LOCATION PREDICTION FOR CONTEXT-AWARE APPLICATIONS

by

Neal H. Walfield

A dissertation submitted to Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

October, 2016

Intended to be blank.

Abstract

Context-aware applications are programs that are able to improve their performance by adapting to the current conditions, which include the user's behavior, networking conditions, and charging opportunities. In many cases, the users location is an excellent predictor of the context. Thus, by predicting the users future location, we can predict the future conditions.

In this thesis, we develop techniques to identify and predict the user's location over the next 24 hours with a minimum median accuracy of 82%. We start by describing the user study that we conducted, and some salient conclusions from our analysis. These include our observation that cell phones sample the towers in their vicinity, which makes cell towers as-is inappropriate for use as landmarks. Motivated by this observation, we develop two techniques for processing the cell tower traces so that landmarks more closely correspond to locations, and cell tower transitions more closely correspond to user movement. Then, we present our prediction engine, which is based on simple sampling distributions of the form $f(t, c)$, where t is the predicted tower, and c is a set of conditions. The conditions that we considered include the time of the day, the day of the week, the current regime, and the current tower. Our family of algorithms, called TomorrowToday, achieves 89% prediction precision across all prediction trials for predictions 30 minutes in the future. This decreases slowly for predictions further in the future, and levels off for predictions approximately 4 hours in the future, at which point we achieve 82% prediction precision across all prediction trials up to 24 hours in the future. This represents a significant improvement over NextPlace, a well-cited prediction algorithm based on non-linear time series, which achieves appropriately 80% prediction precision (self reported) for predictions 30 minutes in the future, but, unlike our predictors, which try all prediction attempts, NextPlace only attempts 7% of the prediction trials on our data set.

Primary Reader: Christian Grothoff
Secondary Readers: Scott Smith, Matthew Green, John Linwood Griffin

Intended to be blank.

Acknowledgments

The following people had a significant influence on this thesis: Yair Amir Randal Burns, Matt Green, John Linwood Griffin, Thomas Gross, Christian Grothoff, Amitabh Mishra, and David Yarowsky.

I'd also like to thank: Zachary Burwell, Aaron Clauset, Gábor Csárdi, Christian Feuersänger, Laura Graham, Benoît Hervier, Birgit Hünig, Isabel Hünig, Werner Koch, Yves Marcoz, Thomas Perl, Cathy Thornton, and Da Zheng.

Finally, I'd like to apologize to Noam and Ada, who often had to wait while I worked on my "book."

Publications & Coauthors

The user study (chapter 2) has been published as a technical report [125] (no co-authors).

The analysis of the user study (chapter 3) was published at MELT16 [128] (co-authors: John Linwood Griffin and Christian Grothoff).

How to deal with oscillations (chapter 4, co-authors: none), and how to aggregate towers (chapter 5, co-author: Christian Grothoff) was submitted and rejected from MobiCase 2016 [126] (co-author: Christian Grothoff).

The prediction algorithms (chapter 6) is currently under submission to PerCom 2017 [127] (co-author: Christian Grothoff).

The transmission manager (chapter 7) was submitted and rejected from MobiCom 2016 [124] (co-authors: none).

How to fit data to a right censored power law and evaluate the fit is currently being prepared for submission (co-authors: none).

Intended to be blank.

Contents

Intended to be blank.

List of Tables

Intended to be blank.

List of Figures

Listings

Intended to be blank.

Chapter 1

Introduction

1.1 Thesis Statement

Location provides useful information about a user's context, in particular, it provides information related to enviromental conditions, such as network connectivity, and charging opportunities. Online cell-tower trace analysis is able to predict a user's location, and consequently, parts of the user's context, over the next 24 hours with 82% accuracy, on average.

1.2 Motivation

Many applications running on mobile devices can improve their user experience by automatically adapting to the user's behavior and the environmental conditions [48, 80, 82]. A simple example of these so-called *context-aware* applications is a podcatcher that learns the user's behavior and opportunistically prefetches data. Such a podcatcher, having mined the user's access history, and learned that the user typically listens to the hourly news on her commute home, could automatically download the latest episode over the Wi-Fi connection at her workplace before she leaves for the day. To do this, the podcatcher must: recognize associations between locations and resource availability, and locations and user behavior; and, predict the user's location in the near future. Here and more generally, exploiting context means using location to link facts and actions together. In this thesis, we study how to use cell towers as landmarks, and how to predict the user's location in the near future with an eye towards their use in context-aware applications.

There are two main types of context-aware applications: applications whose behavior relies on context to function, and those that use context to enhance

Access	Activity	Watts	Ratio
3G	Play 56 Kbit/s stream	1.00	12.5
Edge	Play 56 Kbit/s stream	0.96	12.0
Wi-Fi	Play 56 Kbit/s stream	0.75	9.3
Local Storage	Play 56 Kbit/s files	0.28	3.5
Local Storage	Play 128 Kbit/s files	0.27	3.4
Local Storage	Play 320 Kbit/s files	0.32	4.0
Wi-Fi	Download at 4.7 Mbit/s	1.23	15.4
Wi-Fi	Download at 1.0 Mbit/s	0.91	11.4
Wi-Fi	Download at 256 Kbit/s	0.76	9.5
None	Idle, Wi-Fi Connection	0.08	1
None	Idle, Wi-Fi Connection, LCD on	0.27	3.4

Table 1.1: Energy used on a Nokia N900 to play MP3s, to download via Wi-Fi, and at idle with the display on and off (the baseline). A full charge has about 18 KWs = 5 Wh. 4.7 Mbit/s is the maximum sustained throughput we observed on the N900; other devices on the same network achieved higher throughput transferring the same data. Measurements were done with a stationary device, which had a strong signal.
Streaming audio over 3G requires more than three times as much energy (1 W) as playing a similarly encoded audio file, which is saved locally (0.28 W). In other words, approximately 0.72 W are required to use the 3G network! This is 3.5 times as much energy as having the display on (which uses approximately 0.2 W).

the user experience. In the former case, context has been used to decide what alerts to show [65]; to predict when the user is returning home in order to turn the heater on [78]; and, to detect nearby friends [89, 118]. In the latter case, context has been used to: automatically fill in the destination of a train schedule; to recommend music [132]; to help tourists [84] and, more generally, provide information about nearby places [117]; to help authenticate users [7]; and, like the podcatcher, to more intelligently schedule resources.

Whereas most of the aforementioned uses of context are nice-to-have conveniences, intelligently scheduling resources could significantly change how people interact with their mobile devices. More intelligently scheduling resources addresses a number of more or less serious problems faced by users, which restrict their behavior:

- **Energy**: In Table 1.1, we show that transferring data over Wi-Fi on the N900 requires 3.5 times as much energy as having the display on, and it is significantly less energy intense to access locally available data than streaming it. Schulman et al. observe that transferring data with a weak signal requires up to six times as much energy as when the signal is strong [106]. And, more generally, transferring data from a fast, nearby access point while stationary requires significantly less energy than transferring data from a congested access-point while traveling underground in a subway, for instance [9, 14, 59, 93, 94, 106]. By exploiting context, it is possible to determine what data the user is likely to use in the near future, and prefetch it when the network connection is good or energy is plentiful.

  Using context, it is also possible to time-shift computations or adapt their quality of service [59]. For instance, indexing, housekeeping, and precomputation can be done when the device is connected to the wall charger, and the display can be dimmed or playback fidelity reduced if the amount of remaining energy does not appear to be sufficient to last until the next charging opportunity.

- **Data allowance**: Over the past several years, cellular carriers in the US have been moving from unlimited data plans to data plans with rather modest data transfer allowances [38, 64, 69, 114]. As far as we can tell, this trend is worldwide. Even assuming that the data allowances are sufficiently large, Trestian et al. found that users avoid transferring data and draining the battery by, e.g., not listening to music [118] for fear of running out of energy or exhausting their data allowance and incurring overage charges [23].

  By time-shifting delay-tolerant data transfers to times with better connectivity, a data allowance can potentially last much longer, and users' confidence that the resources won't be exhausted may be improved leading to less conservative behavior. Ironically, in a world where available computing resources tend to dramatically increase each year, data allowances are following the opposite trend, and are making intelligent scheduling *more* important.

- **Congestion**: Cellular providers have often complained of network congestion, and used it to justify traffic shaping and data caps. AT&T's problem was apparently so bad that they encouraged users to offload traffic to Wi-Fi [12]. By intelligently scheduling delay-tolerant data transfers, users can help reduce congestion by offloading data transfers to other networks with more capacity, and to times when the network is not overloaded without

having to change their behavior. Network operators can even encourage this by pricing data according to demand.

- **Latency**: For an operation to feel instantaneous, it must complete in less than 0.1 seconds [92, Ch. 5]. For anything involving network access, this is hard to achieve most of the time. A study of the round-trip times (RTTs) of packets on a UMTS/HSxPA network in Austria found that less than 1% of packets have an RTT shorter than 0.05 second, and half of packets have an RTT that is longer than 0.1 seconds [102]. Since most network operations involve several round trips, few interactions that access data over the network will feel instantaneous. Accessing data that is saved locally, however, generally will.

  Ensuring that latency is low is not simply a matter of convenience: changes in typical access times fundamentally change the way users interact with their devices. O'Donnell and Draper found that users modify their behavior according to application delays: they do not just wait for the task to complete, they organize their behavior around the delays [96].

  Prefetching data can help reduce latency.

- **Network Coverage**: Although cellular coverage is quite good, it is not ubiquitous. Despite what carriers would have the public believe, the FCC's reserve auction to bring 3G coverage to the 25.2% of the US with unserved, travelled areas [36] is evidence of the contrary. But even in areas that have cellular coverage, network connectivity may not be available due, for instance, to the high cost of roaming (particularly, internationally), or to some other network problems. In their study, Rahmati and Zhong's found that, although users were apparently connected to a cell tower 99% of the time, they were only able to transfer data 80% of the time [9].

  Opportunistically transferring data can help hide this spotty coverage.

- **Centralized Infrastructure**: Applications that assume connectivity is always available, become useless when there is no connectivity. A consequence of this is that cellular providers become essential, because users are dependent on them. Transferring data opportunistically makes users less dependent on cellular providers, which weakens their stranglehold.

Solving these problems is not just a matter of increasing convenience, they are fundamental improvements that change how users interact with their devices [96,118], and the public and cellular providers' relationship. More generally, these changes may lead to *emergent behavior*.

The above is contingent on the fact that data transfers and computations can be anticipated and time-shifted. Recent studies suggest that the use of subscription-based services is common among the heaviest data users [34, 39, 77, 118], Ericsson is convinced this usage will become mainstream [33], and NetFlix is considering allowing users to download content, which an AllFlicks survey found was important or very important to over two-thirds of their respondents [75].

A key component of all of the above solutions is predicting resource availability. Rahmati and Zhong explored how to use cell towers to infer Wi-Fi availability [8, 9] and Ravi examined how to predict charging opportunities [94]. Flinn and Satyanarayanan adjust the QoS based on the next anticipated charging opportunity [59], for instance.

To schedule transfers and understand behavior, the most important piece of context is *location*. We observe that the environment remains relatively static at a given location (whether the user charges the device there, network connectivity, etc.). Further, we conjecture that user behavior (e.g., the types of data that a user accesses) is often correlated with the location. For example, we can imagine a user who watches movies at the gym, and on the way to work usually reads the news.

Thus, being able to determine the current location inexpensively, and being able to predict the user's location in the near future are the foundation of context-aware applications.

1.3 Key Problems

A simple approach to determining the user's location is to use GPS. Unfortunately, GPS is very energy intense, which is one of the main resources that we are trying to conserve. It also doesn't work well indoors or in urban canyons. An alternative approach is to use cell towers. Cell towers have globally unique identifiers, are nearly ubiquitous, and determining a nearby tower doesn't require any energy, since it is already tracked as a byproduct of the modem listening for messages.

Raw cell tower traces have several problems. The most important one is that traces are noisy. We observe in chapter 3 that cell phones appear to sample the towers in their vicinity even if the cell phone is stationary (for example, when connected to a wall charger). This means, in order to use cell towers to identify places and user movement, the traces should first be cleaned by somehow aggregating towers so that 1.) a location like home or work corresponds to a single landmark, and 2.) transitions between landmarks correspond to user movement.

Having a good data set is necessary to both understand how people move, and to validate algorithms that process the cell tower traces and predict the user's location. When we started work on this project, all publicly available cell tower traces that we found were relatively small, and highly biased towards the academic community. As such, we conducted our own user study.

The algorithms need to be designed to not only perform well, but run online, and on the device, i.e., with limited computational power, and minimal energy. These requirements mean that the algorithms have to work, at least initially, with extremely limited data. We impose the on-device requirement due to the highly sensitive nature of location tracks [20, 28, 37, 83]: given the frequency of data breaches [74], many people have become skeptical of trusting third parties with their private data. Thus, even if this requirement is not strictly necessary, it is at least interesting to see the potential performance that can be achieved with this restriction to better decide whether a centralized approach, with its communication and management overhead, is required.

Once building context-aware applications is possible, the question becomes how to convince application developers to adopt the technology. Application developers cannot be expected to implement all of the algorithms on their own. First, there are many developers and this would represent a huge amount of redundant work. But, more importantly, these algorithms must run constantly in the background, and access to the shared resources (energy, data allowance, and storage) needs to be coordinated. This suggests some middleware. The question then is: what does this middleware look like, and to what degree does it and can it support developers?

1.4 Contributions

This thesis makes the following contributions:

- In chapter 2, we present a new, publicly available data set, which we have made available via CRAWDAD.

  Unlike most existing, publicly available data sets, the participants are not just university students or faculty, but our participants are drawn from the larger, world-wide population. In fact, our participants came from every populated continent.

- In chapter 3, we present a detailed analysis of the data set. Interesting results include:

  - Dwell time is distributed according to a power law. Although this has been observed in studies of call data records (CDRs) (e.g., [18]), CDRs typically only include at most a few dozen records per day per individual. We confirm this findings with our high-resolution trace.
  - Traces include so-called *regimes*, which are the large geographic areas within which a user moves on a day-to-day basis. We identify regimes as a potentially useful feature for a location predictor.
  - Devices appear to sample the towers in their area rather than report the main tower responsible for the area. This occurs even when the device is stationary (e.g., when the device is connected to a wall charger). This phenomenon makes raw cell towers less appropriate for use as landmarks due to locations being aliased, and tower transitions not necessarily corresponding to user movement. We saw that one common pattern is the presense of oscillation sequences, which account for more than half of all cell tower visits.

- In chapter 4, we analyze oscillations and present a heuristic for intelligently collapsing them.

  We observe that, because oscillation locations correspond to the primary overlap of two towers, collapsing oscillation sequences can potentially increase the resolution of the data. But, because towers are involved in oscillation sequences with multiple towers, naïvely collapsing them may result in a loss of information.

  Our proposed heuristic is firstly based on our intuition that long alternating sequences are unlikely to arise from user movement (people don't usually move between two locations many times on a regular basis). Based on an examination of the distribution of alternating sequence length, we then observe that tower pairs involved in long oscillation sequences, don't appear to be involved in many user movement sequences. Thus, to classify short alternating sequences, we check if the tower pair has been involved in long alternating sequences in the past.

- In chapter 5, based on the tower sampling hypothesis, we present a new family of algorithms to aggregate towers into places.

  Our cell tower aggregation algorithms are based on the observation that dense subgraphs in the induced cell tower network probably correspond to the user being at a single geographic location, e.g., home, work, park, etc. Based on this, we try using different structures, such as cliques and stars, to identify appropriate aggregations. The complication is that because

our algorithm runs online, we need to strike a balance between quickly aggregating towers, such that they are assigned their final label as soon as possible, and too aggresively aggregating towers, which can result in aggregates covering multiple, disjoint contexts.

To evaluate our algorithms, we use techniques from information theory to see whether resources associated with the individual towers match the resources associated with other towers in the aggregate.

In comparing our algorithms to the related work, we discover that several are unimplementable as described, and that PlaceMap [70] creates similar quality aggregations as our algorithms, but takes longer to assign towers their final label.

- In chapter 6, we develop a family of algorithms for predicting the user's location.

  The predictors all build and evaluate a sampling distribution of the form $f(t, \mathbf{c})$, where t is the tower, and $\mathbf{c}$ is a set of conditions. The conditions that we specifically consider are the time of day, the day of the week, the current regime, and the current tower. We consider how much data is needed for the predictors to be confident, and evaluate the effects of aging the data. In the later case, we don't just keep the last x days of data, but instead the data from the last x days for each primary condition, e.g., regime or current tower. In this way, we keep the most recent data for each location, even if that location has not been visited recently. To our surprise, we discover that aging is not that helpful.

  Our evaluation reveals that the current-tower-aggregate-based predictor is able to correctly predict the user's location in half an hour 89% of the time and comfortable over 80% of the time 2.5 hours in the future. For prediction between 4 hours in the future and 1 day in the future, our final predictor correctly predicts the user's location 82% of the time.

  NextPlace, a similar, well-cited prediction scheme, scores 80% for *attempted* predictions half an hour in the future. However, in our reimplementation and on our data set, NextPlace only attempts about 7% of the prediction trials. (We contacted the authors, but they didn't record the prediction attempts in their evaluation, they can't find the source code for their implementation, and they don't remember what subset of the data set they used. This makes reproducing their results on their data sets extremely difficult.)

- In chapter 7, we design, implement, and evaluate a transmission manager

for scheduling opportunistic data transfers on a mobile device.

Our primary focus is on ensuring the interface is simple for application developers to use. Using our interface, application developers need to do just three things. First, they need to map their data to two abstractions: streams, which encapsulate updates of individual subscriptions, e.g., the RSS feed that contains the list of available podcasts; and objects, which correspond to transferable content, e.g., the podcasts themselves. Our evaluation suggests that in many cases this abstraction is a natural fit. Second, application developers need to implement some upcalls, e.g., update this stream, or download this object. Because these typically correspond to existing functions, this is relatively straightforward. Finally, to allow the transmission manager to predict what data the user will likely want, the application developer needs to indicate when objects are used.

We evaluate the interface by modifying three existing applications, writing a new application, and implementing a helper application that enables opportunistic data transfers for a proprietary application. We used two metrics: whether the proposed changes were acceptable to upstream; and, the amount of required changes and their invasiveness. For all three modified applications, our changes were accepted by the upstream developers. In terms of the required changes, typically just a few hundred lines of code were needed to provide basic or intermediate support for opportunistic transfers. This small number is because most of the required infrastructure already exists, and the implementation primarily linked the transmission manager's callbacks to the right functions. In the end over, $54\,000$ users installed our middleware.

However, as the N900 went out of fashion before we could complete the location-based prefetching, we did not integrate the final version of the prediction engine into our transmission manager.

- In Appendix B, we extend Clauset et al.'s methodology for fitting power laws to data and evaluating their goodness of fit [6] to work with right-censored power laws, which arise when an external process imposes an upper bound on the power law. This was motivated by the observation of such data in our analysis of the cell tower traces, and the speculation that this arises due to diurnal effects.

Intended to be blank.

Chapter 2

User Study

In this chapter, we present a new publicly available dataset of smartphone traces. The dataset consists of environmental and behavioral information from 91 Nokia N900 users. The study ran from September 2011 through October 2013 and the traces cover 21.1 years of time. The logging software tracked the currently connected cell tower, nearby Wi-Fi access points, wireless data use, the battery status, when the user interacted with the device, and the programs she used.

2.1 Overview

We recruited users of Nokia's N900 smartphone to install a logging program. This logging program runs in the background and collects data about the cell tower the device is connected to, periodically performs Wi-Fi scans, records statistics about network connections, logs when the user interacts with the device, stores what programs the user runs and records the battery's status.

We collected data over a 25 month period (September 2011 through October 2013). Participants could join and leave the study at any time. Figure 2.1 shows a cumulative distribution plot of the number of days for which we have data for each participant. Although a few participants uninstalled the logging software after a few days, 80% of the participants let it run for at least a week, half left it run for at least a month, and a third let it run for more than three months.

The participants came from all over the world. Table 2.1 shows a breakdown of participants by their primary country (where they spent the most time) or region. Only the top countries are shown; the remaining participants are aggregated by region.[1] 16 participants traveled to multiple countries.

[1]The complete list is: Iran (12), Russia (10), Germany (9), China (8), Egypt (4), Brazil (3), Finland (3), Poland (3), Syria (3), United Arab Emirates (3), USA (3), Viet Nam (3), India (2), Kazakhstan (2), Malaysia (2), Algeria

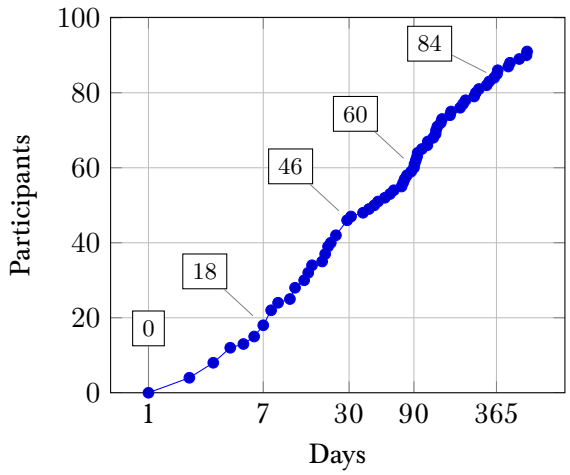


Figure 2.1: CDF of the number of participants for which we have data on at most x days. Note: the x-axis uses a logarithmic scale.

Country	Users	Region	Users
Iran	12	China, Taiwan	9
Russia	10	Southeast Asia	9
Germany	9	Western Europe	9
Egypt	4	Eastern Europe	7
Brazil	3	Middle East	7
USA	3	Africa	3
Kazakhstan	2	Southern Asia	3
Australia	1		

Table 2.1: The participants' main country / region. Regions do not include the explicitly listed countries. Thus, the Middle East does not include Iran.

2.2 Related Work

Several publicly available cell phone traces exist. The most well cited one is the Reality Mining dataset. It consists traces gathered from 100 MIT students and staff over a 9 month period [89]. Drawing participants from a single institu-

(1), Australia (1), Austria (1), Belgium (1), Brunei Darussalam (1), Bulgaria (1), Cambodia (1), Czech (1), France (1), Greece (1), Indonesia (1), Iraq (1), Kenya (1), Lithuania (1), Netherlands (1), Pakistan (1), Philippines (1), Romania (1), Spain (1), Taiwan (1), Tunisia (1) .

tion introduces a strong bias, but was reasonable given a primary focus of the research was studying social relationships.

Rahmati and Zhong, and Yadav et al. also released the traces that they collected in the course of their research [8,9,70]. Like the Reality Mining dataset, the participants in these studies were drawn from the researchers' respective universities (Rice University in the US, and IIIT-D in India). But, these studies did not focus on social relationships, and would have been better served by a more diverse population, which our study provides. In terms of size, the Rahamti and Zhong study included 14 participants, and each trace consisted of at least 3 weeks worth of data. The Yadav dataset included 91 participants, but only 49 of those include at least 3 weeks worth of data. Unfortunately, Yadav et al. only released the raw version of their dataset. Based on our experience, this version requires significant cleaning before it can be used. We contacted the authors about obtaining the version of the dataset that they used in their paper [70], but they were unable to find either the cleaned data, or the code they used to do the cleaning.

The largest publicly available cell phone trace was collected by Wagner et al. and includes traces from 16,000 Android users [112,113]. Its huge size, and significantly smaller bias (anyone with an Android device could participate), make it an excellent starting point for researchers interested in understanding cell phone user, or human mobility. Although the dataset is advertised as being publicly available, getting access actually requires institutional approval. In contrast, our study was designed to protect the privacy of the participants. Consequently, we have institutional review board (IRB) approval for unrestricted publication of the fully anonymized version of the data set.

Mobile operators have also made extremely large cell tower traces available to researchers for analysis. Gonzalez et al. examined a trace of 100,000 users over a 6-month period [18], and Onnela et al. obtained a trace of millions of individuals covering an 18 week period [57]. These traces consist of call data records (CDRs). In particular, those generated when a phone initiates or terminates a call or data session. According to the authors, this results in tens of such entries per participant per day. Consequently, the granularity of this data is much coarser than the data gathered directly on the phones. Further, because these traces are collected by the network operator, the traces are much more restricted in the type of data that they include. For instance, the traces don't include information about available Wi-Fi access points, the device's battery status, etc. Of course, the huge number of users is extremely useful. Thus, these traces should be considered complementary to traces gathered on the device itself.

2.3 Methodology

Before starting the user study, we first obtained approval from our institution review board (IRB). Approval was granted on 18 November 2010 and we were assigned study number 111910.

2.3.1 Recruitment

There were three ways that participants learned about our study: via a recruitment email, which we sent to the primary N900-related mailing lists, via our web page or via the N900's App store. However the participants learned about the study, their decision to participate was their own, which introduces a self-selection bias.

We sent the recruitment email to the official user mailing list (which had approximately 1000 subscribers) and the developer mailing list (1600 subscribers) on 3 October 2011.[2] The email described the project and asked the reader to consider installing the logging software. This request for help was covered by the "Maemo Weekly News," a popular news aggregator for the Maemo community, on 10 October 2011.[3]

We also described the study on our project's homepage.[4] Because we were actively developing and blogging about our middleware to schedule data transfers (the evaluation of which was a primary motivation for the study) and about the programs that we were adapting to use the middleware, some subjects may have found out about the study after reading our blog.

Since the logging software was uploaded to Maemo's App store, some subjects may have discovered the study by browsing the N900's software catalog.

Although we indicated in the recruitment email that participants needed to have their own N900, the consent form explained that users could borrow a device. Four people asked to borrow a device and we provided a device to each of them.

2.3.2 Installation

There were two main ways a potential participant could install the software: she could follow a link in the recruitment email or on our web site, which started the App store program and selected our logging program for installation, or she could directly select the logging program from the App store's catalog.

[2] http://lists.maemo.org/pipermail/maemo-developers/2011-October/028630.html
[3] http://www.mwkn.net/2011/41/front.html
[4] http://hssl.cs.jhu.edu/~neal/woodchuck

Before installing the software, the installer presented the potential participant with a consent form, which provided information about the study including what data would be collected, how it would be used, and how it would be handled as well as the participants' rights, in particular, their right to withdraw from the study.

To continue the installation, the potential participant had to click a button labeled "Agree" to indicate their content. The consent form prominently indicated that if the reader was a child or did not consent to the terms, he should click on the "Cancel" button. In this case, the installation was aborted. Because this is input on the device, we don't have any information about how many users chose not to grant their consent. However, several individuals approached us and indicated that although the study sounded worthwhile, they would not participate, because the data was being uploaded to a server in the United States.

Once the logging program was installed, the participant no longer needed to interact with it: the program ran in the background and started automatically.

If a participant decided to withdraw from the study, that person only needed to uninstall the program. This immediately stopped the logging program and deleted any logging-related data still on the device without uploading it.

2.3.3 Logging Software

Once the logging software was installed, it ran in the background and quietly collected the relevant information. Concretely, we recorded the currently connected cell tower and its signal strength, nearby Wi-Fi access points and their signal strength, statistics about wireless data connections (both Wi-Fi and cellular), when the user interacted with the device, the programs the user ran, the battery level and system events (i.e., when the device started or shut down).

In addition to the aforementioned data, the logging software also collected the files that programs accessed. We don't include this in the dataset, because it is not possible to anonymize it without significantly compromising the usefulness of the data.

Data Collection

Implementation-wise, most of the collected data was harvested by subscribing to event sources. Specifically, the logging program listened for messages emitted from specific D-Bus addresses. The publish-subscribe model ensured that the program was only woken up when something interesting occurred. This style of logging (as opposed to periodic polling) conserves energy as it never forces the device to wake up nor to run code when there is nothing useful to do. On the

N900, there are event sources for cell tower status changes, Wi-Fi scan results, battery level changes, etc.

The only information that we proactively solicited was periodic Wi-Fi scan results. Since the system does not constantly scan for Wi-Fi access points, we set up a timer to ensure that we obtained Wi-Fi scan results *at least* approximately every 3 hours (since the logging program listens for scan result events, it still obtained scan results when the user or the system initiated a scan). We chose a 3 hour period as a compromise between higher resolution results and increased battery load: we observed that more frequent scanning had a noticeable impact on the battery. Since we wanted to avoid the logging software changing the user's experience (and thereby influencing the user's behavior), we conservatively opted for the 3-hour interval.

To determine what programs were run, we listened for D-Bus registrations. According to the developer guide for the N900, every program should register on the D-Bus. All programs don't necessarily do this. As such, we miss some programs.

When the logging program detects a new program, it connects to it using the `ptrace` debugging facilities and dynamically instruments the binary by inserting breakpoints just before it opens or closes a file as well as just before it starts a new process (this can be thought of as a *very* lightweight version of Valgrind [91]). This instrumentation was necessary, because the normal debugging facilities introduced too much overhead: normally, `ptrace` reflects every system call to the process's debugger, which significantly slowed many programs down. Instrumenting the binary to only invoke the debugger for relevant events, which, in our case, are used relatively infrequently, largely mitigates this problem. In particular, using the normal `ptrace` facilities to intercept every system call made it impossible to watch videos whereas when we instrumented the binary, we did not notice any degradation in playback.

Logging

The logging program stored the data in SQLite 3 databases until it could be uploaded.

Since we had a small number of event types with known components, we used a schema in which each event type had its own table and each piece of information had its own column. We also saved some debugging information. Because this information is, at a high level, unstructured, we formatted each event as a simple string and saved it in a catchall table. Automatically analyzing this data is more error prone, but the descriptive text was normally sufficient to unambiguously identify the events that we were interested in.

In addition to the event's data, we also included a time stamp and a sequence number. The sequence number is a per-table monotonically increasing counter (maintained by SQLite), which allows us to trivially order the entries within a table.

Because SQLite uses a global lock, we used a different database for each type of event to reduce contention (the logger was multi-threaded) and increase fault isolation. For instance, if we forgot to close a transaction, updates to other tables wouldn't be blocked.

While testing, we observed that we sometimes had many events per second. This proved to be too fast for SQLite to handle due to its update algorithm, particularly given the speed of the underlying storage. To avoid this problem, the SQLite developers recommend grouping multiple updates into a single transaction, which amortizes the associated overhead [110]. To implement this, we used a 64 KByte buffer per thread and database and appended the SQL code to the buffer. On insertion, we would first check if the buffer was full or if it contained data that was more than five minutes old. If so, we would flush the buffer in a single transaction. A consequence of buffering the events is that the ordering imposed by the sequences numbers is now only per thread. In practice, this only impacts the accessed files, which aren't included in the dataset.

When we detected that the device was shutting down, we would switch to a synchronous mode of operation and always flush the buffer after adding data to it. Unfortunately, we didn't implement a mechanism to flush buffers from another thread. As such, if a given event source didn't have any events after the logger received the shutdown notification, we would lose any buffered data. We would also lose any buffered data if the system crashed or the logging program crashed. Examining the debugging logs, we observed just a few dozen instances of the logging program crashing.

On-Device Anonymization

In case the collected data came into the wrong hands, we partially anonymized location-related data on the device itself. In particular, we hashed sensitive data after combining it with a secret device-local salt. We applied this methodology to cell tower identifiers (CIs) and location area codes (LACs) as well as to Wi-Fi access point SSIDs, network ids and station ids.

Unfortunately, there was an aliasing bug in the code that converted the hash value to an ASCII-based encoding, which was more easily inserted into the database. The result is that only 12 bits of the 128-bit hash were actually saved. Further, because the selected encoding used a 62 character alphabet (just numbers and letters) to encode each 6-bit block of data, 2 letters of the alphabet were

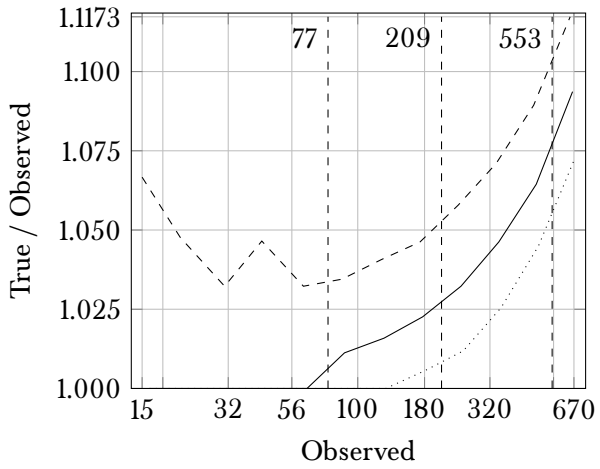


Figure 2.2: The number of observed classes vs. the number of true classes divided by the number of observed classes assuming we have 3968 classes that are equally likely. If we observed, say, 553 values, then we expect there to be approximately 597 actual values ($= 553 \cdot 1.08$) or 44 aliased pairs.

reused. This further reduced the namespace to span just 3968 unique values. This bug probably wasn't caught during testing because thousands of different values is enough to appear random.

Fortuitously, this error has at most a minor impact on the results: the amount of aliasing that occurred is minimal. Consider the cell tower identifiers. Both the CIs and LACs are 16-bit identifiers [2, Ch. 4]. Thus, there are at most 65,536 different values. But, even if all of the values are assigned in a given network or location area, a participant doesn't necessarily visit them all. In fact, the largest number of unique CIs observed by any user in a given location area (i.e., any given MCC, MNC and LAC tuple) is just 553. Similarly, the largest number of unique LACs observed by any user in a given network (i.e., any given MCC and MNC pair) is just 187. These values drop very quickly, as can be seen in Figure 2.2. For instance, at most 77 cell towers are observed in 95% of the location areas.

Using a simple simulation, we computed the distribution of the number of observed classes for different numbers of true classes. That is, if we know that the user visits, say, 100 cell towers in a given location area, we draw 100 values from a uniform distribution (without replacement) over the discrete numbers ranging from 1 to 3968 and count the number of unique values. Repeating this many times, we approximate the distribution of the number of observed CIs.

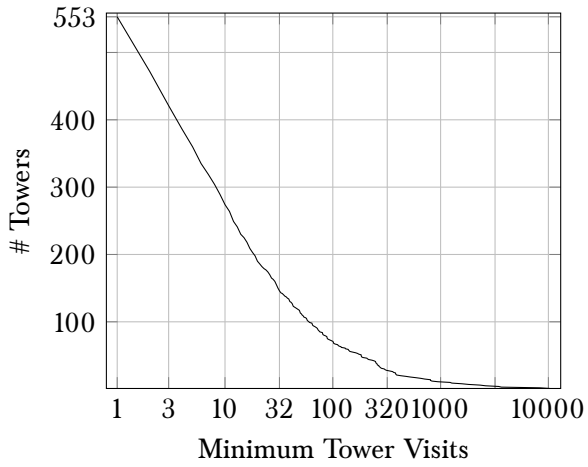


Figure 2.3: CCDF of the number of visits to each tower in the location area with the largest number of observed CIs. (The user spent 331.1 days in the location area.)

Figure 2.2 graphically depicts the results for different relevant values. The x axis is the number of observed classes and the y axis is the ratio of the true classes to the observed classes. Because aliasing only reduces the number, this ratio is always at least one. For each value of true classes, we conducted a million simulations. The three plots, from bottom to top, show the 2.5%, median and 97.5% quantiles. Thus, if we consider observing 553 cell towers, then 95% of the time there are between 1.06 and 1.10 times as many cell towers (i.e., between 586 to 608) actually visited as observed. In other words, 95% of the time between 33 and 55 pairs of identifiers are aliased. This example corresponds to the worse case. If we have 100 classes, which is still a lot, then 95% of the time there will be 0 to 3 aliased pairs.

This problem is even less severe when realizing that most towers are only visited a few times. Figure 2.3 shows the number of times each cell tower is visited for the cell towers in the location area with the most observed CIs. A quarter of the towers are visited at most 3 times and half of the towers visited are visited at most 9 times. Thus, the probability that two significant cell towers are aliased is even lower. Note: The long tail present in this distribution is typical for the distribution of visits in other location areas as well as when considering all of the cell towers a user visits.

The aliasing problem doesn't really impact the Wi-Fi results. Although Wi-Fi identifiers are also aliased, they are not guaranteed to be unique anyways. Thus,

any analysis of this type of data already has to deal with aliasing. This bug just increases the amount of aliasing a bit.

As a final note, the published data set does not include the hashed values. These were further anonymized by renumbering the identifiers so as to prevent a preimage attack. Concretely, for each user and each attribute (CI, etc.), we sorted the identifiers according to the first time that they were first observed, and then numbered them.

2.3.4 Uploading Data

The logging program tried to upload data approximately every 24 hours. To avoid negatively impacting the user or the user's experience, an upload was only attempted if there was a Wi-Fi connection (i.e., the logging program didn't try to connect to the Internet on its own) and the user was idle for at least 5 minutes. We chose to only upload over a Wi-Fi connection, because Wi-Fi is normally not charged by volume (or rather, any volume restrictions are orders of magnitude larger than a typical upload) whereas cellular data can be very expensive.

To upload data, the logging program established an HTTPS connection to our server and verified the server's identity using a certificate included with the logging program. Checking the certificate avoids accidentally uploading the data to the wrong server. Using a certificate provided at install time as opposed to using a central authority is essential to preserving the user's privacy in the workplace: employers often configure devices to use a fake certificate authority in order to perform a man-in-the-middle attack on the any HTTPS connections.

To be able to associate uploads from the same participant, the logging software generated a random identifier and included this with each upload.

After an upload completed successfully, the data that had been uploaded was deleted from the device.

2.3.5 Cleaning the Data

The collected data required four main cleanups before it could be analyzed: we needed to merge and sort the events; we needed to adjust the time zones; we needed to correct the time stamps; and, we needed to fill in the SSIDs for some Wi-Fi connections. There were also a number of minor issues. These minor problems were typically straightforward to address, but not immediately obvious, or required a fair of amount of code to workaround. We first discuss the four main issues and then present a few of the minor issues to give a flavor for the type of corrections that we performed.

Ordering the Events

In section 2.3.3, we described how the logging software saved events. Of particular relevance here is that there is a different table for each event and each event includes a time stamp and a sequence number. Thus, to create a global stream of events we need to merge all of the tables. Unfortunately, since the sequence numbers are per-table, we need to use the time stamps to globally order the events.

On the surface, ordering the events according to the time stamps appears to be straightforward. Unfortunately, the N900 de facto relied on the user to manage the clock. This is due to a confluence of two problems. First, the clock's backup battery was of very low quality and died after a few months. After that point, whenever the main battery died (or was removed), the clock was reset to 1 January 2009, 6:00. (On boot, the user was notified of this and prompted to enter the correct time.) Second, although the N900 included an option to synchronize the local clock with the network operator's clock using the network identity and time zone (NITZ) protocol [130], this did not work reliably. Ignoring the fact that GSM does not require network operators to support NITZ, the N900 appears to have a bug in its implementation of the protocol: there are a number of complaints on the Maemo forums that the N900's clock synchronization option doesn't work on a given network even though other Nokia phones on the same network are able to use this information. The practical result is that the clock is often set incorrectly or just left in its default state. This is probably because the user doesn't know the exact date and time or is simply in a rush.

To work around these problems, we first broke each ordered stream of events into groups that probably don't include a clock change. This reduced the problem from having to correct hundreds of thousands of log entries to having to correct a few dozen groups of entries (since a group of entries doesn't include a clock change, the same correction must be applied to all entries in a group). Clock changes tended to be easy to identify. If an event had an earlier time stamp than the preceding event (a *backwards time warp*), then the clock or time zone was clearly changed between the two events. Similarly, if there is a large gap between two events, then there may have been a time change between the two events. This latter criterion is based on the observation that we typically have multiple events per minute. However, if the device is not moving, e.g., when the user is sleeping, then it is possible that there won't be any events for a few hours. Thus, this test results in some false positives, but it catches many true positives (real clock changes). We also looked for time warps present in event stream, but missing in another. Finally, we examined the results and manually identified missing break points.

Having built the groups, we then took two streams and aligned them. We used two main tools to perform the alignment: we used correlations between streams and the degree of temporal overlap. There was also the natural restriction that the internal ordering of the individual streams be preserved.

There were a number of different types of correlations between the event types. Some information we not only explicitly tracked, but we also wrote to the debugging log. For instance, when we got network scan results, the logging program first noted the results in the debug log and then wrote the actual scan results to the appropriate table. Some events naturally occurred together. For instance, to establish a network connection, the connection manager is used. When started, the program initiates a scan so that it can show the available networks to the user. Thus, the establishment of a network connection is normally immediately preceded by a network scan.

Having merged two streams, we repeated the process with the result and another stream. This was done until we had merged all the streams. This approach did a fairly good job of merging the streams, but it was not perfect. Again, we manually inspected the results, added hints and reran the algorithm. We iterated this process until we were satisfied that the merge was a reasonable approximation of the truth.

Part of the difficulty is that some event types don't continuously have events. Wi-Fi scans and battery status updates, for instance, tend to occur with a particular frequently. Connection statistics, however, are bursty: the logging program only collects statistics when there is actually a network connection. Thus, if the user's battery dies and she sets the time of day correctly, but the date one day in the past, we may have two perfectly reasonable fits.

Fixing Time Zones

Having merged the event streams, we next fixed the time zone. Although the current time zone is available as part of the NITZ data, as discussed in section 2.3.5, this information was effectively ignored by the N900. As such, the user had to set the time zone manually. Although the user is prompted to set the time zone when configuring the time, we found it was often set incorrectly. We know this, because we know the country (from the cell tower entries) and we can easily determine the set of valid time zones for each country using Wikipedia or `www.timeanddate.com`, for example. Some thought reveals that this is not that surprising: although most people are negatively impacted when the time or date is wrong, only people who travel are significantly effected by a misconfigured time zone. Thus, configuring it correctly is for many a waste of time.

As before, we started by breaking up the stream of events into groups. In

addition to using backwards time warps and large gaps as group boundaries, we also broke the streams at other possible clock changes. First, we introduced a group boundary when the configured time zone changed or the user changed countries. Second, we considered internal constraints. Since we recorded the device's uptime both when the logging program started and when the device shut down, we could quickly tell if there had been a clock change in the intervening entries. Similarly, we collected connection statistics every five minutes. As such, if two connection statistics were 65 minutes apart, then the clock was probably advanced by an hour. Identifying where to split the entries in these cases was typically straightforward (there was normally a sufficiently large gap), however, occasionally some manual intervention was needed.

To determine the probable time zone, we made a number of inferences and used some heuristics. First, many countries only have a single standard time zone. This is the case in most of Europe, but also large countries, such as China, only have a single standard time zone. In these cases, the time zone is obvious. We then looked at the uploads. If the difference between the local time in local time and the server time in UTC is a valid time zone (within 15 minutes) for the user's current location, then the difference probably corresponds to the time zone (particularly if it matches the configured time zone).

To fill in the time zone for the remaining groups, we propagated the known time zones via cell towers and Wi-Fi access points. This is reasonable, because nearly all cell towers and most Wi-Fi access points are not mobile and, as such, their standard time zone remains constant. Although our data includes a few mobile Wi-Fi access points that are used in multiple time zones, they tend to be obvious due to the conflicts that result when they are used to propagate time zones. We simply ignore these mobile APs.

Sometimes, after completing this process, we still have some groups that are missing a time zone. In these cases, we fall back to the alleged time zone, if it is reasonable. Occasionally, we manually set a time zone. For instance, user d58 is in Brazil and set the time zone to UTC+1, which is impossible (Brazil's time zones are UTC-2 though UTC-5). We use an educated guess of UTC-3 as the standard time zone, which is the most prevalent time zone in Brazil. Even if this is wrong, it's only off by at most two hours, which is still reasonable.

Correcting Time Stamps

With the correct time zones, we could then correct the time stamps. Working in UTC was essential, because some backwards time warps are expected and legitimate in local time. For instance, changing from DST to standard time results in an apparent time warp as can moving between time zones. These jumps

disappear in UTC. Working in UTC was also useful, because we could meaningfully compare the time stamps saved on our server with their corresponding time stamps saved on the device: the difference between them is the required correction! Unfortunately, not all entries were near an upload. However, a few additional observations were sufficient to correct most of the entries with high confidence.

Given the groups that we created when correcting the time zones, we first corrected those groups that contained an upload, as just described. We then used internal constraints (uptime and connection statistics) to join groups. If one of the groups was already corrected, then we moved the other one. For instance, if groups a and b are joined by connection statistics and b includes an upload, then we moved a such that the last connection statistic entry in a was 5 minutes before the first connection statistics entry in b. If the constraint spanned multiple groups, the intervening groups typically fit tightly between the two groups and their corrections were thus obvious. In the few cases that the intervening groups did not fit tightly, the context suggested a reasonable correction, which we applied manually. Next, we identified clock resets that didn't end in a system shutdown. Many users dismissed the prompt to set the time after a clock reset and instead waited a while (likely until it was more convenient). In these case, we shifted the earlier group to about the later one.

The remaining uncorrected groups fell into two categories: those whose time was plausible given the surrounding groups and those whose time was implausible given the surrounding groups. If the time was plausible, then we assumed that the time was correct. If the time was implausible, oftentimes there was just a single reasonable correction forced by the surrounding groups. Consider three groups, a, b and c. If a and c are 12 hours apart and b spans just under 12 hours, then the correction is obvious within a few minutes. Occasionally, there is a significant amount of slack between the two groups. This primarily happens when the battery dies, the users waits a few hours before recharging the device and turning it back on, and the user doesn't bother to set the clock before the device again runs out of battery power. In this case, the best we can do is guess the time. We mark these groups of entries as floating, which we discuss in the next section when detailing time stamps. We generally chose not to discard these entries since they still contain some valuable information, such as, what programs were run. And, although the absolute time is not correct, the relative time is. That is, the time between the entries is correct. Thus, if we are interested in knowing how long the user stayed at a particular cell tower and we don't care about the actual time of day, these entries are useful. We guessed 0.6% of entries' time stamps.

After letting this algorithm run, we examined the results. Two types of man-

ual corrections sometimes presented themselves. First, sometimes we needed to introduce another group boundary. Other times, the context made the correction obvious. A primary example of these is when the user sets the time correctly, but not the date. (User 020 was prone to this, for instance.) We didn't find a straightforward way to identify such cases algorithmically. As such, when we recognized these cases, we just manually corrected them. Another case that proved problematic to handle were double time corrections: sometimes, a user would correct the time, press save and then correct the date and then save again. The intervening entries with the correct time and wrong date would form their own group. This sometimes caused problems when the algorithm used the internal constraints to correct the groups. As such, we just corrected these occurrences manually. After introducing any required manual corrections, we reran the algorithm and again checked the results.

Filling in Missing Access Points

Due to a bug, the logging program only saved the SSID of the connected Wi-Fi AP for the first Wi-Fi connection. Fortunately, the connection statistics also included the operating system's connection identifier (CID) for the wireless network. Thus, for those Wi-Fi networks for which at least one connection had a known SSID (which account for 426 of the 616 networks the users connected to), we could simply propagate the AP to the other connections. For 28 of the networks, we could unambiguously determine the correct AP from the Wi-Fi scan results that were collected when the connections were established: the intersection consisted of just a single entry. This left 162 networks without a known AP. To cleanup these entries, we again computed the intersection of the Wi-Fi scan results near the connections and simply took the AP with the consistently strongest signal across all connections. Having to guess the AP for so many of the networks sounds bad, however, these 162 networks represent just 396 of the total 26970 network connections. Moreover, the user connected to more than half of these networks (92) once.

Minor Issues

We encountered a number of minor issues when analyzing the data. We present a few of them here to give a flavor for the types of minor cleanups that we performed. Some more are discussed in the context of the dataset description in Section 2.4.

When the phone was not shut down correctly, because, say, the battery ran out of power, the logging program wouldn't write a system shutdown event to the

log. We identified missing shutdown events by looking for system start events that were not preceded by a shutdown event. Because the system start event is not necessarily the very first event recorded after booting (but is certainly within a few seconds of that event), we used the following heuristic to identify the entry corresponding to the real system start and then inserted the system shutdown event just before it. First, we looked for any backwards time warps in the last 30 entries. If so, we assumed the first event is just after the last backwards time warp. Otherwise, we assumed the first entry is just after the first gap preceding the system start event that is larger than 10 seconds.

Occasionally, due to unclear reasons, a system start event would be missing. We applied a similar procedure as above to insert these events.

When the phone loses its connection to the cellular network, it sends an event that looks like a cell tower update event. In this event, the signal strength and the mobile country code (MCC) are set to 0. The remaining fields, including the cell identity (CID), however, appear to contain random data. We set these fields to 0.

We propagated some generally useful information to all entries. In particular, the cell tower information includes the current country and only the debugging entries include the currently configured time zone (due to an oversight). This information is useful in other contexts. To propagate it, we first considered all adjacent entries with the same information (country or time zone, respectively) and copied it to the intervening entries. For adjacent entries with different information, we looked for a backwards time warp or a large gap. These usually indicated a country or time zone change. Occasionally, manual hints were necessary.

2.4 Dataset Description

Our dataset consists of 91 users. Instead of numbering the users sequentially, we used a hash of a random number. (The first three characters of which are sufficient to uniquely identify each user.) We found this system of mixed letters and numbers easier to work with than a simple list of sequential numbers. This ease of use was perhaps due to the greater variety in the names.

The cleaned dataset consists of one file per participant in comma-separated values (CSV) format. The files are named after the user. Since the CSV format has a number of variations, a few remarks are in order. The first line of each file is a header naming the columns. Each line corresponds to a single record. The rows don't include a row name. Since the program arguments occasionally include double quotes, we needed to choose a way to encode them. We used the

double quote convention (rather than using a backslash to escape them). Thus, the string "`foo "bar" bam`" would be encoded as "`""foo ""bar"" bam""`".

We now consider each column grouped by event type. If a column is not relevant for a particular event, its value is set to "?".

2.4.1 Metadata

type: The event type. One of `debug`, `cell`, `wifi.scan`, `connection.stats`, `user.activity`, `process`, `service`, `battery` or `system`.

oid: The per-table sequence number.

uploaded: When the entry was uploaded. Encoded as the number of seconds since the Unix epoch.

country: The current country as a human-readable string. This is inferred from the MCC (mobile country code), which is part of the location area information.

2.4.2 Time Stamps

ts, year, yday, hour, min, sec: The entry's corrected time stamp in UTC. `ts` is the time in seconds since the Unix epoch. `year`, etc. don't provide any additional information and are provided as a convenience. Note: `yday` is the day of the year where 1 January is day 0.

ts.guessed: If we were unable to approximate an entry's true time stamp within four hours (see section 2.3.5), then conclusions drawn from the time stamp should be handled with particular care. We designate such entries by setting this field to a non-zero value. Entries with the same `ts.guess` value are internally consistent (the time between the events is correct).

tz, tz.standard: The current timezone (in hours relative to UTC). `tz` is the timezone that is in effect in the area; `tz.standard` is the standard timezone in the area (i.e., ignoring the effects of daylight saving time).

ts.orig: The uncorrected time stamp encoded as seconds since the Unix epoch.

tz.alleged: The user configured, local timezone.

space.before: Occasionally, we deleted some entries belonging to a very short boot without a clock correction and without cell towers. This is the amount of space these entries took up. (In total, we delete entries spanning 4.3 hours across all users.)

2.4.3 Debug Events

Most debug entries don't actually contain much information. Instead they serve to help merge the various streams.

subtype: The type of debugging message:
`started`: The system just started. This is always the first event in a boot and thus a good anchor when looking for the span of a boot. `shutdown`: The system is shutting down. This is not necessarily the last entry in a boot. Instead, consider the entry preceding the following `debug:start` entry.
`uptime`: The device's current uptime.
`network-scan-result`: We just got a network scan result. `network-scan-result-empty`: We just got an empty scan result.
`new-device`: A network device was brought up. (`new-device` indicates the device's name.) See section 2.3.5.
`network-established`, `network-disconnected`: A network connection was established or disconnected.

2.4.4 Cell Towers

We collected three types of information about the currently connected cell tower: the tower's cell identifier, the current signal strength and changes in GPRS availability. Initially, we had hoped to collect information about neighboring towers, however, we later learned from Nokia developers that the variation of the ISI modem used in the N900 does not export this information.

mcc, network, lac, cell.id: The currently connected cell tower (`network` corresponds to the mobile network code, MNC, and `cell.id` correspond to the cell identifier, CI). If the device is not connected to the network, all of these will be 0. Normally, the values of `cell.id` can be reused across location areas. With the exception of the MCC, we renumber these fields to increase the degree of anonymization and to make them unique (on a per-user basis), which makes analysis of the data easier.

Note: because we renumber before we correct invalid entries (those for which `mcc` is 0), some cell ids may not be used.

signal.strength.dbm: The signal strength in dBm (Decibel-milliwatts). This ranges from -116 to -3. 0 means no signal.

signal.strength.normalized: A human-readable version of the signal strength ranging from 0 to 100. 0 means no signal.

gprs.availability: Whether packet data is available. 0 means that packet data is available. -1 means the current status is unknown. All other values mean

that GPRS has been suspended or is not available. The value is the reason: 3 means detached; 5 means there is no coverage; 7 means the GPRS connection is suspended due to call or SMS signaling; 8 means a call is active; 9 means a routing area update is in progress; and, 10 means a location update is in progress.

2.4.5 Wi-Fi Scans

Each Wi-Fi scan refers to a single scan result. As such, the fields (except `aps`) are formatted as semi-colon separated list in which the n^{th} entry in the each list refers to the same access point.

ap, station.id, network.id, network.type: The SSIDs, station IDs, network IDs and network types (either `WLAN_INFRA` or `WLAN_ADHOC`) of the visible access points.

To increase the degree of anonymization, the APs, station IDs and network IDs have been renumbered.

Note: Once the phone has connected to an ad-hoc network, the network always appears as being available until it is manually removed by the user. However, if no other device is in the vicinity is connected to the ad-hoc network, then the signal strength will be 0.

signal.strength.dbm: The signal strengths in dBm. This ranges from -98 to -3. 0 means no signal.

signal.strength.normalized: A human-readable version of the signal strength ranging from 0 to 100. 0 means no signal.

aps: The number of visible access points. (This is the same as the length of the lists.)

2.4.6 Connection Statistics

state: Either `ESTABLISHED`, `STATS` or `DISCONNECTED` and designating either a new network connection, connection statistics or a torn down network connection, respectively.

ap, network.id, station.id, cid: The SSID, network ID, station ID and connection ID of the current connection. The first two values identify a network; the next value identifies a single station within a network; and, the last value is a device-local name for the network, which is assigned by the N900 the first time the device connects to the network.

Note: the CID identifies a network and not a station. Thus, in a corporate or university environment, there will often be many APs with the same SSID and network ID (and, consequently, CID), but each will have a different station ID.

The AP and CID are set for all `connection.stat` entries, however, see the note in section 2.3.5. The network.id and station.id are only set for the `ESTABLISHED` entries and are not actually collected on the device, but are inferred from the scan results collected at the time of the connect. This is provided as a matter of convenience.

ap.guessed: As described in section 2.3.5, we had to guess some APs identifiers based on the Wi-Fi scans. If this is 0, then we didn't guess; if it is 1, then the intersection of the Wi-Fi scan results collected before the connections to this network indicate that there is only one common AP; if it is 2, then there were multiple possibilities and we took the strongest.

medium: Either `wifi` or `cellular`.

iface: The network interface that was used. There are three variants: `wlann`, `gprsn` and `tether`. The last refers to when the N900 is used as a modem by another device either over USB or Bluetooth.

ip4, gwip4, gwmac: The aligned IP address and the IP and MAC address of the network's gateway. Although we also collected IPv6 information, it appears that IPv6 was not used by any of the participants.

Private IP addresses (e.g., 10.0.0.0/24) [21] were passed through; public IP addresses were renumbered. MAC addresses were also remapped to numbers.

rx, tx: The number of bytes received / transferred.

2.4.7 User Activity

There are two ways the user can become idle: either the user explicitly locks the screen (which can be done using a physical button on the N900) or the device is automatically locked after a short time out (this is user configurable, but probably set to 1 or 2 minutes). Unfortunately, there is no way to distinguish these.

new.state: `active` when the user becomes active; `idle` when the user becomes idle.

2.4.8 Programs

To determine when programs started we listened on the D-Bus for name registrations. According to the developer documentation, all applications should register on the D-Bus and appear to. When we detected a new name registration, we looked up its PID and connected to it using ptrace as discussed in section 2.3.3. We monitored it for fork events and traced any children.

Since the program names and arguments and the service names can contain sensitive information, we carefully went through the strings and identified potentially personally revealing data (there were 526 unique values for `exe`, 677 for `arg0`, 2263 for `arg1` and 556 for `dbus` after censoring). Some personal information was obvious, such as D-Bus names that include a user name. (This convention was used by some instant messaging programs.) Some programs included a long random string. Since we were not sure whether it was local to the device, we simply censored it. For the program arguments, we concealed strings that appeared to be IP addresses, telephone numbers, web addresses or file names. In each cases, we replaced the strings with `xxx`.

Note: When `exe` is `maemo-launcher`, the user didn't start the program. `maemo-launcher` is used to preload a program. It tells a program to set up its internal state up to showing any windows. It then forces it to swap. When the user subsequently starts the program, it appears to start very fast. The result is that we don't actually know when the user starts such programs.

Services

pid: The PID of the program.

exe, arg0, arg1: The executable and the first two arguments to the program (per `/proc/PID/{exe,cmdline}`). `exe` and `arg0` are often the same, however, `arg0` is how the user invoked the program, which may be a relative path. `arg0` may be changed by the program. We include the first real argument to the program to help when dealing with interpreters, such as, Python. In this case, `exe` and `arg0` both normally specify the Python interpreter, but what we are really interested in is what program is being interpreted. In these cases, `arg1` contains the actual program name.

dbus.name: The D-Bus name that the program registered. Note: a program may register multiple names.

running: `started` or `stopped` indicating whether the program registered or released the name.

Processes

exe, arg0, arg1: As above.

event: Either `thread traced` or `thread exited`.

pid: The pid.

tid: The thread's TID. On Linux, the TID of the first thread in a program is the same as its PID.

parent.pid: The pid of the program that created this program or thread. If this is a new thread, then `parent.pid` will normally match `pid`.

exit.code: The thread's exit code. Only set when `event` is `thread exited`.

attach.error: Set if an error occurs attaching to a thread. This can happen if the program is already being ptraced.

2.4.9 Battery Status

is.charging, is.discharging: Whether the battery is charging or discharging. Note: if the device is connected to power, but the power is only sufficient to power the device (and not also charge the battery), then both `is.charging` and `is.discharging` may be `FALSE`.

charger: The type of charger that is connect. This may be `wall`, `usb`, `unknown` or `none`.

voltage, mah: The battery's voltage and charge.

2.4.10 System Events

status: The type of event, either `started`, `crash-recovery`, `shutdown` or `reboot`. `crash-recovery` means that the logging program was restarted after crashing. This condition was detected *a posteriori*.

Note: When looking for the first or last entry in a boot, it is better to use the `debug:started` entry as a reference point, because it is guaranteed to be the first entry in a boot.

uptime: The system's uptime (per `/proc/uptime`).

uptime.guessed: Occasionally, when we insert a missing `system` event we are able to confidently guess its uptime. In these cases, we set the `uptime` and we set this field to `TRUE`. (Otherwise, it is `FALSE`.)

2.5 Conclusion

We've presented a new dataset from smartphones. In collecting this data, we took great care to protect the privacy of our participants. In particular, we anonymized a large amount of data on the device. Unfortunately, this is a trade-off: in protecting the user's device, we lost access to a fair amount of ground truth. For instance, the geographic coordinates of most cell towers is known, thus it would have been possible to identify the approximate location of the user, and the actual distances traveled, if we hadn't obscured the real tower identifiers.

The practical result is that it becomes more difficult to evaluate hypotheses using our data set. Although we believe that we made the right decision, reviewers of papers submitted to several conferences disagreed. Nevertheless, this data should prove useful to researchers working on improving resource usage on mobile devices, and those working on localization. The dataset is available from `http://hssl.cs.jhu.edu/~neal/woodchuck` and from CRAWDAD.

Intended to be blank.

Chapter 3

Data Analysis

As described in the introduction, we want to predict an individual's location in the near future using cell towers as a proxy for their geographic location. In this chapter, we take a close look at the cell tower traces that we collected to identify patterns that may make this job easier. Our focus is primarily descriptive. We identify, for instance, that the distribution of the number of times a tower is visited is consistent with a power law, and that users appear to sample the towers in their vicinity even though they are probably stationary. In subsequent chapters we apply these observations in the design of algorithms that process the cell tower traces, and make predictions.

3.1 Summary Statistics

To provide an initial impression of the traces, we start our analysis with some summary statistics. Table 3.1 shows how much data we have for each user, how many towers each user visits, how many times a user visits a tower, and how those visits are distributed among towers. Note: we only show 59 of the 91 traces that we collected; the remaining traces have less than 14 days worth of cell tower data, which is too little for this purpose. Here, as well as in the remainder of this thesis, we ignore these short traces.

The traces range from spanning the 14 day minimum to just under two years. Most traces have some days on which no data was collected. For very short periods, the phone likely ran out of power, and the user didn't recharge it promptly. For slightly longer periods, the user may have gone on vacation, and left the phone off. Very long gaps with subsequent returns to prior activity levels suggest users who started using a new device, but then returned to their N900. Another possibility is that the logging software crashed, and only resumed when

User	Totals				Joint Ratio	
	Days with Data	Time (Days)	Towers Seen	Tower Visits	Towers : Time	Towers : Visits
e7d	605 / 99%	609	6170	96 600	0.047 : 0.95	0.15 : 0.85
af6	523 / 93%	562	6510	119 000	0.044 : 0.96	0.13 : 0.87
d21	514 / 90%	571	5180	139 000	0.034 : 0.97	0.097 : 0.90
8b4	433 / 91%	475	1620	54 800	0.053 : 0.95	0.19 : 0.81
8be	383 / 58%	657	861	69 800	0.045 : 0.95	0.14 : 0.86
9ed	373 / 100%	373	312	188 000	0.019 : 0.98	0.026 : 0.98
532	367 / 99%	369	1790	76 100	0.036 : 0.96	0.11 : 0.89
715	308 / 98%	315	7060	103 000	0.072 : 0.93	0.15 : 0.85
2ee	296 / 62%	474	170	11 600	0.035 : 0.97	0.15 : 0.86
0b9	261 / 87%	301	425	64 100	0.061 : 0.94	0.13 : 0.87
593	250 / 99%	253	1430	116 000	0.072 : 0.93	0.12 : 0.88
5cd	225 / 74%	306	1600	29 200	0.034 : 0.97	0.13 : 0.87
640	210 / 95%	220	2390	71 600	0.068 : 0.93	0.14 : 0.86
020	207 / 100%	207	4500	78 300	0.084 : 0.92	0.19 : 0.81
7e1	193 / 93%	207	1080	27 300	0.035 : 0.96	0.12 : 0.88
5a9	169 / 97%	174	1330	211 000	0.048 : 0.95	0.052 : 0.95
99e	159 / 89%	179	1400	87 500	0.049 : 0.95	0.095 : 0.90
87e	142 / 97%	147	1740	73 500	0.049 : 0.95	0.12 : 0.88
b37	134 / 82%	163	614	26 800	0.055 : 0.94	0.15 : 0.85
c2b	129 / 92%	140	1300	29 600	0.053 : 0.95	0.11 : 0.89
b84	124 / 74%	168	1050	54 300	0.050 : 0.95	0.084 : 0.92
935	124 / 94%	132	1230	30 600	0.054 : 0.95	0.12 : 0.89
bb7	122 / 35%	345	1920	42 200	0.065 : 0.93	0.15 : 0.85
f14	114 / 100%	114	2710	38 600	0.052 : 0.95	0.17 : 0.83
26c	113 / 99%	114	2320	28 900	0.062 : 0.94	0.17 : 0.83
9cf	96 / 56%	170	871	17 900	0.076 : 0.93	0.16 : 0.84
05b	95 / 96%	99	2460	28 400	0.053 : 0.95	0.13 : 0.87
c5d	94 / 78%	120	419	27 700	0.033 : 0.97	0.062 : 0.94
b7e	93 / 100%	93	840	44 100	0.071 : 0.93	0.14 : 0.86
772	93 / 86%	108	704	23 200	0.045 : 0.95	0.096 : 0.90

Table 3.1: Tabular summary of the cell tower traces that have at least 14 days worth of data. The joint ratio is a measure of how equal something is distributed among a population. On average, the 59 users spend 94.0% of their time connected to 6.0% of the towers (standard deviation: 1.9%) and 14.0% of the tower visits are to 86.0% of the towers (standard deviation: 4.0%). These inequalities suggest that the quantities are distributed according to a heavy tailed distribution, such as a power law distribution.

User	Totals				Joint Ratio	
	Days with Data	Time (Days)	Towers Seen	Tower Visits	Towers : Time	Towers : Visits
0a1	91 / 100%	91	278	11 400	0.029 : 0.97	0.11 : 0.88
062	86 / 100%	86	1540	29 900	0.057 : 0.94	0.14 : 0.87
c6b	80 / 98%	82	759	18 100	0.055 : 0.94	0.15 : 0.85
949	75 / 100%	75	1070	20 900	0.047 : 0.95	0.12 : 0.88
8f4	73 / 95%	77	2390	50 900	0.048 : 0.95	0.075 : 0.93
3a7	69 / 57%	121	1090	16 800	0.064 : 0.94	0.15 : 0.85
137	64 / 100%	64	1440	16 800	0.056 : 0.94	0.18 : 0.82
f60	42 / 21%	202	634	13 200	0.058 : 0.94	0.13 : 0.87
23b	42 / 100%	42	859	12 100	0.049 : 0.95	0.15 : 0.85
3f3	41 / 15%	266	1250	6460	0.049 : 0.95	0.22 : 0.78
c5e	37 / 97%	38	128	5830	0.070 : 0.93	0.19 : 0.81
66d	33 / 58%	57	582	4150	0.065 : 0.93	0.19 : 0.81
bc2	29 / 100%	29	431	13 800	0.065 : 0.93	0.13 : 0.87
fb9	28 / 41%	69	556	5150	0.056 : 0.94	0.17 : 0.83
e6e	28 / 100%	28	507	10 600	0.071 : 0.93	0.16 : 0.84
6c6	28 / 100%	28	191	4680	0.057 : 0.95	0.19 : 0.81
ff2	24 / 100%	24	118	10 200	0.067 : 0.94	0.13 : 0.88
a08	21 / 100%	21	102	23 500	0.068 : 0.93	0.078 : 0.93
3b5	21 / 41%	51	95	6980	0.052 : 0.95	0.094 : 0.91
d60	20 / 100%	20	510	4140	0.067 : 0.93	0.22 : 0.78
cd3	20 / 95%	21	69	1660	0.043 : 0.95	0.14 : 0.86
140	20 / 12%	164	224	7500	0.089 : 0.91	0.16 : 0.84
ccf	19 / 90%	21	573	26 000	0.10 : 0.90	0.13 : 0.87
026	16 / 100%	16	45	1580	0.11 : 0.90	0.24 : 0.78
499	15 / 100%	15	231	8680	0.099 : 0.90	0.17 : 0.83
482	15 / 62%	24	124	3400	0.080 : 0.93	0.13 : 0.87
220	15 / 68%	22	60	1470	0.098 : 0.89	0.15 : 0.85
ef0	14 / 100%	14	336	5830	0.083 : 0.92	0.14 : 0.86
lee	14 / 100%	14	398	4180	0.045 : 0.96	0.15 : 0.85
					0.059 : 0.94	0.14 : 0.86

Table 3.1 (Continued): Tabular summary of the cell tower traces that have at least 14 days worth of data. The joint ratio is a measure of how equal something is distributed among a population. On average, the 59 users spend 94.0% of their time connected to 6.0% of the towers (standard deviation: 1.9%) and 14.0% of the tower visits are to 86.0% of the towers (standard deviation: 4.0%). These inequalities suggest that the quantities are distributed according to a heavy tailed distribution, such as a power law distribution.

the device restarted. For a handful of users including user 30c, there were days or weeks without any cell tower coverage even though other data was collected. These users may have removed the SIM card.

We consider a user to have seen a tower if the user connects to it at least once over the course of the trace. A tower visit is an individual connection, and it spans the time a user connects to the tower until he transitions to a new tower or the N900 is turned off. Thus, the number of times a tower is visited is the number of times the user transitions to that tower; and, the total number of tower visits is the total number of connections that were established to it.

Most users see hundreds or thousands of different towers, and visit these towers tens of thousands of times. Some users, such as 9ed, 2ee, and 0b9, visit an order of magnitude fewer towers than other users with similarly long traces. A possible explanation is that these users live in rural areas. In these areas, cell towers generally cover much larger geographic areas due to the lower population, and the correspondingly smaller network demand. In such an area, towers may be a worse proxy for location than they are in populous areas. Noulas et al. observe in their study of human mobility, however, that distance traveled is related to intervening opportunities [19]. Thus, even though cells are larger in rural areas, we expect them to cover a similar number of opportunities as in an urban area. If this is correct, living in a rural area is not sufficient reason for a user to visit fewer towers. A simple explanation for this behavior is that these users rarely leave home.

To give an impression of how evenly time and visits are distributed among the towers, we show their joint ratios. The joint ratio is a measure of how uniformly something is distributed among a population. It is is related to the Pareto principle or 80-20 rule, and is often mentioned when describing the distribution of wealth, namely, that approximately 80% of wealth is controlled by the richest 20%.[1] On average, the 59 users spend 94.0% of their time connected to 6.0% of the towers (standard deviation: 1.9%), and 14.0% of the tower visits are to 86.0% of the towers (standard deviation: 4.0%). This strong imbalance suggests that time and visits are distributed according to a heavy tailed distribution, i.e., there are a few towers that users visit a lot, but most towers are only visited a few times.[2]

The presence of a heavy tailed relationship means that we do not need to consider the thousands of towers that users see. Instead, if we only worry about being right, say, 99%, of the time, then we just need to consider, say, the 10% of the

[1] See http://en.wikipedia.org/wiki/Pareto_principle.

[2] Appendix B includes a brief introduction to heavy tailed distributions and power laws in particular as well as an explanation of a new technique for fitting and testing the goodness of fit of right-censored power laws, which is relevant later in this chapter.

towers that a user spends the most time at; the other towers are visited so rarely that we can disregard them. Aiming for 99% is perfectly reasonable: when using past behavior to predict future behavior, we will almost always incorrectly predict new behavior; the best that we can hope for is to correctly predict repeated behavior.

Reducing the size of the state space by, for instance, ignoring unimportant towers, is essential when using multiple features to make a prediction. Consider predicting the next tower the user will connect to (t_n) by conditioning on the current tower (t_c), and the previous tower (t_p), i.e., $\operatorname{argmax}_{t_n} P(t_n \mid t_c, t_p)$. This prediction has $O(N^3)$ states. If the user sees 1000 towers (and many of our users see more), then there are a billion states! And, we don't want to stop with just conditioning on two features: we would also like to condition on the time of day, and the day of the week, for instance. Although many interactions will never occur in practice (the user will never directly transition from a tower in Japan to a tower in Morocco, for instance), this high-dimensionality spreads out the data, which makes learning take much longer. This is the so-called curse of dimensionality [4, 76]. Wasserman notes that a consequence of the curse of dimensionality is that $842\,000$ examples in a 10-dimensional problem is similar to having just 4 examples in a 1-dimensional problem [4, Page 319]. To deal with this, we need to constrain the state space. One approach is to take advantage of power-law behavior to identify, and eliminate unimportant towers there by shrinking t_n's domain. Note: we can't as easily use this observation to shrink t_c or t_p, because infrequently visited states actually usually have a high information content.

