can access any files in their file store or that they can reconstruct using chunks in their local chunk store. Any changes to the namespace, such as creations, are cached. When the client reconnects, it downloads any namespace updates from the orchestrator and identifies any conflicts which the user is required to resolve locally. Because it is designed to store media files, PocketLocker does not attempt to merge conflicting

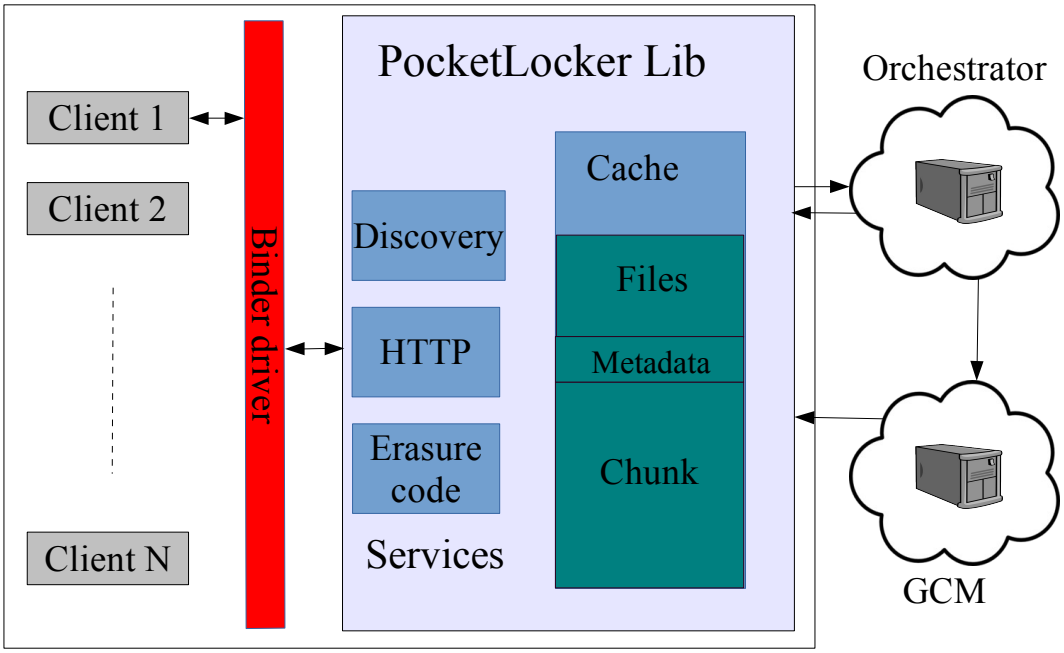


Figure 4.6: **Architecture.** The figure illustrates the different components in the implementation of PocketLocker.

creations or modifications. Instead, users are asked to choose between updates or rename the file.

4.3.8 File Metadata

Finally, to support media files that may require metadata for browsing, such as photo thumbnails, PocketLocker allows metadata files up to a size limit to be associated with files stored in the PSC. Metadata files are stored in a separate part of each client's storage and retrieved during the initial chunk requests that follow file creation. Unlike chunks, however, metadata files are not reclaimed, since we assume that the overhead of storing them will be limited.

4.4 Implementation

We have implemented PocketLocker PSC as an Android background service on both interactive and fixed non-interactive devices. Galaxy Nexus and Nexus

5 smartphones constituted the interactive devices, and Android x86 virtual machines Androidx86 [2014] running on desktops served as the fixed non-interactive. Figure 4.6 illustrates the various components of the PocketLocker service. The PocketLocker service runs in the background and exposes APIs to provide clients access to the files stored in the user's PSC. It also maintains chunk placement in the cache as directed by the orchestrator.

We chose to implement PocketLocker as a user application rather than integrating the service with the file system so that users are not required to have root privileges to install PocketLocker on their devices and so that PocketLocker can be easily distributed via the Android Play Store. Store [2014] On both interactive and non-interactive devices, the PocketLocker service maintains the local file and chunk cache according to the placement directions calculated by the orchestrator. On fixed devices, PocketLocker additionally offers a pair of network services. The first, the DiscoveryService, responds to chunk requests that are issued by interactive devices on the same local network. The second, the HTTP service, facilitates the transfer of newly created files and chunks among the user's PSC devices as per the chunk placement scheme.

PocketLocker exposes its APIs both to the orchestrator, to receive local cache maintenance directions, and to local client applications, to provide access to user files. PocketLocker clients interact with the PocketLocker service via the *binder* driver framework in Android. The binder framework facilitates thread safe inter process communication between applications in Android.

The orchestrator was implemented using the Tornado and Flask web frameworks. The orchestrator listens to status updates by the user's PSC devices and tracks and maintains the cache information at each of the devices in the user's PSC using the *SQLite* database SQLite [2014]. To efficiently push information to user's PSC device, the orchestrator uses the Google Cloud Messaging (GCM) framework to communicate information about new file creation and chunk placement with the user's PocketLocker devices. GCM provides an energy-efficient means to push notifications to energy-constrained devices.

API	Description
Client → PocketLocker	
getFileList()	Returns all user file information
getMetadata(filename)	Returns metadata for the requested file
openFile(filename)	Reconstructs the requested file
PocketLocker → Orchestrator	
createFile(filename)	Requests Orchestrator to add new file to user PSC
erasureencodingdone(chunkinfo)	Update information about the newly created chunks
deletefile(filename)	Request to delete file
statusUpdate(lockerstatus)	Periodic status update
Orchestrator → PocketLocker	
newFileCreated(fileinfo)	Updates the locker with the metadata of the newly created file
downloadChunk(locationinfo)	Instructs a locker to download the chunks as per placement
pinChunks(chunkids)	Request the locker to pin the given chunks in its cache
unpinChunks(chunkids)	Unpin given chunks that to enable storage reclamation

Table 4.3: **Interfaces.** The table summarizes the different endpoints and interface exposed at each of these endpoints by the PocketLocker service.

4.5 Evaluation

We evaluate PocketLocker in two ways. First, we return to the detailed file access traces we collected on PhoneLab and analyze them to determine the impact of parameters important to PocketLocker's design. We also use them as inputs to a trace-based simulation to compare approaches to performing client storage reclamation. Second, we perform detailed measurements of our PocketLocker prototype engaging in the various types of file accesses described previously. Our results indicate how utilizing nearby clients can improve performance, and also how PocketLocker enables energy-efficient operation on battery-powered clients.