Summary: Users visit hundreds or thousands of unique towers. Both the amount of time spent at a tower, and the number of times a tower is visited are probably distributed according to a right heavy tailed distribution. This means that most towers are unimportant—the user only visits them at most a few times and spends little time connected to them. We can potentially exploit this to reduce the state space.

3.2 Visualizing Movement

We now examine user movement. We visualize movement by plotting the time that a user's device connects to a tower vs. that tower's identifier. The difficulty with this is that towers don't have an intrinsic ordering.

Unfortunately, it is not possible to explore all possible orderings: there are factorially many. Further, there are no obvious optimization criteria we could use to automate the assignment of identifiers. Instead, we appeal to intuition.

Ordering towers by when the user first connects to them should reveal the routes that the user traverses, and the places that she visits. This is because we expect that the first time a route is traversed, or a place is visited all of the visited towers are new. Further, we expect subsequent traversals of a route or visits to a place to connect to a similar set of towers. Finally, most towers likely belong to just a single route or place. If this is the case, movement should show up as steep lines and visits to places should show up as wide splotches.

3.2.1 Overview

Figure 3.1 shows the first 18 weeks of the top 10 traces in terms of the total length of the trace. The user name is below and to the left of each plot. The x axis corresponds to time (units: weeks). The vertical grid lines correspond to Sundays and Wednesdays. The y axis is the tower identifier. The total number of towers is shown in parentheses. These plots partially confirm our intuition: we see near vertical lines, dark horizontal bands, and what appear to be dashed and dotted horizontal lines. Unfortunately, many details are obscured due to the large amount of data.

A consistent feature across all of the plots is the presence of a dark, occasionally interrupted, horizontal band. This band is typically at the bottom of the plot. Sometimes, it shifts partway through. This band corresponds to a user's primary location. Given the time and duration of the users' stays at these locations, we label them as "home" for readability. Despite this label, the true nature of these destinations remains speculative.

The dark band is augmented by secondary horizontal bands, which are thinner, and often appear dashed or dotted. The denser bands likely correspond to daily activities, such as work or school; the dotted bands likely correspond to regular activities, such as shopping, going to the gym, or attending a club's meetings. There are several easily identified secondary bands in user e7d's trace. For instance, starting in week 5 as well as in week 9, the user begins to visit a new place a few times each week. These are highlighted with red lines. In user af6's trace, we see what is likely a daily activity starting at week 11. The daily activity's location is first visited in week 9 suggesting a possible interview for a new job.

The bands are sometimes abruptly interrupted. During these times, the user visits a completely different set of towers. We call these abrupt changes regime changes—times at which the user's behavior changes dramatically.

Many of the traces exhibit regime changes. A regime change is often temporary and lasts from a few days to a few weeks. Afterwards, the user returns to the previous regime. This pattern is seen in user e7d's trace in week 3 and week 11 when the user goes away for the weekend, and in weeks 7 and 8 and in

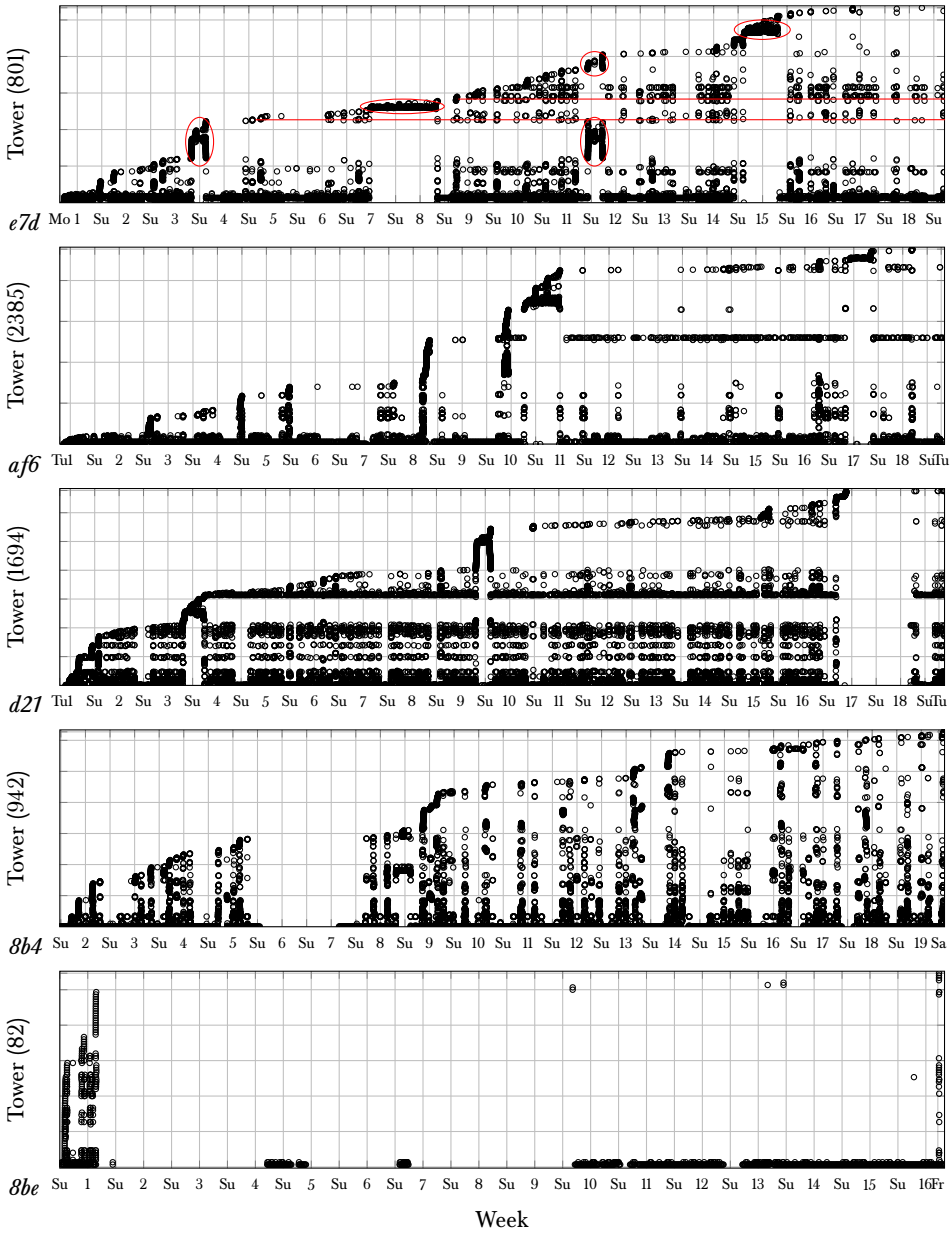


Figure 3.1: The time the cell phone connects to a tower vs. tower identifiers. Towers are assigned identifiers according to the order in which the user first connects to them. The number in parenthesis is the total number of towers seen. The plots reveal that regime changes (moves, trips and changes in secondary activities) are common across users.

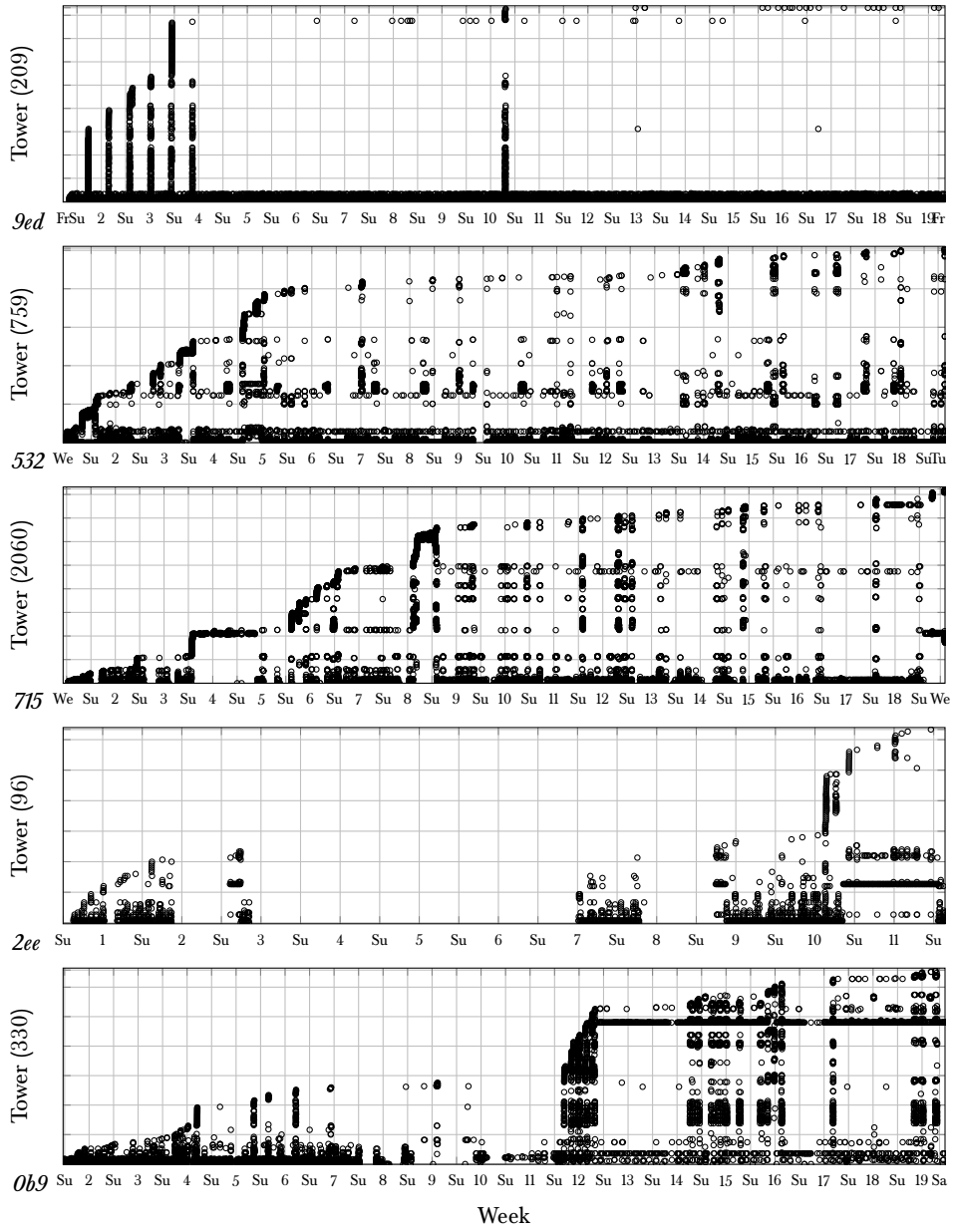


Figure 3.1 (Continued): The time the cell phone connects to a tower vs. tower identifiers. Towers are assigned identifiers according to the order in which the user first connects to them. The number in parenthesis is the total number of towers seen. The plots reveal that regime changes (moves, trips and changes in secondary activities) are common across users.

week 15 when the user goes away for about a week. These regime changes are circled in red on the plot. During these time periods, a new primary band is established. Again, this primary band probably corresponds to where the user sleeps.

These trips are sometimes bookended by two nearly vertical lines, the first rising and the second falling. This is the case for the trips in week 3 and week 11 of user e7d's trace. During the first trip, the user visits about 200 new towers. Most of the towers are visited twice, once at the beginning and once at the end of the trip and form the two nearly vertical lines. The first vertical line means the user is visiting many new towers in quick succession. The user might be traveling by car or train. The near vertical line at the end of the trip is falling. This means that the user is traversing many towers in the opposite order of their discovery. That is, the user is taking the same route in reverse. Note: on the return journey, the user visits some new towers along the way: the segment of the graph actual looks more like alligator jaws: ⟨. Thus, the return route (as viewed via the towers) is similar, but not identical to the route taken on the way.

During the second trip, the user visits many of the same towers (those in the lower circle), and again appears to use nearly the same route to travel to the destination and the return journey. The user also visits some new towers (those in the upper circle) both while traveling and at the destination. This underscores that *human behavior is globally regular, but locally variable.*

Outside of a trip, the towers visited during it are never visited. Similarly, none of the user's normal towers are visited during the trips. During these time periods, the user really is away and operating under a different regime.

Other trips, such as the one in weeks 7 and 8, aren't surrounded by these vertical bars. It could be that the destination was not far away. However, since no usual towers were visited, it is more likely that the user flew to the trip's destination. When flying, we expect to see at most one tower connection.

Sometimes, regime changes are permanent or semi-permanent. An example of a partial, permanent regime change starts at week 12 of user 0b9's trace. Since some of the normal towers continue to be visited, the user likely moved homes within the same city. Interestingly, during week 12, the user moves back and forth between the original location and the new location many times. During this time, the user might be moving his belongings to his new home. Although not shown in these plots, the traces also include instances of a user moving to a completely new location.

Occasionally, traces are interrupted, and there is no activity for a while. These gaps occur if the user leaves the device off for an extended period of time. For instance, a user may go on vacation and decide not to bring her cell phone. This phenomenon can be seen in user d21's trace in weeks 17 and 18.

Summary: Users are regular: they visit the same towers over and over again over a long period of time. However, they also exhibit local variation: users often connect to new towers along previously taken routes. Occasionally, users drastically change their behavior. Significant changes in behavior are regime changes. We observed that major regime changes occur when a user goes on a trip or moves. These are easy to detect: the user makes a clean break with the past, which partitions the towers between the major regimes. Minor regime changes occur when the user starts or ends a secondary activity, such as attending an evening course two nights a week at a community college for 10 weeks. This case will be harder to detect in the short term, because the surrounding context is unchanged.

3.2.2 Week By Week

We now zoom in on a small section of the first four traces shown in the previous figure. The plots, displayed in Figure 3.2, show the towers visited on Monday through Thursday of five consecutive weeks of each of the user's trace. (We exclude Friday through Sunday due to a lack of space.) The same days of the week are stacked on top of each other to simplify week-to-week comparisons. The tower identifiers have been renumbered according to the algorithm presented at the beginning of this section. Only those towers that are seen over the five weeks are considered when computing the tower identifiers.

A quick look at the plots confirms that users don't only visit the same towers over and over again, but that they have identifiable routines. Most of the time, the users stay the night at the same place. And, with the exception of user 8b4, the user's days are also somewhat regular. However, there is a fair amount of variation such that it is clear that precisely predicting the user's location at any given instant will not generally be possible no matter how much data we have.

We first consider the users' main activities.

User e7d leaves his "home" between 6 and 7 in the morning, and stays at his destination until about 4 in the afternoon. Given the time and duration, this is probably the user's workplace. During the first two days of the third week, the user appears to stay at "home." This could be because he was sick or had off. Alternatively, given the complete lack of movement, the user might have simply forgotten to take his cell phone with him. A close examination of the plots reveals small changes in the user's schedule. Sometimes, the user leaves for "work" at about 9 in the morning. The time that he stays there, however, remains about the same. Further, although the user appears to take the same basic route to and from "work," the user often encounters a few new towers. This suggests some variation in the route.

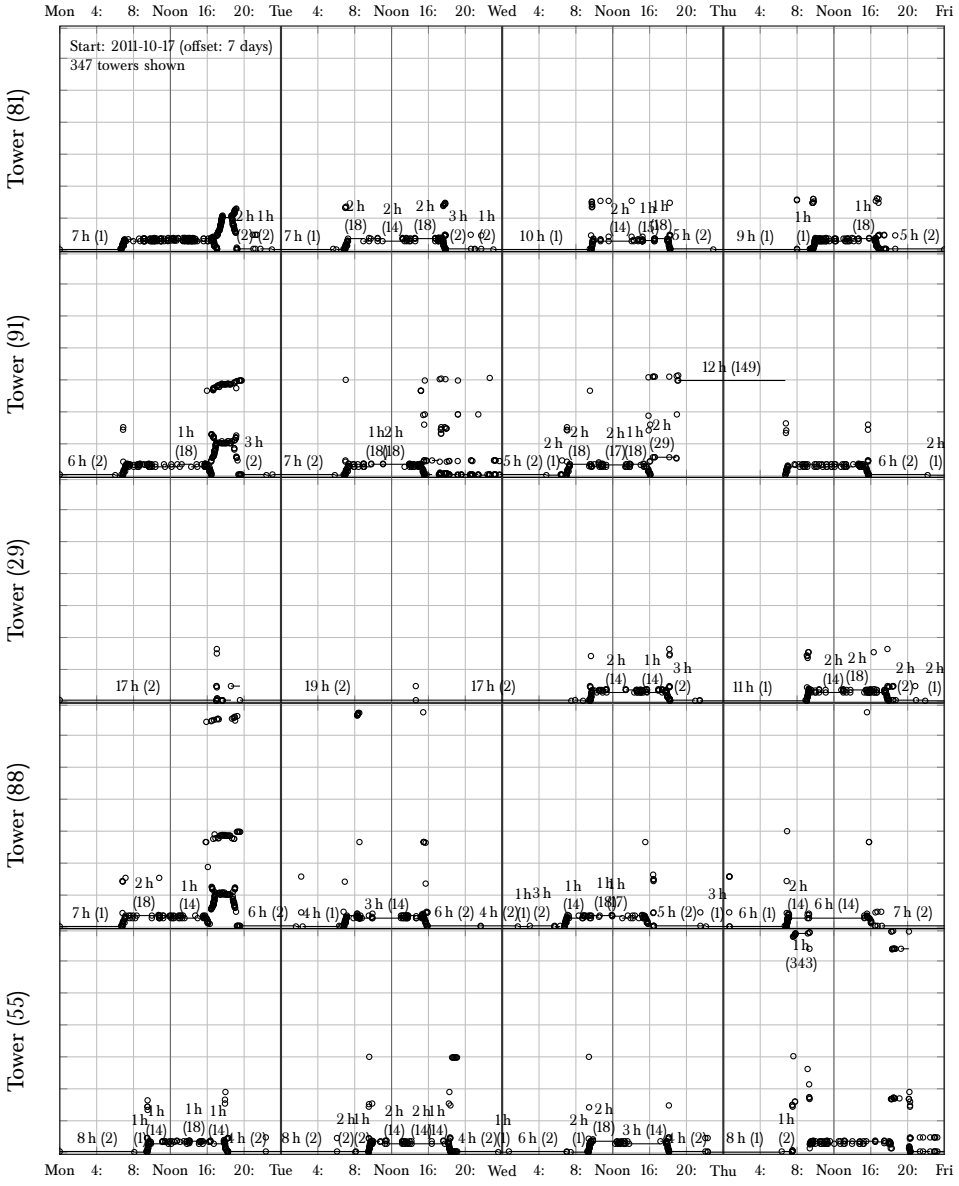


Figure 3.2: A week-by-week view of **e7d**'s trace. This plot shows time vs. tower connections made Monday through Thursday for five consecutive weeks of user e7d's trace. The y axis is the same for all weeks. The number in parentheses is the number of unique towers visited during the week. The horizontal gray lines depict the duration of long visits and the adjacent numbers are the respective dwell times and tower identifiers.

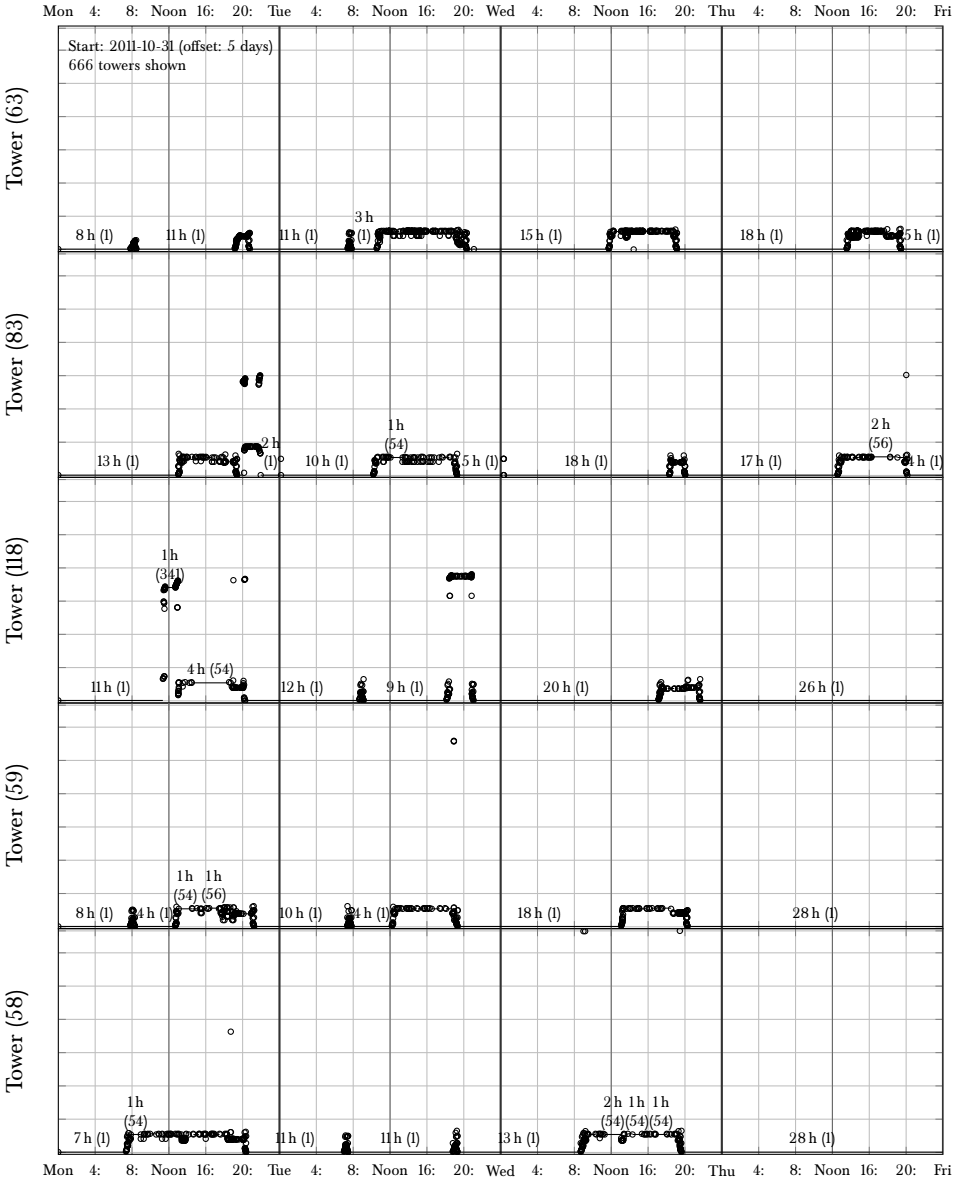


Figure 3.2 (Continued): A week-by-week view of **af6**'s trace. This plot shows time vs. tower connections made Monday through Thursday for five consecutive weeks of user af6's trace. The y axis is the same for all weeks. The number in parentheses is the number of unique towers visited during the week. The horizontal gray lines depict the duration of long visits and the adjacent numbers are the respective dwell times and tower identifiers.

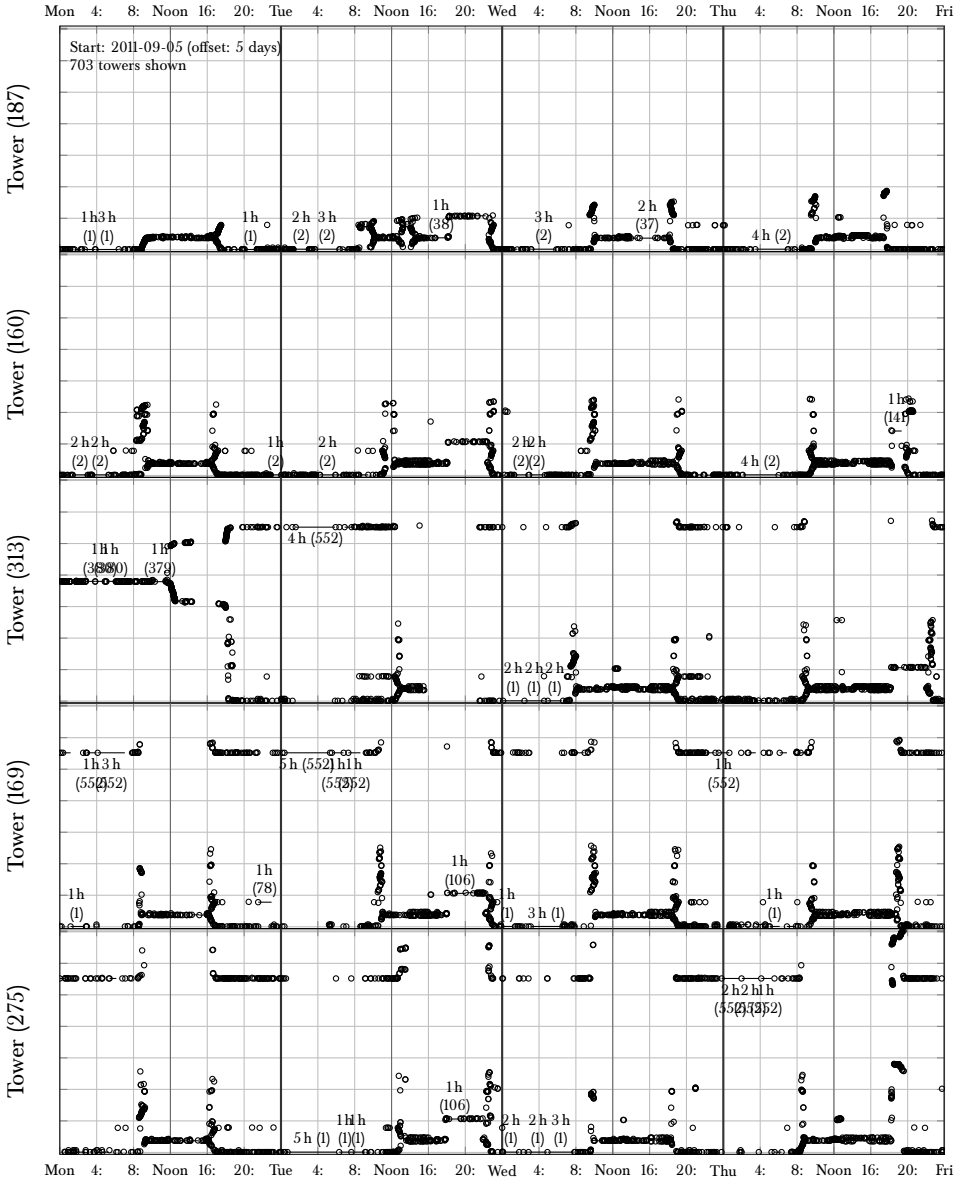


Figure 3.2 (Continued): A week-by-week view of **d21**'s trace. This plot shows time vs. tower connections made Monday through Thursday for five consecutive weeks of user d21's trace. The y axis is the same for all weeks. The number in parentheses is the number of unique towers visited during the week. The horizontal gray lines depict the duration of long visits and the adjacent numbers are the respective dwell times and tower identifiers.

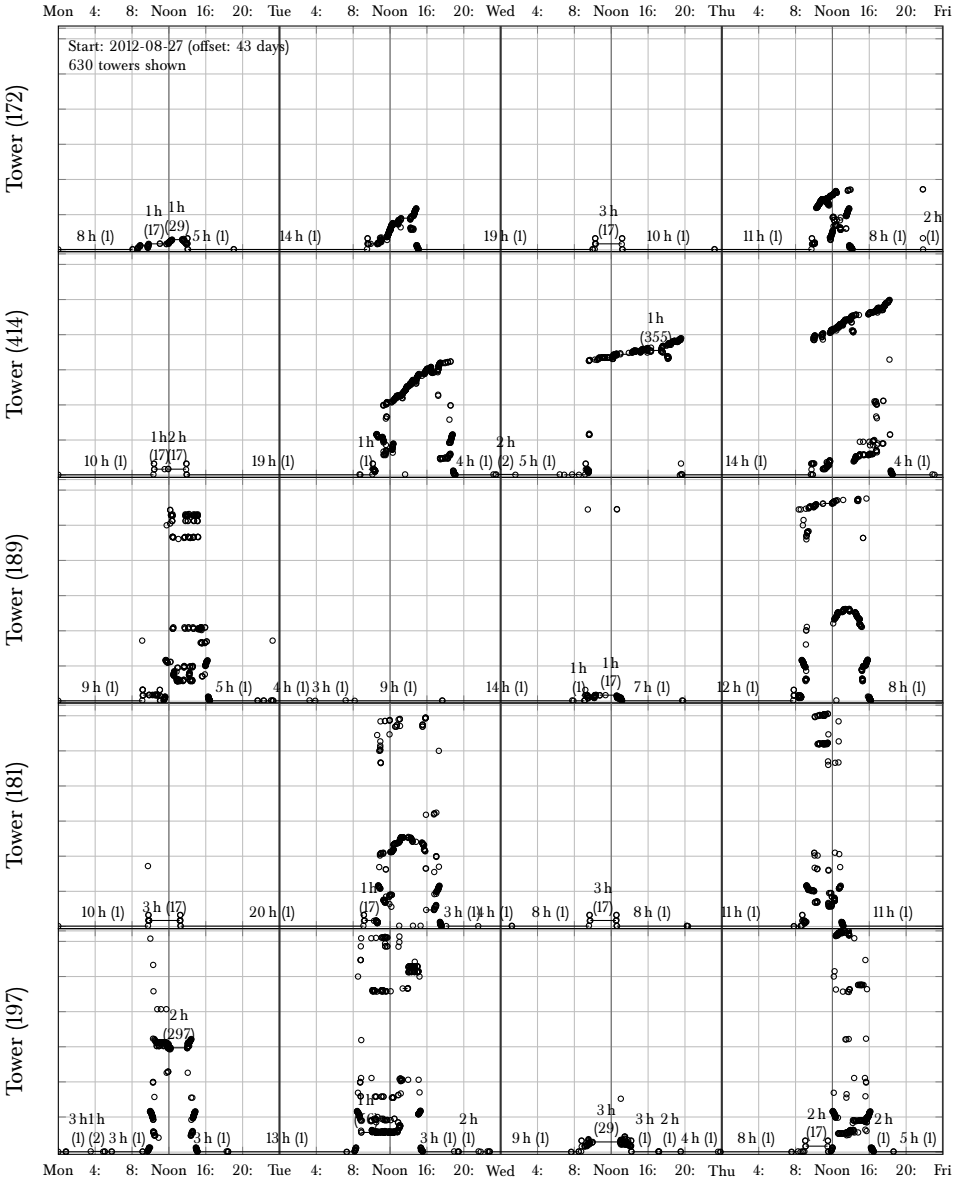


Figure 3.2 (Continued): A week-by-week view of **8b4**'s trace. This plot shows time vs. tower connections made Monday through Thursday for five consecutive weeks of user 8b4's trace. The y axis is the same for all weeks. The number in parentheses is the number of unique towers visited during the week. The horizontal gray lines depict the duration of long visits and the adjacent numbers are the respective dwell times and tower identifiers.

User af6 also appears to have a primary activity and a typical route to get there and back. Unlike e7d, however, af6's hours are much less regular: he leaves anytime between 8 in the morning and 1 in the afternoon and returns between 6 and 8 in the evening. Some days he only stays at his destination a few hours, and other days he stays about 12 hours. Further, there are several days on which the user either goes someplace else (e.g., the third Tuesday) or stays "home." It is difficult to identify a pattern. Perhaps user af6 is a college student.

User d21's plot is broadly similar to e7d's. For instance, she also appears to have an 8-hour-per-day job like e7d. And, she tends to go to "work" at about 9 in the morning, and to leave "work" about 8 hours later. An interesting feature is the presence of a new tower at her "home" location in the third week. This could be due to the network operator adding a new cell tower.

User 8b4 stays in the same place each night, often goes out at around 9 in the morning, and returns between 4 in the afternoon and 8 in the evening. However, this user often goes to different locations. In fact, he is often on the go the whole day. The second week is particularly interesting: from mid-morning until mid-afternoon the user is on the go. Perhaps the user is in sales. Or, this might be a long bike ride.

The plots also reveal periodic activity. In e7d's plot, we see that on the first, second and fourth Mondays, he doesn't go "home" after "work," but goes someplace else for about an hour. Similarly d21 has a three hour activity after "work" on four of the five Tuesdays. During the third week, the activity occurs on Thursday (at the usual time) instead. Even 8b4 has some regular activities. For instance, on three of the five Wednesdays, the user goes to the same place between 10 am and 1 pm in the afternoon. During the second week, this activity appears to take place on Monday instead of Wednesday, and during the last week it occurs on Thursday.

Some exceptional activities also stand out. On the second Wednesday, for instance, e7d doesn't spend the night at his "home". Because the user goes to "work" at the regular time the next day, it is unlikely the user is on a trip. Perhaps he stayed at a friend's house. Similarly, d21 appears to have gone away over the second weekend, and only returned home Monday afternoon. She might have gone away for a long weekend.

A common property of most of the user activities that we have examined—whether they be due to user movement or correspond to a visit to a place—is that they involve a group of towers or communities. The main exception is when a user is probably sleeping: then, the user sometimes remains connected to a single tower for an extended period of time. Otherwise, each time a user engages in an activity, she connects to multiple towers. Moreover, each time she does some activity, the tower sequence is slightly different. This holds for both user

movement as well as visits to a fixed location.

Consider user e7d's commute to "work." If we asked the user, he would probably tell us that he normally takes the same route to and from work. And, this is probably true for a reasonable level of abstraction. However, the sequence of towers that he traverses is probably never the same: each commute results in a slightly different tower sequence. This can be seen by observing that during a commute, the user often visits a few new towers. He may visit new towers, because he makes a detour. However, the weather also changes how signals are propagated, network maintenance means towers are sometimes out of commission, and new towers may be occasionally added.

A similar phenomenon can be observed during user e7d's workday: he frequently switches between a few towers. Looking at the trace, this switching appears to be random. The user may be moving between rooms when his boss calls him, or he could be sitting still, but have poor reception, which results in his cell phone oscillating between two towers. Whatever the case, the sequences are similar, but not identical.

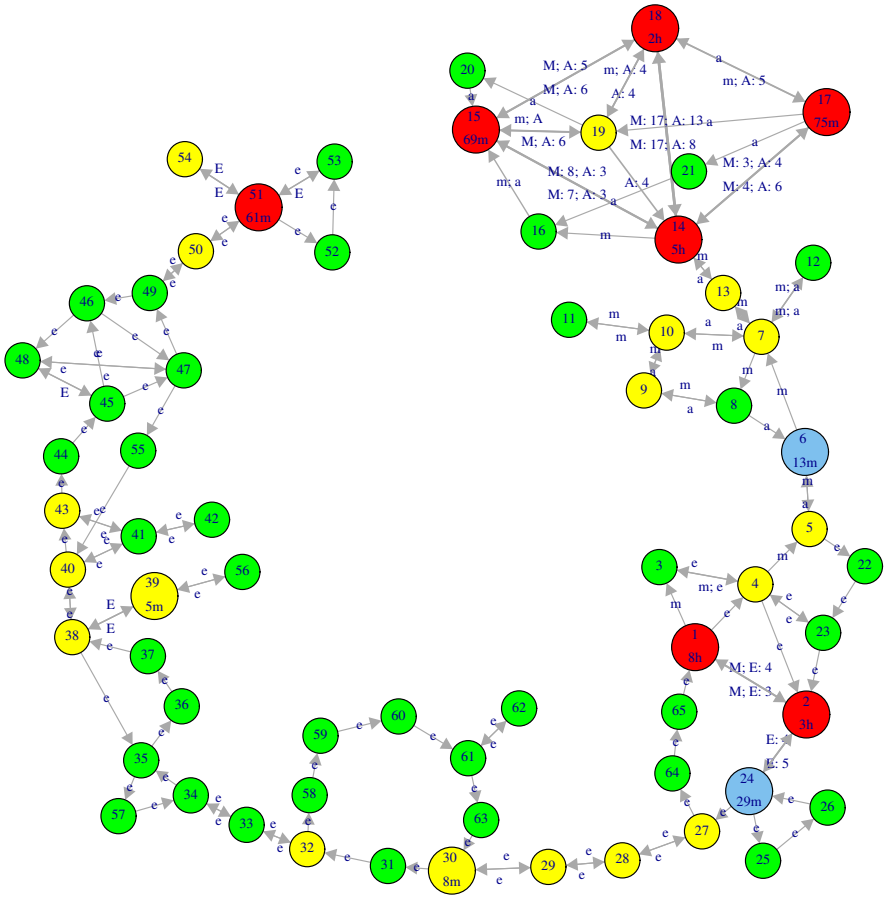
Summary: At a high level, users exhibit regularity: they appear to sleep at about the time, go to "work" at the same time, and stay there for a similar amount of time. Looking closely, however, we see that there is significant variability both in time and location. For instance, there is a two hour window between which user e7d normally leaves for "work". Once there, he moves between several towers. Each day, the resulting sequence is similar, yet different.

3.2.3 A Single Day

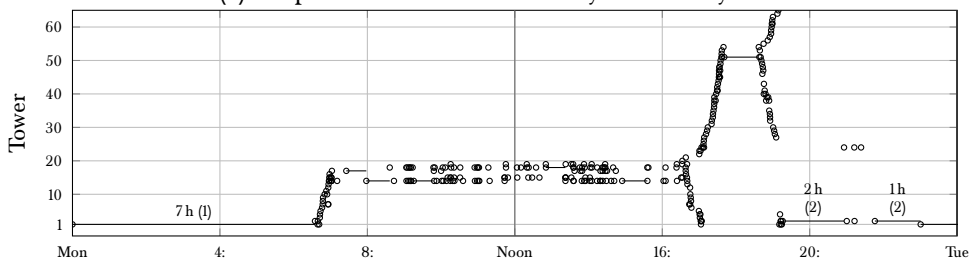
We now narrow our focus some more and consider just a single day.

Figure 3.3 shows the first day shown in each of the plots in Figure 3.2. Recall that, by construction, this is a Monday. The plot in the bottom panel is simply a zoomed-in view of Figure 3.3; the graph at the top is the induced cell tower network for that day.

The induced cell tower network shows the towers visited, and the tower transitions made. Each node includes the tower identifier used in the bottom plot and, for nodes where the user spent at least 5 minutes, the total time spent there. A node's color also indicates approximately how much time the user spent there across all visits. Green nodes correspond to towers that the user was connected to for less than two minutes; yellow nodes are towers that the user was connected to for two to 10 minutes; blue nodes are towers that the user was connected to for 10 to 60 minutes; and, red nodes are towers that the user was connected for at least an hour. The edges are annotated with the transition times and counts. N means a transition was made at night (midnight to

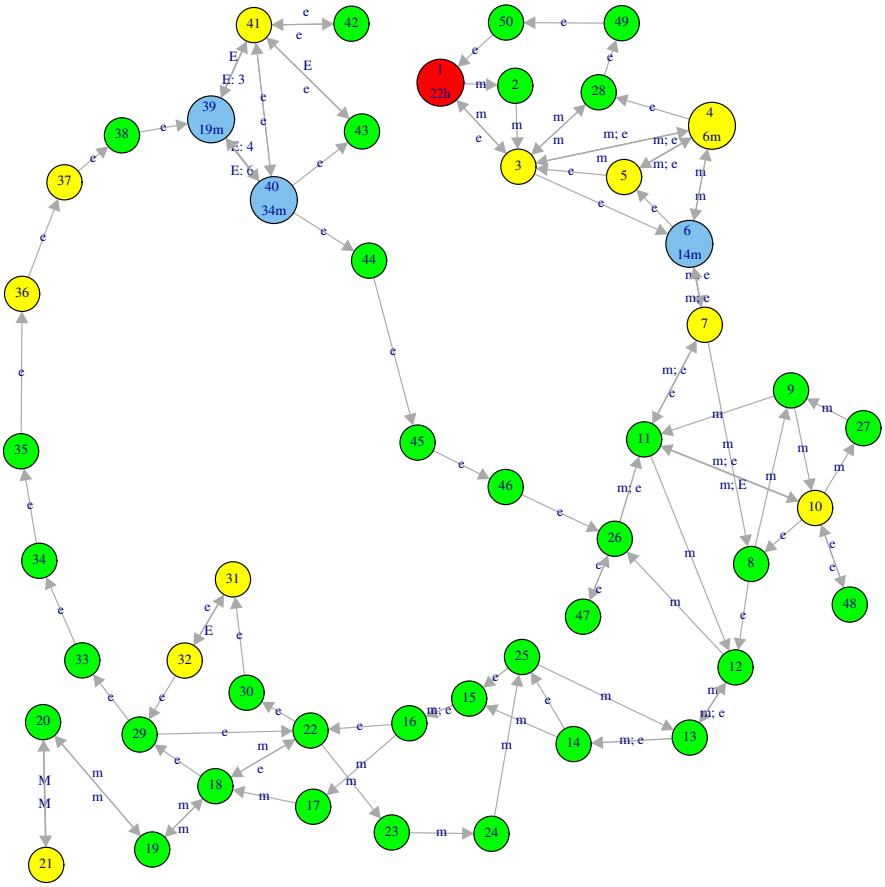


(a) Graph of the tower connectivity on one day.



(b) Tower sequence for same day.

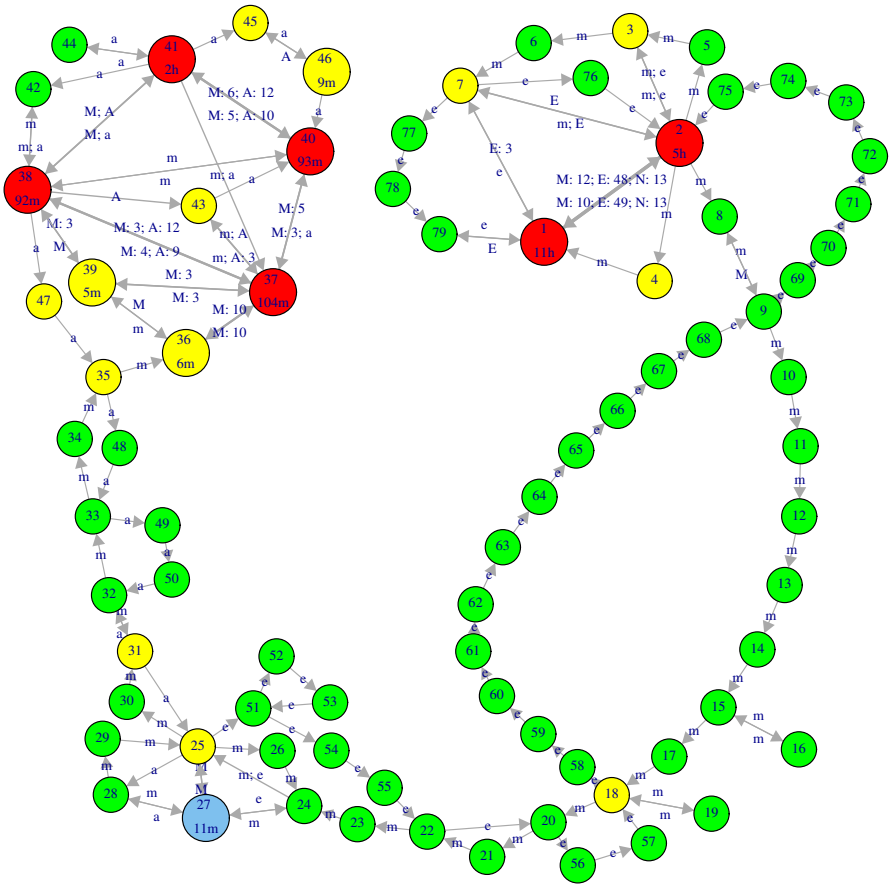
Figure 3.3: Graph of **user e7d**'s movements on the first day shown in Figure 3.2. The node labels are the cell tower identifiers (as numbered in the previous figure) and the amount of time (in minutes) spent at the tower. The bottom right cluster is the user's "home," the top right cluster is the user's "work," the top left cluster is the afternoon activity, and the paths in between them are the user's commute.



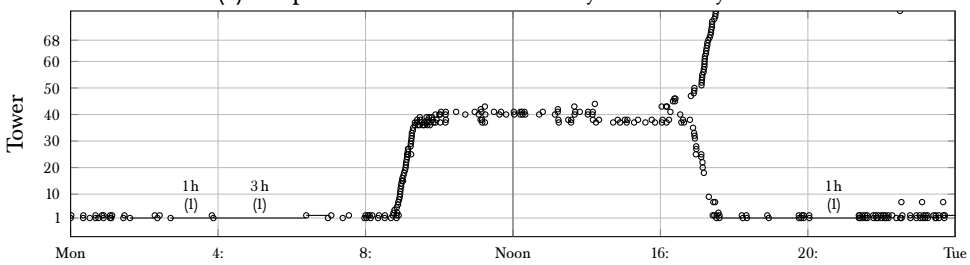
(c) Graph of the tower connectivity on one day.

(d) Tower sequence for same day.

Figure 3.3 (Continued): Graph of **user af6**'s movements on the first day shown in Figure 3.2. The node labels are the cell tower identifiers (as numbered in the previous figure) and the amount of time (in minutes) spent at the tower.

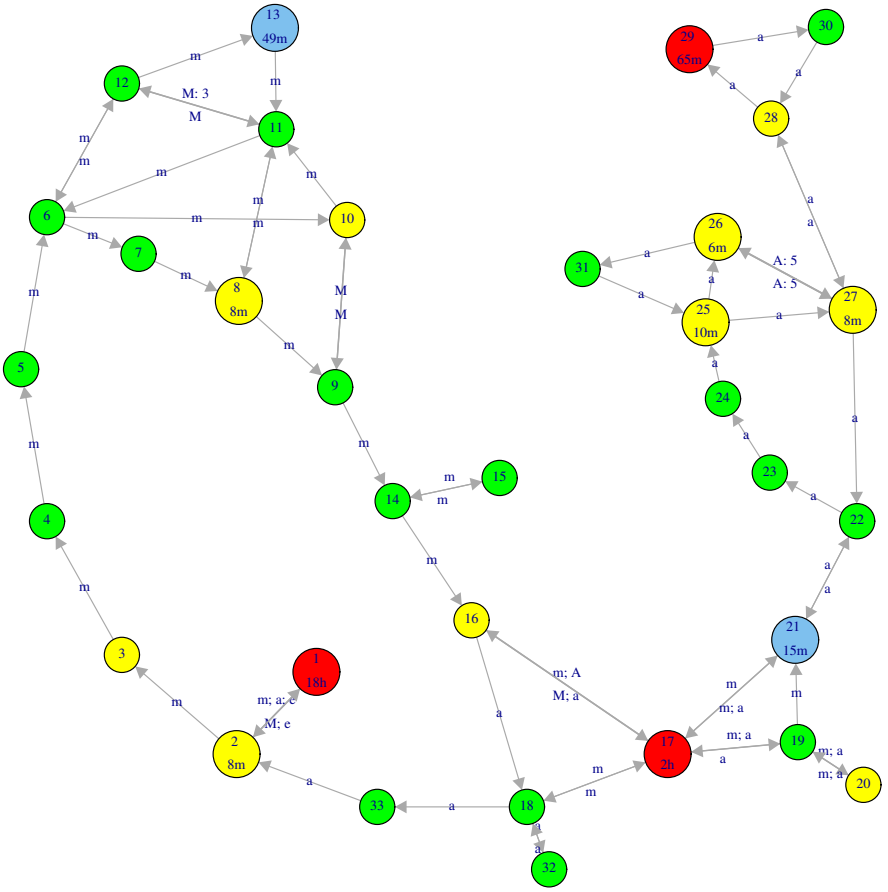


(e) Graph of the tower connectivity on one day.

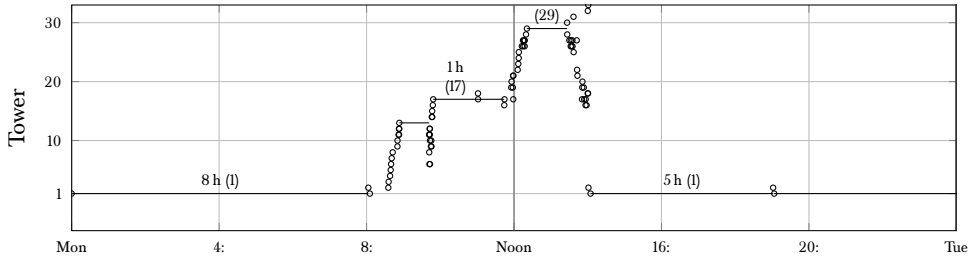


(f) Tower sequence for same day.

Figure 3.3 (Continued): Graph of **user d21**'s movements on the first day shown in Figure 3.2. The node labels are the cell tower identifiers (as numbered in the previous figure) and the amount of time (in minutes) spent at the tower.



(g) Graph of the tower connectivity on one day.



(h) Tower sequence for same day.

Figure 3.3 (Continued): Graph of **user 8b4**'s movements on the first day shown in Figure 3.2. The node labels are the cell tower identifiers (as numbered in the previous figure) and the amount of time (in minutes) spent at the tower.

6 am); `M` means a transition was made in the morning (6 am to noon); `A` means a transition was made in the afternoon (noon to 6 pm); and, `E` means a transition was made in the evening (6 pm to midnight). A lowercase letter means that the transition was taken once during that period; an uppercase letter means the transition was taken at least twice; if the uppercase letter is followed a number, then the transition was taken three or more times during the specified period, and the number is the count. If the user transitions in both directions between 2 nodes, the edge has two lists and the list closer to a node indicates when the user transitioned *to* that node (as opposed away from the node).

The graphs make clear that there are a few places where users spend most of their time. These places appear as communities rather than individual cell towers in the induced cell tower network. The communities generally consist of a few important towers, and a handful of less important towers. In user e7d's graph, we see three main communities. The community at the bottom right corresponds to the user's "home" (the user spends over 11 hours, mostly at night, there); the one at the top right is the user's "workplace" (the user spends 9 hours during the day there), and the one at the top left is the user's evening activity (the user spends a bit more than an hour there). Like e7d, user d21 appears to also go to "work" on the displayed day. Her "home" area corresponds to the community at the top right (she spends over 16 hours there), and her "work" community is at the top left (she spends over 7 hours during the day there). Neither user af6 nor 8b4 *appear* to go to a workplace, however, they may have worked from home (they spend 22 hours and 18 hours there, respectively). Both of these users do visit multiple locations for a short period of time. These probably correspond to errands or an activity.

Although the users spend more time at "home" than they do at "work," there are more towers in the "work" communities than in the "home" communities. This is easily explained. A work area is normally shared by many people. More people implies an increased load on the network and, correspondingly, more cell towers. Further, the work area is physically much larger than the home area. The practical result is that a person's various activities, such as, visiting the toilet, getting coffee, going to meetings and getting supplies are physically further apart at work than the person's activities at home. Thus, as a person goes about her business as work, she moves between more cell towers than she does at home.

Not all tower transitions appear to be due to real movement. User d21 appears to oscillate between the two main towers in her "home" community between 10pm and 2am. Given the time of day, this probably doesn't correspond to user movement, but the network or environment causing her cell phone to repeatedly switch between the two towers.

The location hubs are linked by chains of towers. Their long length com-

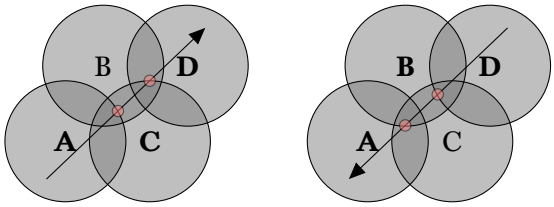


Figure 3.4: When moving, the sequence of traversed towers can depend on the direction of travel. This is because cells immediately next to each other overlap. (They don't interfere, because they use different frequencies.) When the user traverses the above path in the north-east direction, the cell phone needs to switch towers at the first small circle. It is likely to switch from tower A to C, because it is closer to C than to B. Likewise, when it reaches the second circle, C's signal is too weak and it needs to switch again. This time, D is closer than B. When traversing the path in reverse, the handoff points are at different positions and the resulting path is more likely to be $D \rightarrow B \rightarrow A$.

bined with their lack of repetitiveness, and their typically short dwell times indicate that these correspond to user movement. Although a route has a general shape, each time it is traversed, there is a bit of variation. In user e7d's trace, we see that after work, the user goes "home," and then immediately goes to an evening activity via his "workplace." On his way "home," he takes the same basic route in reverse. Although the general route is the same, we observe some variation.

We identify three potential causes for these variations in routes.

First, the variation could be true variation in the sense that the user really did take a different route. For instance, if the user lives in the city, and there are many one-way streets, then the user will necessarily take a slightly different route, and this route may be covered by different towers. Similarly, if the user encounters traffic, she might take an alternate route.

Second, when the user traverses a path in reverse, the handoff positions are different. This is because there is some overlap at the border of towers. The result is that the best tower may be different when coming than when returning. This idea is illustrated in Figure 3.4.

Finally, network and environmental effects can cause the device to connect to different towers. For instance, changing interference patterns can cause different towers to appear stronger at the same geographic location. Similarly, if a tower is overloaded, it may refuse a handoff, which causes the device to select a different tower, with a weaker signal.

As seen in Figure 3.2, this variation increases with time. Each time the user

takes the same route, a few new towers are sometimes discovered. Further, the set of towers actually visited each time is different. Thus, the relatively simple chains seen in Figure 3.3 only arise, because the corresponding routes are only traversed once or twice.

To better illustrate the variability, we show the induced cell tower network for the first workweek (Monday through Friday) of af6's trace that is shown in Figure 3.2. That is, we add four more days of the trace to the single day shown in Figure 3.3. We chose user af6, because this user visits a relatively small number of towers during the first week, and only moves between two places. Although the user only adds a handful of new towers after the first day, the number of new edges is enormous. Because of this, we only label the edges with the total number of transitions rather than breaking them down by the time of day. Note: to ease comparison with the graph in Figure 3.3, we used the same positions for the common nodes.

Our first observation is that the complexity of the network has increased dramatically. The top-left corner of the graph is the user's secondary location. Relative to the graph covering just a single day, it gained several nodes, and the nodes became highly interconnected. The routes that the user took also became more interconnected. Nevertheless, the basic shape remains: we can still easily identify a route, and the direction of travel. This pattern of handoffs suggests a new model for thinking about user movement. When a user traverses a given route, the modem doesn't cleanly switch to the subsequent tower when it reaches a cell boundary, but *the modem samples the towers* along the route. In fact, it may be that the modem is not actually connecting to the towers, but simply reporting a strong, nearby tower. And, due to constantly changing interference patterns, the modem often reports a new current tower even if the user is not moving, which explains why users connect to so many towers at "home" and "work."

The graph also reveals an anomaly: there are several transitions between towers that appear to be physically far apart from each other as inferred by the number of typically intervening towers. For instance, towers 54 and 42 belong to the user's secondary activity and tower 29 belongs to the middle of the user's route. Yet, at one point, the user moves from tower 54 to tower 29 and then 7 seconds later moves to tower 42. Further, after staying connected to tower 42 for just 6 seconds, the user connects to tower 22 (in the middle of the user's main route) to which she stays connected for 12 seconds, and then moves to tower 36 (which is part of the secondary location). Another example is the movement between tower 43 (part of the user's secondary activity), and tower 13 (along the main route). Possible explanations for these anomalies are that the geography suggested by the graph is wrong or that the modem reported bad data. However, this might simply be a manifestation of the tower identifier

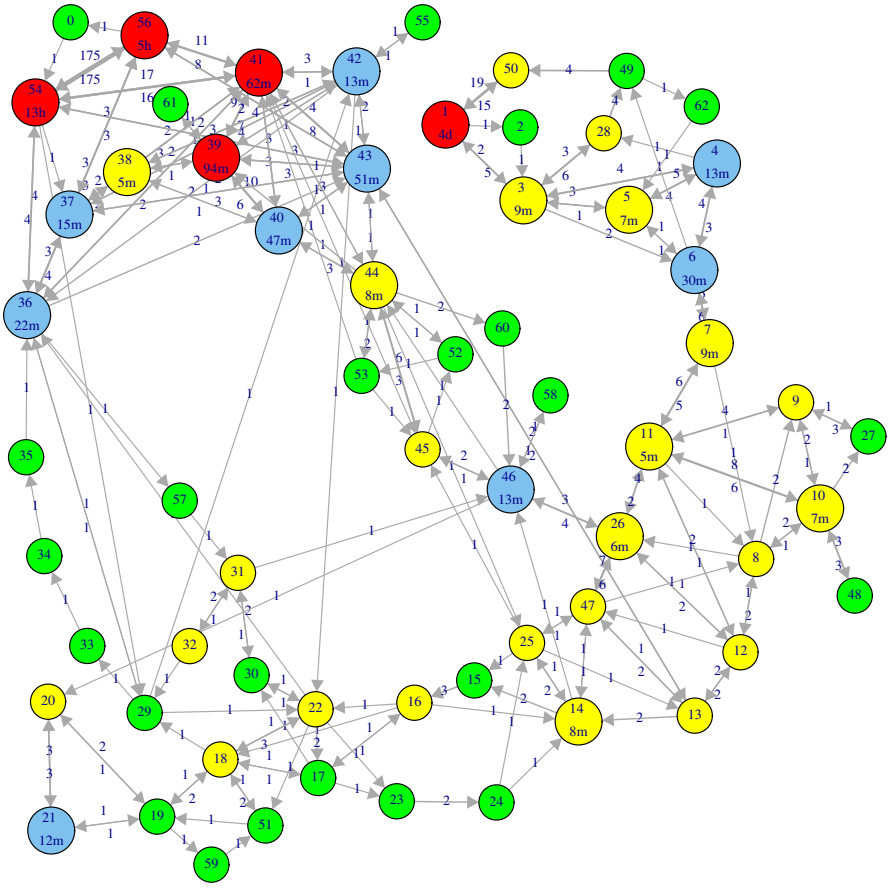


Figure 3.5: The induced cell tower network for 5 days of af6's trace.

aliasing bug described in Section 2.3.3. Given the number of towers, we expect to observe about 2 aliased pairs. It is conceivable that tower 20, tower 22 and tower 13 or 43 are these pairs.

Summary: This closer look at the traces further confirms that there are a few core locations where the users spend most of their time. These generally appear not as individual towers, but as clusters of towers—tower communities—which are highly connected. The cell phones appear to sample the towers in their vicinity. This is true not only along routes, but also for places.

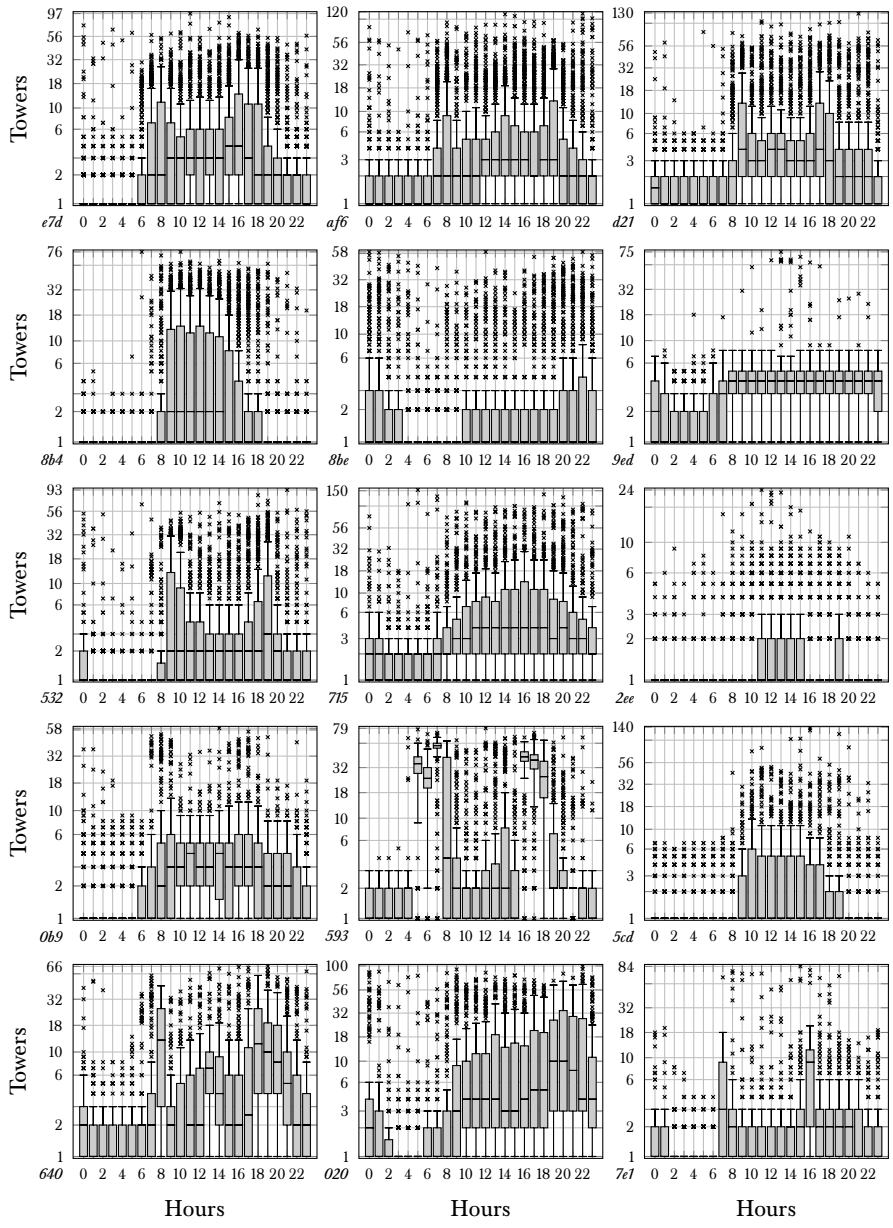


Figure 3.6: A series of box plots that shows the number of towers visited during each hour of the day. Note: the y axis is logarithmic. For most users, there is markedly more activity during the day than at night. We see that the median is often near the lower-quartile and that there are many large outliers. This suggests that the number of towers seen in a given hour is distributed according to a right heavy tailed distribution.

3.2.4 Time of Day

We now take a look at movement from a different perspective. Figure 3.6 shows a series of box plots of the number of towers seen during each hour of the day.

A box plot shows a data set's dispersion. The center of a box plot is a box with a horizontal line near the middle. This line is the median. The upper and lower edges of the box are the upper and lower quartile respectively. Two so-called whiskers extend from the box. These indicate the most extreme values that are not more than 1.5 times the interquartile range. Any data points that exceed these are considered outliers and are shown explicitly.

In terms of movement, people appear to have a daily routine: for most users, there are clear times when they are almost certainly sleeping (and hence, not moving and changing cell towers for an extended period of time) and when they are moving about. For a few users, there are a couple of hours where they are very active. For instance, user d21 tends to move around more between 9 and 10 in the morning and 5 and 6 in the evening, which is when the user commutes to and from "work," respectively. This pattern is also obvious in the plots for user 593, user 640 and user 7e1.

The box plots are asymmetric. This means that the number of towers visited during a given hour is not normally distributed [103, Chapter 4, "The Boxplot"]. This asymmetry manifests itself in two ways: the median is consistently closer to the lower quartile than to the upper quartile; and, most have outliers with high values whereas only one user (583) has any outliers that are below the bottom whisker. Both of these suggest a right skew.

Looking at the box plots, the median is consistently closer to the lower quartile than to the upper quartile. Oftentimes, there is no lower whisker. And, sometimes, the median number of towers visited in a given hour is 1. A median of 1 means that at least half of the time, these users (or rather, their cell phones) did not change towers. Given a typical cell tower's coverage of a circle with a 1000 to 2000 foot radius (as suggested in Figure 3.7), this means that the user is either really stationary, or moving within a small area. This lack of movement should not be surprising: for most people, travelling is a means, not an end; travelling is an overhead that is preferably avoided. With the exception of errands, which we suspect people probably try to keep short, when we go someplace, we often stay for at least an hour. There are two reasons for this. First, there is not much that one can accomplish in less than an hour. Second, the activity needs to justify the travel time.

Given that the lower quartile is typically just 1 or 2, it is not surprising that we don't see many outliers at the box plots' lower ends. The outliers that we do see suggest that people occasionally move a lot. This again is not surprising. If

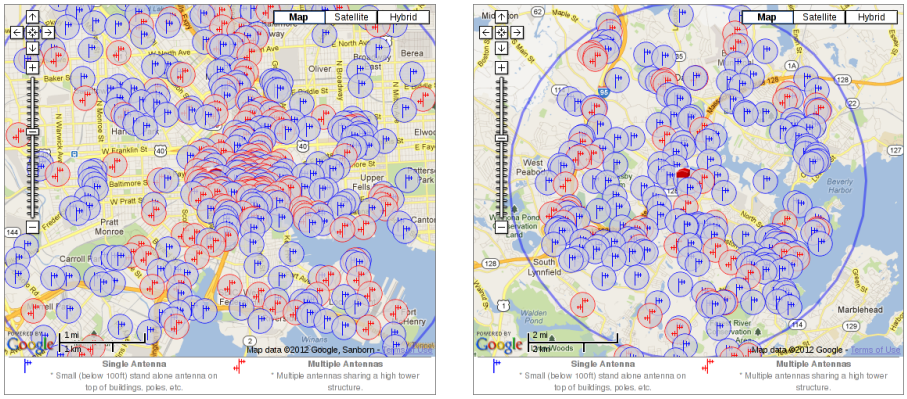


Figure 3.7: Some of the 943 antennas in a three mile radius around Johns Hopkins' Homewood campus and the 401 antennas in a four mile radius around Peabody, MA. Assuming that the towers are distributed equally among the four major cellular networks and assuming circular coverage, then the average radius per antenna for a given network is 1012 feet and 2069 feet, respectively. Data from `http://www.antennasearch.com`. Other urban and suburban areas appear to have similar coverage.

people mostly don't move, then when they do get in a car to go somewhere, they will visit many more towers than usual.

Summary: Using the number of towers visited during a given hour, we can recognize some aspects of typical routines. In particular, at night, the users connect to fewer towers than during the day. Also, regular commutes, such as those to and from the "workplace," show up as a spike in the number of visited towers. Typically, each hour has a number of outliers. We attribute this to occasional trips: people don't run errands everyday, but when they do, they move around a lot.

3.2.5 Conclusions

The most important observation that we made in this section is that people appear to be globally regular, but exhibit local variability. This variability manifests itself both in time and space. First, the exact time that a regular activity, such as going to "work," is done often varies. This is probably true no matter how rigid the user's schedule is: sometimes an errand, such as getting gas, needs to be done beforehand; or, the traffic report convinces the user to leave earlier or later. Second, the set of towers actually visited while traversing a route or at a fixed location varies. We observed that the phone appears to sample the towers

in its vicinity. This is potentially due to the user acting differently, but more likely arises from network and environmental factors. It also suggests that cell towers have a higher resolution than the locations that users visit, which means that towers are a reasonable proxy for location.

The variability in time means that predictions should somehow indicate the expect range. For instance, instead of saying that the user will arrive at work at 9 am, a predictor could say that the user will arrive between 7 and 9 in the morning with 90% confidence.

The variability in space suggests breaking traces into clusters before making predictions. We identify two promising methods to do this. First, biological sequence analysis faces a similar problem: DNA strands often have mutations or a wrong nucleotide is detected during sequencing [30]. The algorithms used in this field could help identify similar sequences, in particular, sequences associated with user movement. Another possible approach is to use graph theory to identify community structure in the form of tower aggregates. We explore this latter approach in chapter 5.

Another important observation that we made is that people sometimes change regimes. A major regime change happens when a person goes away on vacation or on a business trip or when she moves to another location. These are easy to detect: they represent a complete break from the past. Minor regime changes are when the user's routine changes a bit. This happens because, say, a course starts or ends. These are more difficult to detect, because the surrounding context remains the same: the user still goes to the same workplace at the same time, for instance.

To deal with both types of regime changes, we can age data. Ideally, we would age data quickly so that we are more responsive to changes in the user's behavior. However, aging needs to be done carefully: we don't want to completely forget historical data, because users often revisit previous regimes. For instance, even if a person visits a particular set of relatives a few times per year, his routine while there is probably similar. Recall that we observed this phenomenon in user e7d's trace.

3.3 Tower Transitions

In this section, we examine tower transitions in the induced cell tower network.

3.3.1 Transition Directions

Figure 3.8 shows histograms of the towers' outgoing and incoming transition directions, one for each of the top 15 participants. A transition direction is an

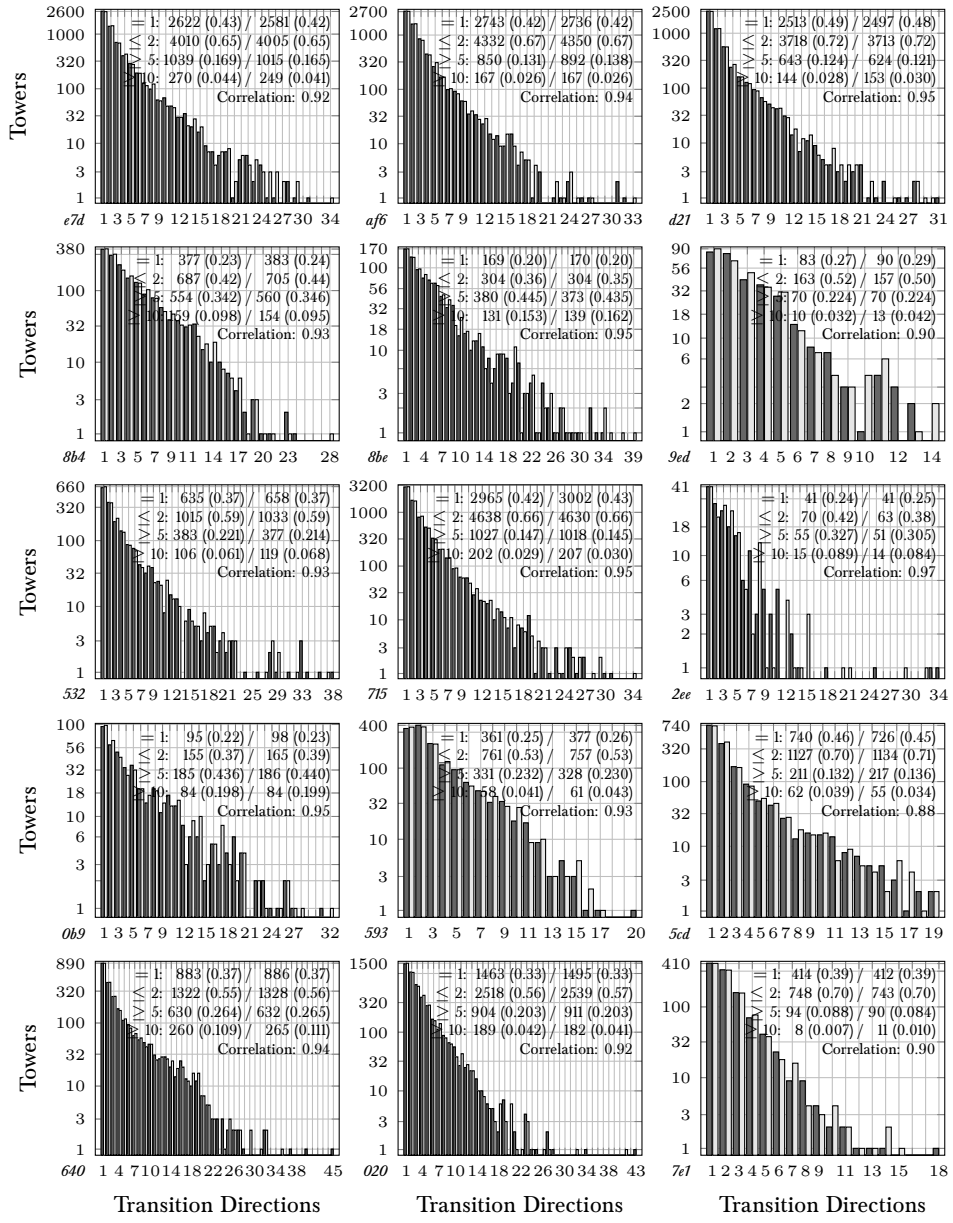
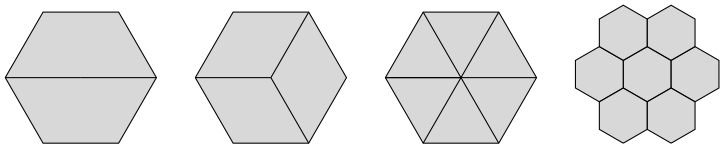


Figure 3.8: Histogram of outgoing (dark left bar) and incoming (light right bar) tower transition directions. A tower a has the two outgoing transition directions $a \to b$ and $a \to c$ if, when at tower a, the user only ever transitions to either tower b or tower c.