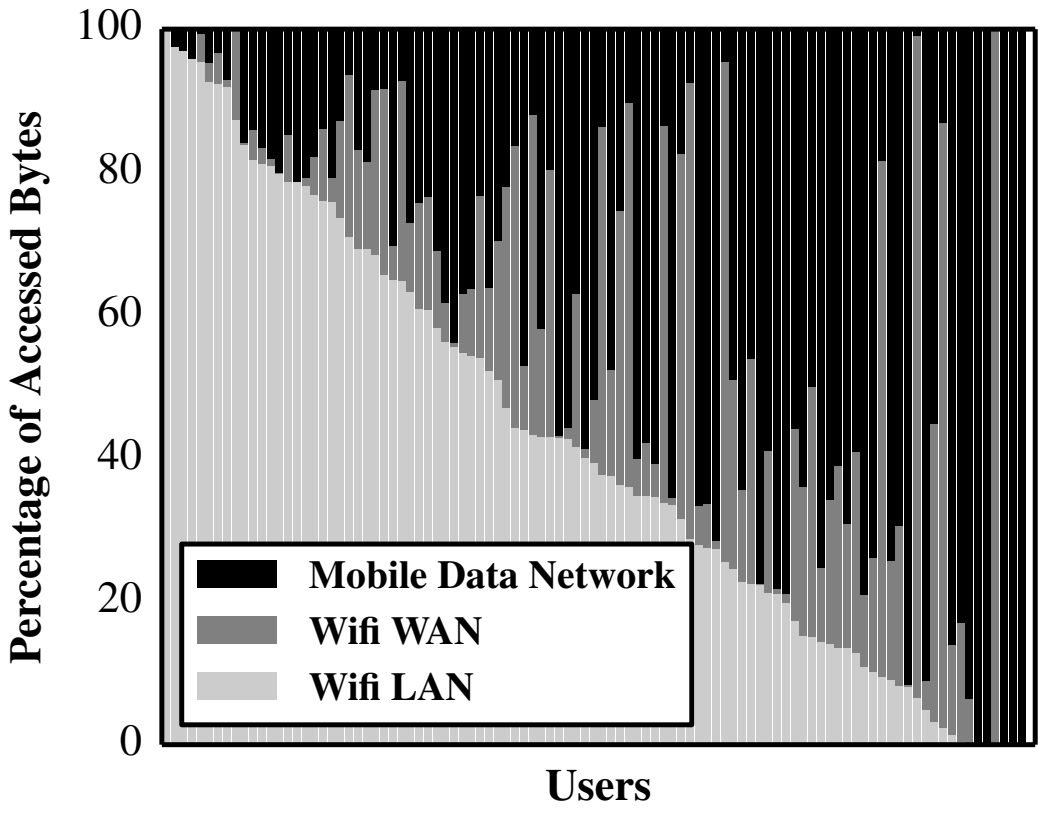


Figure 4.7: **Connectivity During File Accesses.** Placing PSC clients on each user's two most frequently-used Wifi networks could absorb a large portion of their file access activity.

4.5.1 Trace Analysis

PocketLocker relies on nearby clients to improve performance of file accesses. In the best case, other PSC clients are located on the same LAN. To determine whether nearby clients can assist with file accesses, we performed further analysis of the traces described in Section 4.2.

Because PhoneLab only provides visibility into participant smartphones, we have to infer where users would have other PSC clients nearby. To do so, we simulated the presence of PocketLocker PSC clients on the two Wifi networks that each user spent the most time connected to, which could represent home and work networks. We then divided file accesses into three categories: (1) ones

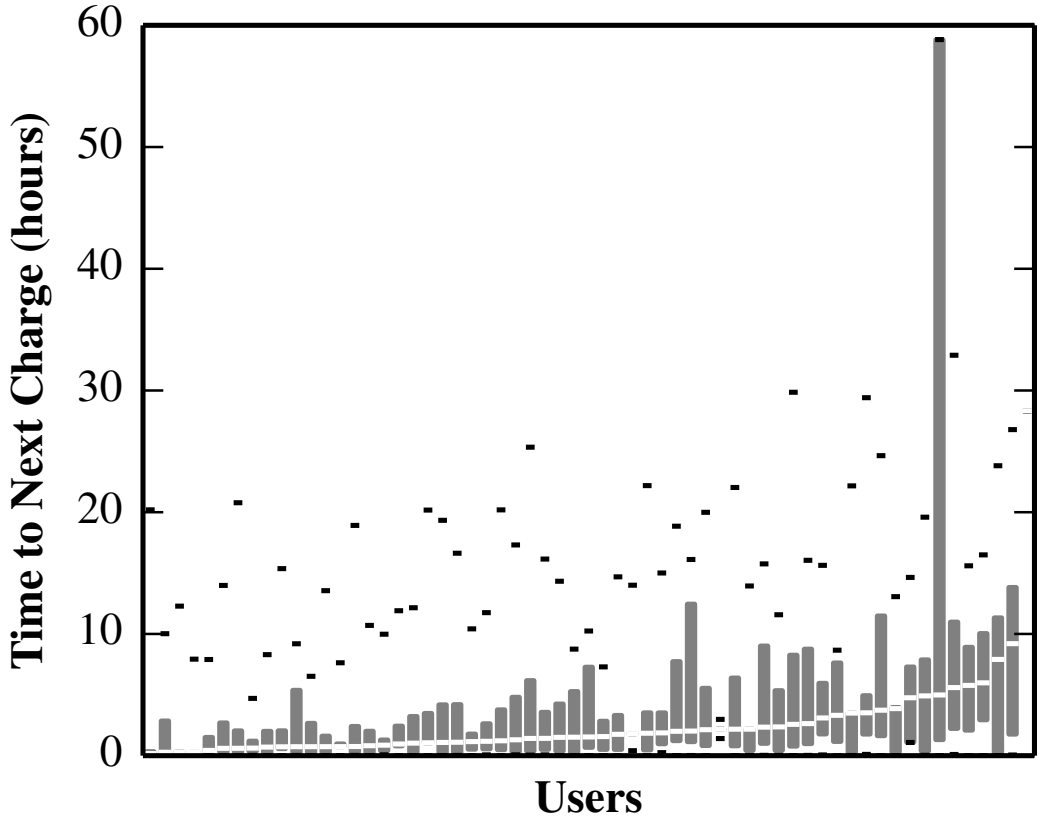


Figure 4.8: **Time Until Next Charge After File Creation.** Separating the process of creating files into two steps allows PocketLocker to reduce energy consumption on battery-powered client by performing transfers during the next charging cycle.

that occur on the same LAN with a simulated PSC client, (2) those that do not occur on a PSC LAN but still occur while the user is connected to a high-speed and energy-efficient Wifi network, and (3) those that occur when the user is connected to a mobile data network[1]. Figure 4.7 shows the results; for around half of the users, even without a local cache half of the file accesses could be served by two PSC clients placed at their most frequently-used Wifi networks.

We were also interested in how many file creations could be offloaded to pow-

[1]We found no file accesses that occurred more than five minutes from log messages indicating the presence of a mobile data network, a reflection of the always-connected nature of smartphones.

ered clients by delaying transfer until the user plugged their smartphone in to charge. Figure 4.8 shows per-user distributions of the of time between file creations and the next charging session. For all users, the median is under 10 hours with worst-case maximums approaching a day. Overall, the results suggest that by delaying the initial file transfer required during creation for a portion of the user's backup window, PocketLocker can enable energy-neutral transfers and reduce overhead on battery-powered clients.

Finally, we built a simple trace-based simulator to experiment with different policies for managing the mobile client chunk store. We configured each PSC client smartphone with 1 GB of storage, considerably less than the amount of file accesses we observed during our one-month experiment, and managed the chunk store using four different algorithms: random eviction, first-in-first-out (FIFO), least-recently-used (LRU), and least-accessed first (Access). Figure 4.9 compares the results. When file accesses missed the chunk store, we classified the access as described previously based on the smartphone's connectivity at that moment. Surprisingly, we did not observe any large performance differences between these algorithms, although they were able to manage the local chunk store to absorb a large number of file accesses. A great deal of inter-user variation is also visible, and we are continuing to study how to better adapt PocketLocker's reclamation algorithms to the specifics of each users file access patterns.