(a) 2 Sectors (b) 3 Sectors (c) 6 Sectors (d) 7 Towers

Figure 3.9: A cell can be divided into multiple sectors using directional antennas or broken apart into multiple, smaller cells.

edge in the induced cell tower network. For example, if we observe that a user transitions from tower a to tower b 100 times and from tower a to tower c 50 times, then we've identified two edges, two outgoing transition directions ($a \rightarrow b$ and $a \rightarrow c$), and two incoming transition directions ($b \leftarrow a$ and $c \leftarrow a$). Note: for the purpose of identifying transition directions, the total number of times that a transition is taken doesn't matter; what is important is whether a transition has been taken at least once. The inset in each plot shows the number of towers with the specified number of in or out degrees as well as the correlation between in degree and out degree.

The histogram doesn't show the relationship between the number of outgoing and incoming transition directions. It is conceivable that they are significantly different. For instance, a tower could have just a single incoming transition direction, and 10 outgoing transition directions. In practice, this is rarely the case. The correlation between the number of outgoing and incoming transition directions is significant. The mean correlation coefficient for the 59 traces is 0.92 with a standard deviation of 0.024. Note: a correlation whose magnitude exceeds 0.7 is considered to be a strong correlation [103, p. 184].

The plots show that about half of the towers have at most two incoming or two outgoing transition directions. It could be that only these towers are correlated. To eliminate this possibility, we computed the correlation for towers that have at least three incoming or outgoing transition directions. The correlation remains significant: the mean correlation coefficient drops a bit to 0.86, and the standard deviation increases to 0.060.

We can roughly divide the towers into three types: towers with at most three incoming or outgoing transitions directions; those with four to 10 incoming or outgoing transition directions; and, those with more than 10 transition directions.

Towers with no more than a few transition directions are the most common. On average, 70% of the towers have no more than three incoming or outgoing transitions direction (standard deviation: 12%). These towers are likely along routes.

The middle group is what we intuitively expect for towers at significant places. As discussed in Section 3.2.3, when a user is at a place that is covered by multiple cell towers, she transitions between most pairs of towers as she moves around the area. Since cells are approximately laid out in a hexagonal tessellation, we expect a given cell tower to have at most 6 neighbors. However, a cell is often subdivided into 2 to 6 sectors using multiple sector antennas instead of a single omni-directional antenna. This is shown in the first three illustrations in Figure 3.9. (See, for instance, Chapter 3 of Schwartz's Mobile Wireless Communications [87] for details.) When dividing cells in this way, each sector is given its own unique identifier, and thus appears as a unique cell. Further, a cell may be subdivided into smaller cells as shown in Figure 3.9 (d). If the neighboring towers are also sectored or subdivided, it is conceivable that a given cell could have about a dozen immediate neighbors.

The last group consists of towers with more than 10 transition directions. These account for 4.2% of the towers, on average (standard deviation: 3.7%). For users who visit 1000 towers, this corresponds to approximately 40 towers. In our analysis of the induced cell tower network during a single day in Section 3.2.3, we hypothesized that the modem doesn't just switch towers when the device crosses a cell's boundary, but effectively samples the towers its vicinity due to constantly changing interference. The high number of transition directions supports this hypothesis: occasionally, some towers that are far away appear to have a strong signal, and the modem reacts accordingly. We examine some alternate explanations in Section 3.3.3.

The distribution of the number of transition directions per tower appears to be distributed according to a right heavy-tailed distribution: we have many towers with just a few transition directions, and a non-negligible number with a huge number of transition directions. However, none of the common distributions (power law, left-truncated exponential or log normal) seem reasonable.

Summary: Towers have between one and several dozen transition directions. The number of outgoing and incoming transition directions are strongly correlated. The number of transition directions per tower appears to be distributed according to a right heavy-tailed distribution: most towers have just a few transitions, but there are a non-negligible number with a huge number. These observations support the hypothesis that the cell phone doesn't cleanly transition at cell borders, but changing interference causes it to effectively sample the nearby towers.

3.3.2 Transition Direction Popularity

We now investigate transition direction popularity, i.e., how often each transition direction is taken.

Figure 3.10 shows a complementary cumulative Pareto plot of the number of times each outgoing transition direction is taken for the top 15 towers (according to the number of outgoing transition directions) for each of the top 4 traces. We don't consider the number of incoming transitions, because, as we just observed, outgoing and incoming transitions are highly correlated, and thus we expect them to exhibit similar behavior.

The first thing to notice is that the number of times a transition direction is taken is not uniformly distributed. Rather, half of the transition directions are taken at most a handful of times, some are taken occasionally, and a few dominate.

This distribution suggests that how often a transition direction is taken is distributed according to a right heavy tailed distribution. To confirm this, we fit the data for each tower with at least 15 transition directions to a power law. There are 915 such towers across all of the traces. Table 3.2 summarizes the findings. The summary statistics for the α are calculated using just the statistically significant results. Note: the value of α for some individual towers as well as each's significance level is inset in the plots in Figure 3.10. The plots also show the corresponding regression.

The table reveals that the fit is generally statistically significant: for 91% of the towers, the fit of the popularity of the transition directions to a power law is significant at the $p = 0.05$ level. Moreover, the value of α is similar across towers and traces: the mean value of the αs is 1.71 with a standard deviation of 0.330. Note: in terms of evaluating the fit, the number of transition directions is relatively small and the fit could partially be a product of overfitting.

The distribution supports our tower sampling hypothesis. Specifically, if interference plays a large role in determining what tower appears strongest, then towers that are far away will only rarely appear to be strongest, which is consistent with the distribution that we observe here.

Summary: The distribution of transition direction popularity appears to be consistent with a power law: most transitions directions are taken just a few times, and a non-trivial number dominate. This is again consistent with the tower sampling hypothesis.

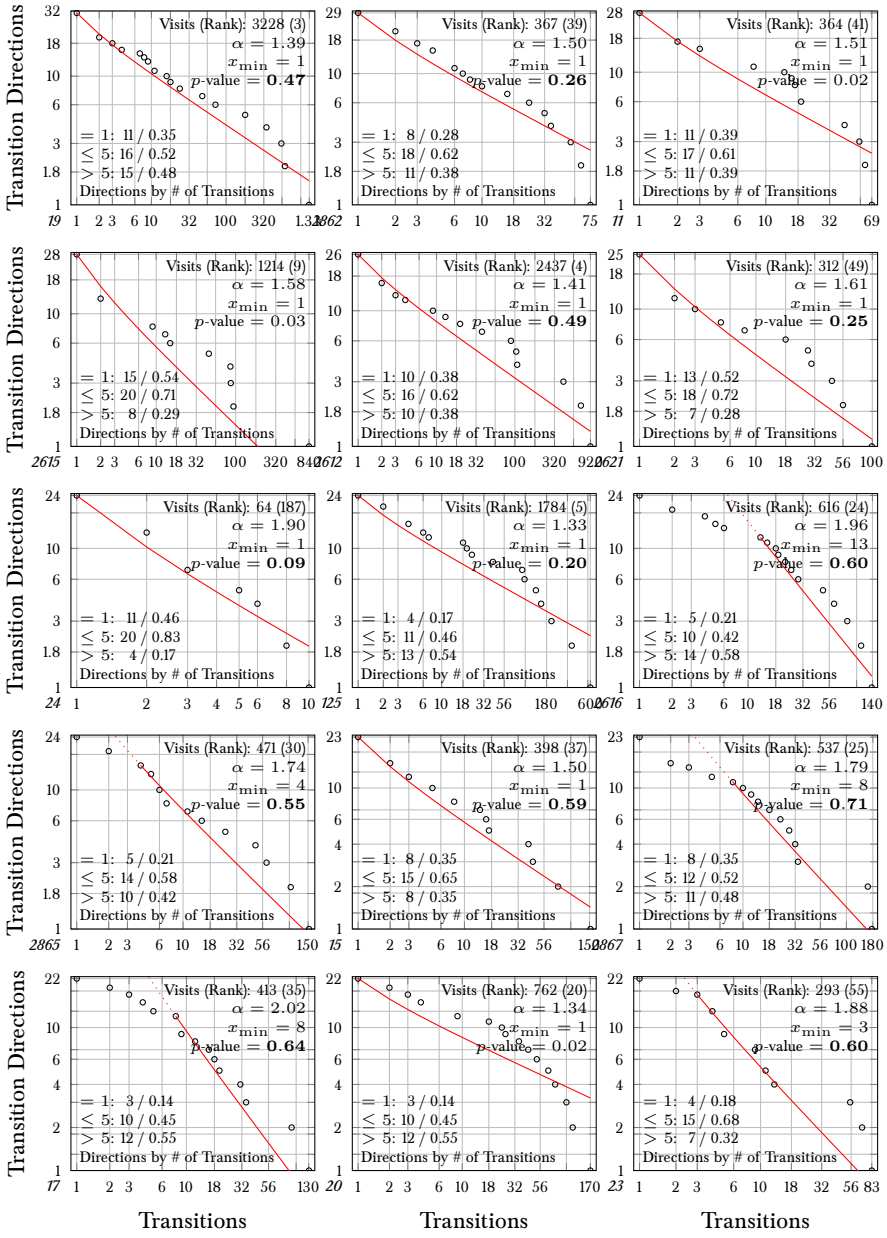


Figure 3.10: Complementary cumulative Pareto plot of the popularity of outgoing transition directions for the top towers (according to the number of outgoing transition directions) in **user e7d**'s trace. The x axis is the minimum number times a transition direction was taken. The y axis is the transition direction frequency.

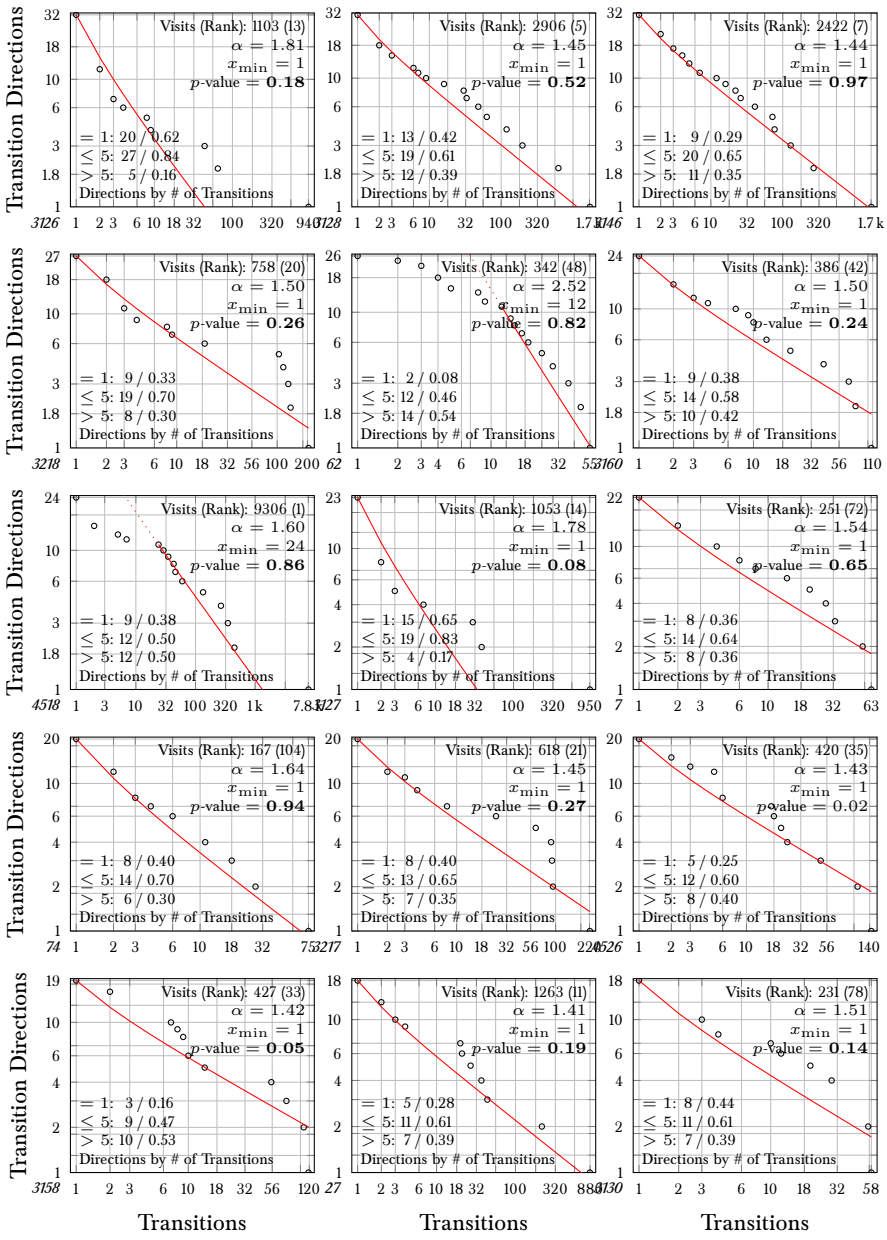


Figure 3.10 (Continued): Complementary cumulative Pareto plot of the popularity of outgoing transition directions for the top towers (according to the number of outgoing transition directions) in **user af6**'s trace. The x axis is the minimum number times a transition direction was taken. The y axis is the transition direction frequency.

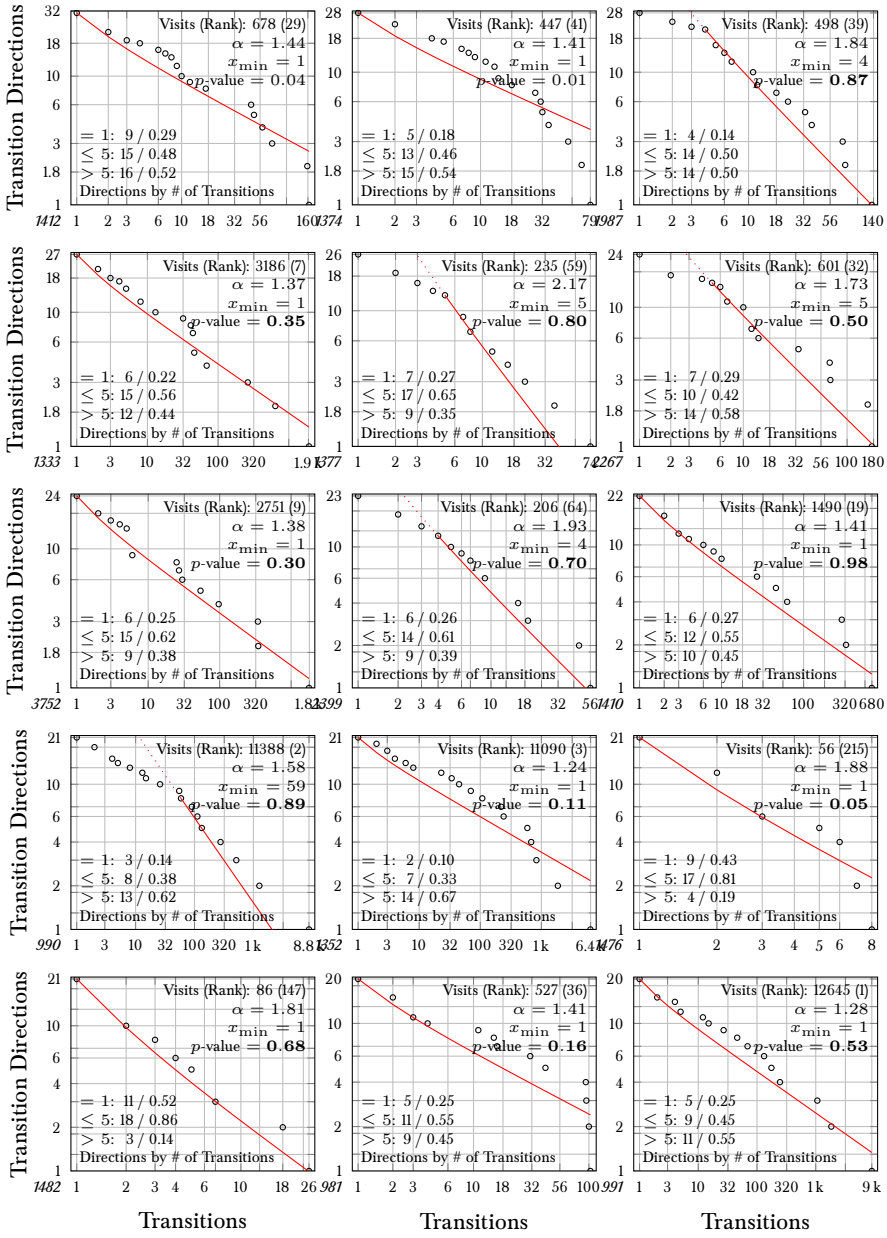


Figure 3.10 (Continued): Complementary cumulative Pareto plot of the popularity of outgoing transition directions for the top towers (according to the number of outgoing transition directions) in **user d21**'s trace. The x axis is the minimum number times a transition direction was taken. The y axis is the transition direction frequency.

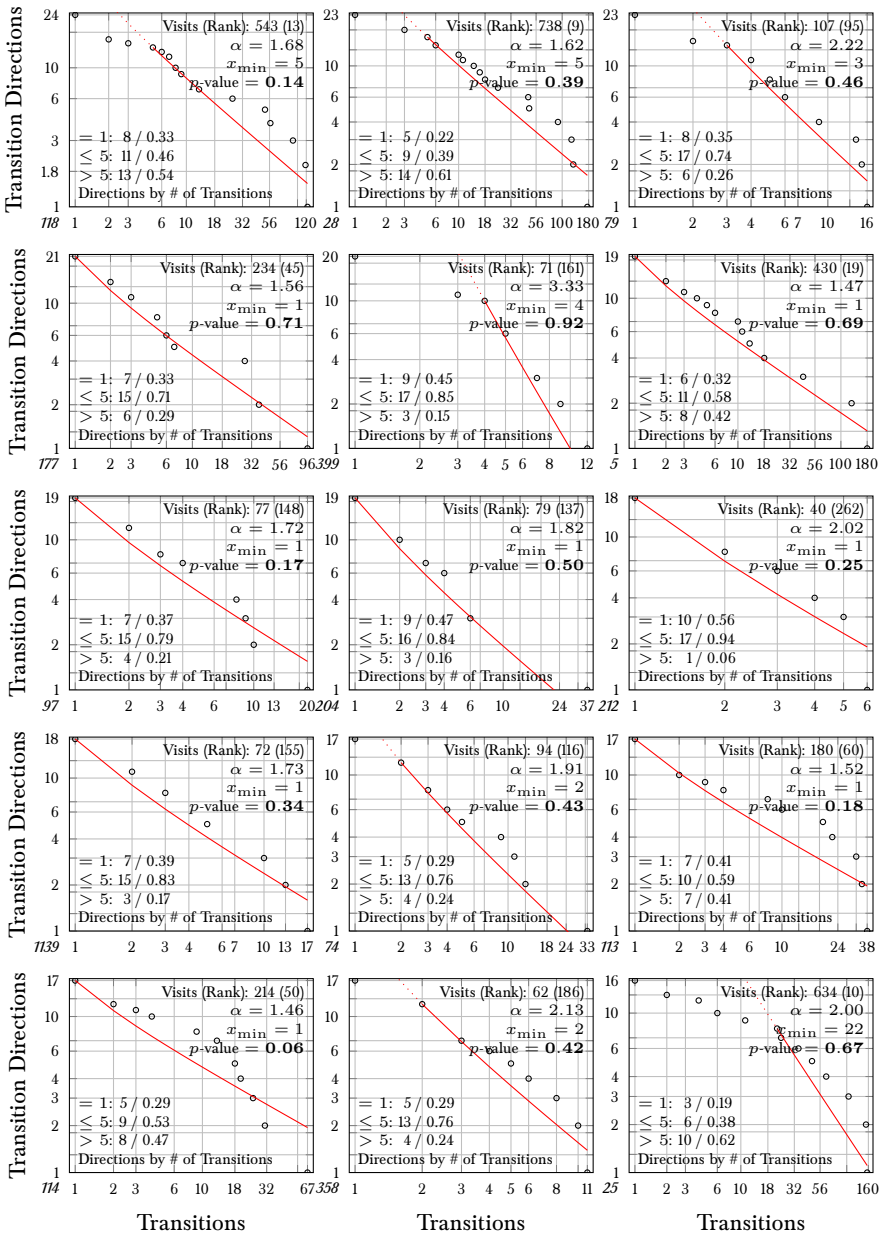


Figure 3.10 (Continued): Complementary cumulative Pareto plot of the popularity of outgoing transition directions for the top towers (according to the number of outgoing transition directions) in **user 8b4**'s trace. The x axis is the minimum number times a transition direction was taken. The y axis is the transition direction frequency.

	Towers		α	
User	Total	$p \geq 0.05$	μ	σ
e7d	62	56 / 90%	1.73	0.311
af6	44	38 / 86%	1.70	0.370
d21	39	35 / 90%	1.65	0.316
8b4	28	27 / 96%	1.76	0.393
8be	62	56 / 90%	1.74	0.389
9ed	0	0 / —		
532	43	40 / 93%	1.69	0.317
715	55	47 / 85%	1.77	0.377
2ee	4	4 / 100%	1.77	0.386
0b9	33	31 / 94%	1.66	0.258
593	5	5 / 100%	1.49	0.177
5cd	11	11 / 100%	1.93	0.633
640	99	86 / 87%	1.78	0.335
020	39	31 / 79%	1.75	0.356
7e1	1	1 / 100%	1.36	—
5a9	38	37 / 97%	1.60	0.357
99e	64	62 / 97%	1.67	0.278
87e	50	45 / 90%	1.75	0.300
b37	12	10 / 83%	1.90	0.323
c2b	21	20 / 95%	1.77	0.341
b84	15	15 / 100%	1.55	0.210
935	20	15 / 75%	1.67	0.175
bb7	10	9 / 90%	1.59	0.317
f14	28	25 / 89%	1.75	0.330
26c	21	21 / 100%	1.86	0.309
9cf	5	4 / 80%	1.58	0.0738
05b	7	7 / 100%	1.60	0.142
c5d	3	3 / 100%	1.54	0.172
b7e	22	20 / 91%	1.74	0.307
772	1	1 / 100%	1.75	—

Table 3.2: The distribution of visits to tower transition directions. We only consider towers with at least 15 transition directions. Visits to nearly all of the towers considered are distributed among the outgress transition directions according to a power law distribution. The average values of α are computed across the statistically significant towers. The mean across all users and towers is $\alpha = 1.71$ with a modest standard deviation of 0.330.

	Towers		α	
User	Total	$p \geq 0.05$	μ	σ
0a1	4	4 / 100%	1.52	0.106
062	2	2 / 100%	1.60	0.163
c6b	14	13 / 93%	1.70	0.288
949	2	2 / 100%	1.45	0.0947
8f4	6	5 / 83%	1.46	0.0467
3a7	4	4 / 100%	1.80	0.315
137	0	0 / —		
f60	10	9 / 90%	1.57	0.135
23b	2	2 / 100%	1.88	0.252
3f3	0	0 / —		
c5e	0	0 / —		
66d	0	0 / —		
bc2	1	1 / 100%	1.46	—
fb9	1	1 / 100%	1.57	—
e6e	4	4 / 100%	1.98	0.482
6c6	0	0 / —		
ff2	0	0 / —		
a08	0	0 / —		
3b5	0	0 / —		
d60	0	0 / —		
cd3	0	0 / —		
140	6	4 / 67%	1.41	0.0691
ccf	1	1 / 100%	1.39	—
026	1	1 / 100%	1.67	—
499	7	6 / 86%	1.71	0.221
482	1	1 / 100%	1.49	—
220	0	0 / —		
ef0	6	6 / 100%	1.68	0.282
lee	1	1 / 100%	1.82	—
	915	829 / 91%	1.71	0.330

Table 3.2 (Continued): The distribution of visits to tower transition directions. We only consider towers with at least 15 transition directions. Visits to nearly all of the towers considered are distributed among the outgress transition directions according to a power law distribution. The average values of α are computed across the statistically significant towers. The mean across all users and towers is $\alpha = 1.71$ with a modest standard deviation of 0.330.

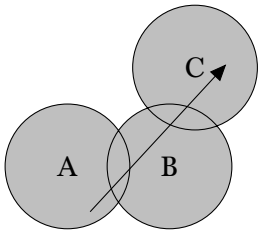


Figure 3.11: Although towers A and C are not adjacent, it is conceivable that the user could transition directly from A to C if B is overloaded and refuses a handoff. In this situation, the user could still remain connected to A until it reaches C: A and B don't interfere, because they use different frequencies; A's reception will, however, be weak.

3.3.3 Tower Sampling

The tower sampling hypothesis is that the modem doesn't report the nearest tower, but the strongest tower in its vicinity, which, due to changing interference, often changes even if the device is not moving. We first proposed in when examining the induced cell tower network. In this section, we argued that the number of towers with a large number of transition directions, and transition direction popularity support this hypothesis. We now consider other ways that towers could have so many transition directions.

Skipped Towers

When looking at transition direction popularity, we found that half of the tower transitions are taken at most a handful of times. If we ignore these transition directions, the hexagonal tessellation of cell towers again appears plausible—the 4.2% of towers with more than 10 transition directions shrinks dramatically. We refer to these transition directions as rare transition directions. They are rare not because they occur infrequently (indeed, they are very common), but because they are taken infrequently.

Rare transition directions could occur if some data is missing from the logs, which could happen if the modem didn't propagate data to user space, because of a bug in the firmware, say.

Another possibility is that towers are occasionally skipped. This could happen if a tower is overloaded and refuses a handoff [115, Section 7.12.3]. This situation doesn't necessarily lead to the user losing service: towers overlap, and the current tower could continue to serve the station well into the new tower's area albeit with a lower quality of service. This idea is illustrated in Figure 3.11.

Finally, skipping towers may just be normal. Since neither a phone nor the cell tower actually knows when the phone is at the cell's border, they use signal strength to determine when the phone should switch towers [99]. Due to interference, thresholds and other dampening processes are used to prevent the phone from switching towers too often. Thus, if cell overlap is large, then it may be perfectly reasonable to effectively skip towers.

In practice, we suspect that the handoff threshold may be raised if the phone is not actively using any network services. This is because, a larger threshold translates to fewer handoffs. The tradeoff is a decrease in the quality of service. However, if the user is not actively using any network services, then the signal just needs to be strong enough to allow a handoff, which is much less demanding than handling a call or transferring data. This policy would lead to an increased number of skipped towers.

We also suspect that the overlap is much larger in cities than in non-urban areas. First, cities are densely populated, which means an increase in the number of users and a corresponding increase in network load. To deal with this, more cell towers with smaller service areas are typically deployed. Because some overlap between cells is required, smaller cells necessarily have more overlap than larger cells.[3] Second, cities have densely packed buildings, which interfere with signal propagation. Thus, the nearest cell tower may not even be visible if a building is in the way. These conditions probably also lead to an increase in the number of skipped towers.

If either of the first two of these possibilities—the modem not propagating transitions or towers refusing handoffs—were the primary cause, then we would expect the target of a rare transition direction to normally be visited indirectly. That is, in the language of Figure 3.11, if we saw $A \to C$, we would expect to also see $A \to B \to C$ in the trace. Indeed, given that these ought to be rare conditions, we ought to see $A \to B \to C$ more often than $A \to C$.

Table 3.3 shows how often the user transitions both directly and indirectly to the target of a rare transition direction. We define a rare transition direction to be a transition direction that is taken less than the $\log$ of the number of visits to the tower. We define an indirect transition to be a sequence of towers $A \rightsquigarrow B$

[3] If the lower limit at which a cell phone can communicate with a cell tower is -100 dBm, then we don't want to configure the tower such that the expected signal at its border is -100 dBm. This would result in dead spots if there was interference. Instead, we might aim for, say, -90 dBm. Since the distance that it takes -100 dBm to decay to -90 dBm is the same (ignoring interference) independent of how far away the source is or how strong the signal initially was, smaller cells will overlap more than larger cells. That is, if there is minimal overlap when a cell's radius is r, then when r is increased by δ, the cell overlaps approximately $\pi (r+\delta)^2 - \pi r^2$ with its neighbors. Thus, the ratio of the overlap to its area ($\pi(r+\delta)^2 - \pi r^2 / \pi r^2$) shrinks as r increases.

Rare Trans. Type	Direct Trans.	Target Indirect Trans.	Target Total Visits
16	2	110	426
24	4	9	64
35	9	15	54
55	1	0	63
89	1	1	9
128	4	10	29
139	3	9	78
175	2	2	6
177	1	2	46
354	1	0	1
442	4	0	859
484	21	54	104
627	1	0	1
810	1	1	2
930	8	2	13
1247	3	1	69
2501	1	0	1
2612	1	1	2448

(a) The number of indirect transitions to the targets of tower 19's rare transition directions.

	Trans. Types		Targets with ≥ 1 Indirect Trans.	
Tower	All	Rare	Count	Percent
19	31	18	13	72%
2862	29	18	14	78%
2615	28	20	19	95%
11	28	17	14	82%
2612	26	16	11	69%
2621	25	18	16	89%
125	24	13	11	85%
2616	24	12	11	92%
2865	24	16	14	88%
24	24	19	14	74%
2867	23	12	11	92%
15	23	15	13	87%
20	22	10	10	100%
2864	22	11	11	100%
25	22	13	8	62%
17	22	10	8	80%
2657	22	12	6	50%
23	22	15	13	87%
2613	21	13	9	69%
2866	21	13	9	69%
Mean				81%

(b) Portion of the top towers' transition directions that have at least one indirect visit. The mean is over the 270 towers with at least 10 transition directions.

Table 3.3: The degree to which the target of rare transition directions are transitioned to indirectly in **user e7d**'s trace. An indirect transition is a transition via one or two other towers. The mean portion of rare transition directions whose target is transitioned to indirectly at least once is 74.0% (standard deviation: 9.3%).

that is either three or four towers long, i.e., the user moves from A to B via one or two intermediate towers. The displayed data is from user e7d's trace. Other traces are similar.

Table 3.3 (a) shows the number of direct and indirect transitions to the target of each of the top tower's rare transition directions. (We ordered the towers by the number of outgoing transition directions.) The number of indirect transitions is sometimes significantly larger than the number of direct transitions. This suggests that the indirect route is the real route. The occasional direct transition can then be explained by the phone forgetting to report some data or to a tower being overloaded and refusing a handoff. Thus, these explanations appear to only hold for a small portion of the transition directions.

Table 3.3 (b) shows the number of rare transition directions and the portion of those that are visited indirectly for the user's top towers (as determined by the number of outgoing transition directions). At the bottom, the mean for all towers that have at least 10 outgoing transition directions is shown. The grand mean across all 59 traces is 74.0% (standard deviation: 9.3%). In other words, skipped towers due to the modem not reporting data or an overloaded tower refusing a handoff only appear to be a conceivable explanation for ***at most*** three-quarters of the rare transition directions.

Note: in terms of the total number of transitions, rare transition directions are taken relatively infrequently: across the 59 traces, the mean portion of tower transitions via rare transition directions is 2.1% with a standard deviation of 1.4%. Thus, whatever the cause of the rare transition directions, it may be reasonable to treat them as noise.

Umbrella Towers

Another explanation for the numerous transition directions is the presence of umbrella cells.

An umbrella cell is a macrocell that overlays a group of microcells [50, 88]. An example is shown in Figure 3.12. The cellular concept is based on fractals: if the system needs more capacity, instead of using more spectrum, a cell is split into a number of smaller cells, say 7, and each is configured to transmit with just enough power to cover 1/7 of the area. This results in (ideally) a 7 fold increase in the amount of capacity in the area. The most obvious additional costs of splitting a cell are the additional equipment, their maintenance, and the rent for the new locations' real estate. There is, however, another cost: cellular stations need to change towers more often when moving. The umbrella cell reduces this overhead: when a station starts to move, instead of connecting to the neighboring microcell, it connects to the umbrella cell. When it is stationary

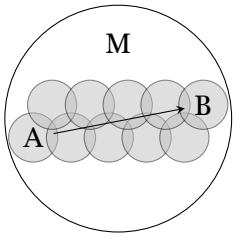


Figure 3.12: An umbrella cell tower with 10 microcells. Note: the microcells need not completely fill the umbrella cell; they only need to be deployed where additional capacity is needed. In such a configuration, moving from A to B could result in the following tower sequence $A \rightarrow M \rightarrow B$.

and again needs to transmit, it switches back to a microcell.

Umbrella cells are a possible cause of the numerous transition directions that many cell towers have: the user can transition not only to the umbrella cell's neighbors, but to any of the umbrella cell's microcells. Many of these transition directions are likely to be infrequent. For instance, when the user moves from A to B in Figure 3.12, he might not immediately transition to M upon leaving A: the handoff to the umbrella cell only happens if the user appears to have reached a velocity that suggests longer movement. Thus, he might first connect to the right neighboring cell and then to M. Another possibility is that the user is transferring data, and the station switches to a microcell when the user is at a stop light, because it has more capacity than the umbrella cell.

Again, based on the observation that the connected cell towers appear to be a sampling of an area as discussed in Section 3.2.3, we don't believe that umbrella cells are the primary cause of towers with high degrees.

Conclusion

We examined four alternative explanations for the existence of towers with many transition directions: the modem failing to report transitions; the modem skipping overloaded towers; high overlap between cells, and high switching thresholds causing the phone to skip cells; and, the presence of umbrella cells. Our analysis suggests that although these may occur, none of them explains most instance of towers with a the large number of transition directions whereas the tower sampling hypothesis does. Applying Occam's razor, we conclude that this phenomenon provides support for the tower sampling hypothesis.

3.3.4 Conclusions

We observed that some towers have many transition directions, and that their popularity is distributed according to a power law. We argued that this is consistent with the behavior predicted by our tower sampling hypothesis. A consequence of this is that tower transitions are not good indicators of user movement; transitions probably occur even when the device is stationary.

3.4 Tower Visits

We now turn from exploring the transitions in the induced cell tower network to the nodes themselves—the cell towers.

In our discussion of the summary statistics in Section 3.1, we noted that the amount of time spent at a tower was highly right skewed and the number of visits to a tower might be distributed according to a heavy tailed distribution. We now examine these observations in detail. Specifically, we look at the number of times a user visits a tower, and the amount of time spent there both over the entire trace as well as during individual tower visits.

3.4.1 Number of Tower Visits

We start by looking at the number of times a user visits each tower that was recorded in her trace.

Figure 3.13 shows complementary cumulative Pareto plots of the number of tower visits. The x axis shows the ***minimum*** number of times the user visited a tower and the y axis shows the number of towers for which this is the case.

In many of the plots in Figure 3.13, the data roughly follows a straight line. There is some minor downward deviation on the left side and some more significant downward deviation in the tail. However, even without explicitly accounting for the deviation in the tail, we find that 35 of the 59 traces (59.0%) are consistent with a power law at the $p = 0.05$ significance level. Further, all of the traces have similar parameters: the average value of α across all 59 traces is 1.84 with a modest standard deviation of 0.22. This suggests that this particular power law behavior may be common among users.

We now turn our attention to the deviation on the left. The deviation—when there is one—is almost always downward. Based on the selection of $x_{\min}$, we see that the deviation is relatively small: the median value of $x_{\min}$ is 6 with a MAD of 6. The downward deviation means that fewer towers are visited at most a handful of times than the model predicts. Nevertheless, the number of towers with only a few visits is enormous: the median portion of towers with at most

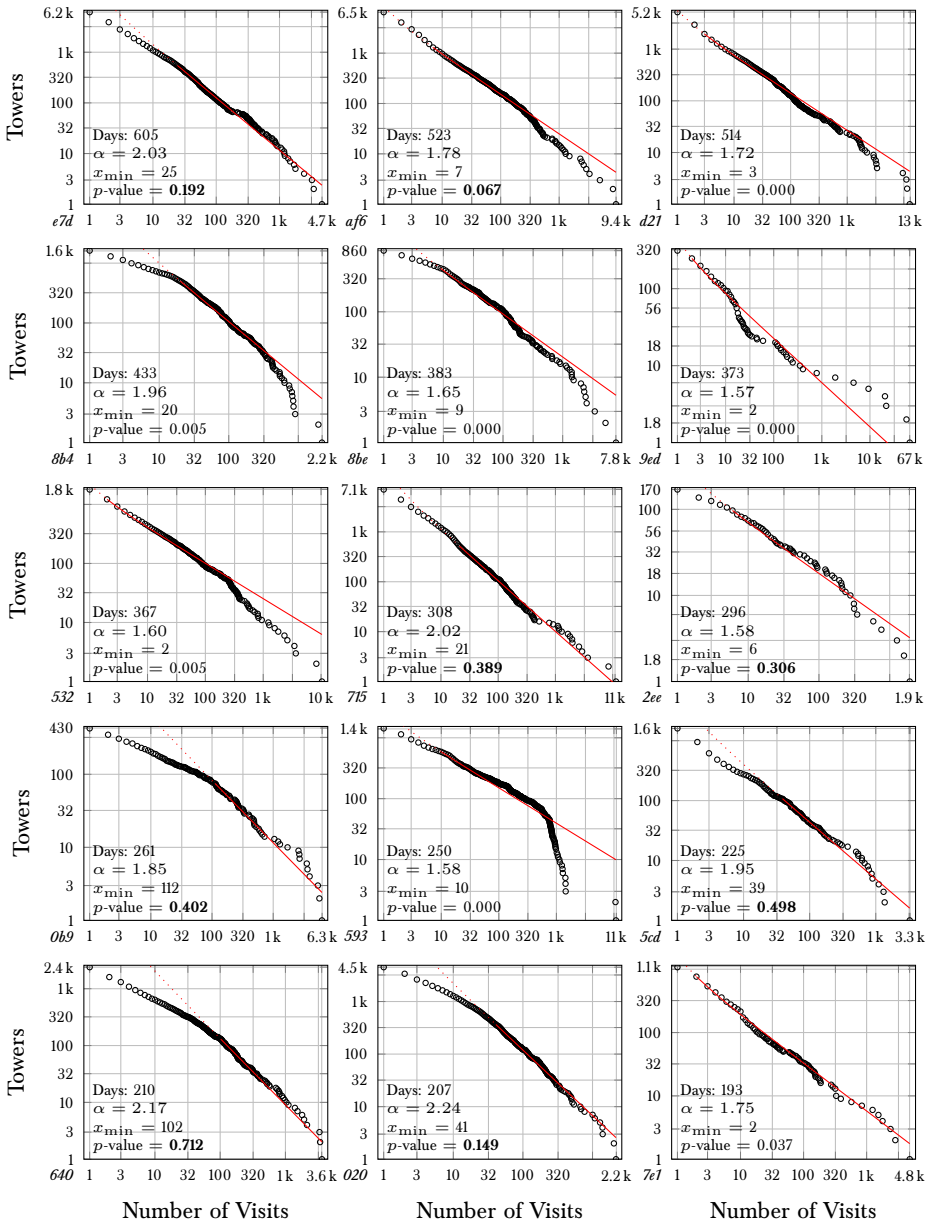


Figure 3.13: Complementary cumulative Pareto plots of the number of times a tower is visited. The x axis is the *minimum* number of times a tower is visited. The y axis is the number of towers for which this is the case. Of the 59 traces, the average α is 1.84, with a standard deviation of 0.22. 35 of the fits (59.0%) are significant at the $p = 0.05$ level. Statistically significant values are shown in bold.

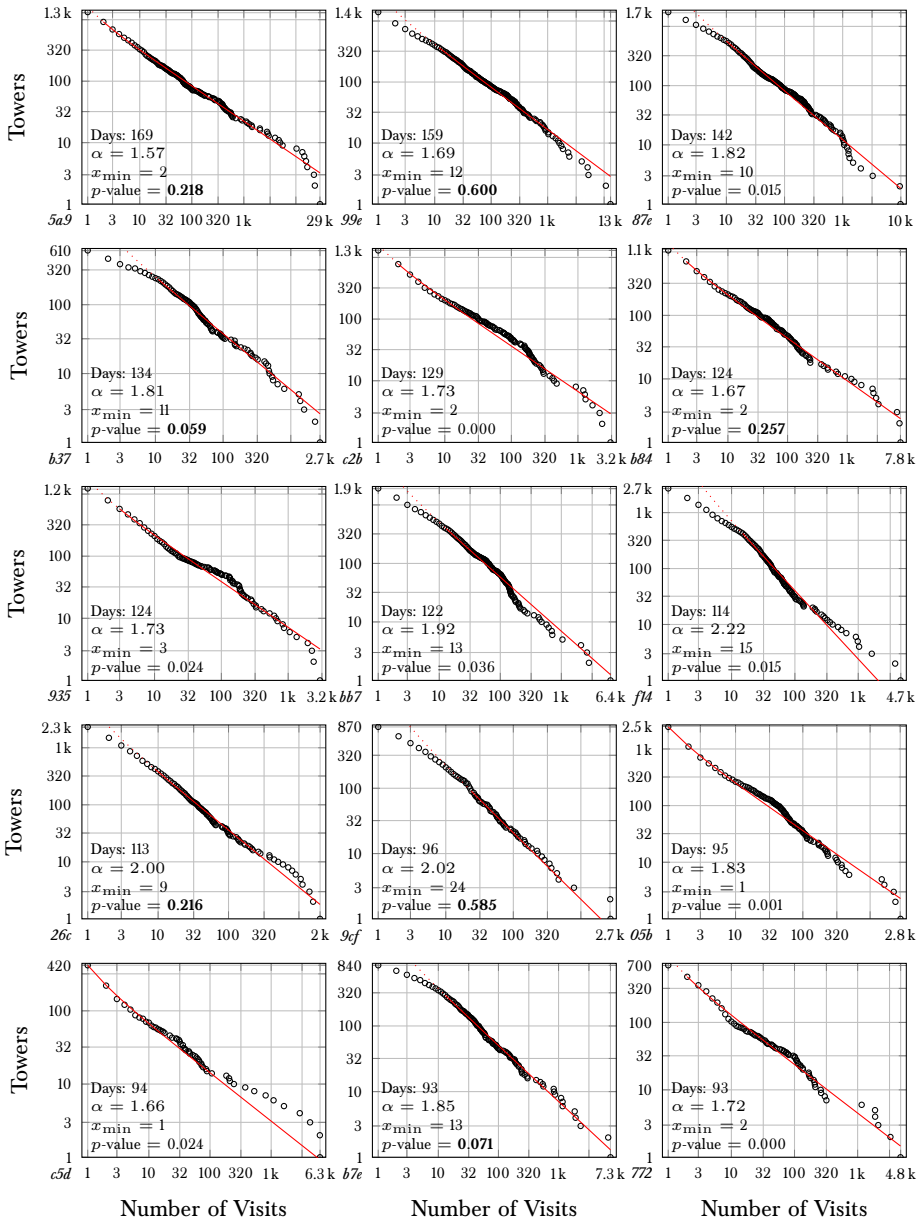


Figure 3.13 (Continued): Complementary cumulative Pareto plots of the number of times a tower is visited. The x axis is the ***minimum*** number of times a tower is visited. The y axis is the number of towers for which this is the case. Of the 59 traces, the average α is 1.84, with a standard deviation of 0.22. 35 of the fits (59.0%) are significant at the $p = 0.05$ level. Statistically significant values are shown in bold.

three visits is 58.1% (MAD: 13.2%). Interestingly, these towers correspond to just 1.0% of the total time on average (MAD: 0.7%).

Based on the number of visits to these towers and the total amount of time spent connected to them, these towers are probably along routes that are rarely taken. Given the large number of such towers, it appears relatively common for users to travel somewhere just once or a few times over the course of a year. Examples of such routes are those taken to visit a furniture store, to go the mechanic to have the car's oil changed or to travel on vacation. In practice, the longest of these routes probably don't actually have any cell towers: very long distance trips are more conveniently made by airplane than by car or train and a cell phone's radio must be turned off while in the air. This lack of cell towers when flying may explain the downward deviation.

The deviation in the tail is also generally downwards. The towers in this region correspond to those few towers that users connect to many, many times. These towers are most likely near where the users live and work. Taking a close look at Figure 3.13, we see that each user's most visited tower is visited thousands of times over the course of the trace (median: 2830, MAD: 2940). On average, this works out to dozens of visits per day (median: 28, MAD: 19). In fact, the most visited tower is visited 67 017 times (user: 9ed), which translates to 180 connections per day, on average (mean)! This is quite unlikely: most people don't go, say, shopping that often. Indeed, most people don't even leave their home that many times per day. What is actually happening here is that the user's cell phone is oscillating between two towers.

Oscillations occur when the user is near the border of two cells, i.e., somewhere where the signal strengths of the two dominant towers are comparable. Because signal quality naturally fluctuates due to environmental factors, the tower with the stronger signal frequently changes and the terminal switches towers accordingly. Although thresholds are used to prevent terminals from switching too often (thereby preventing spurious network load and energy consumption), we know from the tower sampling hypothesis that the reported towers are just observed towers, and not necessarily towers that the device connects to. We look at oscillations in more detail later in Section 3.5.

These observations suggest that the downward deviation on the right is due to an external process that imposes an upper bound on the number of visits to a tower in a given period of time. In the first instance, this threshold is probably due to user behavior: people naturally avoid unnecessary travel. Rather than making a trip to the grocery store for each individual item, most people will buy everything at once. The presence of oscillations raises this upper bound, but even they appear to have an upper limit, which is probably due to the aforementioned switching thresholds.

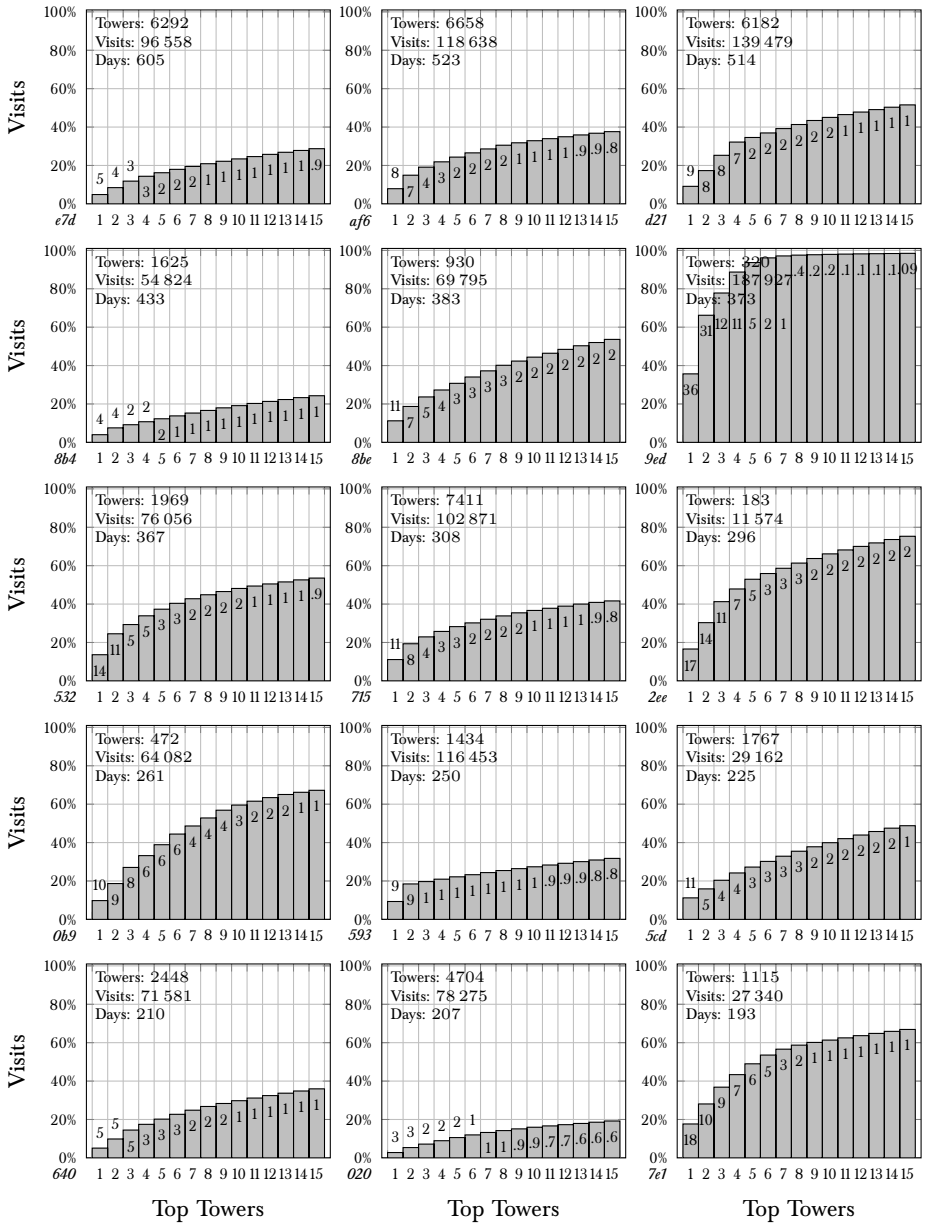


Figure 3.14: The top towers (according to the number of times they are visited) vs. the cumulative portion of total visits. The top few towers dominate. Oftentimes, the top 15 towers account for more than half of all visits. The number at the top of each bar corresponds to the portion of visits (not cumulative) to that tower.

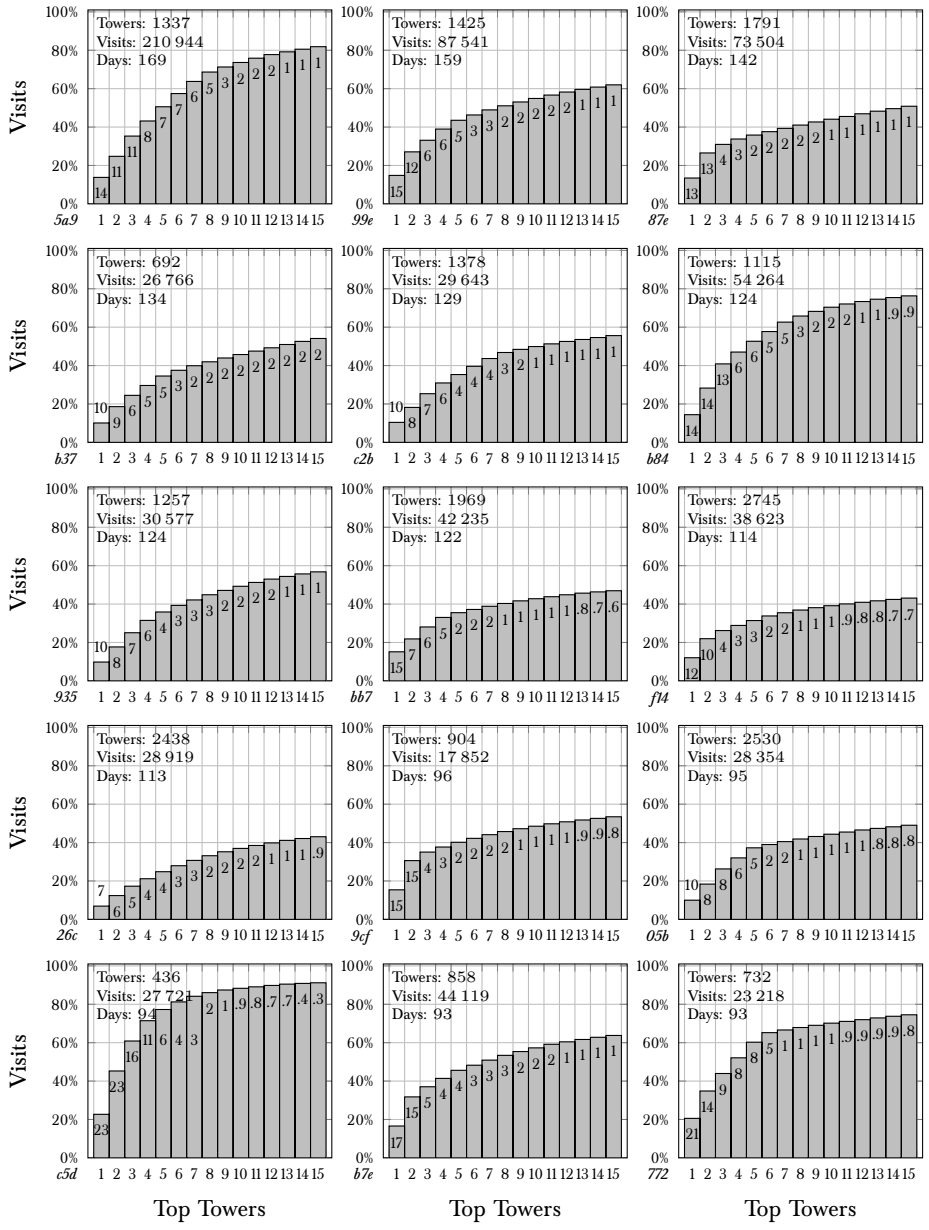


Figure 3.14 (Continued): The top towers (according to the number of times they are visited) vs. the cumulative portion of total visits. The top few towers dominate. Oftentimes, the top 15 towers account for more than half of all visits. The number at the top of each bar corresponds to the portion of visits (not cumulative) to that tower.

The top towers (according to the number of times they are visited), are clearly visited many times. To better understand their importance, we now look at the portion of visits to those towers. Figure 3.14 shows the portion of visits to the top 15 towers. Even though users visit thousands of different towers, the top 15 towers often account for more than half of the total visits. In fact, the median portion of tower visits to the top 15 towers is 57.0% (MAD: 21.0%).

Table 3.4 provides a different perspective on this idea. It shows the minimum number of towers that cover different portions of the total visits. On average (mean), just 13% of the towers that a user visits cover over 84% of the total visits. In short, just a few towers dominate in terms of the number of times a tower is visited.

The observed power law behavior should not be too surprising when considering people's daily routines. Many locations, such as a furniture store or a vacation, are visited just once or up to a few times per year. And, a few locations, such as home and work, are visited nearly every day. Finally, there are a range of locations between these extremes. The power law indicates that predicting infrequent types of activities based on the cell tower traces will be difficult. Instead, other sources of data, such as calendaring data, are needed.

Summary: The number of visits to each tower is often well described by a power law. This means that all towers are not equally important. In particular, most towers are only visited a few times. Of the remaining towers, a few are visited many, many times. These are probably involved in oscillation sequences given that they are often visited dozens or even hundreds of times per day. Predicting the many infrequent activities just using the traces will be hard.

3.4.2 Visit Dwell Times

We now look at the duration of tower visits, i.e., the amount of time spent at a tower during an individual visit.

Figure 3.15 shows complementary cumulative Pareto plots of the duration of tower visits. The x axis shows the ***minimum*** dwell time and the y axis shows the number of visits for which this is the case. Figure 3.16 shows the same data, but using a Pareto plot.

In Figure 3.15, we see that the middle part of the data—between about 3 minutes and 12 hours—roughly follows a straight line, however, the left side of the plots and the tails exhibit strong deviations. The practical result of the upper deviation is that few of the traces are consistent with an untruncated power law distribution: of the 59 traces, only 18 (31.0%) follow a power law at the $p = 0.05$ level. In fact, it is primarily the traces with at most 3 months of data for which a power law is a significant fit. In these traces, the deviation on the right is

	Number / Portion of Towers that Cover Portion of Total Visits					
User	68%	84%	92%	96%	98%	99%
e7d	261/ 4.2%	866/ 14%	1871/ 30%	3111/ 50%	4239/ 69%	5205/ 84%
af6	147/ 2.3%	587/9.0%	1640/ 25%	2939/ 45%	4135/ 64%	5321/ 82%
d21	43/0.83%	229/4.4%	657/ 13%	1515/ 29%	2682/ 52%	3788/ 73%
8b4	166/ 10%	373/ 23%	577/ 36%	777/ 48%	988/ 61%	1186/ 73%
8be	35/ 4.1%	107/ 12%	206/ 24%	331/ 38%	445/ 52%	543/ 63%
9ed	3/0.96%	4/1.3%	5/1.6%	6/1.9%	11/3.5%	25/8.0%
532	39/ 2.2%	118/6.6%	264/ 15%	496/ 28%	803/ 45%	1125/ 63%
715	237/ 3.4%	1040/ 15%	2238/ 32%	3691/ 52%	5001/ 71%	6030/ 85%
2ee	11/ 6.4%	23/ 13%	40/ 23%	64/ 37%	88/ 51%	109/ 64%
0b9	16/ 3.8%	44/ 10%	78/ 18%	113/ 27%	159/ 37%	211/ 50%
593	74/ 5.2%	145/ 10%	249/ 17%	419/ 29%	595/ 42%	797/ 56%
5cd	49/ 3.1%	158/9.8%	351/ 22%	695/ 43%	1022/ 64%	1314/ 82%
640	110/ 4.6%	283/ 12%	538/ 22%	896/ 37%	1292/ 54%	1677/ 70%
020	431/ 9.6%	1014/ 23%	1713/ 38%	2456/ 55%	3155/ 70%	3716/ 83%
7e1	17/ 1.6%	73/6.8%	229/ 21%	430/ 40%	648/ 60%	804/ 75%
5a9	8/0.60%	18/1.4%	43/3.2%	100/7.5%	216/ 16%	403/ 30%
99e	22/ 1.6%	67/4.8%	166/ 12%	316/ 23%	500/ 36%	727/ 52%
87e	42/ 2.4%	143/8.2%	367/ 21%	618/ 35%	885/ 51%	1148/ 66%
b37	25/ 4.1%	84/ 14%	164/ 27%	246/ 40%	325/ 53%	407/ 66%
c2b	33/ 2.5%	89/6.8%	217/ 17%	475/ 37%	744/ 57%	1004/ 77%
b84	9/0.85%	36/3.4%	96/9.1%	206/ 20%	395/ 37%	611/ 58%
935	30/ 2.4%	79/6.4%	239/ 19%	464/ 38%	708/ 57%	929/ 75%
bb7	84/ 4.4%	271/ 14%	539/ 28%	855/ 45%	1203/ 63%	1495/ 78%
f14	179/ 6.6%	510/ 19%	987/ 36%	1496/ 55%	1937/ 72%	2323/ 86%
26c	125/ 5.4%	433/ 19%	869/ 37%	1333/ 57%	1748/ 75%	2037/ 88%
9cf	38/ 4.4%	137/ 16%	275/ 32%	426/ 49%	571/ 65%	694/ 80%
05b	71/ 2.9%	237/9.6%	666/ 27%	1323/ 54%	1890/ 77%	2174/ 88%
c5d	4/0.95%	7/1.7%	18/4.3%	43/ 10%	93/ 22%	183/ 44%
b7e	23/ 2.7%	97/ 12%	215/ 26%	341/ 41%	459/ 55%	566/ 67%
772	9/ 1.3%	34/4.8%	92/ 13%	226/ 32%	359/ 51%	475/ 67%

Table 3.4: The minimum number / portion of towers that cover some portion of a trace's total visits. On average, just 13% of the towers that a user visits cover 84% of the total visits.

	Number / Portion of Towers that Cover Portion of Total Visits					
User	68%	84%	92%	96%	98%	99%
0a1	5 / 1.8%	21 / 7.5%	47 / 17%	80 / 29%	130 / 47%	179 / 64%
062	50 / 3.2%	162 / 11%	381 / 25%	662 / 43%	947 / 62%	1240 / 81%
c6b	46 / 6.1%	112 / 15%	198 / 26%	320 / 42%	462 / 61%	580 / 76%
949	25 / 2.3%	91 / 8.5%	235 / 22%	436 / 41%	650 / 61%	859 / 81%
8f4	2 / 0.084%	5 / 0.21%	129 / 5.4%	787 / 33%	1376 / 57%	1885 / 79%
3a7	40 / 3.7%	142 / 13%	314 / 29%	545 / 50%	754 / 69%	922 / 85%
137	103 / 7.2%	289 / 20%	572 / 40%	858 / 60%	1104 / 77%	1272 / 88%
f60	31 / 4.9%	66 / 10%	121 / 19%	226 / 36%	372 / 59%	504 / 79%
23b	33 / 3.8%	122 / 14%	278 / 32%	448 / 52%	619 / 72%	740 / 86%
3f3	111 / 8.9%	444 / 36%	733 / 59%	991 / 79%	1120 / 90%	1185 / 95%
c5e	11 / 8.5%	30 / 23%	52 / 40%	68 / 53%	83 / 64%	96 / 74%
66d	43 / 7.4%	147 / 25%	287 / 49%	418 / 72%	501 / 86%	542 / 93%
bc2	9 / 2.1%	39 / 9.0%	98 / 23%	169 / 39%	242 / 56%	304 / 70%
fb9	30 / 5.4%	98 / 18%	221 / 40%	351 / 63%	454 / 82%	506 / 91%
e6e	31 / 6.1%	85 / 17%	148 / 29%	253 / 50%	340 / 67%	403 / 79%
6c6	15 / 7.8%	44 / 23%	72 / 38%	99 / 52%	125 / 65%	147 / 77%
ff2	5 / 4.2%	10 / 8.4%	25 / 21%	36 / 30%	46 / 39%	60 / 50%
a08	3 / 2.9%	4 / 3.9%	7 / 6.8%	16 / 16%	29 / 28%	42 / 41%
3b5	2 / 2.1%	5 / 5.2%	11 / 11%	21 / 22%	38 / 40%	54 / 56%
d60	64 / 13%	161 / 32%	271 / 53%	354 / 69%	429 / 84%	470 / 92%
cd3	4 / 5.7%	9 / 13%	20 / 29%	34 / 49%	45 / 64%	54 / 77%
140	13 / 5.8%	34 / 15%	63 / 28%	96 / 43%	132 / 59%	165 / 73%
ccf	22 / 3.8%	62 / 11%	126 / 22%	207 / 36%	290 / 51%	378 / 66%
026	8 / 17%	15 / 33%	20 / 43%	25 / 54%	30 / 65%	35 / 76%
499	14 / 6.0%	42 / 18%	75 / 32%	106 / 46%	133 / 57%	161 / 69%
482	7 / 5.6%	13 / 10%	27 / 22%	47 / 38%	71 / 57%	91 / 73%
220	6 / 9.8%	9 / 15%	15 / 25%	22 / 36%	35 / 57%	47 / 77%
ef0	16 / 4.7%	40 / 12%	92 / 27%	166 / 49%	225 / 67%	279 / 83%
1ee	9 / 2.3%	52 / 13%	160 / 40%	243 / 61%	316 / 79%	358 / 90%
Mean	4.5%	13%	25%	41%	58%	72%

Table 3.4 (Continued): The minimum number / portion of towers that cover some portion of a trace's total visits. On average, just 13% of the towers that a user visits cover 84% of the total visits.

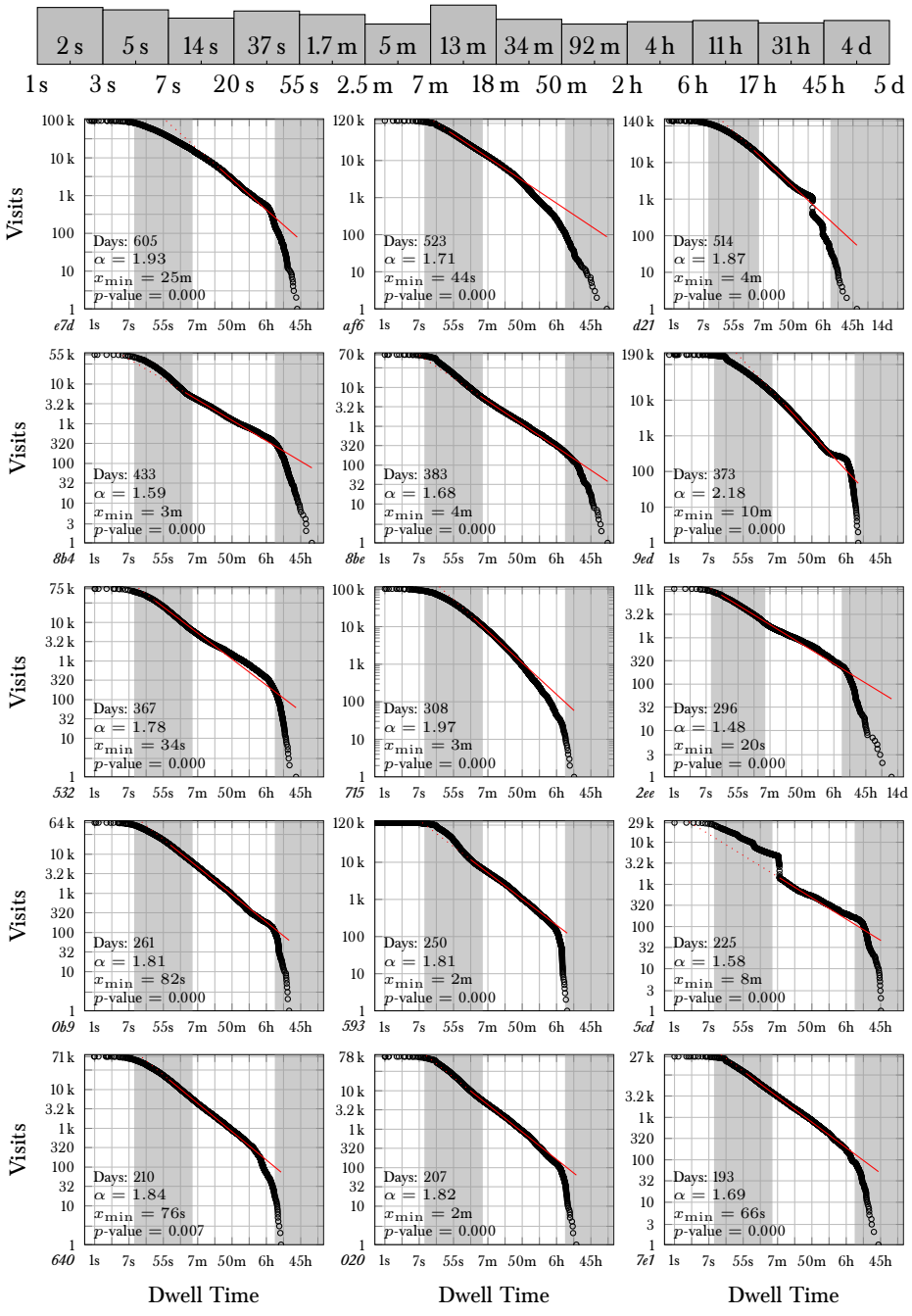


Figure 3.15: Complementary cumulative Pareto plots of the time spent during each tower visit. 59 traces, the average α is 1.86, with a standard deviation of 0.24. 18 of the fits (31.0%) are significant at the $p = 0.05$ level.

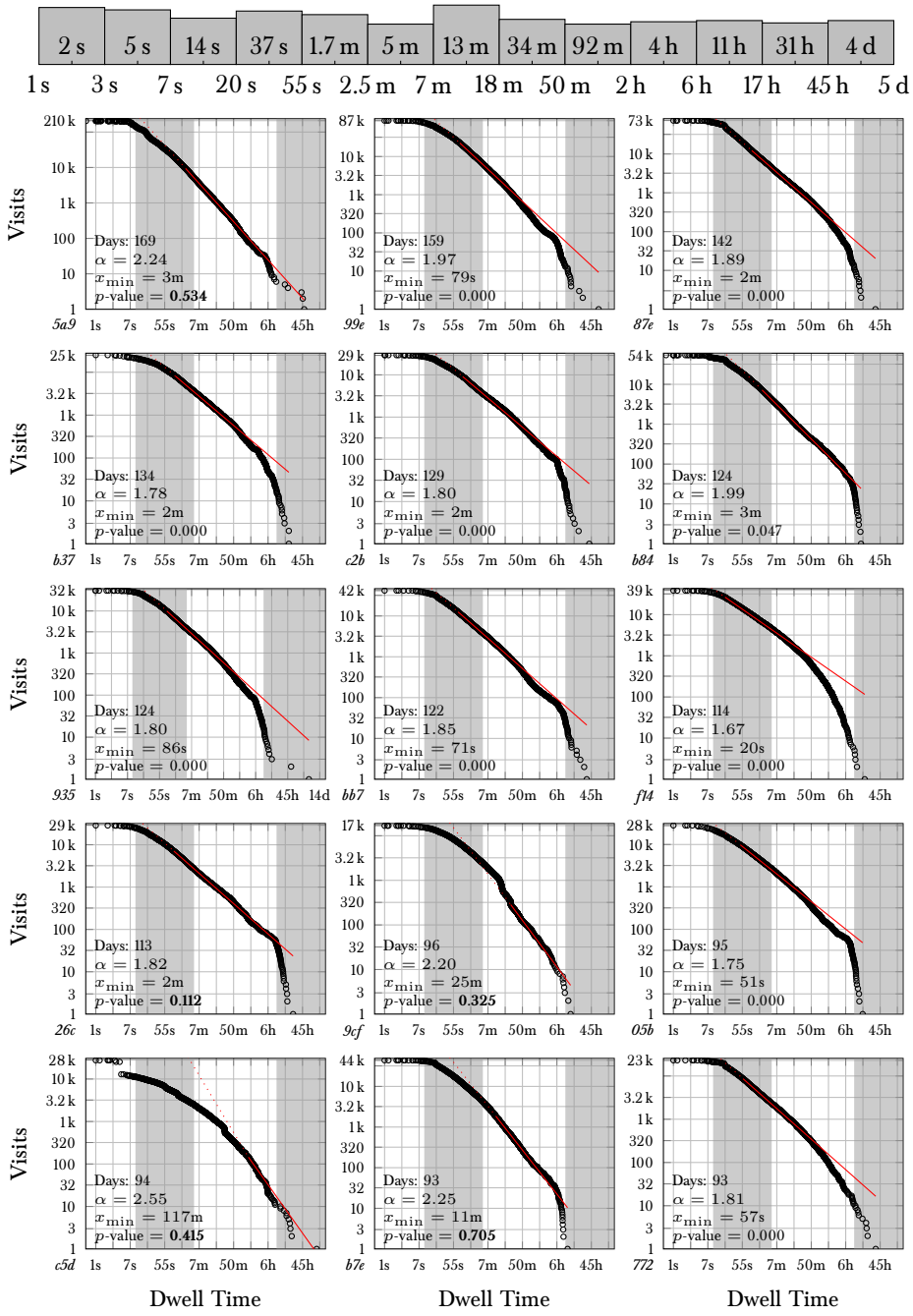


Figure 3.15 (Continued): Complementary cumulative Pareto plots of the time spent during each tower visit. 59 traces, the average α is 1.86, with a standard deviation of 0.24. 18 of the fits (31.0%) are significant at the $p = 0.05$ level.

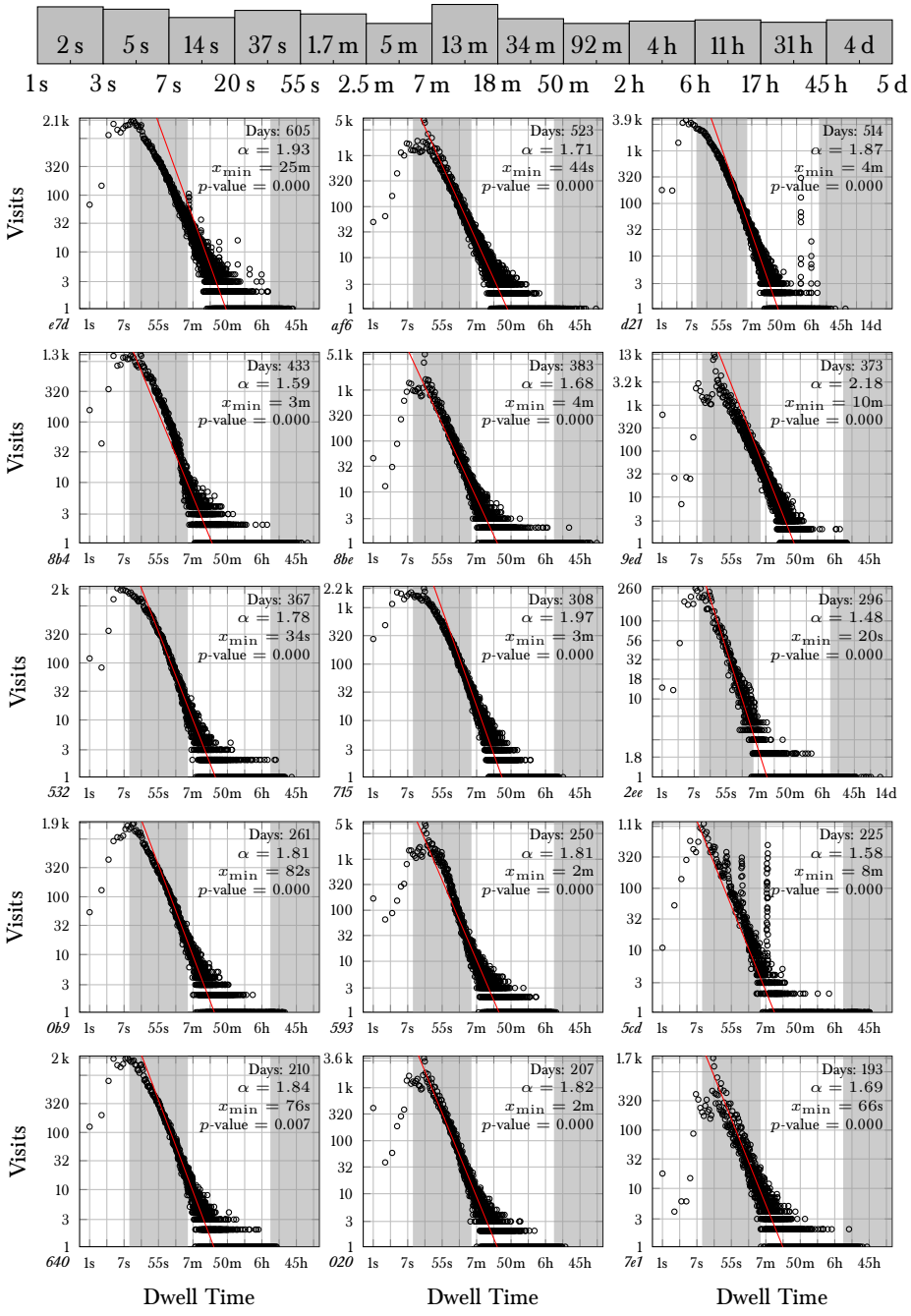


Figure 3.16: Pareto plots of the time spent during each tower visit. The x axis is the dwell time. The y axis is frequency.

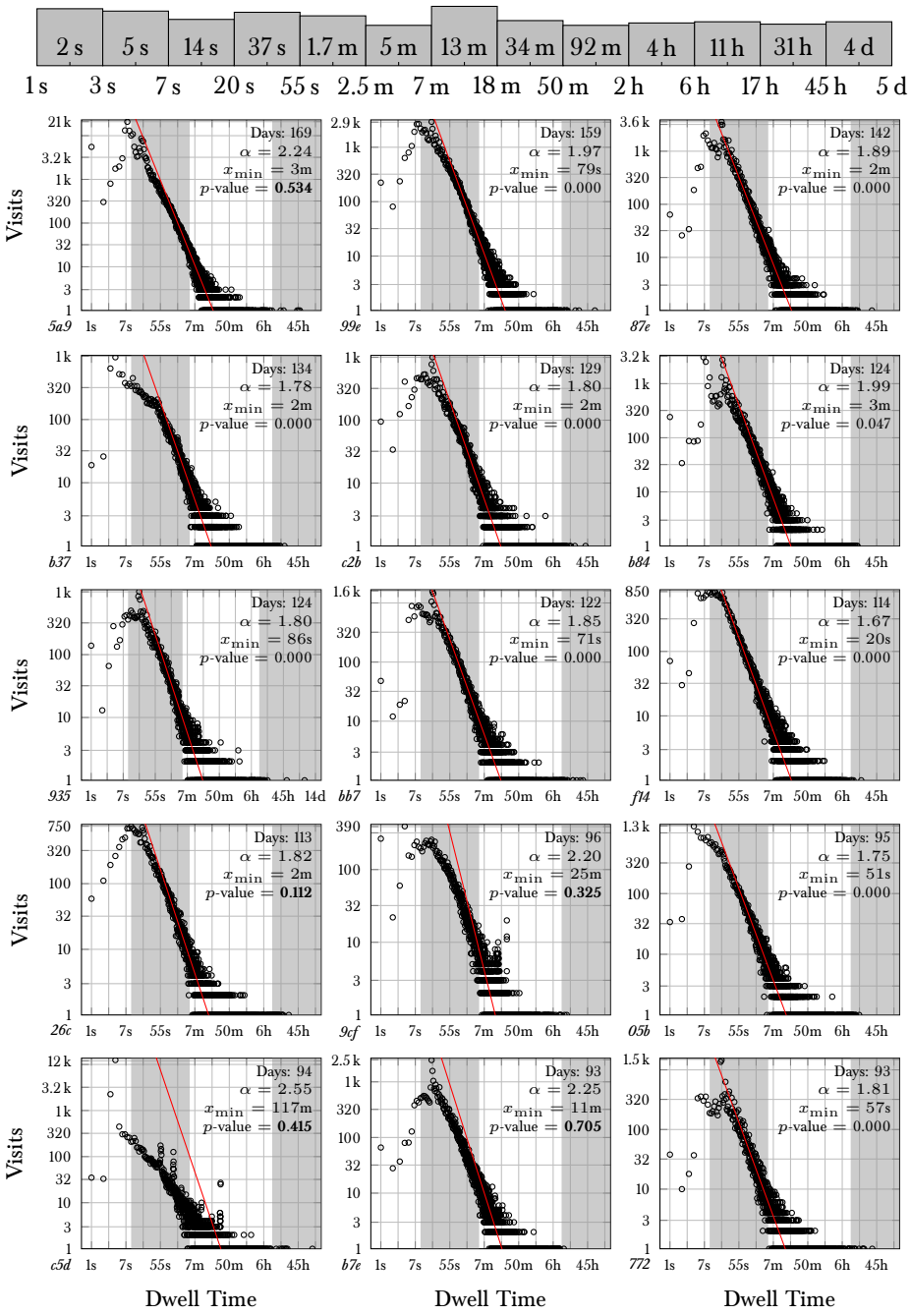


Figure 3.16 (Continued): Pareto plots of the time spent during each tower visit. The x axis is the dwell time. The y axis is frequency.

probably indistinguishable from noise. (The average value of α for all 59 traces is 1.86 with a standard deviation of 0.24.)

A close look at the data suggests four different behaviors depending on the dwell time. These regions are demarcated in the plots by gray bands in the background.

The first region consists of dwell times that are less than about 10 or 20 seconds long. The Pareto plot in Figure 3.16 makes clear that there are far fewer visits with these dwell times than would be predicted by the regression. This is not surprising. If the user needs to traverse 1000 feet of a cellular tower's area before changing to a new tower, the user would need to travel at nearly 70 miles per hour to complete the traversal in 10 seconds. In places where a user can travel that fast, e.g., along a highway, the towers will be laid out to avoid too many handoffs, i.e., the typical traversal will be longer, making such a fast traversal unlikely. Further, because changing towers requires communication, which consumes power on the client and adds load to the network, thresholds are used to prevent handoffs from occurring too frequently. Thus, a lower cutoff of at least 10 seconds seems reasonable and a sharp drop in the number of short visits, as observed in the plots, is expected.

The next region is from about 10 seconds up to approximately 3 minutes. Many of the visits in this region likely correspond to user movement. Recall Figure 3.3, which shows the induced cell tower network for a single day from several traces: the users spend at most a few minutes at each of the towers that are along routes.

The data in this region forms a slight concave downward curve on the log-log plots. We identify two possible causes that explain a deviation from the fitted power law. First, the residual effects of the aforementioned tower switching threshold should not only result in fewer tower visits, but short tower visits should be slightly longer. In this case, we would expect a slight bump around the threshold. Users 8b4, 532, 2ee and 020, for instance, exhibit such a bump. Another possibility is that the downward deviation could also be due to the "missing" towers that we identified when examining the number of times towers are visited in Section 3.4.1. We argued that when travelling very long distances, users don't see any towers, because they fly. Since these "missing" towers are along routes, we expect short dwell times. If such towers were present, they would prop up this curve.

Dwell times in the region between 3 minutes and about 10 to 12 hours primarily correspond to the locations that users visit. For most users, these dwell times appear to follow a power law as can be seen from the roughly straight line that the data forms in the CCDF plots.

There are a few exceptions to this pattern. Users 9ed, 5a9, 99e, and bb7

exhibit a nearly flat line between approximately 2 and 8 hours. This means that there are few tower visits that are between 2 and 8 hours. In the case of 9ed, this may be due to the presence of oscillations at the user's primary locations, which would result in many more short visits and fewer visits in this region. This explanation makes less sense for the other users whose traces don't have an extraordinary number of tower visits. Unlike most people who spend their work day at a single general location, these users might move around more during the day, which results in few long tower visits and correspondingly more shorter tower visits.

The final region starts at between 10 and 12 hours and also appears to roughly follow a straight line. This line, however, is much steeper than the previous one. This region consists of a very small portion of the total tower visits: the median number of visits that are at least 10 hours long is just 13% of the total number of *days* that the corresponding trace covers! This consistent drop in the number of visits is likely due to diurnal effects: most people leave the house every day whether it is to go to work or to do some errands.

Some of the visits in this region are extremely long. For instance, in user d21's trace, we see several visits to a tower for multiple weeks! Some reasons why such long visits occur include: the user left her cell phone at home while she was on vacation; she was sick at home; and, she forgot her phone at home.

As a final remark, we now consider one interesting effect. In user 5cd's CCDF plot, there is a large vertical jump at around 7 minutes. This jump corresponds to a vertical line in the pareto plot. In the latter plot, we can also easily spot other vertical lines, but they are not as pronounced. This jump means that there are a disproportionate number of visits with dwell times within a few seconds of 7 minutes and suggests that dwell times are being influenced externally, perhaps by a network parameter. This parameter is unlikely to be a handoff threshold as it is rather long.

For many users, the length of visits between 3 minutes and 12 hours appears to be distributed according to a power law. This means that these users *do not* stay connected to the same tower while at work. Instead, they connect to many towers for a shorter period of time. This is consistent with what we saw when we zoomed in on a single day of a user's trace in Figure 3.3 in Section 3.2.3.

Summary: Dwell times are not distributed according to a simple power law. The underlying function is more complicated. Ignoring outliers due to the user leaving the cell phone at home, dwell times range from approximately 10 seconds to about 12 hours. Dwell times from 10 seconds up to a few minutes likely correspond to movement. Longer dwell times likely correspond to locations at which the user spends time. The upper cutoff at 12 hours is a diurnal effect.

3.4.3 Tower Dwell Times

We now look at the amount of time spent at a tower.

Figure 3.17 shows complementary cumulative Pareto plots of the total time spent at a tower. The x axis is the minimum amount of time spent at a tower across all visits. The y axis is towers.

For most users, the data forms a roughly straight line starting with dwell times that exceed a few minutes and has a minor, downward deviation on the right. Even without explicitly accounting for the deviation in the tail, we find that 44 of the 59 traces (75.0%) are consistent with a power law at the $p = 0.05$ level. This suggests that this power law behavior is common. Further, all of the data have similar parameters: the average value of α is 1.60 with a modest standard deviation of 0.15. And, the median $x_{\min}$ is 3 minutes (170 seconds) with an median absolute deviation of 3 minutes (203.1 seconds).

The downward divergence of dwell times less than a minute long is probably again due to the tower switching threshold. The lower cutoff of the power law observed here is consistent with the 10 second lower cutoff observed for dwell times of individual visits, which we examined in Section 3.4.2. We see a slightly higher value here, because we are aggregating the time spent from all visits to a given tower.

The divergence in the tail is due to the top 10 to 20 towers and probably reflects the user's lifestyle. These are a user's primary towers—the towers around home and work. The behavior of the data in this region reflects the user's daily routine. For instance, it reflects whether the user works from home or commutes to work.

If a user works from home, then we expect the user to spend some 20 hours per day there. The remaining handful of hours are for other activities, such as running errands, going to the gym, and visiting friends. This behavior results in a sharp drop between the top and the second towers. This is what we see in 8b4 and 2ee's traces, for instance. Their top towers each account for more than 60% of the total time and their second towers account for about 10%.

For users who go someplace during the day, we expect to see a smoother transition, which is more in line with a power law: such users are likely home for 12 hours per day (to eat, sleep, etc.) and at work for about 8 hours. This pattern can be seen in, for instance, e7d's and 715's traces.

The fact that the amount of time spent at a tower appears consistent with a power law means that the top towers are most important. Further, it means that the majority of towers are ephemeral and are not visited for a significant amount of time.

To better understand the implications of the power law behavior, we plotted

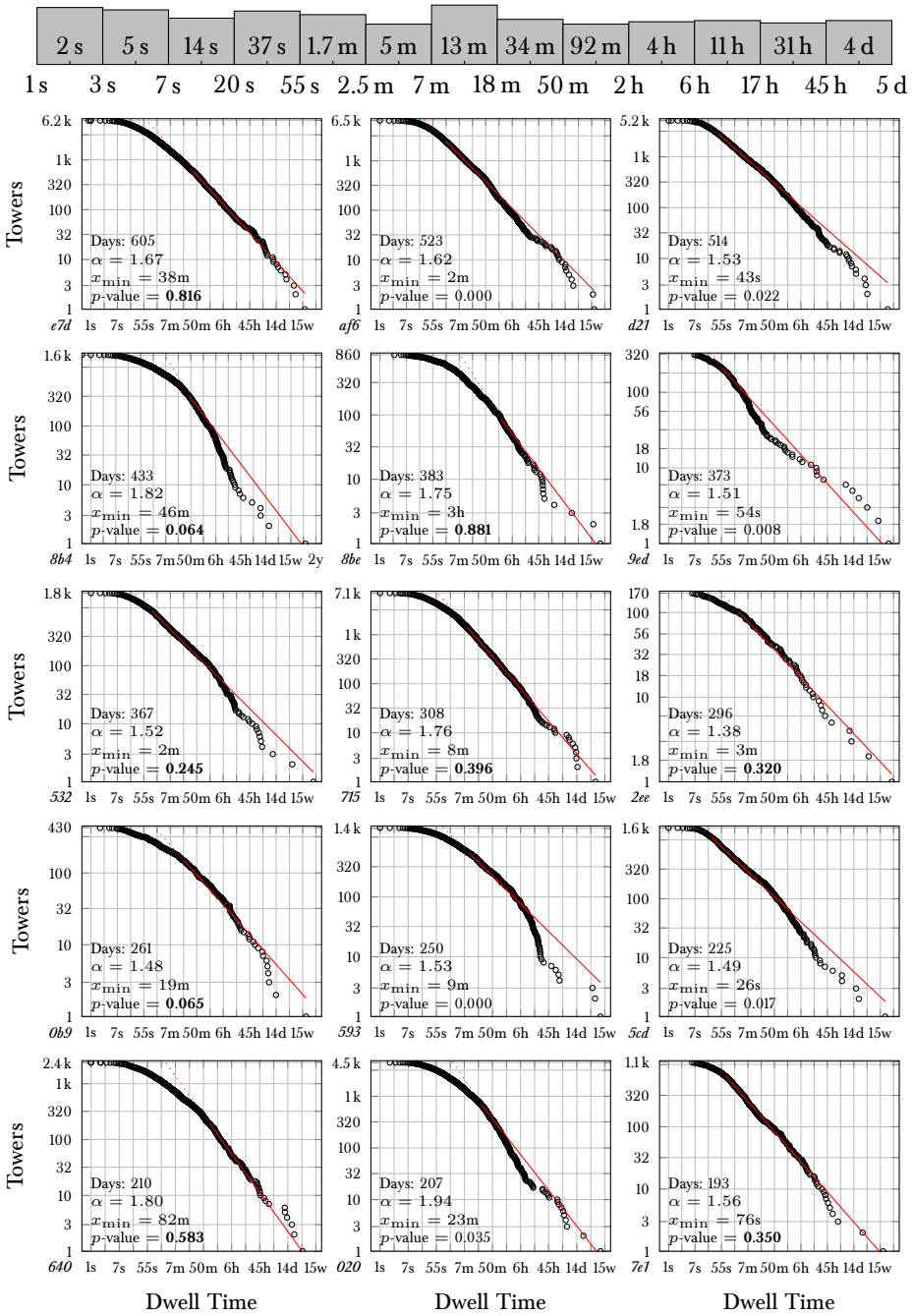


Figure 3.17: Complementary cumulative Pareto plots of the total time spent at a tower (i.e., across all visits). Of the 59 traces, the average α is 1.60, with a standard deviation of 0.15. 44 of the fits (75.0%) are significant at the $p = 0.05$ level.

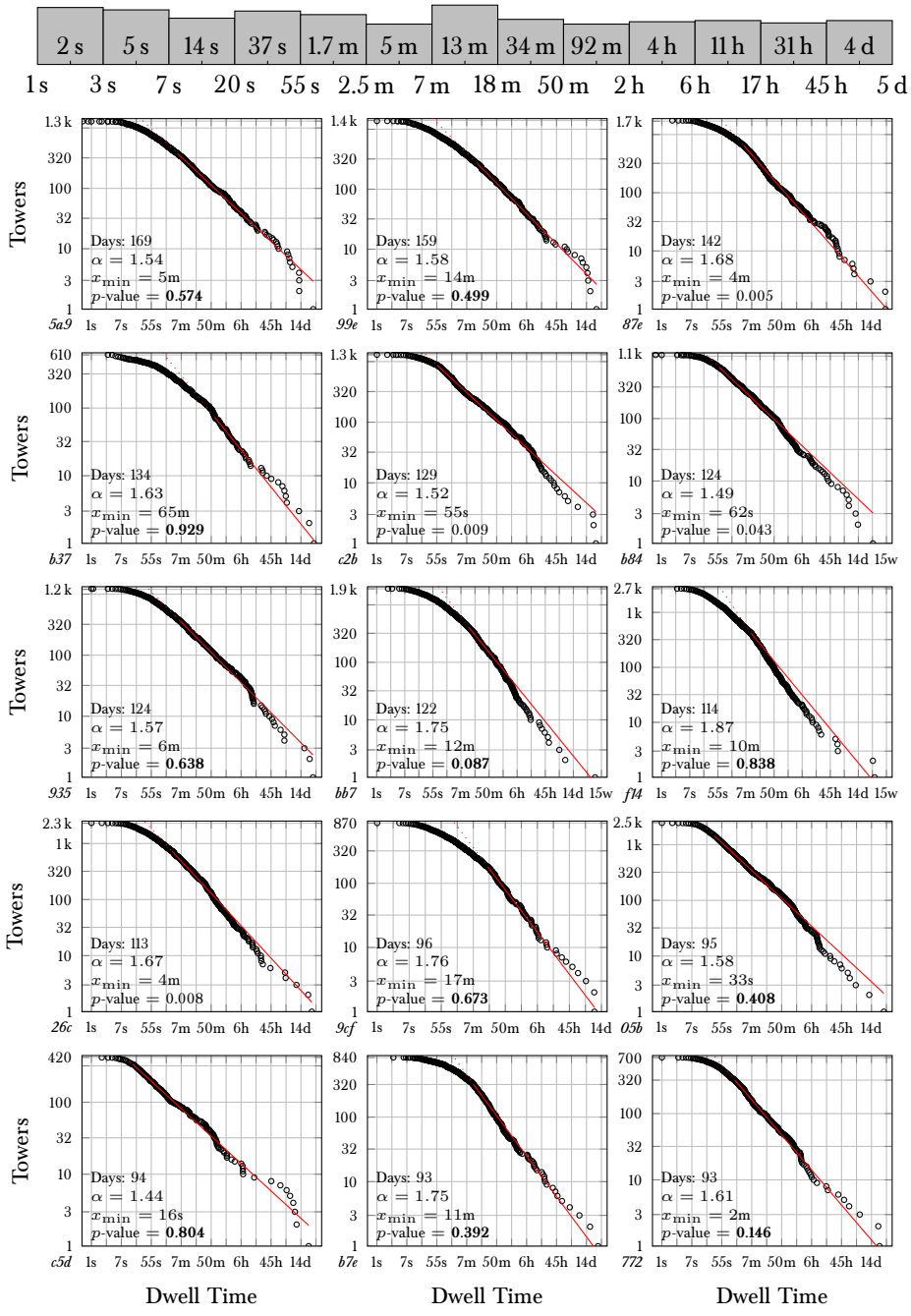


Figure 3.17 (Continued): Complementary cumulative Pareto plots of the total time spent at a tower (i.e., across all visits). Of the 59 traces, the average α is 1.60, with a standard deviation of 0.15. 44 of the fits (75.0%) are significant at the $p = 0.05$ level.

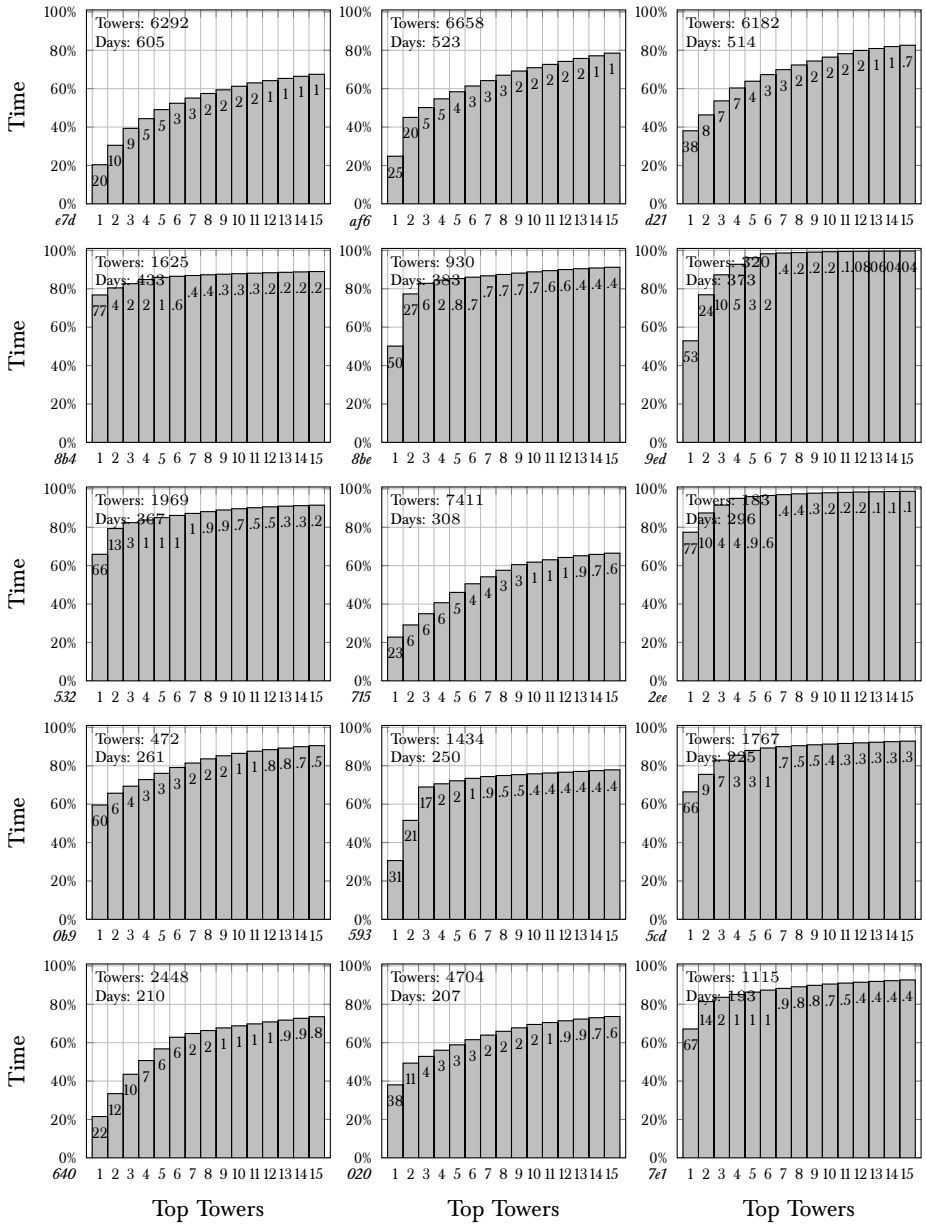


Figure 3.18: The portion of time spent at the top 15 towers (ranked according to the time spent at the towers) vs. the cumulative portion of the total time connected to the top towers. The numbers near the top of the bars indicate the portion of time spent at that tower (*not cumulative*). The plots illustrate that the top few towers dominate in terms of the amount time the user spends at them.

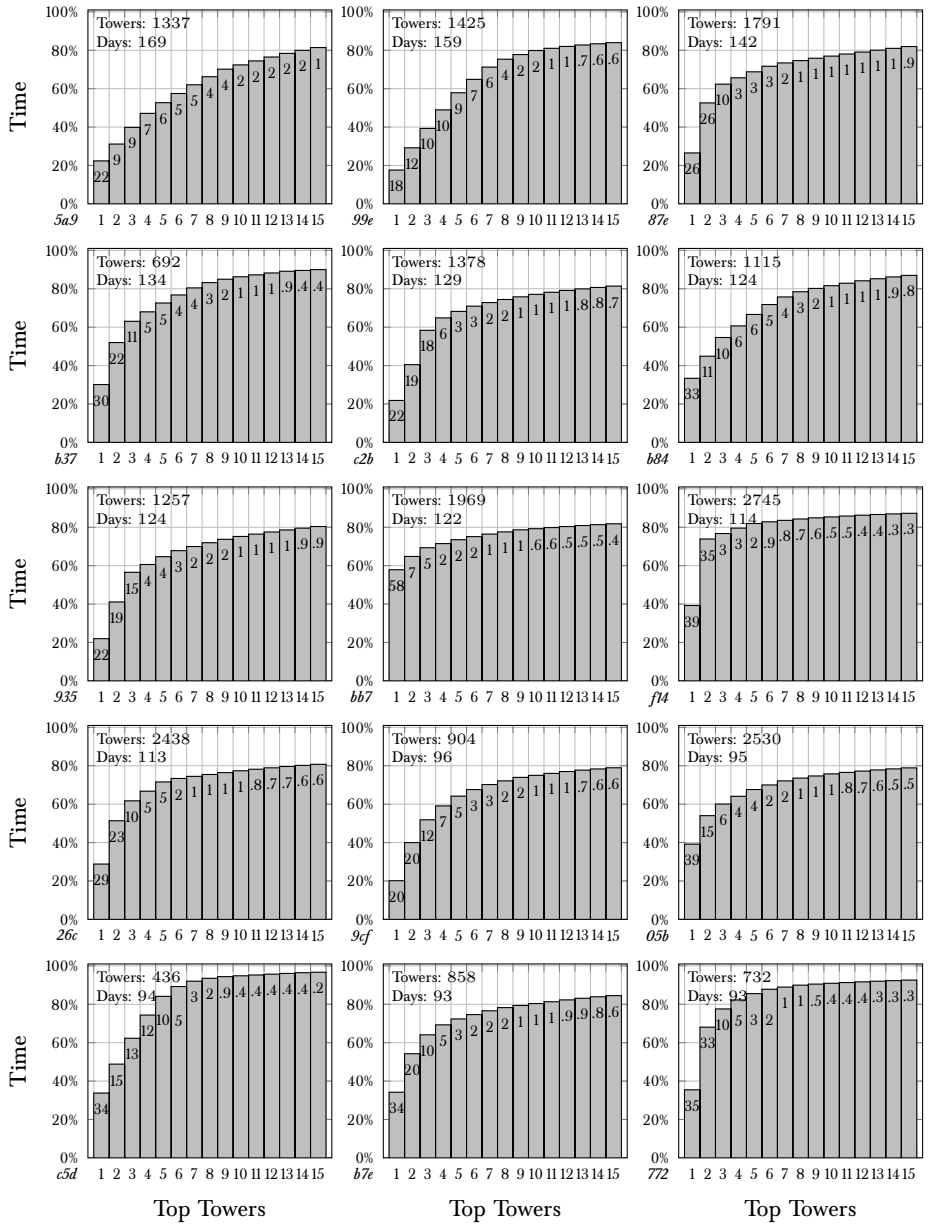


Figure 3.18 (Continued): The portion of time spent at the top 15 towers (ranked according to the time spent at the towers) vs. the cumulative portion of the total time connected to the top towers. The numbers near the top of the bars indicate the portion of time spent at that tower (*not cumulative*). The plots illustrate that the top few towers dominate in terms of the amount time the user spends at them.

the portion of time spent at the top 15 towers in Figure 3.18. We excluded all tower visits that are longer than two days to avoid inflating the amount of influence that the top towers have. (There are 47 such visits across 11 users.) We assume in these cases that the user left her cell phone at home, because she forgot it or she went on vacation and didn't want to be bothered by it. This assumption is conservative in the sense that it is entirely possible that someone will stay at home for a few days if they are sick, for instance. As such, the plots most likely underestimate the amount of time spent at the top towers.

Looking at Figure 3.18, we see that all users spend a lot of time at their top 15 towers. In fact, all of these users spend two-thirds of their time there and most of them spend over 80% of their time there. To keep this in perspective, most users visit hundreds or thousands of towers over the course of their trace (in the plot, the exact number is shown in an inset). In other words, the top 15 towers account for 80% of the time, but correspond to about 1% of the towers that a user ever sees.

Table 3.5 provides a different perspective on this idea. This table shows the minimum number of towers that cover some amount of the total time. We see that on average 2.0% of the towers that the user visits cover over 84% of the total time. To cover 99% of the total time—23 hours and 45 minutes per day—only 28% of the towers are needed.

Summary: The total amount of time spent at a tower closely follows a power law. The lower cutoff is a few minutes. This is easily attributed to the cell tower switching threshold. There is some divergence in the tail, which we attributed to the user's daily routine. When a person works from home their top tower dominates more than the model predicts, for instance. We saw that users spend effectively no time at most of the towers as the power law model predicts. Concretely, only 28% of the towers cover 99% of the total time—23 hours and 45 minutes per day. As with the distribution of tower visits, which we looked at in Section 3.4.1, we can use this knowledge to identify the importance of a tower. Those towers at which the user spends little time—the majority—can be mostly ignored.

3.4.4 Visit Dwell Times by Tower

We now examine the visit dwell time broken down by individual towers.

Since most users visit hundreds or thousands of towers, we can't easily visualize how visits to all towers behave. But, because the number of visits and the amount of time spent at towers are consistent with a power law (as discussed in Section 3.4.1 and Section 3.4.3), we don't need to: we can just concentrate on the most important towers—those which the user visits most often and those which

	Number / Portion of Towers that Cover Portion of Total Time					
User	68%	84%	92%	96%	98%	99%
e7d	16/ 0.26%	46/0.75%	141/ 2.3%	361/5.9%	739/ 12%	1300/ 21%
af6	9/ 0.14%	22/0.34%	95/ 1.5%	324/5.0%	726/ 11%	1439/ 22%
d21	6/ 0.12%	15/0.29%	49/0.95%	141/2.7%	322/6.2%	606/ 12%
8b4	1/0.062%	4/0.25%	39/ 2.4%	128/7.9%	265/ 16%	420/ 26%
8be	2/ 0.23%	3/0.35%	16/ 1.9%	47/5.5%	102/ 12%	174/ 20%
9ed	2/ 0.64%	3/0.96%	4/ 1.3%	5/1.6%	6/1.9%	9/2.9%
532	2/ 0.11%	5/0.28%	18/ 1.0%	57/3.2%	116/6.5%	210/ 12%
715	19/ 0.27%	123/ 1.7%	434/ 6.1%	1067/ 15%	1931/ 27%	2861/ 41%
2ee	1/ 0.58%	2/ 1.2%	3/ 1.8%	5/2.9%	10/5.8%	17/9.9%
0b9	3/ 0.70%	9/ 2.1%	20/ 4.7%	37/8.7%	59/ 14%	88/ 21%
593	3/ 0.21%	36/ 2.5%	93/ 6.5%	173/ 12%	283/ 20%	430/ 30%
5cd	2/ 0.12%	4/0.25%	14/0.87%	42/2.6%	96/6.0%	173/ 11%
640	10/ 0.42%	42/ 1.8%	133/ 5.6%	278/ 12%	461/ 19%	716/ 30%
020	10/ 0.22%	107/ 2.4%	403/ 9.0%	829/ 18%	1367/ 30%	1957/ 44%
7e1	2/ 0.19%	4/0.37%	14/ 1.3%	33/3.1%	71/6.6%	133/ 12%
5a9	9/ 0.68%	18/ 1.4%	38/ 2.9%	76/5.7%	144/ 11%	255/ 19%
99e	7/ 0.50%	15/ 1.1%	40/ 2.9%	87/6.2%	166/ 12%	279/ 20%
87e	6/ 0.34%	19/ 1.1%	44/ 2.5%	113/6.5%	281/ 16%	502/ 29%
b37	5/ 0.81%	9/ 1.5%	21/ 3.4%	50/8.1%	96/ 16%	146/ 24%
c2b	6/ 0.46%	21/ 1.6%	48/ 3.7%	90/6.9%	161/ 12%	278/ 21%
b84	6/ 0.57%	13/ 1.2%	29/ 2.7%	69/6.5%	129/ 12%	220/ 21%
935	6/ 0.49%	20/ 1.6%	46/ 3.7%	93/7.5%	176/ 14%	303/ 25%
bb7	3/ 0.16%	23/ 1.2%	95/ 5.0%	233/ 12%	428/ 22%	659/ 34%
f14	2/0.074%	8/0.30%	55/ 2.0%	215/7.9%	459/ 17%	793/ 29%
26c	5/ 0.21%	24/ 1.0%	96/ 4.1%	245/ 11%	478/ 21%	770/ 33%
9cf	6/ 0.69%	21/ 2.4%	60/ 6.9%	133/ 15%	217/ 25%	315/ 36%
05b	6/ 0.24%	28/ 1.1%	82/ 3.3%	174/7.1%	348/ 14%	653/ 27%
c5d	4/ 0.95%	6/ 1.4%	8/ 1.9%	13/3.1%	26/6.2%	45/ 11%
b7e	4/ 0.48%	15/ 1.8%	49/ 5.8%	136/ 16%	247/ 29%	357/ 42%
772	2/ 0.28%	5/0.71%	14/ 2.0%	38/5.4%	77/ 11%	153/ 22%

Table 3.5: The minimum number of towers that cover a portion of a trace's total time. We excluded tower visits that were longer than two days based on the assumption that the user was not actually carrying the cell phone around with her during this time. On average, just 2.0% of the towers that the user visits cover over 84% of the total time. To cover 99% of the total time—23 hours and 45 minutes per day—only 28% of the towers are needed.

	Number / Portion of Towers that Cover Portion of Total Time					
User	68%	84%	92%	96%	98%	99%
0a1	1/ 0.36%	1/0.36%	3/1.1%	6/2.2%	17/6.1%	35/13%
062	4/ 0.26%	22/ 1.4%	61/4.0%	124/8.1%	251/ 16%	457/30%
c6b	2/ 0.26%	5/0.66%	25/3.3%	63/8.3%	117/ 15%	193/25%
949	6/ 0.56%	15/ 1.4%	28/2.6%	62/5.8%	146/ 14%	279/26%
8f4	2/0.084%	6/0.25%	24/1.0%	188/7.9%	618/ 26%	1075/45%
3a7	12/ 1.1%	24/ 2.2%	55/5.0%	114/ 10%	207/ 19%	330/30%
137	3/ 0.21%	11/0.76%	46/3.2%	126/8.8%	265/ 18%	474/33%
f60	2/ 0.31%	8/ 1.3%	26/4.1%	53/8.3%	93/ 15%	158/25%
23b	2/ 0.23%	5/0.58%	18/2.1%	57/6.6%	135/ 16%	255/30%
3f3	4/ 0.32%	12/0.96%	26/2.1%	107/8.6%	325/ 26%	562/45%
c5e	3/ 2.3%	4/ 3.1%	9/7.0%	17/ 13%	34/ 26%	48/37%
66d	12/ 2.1%	20/ 3.4%	32/5.5%	62/ 11%	117/ 20%	201/34%
bc2	4/ 0.93%	10/ 2.3%	23/5.3%	49/ 11%	90/ 21%	141/33%
fb9	7/ 1.3%	14/ 2.5%	24/4.3%	47/8.4%	93/ 17%	168/30%
e6e	3/ 0.59%	12/ 2.4%	33/6.5%	63/ 12%	106/ 21%	160/31%
6c6	3/ 1.6%	5/ 2.6%	7/3.6%	18/9.4%	35/ 18%	53/28%
ff2	4/ 3.4%	5/ 4.2%	8/6.7%	13/ 11%	23/ 19%	33/28%
a08	2/ 1.9%	4/ 3.9%	6/5.8%	15/ 15%	25/ 24%	36/35%
3b5	2/ 2.1%	2/ 2.1%	3/3.1%	8/8.3%	15/ 16%	22/23%
d60	1/ 0.20%	4/0.78%	24/4.7%	75/ 15%	153/ 30%	233/46%
cd3	1/ 1.4%	1/ 1.4%	2/2.9%	5/7.1%	9/ 13%	14/20%
140	5/ 2.2%	11/ 4.9%	23/ 10%	42/ 19%	67/ 30%	92/41%
ccf	8/ 1.4%	33/ 5.7%	80/ 14%	137/ 24%	201/ 35%	268/47%
026	2/ 4.3%	4/ 8.7%	7/ 15%	10/ 22%	13/ 28%	15/33%
499	7/ 3.0%	13/ 5.6%	29/ 12%	51/ 22%	76/ 33%	99/43%
482	4/ 3.2%	6/ 4.8%	10/8.0%	14/ 11%	18/ 14%	28/22%
220	3/ 4.9%	5/ 8.2%	7/ 11%	10/ 16%	12/ 20%	15/25%
ef0	10/ 3.0%	16/ 4.7%	30/8.9%	56/ 17%	95/ 28%	144/43%
1ee	4/ 1.0%	5/ 1.3%	11/2.8%	21/5.3%	37/9.3%	100/25%
Mean	0.94%	2.0%	4.5%	9.5%	17%	28%

Table 3.5 (Continued): The minimum number of towers that cover a portion of a trace's total time. We excluded tower visits that were longer than two days based on the assumption that the user was not actually carrying the cell phone around with her during this time. On average, just 2.0% of the towers that the user visits cover over 84% of the total time. To cover 99% of the total time—23 hours and 45 minutes per day—only 28% of the towers are needed.

the user spends the most time at.

Figure 3.19 shows 15 towers from the top four users: the top 9 towers according to total time spent at the tower and the top 6 towers according to the total number of visits (excluding those that we already selected). Each plot is a histogram of the amount of time spent during each visit to the tower. The x axis is the same for all plots to facilitate comparisons across towers. The number at the bottom of each bar indicates the portion of the total time spent at this tower that the visits in this range constitute.

Information about the towers is inset in the plots. The first line shows the total amount of time spent at the tower and the tower's rank according to this metric. This is followed by the total number of visits and the tower's rank according to this metric. Then, the number of days on which the tower is visited at least once is shown. Finally, the median number of visits per day for days on which the tower is visited at least once is displayed as well as the corresponding median absolute deviation.

Looking at the histograms, there appear to be two primary types of towers. There are those towers with visits whose durations cover many orders of magnitude and those whose visits have much less variance and tend to be short (at most a few minutes). The former probably correspond to towers at important locations and the latter to towers along a route. The distribution of dwell times for location towers often appears to be consistent with a log-normal distribution.

A close look at the graphs reveals that in nearly all cases the mode is less than about 7 minutes. In other words, short visits dominate even for the towers that users spend the most time at. In terms of the amount of time spent at a tower, however, long visits generally dominate. Consider, for instance, e7d's top tower: 68% of the visits are less than 10 minutes long. However, visits longer than an hour account for nearly 90% of the time spent at the tower.

This distribution is surprising if we assume a fixed location is generally covered by a single tower. However, it is consistent with what we observed in Figure 3.3, in which we zoomed in on a single day of a user's trace and looked at the induced cell tower network. Specifically, when at, say, work, users don't typically stay connected to a single tower, but move between a group of towers that cover the location.

A consequence of this distribution is that if we use the mode for inferring the dwell time, then we will tend to guess at most a few minutes. In a strict sense, this is correct—it is the most frequent dwell time—however, it is probably not what we want. Instead, we want to know when the user will leave the location, i.e., the tower community. For this we need to take a broader view than a single cell phone tower when making predictions. This observation is reassuring: it provides strong evidence that cell towers provide more than ample resolution for

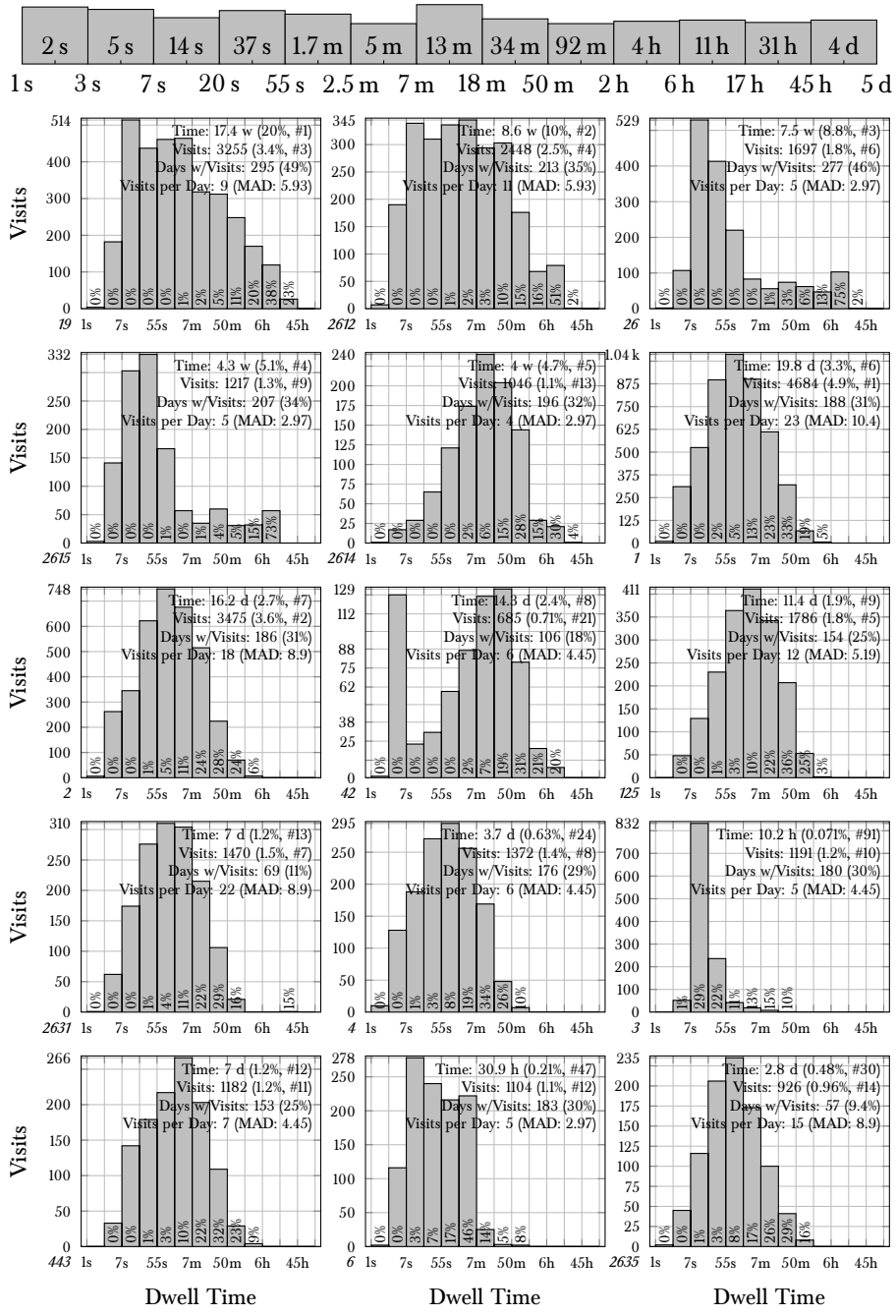


Figure 3.19: Visit dwell time histogram for **e7d**'s top towers. The number at the bottom of each bar is the portion of *time* that the visits constitute.

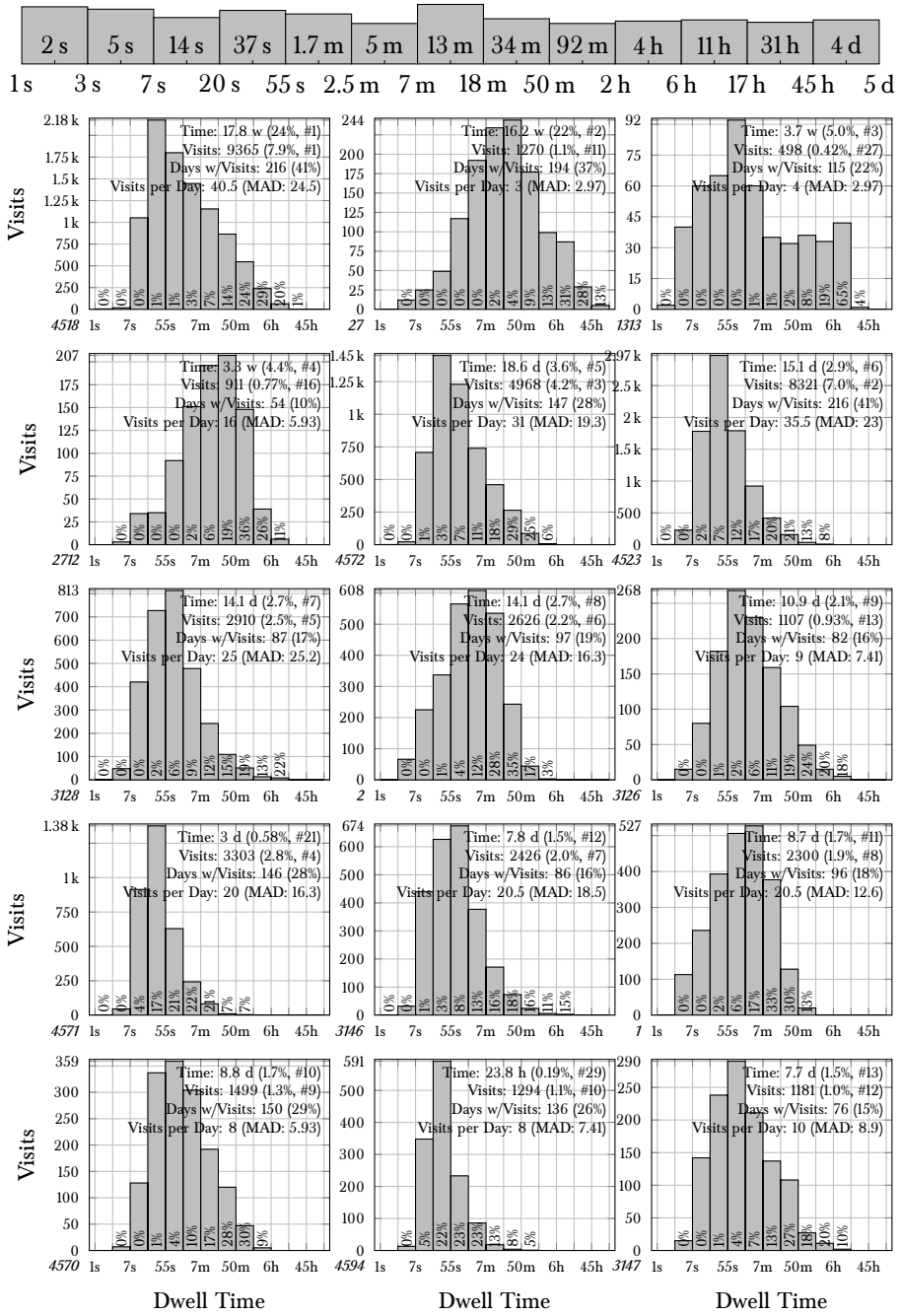


Figure 3.19 (Continued): Visit dwell time histogram for **af6**'s top towers. The number at the bottom of each bar is the portion of *time* that the visits constitute.

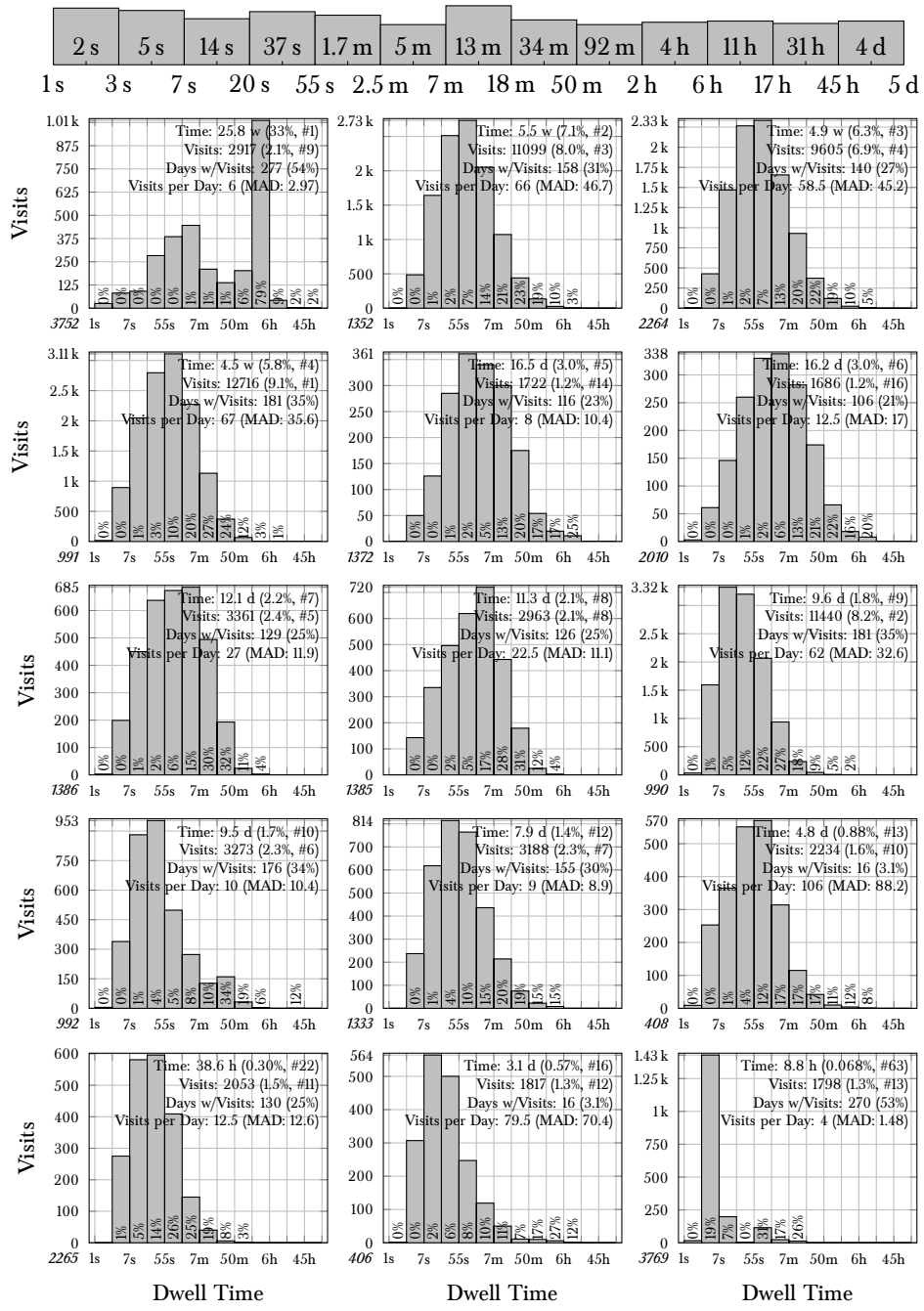


Figure 3.19 (Continued): Visit dwell time histogram for **d21**'s top towers. The number at the bottom of each bar is the portion of *time* that the visits constitute.

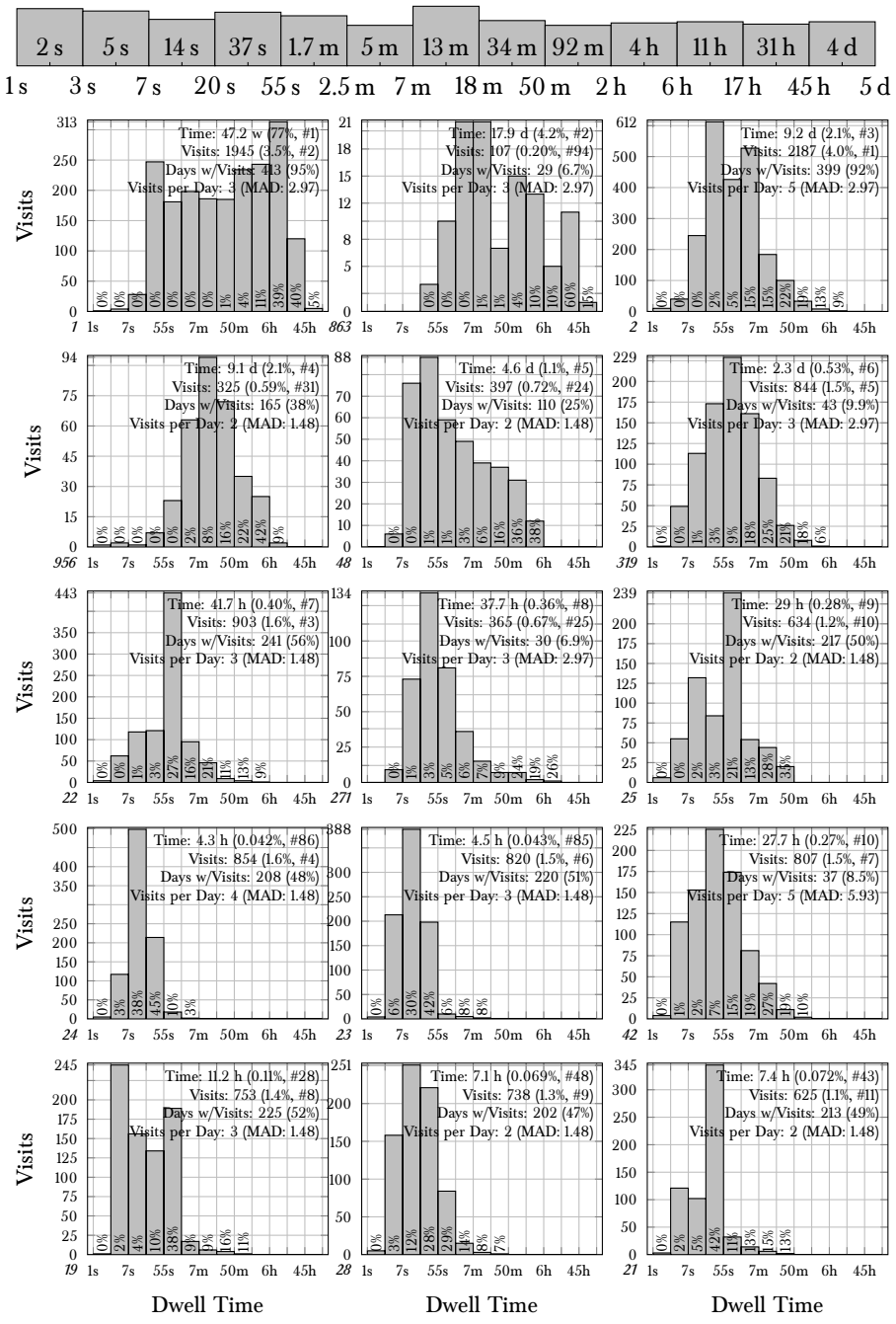


Figure 3.19 (Continued): Visit dwell time histogram for **8b4**'s top towers. The number at the bottom of each bar is the portion of *time* that the visits constitute.

determining the user's location.

The most important towers are typically visited a few times per day (when they are visited at all). However, there are many towers that are visited dozens of times per day. This is the case for af6's top tower (4518), which is visited more than 40 times per day! This tower is most likely involved in oscillation sequences.

Summary: Most dwell times are short. Those that are long dominate in terms of the total time spent at the tower. The presence of many short dwell times provides further evidence that locations are generally covered by multiple towers (as discussed initially in Section 3.2.3). The number of times towers are visited per day suggests that many towers frequently end up in oscillation sequences.

3.4.5 Conclusions

Tower Importance: Number of visits, visit dwell time and tower dwell time are all consistent with right heavy-tailed distributions (and often, at least partially, with a power law distribution). This means that users only spend time at a few locations; most locations are visited in passing.

One way to reduce the state space, is to only consider predicting towers at which the user spends a lot of time. However, due to their high information content (they are unexpected) occasionally visited towers are still useful to predict where the user is going.

Another way to save space is to age data. But, as we saw in the discussion of regime changes in Section 3.2.1, users sometimes return to old regimes. This suggests partitioning the data by regimes. Before making a prediction, we would identify what major regime the user's current location belongs to and only use that data.

Locations and Towers: We observed that the most important towers tend to have many short visits. We attributed this to locations being covered by multiple towers, which we first observed in Section 3.2.3 when examining induced cell tower networks. Happily, this strongly suggests that the resolution of towers is sufficient for determining the user's current location. However, it means that trying to determine the user's future location by simply predicting the string of subsequent towers and the time spent there is probably won't work well. This is because the towers at locations will form highly connected components and the most probable successor will always be drawn from the same component. That is, the user will rarely leave the location. Two possible approaches to dealing with this problem are to cluster towers (i.e., identify tower communities) and to incorporate more context, such as the time of day, when considering a successor

tower. Whereas the former approach reduces the available resolution, the latter approach could result in a state space explosion. Losing some information is acceptable if all of the towers correspond to the same location and context, e.g., different parts of the location have the same network connectivity.

3.5 Oscillations

When we examined the distribution of tower visits in Section 3.4.1, we observed that some towers are visited tens or hundreds of times a day. We suggested that the user was not actually moving, but instead the cell phone was oscillating between two towers. This can happen if a cell phone is on the border of two towers and neither tower's signal is significantly stronger than the other's. In this case, because signal strength constantly fluctuates, the tower whose signal is stronger changes frequently and the cell phone switches towers accordingly. Although thresholds are used to reduce the amount of switching, it still occurs frequently. For our purposes, oscillations are a problem, because they obscure the user's location and movement. However, they actually increase the resolution by allowing us to distinguish more locations.

In this section, we examine oscillations in more detail. We first examine a pair of towers that are involved in many oscillation sequences. We then evaluate whether oscillation sequences are sufficiently common that they need to be dealt with explicitly. Finally, we briefly consider how to deal with oscillation sequences.

3.5.1 An Example

User 9ed's top two towers (in terms of both the number of visits and the total time connected to the towers) are examples of oscillation towers: these two towers, towers 2 and 4, often end up in long alternating sequences with each other.

Figure 3.20 shows histograms of the dwell time of visits to these two towers. (These plots show the same type of data as the plots in Figure 3.19.) We first observe that over the 373 days that we collected data, the towers were visited 67 017 times and 57 370 times, respectively. This is a lot: the median number of visits on days that each of the towers are visited at least once is 167 and 142, respectively (MAD: 63.8 and 63.8). The histograms also reveal that most of these visits are around a minute long. This huge number of short tower visits strongly suggests that the user is not actually moving, but that the cell phone is oscillating.

The histograms also show some long tower visits. This suggests that the cell phone does not always oscillate when connected to these towers. Tower 2, for instance, includes 123 visits that are at least 6 hours long. Given that the

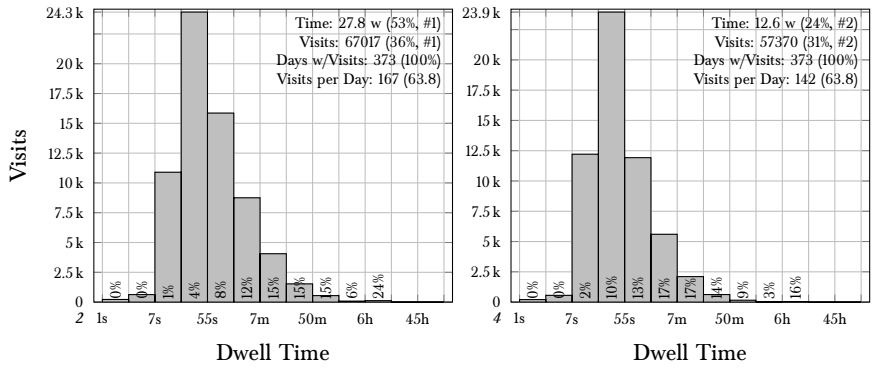


Figure 3.20: Visit dwell time histogram for **9ed**'s top oscillation pair. The number at the bottom of each bar is the portion of *time* that the visits constitute. Inset in each figure are the total time and total visits to the tower as well as the number of days with visits and the median visits per day for days on which the tower was visited at least once and the corresponding MAD.

user spends more than half of her time connected to this tower, this tower likely corresponds to her home. It could be that the oscillations primarily occur when the user is, say, at the front of the house, but not in her bedroom.

Table 3.6 shows tower 2's alternating partners—those towers with which it is in at least one long ($l \geq 7$) alternating sequence. The towers are ordered by the portion of their tower visits that are in a long alternating sequence with tower 2. We just consider those alternating sequences that are at least 7 visits long, because we expect that these are less likely to be due to user movement. This doesn't mean that we expect short alternating sequences to be due primarily to user movement, however.

Looking at the table, we see that tower 2 has several alternating partners. The first one is tower 4, which is the other tower we just examined. Of the 57 370 visits to this tower, 43.0% of them are in a long alternating sequence with tower 2 and the longest such sequence is 177 visits long. These alternating sequences are most likely oscillation sequences.

Note: because we are only considering alternating sequences that are at least 7 visits long, this estimate is almost certainly conservative: short alternating sequences involving these two towers are probably oscillation sequences as well. These probably don't get very long, because the user is not at the location long enough for many oscillations to occur. For instance, the user might drop something off at home and a few minutes later be on his way again. If the median oscillation time is, say, 5 minutes, then the resulting oscillation sequence

Oscillation Partner	Total Visits	Sequences, Length ≥ 7	Max. Length	Tower Visits in Sequences		Partner Visits in Sequences	
4	57 370	3709	177	25 110 /	37%	24 742 /	43%
3	21 760	760	178	4948 /	7.4%	4776 /	22%
1	20 511	681	85	4087 /	6.1%	3977 /	19%
8	9333	185	42	971 /	1.4%	918 /	9.8%
7	783	8	16	35 /	0.052%	35 /	4.5%
6	1899	18	14	76 /	0.11%	72 /	3.8%

Table 3.6: User 9ed's top tower's alternating partners. The partner towers are sorted by the portion of visits involved in an alternating sequence of length 7 or more with the top tower. 43.0% of the top alternating partner's visits are in an alternating sequence of length 7 or more with the top tower.

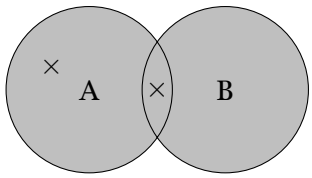


Figure 3.21: A single tower can cover multiple locations. In this case, tower A covers two locations. The first location is entirely within tower A's area and the second is on the border of tower A's and tower B's areas. In the latter case, we can expect to see oscillations. The oscillations allow us to easily distinguish the two locations: if they were both in tower A's area, this would be much harder.

will only be a few visits long.

The next three alternating partners are involved in hundreds of long alternating sequences with tower 2 and the longest alternating sequences provide strong evidence that the user is at least sometimes on the border of these cells and tower 2. The fact that a significantly smaller portion of visits to these towers results in alternating sequences with tower 2 is probably because the towers cover multiple locations and the other location(s) are frequently visited. This idea is shown in Figure 3.21.

The alternating sequences involving the last two towers could be attributed to user movement. The longest alternating sequence is 16 tower visits. This could conceivably happen if the user really moves back and forth several times or if the user walks along the border of the cells, as illustrated in Figure 3.22. However, we suspect that these alternating sequences are more likely to be oscillation

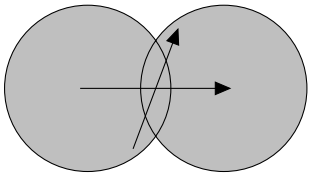


Figure 3.22: Two possible cell tower transitions. The horizontal transition quickly moves from the first tower to the second. The near vertical transition spends more time in the two towers' overlapping area allowing more opportunities for oscillations.

sequences.

Summary: There are at least some towers that are often involved in oscillation sequences. Importantly, visits to these cells don't always end up in oscillations. Indeed, these towers sometimes have multiple oscillation partners. This means that a single cell tower sometimes covers multiple locations. Thus, although oscillations are annoying from an analysis perspective, they can help us distinguish more locations.

3.5.2 The Importance of Oscillations

So far we've looked at a single user and a few, albeit poignant, oscillation pairs thereby establishing that oscillation sequences exist. Now we consider whether oscillation sequences are sufficiently important that they should be dealt with or whether they can be safely ignored.

Table 3.7 shows the number of alternating pairs for which there are at least 5 alternating sequences that are 7 visits long. In this section, we assume that tower pairs that are involved in so many long alternating sequences are oscillation pairs and that any long alternating sequences involving those oscillation pairs are oscillation sequences. Note: recall from Table 3.6 that a single tower can be part of multiple pairs and thus the number of oscillation towers is not necessarily twice the number of oscillation pairs.

The table also shows the number of alternating towers whose p value exceeds different thresholds. We define a tower's p value to be the portion of visits that are part of an oscillation sequence with a particular oscillation partner. That is, if tower A is visited 100 times and 95 of those visits are in an oscillation sequence with B, then A's p-value, which we denote $p_{A \bowtie B}$, is 0.95. If tower B is visited 1000 times and 95 of those visits are in an oscillation sequence with A, then B's p value, $p_{B \bowtie A}$, is 0.095. An oscillation pair's p value is the minimum of the two tower's p values.

	Towers		Oscillation Pairs / Towers			Oscillation Sequences	
User	Total	≥ 50 Visits	Total	$p \geq 0.5$	$p \geq 0.7$	Count	Visits
e7d	6169	260	33 / 52	10 / 16	6 / 8	934	11 404 / 12%
af6	6506	280	47 / 76	32 / 39	8 / 14	1987	37 190 / 31%
d21	5181	250	53 / 73	36 / 40	18 / 25	3286	53 627 / 38%
8b4	1619	217	19 / 33	6 / 10	2 / 4	372	5970 / 11%
8be	861	160	33 / 46	8 / 13	2 / 5	1111	22 049 / 32%
9ed	312	21	23 / 15	4 / 4	2 / 2	6442	82 391 / 44%
532	1788	158	36 / 51	14 / 18	6 / 10	1389	27 319 / 36%
715	7057	218	50 / 77	30 / 34	20 / 24	1389	27 165 / 26%
2ee	170	30	14 / 21	2 / 5	2 / 2	200	2821 / 24%
0b9	425	109	43 / 49	4 / 7	2 / 4	1345	14 893 / 23%
593	1427	239	32 / 55	12 / 17	10 / 13	943	27 469 / 24%
5cd	1604	89	13 / 21	10 / 11	8 / 9	265	4747 / 16%
640	2391	233	51 / 73	14 / 23	6 / 10	1018	17 563 / 25%
020	4497	288	26 / 40	10 / 12	6 / 8	382	7208 / 9.2%
7e1	1076	49	16 / 24	8 / 17	4 / 8	664	13 264 / 49%
5a9	1330	126	51 / 54	6 / 7	2 / 2	3151	38 013 / 18%
99e	1399	133	26 / 40	8 / 13	6 / 6	1665	29 719 / 34%
87e	1742	138	35 / 48	8 / 9	6 / 6	1354	23 715 / 32%
b37	614	64	12 / 20	4 / 8	4 / 4	478	8352 / 31%
c2b	1299	76	11 / 18	6 / 8	2 / 4	454	6225 / 21%
b84	1054	82	23 / 24	4 / 8	2 / 4	1445	22 783 / 42%
935	1233	69	13 / 19	6 / 6	2 / 4	550	9144 / 30%
bb7	1916	123	26 / 38	18 / 22	14 / 17	727	9092 / 22%
f14	2708	84	15 / 21	4 / 5	2 / 3	448	5851 / 15%
26c	2325	67	16 / 23	6 / 10	2 / 4	407	6708 / 23%
9cf	871	43	4 / 8	4 / 5	4 / 4	211	6230 / 35%
05b	2456	82	18 / 31	16 / 19	8 / 13	422	11 840 / 42%
c5d	419	25	10 / 15	4 / 9	2 / 3	430	20 361 / 73%
b7e	840	89	22 / 35	14 / 18	12 / 14	1213	19 959 / 45%
772	704	41	12 / 16	4 / 7	2 / 3	646	11 057 / 48%

Table 3.7: The number of significant oscillation partners in each trace. Two towers are considered to be significant oscillation partners if at least one of them is visited at least 50 times and there are at least 5 oscillations sequences of length 7. p is the portion of tower visits that are part of an oscillation sequence with a particular oscillation partner. An oscillation pair's p value is the maximum of the two tower's p value with respect to the other tower.