4.5.2 Prototype Performance Evaluation

We evaluated the prototype of the implementation described in Section 4.4 in two ways. First, we measured the time required to access files of various sizes with devices connected to different networks types. Secondly, we measured the energy consumption to access files. In our experiments we chose $k = 2$ as the number of chunks required to reconstruct the file. We used Samsung Galaxy S4 and Nexus 5 smartphones as interactive devices, and utilized Android virtual machines running PocketLocker as fixed devices within the same subnet.

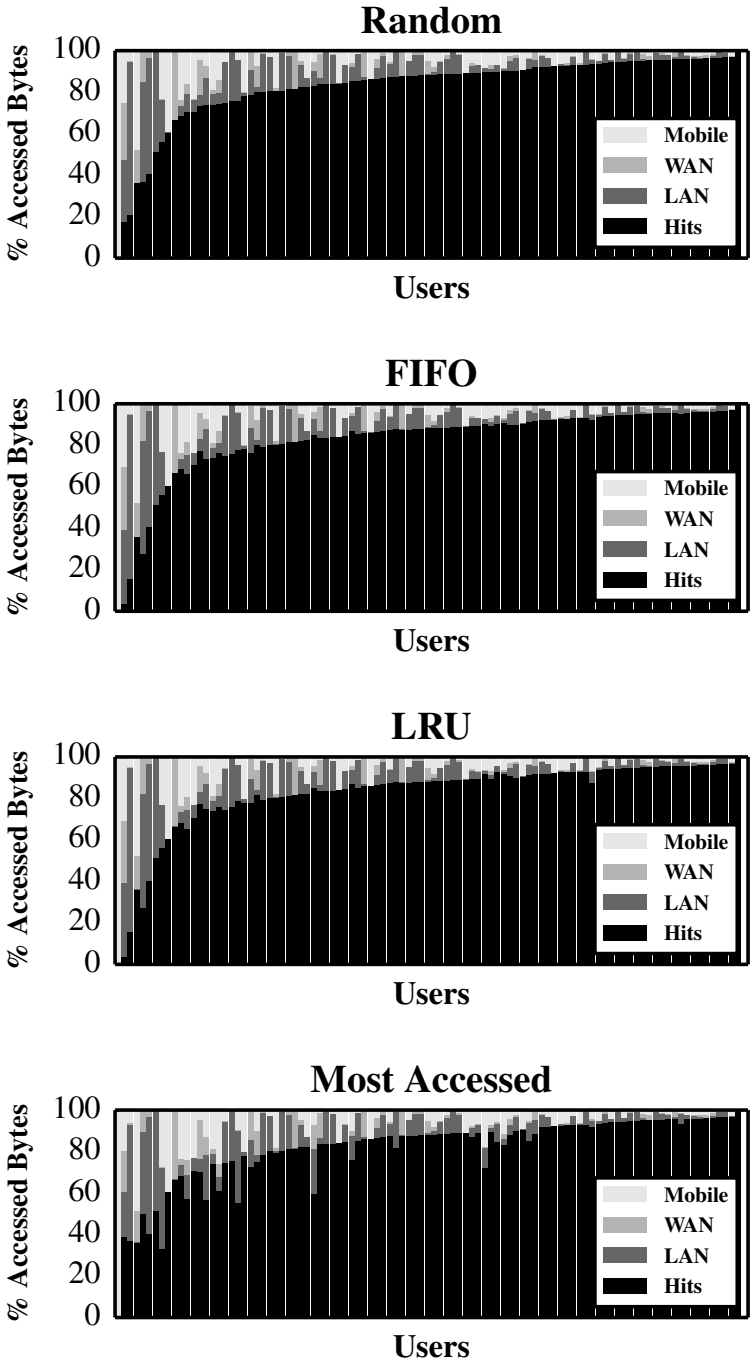


Figure 4.9: **Comparison of Reclamation Algorithms.**

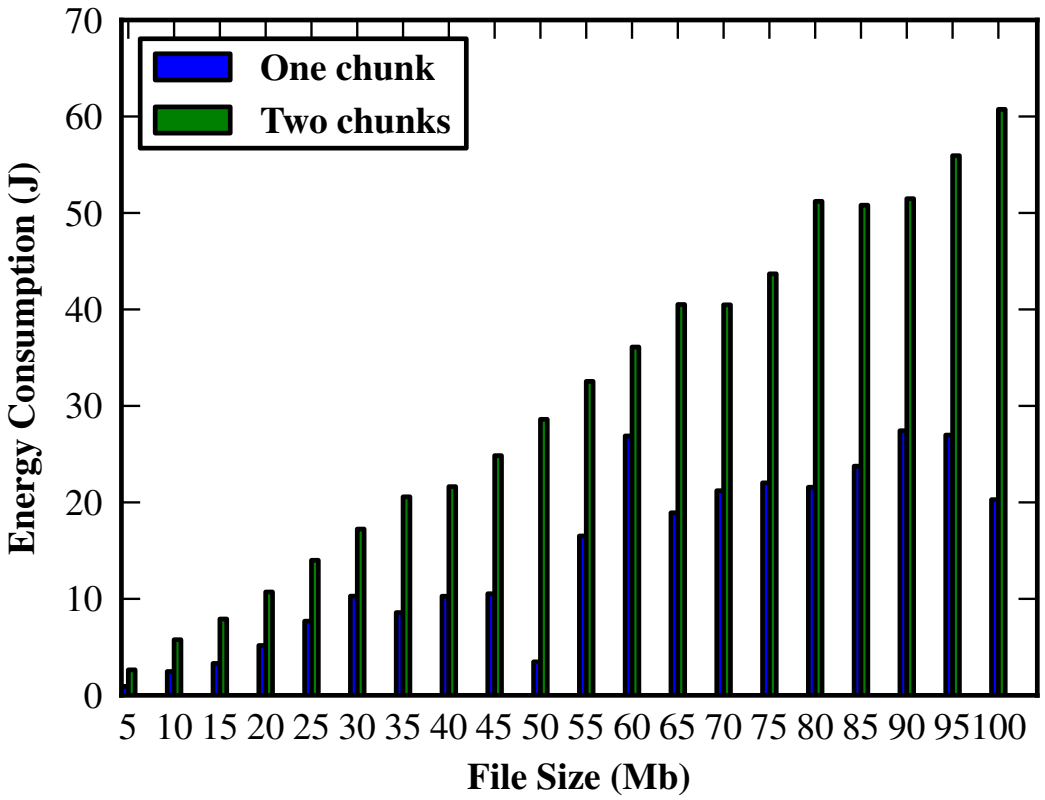


Figure 4.10: **PocketLocker energy savings.** Figure illustrates the savings in energy when an interactive device downloads one chunk compared to downloading two chunks to access the file.