	Towers		Oscillation Pairs / Towers			Oscillation Sequences	
User	Total	≥ 50 Visits	Total	$p \geq 0.5$	$p \geq 0.7$	Count	Visits
0a1	278	22	5 / 9	6 / 6	0 / 2	356	5047 / 44%
062	1538	88	18 / 30	10 / 12	10 / 11	469	6403 / 21%
c6b	759	67	8 / 13	2 / 2	0 / 0	234	3217 / 18%
949	1066	50	18 / 29	12 / 15	8 / 10	513	10 215 / 49%
8f4	2393	22	11 / 15	4 / 8	4 / 5	1377	30 450 / 60%
3a7	1089	45	15 / 22	8 / 11	4 / 6	202	2882 / 17%
137	1438	53	8 / 16	8 / 10	8 / 8	122	5649 / 34%
f60	634	54	10 / 16	2 / 4	0 / 2	124	1640 / 12%
23b	859	26	10 / 15	8 / 11	4 / 7	273	5296 / 44%
3f3	1248	21	10 / 19	16 / 16	12 / 12	93	2411 / 37%
c5e	128	18	5 / 9	4 / 5	4 / 4	91	1648 / 28%
66d	582	20	5 / 10	2 / 3	2 / 2	33	420 / 10%
bc2	431	26	10 / 13	4 / 6	0 / 2	327	5863 / 42%
fb9	556	20	9 / 17	8 / 9	4 / 4	94	1842 / 36%
e6e	507	29	6 / 12	2 / 3	0 / 1	109	1262 / 12%
6c6	191	12	5 / 8	2 / 5	0 / 1	97	1415 / 30%
ff2	118	18	8 / 7	2 / 3	2 / 2	334	4782 / 47%
a08	102	12	7 / 8	2 / 3	0 / 1	895	13 106 / 56%
3b5	95	10	6 / 7	4 / 4	4 / 4	231	4921 / 71%
d60	510	10	2 / 4	2 / 3	0 / 2	32	631 / 15%
cd3	69	5	3 / 5	2 / 2	0 / 1	50	798 / 48%
140	224	22	11 / 12	2 / 2	0 / 0	95	1145 / 15%
ccf	573	69	32 / 42	10 / 17	4 / 7	451	15 127 / 58%
026	45	9	2 / 4	0 / 1	0 / 0	17	164 / 10%
499	231	23	5 / 8	4 / 4	4 / 4	186	3516 / 41%
482	124	11	5 / 9	4 / 5	4 / 4	58	1472 / 43%
220	60	7	1 / 2	2 / 2	2 / 2	9	473 / 32%
ef0	336	20	5 / 9	2 / 2	2 / 2	54	758 / 13%
lee	398	12	5 / 6	2 / 3	0 / 1	95	1655 / 40%
Median	859	50	12 / 19	6 / 8	4 / 4	407	6230 / 32%

Table 3.7 (Continued): The number of significant oscillation partners in each trace. Two towers are considered to be significant oscillation partners if at least one of them is visited at least 50 times and there are at least 5 oscillations sequences of length 7. p is the portion of tower visits that are part of an oscillation sequence with a particular oscillation partner. An oscillation pair's p value is the maximum of the two tower's p value with respect to the other tower.

The table reveals that both oscillation pairs and oscillation sequences are common. The median number of oscillation pairs per user is 12 and the median portion of tower visits that are part of an oscillation sequence is 32.0% (MAD: 17.0%). In other words, at least a third of all tower visits are part of long alternating sequences! If we consider all alternating sequences that are at least three visits long involving the selected oscillation pairs, then this portion increases to 45.0% (MAD: 18.0%).

Summary: Oscillation sequences are very common and can't be ignored as background noise. Nearly half of all tower visits are involved in oscillation sequences with likely oscillation pairs.

3.5.3 Collapsing Oscillation Sequences

Table 3.7 also shows that most oscillation pairs do not have a large p value. When both towers in an oscillation pair have a large p value, then a visit to one of the towers implies an oscillation sequence with the other. These towers can simply be treated as a single tower. When at least one of the towers in an oscillation pair has a low p value, then the towers cover multiple locations. This idea was illustrated in Figure 3.21. If we simply collapse these towers, then we may lose information. Indeed, if we were very liberal in our criteria, we'll end up with just a single tower!

We now examine the effects of collapsing oscillation pairs with high p values. Figure 3.23 shows histograms of visit dwell times for several oscillation pairs from different users. The left two plots in each row are the oscillation pair. They each have with thousands of short visits. The last plot in each row shows the result of collapsing the two towers. That is, it shows what happens when we treat the two towers as if they were a single tower. In this case, each alternating sequence involving the two towers are collapsed to a single tower visit and singleton visits are relabeled appropriately.

Collapsing the towers provides a significantly different impression of how often users visit a location and how long they stay there. First, the total number of visits decreases by at least an order of magnitude. Second, whereas in the left two plots, most visits are less than a minute long, in the right plot there are (proportionally) many more multi-hour visits.

Summary: Most oscillation pairs don't have large p values, which means that the towers probably cover more than one location and that collapsing towers will lose some information. When collapsing tower pairs with a large p value, we see that the number of visits decreases by at least an order of magnitude, however, short visits still dominate in terms of the total number of visits.

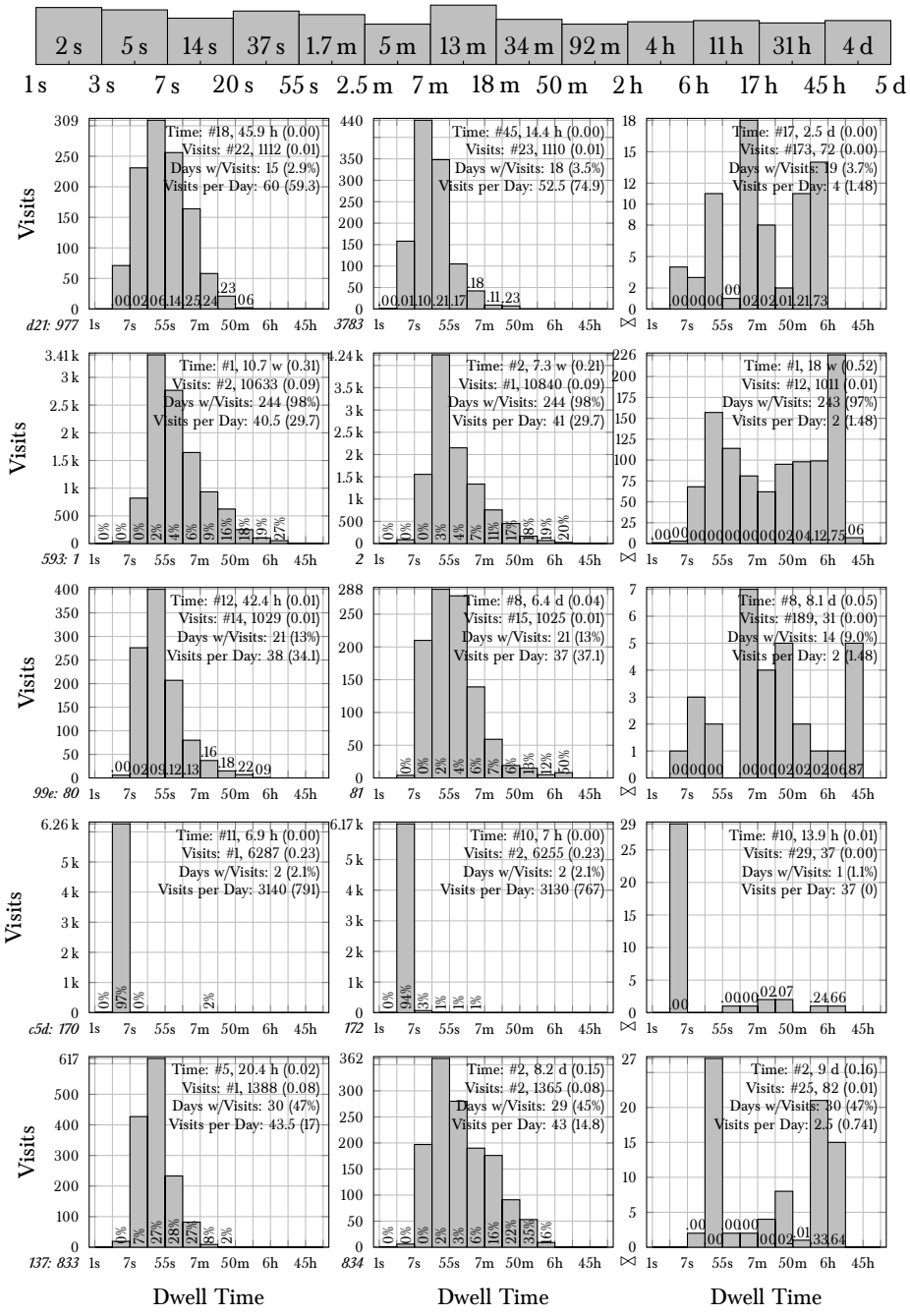


Figure 3.23: Histograms of visit dwell times. The left two plots in each row are an oscillation pair and the right plot is the result of collapsing the two towers. Collapsing the towers drastically reduces the number of observed tower visits, and, in particular, the number of short tower visits.

3.5.4 Alternating Sequence Distribution

We now briefly take a look at the distribution of the number of alternating sequences for each oscillation pair. Figures 3.24 and 3.25 show complementary cumulative Pareto plots of the frequency of each oscillation type (i.e., the number of oscillation sequences for each oscillation pair) broken down by user. Figure 3.24 considers alternating sequences that are at least 3 visits long; Figure 3.25 considers alternating sequences that are at least 7 visits long.

We see that a power law is a statistically significant fit in nearly all cases. Concretely, for the former case, 50 of the 59 are significant at the $p = 0.05$ level (85.0%); and, for the latter case 55 are significant at the $p = 0.05$ level (93.0%). This means that for a given user we will observe a few oscillation pairs with many oscillation sequences, but we will also see many oscillations pairs with just a few sequences.

Interestingly, there is little divergence on the left side of the plots. Indeed, the best fit $x_{\min}$ is often 1 (means: 1.41 and 1.21 with standard deviation 0.83 and 0.55, respectively). We would expect that user movement causes an upward deviation. However, since we don't see this, we suspect that user movement that results in alternating sequences that are at least 3 visits long is rare relative to oscillations. The values of α also remain similar independent of the minimum sequence length that we consider (mean: 1.92, 1.86; standard deviation: 0.18 and 0.20, respectively).

Since we can view an oscillation visit as a single visit to a location, this suggests that the distribution of tower visits is also consistent with a power law. In Section 3.4.1, we found that this was the case for 59.0% of the traces. This suggests that the oscillations were muddling the analysis and if we were to replace oscillation sequences with a single visit, the results might be even more significant. The same goes for the other variables that we considered in Section 3.4.

Summary: The distribution of the number of oscillation sequences per oscillation pair is very consistent with a power law. Since an oscillation sequence is effectively a tower visit, this suggests that tower visits are also distributed according to a power law.

3.5.5 Conclusion

We have seen that oscillations are common and contain valuable information—they increase the apparent resolution thereby helping to distinguish locations that would otherwise be aliased. To deal with oscillations, we can collapse oscillation sequences making tower transitions better correspond to user movement.

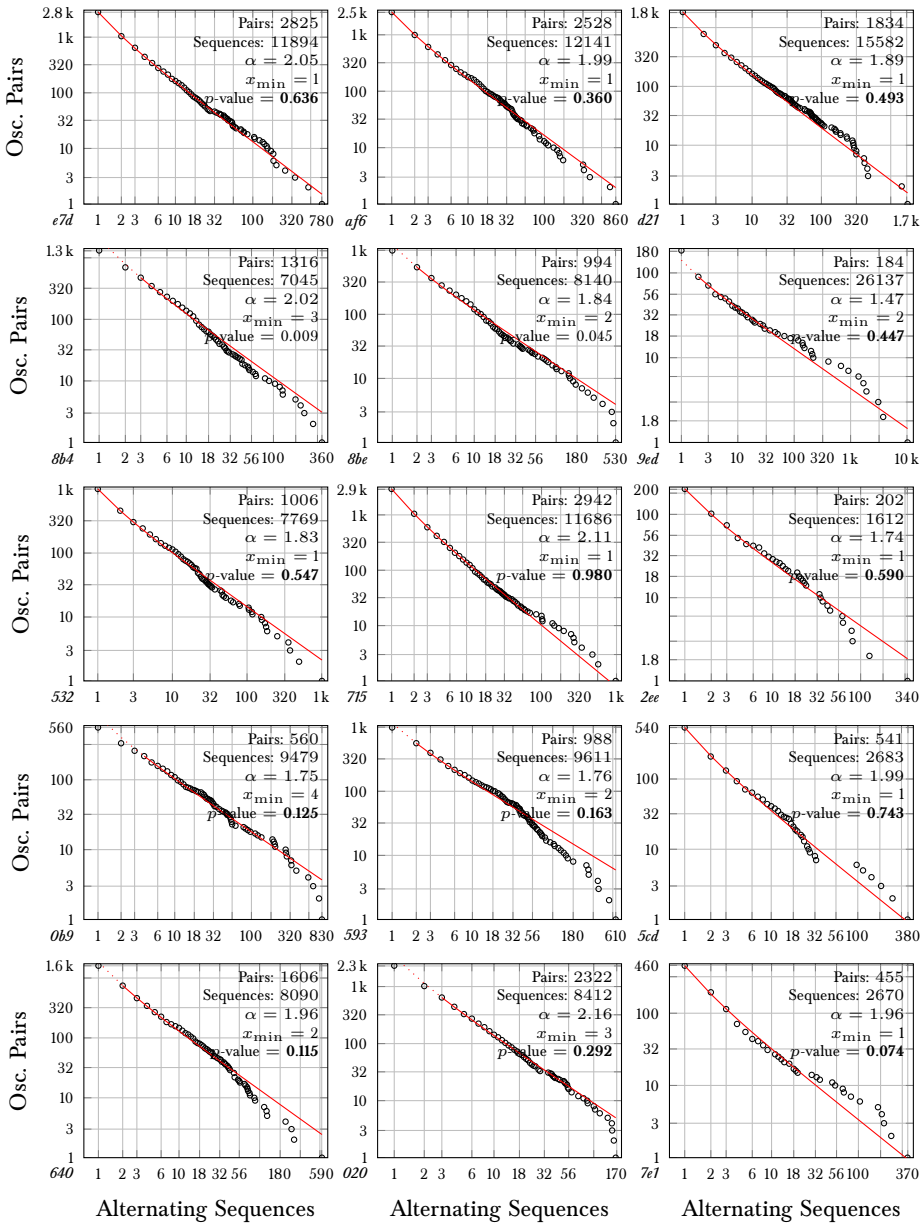


Figure 3.24: The distribution of alternating sequences by oscillation pair. The x axis is the minimum number of alternating sequences that are at least 3 visits long. The y axis is the number of oscillation pairs for which this is the case.

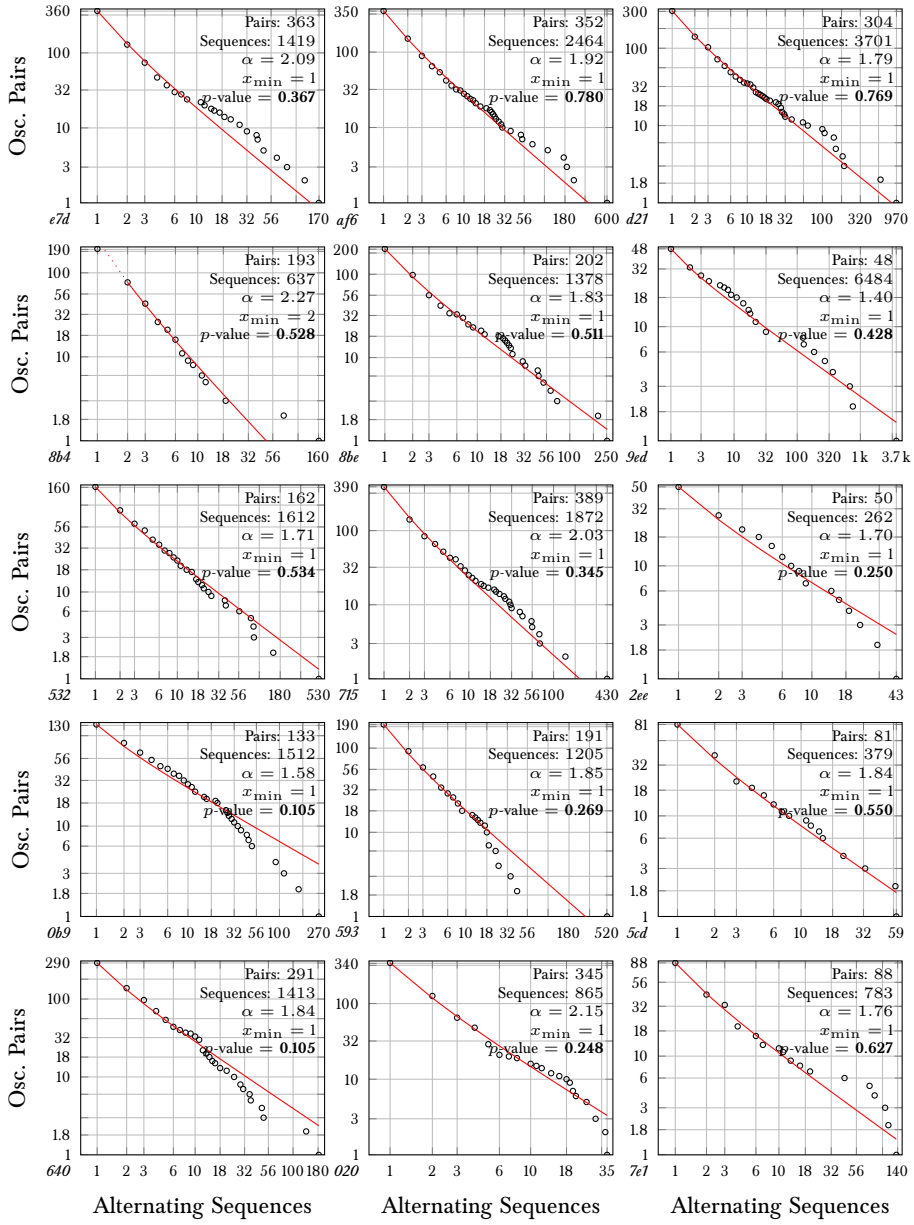


Figure 3.25: The distribution of alternating sequences by oscillation pair. The x axis is the minimum number of alternating sequences that are at least 7 visits long. The y axis is the number of oscillation pairs for which this is the case.

There are two main issues. First, we need to classify whether an alternating sequence is an oscillation sequence, in which case it should be collapsed, or whether it is due to user movement, in which case it should be left alone. An added complication is that because we want to estimate the user's location in real time, we will need to classify an alternating sequence before we've seen the whole thing and perhaps before it's even very long. Second, we need to decide how to label a collapsed sequence. If the oscillation partners are A and B, should it be labeled A, B or something else entirely? If the tower predictions are exposed to applications, it is important to quickly use stable identifiers and stick with them. Otherwise, associations saved by an application (e.g., resource x is available at location y) are useless.

3.6 Conclusions

We observed multiple times that user behavior tends to be globally regular, but locally variable. This variability appears both in time (e.g., when the user goes to work) and in space (the exact towers visited along a route or at a location appear to be a sample of the towers in the area). This firstly indicates that the best we can do is guess where a person will be in a few hours; attempting to determine where a person will be in, say, exactly 60 minutes is probably futile. For our purposes of predicting near-term connectivity and the user's behavior, this is probably sufficient. Nevertheless, infrequent activities will likely be difficult to predict based on the traces alone. One promising source of information is the user's calendar.

User behavior also varies at a coarser level. Secondary activities, such as, evening courses and club meetings, and habits, such as going to the gym, change regularly. People also switch jobs and move homes. We refer to these changes as regime changes. Major regime changes are when the user completely breaks with her normal routine such as moving or going on vacation.

To be able to quickly adapt to changes in the user's behavior, we want to aggressively age data. Old data should not be completely forgotten, however. We observed that users often return to major regimes, e.g., when visiting relatives. If we age data too aggressively, we won't have any information to work with when the user returns to these locations.

We observed that a place is rarely covered by a single cell. Instead, there are typically tens of cells in an area. This indicates that cell towers provide sufficient resolution to distinguish coarse gained location information, such as, home and work. The additional information may indicate where the user is at work (e.g., in her office, in a meeting or at the toilet). This information is further augmented

by oscillation sequences, which allow us to distinguish even more detail, since oscillations occur near the border of two cells. An implication of this is that if we use tower transition probabilities to predict the user's location, then we need to collapse the towers: the tower groups form a highly-connected component.

We observed a similar behavior along routes: the same route is rarely exactly the same as seen from the tower trace; each time a route is traversed, the set of towers visited changes a bit. We argued that the towers are a statistical sampling of the towers in the area due to highly overlapping cells. The implication is that if we compare routes then we need to use some similarity metric to compare strings of towers or to collapse the towers in some way.

When looking at the distribution of time and visits, we saw that both appear to be distributed according to a power law. This means that only a small fraction—dozens, perhaps—of the hundreds or thousands of towers that a user visits are really important locations. On average, the top 15 towers accounted for more than half of all tower visits and two-thirds of the total time. This should be exploited to weigh towers when making predictions and to potentially prune towers to avoid a state space explosion. Further, the fact that most tower visits are short—even to towers at important locations, such as, a user's home—suggests that some clustering of towers is appropriate.

While looking at tower visits, we saw that some towers are visited 100s of times per day for just a few minutes for each visit. In fact, half of all tower visits are part of alternating sequences involving probable oscillation pairs. Unfortunately, simply collapsing oscillation pairs does not appear to be a good idea: this would result in a loss of resolution since many towers appear to cover multiple locations. At the very minimum, it appears necessary to at least collapse oscillation sequences.

Intended to be blank.

Chapter 4

Oscillation Sequences

In Section 3.5, we found that cell phones often oscillate between two towers even though the user is not moving. These oscillations create a number of problems. For instance, they complicate: identifying the current location; predicting how long a user will stay at a location; and, predicting the user's location in the near future. But, oscillations are also a boon, because they allow us to distinguish more locations. In this chapter, we investigate how to identify and handle oscillation sequences.

Note: in our analysis of the trace data, we observed that a place is rarely covered by a single tower. Instead, a phone typically moves between several core towers and some minor towers at a fixed location (see, for instance, Section 3.2.3). The implication is that towers provide more resolution than is needed to identify locations such as home and work. This suggests collapsing communities of towers, which would automatically collapse pairs of towers that often end up in oscillation sequences. One disadvantage of this approach is that all areas of a place may not have the same properties (e.g., network connectivity) and sometimes it is useful to distinguish finer grained locations, such as, a meeting room, so that the phone can automatically change its profile to silent. As such, it still makes sense to first collapse oscillation sequences, which actually increases the data's resolution. Then, if more coarse-grained locations are useful, it is still possible to collapse the tower communities and segment towers along routes. In fact, this can be done in addition to collapsing oscillation sequences and the more appropriate piece of information can be used.

4.1 Introduction

An oscillation sequence is a sequence of tower visits that consists of alternating visits to two towers, but does not correspond to user movement. Oscillation sequences occur when a user stays at a location where the two strongest towers have comparable signal strengths. Because signal strength constantly fluctuates, once or twice a minute the other tower's signal is significantly stronger and the cell phone switches to it. Such locations are frequent and geographically stable: they define the borders between cell towers.

Oscillations would rarely occur if a signal's strength were just a function of the receiver's distance from the sender. Propagation effects such as the scattering, diffraction and reflection of radio waves, however, cause the received signal's strength to vary constantly. For instance, Schwartz observes that if a receiver moves just half of a wave length—21.4 cm for 700 MHz radio waves and 7.5 cm for 2 GHz radio waves—the signal "may vary many dB" due to multipath fading, which is "the destructive/constructive phase interference of many received signal paths" [87, Section 2.2]. If a user's cell phone is in her pocket, she needn't stand up to move this far: she just needs to adjust the way she is sitting, swivel her chair or stretch a bit. Of course a user's environment is not static: if a street is nearby, for instance, moving cars can influence the signal.

Not all alternating sequences are oscillation sequences. It is not difficult to imagine scenarios in which a person actually moves between two towers multiple times. A store's parking lot, for instance, may be covered by one tower (P) and the store itself by another (S). When a person goes shopping at this store, he parks, goes inside and then returns to his car. This results in an alternating sequence of length three: $P \rightarrow S \rightarrow P$. If he then realizes that he forgot something while loading the groceries into his car, he might go back into the store. This would result in an alternating sequence of length five: $P \rightarrow S \rightarrow P \rightarrow S \rightarrow P$.

Even longer alternating sequences corresponding to user movement are conceivable. For instance, if a person loads or unloads a moving van then he'll go back and forth many times. If the two locations are covered by different cell towers that are next to each other, then a long alternating sequence will result. For most users, these long alternating sequences are unlikely to correspond to user movement and can be safely classified as oscillation sequences. Short sequences, however, remain a problem, because there are no obvious features that distinguish user movement from oscillations. Unfortunately, as we will shortly see, the length of alternating sequences is distributed according to a right heavy tailed distribution meaning that most oscillation sequences are short.

4.2 Issues

Oscillations obscure a user's location and movement and make predicting the user's trajectory more difficult. These problems can be overcome by treating oscillation sequences as visits to a single location.

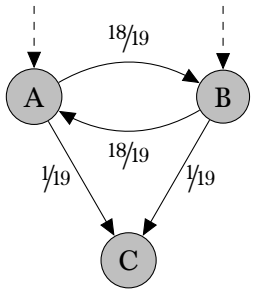
The fact that oscillations obscure a user's location and movement is straightforward to see. Since we use cell towers as a proxy for a user's location and cell tower transitions as a proxy for movement, when a user oscillates between two towers, it appears as though the user is moving between two locations.

A decreased correspondence between the modeled location and the actual location negatively impacts anything that depends on the user's location. Consider associating network conditions with location. If we use the wrong location when predicting the current network conditions, our guess will be poor. This prediction could cause us to waste resources or miss an opportunity. Even worse, however, is if we determine the current network conditions and associate them with the wrong location. In this case, when we make a prediction while at that location, our prediction will be based on incorrect historical conditions.

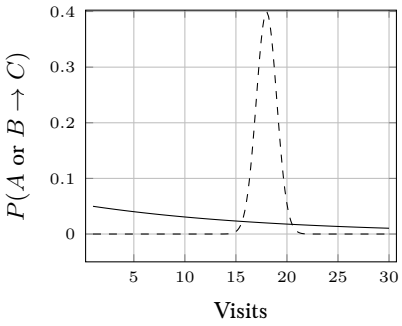
To understand why oscillation sequences make predicting a user's trajectory more difficult, consider the following scenario. When a user connects to tower A or B, he oscillates between them on average 9 times with a standard deviation of 0.5 after which he then transitions to tower C. This results in an oscillation sequence consisting of 19 tower visits (9 oscillations), on average. The induced network with the empirical transition probabilities is shown in Figure 4.1a.

A simple prediction algorithm uses the current tower's empirical transition frequency to predict the subsequent tower. This algorithm would predict B as A's most likely successor and A as B's most likely successor with probability $18/19$. Since we only ever consider the current tower, the prediction is independent of how far into an oscillation sequence we are. That is, even if we are at A after 9 oscillations, we will still predict B as the visit's the successor with probability $18/19$. By analogy, consider flipping a coin: if we have a fair coin and we flip 10 heads in a row, the probability of getting a head on the next flip is 0.5 even though a sequence of 11 heads is highly unlikely. Put differently, the models assume the Markov property.

Consider what happens if we try to predict whether we will transition to C after x steps at the start of an $A \leftrightarrow B$ oscillation sequence. Our simple algorithm predicts that the probability of immediately transitioning to C is $1/19$, the probability of transitioning to C after one step is $18/19 \cdot 1/19$, etc. This is the geometric distribution and is shown as the solid line in Figure 4.1b. The actual distribution, as described above, is the dashed line. Our model is clearly a poor match.



(a) An induced network. The nodes are towers and the numbers are the transition probabilities.



(b) The probability of transitioning to C after x steps of an alternating sequence of A and B. The solid line corresponds to a prediction algorithm that uses just the currently connected tower to predict the subsequent tower; the dashed line is the actual probability.

Figure 4.1: A simple induced tower network with a pair of oscillating towers, A and B. The numbers correspond to transition probabilities. Although C is the actual successor in terms of location, it's transition frequency in terms of tower transitions is relatively small. If we consider recent history when making the prediction, however, we can accurately predict that the user will transition to C after approximately 9 oscillations.

One way to fix this is to consider more history. We could, for instance, consider all possible 20 visit sequences and choose the most likely successor. Unfortunately, this increases the size of the state space exponentially. Assuming the probabilities are stored using a naïve matrix representation, this raises the size from N^{1+1} to N^{20+1} where N is the number of towers the user has visited, i.e., the number of nodes in the network, and the exponent is the amount of history plus one for the result. Since the state space is so large, the collected data will be distributed very thinly and the states that have any examples will likely have just 1. Although this captures more history, it results in very poor estimations of the actual probabilities. This is the curse of dimensionality [4,76], which Wasserman illustrates as follows: $842\,000$ examples in a 10-dimensional problem is similar to having just 4 examples in a 1-dimensional problem [4].

We need to be smarter. In the case of oscillations, we don't actually care about the length of the sequences; we are interested in the oscillation sequences' dwell times and their successors. If we simply collapse oscillation sequences

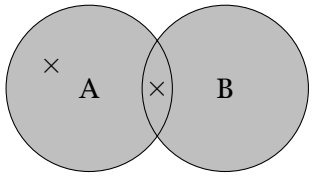


Figure 4.2: A single tower can cover multiple locations. In this case, tower A covers two locations. The first location is entirely within tower A's area and the second is on the border of tower A's and tower B's areas. In the latter case, we can expect to see oscillations. The oscillations allow us to easily distinguish the two locations: if they were both in tower A's area, this would be much harder.

into a single visit to a sort of super tower and use our original naïve prediction method, our predictions for both dwell time and the subsequent tower will be as good as those for normal tower visits. In the above example, we would have just a single transition direction to consider: $A/B \rightarrow C$.

4.3 A Boon: Increased Geographical Resolution

Collapsing oscillation sequences cleans up the trace by throwing away misleading information. But, oscillation sequences are not only a distraction. They are also a boon: oscillation sequences allow us to distinguish more locations.

Cell towers can cover a fair amount of area (recall Figure 3.7 on page 61). As a result, a single cell tower may cover multiple significant locations. If we associate a location with each tower, then we cannot distinguish two locations covered by the same tower. If one location is on the cell tower's border, however, then visits to that location will show up as oscillation sequences with the neighboring tower. This idea is shown in Figure 4.2. In the last section, we noted that collapsing oscillation sequences allows us to recognize such a location as a single location instead of two apparent locations that the user moves between. We can do even better, however: the oscillations allow us to distinguish the location on the border from the location that is entirely within A.

This observation answers the tacit question of how to label collapsed oscillation sequences. Since oscillation sequences occur at different locations from where they don't occur, they should get different labels. Thus, in addition to each tower having its own location label, each oscillation pair should be assigned a unique label.

A difficulty arises if the user moves from A/B to A or vice verse. If we just use the presence of an alternating sequence as an indicator of an oscillation

sequence, then we will not realize that the user has moved. That is, $A \to B \to A \to B \to A \to B \to A$ might be sufficiently long that we conclude that we have strong evidence of an oscillation sequence, however, the user may have moved to the location entirely in A in the middle of the sequence and then returned to the oscillation location. We refer to such sequences as mixed mode alternating sequences.

To deal with these types of sequences, the classifier needs to consider more than just an alternating sequence's length. A possibly helpful feature is a visit's dwell time. If oscillation visit dwell times are distributed according to a different distribution from non-oscillation visits, then we can distinguish movements from A/B to A and vice versa.

It may, however, often be safe to simply treat mixed mode alternating sequences as oscillation sequences. If the user's living areas are on A and B's borders, for instance, but his bedroom is entirely within A, it is probably not necessary to distinguish these locations.

4.4 Solution Space

As discussed at the start of this chapter, not all alternating sequences are oscillation sequences: some alternating sequences correspond to real user movement. We don't want to collapse these. What we need is a way to distinguish alternating sequences that arise from user movement from those that arise due to oscillations. Further, our approach should correctly deal with mixed mode alternating sequences, which we mentioned in the last section.

In practice, we could conclusively determine whether an alternating sequence corresponds to a user movement sequence or an oscillation sequence if we knew the user's actual trajectory. We could get this information using GPS: if the user remains at the same geographic position, then any alternating sequence is almost certainly an oscillation sequence; if not, then the alternating sequence is most likely a user movement sequence. However, the reason that we are using cell towers as a proxy for location is that other mechanisms for determining the user's location, in particular, GPS, are too expensive. Moreover, using GPS is not completely reliable either: there are many places where obtaining a GPS signal is difficult, in particular, indoors.

Fortuitously, the tower traces may already contain enough information to distinguish oscillation sequences from user movement sequences. Features such as sequence length, sequence dwell time and tower visit dwell time may often be sufficient discriminators. If not, it is still possible to use GPS to augment the cell tower traces when we don't have enough information. If this is sufficiently

Constraint	Description
Real Time	The classifier can only use data collected prior to the event; it does not do an a posteriori analysis.
Low Latency	The classifier must not wait too long after the start of an alternating sequence to classify it.
Consistency	Locations should be labeled consistently, even if this slightly increases the error rate.

Table 4.1: Summary of constraints on the solution space.

infrequent, the cost will probably be acceptable. As we get more information about individual alternating pairs, falling back to GPS will hopefully become increasingly unnecessary. Since our traces don't include GPS tracks, we don't explore this option further here, but leave it as future work.

Our goal then is simply stated: determine whether a tower visit arose from tower oscillations or user movement. The solution space, however, is broad and there are a number of issues that we need to consider.

4.4.1 Constraints

We are interested in classifying tower visits on the user's phone in real time. This setting is more constrained than, say, an a posteriori analysis, which, in addition to being able to use all of the available data, can use more computing power and does not need to worry about energy use (and thus can use more complicated models).

The first practical constraint that this setting imposes is that the classifier must work in real time and not after all of the data have been collected. That is, the first time the device alternates between a pair of towers, the algorithm needs to make a classification decision even though its history includes no information about the towers.

The second constraint is that classifications need to be prompt. This means that after the device starts alternating between two towers, the classifier shouldn't wait a long time before making a decision. If the classifier only makes a decision after observing, say, at least half an hour of data, the user may have already moved on and a data transfer opportunity may have been missed. Waiting a few minutes, however, is probably reasonable: when a user is moving and the device switches to a new cell that is associated with a fixed location, the user is

probably not yet at the fixed location covered by the cell and thus Wi-Fi may not yet be available.

Finally, alternating sequences should be classified consistently. That is, the classifier should try to avoid sometimes labeling oscillation sequences from a given tower pair as oscillation sequences and sometimes as user movement sequences. This can happen if a relative threshold is used (e.g., 10% of all visits are in alternating sequences that are at least 7 visits long). This constraint is important, because this data is exposed to applications. When we expose the current location to the upper software layers, two things happen. First, the software uses the information to determine what actions to take. If the reported location is incorrect, then the software will make poor decisions. Second, and perhaps more importantly, the upper layers build associations with the location to determine what actions to take in the future. Thus, if we report the wrong location, the current connectivity information will be associated with the wrong location, which negatively impacts future prediction about available connectivity, for instance. It is possible to include an interface to relabel past classifications, but this would almost certainly place too much of a burden on application developers.

4.4.2 Misclassifications

Ideally, we want to avoid misclassifications. Completely avoiding misclassifications is unlikely to happen in practice and we shouldn't plan for it. By acknowledging that mistakes will occur, we can try to design our system to minimize their impact. In particular, we can take advantage of that fact that different mistakes often have different consequences. In our case, there are two possible mistakes: a user movement sequence could be labeled as an oscillation sequence and an oscillation sequence could be labeled as a user movement sequence.

When we label a user movement sequence as an oscillation sequence, we are most likely mislabelling a short alternating sequence. Since the user is moving, the sequence is probably short not only in terms of the number of visits, but also in terms of the sequence's total time. Hence, for this mistake, the amount of time that we are wrong is typically relatively small. A user movement sequence also represents a different mode of operation from simply staying at a fixed location. That is, the available resources and the expected behaviors while moving are probably different from those when the user is at the fixed location. As such, giving it a different label is not unreasonable. In the worst case, we effectively create an alias for the location. This spreads the data out a bit more and it takes longer for the algorithm to learn about the location, but in the long term a limited amount of aliasing should have little negative impact. Based on these two observations, we conclude that mislabelling user movement sequences as os-

cillation sequences is unfortunate, but will have minimal negative consequences in practice.

When we label an oscillation sequence as a user movement sequence, we end up exposing a potentially large number of tower transitions instead of a single visit to an oscillation location. As discussed in Section 4.2, long oscillation sequences make it more difficult to predict how long the user will stay at the actual location and what the subsequent tower will be. As such, mislabelling oscillation sequences as user movement sequences is markedly worse than mislabelling user movement sequences as oscillation sequences and should be avoided.

Another point to consider when deciding how to err is the prior probability of each type of event. In Section 3.5.2 we observed that approximately 45.0% of all tower visits are part of oscillation sequences. Thus, the prior probability that an alternating sequence is an oscillation sequence is much higher than the prior probability that an alternating sequence is a user movement sequence. As such, our misclassification rate will probably be significantly lower if we classify alternating sequences that we are uncertain about as oscillation sequences rather than user movement sequences.

Based on these observations, the best strategy is to prefer to mislabel user movement sequences than to mislabel oscillation sequences.

4.5 Training Data

Our classification problem is to identify whether a tower visit is part of an oscillation sequence or not. For this, we need examples of tower visits. There are two basic approaches: supervised learning, in which we have labeled examples and attempt to directly separate the classes; and, unsupervised learning, in which the examples are unlabled and we attempt to discover structure, i.e., groupings. Supervised learning is more powerful, but our data is unlabeled. With a bit of work, however, we are able to come up with reasonable labels for a set of arguably unbiased examples.

4.5.1 Supervised vs. Unsupervised Learning

There are two main types of classification algorithms: those based on supervised learning and those based on unsupervised learning. Both supervised and unsupervised learning techniques take a number of examples as input. These examples are used to compute decision boundaries, which (hopefully) cleanly separate the different classes. For example, if we want to use sequence length to classify alternating sequences, then we would provide examples of sequence

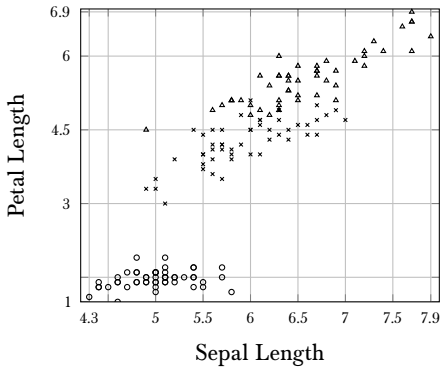


Figure 4.3: Plot of the sepal length of three species of Irises vs. their petal length. The data is from Anderson's famous study [31]. An unsupervised learning algorithm could easily identify two major groupings, but would be unable to separate the top right oval since the two groups bleed into each other. A supervised learning algorithm, however, could identify a reasonable decision boundary for the top right group, since it knows each example's class.

lengths to the training algorithm. The difference between supervised and unsupervised learning is that supervised learning algorithms are also provided with the examples' labels (i.e., their true class). Continuing with our example, this means that we would also indicate whether the sequence was an oscillation sequence or a user movement sequence. This makes supervised learning algorithms significantly more powerful than unsupervised learning algorithms.

To understand the difference in power between supervised and unsupervised learning algorithms, consider Figure 4.3, which shows a plot of the sepal length vs. the petal length of three different types of Irises. The data come from Anderson's famous study [31]. The different types of Irises are shown using different markers—circles, exes and triangles. A supervised learning algorithm is given these designations along with each example's features (in this case, the sepal length and the petal length); an unsupervised learning algorithm would be given just the examples' features. The practical result is that although an unsupervised learning algorithm is able to recognize that there are two groupings—the oval at the bottom left and the elongated oval higher up—it is unable to separate the examples in the upper right oval even though there is minimal overlap between the two classes. Indeed, it is unable to even recognize that that the bottom left oval consists of examples from a single class and that the upper oval consists of examples from two different classes. A supervised learning algorithm can, however, find a relatively clean separation, since there is little overlap between

the classes.

4.5.2 Finding Labeled Examples

Given that supervised learning can come up with better decision boundaries than unsupervised learning, we would prefer to use supervised learning techniques. Unfortunately, the data that we collected is unlabeled. This is because at the start of the study we didn't realize that oscillation sequences would be a problem and, as such, didn't take any measures to determine their true labels. As already mentioned, we could have obtained ground truth using GPS. Of course, for our data set, it is too late to get this information.

Happily, we are able to easily label some tower visits with relatively high confidence. First, nearly all very long alternating sequences consist primarily of oscillation visits. Second, there are a number of towers with many visits that rarely end up in even medium length alternating sequences. Visits to these towers are probably normal visits and any alternating sequences probably correspond to real user movement.

Oscillation Tower Visits

Since oscillation visits tend to be part of oscillation sequences, we can instead focus on identifying some representative oscillation sequences. As we will argue in detail when looking at the use of sequence length to classify alternating sequences in Section 4.6.2, alternating sequences that are at least 10 visits long are almost certainly oscillation sequences—people rarely move between two locations so many times.

This criterion is not perfect: not all tower visits in a long alternating sequence are necessarily oscillation visits. Recall the mixed mode alternating sequences that we looked at in Section 4.3: if the user is at location A/B for a while and then moves to location A, then there is no tower transition to indicate the movement. Mixed mode alternating sequences are probably infrequent relative to oscillation sequences and only present for some tower pairs, namely, those that cover multiple locations. Thus, this issue is probably minor, but it is a potential bias to keep in mind.

A simple second test that we can use to classify oscillation sequence is whether a pair of towers has many long alternating sequences. Because we expect cellular networks to change slowly, we expect oscillation locations to remain oscillation locations and non-oscillation locations to remain non-oscillation locations. Consequently, if a user visits an oscillation location for long enough, we will probably observe an oscillation sequence. Further, as already observed, long

user movement sequences are rare, but many long user movement sequences involving the same two locations are probably very rare.

Using these criteria to label oscillation visits results in a biased sample. In particular, sequence length is not representative: there are most certainly short oscillation sequences as well as long oscillation sequences. Sequence length is effectively left truncated. Oscillation visit dwell time, however, is likely to be representative: visit dwell time is independent of sequence length. Whereas sequence length is related to how long the user stays at a location, dwell time at an oscillation tower depends on the underlying oscillation process, which is a function of the network's configuration and the user's physical location and has nothing to do with how long the user is at the oscillation location. However, as already mentioned, oscillation visit dwell time will be slightly biased by mixed mode alternating sequences.

Given these observations, we conservatively label alternating sequences that are at least 10 visits long and come from an alternating pair with at least 20 such alternating sequences to be oscillation sequences. (We refer to these tower pairs as significant oscillation pairs.) Note: we don't select all alternating sequences that these towers are involved with, but only those that are at least 10 visits long. Some of the shorter sequences are likely to be user movement sequences.

Only considering pairs with a minimum number of long alternating sequences has a helpful secondary effect: any statistics that we compute on a per-pair or per-tower basis are likely to have a sufficiently large sample size that we avoid overfitting the data, which would lead to incorrect conclusions.

Selecting oscillation sequences that are at least 10 visits long and come from tower pairs with at least 20 such sequences across all of the traces provides 444 990 examples of oscillation visits (19% of the total visits) from 18 788 oscillation sequences. These sequences come from 189 different oscillation pairs. The median number of tower visits for each pair (considering just the long sequences) is 1031 with a median absolute deviation (MAD) of 850. The median number of long sequences per pair is 49 with a MAD of 36. The median number of tower pairs per user is 3.

Normal Tower Visits

Unfortunately, it is not as easy to identify a representative sample of normal tower visits as it is to identify a representative sample of oscillation tower visits. We can't just select tower visits that aren't involved in any alternating sequences: short visits to oscillation locations likely also result in singleton visits. We also can't just select towers that are never involved in an alternating sequence: this would exclude nearly all significant towers; most towers end up in short alter-

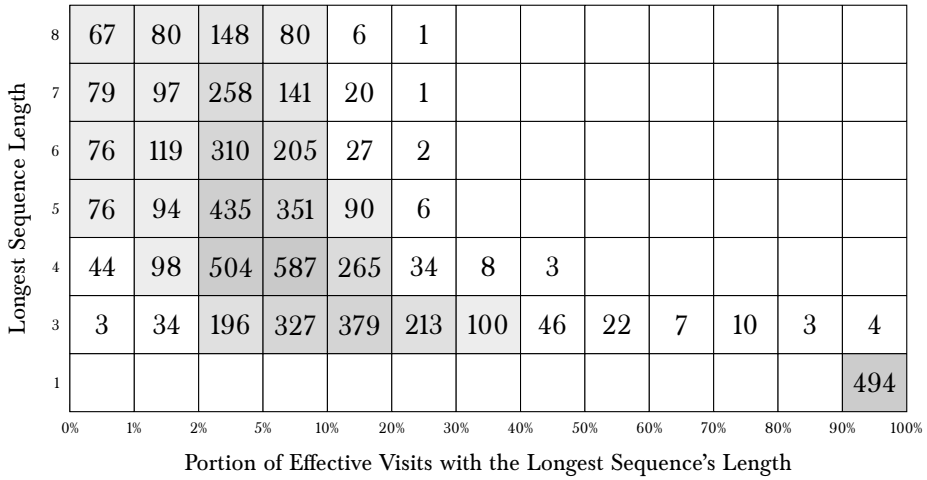


Figure 4.4: Heat map of towers with at least 15 effective visits broken down by longest alternating sequence and portion of effective visits involved in such sequences. Note: the x axis is non-linear. Of the relevant 9641 towers, 3491 have at least one sequence that is longer than 8 visits.

nating sequence at some point. Indeed, nearly all significant towers end up in a few long alternating sequences.

Figure 4.4 shows a heat map of towers with at least 15 effective visits broken down by the length of the longest alternating sequence that they are involved in, and the portion of effective visits with that length. An effective visit is the approximate number of visits to a location as opposed to visits to a tower; effective visits count each alternating sequence as a single visit. This way of counting visits makes sense for oscillation sequences, since the user isn't really moving back and forth, but staying at a single location, it undercounts visits that are part of user movement sequences for the same reason. Since alternating sequences arising from user movement are intuitively the exception rather than the rule, and an oscillation location will normally result in oscillation sequences (although they may be very short for short visits), this trade off seems reasonable.

Looking at the heat map, we see that there are 494 towers that are visited at least 15 times and are never involved in an alternating sequence (i.e., the maximum sequence length is less than 3). This isn't that many given that there are 9641 towers that are visited at least 15 times. There are 1344 towers whose maximum alternating sequence is 3 visits long. For most of these towers, the alternating sequences are infrequent. But, for a handful, alternating sequences that are 3 visits long are the rule. This isn't unreasonable: many places that

people go are often left the same way that they are entered, just in reverse. Most stores, for instance, have a primary way to enter and leave. If the entrance is on a tower boundary, then such alternating sequences will be usual for that location. Also, microcells, which cover at most a single building, tend to have just one or two neighboring towers. Given this, it seems reasonable to classify these alternating sequences as user movement sequences. Similarly, it seems gratuitous to not count a tower as a non-oscillation tower just because it has a few slightly longer alternating sequences. We *do* expect to see occasional longer user movement sequences at locations that are important to the user. For instance, when doing the weekly grocery shopping, it would not be surprising if the person walks between the house and the car a few times to unload everything.

Given these observations, we conservatively classify towers as non-oscillation towers if: they consist of at least 15 effective visits; they have at most one alternating sequence whose length exceeds 10 visits; the alternating sequences that are at least 7 visits long do not exceed 1% of the total number of effective visits; those that are at least 6 visits long do not exceed 3% of the total number of effective visits; and, those that are at least 5 visits long do not exceed 9% of the total number of effective visits.

We don't expect this rule to classify many oscillation towers as non-oscillation towers. Because we expect an oscillation sequence to develop most every time the user visits an oscillation location for a non-trivial amount of time, for a false positive to occur, most visits to the oscillation location would have to be singletons or be part of alternating sequences that are at most three visits long. For this to happen, the typical oscillation period must be long compared to the amount of time that the user normally spends at the location. However, as we will discuss shortly, the typical oscillation visit is about a minute long. Thus, this is unlikely.

This rule will exclude many non-oscillation tower visits. First, we believe that the rule is conservative. For instance, if a tower is involved in just two alternating sequences that are 7 visits long, then there must be at least 198 other shorter effective visits for the tower to be classified as a non-oscillation tower. This is probably too conservative, however, this is by design: the rule should minimize the number of false positives. Second, recall from Section 4.3 that a single tower sometimes corresponds to multiple locations. If one of these is an oscillation location, then we will observe that the tower is involved in oscillations and completely exclude that tower.

This rule classifies 4825 of the towers as non-oscillation towers. By comparison, there are 299 towers involved in the significant oscillation pairs (some towers are part of multiple oscillation pairs). This leaves 47% of the 9641 top towers unclassified. The non-oscillation towers consist of 304 533 (13% of the

total visits vs. 19% for the oscillation visits). The median number of visits per non-oscillation tower is 30 and the MAD is 16. In terms of long sequences, 211 of the 4825 towers have sequences that are 7 visits long or longer. 50 of these have more than one such sequence.

Transitory Towers

Oscillation pairs only ever correspond to locations. If an oscillation location corresponded to a transitory location, then the user would rarely stay there long enough for a long oscillation sequence to develop and the tower would not be classified as an oscillation tower. Indeed, for our purposes, it isn't an oscillation tower. To facilitate the direct comparison of oscillation and non-oscillation towers, we further subdivide non-oscillation towers into transitory and location towers.

A transitory tower is a tower that is primarily used along a route, such as a commute to work. Thus, we expect the typical dwell time to be rather short. The dwell time will be a function of how large the cell is, how much of the cell the user traverses, and the speed of the traversal. Thus, we can expect the user to stay connected to a cell for just tens of seconds when driving in a city unless there is a lot of traffic. The traversal could also be much longer if the user is walking.

Since transitory towers make up a route, we expect that a user's trace will consist of many more transitory towers than location towers. The user's work location may be covered by a handful of towers, but unless the user works from home, her commute will likely cover a few dozen towers, if not more. This observation is consistent with Figure 3.3, which showed the towers visited over the course of a day of several different users.

Figure 4.5 shows a histogram of the 98.0% dwell time quantile on our training set of normal tower visits. On the far left, we see that for a quarter of these towers, nearly all visit dwell times are less than 30 seconds. These are probably transitory towers. As we extend the threshold, we can continue to make the same argument: if nearly all visits to a location are less than 1 minute long, then the user probably only traverses the location and doesn't stay there. The same is true if the 98.0% quantile is 2 minutes or 4 minutes. At around a quarter of an hour, the argument begins to break down. It is conceivable that the user could drop his child off a school in the morning and pick her up in the afternoon in this amount of time, for instance.

Given the lack of deviations in the curve implied by the data, it is difficult to choose a particular cutoff point. It seems reasonable, however, to classify any significant tower for which at least 98.0% of its visits are less than 10 minutes

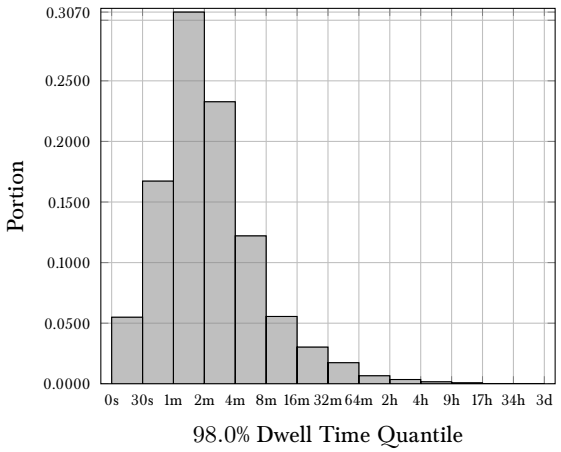


Figure 4.5: Histogram of the 98.0% dwell time quantile of the significant non-oscillation towers. Note: the x axis is logarithmic.

long and no visit exceeds an hour as a transitory tower.

Excluding these transitory towers from the non-oscillation towers leaves us with 464 towers or just 10% of the significant non-oscillation towers. This amount of pruning is reasonable given that we expect to see many more transitory towers than location towers. These towers consist of just 34 541 visits (1.5% of the total visits vs. 19% for oscillation visits). The median number of visits per tower is 36; the MAD is 22.

Figure 4.6 shows a comparison of the oscillation visits, normal visits and transitory visits.

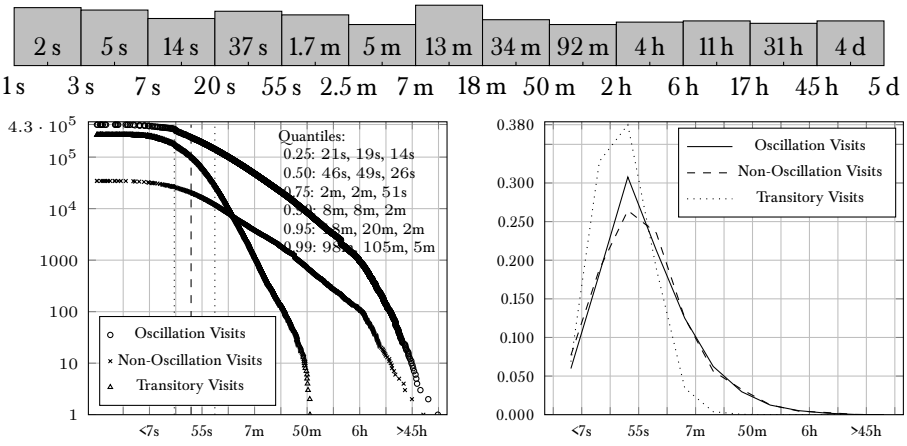
4.6 Features

We now consider various features that could be used to classify tower visits as part of an oscillation or arising from user movement. In particular, we look at tower visit dwell time, a sequence's length and prior alternating sequences.

4.6.1 Visit Dwell Time

We start by looking at tower visit dwell time, i.e., the amount of time the device is connected to a tower. Intuitively, we expect dwell times at oscillation locations to be short and dwell times at non-oscillation locations to be long.

Unlike sequence length, visit dwell time is a property of a tower visit and not an alternating sequence. This is particularly useful for breaking apart mixed



(a) Complementary cumulative Pareto plot.

(b) Histogram.

Figure 4.6: Plots comparing the distribution of oscillation tower visits, non-oscillation tower visits, and transitory tower visits. The distribution of the oscillation visits' dwell times and the non-oscillation dwell times are very similar. The drop at 10 minutes for transitory visits is due to how we selected towers: we didn't choose any towers if more than 2% of the entries exceeded 10 min.

mode sequences, i.e., alternating sequences in which the user spends part of his time at an oscillation location and part of his time at one of the towers (see Section 4.3); and, for fast identification of oscillation sequences, which is desirable when classifying the sequences in real time, because the sooner we can confidently identify the user's location, the sooner we can act.

Oscillation Tower Visits

We first examine tower visit dwell time broken down by oscillation pair and tower. We consider individual towers rather than grouping the dwell times from an oscillation pair together, because the distribution of dwell times is likely different for each tower: dwell time likely depends on the towers' relative signal strengths and the stability of their signals.

Figure 4.7 and Figure 4.8 show the dwell times of some of the top oscillation pairs. Note: We exclude the dwell time of the last tower visit in each sequence, because these dwell times are truncated: the transition away from the tower was due to user movement and not due to an oscillation. The first set of plots shows the data as a series of complementary cumulative Pareto plots; the second

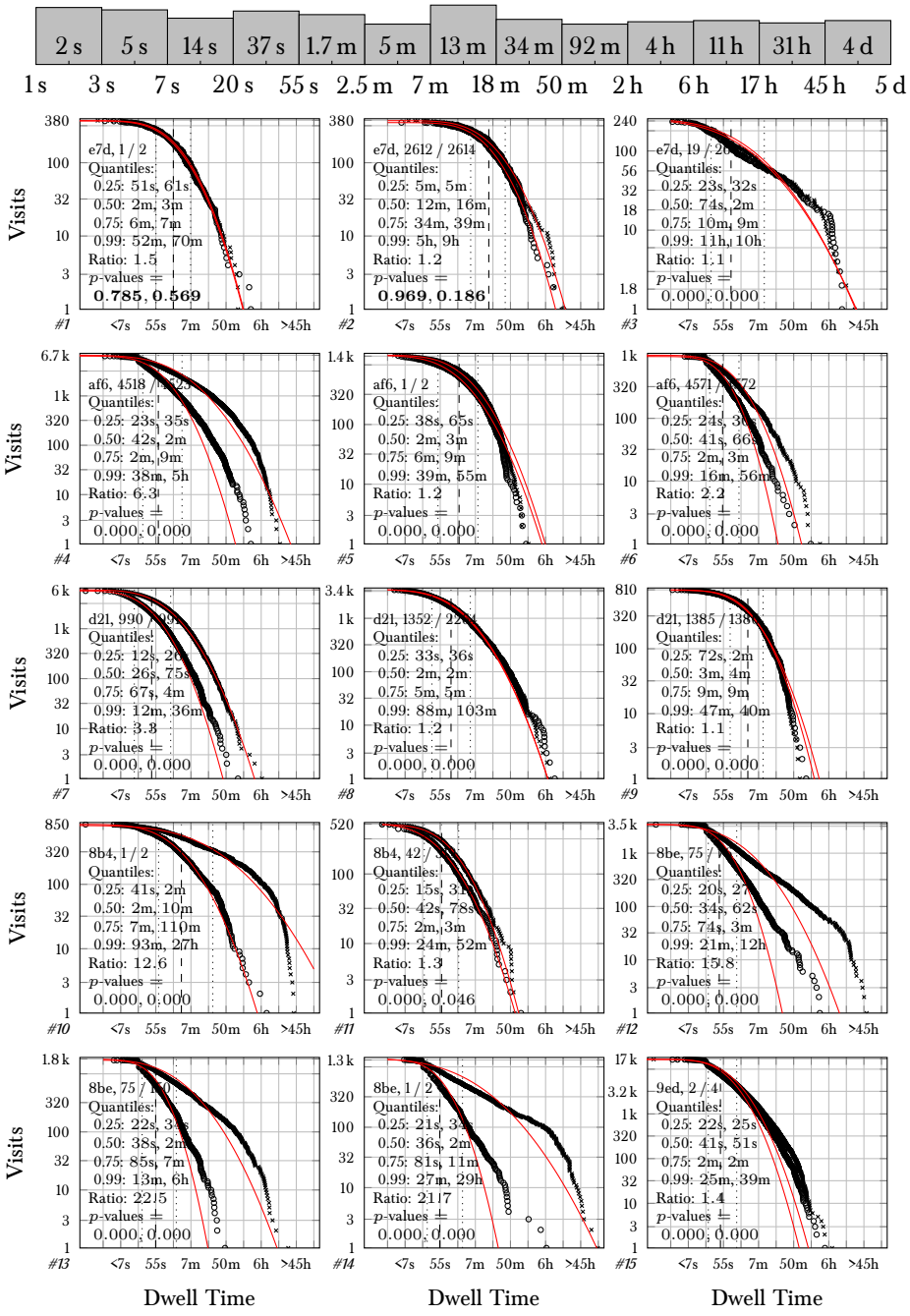


Figure 4.7: Complementary cumulative Pareto plots of the dwell time of tower visits involved in oscillation sequences. Each plot corresponds to an oscillation pair. The data is fit to a log-normal distribution. Just 27 of the 378 regressions (0.07) are significant fits at the $p = 0.05$ level.

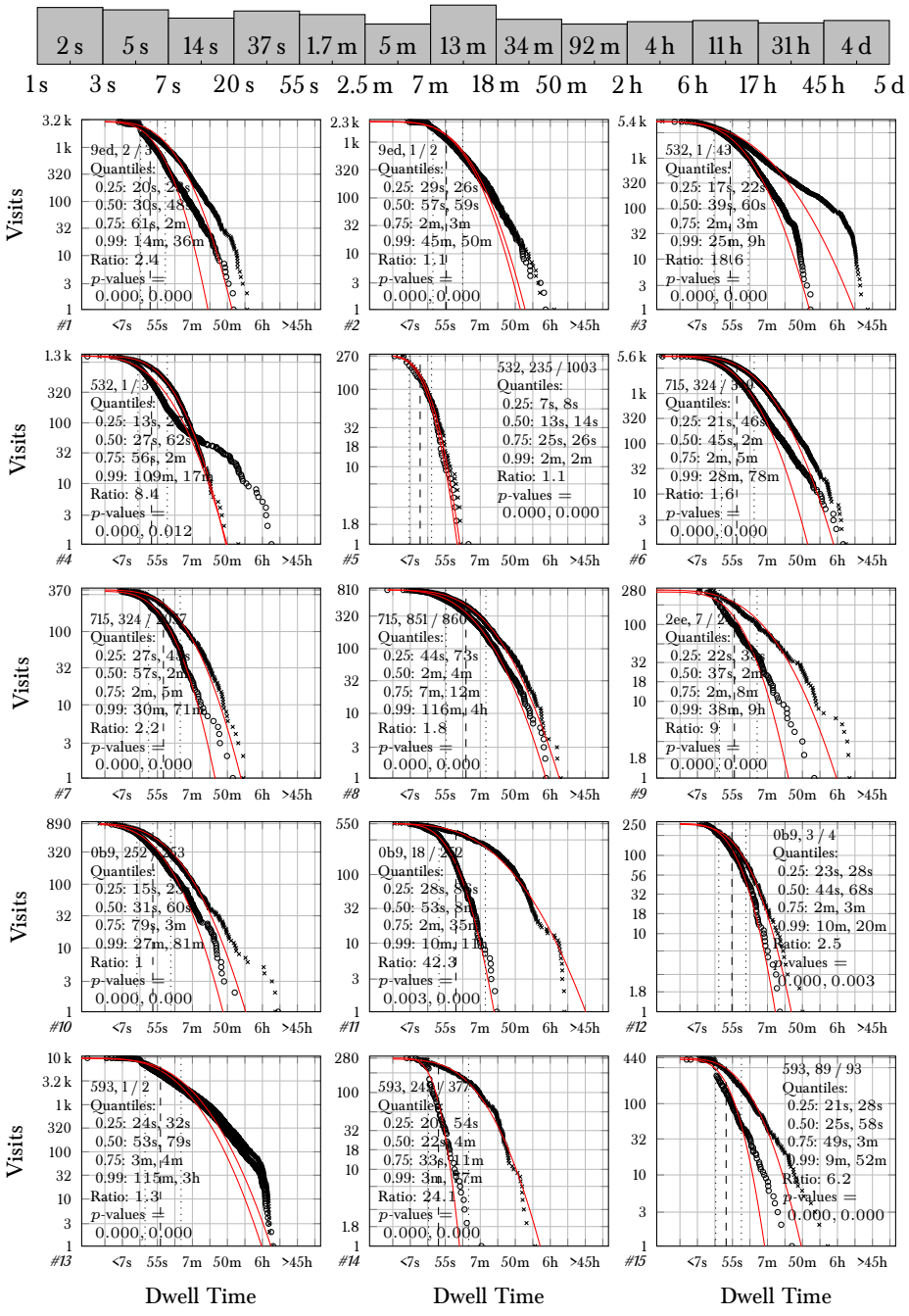


Figure 4.7 (Continued): Complementary cumulative Pareto plots of the dwell time of tower visits involved in oscillation sequences. Each plot corresponds to an oscillation pair. The data is fit to a log-normal distribution. Just 27 of the 378 regressions (0.07) are significant fits at the $p = 0.05$ level.

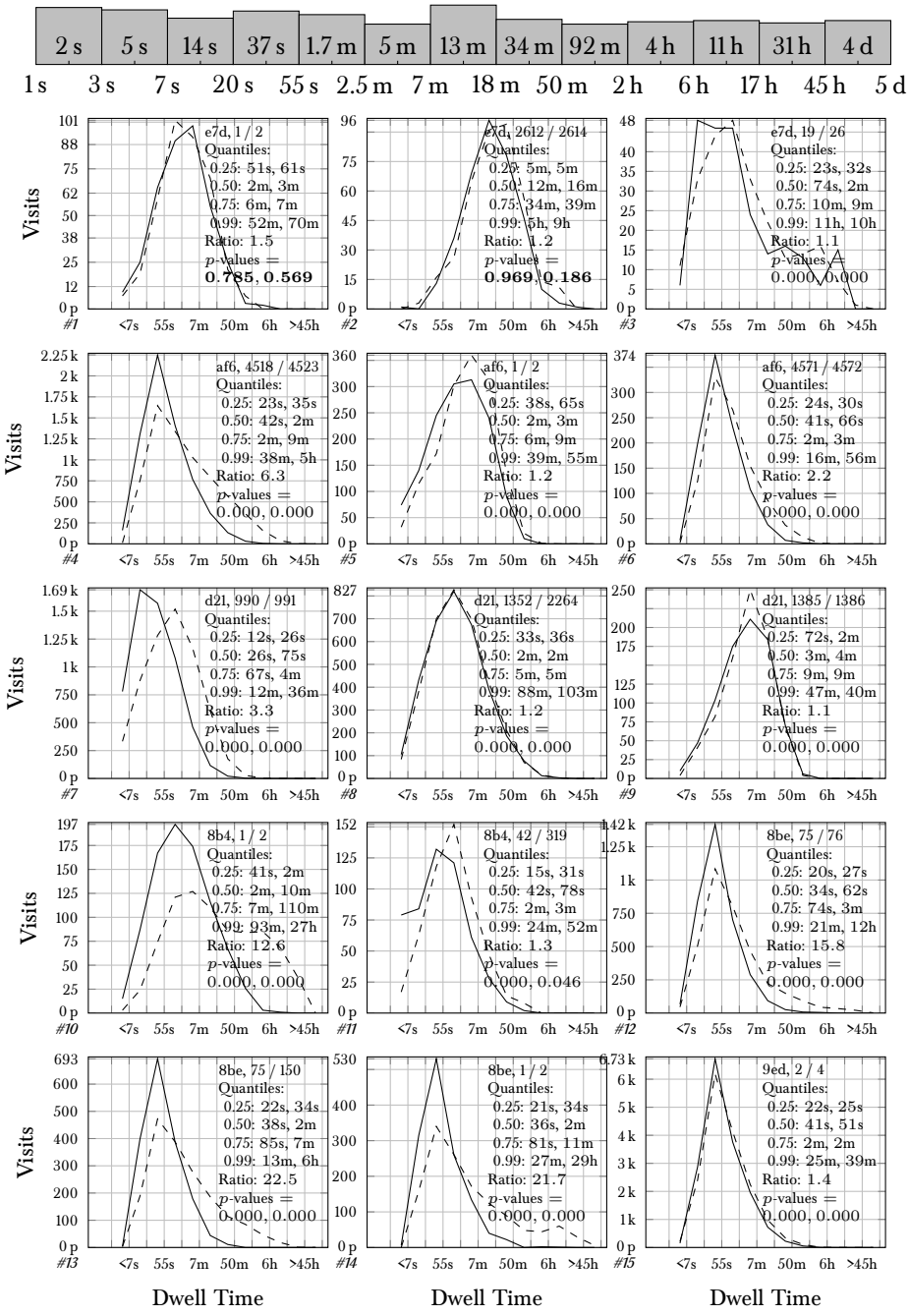


Figure 4.8: Histograms of the dwell time of tower visits involved in oscillation sequences. The histograms provide a different view of the data presented in Figure 4.7.

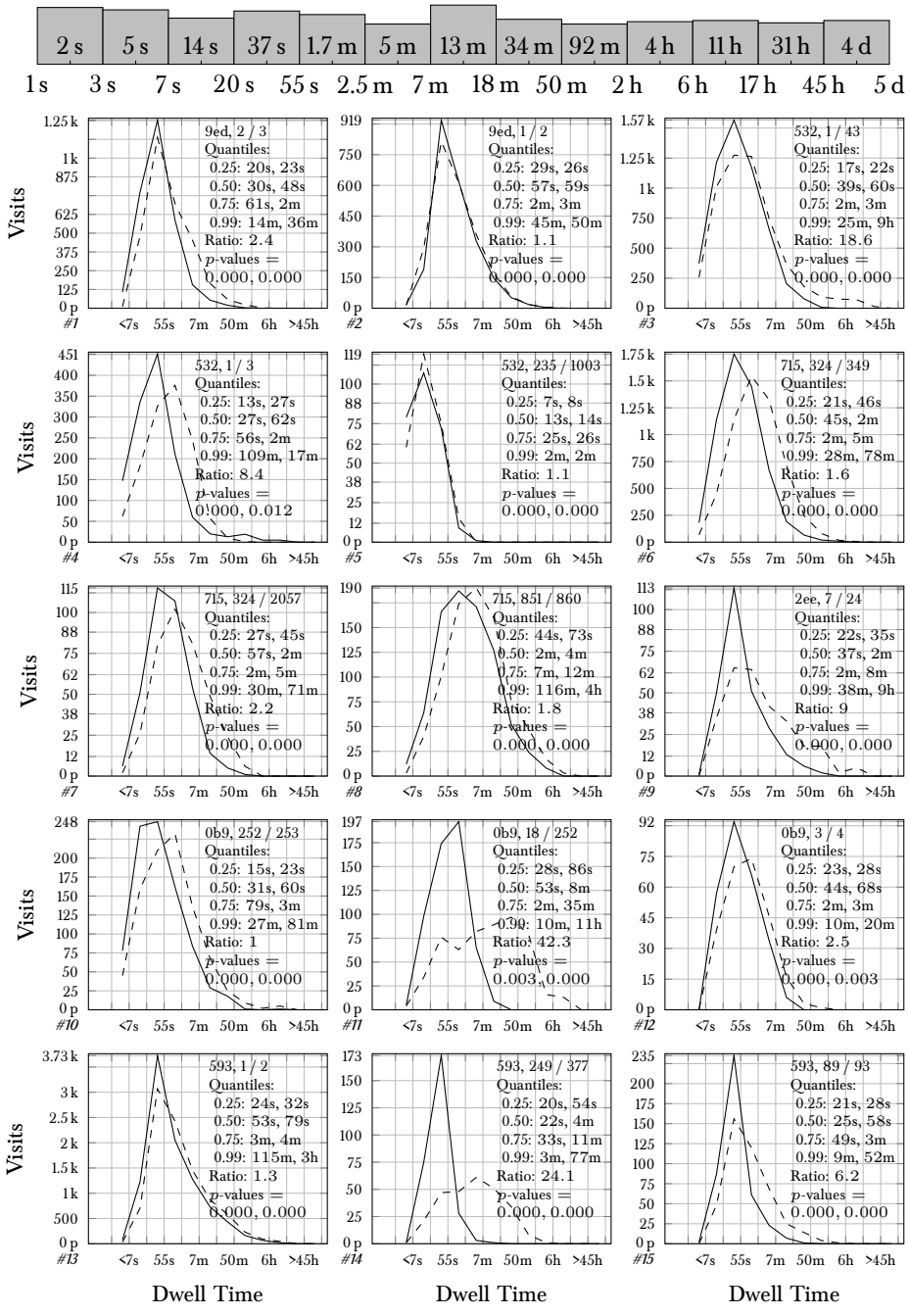


Figure 4.8 (Continued): Histograms of the dwell time of tower visits involved in oscillation sequences. The histograms provide a different view of the data presented in Figure 4.7.

set shows the data using histograms. To improve the representativeness of the displayed data, we do not plot more than three oscillation pairs for any user (recall: the data set includes 189 oscillation pairs). The solid lines correspond to log-normal regressions. The dashed vertical lines correspond to the combined median and the dotted vertical lines to the combined lower and upper quartiles.