4.5.2.1 File Access

Figure 4.11 illustrates the time required to download files of different sizes with clients having to download k chunks to reconstruct the file when connected to different networks. The *On Device* scenario denotes the time required only to reconstruct the chunks locally to reconstruct the original. This is the best scenario as there is no download of chunks involved. In *LAN Wifi*, we have a fixed device present on the same LAN as the interactive device. The device downloads both chunks from the fixed device and is the fastest compared to any other connection type. *WAN Wifi* has fixed devices that are publicly accessible over the Internet to interactive devices. WAN Wifi is analogues to downloading files from the

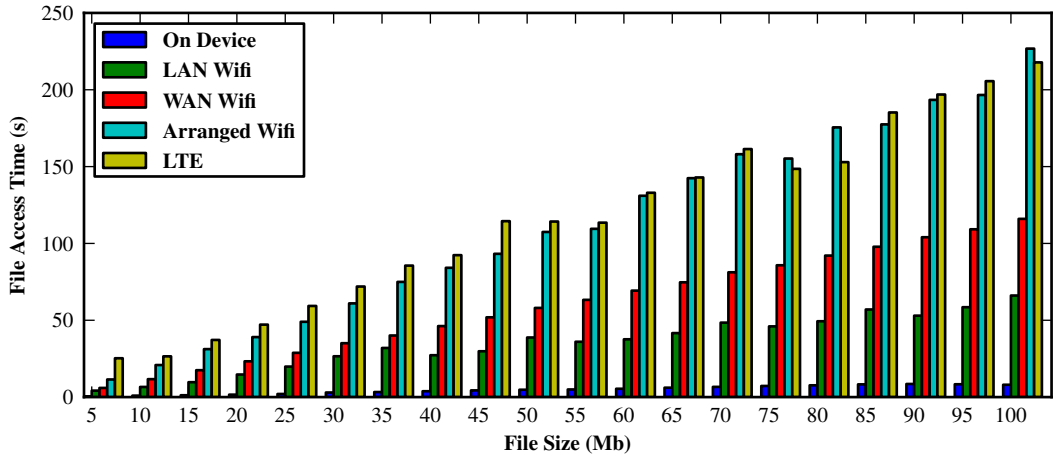


Figure 4.11: **PocketLocker file access times.** Figure illustrates the times required to access files of various sizes by PSC in different types of connectivity.

cloud today. *Arranged Wifi* presents the scenario where the fixed devices are not publicly accessible and data transfers are done via a relay. Here, the access times to open a file when the chunks are downloaded on the LAN Wifi are almost 50% faster when compared to WAN Wifi scenario. This result is encouraging, as we envision most chunk transfers happening in the LAN Wifi mode.

4.5.2.2 Energy Consumption

We used the Monsoon power monitor Monsoon [2014] to measure the energy consumption for file access time for the scenarios described in Section 4.5.2.1. Figure 4.12 illustrates the energy consumption on the interactive device. As expected, data transfers over the cellular network consumes the most amount of energy. We also noticed that the energy consumed for the WAN Wifi was lesser when compared to LAN Wifi. We believe this occurred because the smartphone was connected to an open access point for WAN transfers whereas the interactive device was connected to an access point with WPA security for LAN Wifi. Figure 4.10 compares the energy consumption of file access when downloading two chunks with the energy required to access the file by downloading one chunk. The energy consumption of PocketLocker when accessing the file by downloading one chunk is less than the consumption to access the file by downloading

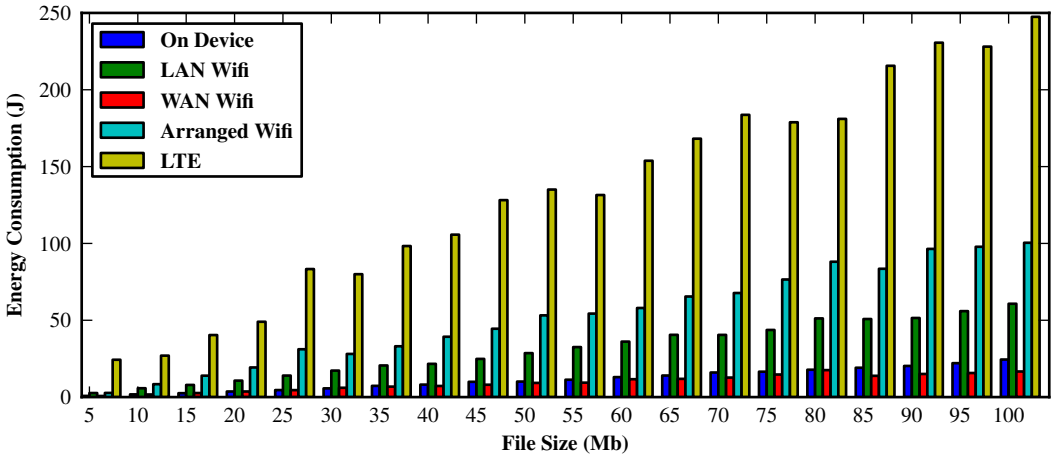


Figure 4.12: **PocketLocker energy consumption.**Figure illustrates the energy consumption on interactive device to access files of different sizes from fixed devices in various types of network.

both chunks. This is a positive result for PocketLocker as it stores only some of the chunks required to reconstruct the file instead of all chunks under storage pressure.

4.6 Summary

PocketLocker addresses an emerging need of mobile systems by crafting a personal storage cloud from multiple personal devices. It targets storing rarely changing files. An intelligent Orchestrator arranges storage to maximize usage of devices of different sizes and minimize network costs. PocketLocker is free and uses no additional devices. System storage and backup policies are based upon data gleaned from an extended testing using 100 smartphones. The next chapter reviews the related work to topics discussed so far in this dissertation.

5

Related Work

This chapter discusses the related work to the topics discussed so far in the dissertation. The related work is grouped into three sections with each section discussing the related work to the topic associated with that section. Section 5.1 before discusses the related work to PhoneLab. Section 5.2 distinguishes PocketParker from multiple previous efforts at parking monitoring. and finally Section 5.3 compares PocketLocker to similar systems.

5.1 Related work in smartphone testbeds

The related work to PhoneLab is discussed in three categories—testbeds, measurement tools, and smartphone measurement studies.

Testbeds: Testbeds in other domains have chosen their design points to meet domain-specific needs. For example, PlanetLab pla, Peterson et al. [2003] operates more than 1,000 machines world-wide in order to enable large-scale, realistic Internet research. Emulab emu, White et al. [2002] provides emulated network environments to enable controlled, repeatable network experiments. MoteLab Werner-Allen et al. [2005] targets realistic sensor network experiments by deploying a sensor network testbed in a building at Harvard. ORBIT Raychaudhuri et al. [2005] takes a two-tier approach allowing emulated experiments as well as real deployments, targeting reproducibility and realism at the same time. OpenCirrus ope, Avetisyan et al. [2010] and VICCI vic are geographically distributed clusters, designed to support cloud computing research.

Measurement Tools: Researchers have developed special-purpose measurement tools for smartphone usage measurement. These tools are designed to measure certain metrics such as power consumption Pathak et al. [2012], Zhang et al. [2010] or 3G performance Huang et al. [2010]. Though we have developed a measurement tool for our analysis as well, our focus for this study is not the tool design, but the power of PhoneLab.

Smartphone Measurement Studies: Though the primary purpose of our analysis is to demonstrate the power of PhoneLab, our findings complement what previous studies have reported. Falaki et al. Falaki et al. [2010] are among the first ones to study smartphone usage. Their central finding is that in many of the metrics they studied, there was significant diversity without a clear pattern; the metrics include the mean interaction length, the mean number of applications used, the mean amount of traffic, etc. Xu et al. Xu et al. [2011] use a network-level trace to analyze smartphone application usage. The key findings are that smartphone users use many regional applications such as local news apps; certain applications are installed together; and mobility patterns affect the types of applications used. Trestian et al. Trestian et al. [2009] study users of a 3G network and report a similar finding: people's movement and locations correlate with applications they use. They also report that there is a correlation between content upload and location in another study Trestian et al. [2012]. Shye et al. Shye et al. [2009] studies how power consumption is distributed over different hardware components. Kim et al. Kim et al. [2012] focus on correlation between storage and application performance and find multiple factors that affect application performance. Many studies have also looked at Wifi and 3G network characteristics using smartphones Gember et al. [2011], Keralapura et al. [2010], Maier et al. [2010].

5.2 Related work in parking availability

Many previous projects have addressed components of our problem, including activity detection, parking lot monitoring using various types of infrastructure and additional sensors, and transportation-related tracking. Below we discuss related efforts and their relevance to PocketParker.

5.2.1 Activity Detection

Activity detection is a basic primitive that enables other higher-level functionalities. Several previous systems Constandache et al. [2010], Keally et al. [2011], Reddy et al. [2010], Wang et al. [2009], Yang et al. [2011] have proposed techniques for activity detection, solving the main challenges of energy efficiency and accuracy. Yang et al. Yang et al. [2011] develop an algorithm that detects if a driver is using a phone by sending high-frequency beeps via in-car speakers. EEMSS Wang et al. [2009] is a system that provides continuous identification of general human states such as walking, driving, and being in an office. Reddy et al. Reddy et al. [2010] propose a classification approach that determines human movement states such as walking, running, biking, or vehicle-traveling. Constandache et al. Constandache et al. [2010] use smartphone accelerometers to determine users' walking trails. Yan et al. Yan et al. [2012b] propose an adaptive approach that dynamically adjusts the accelerometer sampling frequency for conserving energy. Keally et al. Keally et al. [2011] use a combination of on-body wireless sensors and smartphones to classify human states. Our system can benefit from these techniques for detecting parking-related events; however, we find it sufficient to use a simple detector since it avoids the complexity of detecting events unrelated to parking.

5.2.2 Parking Lot Monitoring

There are a large number of parking lot applications available in the online smartphone application stores. A typical parking lot application provides location as well as pricing information and allow reservation of a parking spot. Some applications also report parking lot availability based on publicly available information. To the best of our knowledge, there is no application that automatically infers parking lot availability by monitoring drivers.

Most close to our work is ParkNet Mathur et al. [2010], a system that estimates street parking availability. ParkNet uses vehicles equipped with GPS and a ultrasonic range finder that scan the surrounding areas and detect empty street parking spots. Compared to ParkNet, our approach relies only on smartphones and do not require any additional input.

A few systems have been proposed to assist parking via vehicular ad-hoc networks or crowdsourcing. Delot et al. Delot et al. [2009] propose a parking lot reservation system in a vehicular network. Chen et al. Chen et al. [2012] proposes a crowdsourcing approach that asks participants to report their surrounding's parking availability. Caliskan et al. Caliskan et al. [2007] propose an availability prediction model based on information exchanged by vehicles in a vehicular ad-hoc network. Their approach assumes that, for each parking lot, vehicles that drive by the parking lot receive the parking lot information such as the capacity, the occupancy, the arrival rate, and the departure rate. Since the propagation delay in a vehicular network makes this information stale, a prediction model is used to estimate current availability. In contrast to our work, this approach assumes that the necessary information is initially accurately measured at each parking lot.

5.2.3 Tracking-Related Projects

A few previous systems have investigated techniques for automatic transit tracking Biagioni et al. [2011], Thiagarajan et al. [2010], Zhou et al. [2012]. From the high-level point of view, some of the techniques such as activity detection and location tracking that transit tracking requires bear similarities to our techniques; however, these techniques need to be optimized and tailored towards different scenarios, hence the specifics vary widely.

Thiagarajan et al. Thiagarajan et al. [2010] propose an automatic, real-time transit tracking approach that uses smartphones of public transit riders as data sources. They propose an algorithm to detect when a user is traveling in a vehicle and an algorithm to detect if a vehicle is a public transit vehicle. EasyTracker Biagioni et al. [2011] uses smartphones deployed in buses to enable automatic transit tracking. The goal of EasyTracker is to require no other input than what is from the deployed smartphones. The system combines a few mechanisms to realize this goal such as route extraction, stop extraction, and arrival time prediction. Zhou et al. Zhou et al. [2012] propose a bus arrival time estimation system based on smartphones used by public transit riders. They combine multiple sources

such as accelerometer data, audio, and cell tower signals to detect if a rider is in a public transit vehicle and if so, which bus it is.

Other systems have used smartphone sensors to enable a variety of tracking tasks. VTrack Thiagarajan et al. [2009] is a traffic monitoring system that combines readings from multiple sensors for travel time estimation. StarTrack Ananthanarayanan et al. [2009] provides general abstractions for appliations that need tracking functionalities such as recording, comparing, and querying tracks.

5.2.4 Urban On-Street Parking

A recent academic study also focus on the problem of locating on-street parking in urban areas. Nawaz et al. Nawaz et al. [2013] leverage the ubiquity of WiFi beacons to monitor on-street parking events in a similar fashion to that performed by PocketParker. Our study, borne out of suburban campus locale, must cope with alternative sensing mechanisms in the wake of no proximate WiFi signals. We have also tuned our tracking and reporting methodology to address the needs of lot rather than street parking. Nationally, data suggests that the proportion of spots in lots is anywhere from somewhat less than to five times the number of spots on streets. Chester et al. [2010]

5.3 Related work in distributed storage

Mobile devices, being relatively new, did not contribute to the design of prototype distributed file systems. Early systems such as Coda Kistler and Satyanarayanan [1992] and Ficus Guy et al. [1990] were concerned with addressing the base problem of file caching and replication. The limitations of mobile devices, particularly constrained storage and energy and intermittent connectivity, were not relevant. Standard network file systems such as NFS Nowicki [1989] did not provide direct offline access or redundancy.

By contrast, there are robust commercial solutions such as TimeMachine Machine [2014] that furnish redundant storage from any device. These cloud solutions are also typically limited in space and use third party storage.

The approach taken by EnsemBlue Peek and Flinn [2006] focuses on replicating files among mobile devices. Users can specify file groups that are automatically replicated. Cimbiosys Ramasubramanian et al. [2009] narrows this approach, implementing data filters such as file type to determine replication policy. Files that do not match the filter are not replicated. These approaches limit access to files that can fit on a particular user's device. Additionally, since a file will not always be replicated, there is no specific attempt to provide file backup. Since offline edits are allowed, conflicts occur and must be resolved. PRACTI Belaramani et al. [2006] focuses on maximizing the tradeoffs of the general goals of consistency, replication and independence. This necessarily unfocuses the specific needs of mobile storage.

PocketLocker aims to make all files in the PSC available. Which files are maintained locally are determined by usage patterns and network conditions. Those that are not are still available with a possible delay. The size of the PSC can thus greatly exceed the local storage of a particular device. The chunk distribution system of PocketLocker minimizes the impact of device failure and ensures file redundancy.

The Eyo system Strauss et al. [2010] provides a distributed unified namespace. While file metadata is automatically replicated, replication of file data is left to rules specified by client programs. Thus, files may not be replicated against failure. If a user wants to access a nonlocal file, the system can furnish its current location but does not automatically retrieve it. Editing a file offline can result in a conflict that must be resolved. The system addresses storage pressure by pruning file version history without respect to possible loss of redundancy.