Looking at the plots, we see that the basic shape of the distributions is similar across pairs and towers. In the CCDF plots, the data appears to follow a convex curve, which suggests a log-normal distribution. (The typical bell shape of a normal curve is perhaps easier to recognize in the semi-log histograms.) A visual comparison with the log-normal regressions suggests that this conclusion is often reasonable, however, nearly all towers have heavier tails than the log-normal regression predicts. A few of the pairs also have truncated tails. These are demarcated by the sudden change to nearly vertical lines in the CCDF plots. User e7d's third oscillation pair and 8be's third oscillation pair exhibit this feature. These very long visits may be due to the presence of mixed mode sequences (i.e., sequences in which the user moved between the oscillation location and a non-oscillation location covered by one of the towers).

Running the Shapiro-Wilks test of normality on the logarithm of the dwell times shows that a log-normal distribution is only a significant fit for 27 of the 378 data sets (7.0%) at the $p = 0.05$ level. Nevertheless, it is clear that the data is consistent with a right heavy tailed distribution: the data spans multiple orders of magnitude and the mean is much larger than the mode.

Although the basic shape of the distribution remains the same, the spread varies between oscillation pairs as well as within oscillation pairs. User af6's first plotted oscillation pair is an example of differing dwell time distributions for an oscillation pair: the time spent at one tower is typically much longer than the time spent at the other tower.

We can quantify the difference in dwell times for an oscillation pair by estimating the ratio between the empirical distributions. We do this by finding the value of f that minimizes the following equation:

$$J(f) = \sum (f \cdot A - B)^2 \tag{4.1}$$

where A and B are the ordered dwell times for each tower and f is the so-called factor. If the length of A and B don't match, then we select an evenly spread out sample from the longer set that is the same size as the shorter set. Thus, if A has 97 data points and B has 99, then we remove the 33^{rd} and 66^{th} data points from B. (Note: the number of elements in A and B will typically be very close, because the device is alternating between them.) If $f < 1$, then we set the ratio to $1/f$, otherwise we set it to f. In other words, we decide a posteriori that A has the shorter typical dwell time.

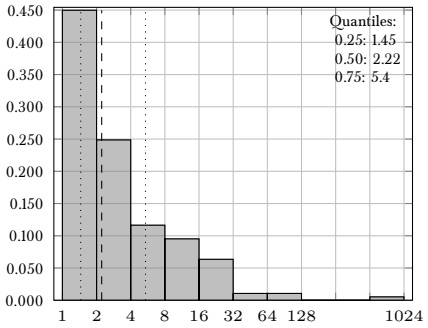


Figure 4.9: Histogram of the ratios of the dwell times of the towers in each oscillation pair. About half are within a factor of 2 of each other. Some of the pairs, however, exhibit dwell times that are orders of magnitude apart.

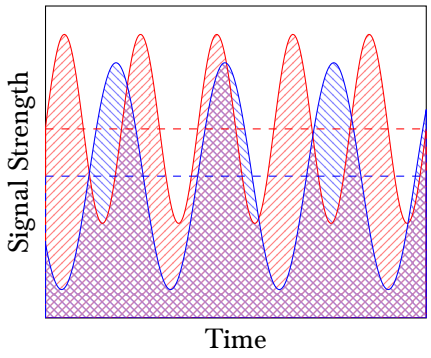


Figure 4.10: The hypothetical signal strength of two towers in an oscillation pair. The horizontal dashed lines show the signals' mean strengths. The tower with the stronger average signal (☑) is stronger approximately 67% of the time. Note that although the signal strengths fluctuate, their periods are not necessarily synchronized or even constant. Thus, we expect the stronger tower's dwell time will be about 1.5 times as long *on average*.

Figure 4.9 shows the ratio of the empirical distributions for the oscillation pairs in the form of a histogram. The histogram reveals that nearly half of the oscillation pairs have towers with comparable dwell times—i.e., within a factor of two of each other. A non-trivial number, however, have a much larger ratio. This confirms our theory that the distribution of dwell times for the towers in an oscillation pair are potentially different.

To better understand why the dwell time distributions have different spreads, consider again why oscillations occur: the towers in an oscillation pair have *com-*

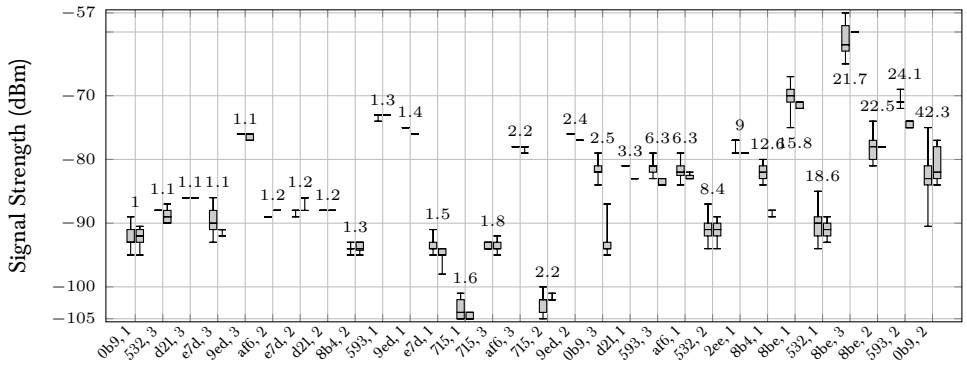


Figure 4.11: Comparison of the signal strength and the dwell time ratio of the oscillation pairs shown in Figure 4.7 ordered by the towers' dwell time ratio. (The whiskers correspond to the 2.5% and 97.5% quantiles.) The embedded text is the dwell time ratio computed using Equation 4.1.

parable signal strengths and these signal strengths are *unstable*. (If one tower always dominated, then the cell phone would not oscillate.[1]) These changes in signal strength are likely due to environmental factors, such as, signal propagation and background noise. Whatever the actual cause is, the practical result is that the signal strengths vary and the cell phone switches to the tower with the stronger signal.

This explanation suggests two underlying causes for differences in the spreads: the less stable a signal is, the more often the dominant tower changes; and, if one tower's signal is significantly stronger on average, then the cell phone will stay connected to that tower longer. The latter cause explains the high ratios observed in Figure 4.9 and the idea is illustrated in Figure 4.10.

Unfortunately, we cannot directly confirm these explanations using our data: we only know the signal strength of the connected tower; we don't know the signal strength of the other tower in an oscillation pair. Nevertheless, we can examine the signal strength of the towers when the user is connected to them. Figure 4.11 shows the range of signal strength for the pairs of towers shown in Figure 4.7. The displayed pairs are ordered by the ratio of their empirical dwell time distributions. The range of the observed signal strength is depicted using a box plot in which the whiskers correspond to the 0.025 and 0.975 quantiles.

With two exceptions (d21, 1 and 8be, 1), the observed signal strengths are comparable. This is particularly true for oscillation pairs with a small ratio

[1]Ignoring handoffs due to load balancing.

(< 2). Further, for most of these towers, the signal strength is very stable: the 0.025 and 0.975 quantiles of the observed signal strength are often nearly the same. Looking at the oscillation pairs with large ratios, we see that there are larger differences between the observed signal strengths and that at least one of the tower's observed signal strength is much less stable. This lends credence to our explanation of the causes.

Independent of the ratio, the typical dwell time is typically just a few minutes long. Indeed, the upper quartile rarely exceeds 5 minutes. However, as is the nature of right heavy tailed distributions, there are a significant number of visits with large dwells, which we cannot easily discount as outliers. Further, because there is rarely a break, it is difficult to attribute the long visits to user movement to a location covered exclusively by one of the towers.

In conclusion, *it appears that there is a common underlying process for the distribution of oscillation dwell times, which results in a single dominate distribution family albeit with differing scale parameters.* Thus, when formulating rules to classify tower visits, these rules should probably include oscillation pair specific knowledge.

Normal Tower Visits

We now consider the distribution of dwell times for normal towers. Figure 4.12 shows complementary cumulative Pareto plots of some of the non-oscillation location towers and Figure 4.13 shows the same data using histograms. Again, we only show at most three towers from any given user.

The distribution of dwell times looks surprisingly similar to the distribution of oscillation visit dwell times that we saw in Figure 4.7 and Figure 4.8. For instance, the dwell times span multiple orders of magnitude and appear to be distributed according to a right heavy tailed distribution and the median visit dwell time for most towers is still around a minute. The implied curves, however, are a bit less regular: there is typically a strong deviation in the middle.

We again fit the data to a log-normal function. Running the Shapiro-Wilks test of normality on the logarithm of the dwell times reveals a log-normal distribution is a significant fit for 145 of the 464 data sets (31.0%) at the $p = 0.05$ level despite the frequent deviation in the middle.

Although more of the data sets appear to follow a log-normal distribution, the best-fit parameters are again highly variable. This is perhaps due to the presence of different types of locations. Whatever the case, the most we can again say is that the data is consistent with a right heavy trailed distribution.

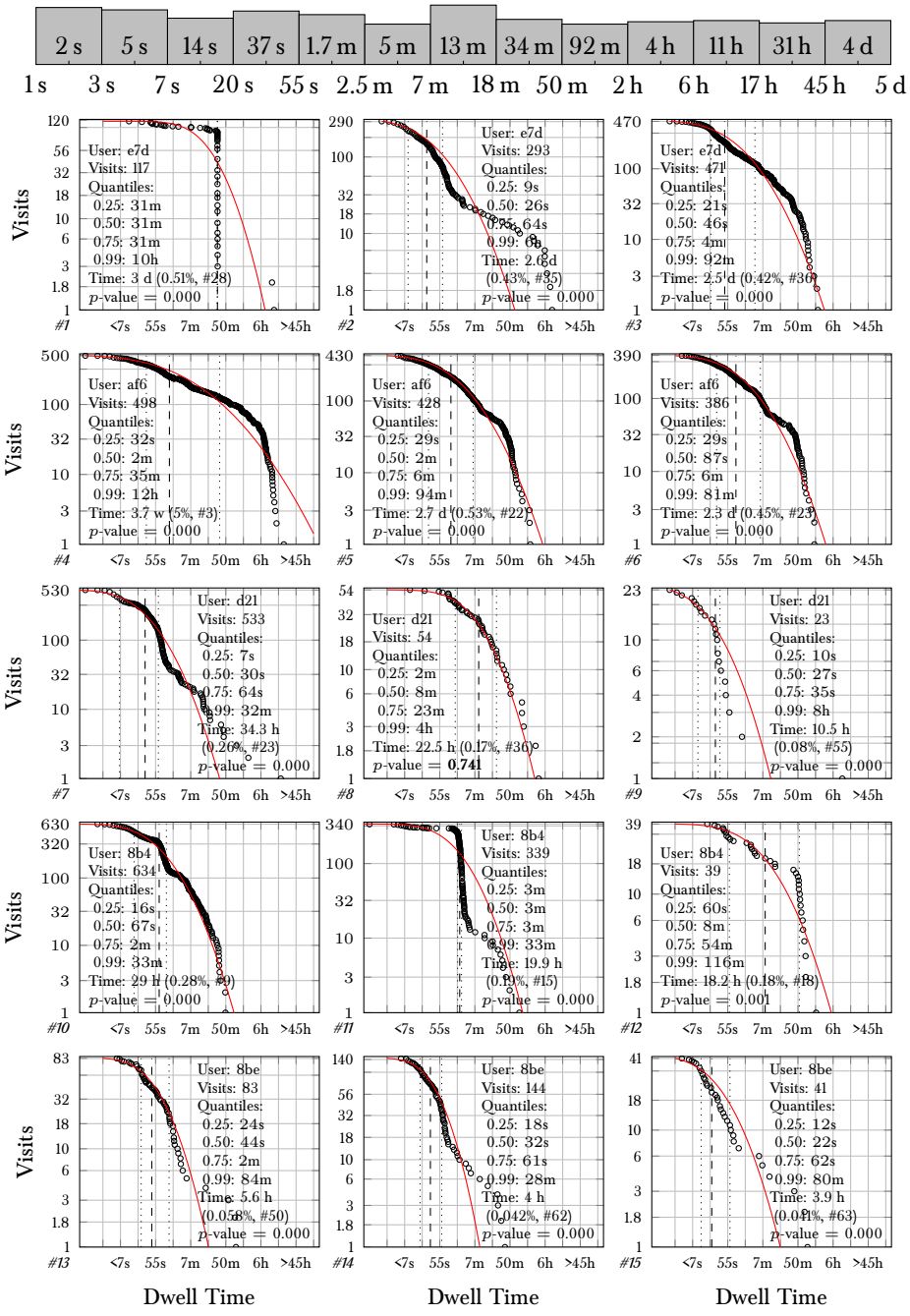


Figure 4.12: Complementary cumulative Pareto plots of the dwell time of tower visits to non-oscillation towers. Each plot corresponds to a single tower. The regression is to a log-normal function.

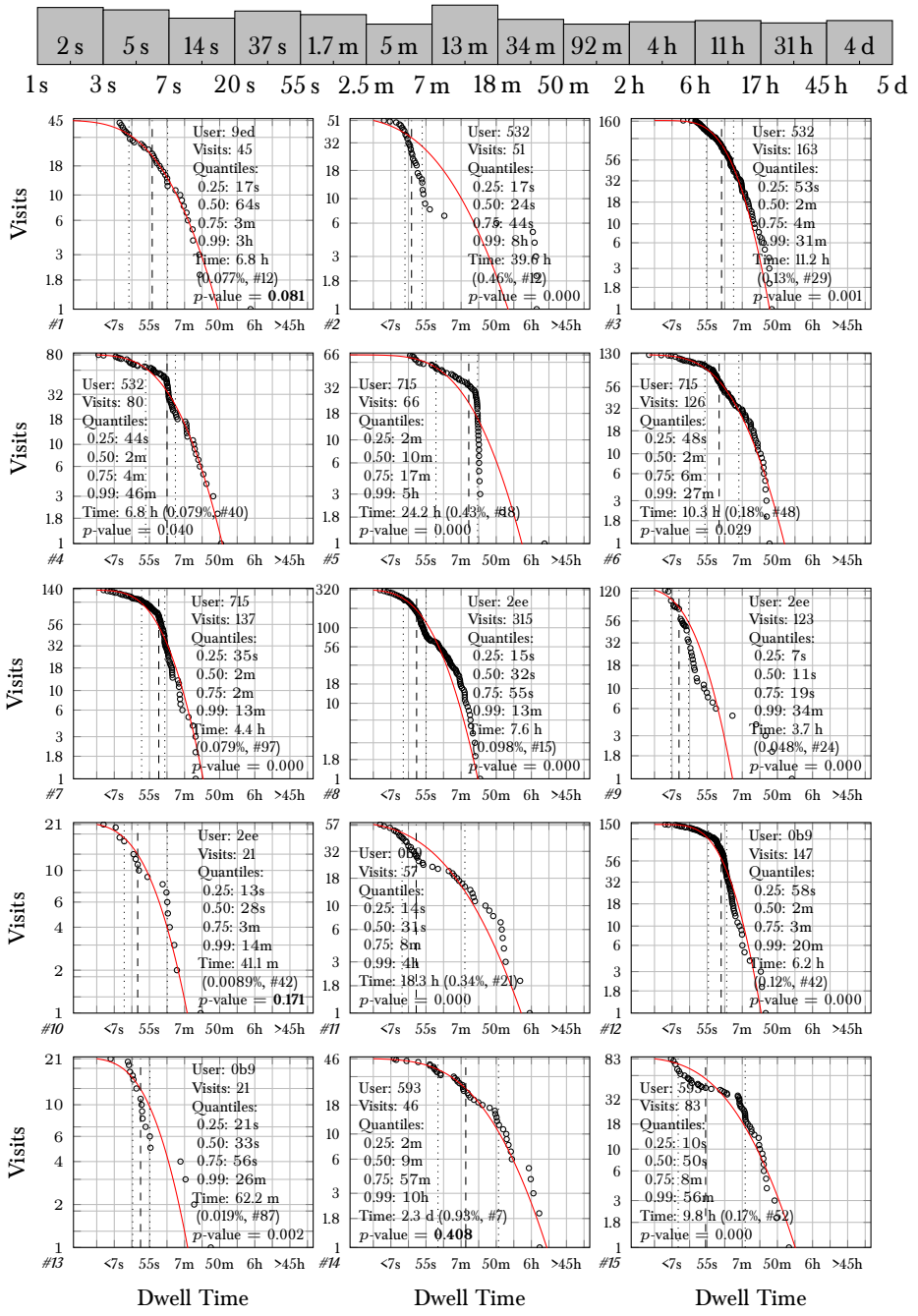


Figure 4.12 (Continued): Complementary cumulative Pareto plots of the dwell time of tower visits to non-oscillation towers. Each plot corresponds to a single tower. The regression is to a log-normal function.

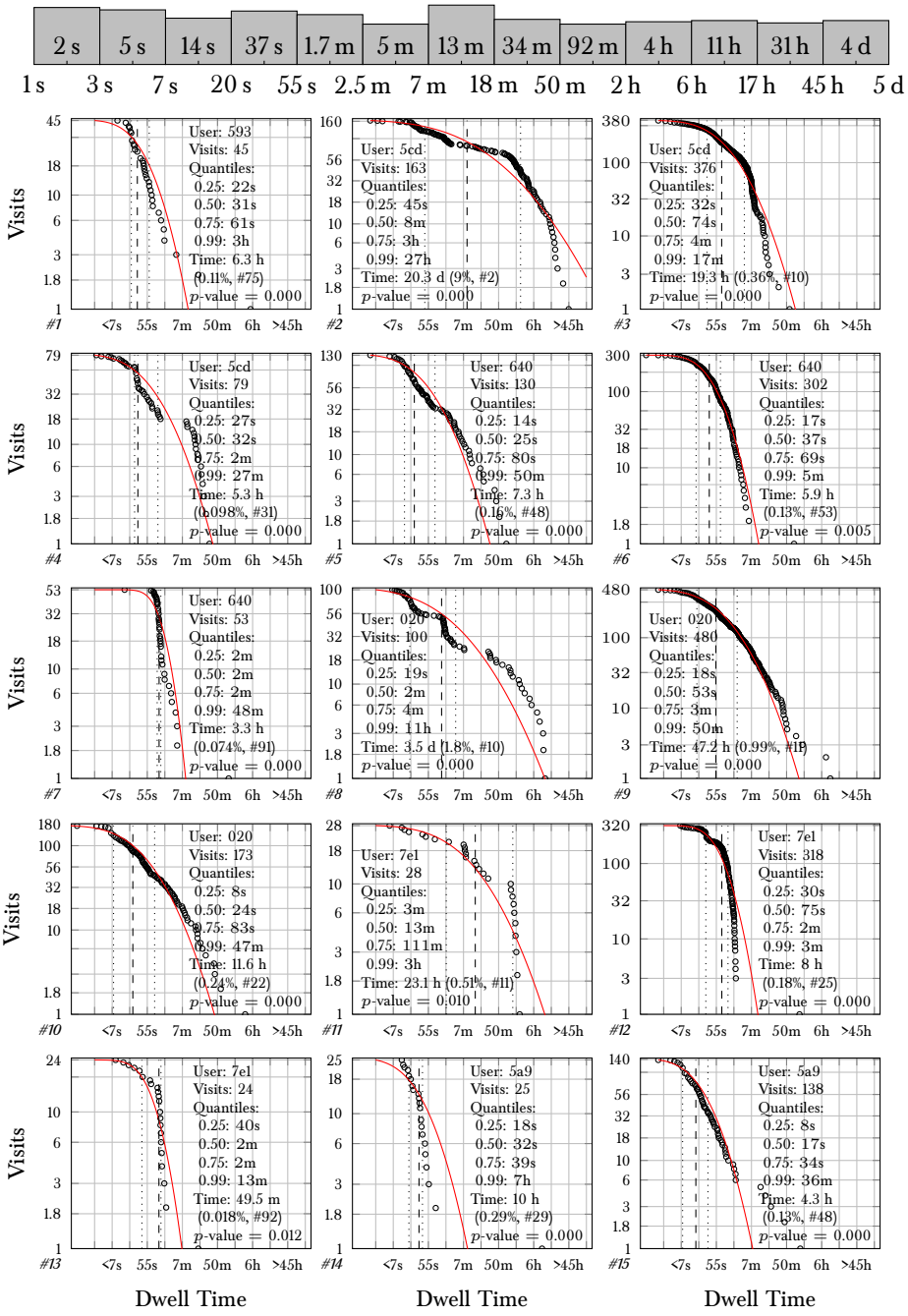


Figure 4.12 (Continued): Complementary cumulative Pareto plots of the dwell time of tower visits to non-oscillation towers. Each plot corresponds to a single tower. The regression is to a log-normal function.

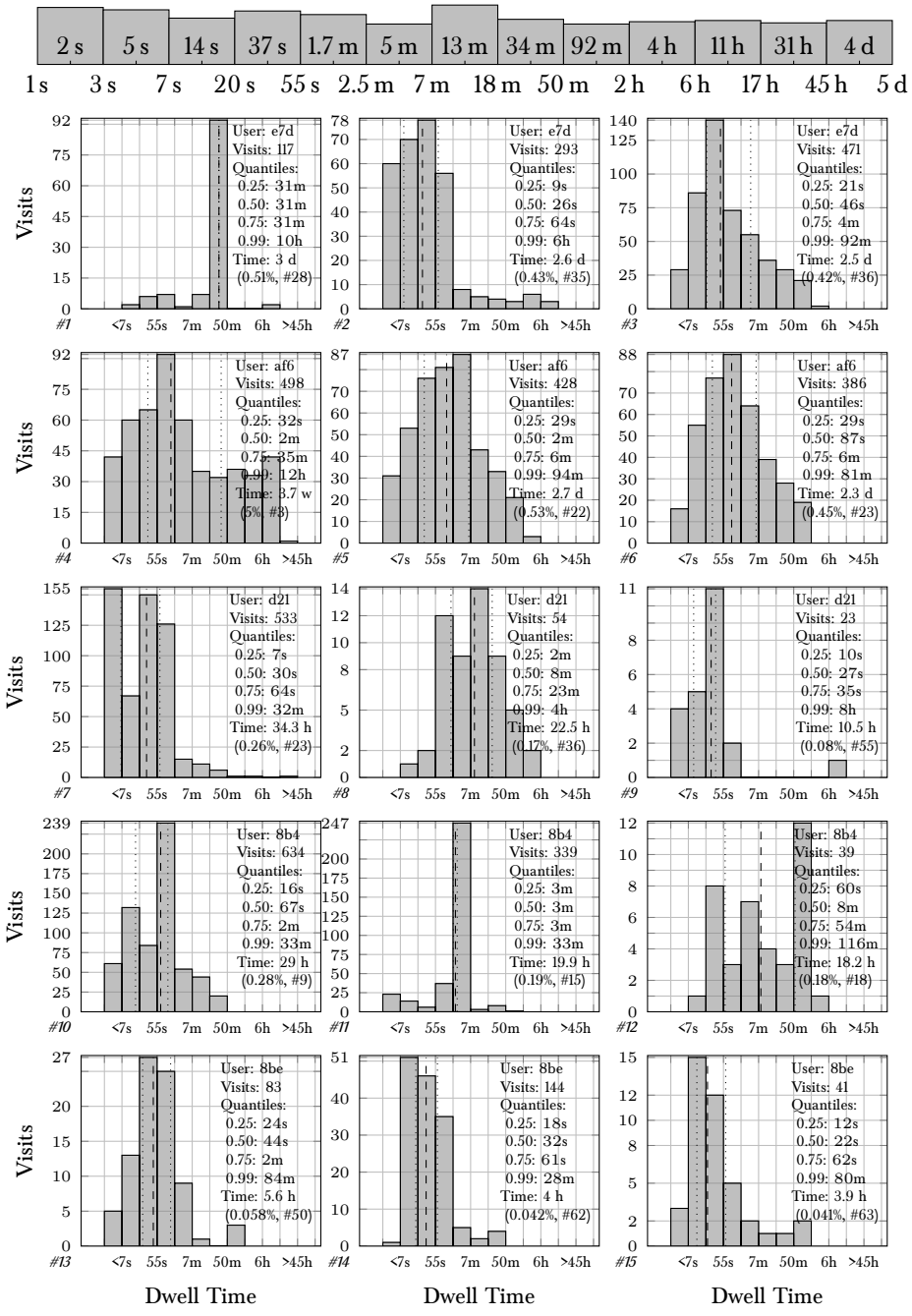


Figure 4.13: Histograms of the dwell time of tower visits to non-oscillation towers. The histograms provide a different view of the data presented in Figure 4.12.

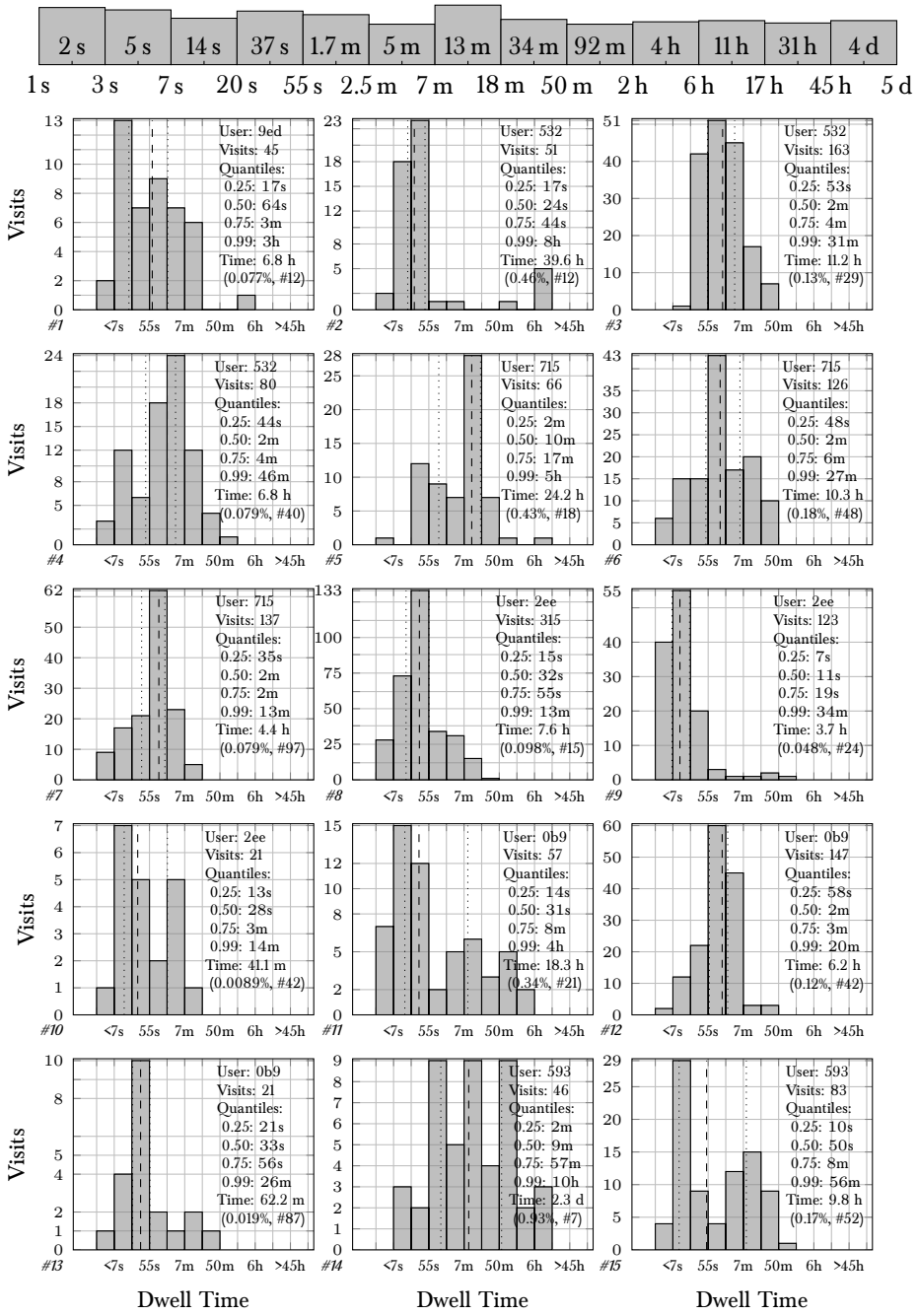


Figure 4.13 (Continued): Histograms of the dwell time of tower visits to non-oscillation towers. The histograms provide a different view of the data presented in Figure 4.12.

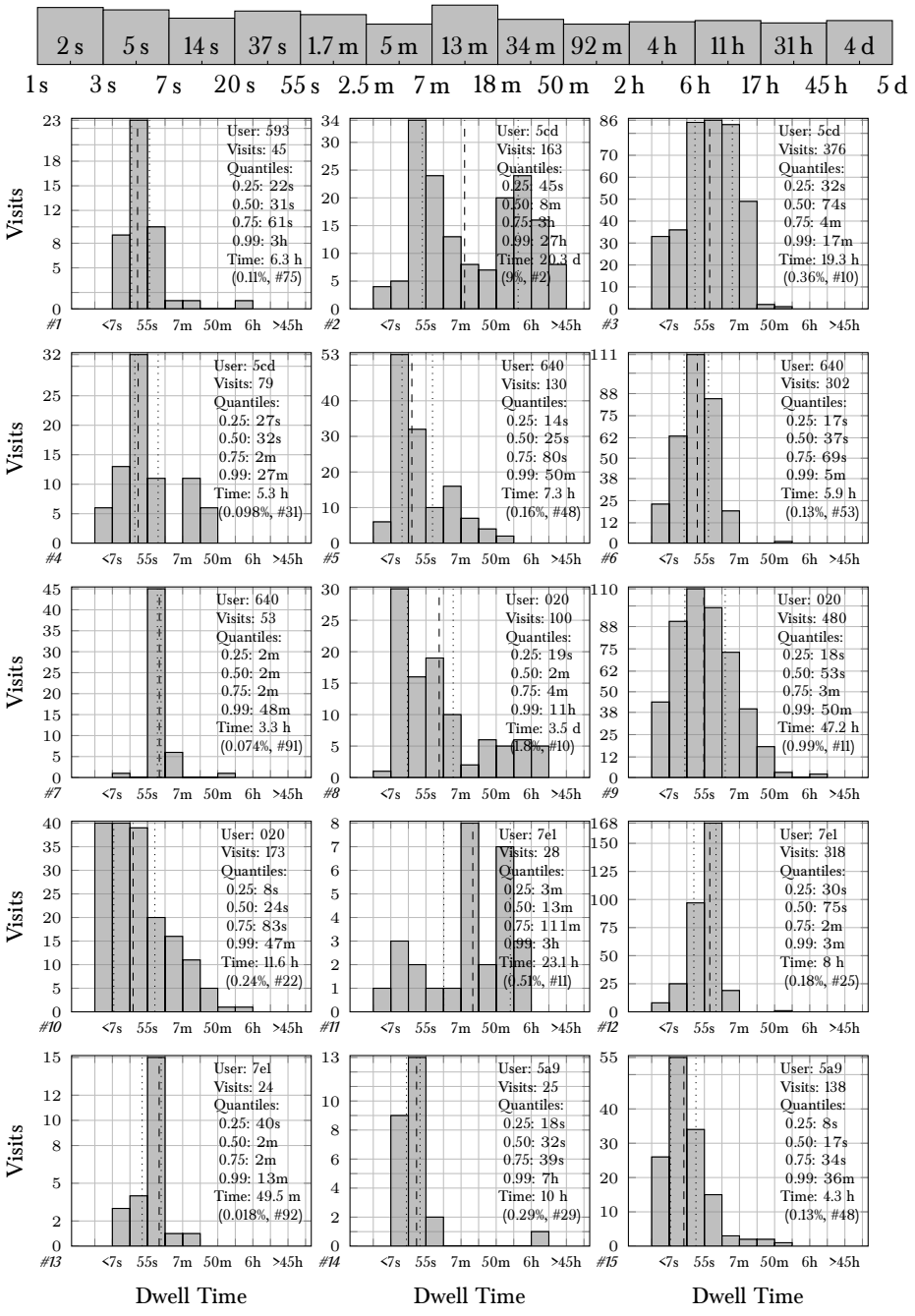


Figure 4.13 (Continued): Histograms of the dwell time of tower visits to non-oscillation towers. The histograms provide a different view of the data presented in Figure 4.12.

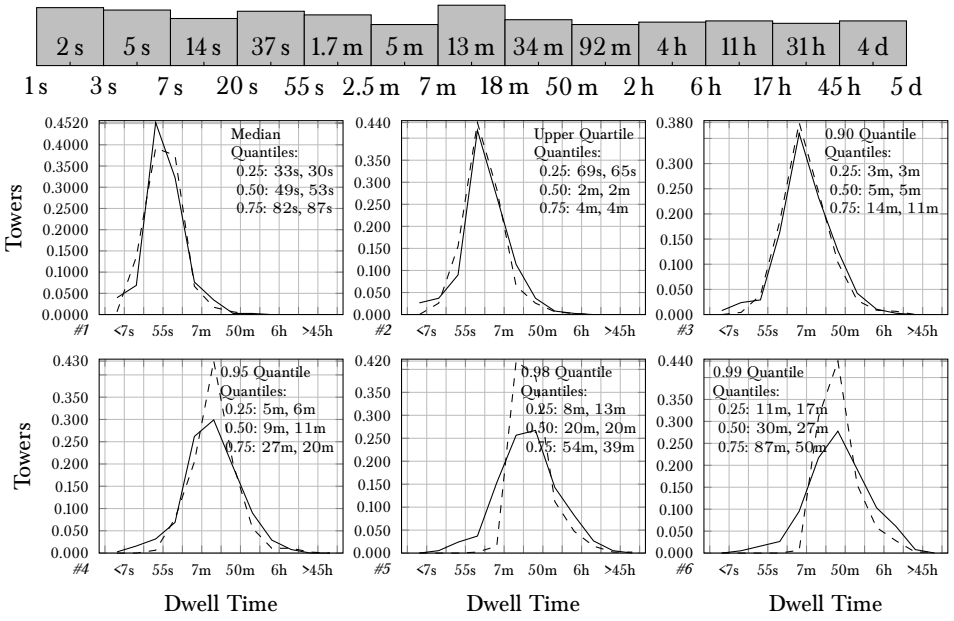


Figure 4.14: Histograms comparing different visit dwell time quantiles of oscillation pairs and normal towers across all users. The solid line corresponds to oscillation towers; the dashed line to normal towers.

Classification

Classification is a form of discrimination. Unfortunately, given the similarity between the distribution of the oscillation tower visit dwell times and the normal tower visit dwell times, we don't have much discriminative power.

To reinforce this negative result, consider briefly Figure 4.14, which shows a series of histograms depicting the distribution of various dwell time quantiles of the visit dwell time of the oscillation pairs and of the normal towers across all the users. For all quantiles, the two types of towers have very similar distributions.

It may be that normal towers really do include a lot of short visits and oscillation towers really do have some long time between visits. Of course, the long dwell times in oscillation sequences might be due to mixed mode sequences. Similarly, the short dwell times to normal towers might be due to clustering in which the user is actually oscillating between multiple towers. Our investigation suggests this latter possibility is, however, unlikely. Alternatively, it could be the criteria we used to select normal towers was poor. Ideally, an additional study in which GPS coordinates are also obtained would be run to better understand how user movement relates to the cell tower connection.

4.6.2 Sequence Length

In this section, we consider the use of an alternating sequence's length to classify the sequence as a whole. That is, we try to estimate $P(\textbf{type} \mid \text{sequence length})$ where **type** is either oscillation sequence or user movement sequence.

Problem Characterization

In Section 4.1, we observed that long alternating sequences most likely correspond to oscillations rather than user movement: most people rarely go back and forth between two places more than a few times. Consider user 9ed. This user's trace includes an alternating sequence that is 177 visits long (see Table 3.6 on page 109). It's hard to imagine someone going between two places 88 times without interruption. Indeed, even an alternating sequence consisting of 17 visits—one that is an order of magnitude shorter—seems unlikely to correspond to real user movement. Based on this observation, classifying long alternating sequences is easy: they are almost certainly oscillation sequences. In other words:

$$\lim_{\text{sequence length}\to\infty} P(\text{oscillation sequence} \mid \text{sequence length}) = 1 \qquad (4.2)$$

The problem, then, is two-fold: determining how long an alternating sequence needs to be to confidently conclude that it is an oscillation sequence; and, determining whether we can use a sequence's length to help classify shorter alternating sequences.

Long Alternating Sequences

We first try to come up with an appropriate value for the `long` parameter in Equation 4.2 (the minimum alternating sequence length that still almost certainly indicates that an alternating sequence is an oscillation sequence). Since our training data is largely based on finding towers with many long sequences, we can't use the data for finding the best value for this parameter. Instead, we reason based on intuition.

It is easy to imagine scenarios in which user movement results in alternating sequences that are just a few visits long. Consider a person who retraces her steps. This happens when she enters and leaves a building through the same door or goes home and then leaves a few hours later. This type of user movement can result in alternating sequences that are three visits long. For most people, this type of movement likely occurs many times each day. Of course, such movement will not always result in an alternating sequence; an alternating sequence will only arise if the user moves between two towers. It is entirely plausible, for

instance, that a store's interior and its parking lot are covered by the same tower. Alternating sequences that are five visits long result when, after visiting a store or leaving home, the person realizes that she forgot something and returns to get it. This type of movement likely occurs every day, but happens less often than the previous pattern. Longer alternating sequences arising from user movement are also conceivable. These can come about when loading or unloading a car, for instance. For most people, however, such activities are probably rare and occur at most a couple of times per day.

These scenarios are not exhaustive, but we think they are representative of user movement that results in alternating sequences. Given this characterization, we formulate two hypotheses draw two conclusions. First, *the number of alternating sequences corresponding to user movement decreases as sequence length increases.* That is, for user movement sequences, there is a negative correlation between sequence length and sequence length frequency. Second, *the number of alternating sequences corresponding to user movement approaches zero starting with sequences that are about seven visits long.*

We can empirically confirm the negative correlation between sequence length and sequence length frequency. This is easy to do, because it is true for nearly all towers involved in alternating sequences independent of whether they consist of user movement sequences or oscillation sequences.

Figure 4.15 shows a histogram of the Spearman correlation coefficient of alternating sequence length and frequency for towers that are involved in at least 10 alternating sequences that are at least 3 visits long. There are 3145 such towers across all of the traces. Requiring at least 10 alternating sequences helps ensure that we have a large enough sample to draw a statistically significant conclusion. When computing the correlation coefficient, we consider the alternating sequences that are at least 3 visits long. We use Spearman's correlation coefficient, because, unlike Pearson's correlation coefficient, which is more common, it is computed using ranks and is thus also able to evaluate the strength of non-linear relationships. In particular, this means that a monotonically increasing function will have a high correlation coefficient even if the function is non-linear.

Looking at the histogram, we see that nearly all of the towers have a moderate to strong negative correlation. Concretely, 49.0% of the towers have a correlation that is less than -0.7 and 75.0% of the towers have a correlation that is less than -0.5. These thresholds are crude thresholds that are used for identifying strong and moderate negative correlations, respectively [103, p. 184]. 31 of the towers (1.0%) have a non-negative correlation. Examining the cases in more detail, 19 of these (61.0%) are a simple inversion: the tower is involved in sequences that are either 3 or 4 visits long and there happen to be a few more sequences that are 4 visits long than 3 visits long. These exceptions do not con-

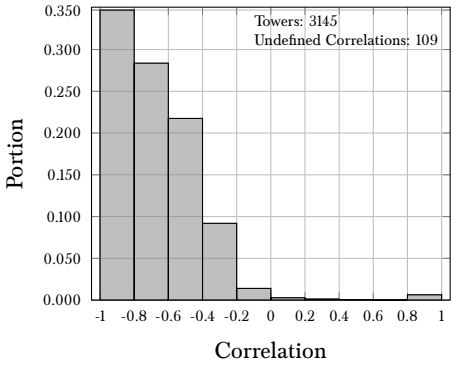


Figure 4.15: Histogram of the correlation of sequence length and sequence length frequency for towers involved in at least 10 alternating sequences that are at least 3 visits long. We only consider sequences that are at least 3 visits long.

tradict the general rule. The remaining 12 towers, however, really are outliers. These represent just 0.0% of the towers. There are also 109 towers (3.0%) that have an undefined correlation. This happens if all sequence lengths have the same number of sequences. In these cases, the standard deviation is 0, if there is more than one length, or undefined, if all the sequences are the same length. 104 of these towers (95.0%) fall into the latter category. Most of the remaining 5 towers with undefined correlations are involved in sequences that are equally divided between two different sequence lengths. These towers don't contradict the general rule either.

Given the moderate to strong negative correlation between sequence length and sequence length frequency for nearly all tower pairs, we conclude that both oscillation and user movement tower pairs exhibit this behavior. Although we can't confirm the hypothesis that user movement sequences are rarely longer than 7 visits without ground truth, we feel that the logical argument is sound.

Based on these two conclusions—that the number of alternating sequences corresponding to user movement decreases as sequence length increases and that the number of alternating sequences corresponding to user movement approaches zero starting with sequences that are about seven visits long—we conclude that *alternating sequences that are at least seven visits long are likely oscillation sequences and those that are at least ten visits long are almost certainly oscillation sequences.* Thus, in Equation 4.2, we should set `long` to be between seven and ten. We should choose a smaller value if we prefer to misclassify user movement as oscillations and a larger value if we prefer to misclassify oscillations as user movement.

Of course, no matter how conservative we are with respect to classifying user movement, there will always be an occasional long alternating sequence that corresponds to user movement. Our training data for non-oscillation locations, for instance, includes 48 towers (of 464 total non-oscillation location towers) with at least one sequence that is 7 visits long or longer. This is partially by construction, since we only include towers for which at most 1% of the effective visits are alternating sequences that are 7 visits or longer. It is probably reasonably safe to misclassify these as oscillation sequences: they are likely so rare in practice that they are essentially unpredictable.

Short Alternating Sequences

Classifying short alternating sequences is not as easy as classifying long alternating sequences: whereas long alternating sequences are almost always due to oscillations, *short alternating sequences can arise from either user movement or oscillations making the use of sequence length to classify short alternating sequences ineffective.*

We already argued that user movement typically results in short alternating sequences. In this section, we show that oscillation sequences can also be short by looking at the distribution of alternating sequence lengths on a per user basis.

Figure 4.16 shows a complementary cumulative Pareto plot of the length of alternating sequences that are at least three visits long on a per-user basis. Although oscillation sequences may be shorter than this, we don't consider them: every tower visit has a predecessor and a successor and thus could be considered to be involved in alternating sequences that are at least two visits long.

Looking at the plots, the lengths appear to follow a right heavy tailed distribution. We fit the data to a right censored power law as described in Appendix B and found that a power law is a statistically significant fit for 23 of the 59 traces (39.0%) at the $p = 0.05$ level.

The truncation in the tail is likely due to an upper limit on how long the user stays at a given location. A possible upper limit on visit length due to diurnal effects was discussed in Section 3.4.2 on page 84. Since oscillation sequence length and oscillation sequence dwell time are correlated, these effects will apply here as well.

The upward deviation on the left nearly always manifests itself as a smooth convex curve. The deviation typically occurs over the range $3 \leq \ell \leq 6$. For $\ell \gtrsim 7$, the data suggests a relatively straight line until the truncation point. Since the curve is upwards, i.e., we have more short alternating sequences than the power law predicts, and the transition to the straight line is smooth, it is plausible that there are two significant processes in this range. Given the shape,

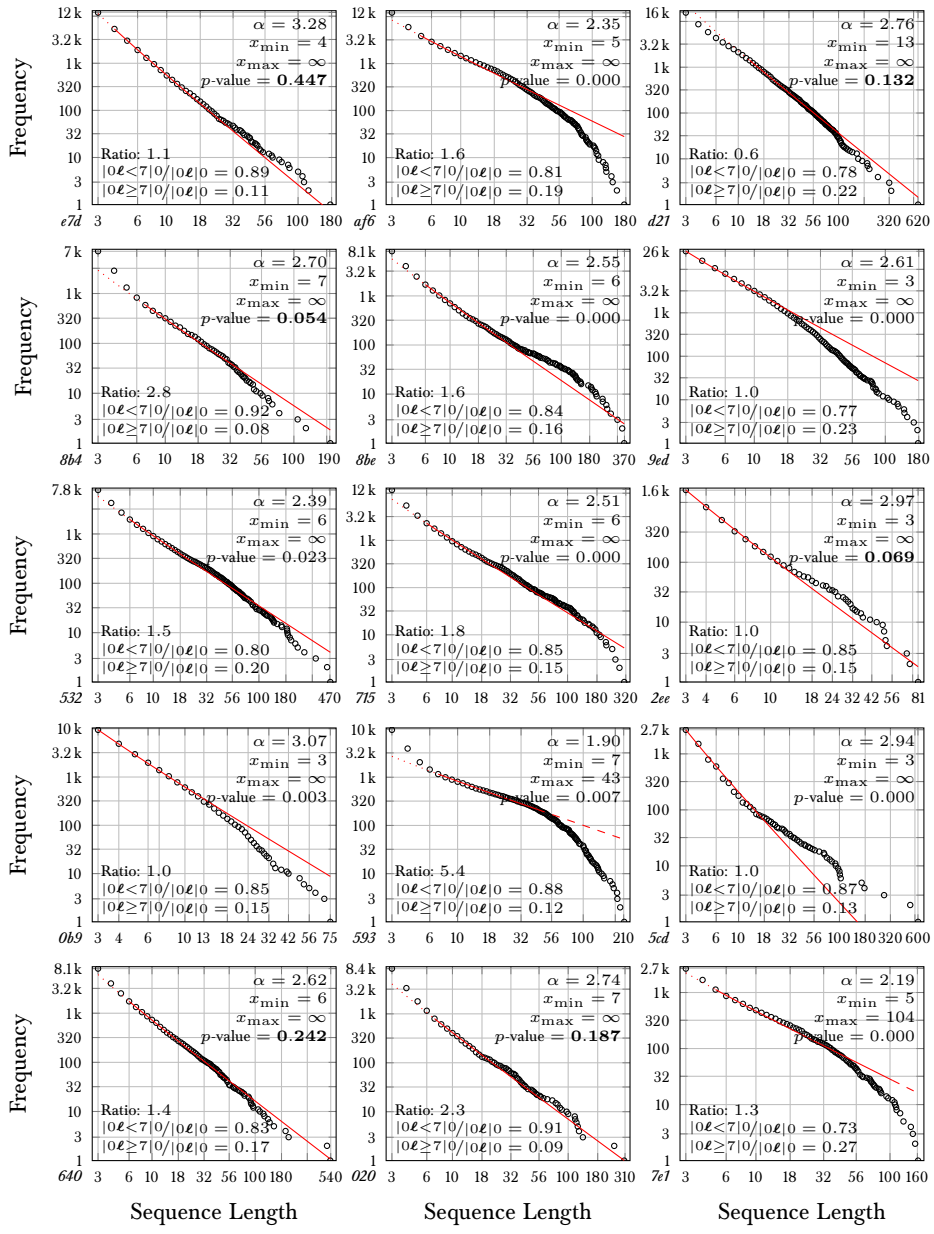


Figure 4.16: Complementary cumulative Pareto plots of the length of alternating sequences that are at least 3 visits long. 23 of the fits (39.0%) are significant at the $p = 0.05$ level. Of these, the average α is 2.59, with a standard deviation of 0.34. In the inset text, the ratio is the number of observed short sequences ($3 \leq \ell \leq 6$) to the number of expected short sequences based on the best fit power law, which we compute as follows: $P(3 \leq \ell \leq 6; \alpha, x_{\min} = 3) \cdot {}^{|0x \geq x_{\min}|0}/_{P(x \geq x_{\min}; \alpha, x_{\min}=3)}$.

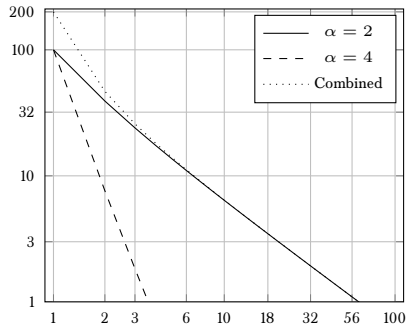


Figure 4.17: Complementary cumulative Pareto plot of two power laws added together. The left of the combined curve is concave, but quickly becomes straight; the curve with the higher α is steeper and its influence on the combined curve disappears quickly.

both processes could be power laws one of which has a significantly larger value of α than the other. Such a curve is shown in Figure 4.17 and looks quite similar to the curves in Figure 4.16.

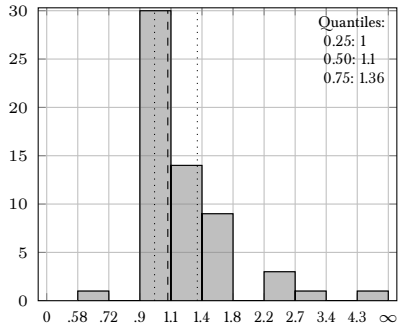
The steeper power law is an excellent fit for user movement sequences: $3 \leq \ell \leq 6$ is precisely the range over which we expect to see them. Further, as already discussed, we expect there to be a negative correlation between sequence length and sequence length frequency, which is consistent with a power law. The other power law is a good match for oscillation sequences for similar reasons. It thus seems reasonable to conclude that *short alternating sequences may be either oscillation sequences or user movement sequences.* Further, *the upper limit on the length of user movement sequences is approximately 6.*

We can estimate the ratio of user movement sequences to short alternating sequences by dividing the actual number of short alternating sequences by the expected number of short oscillating sequences. The algorithm for computing the best fit of a power law to data detects and ignores significant deviations on the left. Thus, the selected α will primarily reflect the α of the curve corresponding to the oscillation sequences, i.e., the shallower curve with the smaller value of α.

Figure 4.18 shows the ratio of observed short sequences to expected short sequences using this method. The histogram in Figure 4.18b shows the distribution ratios for sequences in the range $3 \leq \ell \leq 6$. The 7 outliers correspond to data like that which we looked at in Figure 4.16. The ratios of the rest are around one. The median is 1.1 and the lower and upper quartiles are 1 and 1.36, respectively. Approximately a quarter of the ratios are less than 1. These ratios, however, are

	Empirical/Expected	
Length	Median	MAD
3	1.20	0.31
4	1.03	0.12
5	1.02	0.12
6	0.969	0.14
$3 \leq \ell \leq 6$	1.10	0.18

(a) Table of the median observed short sequences to expected short sequences for different sequence lengths and ranges.



(b) Histogram of the observed short sequences to the expected short sequences for sequences in the range $3 \leq \ell \leq 6$

Figure 4.18: Ratio of the observed short sequences to the expected short sequences by user.

very close to 1. The implication is that these users don't have any user movement sequences. This is possible, but it seems more likely that there is some noise. All things considered, it seems that *approximately 1 out of 6 short alternating sequences correspond to user movement sequences.* As such, since neither user movement nor oscillation sequences clearly dominate, *we conclude that sequence length does not help in the classification of alternating sequences that are shorter than seven visits long.*

Note: there are only a tenth as many long alternating sequences—those that are at least seven visits long—as there are short alternating sequences—those whose length is between 3 and 6 visits long. This is expected given that sequence length is distributed according a power law distribution. The practical implication is that we will only confidently classify a small portion of the alternating sequences using Equation 4.2. Nevertheless, those that we do confidently classify are the more important long (in terms of total time) alternating sequences.

Conclusion

Both the length of oscillation sequences and the length of user movement sequences appear to be roughly consistent with a power law distribution. The distribution of the length of user movement sequences is, however, much steeper. The effect is that alternating sequences that are at least 7 visits long are most likely oscillation sequences. Indeed, the longer the alternating sequence, the

more likely this is the case. There are certainly a few long alternating sequences that arise from user movement, however, these are likely so rare that they are unpredictable in practice and misclassifying them as oscillation sequences is benign. Given the overlap of the distributions—there is about 1 user movement sequence for every 5 short oscillation sequences—it appears that the use of sequence length to classify alternating sequences less than about 7 visits long will be mostly ineffective.

4.6.3 Alternating Pair Type

We now investigate whether pairs of towers that often end up in alternating sequences consist primarily of oscillation sequences or user movement sequences. If so, then once we establish an alternating pair's type, labeling all future alternating sequences is straightforward. In fact, we don't have to wait very long: as soon as we detect the start of an alternating sequence, we can classify it.

From Section 4.2 and Section 4.6.2, we know that long alternating sequences are almost certainly oscillation sequences. We don't know, however, whether a given short alternating sequence is due to oscillations or user movement. To try and determine how short alternating sequences arose, we look at the distribution of the length of alternating sequences on a per-tower basis. If the distribution appears to smoothly transition between short and long alternating sequences, then the alternating sequences are probably generated by a single process. If, on the other hand, there is an upward deviation on the left, then we probably have a mixture of multiple processes.

We start by considering alternating pairs with many long oscillation sequences. Concretely, we again restrict our attention to the 189 alternating pairs with at least 20 alternating sequences that are at least 10 visits long. This time, however, we consider not just the alternating sequences that are at least 10 visits long, but all alternating sequences that are at least 2 visits long.

Figure 4.19 shows complementary cumulative Pareto plots of the length of alternating sequences for different alternating pairs. The figure includes at most 3 pairs from any given user.

Looking at the figure, all of the plots appear to follow the same basic pattern: there are two relatively straight line segments that are joined by a relatively smooth concave-downward curve. Sometimes, the curve is gradual and the line segments disappear; other times, the transition is more sudden and the curve better described as a joint. This pattern indicates that the data is extremely unlikely to have come from an untruncated power law. A right censored power law, i.e., a power law for which some external process limits values beyond some threshold, is, however, a possibility.

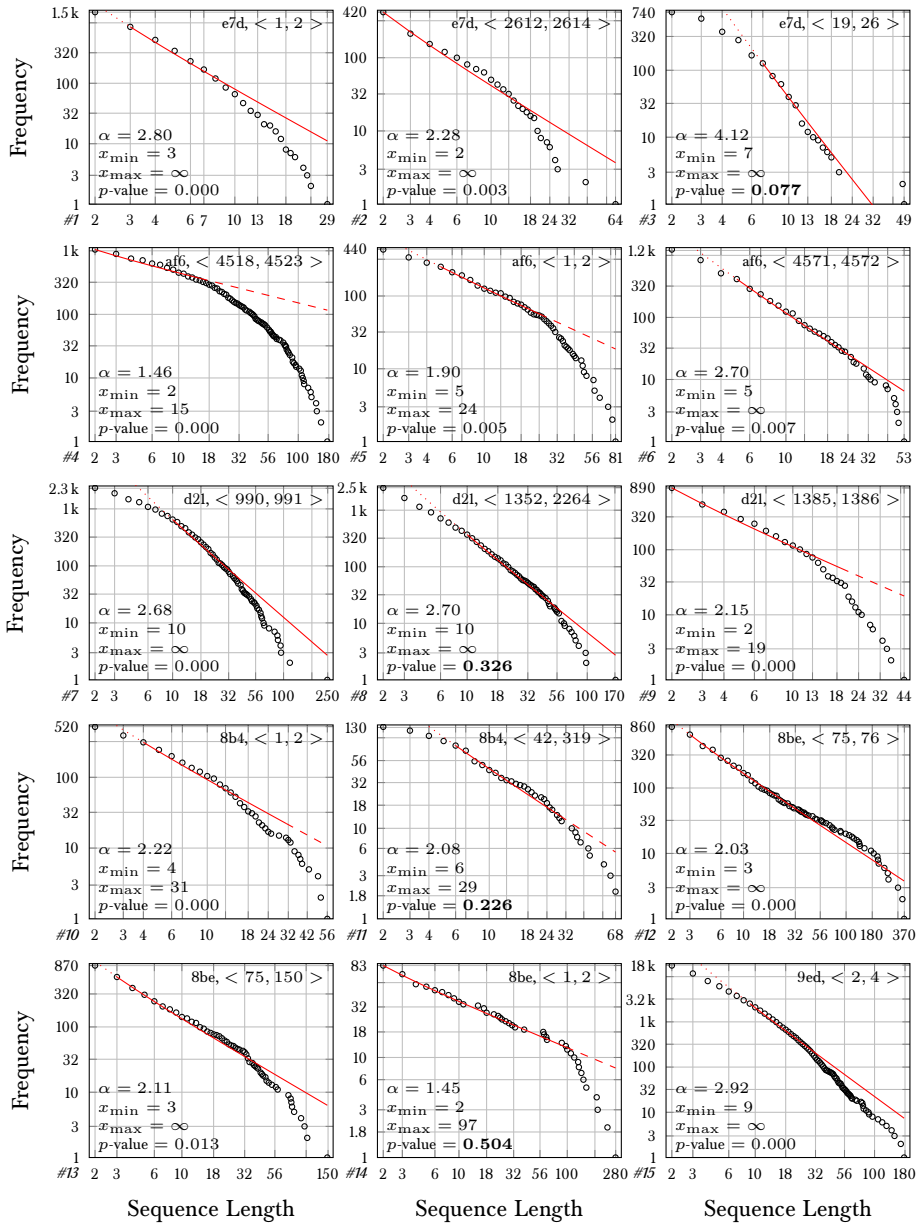


Figure 4.19: Complementary cumulative Pareto plots of the length of alternating sequences by alternating pair. Each plot corresponds to a significant oscillation pair. Of the 189 pairs, 90 (48.0%) are statistically significant fits to a right-censored power law at the $p = 0.05$ level. Of these, the average α is 2.38 with a standard deviation of 0.65. The median is 2.27.

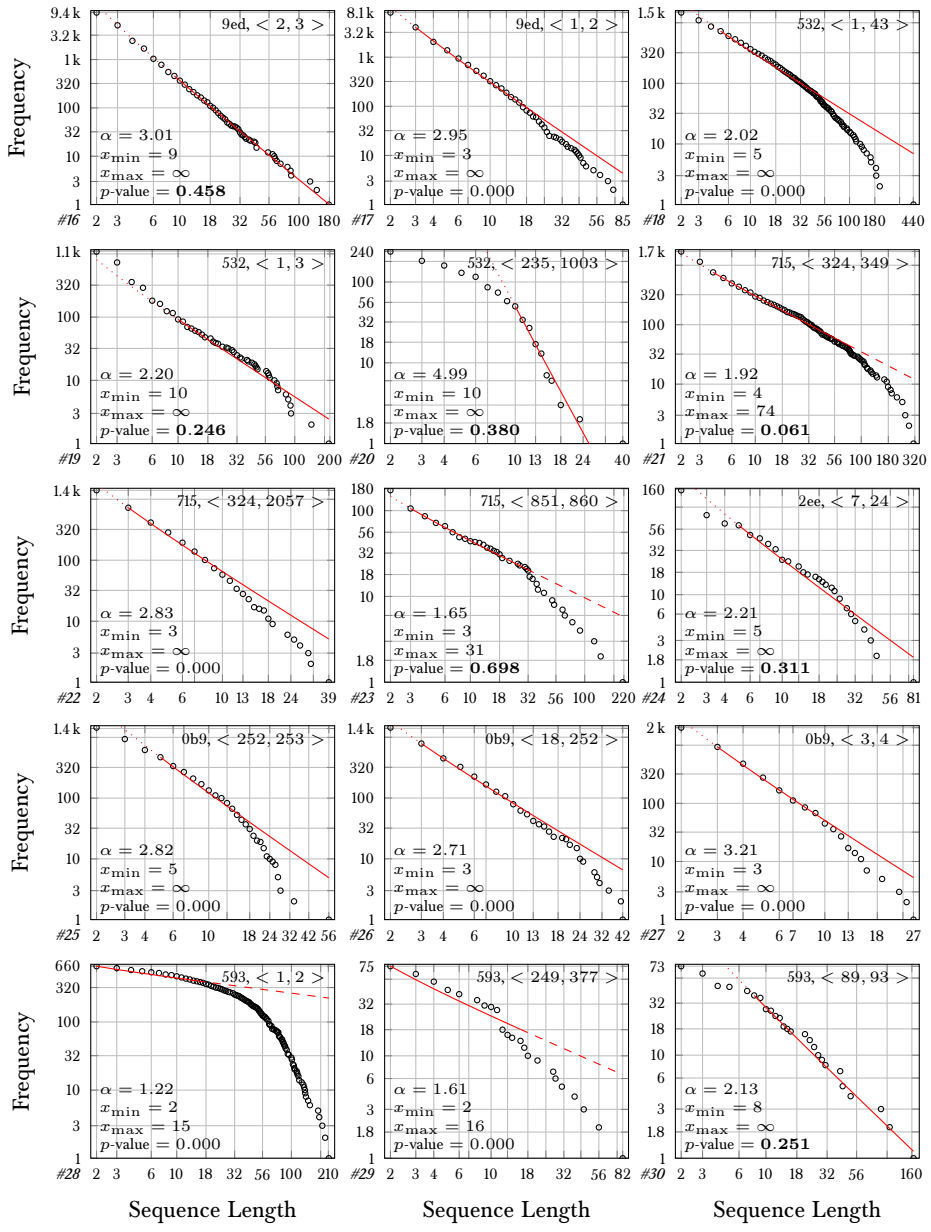


Figure 4.19 (Continued): Complementary cumulative Pareto plots of the length of alternating sequences by alternating pair. Each plot corresponds to a significant oscillation pair. Of the 189 pairs, 90 (48.0%) are statistically significant fits to a right-censored power law at the $p = 0.05$ level. Of these, the average α is 2.38 with a standard deviation of 0.65. The median is 2.27.

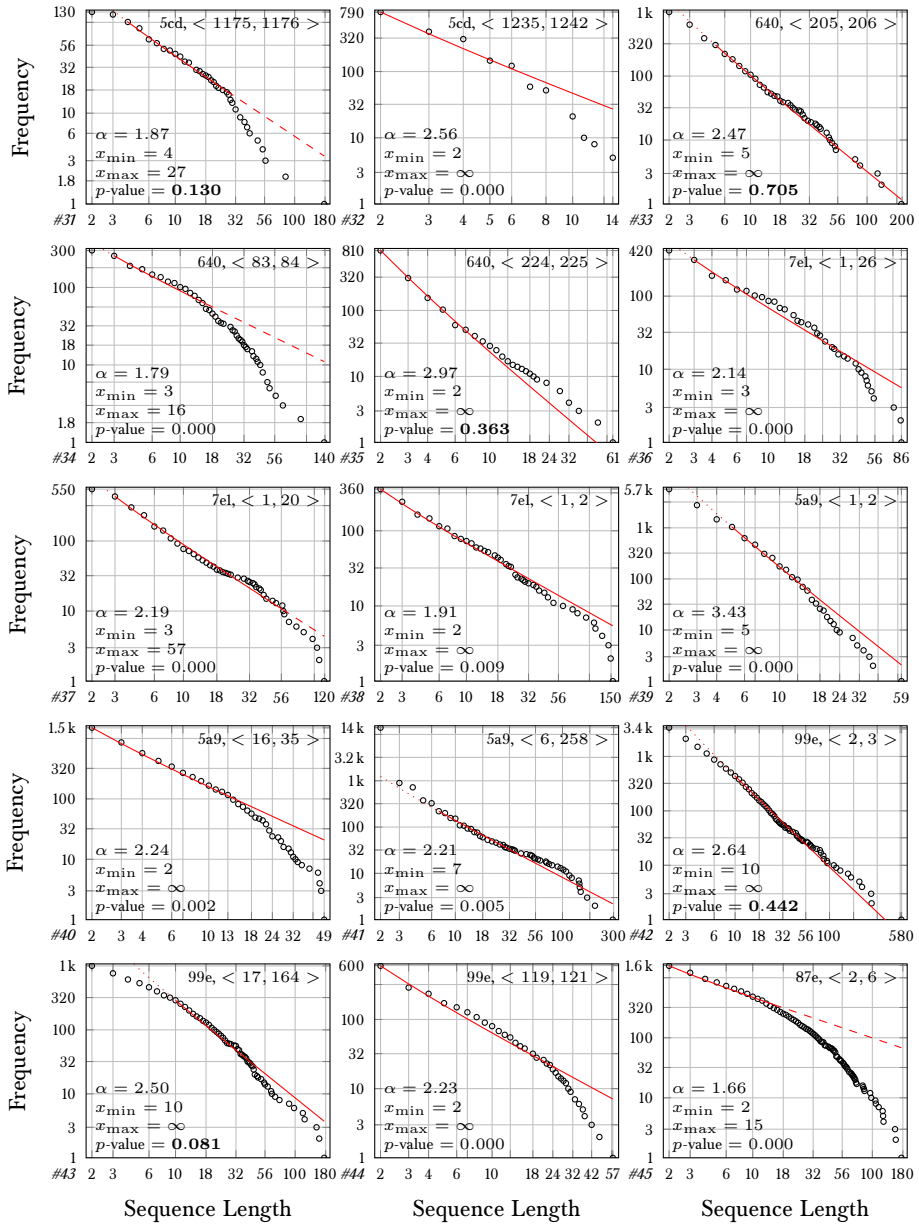


Figure 4.19 (Continued): Complementary cumulative Pareto plots of the length of alternating sequences by alternating pair. Each plot corresponds to a significant oscillation pair. Of the 189 pairs, 90 (48.0%) are statistically significant fits to a right-censored power law at the $p = 0.05$ level. Of these, the average α is 2.38 with a standard deviation of 0.65. The median is 2.27.

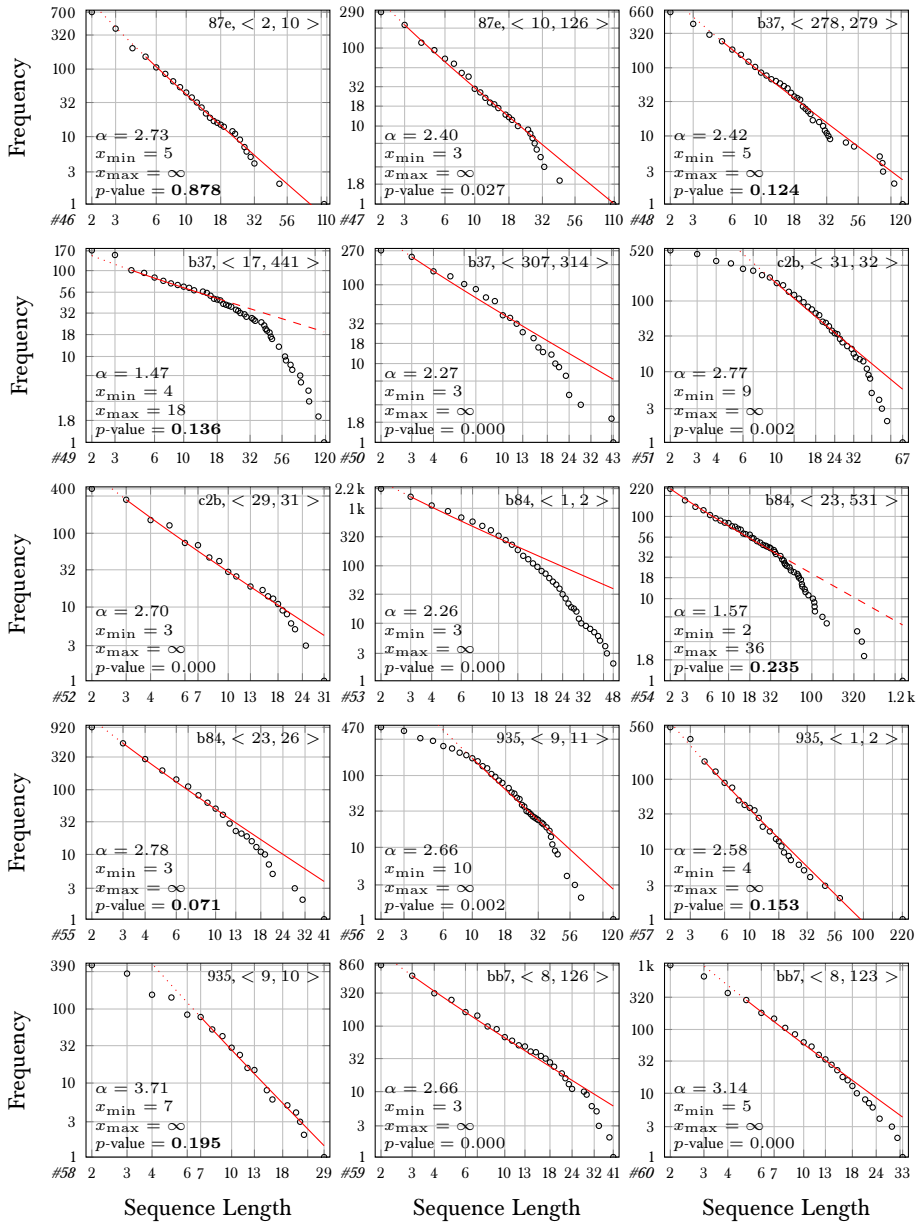


Figure 4.19 (Continued): Complementary cumulative Pareto plots of the length of alternating sequences by alternating pair. Each plot corresponds to a significant oscillation pair. Of the 189 pairs, 90 (48.0%) are statistically significant fits to a right-censored power law at the $p = 0.05$ level. Of these, the average α is 2.38 with a standard deviation of 0.65. The median is 2.27.

The downward deviation on the right is probably a result of diurnal effects. Recall that when we looked at visit dwell times in Section 3.4.2 that we also saw a consistent downward deviation in the tail starting at about 12 hours. We are likely seeing the same diurnal effect here. The fact that the downward deviation starts at different sequence lengths for different alternating pairs is easily explained: different tower pairs have different typical oscillation periods. It's unclear, however, why the transition is sometimes so gradual and other times quite sudden.

Based on these observations, we fit the data to a right-censored power law. We limited $x_{\min}$ to be at most 10 to ensure that the fit is to the left line segment and not to the tail. We chose 10 based on two factors. First, the curve usually begins after this point. Second, we don't want to force the fit to include user movement sequences, which will probably show up as an upward deviation on the left. Since we don't expect many user movement sequences to be longer than 10 visits, this limitation allows the algorithm to maximize the amount of data used in the fit while not forcing the fit to include a deviation.

The fits indicate that a right-censored power law is often a reasonable fit. Of the 189 pairs, 90 (48.0%) are statistically significant at the $p = 0.05$ level. Of these, the average α is 2.38 with a standard deviation of 0.65. The median is 2.27.

Looking at the plots, the fits that are not statistically significant tend to fall into two categories. There are those that have a more gradual curve like the fourth and seventh plots and there are those whose domain doesn't span multiple orders of magnitude, such as the first and second plots. The poor fit to the data sets in the latter category may be due to the penalty that the fit procedure applies if $x_{\min}$ and $x_{\max}$ are too close based on the observation that power laws tend to span multiple orders of magnitude. This effectively forces the selection of $x_{\max} = \infty$. The limited span may be due to a large typical oscillation period. As such, these data sets could really be distributed according to a right-censored power law.

Although the data is relatively well described by a right-censored power law, there is relatively high variability in the estimated values of α. Figure 4.20 shows the distribution of the estimated values of α. The values range from 1.12 to 4.99. To better understand the amount of variance, consider that Clauset et al. observe that a typical value of α for power laws occurring in nature is between 2 and 3 [6, Page 2]. The cause of this enormous variance is likely again the variation in the typical oscillation period: since the length of an oscillation sequence is correlated with the sequence's dwell time, a shorter oscillation period will result in more oscillations and thus longer sequences over a given amount of time. This corresponds to a steeper curve (proportionally more shorter visits)

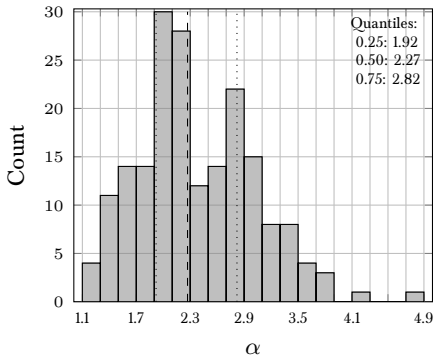


Figure 4.20: Distribution of the estimates of α for the best fits of the right-censored power law to each pair's data.

and larger values of α.

So far, we have explored the obvious deviations. We now consider a notable consistency: the straight line implied by the data on the left remains straight even in the region where we expect to see user movement sequences (i.e., $\ell \lesssim 6$). There are only a handful of tower pairs with some minor upward deviation (e.g., plots 2, 12, 19 and 21). Interestingly, there are also some pairs with a downward deviation in this area (e.g., plots 3, 20 and 30), which we can't explain. The general lack of a deviation in this region is notable, because if the pairs had a significant number of user movement sequences, then we would expect an upward deviation in this region. Since it is unlikely that the combination of two unrelated processes would so consistently result in a smooth transition, the simplest (and, applying Occam's razor, the most likely) explanation is that the observations arises from a single process.

Conclusions: Since we are convinced that long alternating sequences are due to oscillations, we conclude that in these cases the short alternating sequences are too. This conclusion is reinforced by observing that about a tenth of the plots do, in fact, exhibit a slight upward deviation for small values of ℓ, i.e., for those values of ℓ for which user movement sequences are conceivable. The increased steepness means that there are more alternating sequences with these lengths then the censored power law predicts. This suggests an additional underlying process, which we attribute to the added presence of user movement sequences. Nevertheless, oscillation sequences dominate in these cases. Thus, it seems reasonable to conclude that when there are oscillation sequences, they dominate any user movement sequences.

4.7 A Heuristic

We now develop a simple heuristic for classifying alternating sequences. We focus on finding a reasonable heuristic rather than an optimal algorithm due to the limited success of our search for discriminating features in the previous section. Despite this compromise, we still first appeal to data rather than intuition to develop the rule.

4.7.1 Possible Features: A Review

In the previous section, we examined three features that could help determine whether tower visits are due to oscillations or user movement.

First, we looked at visit dwell time and determined that without more data, in particular, without a better ground truth reference, we can't use visit dwell time to classify tower visits. An important practical result is that we are unable to identify mixed mode sequences, i.e., alternating sequences in which the user spends time at two locations, an oscillation location and one or more fixed locations each covered by just one of the alternating towers (see Section 4.3).

We then considered sequence length. We concluded that long alternating sequences are almost certainly oscillation sequences. The difficulty was defining long. We argued that alternating sequences that are at least 7 visits long are probably oscillation sequences and that alternating sequences that are at least 10 visits long are almost certainly oscillation sequences.

Finally, we examined whether alternating sequences involving a particular tower pair consist primarily of oscillation sequences or user movement sequences. We observed that tower pairs with many long alternating sequences appear to be dominated by oscillation sequences and concluded that alternating tower pairs are probably typed. That is, if an alternating pair has oscillation sequences, then nearly all alternating sequences involving this pair will be oscillation sequences.

4.7.2 The Heuristic

We now propose a simple heuristic for classifying an alternating sequence. The basic idea is as follows: once we see a certain number of long alternating sequences involving a particular alternating pair, we declare that all alternating sequences involving that pair are oscillation sequences. In effect, this heuristic imposes an absolute threshold. It initially assumes that all alternating sequences are user movement sequences. Once it has sufficient evidence to the contrary, it assumes that all alternating sequences involving the given alternating pair are oscillation sequences.

Parameter	Description
`long`	Minimum length for an alternating sequence to be considered long.
`seq count`	Number of long sequences before a tower pair is considered an oscillation pair.
`lead`	Number of tower visits before an alternating sequence is classified.

Table 4.2: Summary of the heuristics parameters.

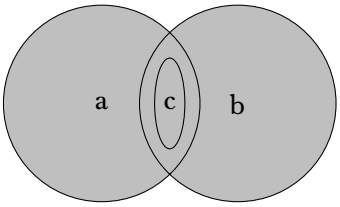


Figure 4.21: Oscillation locations, such as c, are bordered by exactly two towers (in this case, a and b).

The detailed algorithm is shown in Listing 4.1. There are three parameters. `long` is how long a sequence needs to be to be considered a long alternating sequence. `seq count` is the number of long alternating sequences it must observe before it starts classifying alternating sequences as oscillation sequences. `lead` is how long to wait (in terms of tower visits) before classifying an alternating sequence. These parameters are summarized in Table 4.2.

The first two parameters are straightforward: they determine the threshold; the last parameter, `lead`, can use some explanation. When considering a tower, if the length of the observed sequence so far is less than `lead`, then we just emit the tower as is. The idea is that if the sequence is short, then the user didn't stay in the location very long and thus the sequence was probably due to user movement. (An improvement on this parameter would be to consider time as well. However, this added complexity is not justified without further research.) Further, by always emitting at least one tower, the user always appears to move to the oscillation location via one of the two oscillation towers. This is consistent with the likely geographical layout as shown in Figure 4.21: an oscillation location is only bordered by the oscillation towers. Thus `lead` should be at least 2.

The `lead` parameter impacts classification as follows. Consider the sequence

```
1  // − towers: list of all observed towers so far.
2  // − tower: the most recently seen tower (i.e., the tower under consideration).
3  // − Returns the tower id to emit in place of tower (if this is the same as the last
4  // emitted id, then nothing is emitted).
5  function observe_tower(towers, tower)
6  {
7     // The previous tower (towers[length(towers)]) and the current tower
8     // (tower) determine the tower pair.
9
10    // Find the start of the alternating sequence.
11    i = length(towers)
12    while (towers[i − 1] == towers[length(towers)] || towers[i − 1] == tower)
13       i −−
14
15    // Extract it.
16    sequence = { towers[i:length(towers)], tower }
17
18    // We never modify the initial $lead−1$ visits in a sequence.
19    if (length(sequence) < lead)
20       return (tower)
21
22    tower_a = sequence[1]
23    tower_b = sequence[2]
24
25    if (length(sequences_at_least_x_long(towers, tower_a, tower_b, long)) >= seq count)
26       // The threshold has been met. This is an oscillation sequence.
27       return (oscillation_id(tower_a, tower_b))
28    else
29       // The threshold has not yet been met. Emit the tower as is.
30       return (tower)
31 }
```

Listing 4.1: Heuristic for determining the tower id to emit

$a \to b \to a \to b$ and `lead` $= 2$. If a and b have already reached the threshold, then the algorithm will emit $a \to c$. That is, it waits until the second tower is seen before starting to collapse the sequence.

As a further extension to the algorithm (which is not shown in Listing 4.1), when the user leaves an oscillation location, we emit a short visit (100 ms) to the last tower visited. Thus, the above sequence is actually emitted as $a \to c \to b$. The practical result is that oscillation identifiers are only ever transitioned to or away via the underlying towers. The motivation for this is the same as for

setting `lead` to at least 2. Another reason for doing this is that the last tower also provides some evidence of the user's direction of travel.

4.7.3 Analysis

This algorithm fulfills the basic requirements laid out in Section 4.4.1. First, the classifier works online. When it classifies an alternating sequence, it just relies on historical data. Second, the classifier makes prompt decisions. After it sees the first `lead` visits of an alternating sequence, it can immediately make a decision: it just needs to check whether the threshold has been reached. Finally, the classifier labels the alternating sequence consistently. Until a pair's threshold has been reached, it labels alternating sequences involving that pair as user movement sequences; once the threshold has been reached, it labels them as oscillation sequences. Depending on how large the threshold is, it can take some time before the threshold is reached. However, the classifier doesn't ping-pong: once it starts labeling alternating sequences as oscillation sequences it continues to do so.

This classifier will mislabel some user movement sequences as oscillation sequences. This is because, whatever reasonable threshold we choose, eventually some user will generate enough long user movement sequences that exceed it. As argued above, these misclassifications are unfortunate, but they are not horrible. Although the classifier will necessarily mislabel an oscillation pair's initial oscillation sequences, we expect a true oscillation pair to quickly generate a number of long oscillation sequences. As already observed, the typically oscillation half-period is about a minute long. As such, the number of mislabeled oscillation sequences should be minimal.

The only major concern is whether non-oscillation pairs always end up being classified as oscillation pairs in the long run due to the use of an absolute threshold. This seems likely given the small number of non-transient, normal towers in the training data in Section 4.5. As discussed in Section 4.4.2, this type of misclassification effectively increases location aliasing, which slows down learning and spreads out the data, but should have few negative consequences in the long run.

4.7.4 The Heuristic's Parameters

The algorithm presented above didn't provide concrete values for the parameters. In this section, we use a data driven approach to try and find reasonable values for them. We start by considering `long` and `seq count` and then turn to `lead`.

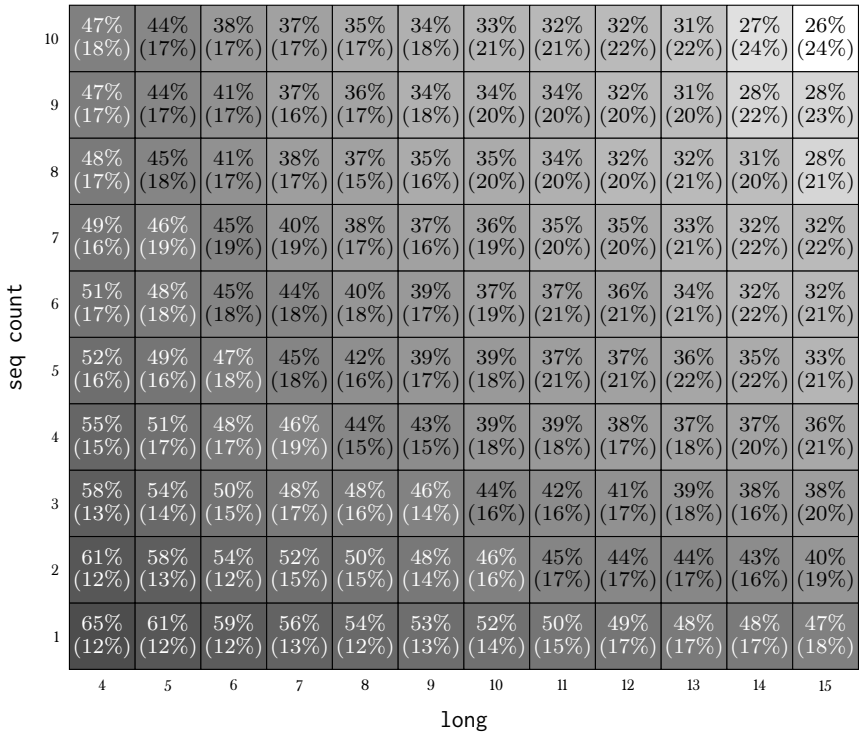


Figure 4.22: The median portion of visits in oscillation sequences that are at least 3 visits long as determined by different settings of the `long` and `seq count` parameters and the median absolute deviation, in parentheses.

`long` and `seq count`

To gain an intuition for how these parameters impact classification, we created a heat map, shown in Figure 4.22, that shows the median portion of visits classified as oscillation visits for different settings of the `long` and `seq count` parameters. The median absolute deviation is shown in parentheses. To compute this, for each user, we found all of the tower pairs that exceed the specified threshold and counted all visits to alternating sequences involving these pairs that are at least three visits long. In other words, unlike the proposed heuristic, we do an a posteriori analysis and count not only those alternating sequences that are classified as oscillation sequences after the threshold is reached, but also those that occur before the threshold is reached for those tower pairs that eventually reach the threshold.

The general trend is as expected: as we raise the threshold either by in-

creasing long or seq count, the portion of tower visits that are classified as oscillation visits decreases. The reason is straightforward: increasing the threshold means fewer tower pairs meet the threshold. Consider the extremes shown in the heat map. At the bottom left, we see that an extremely lenient and unrealistic setting of the threshold to $< 4, 1 >$ ($<$ long, seq count $>$) causes two-thirds of all tower visits to be classified as oscillation visits. At the top right, a rather conservative setting of $< 15, 10 >$ captures a quarter of the tower visits, which is still a large amount.

As we increase the value of long and seq count, the portion of visits classified as oscillation visits decreases gradually. Increasing long by 1 typically decreases the portion of oscillation visits by approximately 1 to 2 percentage points; increasing seq count by one typically decreases it by approximately 2 to 3 percentage points. Given the large portion of visits classified as oscillation visits, the magnitude of these reductions is relatively small. An exception to this rule is the bottom left corner of the plot. Here, changes in the classification rate are about twice as large for the same change in the parameters. One interpretation of this is that the parameter settings in this region capture not only oscillation sequences, but also user movement sequences. But, since there are few long user movement sequences, we eventually observe a drop. If this is the case, then selecting parameter values near the boundary is nearly optimal.

Looking at the heat map, the boundary appears to be near the diagonal line joining the points $< 4, 5 >$ and $< 8, 1 >$. Interestingly, the portion of visits classified as oscillation visits for each setting along this line (as well as along other parallel lines) is approximately constant (in this case, between 50% and 52%) except for seq length $= 1$, which is slightly larger (for $< 8, 1 >$, 54%) suggesting that these different parameter settings capture about the same set of towers and hence are approximately equivalent. The exception at the bottom suggests that some user movement does result in long alternating sequences and classifying a pair as an oscillation sequence after seeing just a single long sequence may be too aggressive.

We argued in Section 4.6.2 that alternating sequences that are at least 7 visits long are likely to be oscillation sequences and that alternating sequences that are at least 10 visits long are almost certainly oscillation sequences. This conclusion is consistent with the boundary and suggests that the parameter settings along the boundary are reasonable.

More conservative settings of the parameters result not only in fewer pairs being classified as oscillation pairs, but a longer time to recognize a pair as being an oscillation pair. Thus, a more conservative setting classifies more of the initial tower visits as user movement sequences. Figure 4.23 shows a heat map of the median portion of visits to oscillation pairs (according to the parameter

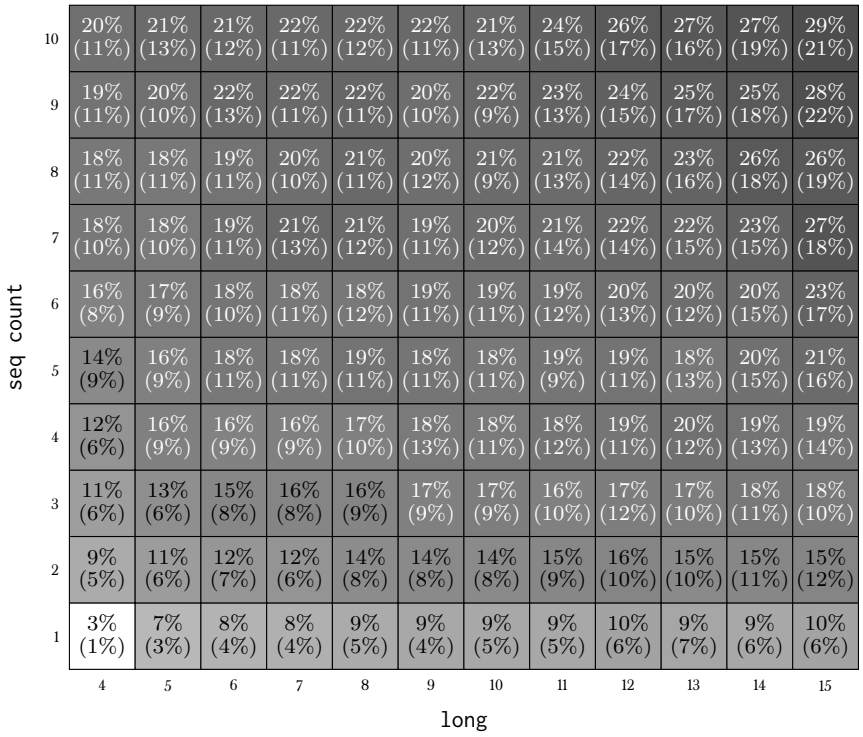


Figure 4.23: The median portion of visits that belong to oscillation pairs, but are not classified as oscillation visits because they occur before the threshold is met. The median absolute deviation is shown in parenthesis.

setting) that are not classified as oscillation visits, because they occur before the threshold is reached. Again, we just consider visits that are part of alternating sequences that are at least 3 visits long.

Looking at this graph, we see that as we increase the value of the parameters and move from the bottom left to the top right, the portion of "mislabeled" oscillation visits increases relatively slowly. The change, however, is less uniform than in the previous heat map. In particular, this time `long` has little impact for `long` $\gtrsim 6$. This may be because the first few long sequences for a given oscillation pair tend to be longer than any fixed value of `long` and thus the threshold is often reached at about the same point for similar values. We also see slightly larger changes for `seq count` $\lesssim 3$.

In light of the Figure 4.22, Figure 4.23 suggests that for each set of approximately equivalent parameters (e.g, the diagonal defined by the line passing through $< 4, 5 >$ and $< 8, 1 >$), it is probably better to choose a smaller value

for `seq count`. Smaller values of `seq count` decrease the number of inconsistent classifications (i.e., the portion of visits prior to the threshold being met). Since the portion of towers classified as oscillation visits is about the same for each parameter setting, this also means that, e.g., $< 4, 5 >$ is including pairs that don't include any sequences that are 8 visits long or longer and that we have some pairs of towers that do include a sequence that is at least 8 visits long, but don't include 5 sequences that are at least 4 visits long. Since the portion of "mislabeled" visits is small, these latter towers are probably not visited very often. Thus, we'd rather include them than the towers with a lot of sequences just meeting `long` and few sequences significantly longer.

Because we want to be conservative with respect to misclassifying oscillation sequences, based on the observations above, we should choose a value along the line formed by the points $< 4, 5 >$ and $< 8, 1 >$. Smaller values of `seq count` are better, however, as noted above, `seq count` $= 1$ is probably too lenient. Thus, we recommend $< 7, 2 >$. This is consistent with out intuition about in Section 4.6.2 that alternating sequences that are at least 7 visits long are probably oscillation sequences.

Even if this analysis proves to be wrong, we think that these parameters are a reasonable starting point. Moreover, given that the classification rates change gradually, the exact values are not critical.

`lead`

The `lead` parameter is how long (in terms of the number of tower visits) the algorithm should wait before classifying an alternating sequence. Setting this value to 1 is almost certainly too short. In this case, we don't know the alternating partner. We can reasonably guess the alternating partner if the tower in question nearly always ends up in an oscillation sequence with exactly one other tower, but as we saw in Section 3.5.2, this is rarely the case. Larger values of `lead` misclassify user movement sequences less often, however, they increase the misclassification rate of oscillation sequences, which we want to avoid. Larger values also reintroduce some of the problems outlined in Section 4.2. Thus, we conclude that the best setting of `lead` is 2 and this is the value that we use in the rest of this document.

4.8 Conclusion

Oscillation sequences are prevalent. A conservative estimate indicates that they account for at half of all tower visits. If cell towers are to be used as a proxy for

location and cell tower transitions as a proxy for movement, then it is necessary to somehow collapse them.

As discussed in chapter 3, the towers visited at a fixed location appear to be a random sampling of the towers in the area. This sampling probably reflects movement within the fixed area. Thus, depending on the context, a more aggressive approach, such as identifying and collapsing tower communities, might be reasonable.

This higher-level approach, however, necessarily loses information. Intelligently dealing with oscillations not only preserves this information, but can actually increase the resolution of the data: oscillations potentially allow us to distinguish more locations. Since it is always possible to further compress the data later, systematically dealing with oscillation sequences first maximizes our options.

Since alternating sequences can be due to oscillations or arise from user movement and we only want to collapse the former, it is necessary to classify alternating sequences. We examined three features: the distribution of visit dwell times, sequence length and whether alternating pairs are typed.

The results of our analysis were mixed. We are not able to use visit dwell time, which also means it is not possible to identify mixed mode sequences. We are only able to confidently classify long alternating sequences. And, when oscillation sequences are present, they probably dominate, which suggests that tower pairs are typed.

These results are not definitive. The problem with our analysis is that our trace data does not include ground truth (i.e., the user's actual location). To work around this, we manually label some of the tower visits as either due to oscillations or arising from user movement. Better data will yield more definitive results. Future work is to gather additional traces that include GPS tracks in addition to cell tower transitions. We didn't do this in our study, because we didn't realize that oscillation sequences would be so prevalent.

Despite these results we proposed a straightforward algorithm to deal with oscillation sequences. We found reasonable values for the parameters based on the data. Happily, these values agree with our intuition.

Intended to be blank.

Chapter 5

Cell Tower Aggregations

In chapter 3, we observed that even if a device is stationary, the modem appears to sample the towers in its vicinity rather than stay connected to a single tower. For applications that use the user's current location—whether they be programs or studies of human mobility—this phenomenon makes it inconvenient to work with the cell tower traces directly; the raw cell tower traces provide the wrong level of detail. Consider an application that predicts the user's movement, and uses cell towers as a proxy for location. Since no one tower is associated with a location, this application cannot just predict a single tower as the future location, but must predict a collection of towers. But, even if all of the predicted towers are in fact in the location's vicinity, the user is unlikely to visit them all while at the location. This problem motivates aggregating towers such that a place corresponds to a single aggregate.

5.1 Tower Sampling

In our visual analysis of the induced cell tower network in Section 3.2.3, we observed that the towers at which the user spends a fair amount of time tend to be in a relatively dense part of the network. Although this could be caused by user movement, we postulated that this more likely arises from the cell phone sampling the towers in its vicinity.

We found additional evidence for our tower sample hypothesis when examining tower transitions in Section 3.3, and tower visits in Section 3.4. For instance, in Section 3.3.2, we saw that the number of transition directions is distributed according to a power law, which means that a third to half of all transition directions are taken just once (and some towers have dozens of transition directions). We concluded that these rarely transitioned-to towers are probably far away and

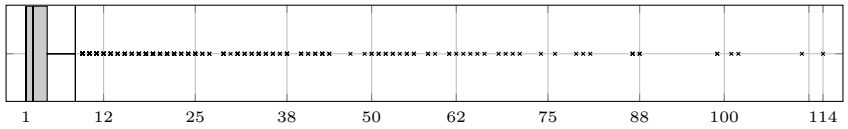


Figure 5.1: Box plot of the number of towers seen during each significant (> 30 min.) wall charge. During 275 of the 6289 charges (4.4%), at least 10 towers are seen.

only visible due to interference, but the modem finds them because it doesn't report the currently connected tower, the strongest visible tower.

In Section 3.4.2, we saw that a tenth of all tower visits are less than 10 seconds long, which we observed is probably shorter than the tower handoff threshold. Further, in Section 3.4.4, we observed that most visits to towers at which the user spends the most time are just a few minutes long, and that the few long visits are not long enough or frequent enough to account for the time that the most people sleep, which suggests that tower transitions do not necessarily imply movement.

Now, we directly test our tower sampling hypothesis by observing how many different towers are visited while the device is stationary. In our traces, we are confident that a device is stationary if it is connected to a wall charger. Although it is conceivable that a cell phone is connected to a wall charger while in, say, a train, this is probably rare.

Figure 5.1 shows the number of towers seen while a device is connected to a wall charger for at least half an hour. Although seeing one or two towers is typical (median: 2, MAD $= 1.48$), during 4.4% of the charges, the cell phone reports at least 10 different towers. Based on this, we conclude that tower sampling is a real phenomenon.

5.2 Places

Conceptually, a place is a physical location, such as home, work, or a fitness center, where a person stays for an extended period of time. Further, to allow applications to associate context with a place, a place should also capture homogeneous conditions. These conditions include environmental conditions, such as the presence of a Wi-Fi network, or a charging opportunity, as well as relevant user behavior, such as whether the user uses the device, or uses the network.

Although places are compact geographic locations, defining places by way of geographic boundaries is problematic, because different boundaries are appropriate for different people. For instance, if a person goes to a part of her work's campus that she rarely visits, this probably signals unusual behavior, and, for the

purpose of identifying the user's context, this location should be recognized as a different place even if the colleagues with whom she shares an office spend time there everyday. This individualized notion of places is reasonable as long as we don't need to compare *geographic* places across multiple users, which we don't need to do to understand a user's context. (When comparing abstract places, such as the amount of time people spend at home, or typical commute times, this notion of place is probably ideal.)

Using individualized places doesn't necessarily require that users explicitly define places. If we know where a user spends time, we just need to identify islands of significant activity. Each such island is a place.

5.3 Related work

There are various algorithms described in the literature for identifying places from raw measurement data.

Scellato et al. and Kim et al. sample the user's geographic location using GPS [86,105]. Each measurement is overlaid with a 2-D Gaussian with a parameterized radius weighted by the dwell time at that point. Scellato et al. designate the peaks in the resulting graph that exceed 0.15 of the maximum value as a place. The main problem with this technique is that GPS is energy intense, and mostly useless indoors. Another problem is the threshold. As discussed in relation to Figure 3.17, dwell time appears to be distributed according to a power law. Thus, as clearly shown in Figure 3.18, most of the probability mass is concentrated in just a handful of towers. Consequently, this technique will only recognize 2 or 3 places. Further, when there is a regime change, it will take a long time before a new place is recognized as such.

Kang et al. propose a time-based clustering algorithm based on geographic coordinates [47]. Instead of using GPS, they use the PlaceLab service to determine the user's location from the visible Wi-Fi APs [11]. This type of triangulation can also be done using GSM towers [62,133]. These techniques still require querying a database (often over the network) to convert the sensed beacons to geographic coordinates.

One way of identifying places from GSM towers is to find communities in the induced cell tower network. Community detection algorithms partition a graph into subgraphs called *communities*, such that the number of edges between the subgraphs is lower than expected [81]. As seen in, for instance, Figure 3.3, this is precisely the type of network structure that characterizes places: towers where the user spends a significant amount of time are in denser parts of the graph. Community detection algorithms are particularly interesting, because they work

without knowledge of tower locations or network topology.

Eagle et al. explore using community detection to detect places [90]. Because community detection is computationally expensive, the specific techniques that they suggest are only appropriate for offline analysis. Although streaming community detection heuristics exist (e.g., [107]), for the purpose of identifying places, community detection has a more serious problem: communities significantly overestimate a place's expanse, as they tend to not only include a place, but also the surrounding towers, which belong to routes.

Several ad-hoc strategies for identifying tighter bounds have been proposed [35, 68, 70, 135]. The common strategy is to identify a network structure that distinguishes places in the network. These heuristics are sometimes augmented by additional checks to avoid false positives.

Erbas et al. use a heuristic that requires the three strongest visible towers [35]. We were unable to evaluate this algorithm, because our trace only includes the current tower, which is a common limitation [135].

Laasonen et al. identify subgraphs that have a diameter of at most two [68]. To make sure the resulting clusters do not encompass multiple distinct locations, they also check that the average length of a visit to the cluster C, t_{avg_C}, is greater than $|C| \max_{c \in C} t_{avg_c}$, and that this does not hold for ***any*** proper subset of the cluster. After identifying the set of valid clusters, any overlapping clusters are simply merged. Unfortunately, the authors do not discuss how to name the clusters. In particular, they do not deal with how to choose a name when two existing clusters are merged. We implemented this algorithm, and found evaluating the t_{avg_C} test for a ***single*** cluster consisting of 20 vertices takes half an hour on a recent workstation. This performance profile makes this algorithm only appropriate for offline computations on small data sets.

In [135], Chon et al. check if the towers transitioned to in the preceding 10 minutes form a star in the induced cell tower graph. If so, these towers are aggregated and considered to be a place. The actual place is looked up using the tower sequence. Unfortunately, the paper does not describe how places are formed, how to look up a place given the recent towers, or how to name places. We contacted the authors asking for clarification, but did not get a response. Thus, we were unable to evaluate this algorithm on our trace.

In PlaceMap, Yadav take the induced cell tower network ***at the end of each day*** and drop all edges that aren't either traversed three times or incident on a vertex whose degree is at least three. A component in the resulting graph is merged with an existing clusters if the size of the intersection divided by the size of the smaller cluster is at least 0.55. Otherwise, a new cluster is created [70]. Yadav et al. do not describe what should happen if a component overlaps with multiple existing clusters, nor do they indicate how to name clusters. We again contacted the

authors for clarification, but received no response. In our implementation of the PlaceMap algorithm, we merge a component with each significantly overlapping cluster and use a monotonic counter, which is incremented each time a new cluster is created, to name new clusters. When determining a node's cluster, we take the largest cluster it is a part of. If there are multiple such clusters, we take the lowest-numbered cluster. In this way, a cluster's name is stable, but because a cell tower can be assigned to multiple clusters, its name can change. PlaceMap has two potential issues: because the algorithm only runs once a day, there is a delay before new data is integrated; and, due to the high thresholds, a tower may be visited many times before it is identified as belonging to a cluster.

5.4 New Aggregation Heuristics

Based on our observation that places correspond to dense subgraphs, we explored how other structures perform as indicators of a place. In particular, we considered the following: cliques that contain at least four vertices; overlapping cliques in which the primary clique consists of at least four vertices, any overlapping cliques consist of at least three vertices, and the overlapping cliques have at least two thirds of the small clique's vertices in common; cliques consisting of at least 4 vertices plus their blanket (i.e., cliques and their immediately adjacent vertices); subgraphs whose diameter is at most two (similar to [68]), and consist of at least seven vertices; and stars (similar to [135]) with at least six vertices. There are many other possible structures that could suggest places (e.g., various clique relaxations [61]), but we do not explore these here. We ran these algorithms on both the raw traces, and the traces with oscillations removed. Further, setting an edge's weight to the number of times it was traversed, we considered both any edge, and only edges whose weight was at least three (similar to PlaceMap [70]) when looking for subgraphs.

We looked for new relevant subgraphs whenever an edge first reaches the required weight. Since the structures are relatively compact, and any new subgraphs must involve the new edge, the implementation can restrict its search to just the neighborhood around the new edge. This is important given that searching for cliques, for instance, is NP-complete. This allows these algorithms to run incrementally and online, unlike PlaceMap, which runs online, but requires significantly larger chunks of data, specifically, the data from the past day.

To name a tower, we used the name of the largest subgraph that it is a part of. If there are multiple such subgraphs, then we chose the one that is oldest.

To name a subgraph, we chose the name that has been used the longest by

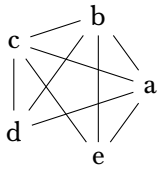


Figure 5.2: Two overlapping cliques, *a–b–c–d*, and *a–b–c–e*.

any of the constituent towers. For instance, if a subgraph consists of vertices *a* and *b* that were visited for 3 and 4 hours, respectively, then we would choose *b*. If, however, *b* had been a part of subgraph *x* for 3.5 hours and only been called *b* for half an hour, then we would instead choose *x*. In this way, a place's core cells tend to keep their name and overlapping groups share names. For instance, as shown in Figure 5.2, before we have a 5-clique, we have two 4-cliques that are one edge short of being a 5-clique. Using the above technique, these two cliques are likely to share a name.

5.5 Evaluation

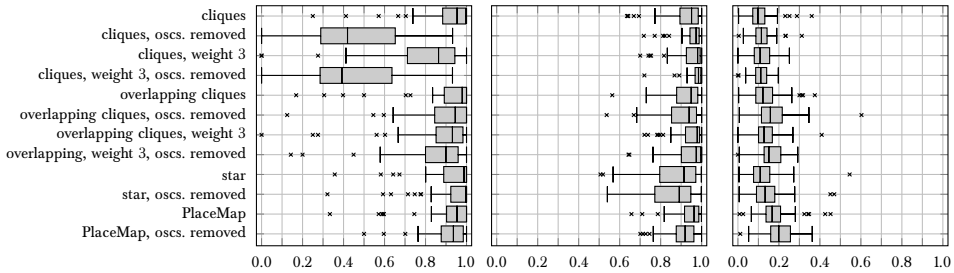
We evaluated our proposed algorithms and the PlaceMap algorithm. We ran the algorithms on both the raw traces, and on the traces after collapsing the oscillations as described in chapter 4. Further, for our algorithms, we considered both any edge, and only edges whose weight was at least three when looking for subgraphs. Neither pre-processing the data to eliminate oscillations nor excluding edges with weights smaller than three improved the results. (In order to save space, we only show the results of the most promising and the most surprising heuristics, and PlaceMap.)

We evaluate the aggregation algorithms using criteria that exemplify our notion of a place as defined in Section 5.2, and exploit the breadth of the data collected from our user study.

Our first two tests are based on the observation that when a device is connected to a wall charger, it is almost certainly stationary. In this way, we are able to establish both an upper and a lower bound on a place's real size.

For the first test (the results of which are shown in Figure 5.3 (a)), we found the portion of wall charges that occur entirely within a single place. The idea is: if the algorithm indicates that the charge occurs in multiple places, then the places are too small: the algorithm was not aggressive enough; some places should have been merged.

The second test (the results of which are shown in Figure 5.3 (b)) finds the



(a) Charges in one place (b) Cell towers covered (c) Tower rename penalty

Figure 5.3: Evaluation of different cell tower aggregation algorithms. The number of charges at a single place (a) establishes a lower bound on the appropriate aggregation size whereas the portion of an aggregation's towers visited while charging (b) establishes an upper bound. (c) shows the portion of visits prior to each renaming of a tower and indicates the algorithm's appropriateness for online use.

portion of cell towers that make up a place that are visited while the device is connected to a wall charger. If the portion is significantly less than 1, then the place is much too large: the algorithm included towers that are never visible where the device was charged. In practice, we expect there to be a few towers that are not seen at a place where the device is charged, because a place is larger than just the spot where the outlet is.

For these two tests, we don't consider all wall charges. First, we only considered charges that lasted at least half an hour. Further, we only considered charges for which at least one place was visited during three separate charges. The reason for these restrictions is that since the cell phone appears to sample the towers in its vicinity, we need a fair amount of time to observe most of the towers it can potentially see. Similarly, we also ignore places that are only visited once during any charge, and are visited for less than 10 minutes. In practice, these restrictions only removed a handful of charges and places per user.

When computing the results, we only considered the *a posteriori* place assignment, i.e., the mapping of cell ids to places at the end of the trace. This avoids a confounding variable—how fast towers are assigned to their final place—which we evaluate in our last test.

The last test (the results of which are shown in Figure 5.3 (c)) checks whether the algorithm is appropriate as an online algorithm. For each tower, we compute the number of times it was visited before each name change. Smaller is better. This construction results in a larger penalty for towers whose name changes

multiple times. For instance, if a tower's name changed after 100 visits and again after 300 visits, then its penalty would be 400. The resulting score is computed by dividing the penalty across all towers by the total number of transitions in the trace. Although this could result in a penalty larger than 1, in practice, this was not the case.

To combine the scores, we square the individual scores except for the last test, for which we square one minus the score. We square the scores based on the observation that moving from, say, 0.50 to 0.51 is significant easier than moving from 0.98 to 0.99; squaring the scores partially captures this. The combined scores are shown in Table 5.1.

In terms of its absolute performance, PlaceMap appears to perform well. However, it particularly suffers due to how long it takes to identify a tower's final place. According to this evaluation, the clique, overlapping clique, and star variants perform best. Of these, the clique technique is most balanced. The overlapping clique technique chooses places that are a bit larger, and the star technique chooses places that are even larger.

Finally, we evaluated the aggregation heuristics with respect to the inherent loss of utility created by the aggregation of multiple towers and the resulting coarsening of the resolution.

Any utility of knowing that the device is (or soon will be) at a particular place is derived from the predictive power of the place. Given our data set, facts that we know about a place include whether the device is connected to a wall charger, whether the user is interacting with the device, and whether the device is connected to a GSM or Wi-Fi network. A simple way to summarize the knowledge we have accumulated about a place is to track the respective frequency counts for each of these facts.

If a heuristic is used to aggregate a set of towers to represent a place, we can quantify the loss of resolution by comparing the entropy of these probability distributions before and after the aggregation. Specifically, let $e \in E$ be the set of possible facts, and $P_e(a)$ be the probability of fact e holding in aggregate a. For a partitioning A of the towers into aggregates $a \in A$, and $l(a)$ data points (or, total time spent) asserting e in aggregate a, we define the summarization loss $S(A)$ as:

$$S(A) := -\sum_{a \in A} l(a) \sum_{e \in E} P_e(a) \log_2 P_e(a) \tag{5.1}$$

By construction, $S(A)$ becomes larger when towers with non-equivalent probability distributions are aggregated.

$S(A)$ is a sensible metric, as it measures how well aggregates align with our definition of a place: smaller $S(A)$ values imply that the aggregate truly com-

Algorithm (abbreviation)	One Place	Towers Covered	Rename Penalty	Score (Rank)
cliques (c)	0.95 (0.068)	0.95 (0.051)	**0.90** (0.041)	**2.63** (2)
cliques, oscs. removed (co)	0.42 (0.22)	0.97 (0.026)	0.89 (0.041)	1.91 (12)
cliques, weight 3 (c3)	0.86 (0.14)	**0.98** (0.025)	**0.89** (0.053)	2.50 (7)
cliques, weight 3, oscs. removed (co3)	0.39 (0.23)	**0.98** (0.020)	0.89 (0.041)	1.91 (11)
overlapping cliques (o)	**0.98** (0.031)	0.95 (0.055)	0.88 (0.065)	**2.63** (1)
overlapping cliques, oscs. removed (oo)	0.94 (0.082)	0.94 (0.071)	0.84 (0.069)	2.48 (8)
overlapping cliques, weight 3 (o3)	0.93 (0.086)	**0.98** (0.024)	0.87 (0.052)	2.59 (4)
overlapping, weight 3, oscs. removed (oo3)	0.90 (0.11)	0.97 (0.034)	0.85 (0.054)	2.48 (9)
star (s)	**0.99** (0.018)	0.91 (0.10)	**0.89** (0.066)	**2.61** (3)
star, oscs. removed (so)	**1.0** (0.0068)	0.89 (0.12)	0.87 (0.064)	2.54 (5)
PlaceMap (p)	0.95 (0.069)	0.96 (0.043)	0.83 (0.057)	2.53 (6)
PlaceMap, oscs. removed (po)	0.94 (0.076)	0.92 (0.064)	0.80 (0.081)	2.36 (10)

Table 5.1: The algorithms' scores (median and MAD), and their combined scores (the sum of the squared individual scores).

(a) Wall charges. (b) User activity. (c) GSM data connection. (d) Wi-Fi connection.

Figure 5.4: Information loss (in bits) vs. number of aggregates by aggregation heuristic. (See Table 5.1 for abbreviations used.) Using our oscillation detection heuristic without any aggregation heuristic ("0") increases the total number of places and thus provides a higher resolution picture (top left data point). The other extreme (bottom right data point) is a trivial aggregation heuristic that puts all towers into a single aggregate ("1").

bines towers with similar user behavior and similar environmental conditions. Figure 5.4 relates the $S(A)$ scores of some of the aggregation heuristics with the total number of aggregates $|A|$. The summarization losses are plotted relative to $S(r)$, where r is the original raw tower trace. To ensure that the probability distributions are meaningful and the number of aggregations is not artificially low due to merging of insignificant towers, we only consider towers at which the user spent at least 1 hour.

For random combinations, we expect to see a linear relationship between the number of aggregates and the loss of information. All of the examined algorithms perform better than that. Since the aggregation heuristics did not use the metrics in anyway when they aggregated towers, future heuristics can make even more useful predictions by taking the utility metrics into consideration. Interestingly, our heuristic to collapse oscillations increased the amount of information relative to the raw trace confirming that oscillations contain useful data.

5.6 Conclusion

Motivated by the fact that places are covered by multiple cell towers, and our desire to use cell towers to identify places, we developed a new family of algorithms to aggregate towers to identify places. Using PlaceMap as a baseline, we found that our algorithms, specifically, the clique-based variant, perform a bit better in terms of determining an appropriate boundary, and significantly better in terms of how quickly a tower is assigned to its final place. Further, we have introduced several new metrics that are helpful in evaluating cell tower aggregation techniques. These metrics should be useful for future work in this area.

Intended to be blank.

Chapter 6

Predicting Location

In this chapter, we present a family of algorithms for predicting a user's approximate location in the near future. The algorithms are all probability based and only use the user's own history of movements. Concretely, they all build and evaluate a sampling distribution of the form $f(t, \mathbf{c})$, where t is a tower aggregate, and $\mathbf{c}$ is one or more conditions.

To evaluate the algorithms, we investigate the performance of $\operatorname{argmax}_t P(t \mid \mathbf{c})$,[1] i.e., the most likely tower aggregate given a set of conditions. For some applications, returning the n most likely tower aggregates may be more appropriate. For others, returning the probability assigned to all known locations may be more useful. We chose the most likely tower aggregate, because it simplifies the evaluation methodology, and the conclusions should generalize to the other cases.

We start by presenting our evaluation methodology. Then, using two trivial predictors, we establish a baseline. After that, we present four basic features, and we evaluate their individual performance as predictors of a person's future location. Because human behavior changes with time, we then add aging in an attempt to better capture this dynamic component. Subsequently, we explore one way to combine the simple predictors to improve the overall performance. Finally, we present future directions and related work.

6.1 Related Work

Most work on location prediction on mobile devices has focused on identifying the next access point (typically, the next cellular tower, but also the next Wi-Fi

[1] We abbreviate $P(t \mid \mathbf{c})$ to $\langle \mathbf{c} \rangle$.

AP) that the user will visit. This work is motivated by improving network resource allocation. In short, in order to ensure a smooth handoff, it is necessary to reserve resources at the next access point. Being able to predict the user's next access point means not having to reserve resources at all neighboring access points, but only a small number of likely neighbors. This work generally does not consider when the user will transition to the next access point.

François et al. use a hidden Markov model to predict the next access point the user will transition to in a wireless network and when that will occur [60]. Song et al. examine the use of Markov and LZ-based predictors to predict the user's next Wi-Fi AP (they don't try to predict when the transition will occur) [73]. Eagle et al. develop dynamic Bayesian networks for predicting the next tower aggregation that a user will visit, and how long the user will stay at a given tower aggregation. The authors include a hidden "abnormality" variable in their model, which helps them detect when a user deviates from her usual routine [90].

Another approach leverages collective behavior to predict related users' location in the near future. This is taken by Xiong et al. [44] and Zhang et al. [25]. Costa et al. explore how to identify similar trajectories with their SmartTrace algorithm [24]. A fundamental issue with this type of work is that it relies on providing sensitive information to a third party. In contrast, using our approach, none of the users' data needs to leave the device.

NextPlace uses non-linear time series to predict a person's location in the near future [105]. This work is the closest to ours and we present a detailed overview and a comparison to our approach in Section 6.10. Laasonen uses the current trajectory to predict the user's route [67]. This is approach is less useful for prefetching. Burbey and Martin consider predicting location based on the current time, and its complement, when someone will next visit a location [52]. Our methodology extends theirs, and includes an evaluation on a much larger data set. Etter et al. investigate a complementary problem to ours: predicting a user's next destination [120]. They use graphical models, neural networks and decision trees.

Chon et al. present a Markov-based model to predict the amount of time that a user will stay at a given location, but don't predict locations [134].

Instead of predicting a user's location, Sohn et al. examine how to predict whether a user is walking, driving or stationary using their cell tower traces [116].

6.2 Evaluation Methodology

To evaluate the algorithms presented in this chapter, we again use the 59 traces that have at least 14 days worth of cell tower data.

Before allowing the algorithms to observe any of the data, we first aggregate the towers using the online version of the clique-based algorithm described in Section 5.4. We deemed the clique-based aggregation algorithm to be the best of those that we evaluated partially because it did not too aggressively collapse towers, and because it was able to quickly assign towers to aggregates.

We also used the aggregated version of the trace to determine the ground truth. A consequence of this is that a more aggressive tower aggregation algorithm will appear to result in better performance. The aggregations, however, will be less useful in identifying the user's context, because individual tower aggregates will encompass more varied user behavior and a less homogeneous environment.

To evaluate the performance of the prediction algorithms, we use the following methodology. We start by partitioning each trace into half-hour segments starting on the full and the half hour. For each segment, we first allow the algorithm under consideration to observe any data up to the end of the current segment. Then, we have the algorithm make a series of predictions: a prediction 30 minutes in the future, 1.5 hours in the future, etc., up to 23.5 hours in the future.

To determine whether a prediction is correct, we check whether the user visited that tower aggregate within 15 minutes of the prediction time. That is, if the time at the end of the segment is 8:30am, a prediction 1.5 hours in the future is correct if the predicted tower aggregate is actually visited between 9:45am and 10:15am. Using a window rather than the exact time provides a small amount of smoothing, which should elide occasional oscillations (recall from, e.g., Section 3.2.3, that phones appear to sample the towers in their vicinity). If we have no data, e.g., because the device was off, then we don't consider the prediction when scoring the result.

We also considered determining that a prediction was correct by checking whether the prediction matched the dominant tower (i.e., the tower aggregate at which the user spent the most time in the window ± 15 minutes around the prediction time). Using this approach, the results were similar, but the median precision was up to 2 percentage points worse. The similarity of the results is reassuring: this confirms that there is really just a single dominant tower aggregate at a given location. In the end, we chose not to the use this method to make comparing our results to other work, in particular, NextPlace [105], easier.

To score the results, we use two metrics: the portion of correct prediction *attempts*, and the portion of the correct prediction *trials*, which are approximately precision and accuracy, respectively, in information retrieval terms. (These two metrics are related by the portion of prediction attempts.) An algorithm may not attempt a prediction trial if it doesn't have enough data. Unless otherwise stated,

all of the algorithms require that the observed data include at least 2 hours worth of data for the relevant condition before it attempts a prediction trial. For example, when conditioning on the hour of the day (i.e., $\langle h \rangle$), the predictor will only make a prediction for 9:30am after it has observed 2 hours worth of data between 9:00am and 10:00am, which takes at least 2 days. By reporting both the precision and the correct trials, we are able to distinguish algorithms that don't make predictions all of the time, but when they do are highly accurate, from those that always guess, but make mediocre predictions and, hence, are often wrong.

When computing the score, we allow the algorithms a week of training time. That is, we don't count predictions made between the start of the trace and exactly 7 days later. A week of training time is modest, but not insignificant. In practice, we want to make predictions as soon as we have enough data, however, to avoid confounding variables we prefer a little too much training data to not enough in our evaluation. We examine the issue of the appropriate amount of training time in Section 6.5.

6.3 Baseline

To establish a baseline, we consider two trivial prediction algorithms. The first algorithm simply returns the tower aggregate t that the user spent the most time at since the beginning of the trace. That is, it maintains the unconditional probability distribution $f(t)$ and makes predictions by solving $\operatorname{argmax}_t P(t)$, i.e., $\langle \text{true} \rangle$. For most people, the predicted tower aggregate will probably correspond to their home. The second prediction algorithm simply returns the current tower aggregate (i.e., $\langle c \rangle$, where c is the curren tower), and is based on the observation that people don't move around that much.

Figure 6.1 shows the performance of the unconditional prediction algorithm. The figure consists of a series of box plots. Each box plot shows the performance across the users for a given hour of the day. The box plots split the predictions according to the prediction time and not when the predictions were made. That is, if a prediction were made 1.5 hours in the future at 8:30am, the result would appear under the 10am box plot. Likewise, if a prediction were made 2.5 hours in the future at 7:30am, the result would also appear under the 10am box plot. The box plots show the portion of correct prediction ***attempts*** (i.e., the precision) and not the portion of correct prediction trials. (The portion of attempted trials is shown above the plot along with the median portion of correct prediction ***attempts*** and its MAD.)

The unconditional predictor performs surprisingly well. It is correct 61.3%

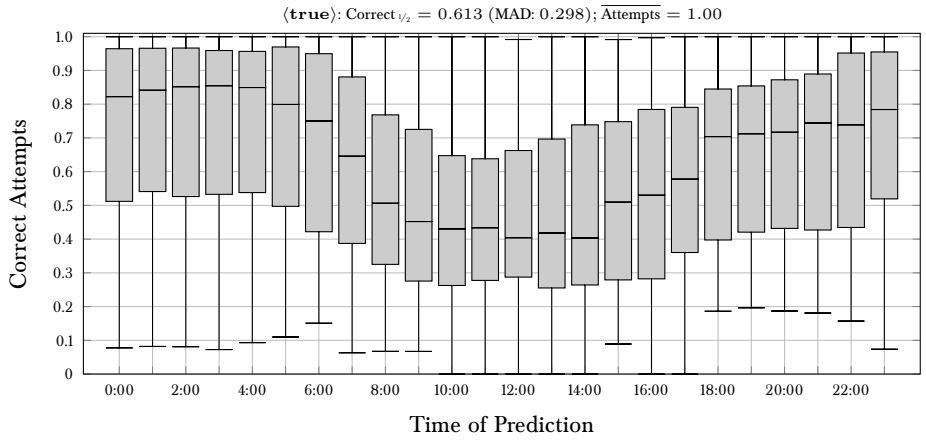


Figure 6.1: Box plots of the portion of correct prediction attempts of the predictor $\langle \mathbf{true} \rangle$ for different prediction times. The x-axis is the time of the prediction. Thus, predictions made 1.5 hours in the future at 8:30am are included in the 10am box plot as are predictions made 2.5 hours in the future at 7:30am.

of the time (MAD: 29.8%) and attempts all trials. The box plots reveal that the predictor does particularly well at predicting the user's location at night and performs worst at predicting the user's location during the day. This is consistent with someone whose dominant location is her home, who sleeps at home at night, and who is often, but not always away during the day, e.g., at work or at school during the week, and at home on the week end.

A possible explanation for why this predictor performs so badly for a few users is that these users may have moved. Consider a user who lived somewhere during the first 3 months of the trace and spent about 60% of her time at home. If the user moves to a new location and she spends about 60% of her time at her new home, then it will take another 3 months before the new home is the dominant tower at which point the predictor's precision will have dipped to about 30%. This suggests that a mechanism to age data is required. We examine this issue in Section 6.6.

Figure 6.2 shows the performance of the current-tower aggregate predictor. Unlike the previous plot, this plot breaks down the prediction performance by the prediction offset rather than by the time of day. For the unconditional probability, the results are mostly independent of the prediction offset whereas the reverse is true for the current-tower predictor.

Unsurprisingly, the predictor performs best for small prediction offsets, and attains 79.6% median precision for predictions 2.5 hours in the future. As the

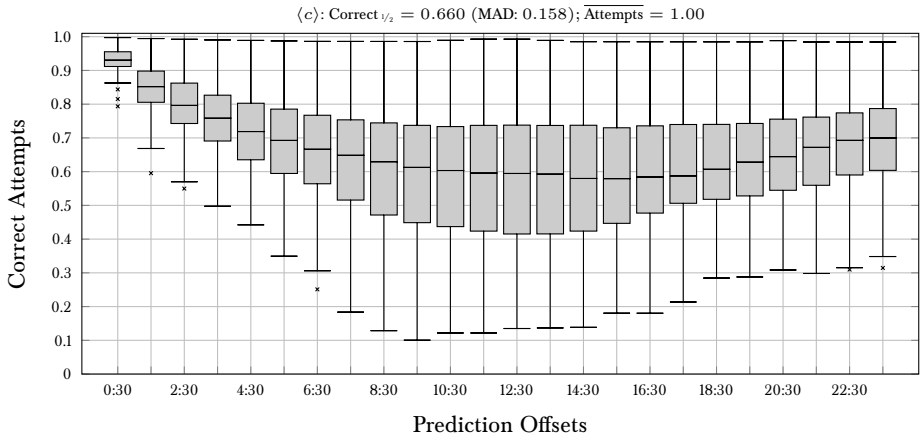


Figure 6.2: Box plots of the portion of correct prediction attempts of the current tower predictor ($\langle c \rangle$) for different prediction times. The x-axis is the time of the prediction (not the prediction offset).

prediction offset increases, the precision goes down steadily, but the median precision never falls below 50%. For instance, for predictions 12.5 hours in the future, the precision drops to 59.5%. This provides strong evidence that people usually stay at the same location for extended periods of time. Starting with predictions approximately 17 hours in the future, the precision starts to rise again. This is consistent with a daily routine.

6.4 Features

We've identified four features that appear to be useful for predicting the user's location: the time of day, the day of the week, the current regime, and the current tower aggregate. In this section, we examine and evaluate each feature.

6.4.1 Time of Day

Intuitively, we expect most people to have daily routines. In the very least, most people usually sleep at the same time and in the same bed, and they are at work or at school at approximately the same time most days of the week. In fact, daily routines are so common, that we saw evidence of them when examining the current-tower-aggregate predictor in the last section—the user will often visit the same tower aggregate approximately 24 hours in the future. This provides a strong motivation for making predictions based on the time of day.

The main issue in designing a predictor that uses the time of day is determining an appropriate granularity for segmenting the day. Using a too fine granularity unnecessarily spreads out the data. For instance, a predictor that conditions on the second of the day only considers $1/86400^{\text{th}}$ of the available data when making a prediction. A small segment size also makes this predictor more sensitive to noise. Random oscillations, which would have been smoothed out by using a larger period, may now appear significant. On the other hand, using a too coarse granularity can result in the predictor not recognizing important events. For instance, using three-hour segments can result in the predictor ignoring the user's hour-long lunch break.

In our examination of the user traces in Section 3.2.2, we observed that people usually start or end their primary daily activity, e.g., work or school, within a two hour window. We also intuitively expect most activities to last at least an hour to justify their overhead—the commute time, the time to get started, etc. This matches Choujaa and Dulay observation in their analysis of the Reality Mining data set that users usually spend several consecutive half-hours on the same activity [27]. Consequently, we suspect that dividing the day into half-hour or hour segments provides the best trade off.

Figure 6.3 shows the results of conditioning on the time of day using both half-hour long segments and hour-long segments. Both plots have the same basic shape as the plot for the unconditional predictor (Figure 6.1): the predictors are all better at predicting the user's location at night; during the day their performance decreases until a minimum around midday.

Conditioning on the hour of the day increases the median prediction precision by 9.1 percentage points relative to the unconditional predictor. The predictor that uses the half-hour segment size has a nearly identical prediction precision (the difference is just 0.0 percentage points) and both attempt nearly all trials. Given this inconsequential difference, we prefer the predictor that uses fewer states, i.e., the hour-based predictor, and we use it in the rest of this chapter.

6.4.2 Day of Week

Conditioning on the time of day assumes that people have the same routine every day. In practice, however, this is not often the case. At a coarse granularity, we expect most people to have at least one or two days of rest per week. At a finer granularity, free-time activities often occur on a weekly basis. We saw some evidence of this when examining the cell phone traces in Section 3.2.2.

A simple approach to distinguishing daily routines is to assume hebdomadal routines and, correspondingly, condition on the day of the week. By itself, how-

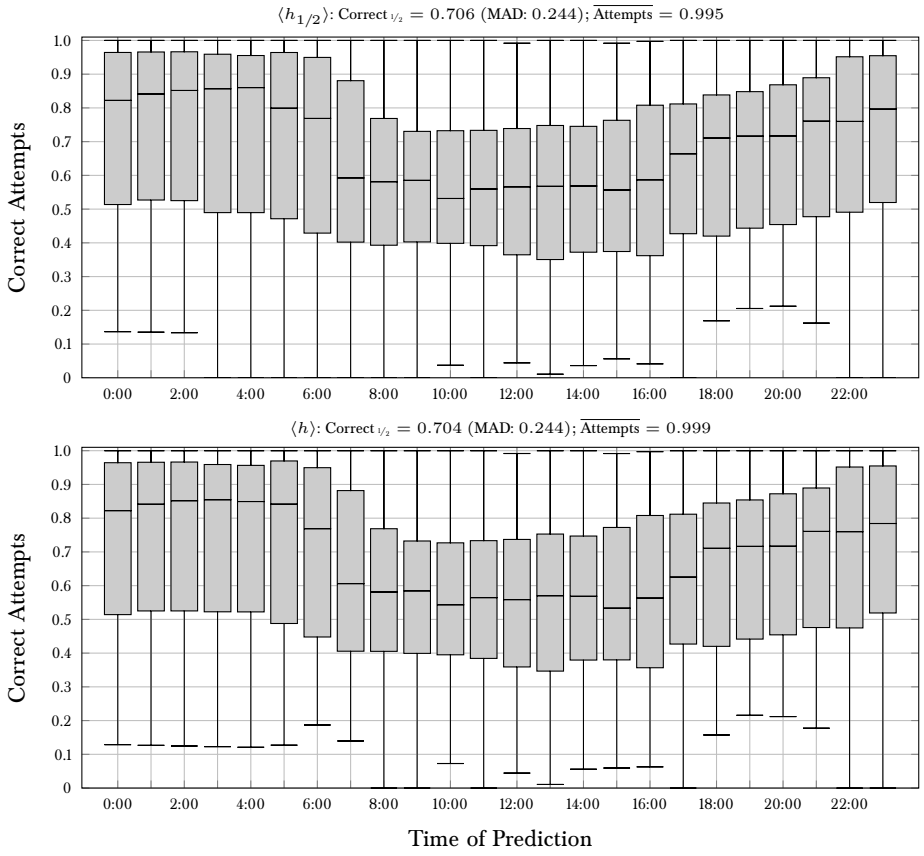


Figure 6.3: Box plots of the portion of correct prediction attempts of the $\langle h_{1/2} \rangle$ and $\langle h \rangle$ predictors. The x-axis is the time of the prediction.

ever, conditioning on the day of the week is not terribly helpful. The point of conditioning on the day of the week is to tease out nuances in a user's day-to-day routine; when considering a day as a whole, where the person sleeps will typically dominate and hence be that day's prediction, which is the same thing that the unconditional predictor does. This can be seen in Figure 6.4. In fact, the predictor $\langle d \rangle$ actually performs slightly worse than the unconditional predictor (Figure 6.1): its median prediction precision is 4.4 percentage points lower.

Accordingly, we also need to condition on the time of day for conditioning on the day of the week to be useful. Conditioning on multiple variables, however, can result in a state space explosion, which occurs when the available data is so thinly spread across the states, that the prediction algorithm rarely has enough

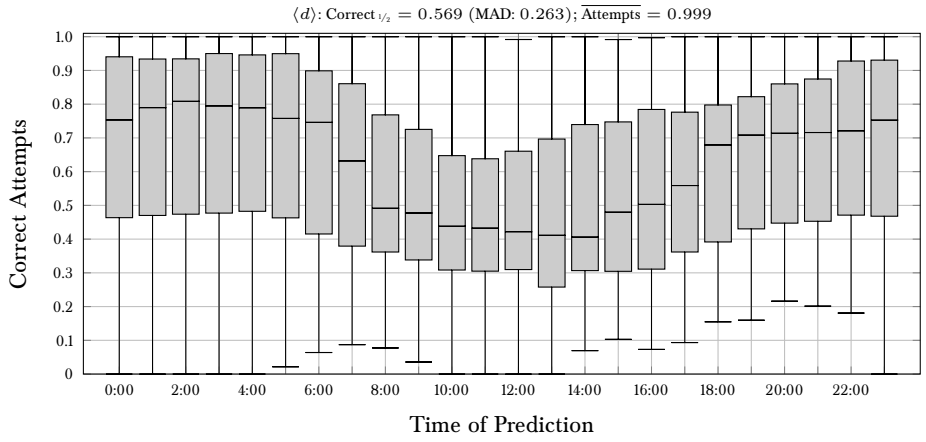


Figure 6.4: Box plots of the portion of correct prediction attempts of the predictor $\langle d \rangle$. The x-axis is the time of the prediction.

data to make a confident prediction. Conditioning on both the hour of the day and the day of the work, for instance, results in $24 \cdot 7 = 168$ different states. Although 168 states is not huge, it is large enough that each state is only sampled for one contiguous hour each week. Assuming we require that a state have at least 2 hours worth of data before we are confident in its predictions, we would only be able to make prediction starting at the third week of the trace! Because the prediction algorithm is intended to run online and only use locally obtained data, this significantly impacts the utility of this predictor.

One way to reduce the state space is to collapse similar states. For instance, we can condition on whether the day is a workday or a day of rest. The tradeoff, in this case, is that our day of week predictor won't be able to recognize weekly activities.

Conditioning on whether the current day is a workday is not as simple as classifying Monday through Friday as workdays and Saturday and Sunday as days of rest. First, although many people have a schedule consistent with this classification, people who work in retail, for instance, often work on the weekend. Second, different countries have different workweeks. In particular, many Muslim countries have a Sunday-through-Thursday workweek. Most of the rest, such as Iran still use the more traditional Saturday-through-Wednesday workweek. (The others changed over the past decade to facilitate international cooperation.) Brunei Darussalam has a non-contiguous workweek; there, Friday and Sunday are the standard days of rest [131]. Third, in their study of call data records (CDRs) of 5% of all cell phone numbers registered in the New York and Los

Angeles areas, Isaacman et al. found that Fridays are more similar to weekend days than to workdays in terms of how much people travel [108]. These observations suggest that better prediction performance may be obtained by learning a person's workdays from the data and not assuming them a priori. Nevertheless, per-country priors are probably reasonable.

Since a quarter of our traces include data from countries with different workweeks from the rest, we tried aggregating on country-specific workdays. To our surprise, this did not result in a statistically significant difference from just assuming that the workweek is Monday through Friday. According to an Iranian associate, it is not unusual for companies in Iran to have a more Western-style workweek. Thus, it might be that *some* of our users have non-traditional workweeks for their country. More data is required to understand what is really going on, however, it is likely more useful to invest resources in studying how to automatically detect a person's workdays.

Based on this result, we just use a Monday-through-Friday workweek in the rest of this chapter. (However, for comparison purposes, we include the results of using both western and local workdays in the next figure.) We leave learning the users' personal workdays from their traces to future work.

Figure 6.5 shows the results for the hour-of-day and day-of-week predictor, $\langle h, d \rangle$, and the hour-of-day and workday predictor, $\langle h, w \rangle$. As can be seen from the large lower quartile and the whisker extending all the way to 0, the day-of-the-week-based predictor performs very poorly on a few users. A close look at the data reveals that this poor performance is on those traces with the least amount of data. This is due to the large state space, which requires two weeks worth of data before any state has enough data to make a prediction. The workday-based predictor also shows some signs of this problem, but to a smaller degree.

Figure 6.6 shows how well these predictors can perform when given enough data. These plots just include the traces that have at least 16 weeks worth of data. Although the median prediction precision of the workday-based predictor stays the same, its MAD goes from 21.1% to 16.6%. In contrast, the day-of-week-based predictor's performance increases dramatically. The median prediction precision goes from 66.7% to 74.1%, its MAD goes from 29.4% to 16.3%), and it goes from attempting just 70.5% of the prediction trials, on average, to 95.8%. Nevertheless, its median prediction precision does not significantly exceed that of the workday-based predictor (74.1% vs. 72.1%), but it does have significantly better worse performance, which can be seen by its much smaller whiskers for predictions made during the day.

Relative to the plain time-of-day predictor, the workday-based predictor performs slightly better. Its median prediction precision on all traces increases from

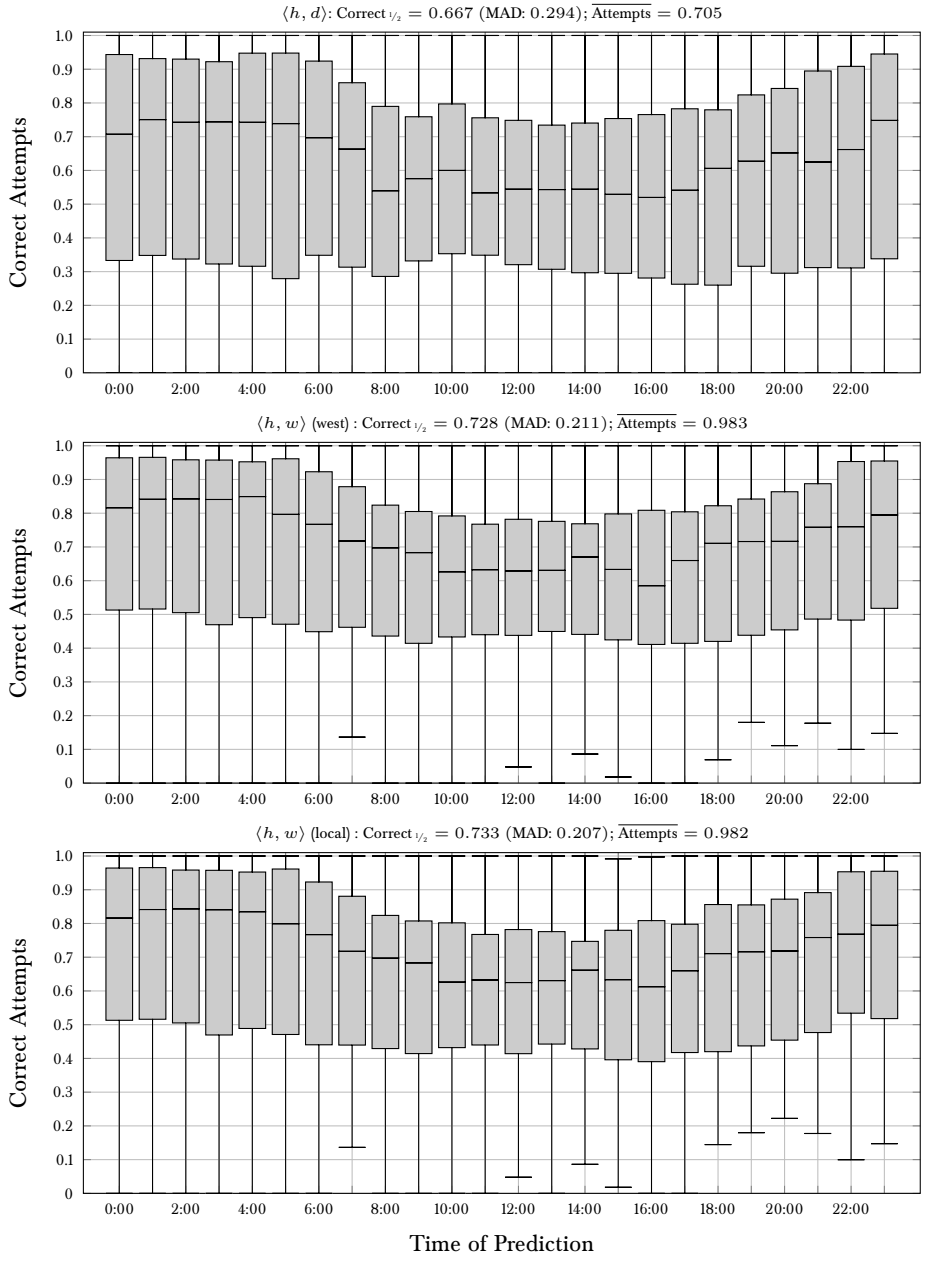


Figure 6.5: Box plots of the portion of correct prediction attempts of the predictors $\langle h, d\rangle$ and $\langle h, w\rangle$ for both western workweeks and local workweeks for different prediction times. The x-axis is the time of the prediction.

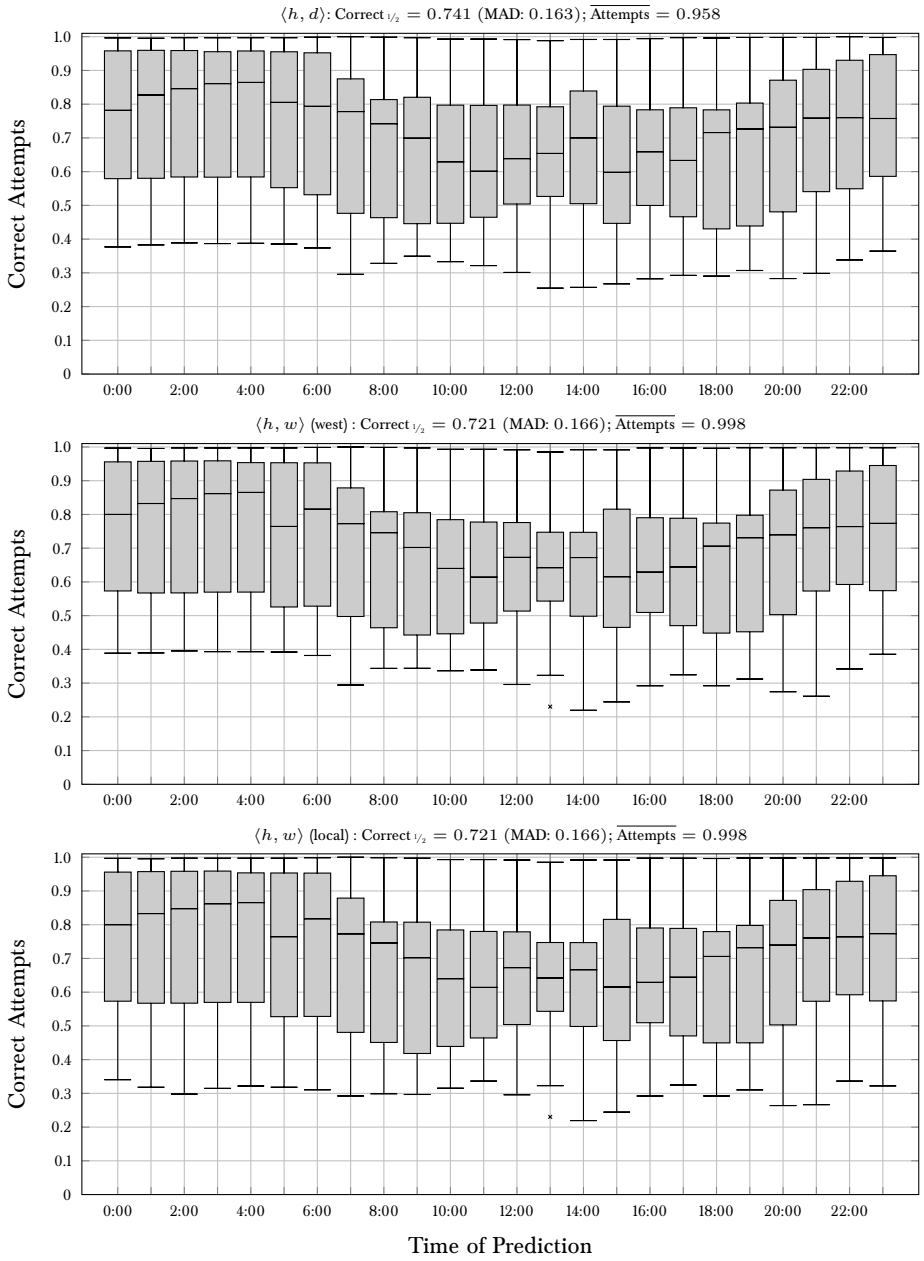


Figure 6.6: Box plots of the portion of correct prediction attempts of the predictors $\langle h, d\rangle$ and $\langle h, w\rangle$ for both western workweeks and local workweeks for different prediction times, and only considering traces with at least 16 weeks work of data. The x-axis is the time of the prediction.

70.4% to 72.8%, and its MAD decreases from 24.4% to 21.1%. The mean attempts, however, decreases slightly (99.9% vs. 98.3%).

6.4.3 Regime

In Section 3.2.1, we observed that most traces include so-called *regime changes*, extended periods of time during which a user visits a completely new set of towers. Regimes are, thus, areas in which a user moves on a multi-day timescale, and a regime change occurs when a user moves to a new geographic area. As such, conditioning on a user's regime should improve the prediction results.