The concept of separating the distribution of file metadata from data underpins another system, Ori Mashtizadeh et al. [2013]. Accessing remote file data depends on being able to access that device directly. Otherwise, the call fails. Ori permits users to move versioned file histories among devices—permitting offline editing but incurring storage overhead and producing conflicts. File backup focuses on versioning. Whether a file is replicated depends upon whether the user has mounted a remote system. Implementation of deliberate redundancy, in the form of multiple copies on multiple devices, remains a function of user choices.

PocketLocker handles replication of both file metadata and data directly. The system, having a bird's eye view of all storage devices, can ensure that files are always chunked and replicated to disparate devices to guard against failure. Distribution of the chunks is tuned to the differing storage capacities of different devices. Storage reclamation policy follows file history and usage patterns in order to maximize backup potential. The centralized design of PocketLocker also allows it to handle potential remote access issues. If a file or chunks are not directly reachable from a client device due to firewall issues, the Orchestrator can often mediate an indirect relay transfer rather than simply failing on the call.

5.4 Summary

This chapter discussed the existing work related to the topics described in the dissertation. In addition, the survey presented in this chapter distinguishes the work done in this dissertation with the existing work. The following chapter concludes the dissertation.

6

Conclusion

This dissertation presented PhoneLab, a new large-scale programmable smartphone testbed. PhoneLab enables next generation mobile system research by supporting experimentation at the application, platform and kernel levels. In addition to PhoneLab, this dissertation presented two new mobile system frameworks–PocketParker and PocketLocker. Both PocketParker and PocketLocker were evaluated using PhoneLab.

PocketParker is a crowdsourcing solution for predicting parking lot availability. PocketParker requires no explicit user input and can provide parking lot predictions without being removed from a user's pocket. PocketParker introduces and addresses the challenges pocketsourcing, a subset of crowdsourcing that does not require any manual user input.

PocketLocker addresses an emerging need of mobile systems by crafting a personal storage cloud from multiple personal devices. It targets storing rarely changing files and presents a system enabling scalable, reliable, and performant personal storage clouds.

Relevant Publications

Anandatirtha Nandugudi, Anudipa Maiti, Taeyeon Ki, Fatih Bulut, Murat Demirbas, Tevfik Kosar, Chunming Qiao, Steven Y Ko, and Geoffrey Challen. Phonelab: A large programmable smartphone testbed. In Proceedings of First International Workshop on Sensing and Big Data Mining, pages 1–6. ACM, 2013.

Anandatirtha Nandugudi, Taeyeon Ki, Carl Nuessle, and Geoffrey Challen. Pocketparker: Pocketsourcing parking lot availability. In Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing, pages 963–973. ACM, 2014b. 2

A. Nandugudi, C. Nuessle, G. Challen, E. Miluzzo, and Yih-Farn Chen. The pocketlocker personal cloud storage system. In Mobile Computing, Applications and Services (MobiCASE), 2014 6th International Conference on, pages 235–244, 2014a. 3

References

Broadcom BCM4751 Integrated Monolithic GPS Receiver. http://www.broadcom.com/products/GPS/GPS-Silicon-Solutions/BCM4751. 22

Emulab. http://emulab.net/. 12, 89

Android 4.1, Jelly Bean. http://www.android.com/about/jelly-bean/. 7

Nexus S 4G from Google. http://www.android.com/devices/detail/nexus-s-4g-from-google. 7

OpenCirrus. https://opencirrus.org/. 12, 89

Planetlab. http://planet-lab.org/. 12, 89

VICCI: A programmable cloud-computing research testbed. http://www.vicci.org/. 89

CTIA: 96 million smartphones in US. http://mobihealthnews.com/13796/ctia-96-million-smartphones-in-us/, 2011. 32

Quick, Find a Parking Space. http://goo.gl/Ybpyj, 2011. 31

Parking Lot Design Standards. http://goo.gl/v0F7u, 2012. 36

How to Find a Parking Spot on Campus. http://www.wikihow.com/Find-a-Parking-Spot-on-Campus, 2012. 31

OpenStreetMap. http://www.openstreetmap.org/, 2013. 36

6. REFERENCES

Recognizing the user's current activity, December 2013. URL http://developer.android.com/training/location/activity-recognition.html. 34

Ganesh Ananthanarayanan, Maya Haridasan, Iqbal Mohomed, Doug Terry, and Chandramohan A. Thekkath. Startrack: a framework for enabling track-based applications. In Proceedings of the 7th international conference on Mobile systems, applications, and services, MobiSys '09, pages 207–220, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-566-6. doi: 10.1145/1555816.1555838. URL http://doi.acm.org/10.1145/1555816.1555838. 93

Androidx86. Run Android on Your PC. http://www.android-x86.org/, 2014. 80

Arutyun I. Avetisyan, Roy Campbell, Indranil Gupta, Michael T. Heath, Steven Y. Ko, Gregory R. Ganger, Michael A. Kozuch, David O'Hallaron, Marcel Kunze, Thomas T. Kwan, Kevin Lai, Martha Lyons, Dejan S. Milojicic, Hing Yan Lee, Yeng Chai Soh, Ng Kwang Ming, Jing-Yuan Luke, and Han Namgoong. Open Cirrus: A Global Cloud Computing Testbed. IEEE Computer, 43(4):35–43, April 2010. ISSN 0018-9162. 12, 89

Nilanjan Banerjee, Ahmad Rahmati, Mark D. Corner, Sami Rollins, and Lin Zhong. Users and Batteries: Interactions and Adaptive Energy Management in Mobile Systems. In Proceedings of the 9th International Conference on Ubiquitous Computing (UbiComp), 2007. 20, 24

Nalini Moti Belaramani, Michael Dahlin, Lei Gao, Amol Nayate, Arun Venkataramani, Praveen Yalagandula, and Jiandan Zheng. Practi replication. In NSDI, volume 6, pages 5–5, 2006. 94

James Biagioni, Tomas Gerlich, Timothy Merrifield, and Jakob Eriksson. Easytracker: automatic transit tracking, mapping, and arrival time prediction using smartphones. In Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems, SenSys '11, pages 68–81, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0718-5. doi: 10.1145/2070942.2070950. URL http://doi.acm.org/10.1145/2070942.2070950. 92

6. REFERENCES

Business Insider. Google Is Activating 1.3 Million Android Devices On A Daily Basis. http://goo.gl/SZHqV. 5

M. Caliskan, A. Barthels, B. Scheuermann, and M. Mauve. Predicting parking lot occupancy in vehicular ad hoc networks. In Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th, pages 277–281, 2007. doi: 10.1109/VETECS.2007.69. 31, 92

Xiao Chen, Elizeu Santos-Neto, and Matei Ripeanu. Crowdsourcing for on-street smart parking. In Proceedings of the second ACM international symposium on Design and analysis of intelligent vehicular networks and applications, DIVANet '12, pages 1–8, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1625-5. doi: 10.1145/2386958.2386960. URL http://doi.acm.org/10.1145/2386958.2386960. 31, 92

Mikhail Chester, Arpad Horvath, and Samer Madanat. Parking infrastructure: energy, emissions, and automobile life-cycle environmental accounting. Environ. Res. Lett., 5(034001), July - September 2010. doi: 10.1088/1748-9326/5/3/034001. URL http://iopscience.iop.org/1748-9326/5/3/034001. 93

Ionut Constandache, Xuan Bao, Martin Azizyan, and Romit Roy Choudhury. Did you see bob?: human localization using mobile phones. In Proceedings of the sixteenth annual international conference on Mobile computing and networking, MobiCom '10, pages 149–160, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0181-7. doi: 10.1145/1859995.1860013. URL http://doi.acm.org/10.1145/1859995.1860013. 34, 91

Thierry Delot, Nicolas Cenerario, Sergio Ilarri, and Sylvain Lecomte. A cooperative reservation protocol for parking spaces in vehicular ad hoc networks. In Proceedings of the 6th International Conference on Mobile Technology, Application and Systems, Mobility '09, pages 30:1–30:8, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-536-9. doi: 10.1145/1710035.1710065. URL http://doi.acm.org/10.1145/1710035.1710065. 31, 92

Dropbox. Your Stuff, Anywhere. http://www.dropbox.com/, 2014. 62

6. REFERENCES

Hossein Falaki, Ratul Mahajan, Srikanth Kandula, Dimitrios Lymberopoulos, Ramesh Govindan, and Deborah Estrin. Diversity in Smartphone Usage. In Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, 2010. 27, 90

Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android Permissions: User Attention, Comprehension, and Behavior. In Proceedings of the Eighth Symposium on Usable Privacy and Security (SOUPS), 2012. 20

Gartner. More than 50 percent of users will use a tablet or smartphone first for all online activities. In http://www.gartner.com/newsroom/id/2939217, 2014a. 1

Gartner. Tablets to overtake pc sales in 2015. In http://techcrunch.com/2014/07/06/gartner-device-shipments-break-2-4b-units-in-2014-tablets 2014b. 1

Aaron Gember, Ashok Anand, and Aditya Akella. A Comparative Study of Handheld and Non-Handheld Traffic in Campus Wi-Fi Networks. In Proceedings of the 12th International Conference on Passive and Active Measurement (PAM), 2011. 90

Google. One account. All of Google. `https://drive.google.com`, 2014. 62

Richard G Guy, John S Heidemann, Wai-Kei Mak, Thomas W Page Jr, Gerald J Popek, Dieter Rothmeier, et al. Implementation of the ficus replicated file system. In USENIX Summer, pages 63–72, 1990. 93

Junxian Huang, Qiang Xu, Birjodh Tiwana, Z. Morley Mao, Ming Zhang, and Paramvir Bahl. Anatomizing Application Performance Differences on Smartphones. In Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys), 2010. 90

International Data Corporation. Worldwide Mobile Phone Growth Expected to Drop to 1.4Despite Continued Growth Of Smartphones. `http://goo.gl/ROLYO`. 5

6. REFERENCES

Sibren Isaacman, Richard Becker, Ramón Cáceres, Margaret Martonosi, James Rowland, Alexander Varshavsky, and Walter Willinger. Human mobility modeling at metropolitan scales. In Proceedings of the 10th international conference on Mobile systems, applications, and services, MobiSys '12, pages 239–252, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1301-8. doi: 10.1145/2307636.2307659. URL http://doi.acm.org/10.1145/2307636.2307659. 1, 5

Matthew Keally, Gang Zhou, Guoliang Xing, Jianxin Wu, and Andrew Pyles. Pbn: towards practical activity recognition using smartphone-based body sensor networks. In Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems, SenSys '11, pages 246–259, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0718-5. doi: 10.1145/2070942.2070968. URL http://doi.acm.org/10.1145/2070942.2070968. 34, 91

Ram Keralapura, Antonio Nucci, Zhi-Li Zhang, and Lixin Gao. Profiling Users in a 3G Network Using Hourglass Co-Clustering. In Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking (MOBICOM), 2010. 90

Donnie H. Kim, Younghun Kim, Deborah Estrin, and Mani B. Srivastava. SensLoc: Sensing Everyday Places and Paths Using Less Energy. In Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys), 2010. 27

Hyojun Kim, Nitin Agrawal, and Cristian Ungureanu. Revisiting Storage for Smartphones. In Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST), 2012. 90

James J Kistler and Mahadev Satyanarayanan. Disconnected operation in the coda file system. ACM Transactions on Computer Systems (TOCS), 10(1): 3–25, 1992. 93

Mikkel Baun Kjaergaard, Sourav Bhattacharya, Henrik Blunck, and Petteri Nurmi. Energy-Efficient Trajectory Tracking for Mobile Devices. In Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys), 2011. 27

6. REFERENCES

Youngki Lee, Younghyun Ju, Chulhong Min, Seungwoo Kang, Inseok Hwang, and Junehwa Song. Comon: cooperative ambience monitoring platform with continuity and benefit awareness. In Proceedings of the 10th international conference on Mobile systems, applications, and services, MobiSys '12, pages 43–56, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1301-8. doi: 10.1145/2307636.2307641. URL http://doi.acm.org/10.1145/2307636.2307641. 1, 5

Rongxing Lu, Xiaodong Lin, Haojin Zhu, and Xuemin Shen. Spark: A new vanet-based smart parking scheme for large parking lots. In INFOCOM 2009, IEEE, pages 1413–1421, 2009. doi: 10.1109/INFCOM.2009.5062057. 31

Time Machine. Time Machine. www.apple.com/support/timemachine, 2014. 93

Gregor Maier, Fabian Schneider, and Anja Feldmann. A First Look at Mobile Hand-Held Device Traffic. In Proceedings of the 11th International Conference on Passive and Active Measurement (PAM), 2010. 90

Ali José Mashtizadeh, Andrea Bittau, Yifeng Frank Huang, and David Mazières. Replication, history, and grafting in the ori file system. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, pages 151–166. ACM, 2013. 62, 94

Suhas Mathur, Tong Jin, Nikhil Kasturirangan, Janani Chandrasekaran, Wenzhi Xue, Marco Gruteser, and Wade Trappe. Parknet: drive-by sensing of road-side parking statistics. In Proceedings of the 8th international conference on Mobile systems, applications, and services, MobiSys '10, pages 123–136, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-985-5. doi: 10.1145/1814433.1814448. URL http://doi.acm.org/10.1145/1814433.1814448. 31, 91

Monsoon. Monsoon Solutions Inc. Power Monitor. http://www.msoon.com/LabEquipment/PowerMonitor/, 2014. 87

Anandatirtha Nandugudi, Anudipa Maiti, Taeyeon Ki, Fatih Bulut, Murat Demirbas, Tevfik Kosar, Chunming Qiao, Steven Y Ko, and Geoffrey Challen. Phonelab: A large programmable smartphone testbed. In Proceedings of First

International Workshop on Sensing and Big Data Mining, pages 1–6. ACM, 2013. 65

Suman Nath. Ace: exploiting correlation for energy-efficient and continuous context sensing. In Proceedings of the 10th international conference on Mobile systems, applications, and services, MobiSys '12, pages 29–42, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1301-8. doi: 10.1145/2307636.2307640. URL http://doi.acm.org/10.1145/2307636.2307640. 1, 5, 20

Sarfraz Nawaz, Christos Efstratiou, and Cecilia Mascolo. Parksense: A smartphone based sensing system for on-street parking. In Proceedings of the 19th Annual International Conference on Mobile Computing & Networking, MobiCom '13, pages 75–86, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1999-7. doi: 10.1145/2500423.2500438. URL http://doi.acm.org/10.1145/2500423.2500438. 93

Bill Nowicki. Nfs: Network file system protocol specification. 1989. 93

Jeongyeup Paek, Joongheon Kim, and Ramesh Govindan. Energy-Efficient Rate-Adaptive GPS-Based Positioning for Smartphones. In Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys), 2010. 27, 28

Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. Where is the Energy Spent Inside My App?: Fine Grained Energy Accounting on Smartphones with Eprof. In Proceedings of the 7th ACM european conference on Computer Systems (EuroSys), 2012. 90

Daniel Peek and Jason Flinn. Ensemblue: Integrating distributed storage and consumer electronics. In Proceedings of the 7th symposium on Operating systems design and implementation, pages 219–232. USENIX Association, 2006. 62, 94

Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. SIGCOMM Comput. Commun. Rev., 33(1):59–64, January 2003. ISSN 0146-4833. doi: 10.1145/774763.774772. URL http://doi.acm.org/10.1145/774763.774772. 12, 89

6. REFERENCES

Feng Qian, Kee Shen Quah, Junxian Huang, Jeffrey Erman, Alexandre Gerber, Zhuoqing Mao, Subhabrata Sen, and Oliver Spatscheck. Web caching on smartphones: ideal vs. reality. In Proceedings of the 10th international conference on Mobile systems, applications, and services, MobiSys '12, pages 127–140, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1301-8. doi: 10.1145/2307636.2307649. URL http://doi.acm.org/10.1145/2307636.2307649. 1, 5

Ahmad Rahmati, Angela Qian, and Lin Zhong. Understanding Human-Battery Interaction on Mobile Phones. In Proceedings of the 9th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI), 2007. 24

Venugopalan Ramasubramanian, Thomas L Rodeheffer, Douglas B Terry, Meg Walraed-Sullivan, Ted Wobber, Catherine C Marshall, and Amin Vahdat. Cimbiosys: A platform for content-based partial replication. In Proceedings of the 6th USENIX symposium on Networked systems design and implementation, pages 261–276, 2009. 94

D. Raychaudhuri, M. Ott, and I. Secker. ORBIT Radio Grid Tested for Evaluation of Next-Generation Wireless Network Protocols. In Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks andA COMmunities (TRIDENTCOM), 2005. 89

Sasank Reddy, Min Mun, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. Using mobile phones to determine transportation modes. ACM Trans. Sen. Netw., 6(2):13:1–13:27, March 2010. ISSN 1550-4859. doi: 10.1145/1689239.1689243. URL http://doi.acm.org/10.1145/1689239.1689243. 34, 91

Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. Journal of the Society for Industrial & Applied Mathematics, 8(2):300–304, 1960. 63

6. REFERENCES

Alex Shye, Benjamin Scholbrock, and Gokhan Memik. Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures. In Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2009. 20, 22, 27, 90

SQLite. SQLite. `https://sqlite.org/`, 2014. 80

Play Store. Google Play. `https://play.google.com/`, 2014. 80

Jacob Strauss, Chris Lesniewski-Laas, Justin Mazzola Paluska, Bryan Ford, Robert Morris, and Frans Kaashoek. Device transparency: a new model for mobile storage. ACM SIGOPS Operating Systems Review, 44(1):5–9, 2010. 94

Arvind Thiagarajan, Lenin Ravindranath, Katrina LaCurts, Samuel Madden, Hari Balakrishnan, Sivan Toledo, and Jakob Eriksson. Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones. In Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, SenSys '09, pages 85–98, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-519-2. doi: 10.1145/1644038.1644048. URL `http://doi.acm.org/10.1145/1644038.1644048`. 93

Arvind Thiagarajan, James Biagioni, Tomas Gerlich, and Jakob Eriksson. Cooperative transit tracking using smart-phones. In Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, SenSys '10, pages 85–98, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0344-6. doi: 10.1145/1869983.1869993. URL `http://doi.acm.org/10.1145/1869983.1869993`. 92

Ionut Trestian, Supranamaya Ranjan, Aleksandar Kuzmanovic, and Antonio Nucci. Measuring Serendipity: Connecting People, Locations and Interests in a Mobile 3G Network. In Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference (IMC), 2009. 90

Ionut Trestian, Supranamaya Ranjan, Aleksandar Kuzmanovic, and Antonio Nucci. Taming the Mobile Data Deluge with Drop Zones. IEEE/ACM Trans. Netw., 20(4):1010–1023, August 2012. ISSN 1063-6692. doi: 10.1109/TNET.2011.2172952. URL `http://dx.doi.org/10.1109/TNET.2011.2172952`. 90

He Wang, Souvik Sen, Ahmed Elgohary, Moustafa Farid, Moustafa Youssef, and Romit Roy Choudhury. No need to war-drive: unsupervised indoor localization. In Proceedings of the 10th international conference on Mobile systems, applications, and services, MobiSys '12, pages 197–210, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1301-8. doi: 10.1145/2307636.2307655. URL http://doi.acm.org/10.1145/2307636.2307655. 1, 5

Yi Wang, Jialiu Lin, Murali Annavaram, Quinn A. Jacobson, Jason Hong, Bhaskar Krishnamachari, and Norman Sadeh. A framework of energy efficient mobile sensing for automatic user state recognition. In Proceedings of the 7th international conference on Mobile systems, applications, and services, MobiSys '09, pages 179–192, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-566-6. doi: 10.1145/1555816.1555835. URL http://doi.acm.org/10.1145/1555816.1555835. 34, 91

Geoffrey Werner-Allen, Patrick Swieskowski, and Matt Welsh. MoteLab: a Wireless Sensor Network Testbed. In Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN), 2005. 12, 89

Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI), 2002. 12, 89

Wikipedia. Galaxy Nexus. http://en.wikipedia.org/wiki/Galaxy_Nexus. 65

Qiang Xu, Jeffrey Erman, Alexandre Gerber, Zhuoqing Mao, Jeffrey Pang, and Shobha Venkataraman. Identifying Diverse Usage Behaviors of Smartphone Apps. In Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference (IMC), 2011. 26, 90

Tingxin Yan, David Chu, Deepak Ganesan, Aman Kansal, and Jie Liu. Fast app launching for mobile devices using predictive user context. In Proceedings of the 10th international conference on Mobile systems, applications, and services,

MobiSys '12, pages 113–126, New York, NY, USA, 2012a. ACM. ISBN 978-1-4503-1301-8. doi: 10.1145/2307636.2307648. URL http://doi.acm.org/10.1145/2307636.2307648. 1, 5

Zhixian Yan, V. Subbaraju, D. Chakraborty, A. Misra, and K. Aberer. Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach. In Wearable Computers (ISWC), 2012 16th International Symposium on, pages 17–24, 2012b. doi: 10.1109/ISWC.2012.23. 91

Jie Yang, Simon Sidhom, Gayathri Chandrasekaran, Tam Vu, Hongbo Liu, Nicolae Cecan, Yingying Chen, Marco Gruteser, and Richard P. Martin. Detecting driver phone use leveraging car speakers. In Proceedings of the 17th annual international conference on Mobile computing and networking, MobiCom '11, pages 97–108, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0492-4. doi: 10.1145/2030613.2030625. URL http://doi.acm.org/10.1145/2030613.2030625. 34, 91

Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES/ISSS), 2010. 18, 90

Pengfei Zhou, Yuanqing Zheng, and Mo Li. How long to wait?: predicting bus arrival time with mobile phone based participatory sensing. In Proceedings of the 10th international conference on Mobile systems, applications, and services, MobiSys '12, pages 379–392, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1301-8. doi: 10.1145/2307636.2307671. URL http://doi.acm.org/10.1145/2307636.2307671. 92