A user's primary regime includes her home, her workplace, those places at which she participates in regular activities or sports, stores at which she goes shopping, etc., as well as the routes that she takes between those places. Although the primary regime will usually dominate in terms of total amount of time the user spends there, we expect most people to have multiple regimes, which they visit repeatedly. For instance, many people have relatives or friends who live far away and whom they occasionally visit for a few days at a time. Some people travel to regular events, e.g., a festival, such as Burning Man. And, some people travel for work now and again, e.g., to a conference or to another work site. Whereas a conference's venue may change each year, office moves are much less frequent. In both cases, it may be useful to identify these as separate regimes. In the former case, we identify new behavior (the user is not at home and is unlike to go to work in the morning or to the gym in the afternoon), and, in the latter case, we can use the user's past behavior at that location to help make predictions about how the user will behave next time.

Identifying Regimes

Since a regime is the area in which a user moves on a multi-day timescale, most people sleep at the same location each night, and the location at which most people spend the most time is where they sleep, a simple way **(0)** to identify the current regime in an online fashion is to use the tower aggregate at which the user spent the most time over the past 24 hours. For some people, the tower aggregate at which they spend the most time is where they work and not where they sleep. As long as the identifier is relatively stable, exactly which aggregate is used to represent a given regime is not important. However, we would like to avoid having multiple aliases for a single regime. This splits the data and, consequently, increases the learning time. A shortcoming of this simple scheme for identifying regimes is that it will take a day to adapt to regime changes.

We can adapt more quickly by **(1)** using a tower aggregate's typical regime,

which we can find by solving $\text{argmax}_r P(r \mid t)$, where r is the regime, and t is the current tower. (To build the conditional probability table, we set the regime to the dominant tower in the past 24 hours, i.e., we use (0), and weigh it according to the dwell time at the tower aggregate.) Thus, even if the user spent the previous night at a friend's home, which then appears to be a new regime the next morning, once she arrives at work, this algorithm ensures that further predictions will be made in the context of her primary regime.

Because we occasionally see new towers at regularly visited places due to how phones appear to sample towers in their vicinity (recall the discussion in Section 3.2.3), and to reduce the impact of aliasing, we mix the conditional probability tables of all of the tower aggregates visited over the past 15 minutes weighted by their dwell time after giving the current tower aggregate a five minute bonus.

Unfortunately, this approach to identifying regimes only reliably captures regime changes after they occur, not as they occur. Consider someone who occasionally visits her parents for the weekend. The towers along the route will be associated equally strongly with both her primary regime and her parent's regime. As she is traveling to her parents' home, the regime in the near future will be the area around her parents' home and any predictions should be made in this context. Likewise, on her way home, predictions should be made in the context of her primary regime, not her parents' regime.

To predict the regime in the near future, **(2)** we can solve $\text{argmax}_{r_{new}} P(r_{new} \mid r_{current}, t)$, where r_{new} is the regime in the near future, $r_{current}$ is the current regime as per (0), and t is the current tower aggregate. This predictor should correctly distinguish the above two cases: when we observe the user along the route between her home and her parents' home, and the current regime is her primary regime, then she is traveling to her parent's home and we should predict that the regime in the near future will be her parents' home; when she is along the same route, but her current regime is the one around her parents' home, then she is traveling home and we should predict that the regime in the near future will be her primary regime. Again, we don't just use the currently connected tower aggregate, but all tower aggregates from the past 15 minutes.

When using (2), if insufficient data is available to make a prediction, then we fall back to (1). This can happen, because (2) has to wait, on average, half a day to integrate data into its conditional probability table; before that, r_{new} is not known. In contrast, the primary regime predictor can update its conditional probability table immediately since it only relies on past data.

If even (1) is unable to make a prediction, then we fall back to (0), i.e., the dominant tower in the past 24 hours, but only if the user spent at least two hours there and there is not a two hour gap in the trace in the recent past. The former

condition ensures that we only use significant locations to identify regimes. The latter constraint ensures that if the user turns the device off for a flight, we don't assume the user is still at her previous regime. Otherwise we assume the user is at a new regime.

We also consider three minor variations of algorithm (2)). Algorithm **(3)** identifies the current regime using the tower aggregate with the longest visit between midnight and 6am. The idea is to avoid an aliasing problem in which sometimes the user's home is used to identify the regime, and sometimes the user's workplace is used. This algorithm assumes that the user sleeps at night. However, at least in the US, approximately 1.9% of the population *leave* work between 5:30am and 7:30am [111]. Hence, this stronger assumption may result in better regime assignments for most, but it is less robust for a few. A similar approach to this was used by Google Latitude to identify a user's home location. Their approach to dealing with this error is to provide an option for users to manually override their detected home location [10].

Algorithm **(4)** considers the dominant visit over the past 24 hours for identifying the current regime. This is another attempt to use the place where the user sleeps to represent the regime even if the user doesn't always spend the most time there on a given day. This tweak is based on the observation that when a person is sleeping, he normally doesn't move, but when he is at work, he may connect to different towers when he goes to lunch, to a meeting, etc. Thus, even if the total time at a given tower aggregate at a person's workplace dominates, because the user moves, this is spread across many visits. In contrast, most people don't walk around when sleeping. Of course, the phone still changes due to tower sampling.

Finally, algorithm **(5)** attempts to better account for tower sampling effects, by squaring the total time of each tower visit, and then finding the dominant tower. By squaring the dwell time, we give significantly more importance to long tower visits, which we again hypothesize occur primarily while the user is sleeping.

When determining the dominant visit in algorithms (3) and (4) (as opposed to the dominant tower), we perform some basic smoothing. Specifically, if the user moves away from the current tower aggregate and returns within 5 minutes, then we merge the two visits. This is motivated by our tower sampling observation (see Section 3.4.2).

Evaluation

Table 6.1 shows the prediction performance of the predictors $\langle r \rangle$, $\langle r, h \rangle$, $\langle r, h, d \rangle$ and $\langle r, h, w \rangle$ for each of the regime labeling algorithms. Figure 6.7 shows plots

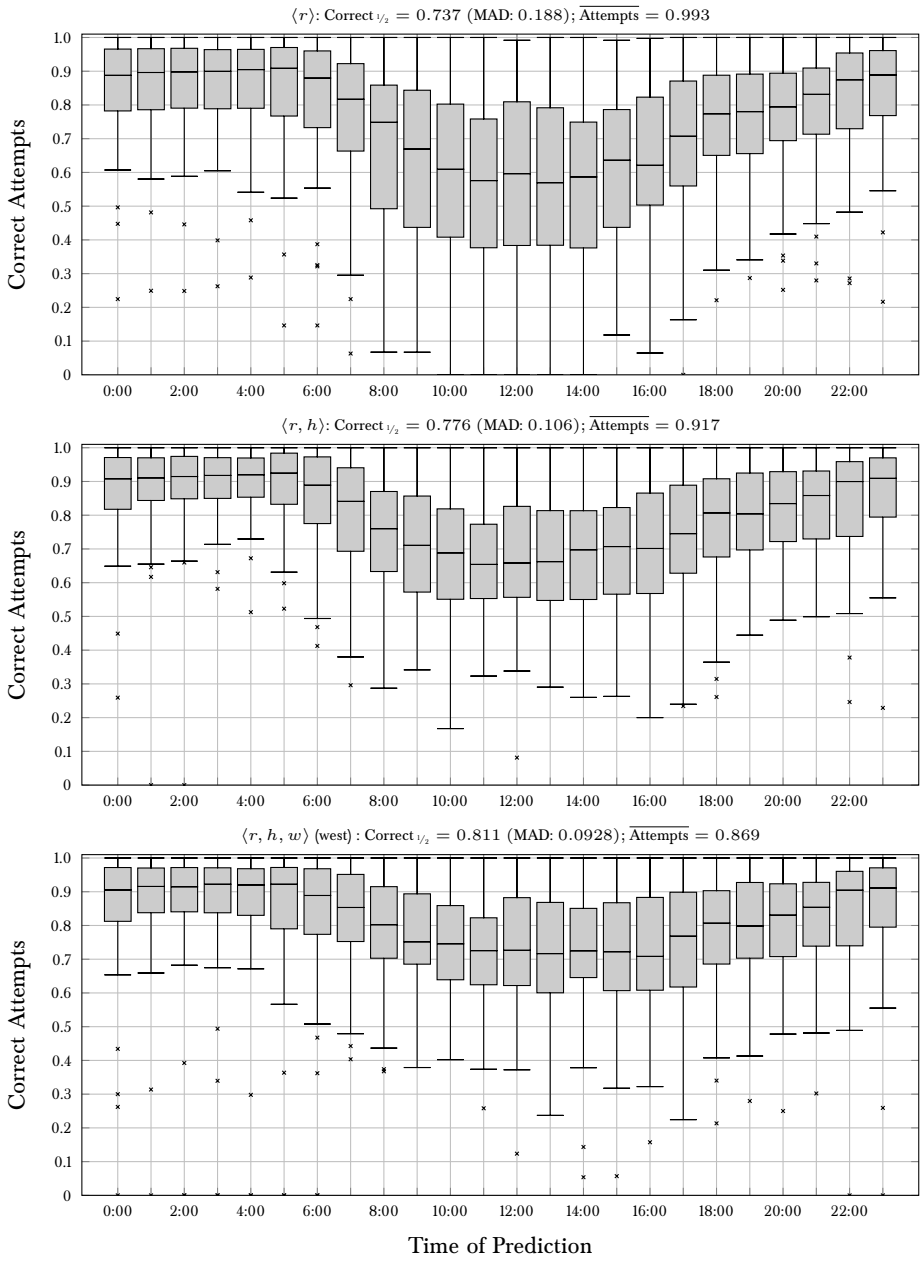


Figure 6.7: Box plots of the portion of correct prediction attempts of the predictors $\langle r \rangle$, $\langle r, h \rangle$, and $\langle r, h, w \rangle$ using algorithm **(1)** to identify regimes for different prediction times. The x-axis is the time of the prediction.

		Correct Attempts		Attempts		Correct Trials	
Predictor	Algo.	Median	MAD	μ	σ	Median	MAD
$\langle r \rangle$	(1)	74%	19%	99%	1%	74%	20%
	(2)	74%	19%	99%	1%	74%	19%
	(3)	74%	17%	99%	1%	74%	18%
	(4)	73%	18%	99%	1%	73%	18%
	(5)	74%	18%	99%	1%	74%	18%
$\langle r, h \rangle$	(1)	78%	11%	92%	13%	74%	16%
	(2)	78%	11%	91%	15%	73%	16%
	(3)	78%	11%	91%	13%	73%	15%
	(4)	78%	11%	90%	14%	73%	16%
	(5)	78%	11%	90%	14%	73%	15%
$\langle r, h, d \rangle$	(1)	77%	12%	56%	34%	51%	37%
	(2)	77%	14%	54%	34%	51%	36%
	(3)	77%	11%	56%	34%	50%	35%
	(4)	77%	13%	53%	34%	49%	38%
	(5)	78%	12%	54%	34%	51%	38%
$\langle r, h, w \rangle$	(1)	81%	9%	87%	18%	76%	12%
	(2)	80%	9%	85%	19%	74%	12%
	(3)	81%	9%	86%	18%	76%	12%
	(4)	81%	9%	84%	19%	73%	14%
	(5)	81%	9%	84%	19%	74%	13%

Table 6.1: Comparison of the different regime classifiers.

of some selected, representative results.

As can be seen from the table, all of the regime labeling algorithms perform similarly across all metrics. The only consistent deviation is that algorithm **(1)** tends to try 1 to 2 percentage points more trials than the other algorithms, which corresponds to a slighly higher portion of correct trials. It seems that the added complexity to try and detect the direction of movement causes a slight decrease in performance. Likely, the situations where detecting the direction of travel is helpful is too rare to overcome the performance impact of the additional complexity.

Conditioning on the regime provides a significant performance improvement. Independent of the other variables on which the predictor conditions, it increases the median prediction precision by approximately 10 precentage points and reduces the MAD by about 10 percentage points relative to conditioning on

the same variables, but excluding the regime. Concretely, when conditioning on the hour of the day, the median precision increases from 70.4% to 77.6%, and the MAD decreases from 24.4% to 10.6% when also conditioning on the regime. Similarly, when conditioning on the hour of the day and whether the day is a workday, the median precision increases from 72.8% to 81.1%, and its MAD decreases from 21.1% to 9.3%. The trade-off is a decrease in the portion of prediction attempts. Specifically, when conditioning on the hour of the day, the mean attempts decreases from 99.9% to 91.7%, and when conditioning on the hour of the day and whether the day is a work day, the mean attempts decreases from 98.3% to 86.9%.

6.4.4 Current Tower

Conditioning on the regime is a type of location-dependent predictor. The location, however, is very coarse. We now consider conditioning on the current tower aggregate.

The predictor that just conditions on the current regime answers the question: what is the dominant tower for this particular regime? The response is informative; the predictor tells us where the user spends the most time while at that particular regime. Just conditioning on the current tower, however, is not very useful; it just answers the question: what tower is the user currently at? This is the current tower baseline, which we looked at in Section 6.3.

To make conditioning on the current tower more useful, we can condition on some other variables. For instance, we can condition on where the user will be in Δ hours, i.e., $\langle c, \Delta \rangle$, where c is the current tower aggregate. Unfortunately, this predictor is less useful than it perhaps initially appears. Assume c corresponds to the user's home, and the user usually arrives home at 7pm, leaves for the day at 7am, and otherwise isn't home. Further, assume she arrives at work at 7:30am and stays for eight hours, everyday. In this scenario, for $\Delta = 1$ hour, the predictor will always return the current tower aggregate with probability $^{11}/_{12}$ whether it is 7pm or 6am. Although this is clearly usually correct, it completely ignores the fact that at 6:45am the user will shortly leave for the day. This predictor is also inadequate for large values of Δ. If Δ is, say, 8 hours, then the predictor will always predict that the user will be at work in 8 hours (with probability $^{8}/_{12}$), even if the user just arrived home!

We can improve this predictor by adding a temporal reference. Specifically, we can use the current hour of the day. This results in the $\langle h, c, \Delta \rangle$ predictor. This predictor considers questions of the form: given that it is currently between 5am and 6am, and the user is at home where will she be in 3 hours? This predictor should perform much better on the above scenarios.

Another possible temporal reference is how long the user has been at the current tower. Recall from Section 3.2.2 that user e7d typically arrives at work between between 6am and 10am, but tends to stay for 8 hours. A predictor based on how long the user has been at work would do better at predicting when this user will leave than the time-of-day based predictor. The first problem with using dwell time is that it breaks down for places that the user visits multiple times per day for different amounts of time. In this case, it might make sense to break the day into, say, six four-hour blocks, and condition on the current block (i.e., include a very coarse-grained reference to the current time). Another problem is that the tower sampling that we observed will cause the total time at the current location to often be reset. Thus, it is probably necessary to aggressively smooth the trace before using this predictor. We leave exploring this option to future work.

There are two ways to deal with Δ in the time-of-day based predictor. Using Markov chains, we can fix Δ to, say, an hour, and when we want to predict the user's location in 12 hours, we take the current trasition probability matrix, and raise it to the 12^{th} power. Unfortunately, each transition in a markov chain adds a fair amount of uncertainly.

Instead, we propose maintaining a direct transition probability matrix for each interesting prediction offset. Since we are primarily interested in predicting the user's location in the near future, and since we have a fair amount of tolerance, i.e., the utility of predicting that the user will arrive at the gym at 3pm when she actually arrives at 3:15pm, is approximately the same as predicting that she will arrive at 3:00pm, maintaining a transition probability matrix for predictions 1 hour in the future, 2 hours in the future, etc., up to a day may be reasonable. The trade-off with this scheme is that it increases the storage requirements by a factor of the number of offsets that are maintained. If this storage overhead is too high, a possible compromise is to make all interesting offsets reachable in, say, two steps. Then, to determine the user's location at, say, $\Delta = 3$ hours, we would multiply the transition probability matrix for $\Delta = 1$ hour and $\Delta = 2$ hours; for $\Delta = 4$ hours, we would simply square the transition probability matrix for $\Delta = 2$ hours, etc. For the prediction offsets that we consider, this scheme requires just a third as much storage to maintain as the direct variant.

Table 6.2 shows the results for the simple predictor presented above as well as predictors based on the day of the week (d) and whether the day is a workday (w). For each predictor, the table also shows the results for three different ways to determine the current tower. First, we consider the simplest method: we use the tower that the device currently sees. If the conditional probability table for this tower is empty, then we fallback to the previous tower, etc. up to 15 visited

		Correct Attempts		Attempts		Correct Trials	
Predictor	Current Tower	Median	MAD	μ	σ	Median	MAD
$\langle h, c, \Delta \rangle$	current	81%	10%	74%	19%	60%	19%
	dominant	82%	10%	73%	19%	60%	20%
	mixture	81%	10%	73%	20%	60%	20%
$\langle h, d, c, \Delta \rangle$	current	81%	12%	38%	28%	31%	30%
	dominant	82%	11%	38%	28%	30%	30%
	mixture	82%	11%	37%	28%	30%	30%
$\langle h, w, c, \Delta \rangle$	current	83%	8%	66%	22%	57%	18%
	dominant	83%	8%	65%	22%	56%	19%
	mixture	83%	8%	65%	22%	56%	20%

Table 6.2: Comparison of current-tower based predictors. The "current tower" is either the *current* tower that the device sees, the *dominant* tower over the past 15 minutes, or a mixture of all towers visited over the past 15 minutes weighted according to their respective dwell times.

in the past 15 minutes. Second, we consider the dominant tower over the past 15 minutes, i.e., the tower at which the user spent the most amount of time over the past 15 minutes. The motivation for this is to avoid towers that the device only sees due to interference based on the observation that visits to these towers are relatively short. Finally, we use all of the towers visited over the past 15 minutes to created a mixture in which each tower's conditional probability table is weighted by the amount of time the device saw that tower over the preceding 15 minutes.

The table shows that using the current tower, i.e., the first method, is nearly as good or slightly better than the other two methods. Since the other methods are more complex and computationally intense, we recommend this method, and we use it in the rest of this chapter.

Plots of the predictors' performance are shown in Figure 6.8. In terms of their prediction precision, these predictors perform a bit better than their regime-based counterparts (see Table 6.1). Specifically, the median prediction precision for the hour-based predictor increases from 77.6% to 81.3%; for the day-of-week based predictor, it increases from 76.7% to 81.4%; and for the workday-based predictor, it increases from 81.1% to 83.0%. These complementary predictors have similar MADs, but the prediction attempts decreases by 19%, on average, when conditioning on the tower. The fewer attempts are expected given that

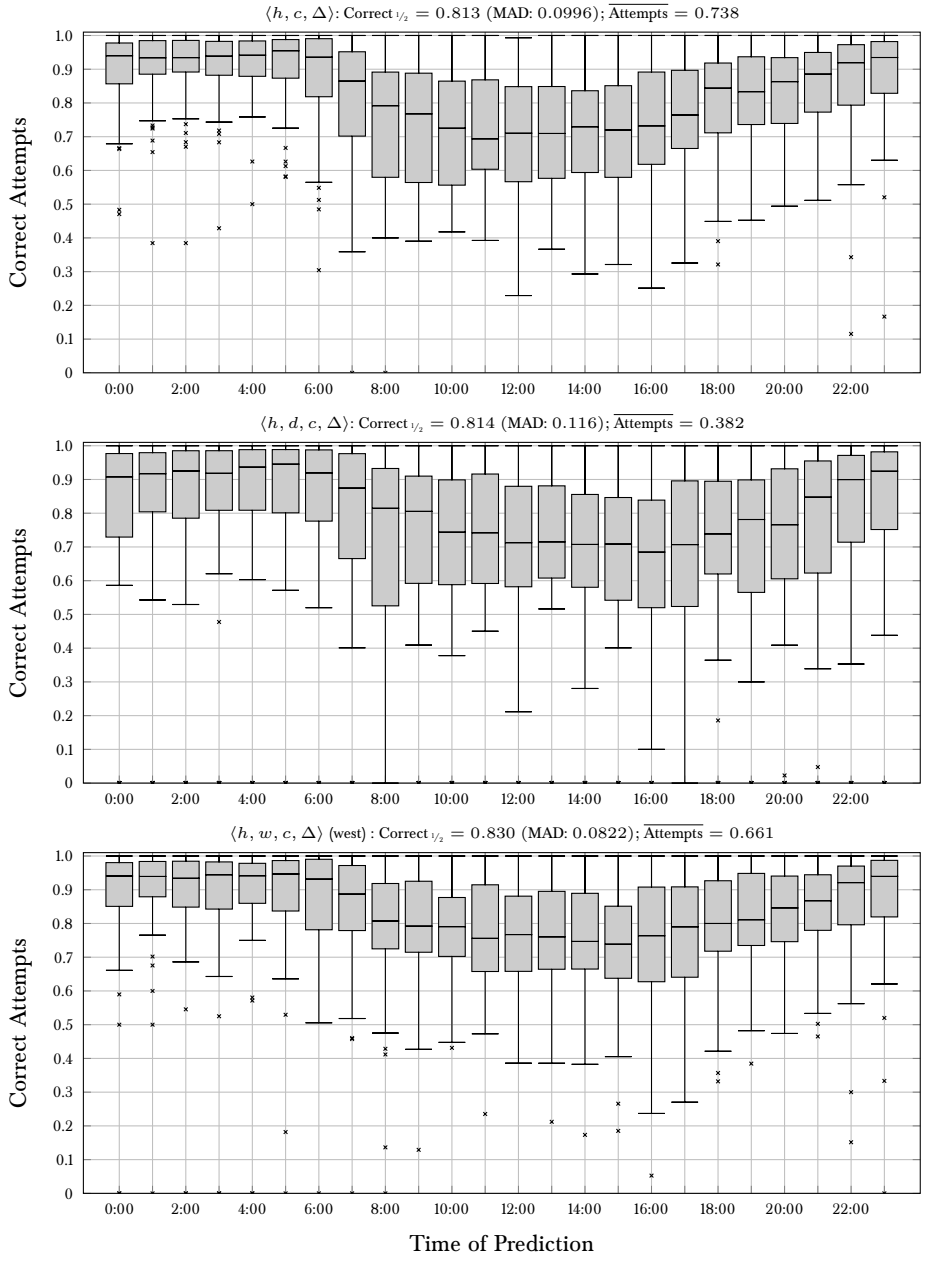


Figure 6.8: Box plots of the portion of correct prediction attempts of the predictors $\langle h, c, \Delta\rangle$, $\langle h, d, c, \Delta\rangle$, and $\langle h, w, c, \Delta\rangle$, respectively for different prediction times. The x-axis is the time of the prediction.

the many more tower aggregates compared to regimes spread the data out much more. When we just consider the traces with at least 16 weeks worth of data, the median performance increases to 84.7% with a very low MAD of 7.8%. Although the portion of attempts is not huge, the mean portion of attempts increases from 38.2% to 63.0%.

6.5 Weight Threshold

In our evaluation so far, we only made a prediction if there was at least 2 hours worth of data for the current condition. We now explore how changing this value impacts the performance. Table 6.3 shows how the performance changes when varying the weight threshold for the regime- and current-tower-based predictors. (Note: because days are broken into hour segments, thresholds less than an hour won't change the results for predictions at least an hour in the future.)

The table reveals that, in general, using a 2-hour threshold for the weight results in the highest median precision, or nearly so. In fact, for settings up to about 8 hours, the portion of correct attempts is mostly unchanged. This suggests that these predictors do not require a lot of training data.

In contrast, increasing the weight threshold significantly decreases the number of prediction attempts. For instance, for a 2 hour threshold, the $\langle r, h \rangle$ predictor attempts 91.7% of the predictions, on average. However, increasing the threshold to 16 hours causes this to drop to just 51.7%. This drop in the number of attempts is expected.

Surprisingly, the drop in prediction attempts is accompanied by a drop in the median prediction precision. Normally, we expect more data to result in better performance. The issue here is that raising the threshold excludes conditions that never get a lot of data. In other words, a higher threshold means the predictors only make predictions for significant locations. We can imagine scenarios that explain why a user's behavior at significant locations may be less predictable. A person's home, for instance, is typically visited between many different activities, whereas after going to shopping, a person might always go home to make sure perishable items are returned to a refrigerator as quickly as possible. Further, behavior changes with time. An infrequently visited place may never be visited again, but people don't normally move just because they've changed their afternoon activity.

In conclusion, a small weight threshold appears to be best: it provides the highest prediction precision and makes the most prediction attempts.

		Correct Attempts		Attempts		Correct Trials	
Predictor	Weight	Median	MAD	μ	σ	Median	MAD
$\langle r, h \rangle$	60 m	77%	11%	95%	8%	75%	14%
	2 h	78%	11%	92%	13%	74%	16%
	4 h	78%	11%	88%	18%	73%	18%
	8 h	77%	12%	75%	27%	68%	17%
	16 h	76%	15%	52%	36%	47%	35%
$\langle r, h, d \rangle$	60 m	78%	12%	78%	25%	69%	18%
	2 h	77%	12%	56%	34%	51%	37%
	4 h	73%	22%	38%	36%	23%	34%
	8 h	62%	55%	25%	31%	0%	0%
	16 h	0%	0%	12%	22%	0%	0%
$\langle r, h, w \rangle$	60 m	81%	10%	92%	11%	78%	13%
	2 h	81%	9%	87%	18%	76%	12%
	4 h	81%	10%	76%	24%	68%	18%
	8 h	79%	12%	56%	32%	48%	34%
	16 h	74%	19%	38%	34%	24%	36%
$\langle h, c, \Delta \rangle$	60 m	79%	10%	80%	14%	64%	18%
	2 h	81%	10%	74%	19%	60%	19%
	4 h	82%	10%	67%	23%	54%	21%
	8 h	82%	10%	52%	27%	46%	25%
	16 h	79%	14%	34%	30%	26%	37%
$\langle h, d, c, \Delta \rangle$	60 m	81%	11%	57%	24%	47%	24%
	2 h	81%	12%	38%	28%	31%	30%
	4 h	78%	20%	25%	28%	11%	17%
	8 h	65%	52%	15%	22%	0%	0%
	16 h	0%	0%	7%	16%	0%	0%
$\langle h, w, c, \Delta \rangle$	60 m	82%	9%	75%	18%	62%	17%
	2 h	83%	8%	66%	22%	57%	18%
	4 h	84%	8%	54%	25%	46%	21%
	8 h	83%	12%	38%	28%	29%	33%
	16 h	79%	17%	24%	27%	13%	19%

Table 6.3: Comparison of different weight thresholds for several predictors.

		Correct Attempts		Attempts		Correct Trials	
Predictor	↓Hour	Median	MAD	μ	σ	Median	MAD
$\langle h \rangle$	∞	70%	16%	100%	0.1%	70%	16%
	21 d	72%	12%	100%	0.1%	72%	12%
	14 d	73%	12%	100%	0.1%	73%	12%
	7 d	75%	13%	100%	0.1%	75%	12%
	4 d	71%	15%	100%	0.2%	71%	15%

Table 6.4: Comparison of aging for hour-related predictors for traces with at least 16 weeks worth of data.

6.6 Aging

So far, when the predictors observe new data, they have simply added it to their conditional probability tables. This is reasonable assuming that the system is static: according to the law of large numbers, more data will improve performance. People, however, are not static. A college student has different classes at different locations each semester, for instance. And, it is also not unusual for people to change some of their free-time activities each season. In other words, experience indicates that people's behavior is dynamic, and our predictors should adapt accordingly.

A simple way to age data is to only keep, say, the last month worth of data and throw the rest away. This is a bit unfortunate, as it throws away not only the old data about what a person did at her primary regime, but it also discards old data at occasionally visited locations where the person's behavior may remain relatively stable, i.e., other regimes.

Based on this, we propose keeping some amount of data on a per-primary condition basis. (The primary condition is either the current tower aggregate or regime. If neither is used, then we consider the variable corresponding to the time of day to be the primary condition.) According to this scheme, a predictor that conditions on the current regime would keep the most recent month worth of data for each regime.

In Section 3.4.3, we observed that the amount of time spent at individual towers is consistent with a power law. Thus, if we decide to keep the last two weeks worth of data for a given tower aggregate, then we will keep about the last three to four weeks worth of data for the tower aggregate at the user's home (since a person spends about half her time there), but the last year worth of data for the supermarket that is visited once each week for an hour. To enable aging of

		Correct Attempts		Attempts		Correct Trials	
Predictor	↓Regime	Median	MAD	μ	σ	Median	MAD
$\langle r, h \rangle$	∞	76%	10%	96%	4%	74%	12%
	42 d	77%	10%	96%	4%	75%	11%
	28 d	77%	10%	96%	5%	75%	11%
	21 d	77%	10%	96%	5%	75%	11%
	14 d	77%	10%	96%	5%	75%	11%
	7 d	77%	11%	94%	7%	74%	13%
$\langle r, h, d \rangle$	∞	83%	9%	86%	11%	72%	12%
	42 d	84%	8%	85%	12%	71%	13%
	28 d	83%	6%	82%	15%	70%	13%
	21 d	83%	6%	73%	16%	63%	17%
	14 d	86%	5%	35%	8%	31%	8%
	7 d	0%	0%	0%	1%	0%	0%
$\langle r, h, w \rangle$	∞	82%	8%	94%	6%	78%	10%
	42 d	83%	6%	94%	6%	79%	8%
	28 d	83%	7%	94%	6%	78%	9%
	21 d	83%	7%	93%	7%	78%	9%
	14 d	83%	7%	92%	9%	77%	9%
	7 d	82%	8%	76%	9%	64%	6%

Table 6.5: Comparison of aging for regime-related predictors for traces with at least 16 weeks worth of data.

less frequently visited locations, we instead propose keeping the data for the last n days with data for each primary condition.

To evaluate how effective aging is, we only use the traces with at least 16 weeks worth of data. The whole point of aging is to adapt to dynamic behavior, and we are unlikely to see much dynamic behavior with just a few weeks worth of data.

Table 6.4 shows the performance of the time-of-day predictor with different settings of the aging parameter (including no aging, which we denote by ∞). A week of aging results in a modest performance boost: the median precision increases from 70.4% to 74.7% and the MAD decreases from 16.2% to 12.6%.

Table 6.5 and Table 6.6 show the performance of aging for the regime- and current-tower-based predictors, respectively. Surprisingly, aging the data does not appear to improve the performance of these predictors. On the one hand, it makes sense that aging does not help detect major changes, such as a new job.

		Correct Attempts		Attempts		Correct Trials	
Predictor	↓Tower	Median	MAD	μ	σ	Median	MAD
$\langle h, c, \Delta \rangle$	∞	80%	9%	81%	10%	64%	15%
	28 d	79%	10%	80%	10%	63%	14%
	21 d	79%	10%	80%	10%	62%	14%
	14 d	78%	10%	79%	11%	61%	15%
	7 d	76%	12%	75%	13%	57%	17%
	4 d	71%	12%	70%	15%	50%	19%
$\langle h, d, c, \Delta \rangle$	∞	85%	8%	63%	17%	54%	21%
	28 d	84%	8%	56%	19%	44%	22%
	21 d	84%	8%	51%	19%	39%	20%
	14 d	80%	9%	41%	18%	30%	15%
	7 d	56%	12%	12%	6%	7%	5%
	4 d	14%	15%	4%	4%	0%	1%
$\langle h, w, c, \Delta \rangle$	∞	84%	8%	77%	12%	64%	16%
	28 d	84%	8%	75%	13%	61%	16%
	21 d	83%	8%	74%	13%	60%	16%
	14 d	81%	8%	71%	14%	57%	16%
	7 d	78%	8%	63%	16%	48%	16%
	4 d	68%	11%	47%	12%	30%	9%

Table 6.6: Comparison of aging for current tower-related predictors for traces with at least 16 weeks worth of data.

This type of change is often accompanied by a move, and thus a new regime would be established, which is exactly what these predictors already adapt to. But, even if a person does not move after start a new job, we still might not observe the change in our traces: a new job sometimes means a new cell phone either because the person can suddenly afford the upgrade, or the employer provides the person with a new device. On the other hand, we still expect aging to help with minor changes in behavior, which, as we already observed, it apparently does in the case of the hour-based predictor. Thus, it seems that the impact of minor changes to behavior is too small to detect, or the regime or the current tower aggregate already largely capture these changes.

Even if aging does not increase performance on our metrics, it does provide a mechanism to reduce the amount of stored data. Thus, in practice, it probably makes sense to perform some kind of aging. Based on our results, parity with respect to the unaged predictors on prediction precision *and* prediction attempts

requires aging to preserve about 2 to 3 weeks worth of data for the regime-based predictors case and 2 weeks of data for the tower aggregate-based predictors.

6.7 Combining Predictors

So far we've examined three predictor subfamilies: the temporal predictors in Section 6.4.1 and Section 6.4.2; the weak-location based predictors in Section 6.4.3; and, the strong-location based predictors in Section 6.4.4. For instance, using a weight threshold of just 2 hours, $\langle h, w, c, \Delta \rangle$ had a median prediction precision of 81.4%, but only attempted 38.2% of the prediction trials. On the other hand, the less specific predictor $\langle r, h \rangle$ had a lower median prediction precision of 77.6%, but attempted 91.7% of the prediction trials.

This suggests a simple approach to combining predictors: try the predictors with the highest prediction precision first and fallback to the less specific, but still good performing predictors, if there is not enough data. This is similar to prediction by partial matching (PPM), which was originally developed in the context of data compression, and first tries to use an order n Markov model to make a prediction, but falls back to an order $n-1$ Markov model if not enough data is available, etc. [63].

We make two small variations to this approach. First, we don't just marginalize out a variable, e.g., by first trying $\langle h, d, c, \Delta \rangle$ and then fall back to $\langle h, c, \Delta \rangle$ (i.e., marginalizing out d) if not enough data is available, but we are willing to try a different predictor, e.g., $\langle r, d, h \rangle$ instead of $\langle h, d, c, \Delta \rangle$. Second, to determine whether there is enough data, we consider two variants. First, we consider whether the current predictor has enough data, and whether the cumulative time is enough. We refer to the former as a hard threshold and the latter as a cumulative threshold. In practice, the results are similar, and, as such, we use the hard threshold unless otherwise specified. In both cases, we mix the results according to all of the used predictors' available data. Thus, if the first predictor only had half an hour worth of data available for the particular condition, and the weight threshold is 2 hours, then we mix the resulting conditional probability table weighted by $0.5/2$ with the conditional probability table of the fallback predictor weighted by $\min(w, 1.5/2)$, where w is the amount of data the fallback predictor has for the current condition. If the first fallback predictor doesn't have at least 1.5 hours worth of data, then we can fallback again in a similar fashion.

There are many ways to combine multiple predictors. A well-known approach is boosting. We leave exploring these other methods to future work.

Table 6.7 shows the results for the different predictor chains using a weight threshold of 2 and 16 and setting the age parameter for regimes to 28 and the

Combined Predictor	Weight	Predictor Attempts 1	2	3	4	5	Correct Attempts Median	MAD
$\langle r, h\rangle, \langle r\rangle$	2	91.6%	7.7%				76.8%	13.3%
	16	49.5%	45.9%				76.6%	12.9%
$\langle r, h, d\rangle, \langle r, h\rangle, \langle r\rangle$	2	53.2%	38.3%	7.7%			78.1%	12.9%
	16	0.0%	49.5%	45.9%			76.6%	12.9%
$\langle r, h, w\rangle, \langle r, h\rangle, \langle r\rangle$	2	86.5%	5.1%	7.7%			80.4%	11.2%
	16	24.9%	24.6%	45.9%			76.6%	12.9%
$\langle h, c, \Delta\rangle, \langle r, h, d\rangle, \langle r, h\rangle, \langle r\rangle$	2	73.0%	9.1%	10.5%	6.6%		76.7%	13.5%
	16	23.4%	0.0%	26.3%	45.7%		76.4%	12.7%
$\langle h, d, c, \Delta\rangle, \langle h, c, \Delta\rangle, \langle r, h, d\rangle, \langle r, h\rangle, \langle r\rangle$	2	31.5%	41.7%	9.0%	10.5%	6.6%	77.1%	13.3%
	16	0.0%	23.4%	0.0%	26.3%	45.7%	76.4%	12.7%
$\langle h, w, c, \Delta\rangle, \langle h, c, \Delta\rangle, \langle r, h, w\rangle, \langle r, h\rangle, \langle r\rangle$	2	64.4%	8.7%	17.6%	2.0%	6.6%	78.8%	12.4%
	16	7.0%	16.5%	10.9%	15.3%	45.7%	76.2%	12.4%

Table 6.7: Portion of trials attempted by different predictor chains. Note: later predictors can only attempt prediction trials that preceding predictors did not try. All predictors chains make at least 95% attempts, on average.

age parameter for towers to 21. In all the cases that we tried, the total prediction attempts is at least 95% and often significantly closer to 100%. Again, we see that better performance is obtained by using the most specific predictor possible (i.e., a smaller weight threshold), and only falling back if absolutely necessary.

6.8 Final Evaluation

So far, we have looked at the performance of the predictors without regard to the prediction offsets. This is because the performance is significantly less dependent on the prediction offset than it is on the time of day, at least for the prediction offsets that we considered (from 0.5 to 23.5 hours in the future with a one hour step size). Figure 6.9 shows the performance for different prediction offsets for regime-based predictors, and Figure 6.10 shows the performance for different current-tower-aggregate-based predictors. For regimes, the performance is nearly independent of the prediction offset. In contrast, when using current tower aggregates, the performance is better in the near future, but flattens out for predictions further than about 4 hours in the future.

The reason that the regime-based predictor performs equally well in the immediate future as in the near future is that it doesn't consider the user's current location (it just incorporates the general geographic area traversed on a day-to-day basis). That is, it is relying on people's tendency to return to the same places at the same time. In contrast, the current-tower-aggregate-based predictor is tightly coupled with the current location, and a user's behavior in the immediate future is closely bound to the current location whereas behavior further in the future quickly becomes independent of the user's current location. For instance, if a person buys groceries, he is likely to go home to drop them off, but what he does after that is probably not causally related to the visit to the grocery store.

Figure 6.11 shows the performance broken down by the hour of the day, but only for predictions made half-an-hour into the future. Figure 6.12 and Figure 6.13 show the same results, but for predictions made 2.5 and 12.5 hours into the future, respectively. For predictions in the immediate future, the current-tower-aggregate-based predictors attain a median prediction precision of 87% and they attempt nearly every trial. For 2.5 hours in the future, which is arguably the most important offset for prefetching applications, because it allows them time to prefetch fresh content appropriate to the location the user will travel to in the near future, the median prediction precision is just over 80%.

Given this performance profile, it makes sense to use different predictors for different prediction offsets. For half an hour in the future, the current tower

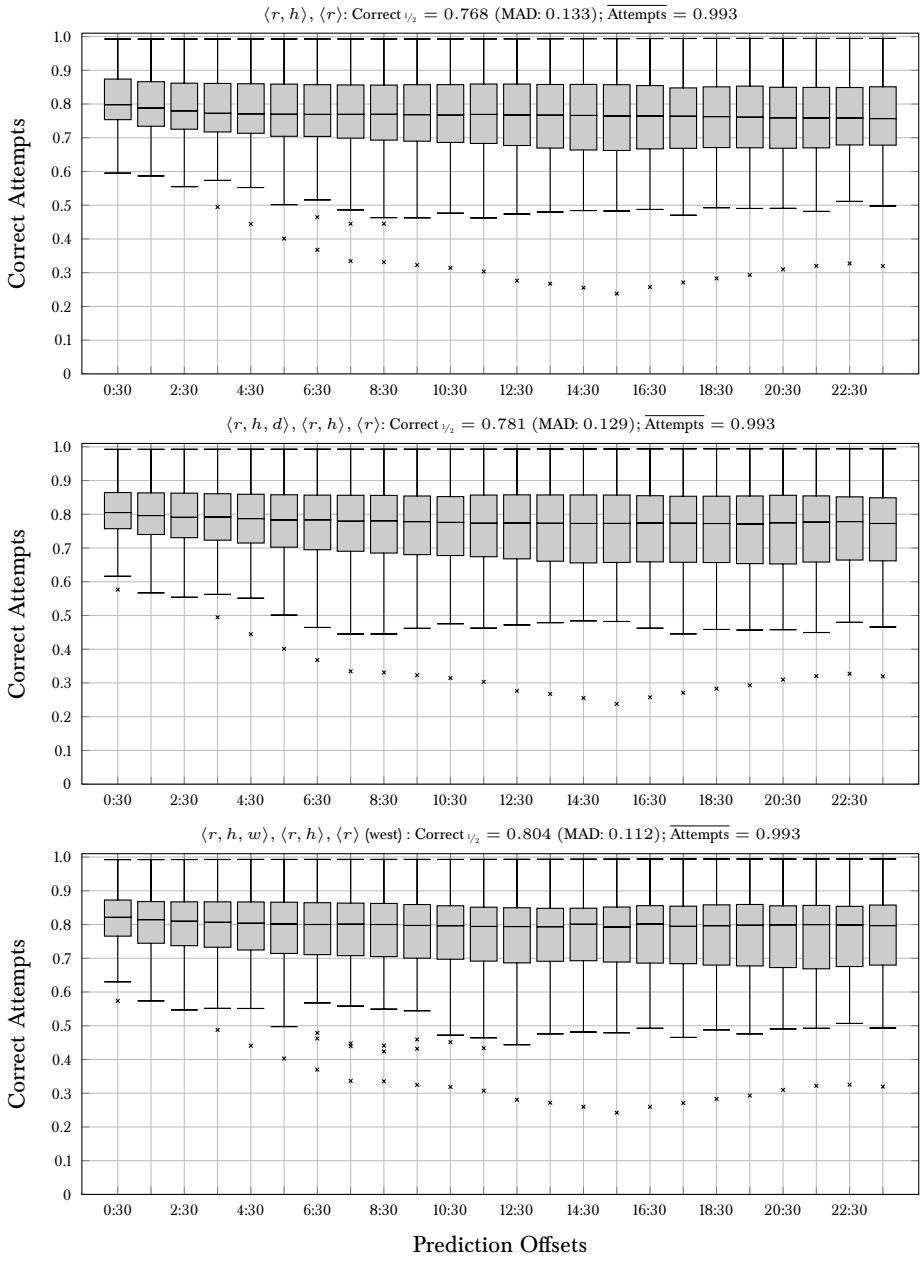


Figure 6.9: Performance of regime-based prediction chains broken down by prediction offsets.

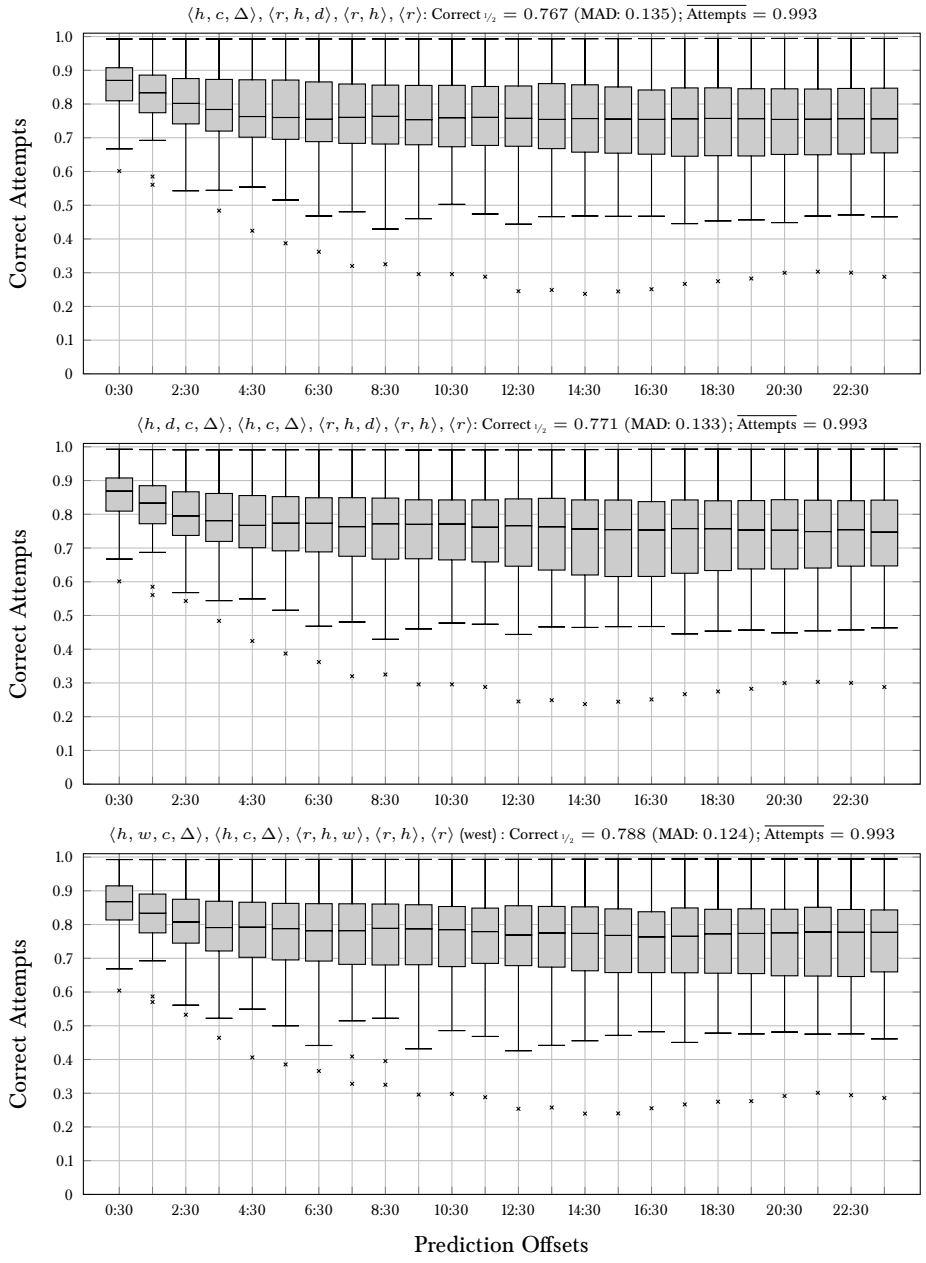


Figure 6.10: Performance of current-tower-aggregate-based prediction chains broken down by prediction offsets.

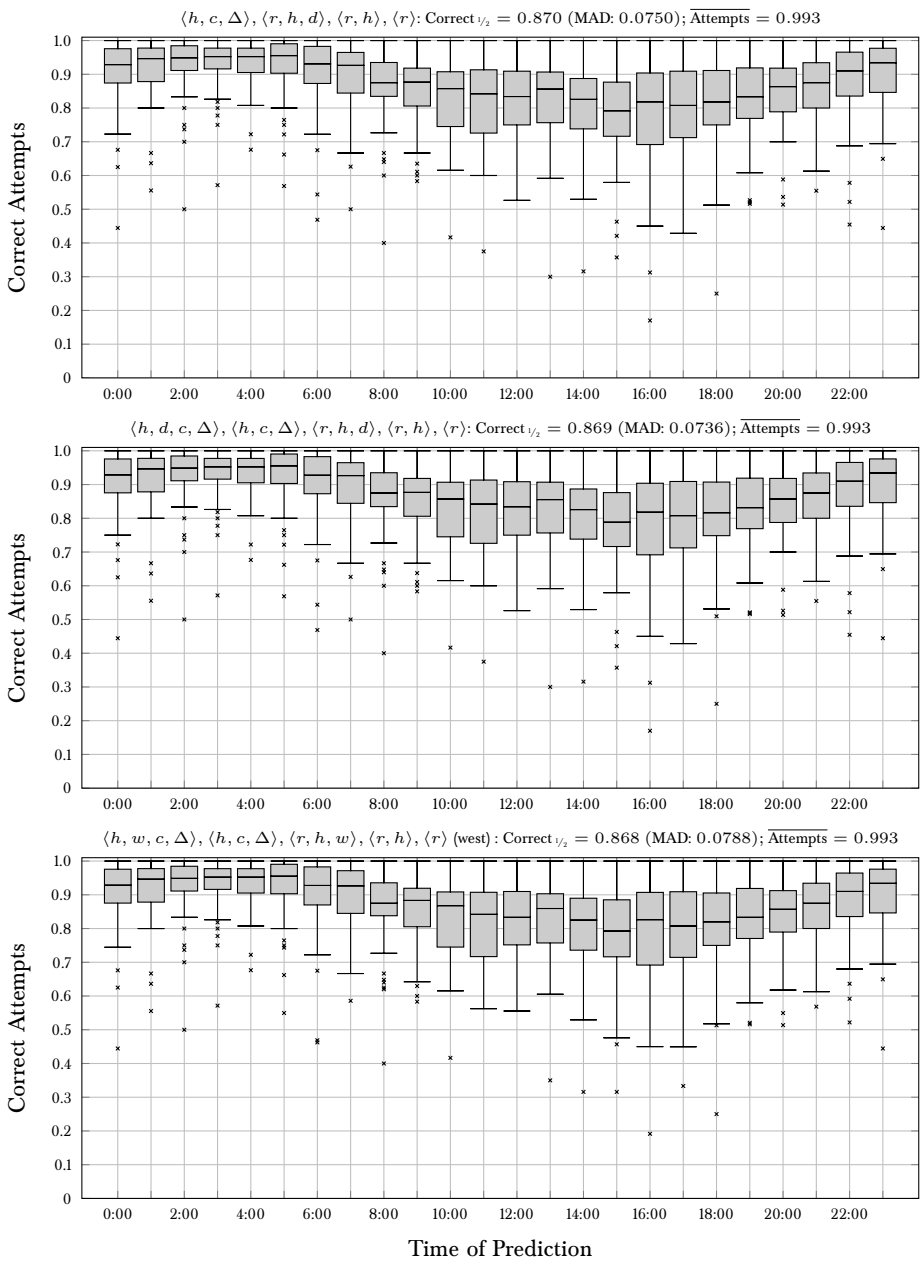


Figure 6.11: Performance of current-tower-aggregate-based prediction chains in the immediate future (half hour).

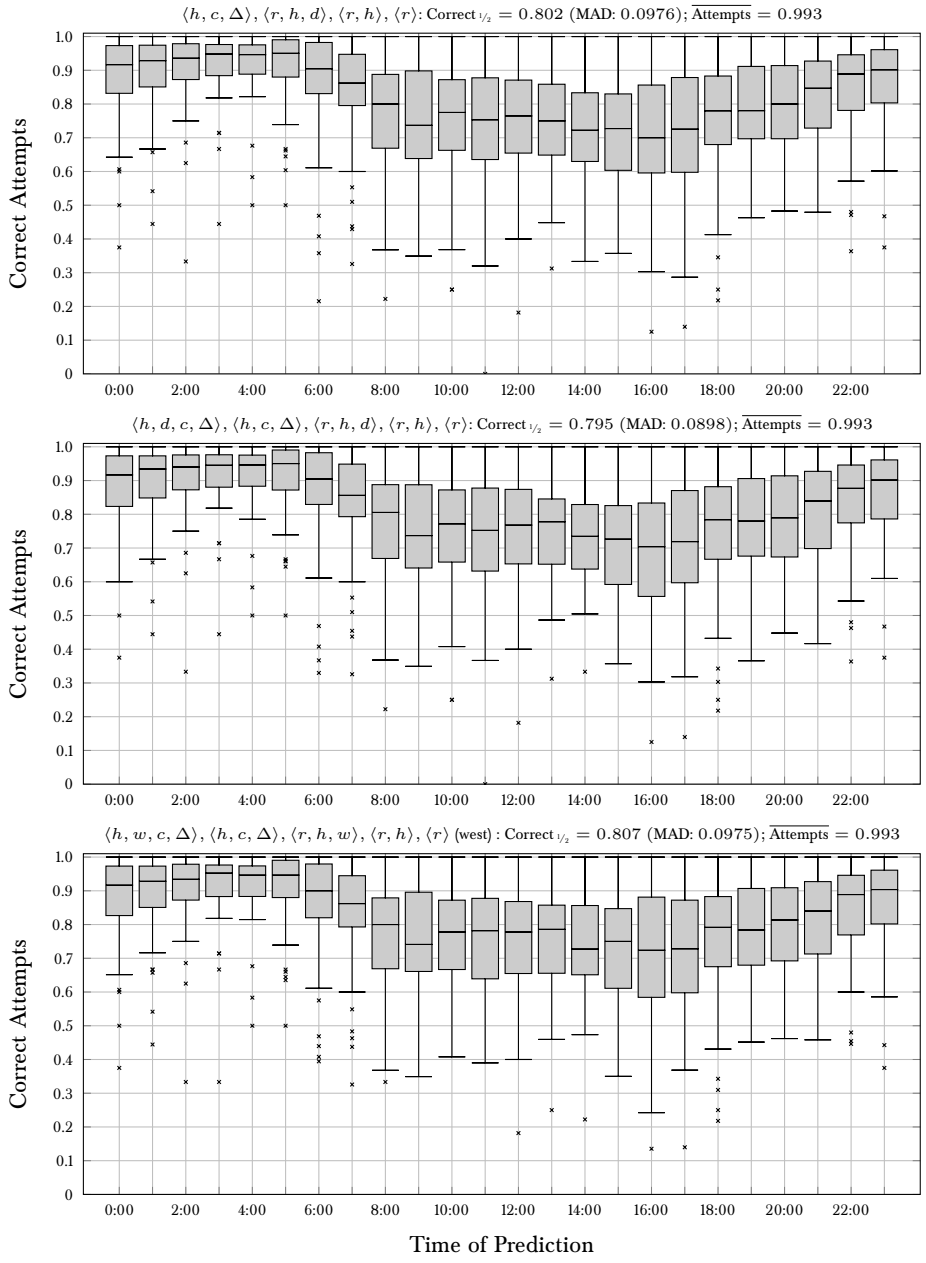


Figure 6.12: Performance of current-tower-aggregate-based prediction chains in the near future (2.5 hours).

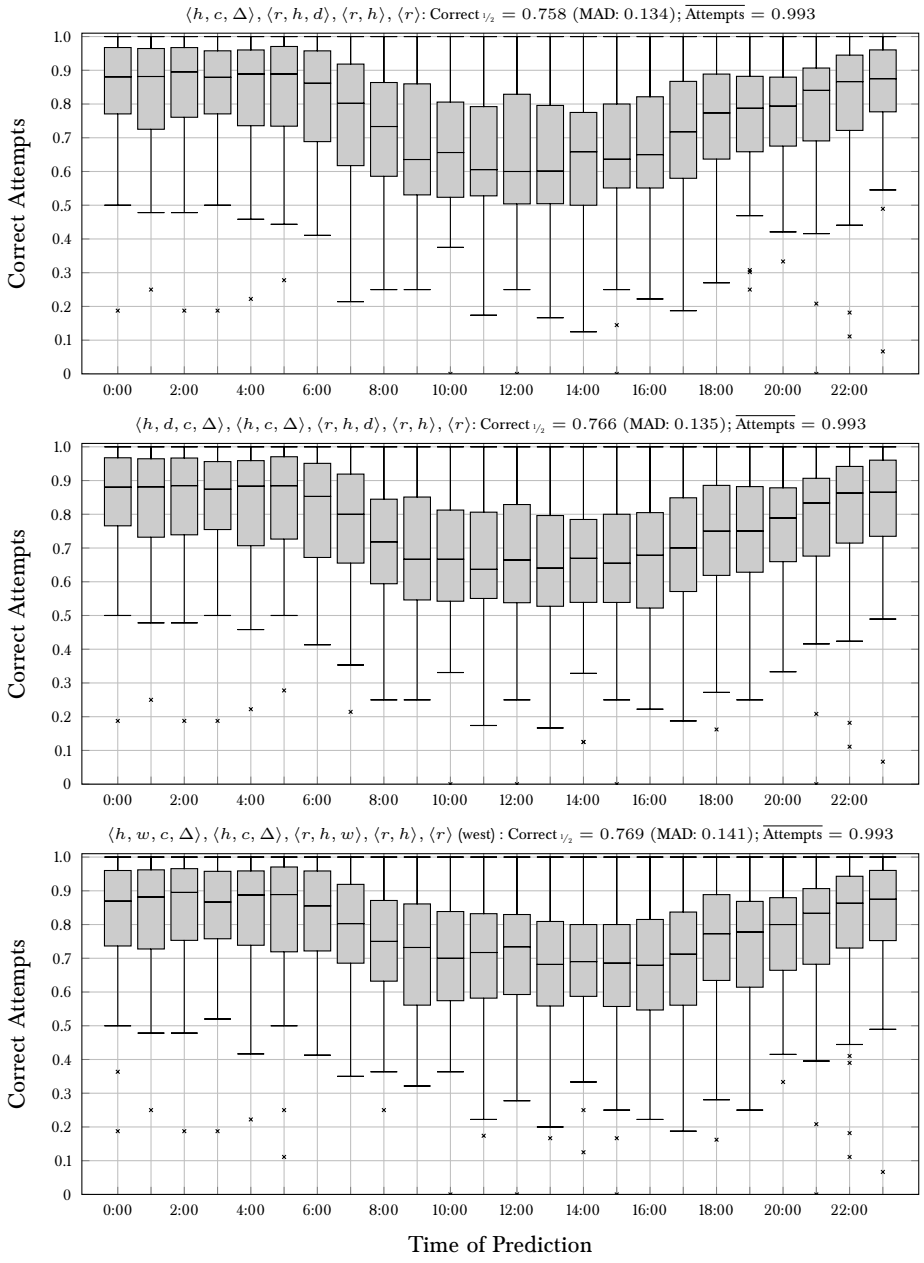


Figure 6.13: Performance of current-tower-aggregate-based prediction chains in 12.5 hours.

aggregate based predictor is best (Figure 6.2). Between 1.5 and 3.5 hours in the future, current tower aggregate based predictors performance best. And, further in the future, regime-based predictors perform best. Such a predictor results in correctly predicting 82% of the trials.

6.9 Future Directions

Our predictors don't consider visits as a whole, but simply look at the current conditions at a given instant in time. Treating visits as a whole would help identify behavior, such as, if the user is not at work by 10am, then the user is unlikely to go to work today. It could also identify things such as: independent of when the user arrives at work, she will stay there for x hours.

Predicting the user's location in the near future provides nearly immediate feedback about the predictor's performance. This can be exploited using reinforcement learning to adapt our model on the fly. A simple approach, is to use a multi-armed bandit (i.e., boosting) to chose among multiple predictors, or use ensemble learning to combine multiple predictors, e.g, using AdaBoost.

Our predictors don't currently take advantage of long-range dependencies. For instance, where a user is at 9am might more strongly determine where the user will be at 4pm (i.e., in 7 hours) than where the user is at 3pm (i.e., an hour). Although we do not exploit them, the current tower aggregate predictors capture these dependencies. Instead, we currently always use the predictor with the smallest prediction offset based on the assumption that shorter dependencies that incorporate the user's current location are usually better than longer predictions that ignore the user's current location, and instead use the user's past location. One way to automatically identify the most useful long-term dependencies would be to use boosting.

Our implementation integrates data online. Currently, we do not go back into the past and revise facts based on new knowledge. For instance, when the user transitions to a new regime, she initially observes new towers. Some of these new towers will be aggregated into places. When towers are aggregated, we do not transfer our knowlege about the user's behavior at those towers to the aggregate. This has two negative consequences. First, we need to learn the user's behavior at the new aggregate from scratch even though we already have some information about that aggregate. Second, if a tower has been aggregated and assigned a new identifier, then the predictor will never see the old identifier again, but it will continue to assign it a non-zero probability of being visited.

Currently, we do not attempt to identify routes. One way to make predictions on routes, would be to indicate that the user is going towards some significant

location. Then, ground truth is not some rarely visited tower along a route, which is impossible to predict, but a meaningful label.

6.10 NextPlace Comparison

The NextPlace algorithm uses non-linear time series to predict a person's future location [105]. Although the evaluation uses GPS and Wi-Fi traces to determine a user's location, the authors emphasize that the approach is general and "can be adopted in systems without any spatial or geographical information about locations, i.e., access points in 802.11 WLANs" [105, Sec. 2].

6.10.1 Algorithm

The idea behind NextPlace is relatively straightforward. For each *significant* location (as determined by an algorithm described in the paper), NextPlace maintains two time series: a list of arrival times as seconds since the start of the day (C); and, a list of each visit's dwell time (D). To predict when a location will be next visited, NextPlace takes the last m values from its arrival time series, and looks for similar subsequences. (The authors found that $m = 3$ resulted in the best performance.) A subsequence is considered to be similar to another subsequence if their Euclidean distance is less than ϵ, which Scellato et al. set to 10% of the time series' standard deviation based on Kantz's and Schreiber's recommendation [51]. If there are no similar subsequences, then there isn't enough information to make a prediction. If there are similar subsequences, then NextPlace predicts the next visit will occur at the mean of the subsequent arrival times. To predict when the location will be visited a *second* time, NextPlace just averages the *second* entries after the end of the matched subsequences, etc. Because this can result in wrap around (e.g., the first prediction is at 10am and the second is at 8am), if the n^{th} prediction appears to occur before the $n-1^{\text{th}}$ prediction, then the latter prediction is considered to occur the following day, i.e., we add 24 hours.

To predict how long the user will stay at a place, NextPlace uses the mean of the entries in the dwell time series corresponding to the entries used to predict when the user will visit the location.

Figure 6.14 shows a simple example in which we imagine a person who arrives at work at around 9:00, and returns to work after lunch at around 13:00. The last four entries (underlined) are similar to the two subsequences starting at the second and fourth positions (circled). These subsequences are followed by 13:01 and 13:05 (boxed) in the arrival time series. Taking the mean, NextPlace predicts the person would return to work at 13:03. The corresponding entries in

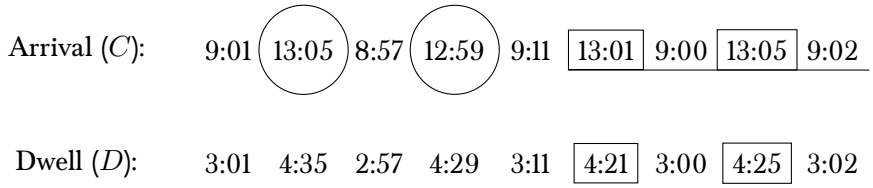


Figure 6.14: A fictitious example of a time series showing a person's arrival time at work and another showing the dwell time. The last $m = 4$ elements (underlined) form the recent history that is used to identify similar past behavior. The first element of each of the two similar subsequences are circled in the arival time series. The entries used to predict the next visit and its dwell time are boxed.

the dwell time series are 4:21 and 4:25 (boxed), which NextPlace takes the mean of to predict how long the person will stay at the location.

To predict where the user will be in t seconds in the future, say, noon, NextPlace iterates over each significant location, and makes a prediction. If a prediction includes t (e.g., arrival is 11:30 and dwell is 1 hour), it is returned. If the predicted location is left before t (e.g., arrival is 11:30am and dwell is 15 minutes), then another prediction is made, and the process is repeated. If there is no prediction that includes t across all significant places, then NextPlace predicts that the user will be at an insignificant place. The authors count this as a correct prediction if the user does in fact visit a not-significant place, but they don't include predictions of non-significant places in their evaluation, because there are so many in their data sets [105, Sec. 3.4].

6.10.2 Analysis

The idea behind keeping only the seconds since midnight is that "the sequence of important locations that an individual visits each day is more or less fixed" [105, Sec. 2]. In other words, NextPlace assumes that people follow the same routine *every* day. This may be true when considering user activity on, say, a college campus (which one of the data sets the authors use to evaluate NextPlace is) or at work (which another data set the authors use to evaluate NextPlace is), but it is not true in general. The simplest example is that behavior on workdays is different from behavior on days of rest.

To understand how just considering a daily routine can cause performance to break down, consider a user's work location, w, which is visited at the same time every workday for the same amount of time. For such a location, NextPlace will predict that it will be visited every day, including on days of rest. Further,

because NextPlace is a location-independent predictor, i.e., it doesn't consider the current location, and, further, it doesn't age data, if the user goes on vacation or changes jobs, NextPlace will continue to predict that the user will be at w everyday during the same time period. This is because the last m visits to w had (and will always have) a match in w's history.

NextPlace will also have difficulty recovering from abnormal behavior. Consider a person who goes to work at 9am and stays for 8 hours. When predicting subsequent visits to w, the last m visits will have been at 9am and NextPlace will predict the next visit to w to be at 9am and last 8 hours. Now, imagine that one day, the user has an appointment in the morning and arrives at work at 11am. When NextPlace predicts the next visit to w, it will fail to make a prediction, because $\langle \ldots, 9, 9, 11 \rangle$ is not similar to anything in w's history. Assuming the user resumes her usual routine the next day, NextPlace will still be unable to make a prediction, because $\langle \ldots, 9, 11, 9 \rangle$ is not similar to anything in w's history. NextPlace will only recover after m days of her regular work routine.

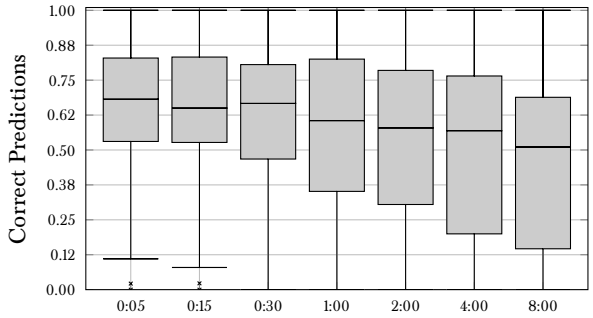
6.10.3 Reevaluation

We implemented the NextPlace algorithm, and evaluated it on our data set. To find the set of significant places, we took the top places that correspond to 90% of the total dwell time. In practice, this is just a dozen or so tower aggregates (recall from Figure 3.18: our traces contain 100s or 1000s of towers). This is similar to the number of significant places detected in the different traces that Scellato et al. evaluate, but they report that their significant places cover between just 0.18% and 15% of the trace's total time, i.e., between just 2.5 minutes and 3.6 hours per day. This is extremely low and quite surprising. Recall from Section 3.4.3, that the amount of time spent at towers is distributed according to a power law, and the top two towers account for 2/3s of the total time.

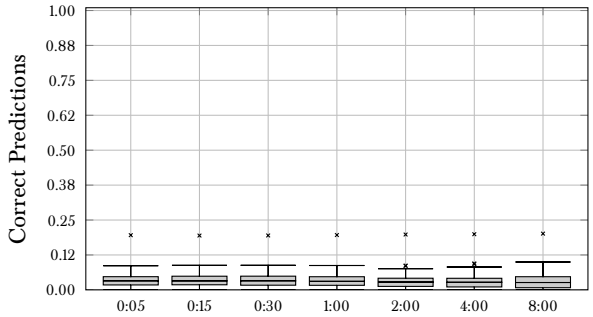
The results for $m = 3$, $\epsilon = 0.1$, and our standard tower aggregation algorithm are shown in Figure 6.15.

The first plot shows the prediction precision, i.e., the portion of correct prediction attempts. The results of running NextPlace on our trace are a bit lower than what the authors report for their traces, but they are still comparable, and the general trend is consistent. We suspect that this difference is primarily due to differences between the evaluated data sets. As previously mentioned, the data sets used to evaluate NextPlace didn't track users continuously, but only during part of their day, e.g., when the participant was driving his cab or she was at university. Our trace includes each user's whole day and weekend. As previously mentioned, NextPlace can't deal with days with different routines.

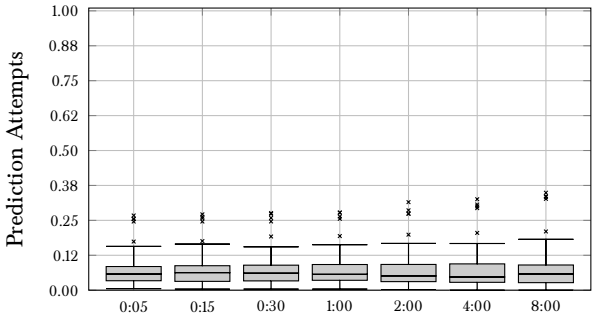
The second plot shows the portion of correct prediction trials. This is ex-



(a) Portion of correct, attempted predictions (i.e., the prediction precision).



(b) Portion of correct prediction trials.



(c) Portion of prediction attempts.

Figure 6.15: NextPlace evaluation for $m = 3$, $\epsilon = 0.1$ and no extra smoothing. The first plot shows the ratio of correct predictions to prediction requests for different prediction times. The second plot shows the "prediction precision," the ratio of correct predictions to prediction *attempts*. The final plot shows the ratio of prediction attempts to prediction requests that were actually attempted. NextPlace is apparently only trying a small portion of the total prediction requests.

			Correct Attempts		Attempts		Correct Trials	
m	ϵ	Smooth	Median	MAD	μ	σ	Median	MAD
1	0.1	—	52%	29%	21%	14%	7%	4%
1	0.1	5m	50%	27%	33%	16%	12%	8%
1	0.1	10m	51%	23%	34%	16%	14%	10%
1	0.1	15m	50%	22%	35%	16%	15%	11%
1	0.1	20m	56%	22%	35%	16%	14%	10%
1	1	—	44%	28%	19%	15%	5%	3%
1	1	5m	51%	25%	38%	18%	14%	10%
1	1	10m	55%	25%	42%	17%	16%	12%
1	1	15m	52%	27%	44%	16%	17%	12%
1	1	20m	53%	28%	46%	16%	19%	14%
2	0.1	—	60%	23%	14%	10%	5%	3%
2	0.1	5m	53%	33%	14%	11%	6%	6%
2	0.1	10m	59%	26%	13%	11%	5%	5%
2	0.1	15m	61%	26%	12%	10%	5%	4%
2	0.1	20m	54%	34%	13%	11%	5%	6%
2	1	—	49%	28%	20%	13%	7%	4%
2	1	5m	52%	26%	35%	15%	13%	6%
2	1	10m	55%	23%	38%	15%	16%	12%
2	1	15m	55%	23%	40%	16%	16%	12%
2	1	20m	53%	23%	42%	16%	17%	12%
3	0.1	—	61%	28%	8%	7%	3%	2%
3	0.1	5m	63%	45%	4%	7%	1%	1%
3	0.1	10m	54%	44%	4%	6%	0%	1%
3	0.1	15m	52%	44%	4%	6%	1%	1%
3	0.1	20m	56%	48%	3%	6%	1%	1%
3	1	—	51%	26%	18%	12%	6%	4%
3	1	5m	52%	24%	31%	15%	13%	7%
3	1	10m	56%	23%	34%	16%	14%	11%
3	1	15m	55%	27%	34%	16%	15%	10%
3	1	20m	54%	24%	36%	15%	16%	11%

Table 6.8: Comparison of several NextPlace variations. m is the amount of history to use (the NextPlace authors recommend setting m to 3); ϵ is the amount of tolerance when identifying matches in the tower's history (default: 0.1); and ***smooth*** is an additional smoothing step in which we just use the dominant tower in each window of the specified size.

tremely low. The reason for this is clear from the third plot, which shows the portion of prediction attempts: NextPlace only attempts a prediction trial about 7% of the time.

We contacted the authors of NextPlace and tried to determine what could have gone wrong. Unfortunately, the authors did not remember the portion of prediction attempts, but suggested it should be close to the portion of the significant towers' dwell time, which, in our case, is 90%. We also discussed the implementation. Unfortunately, we couldn't identify any problems, and the authors were unable to find their implementation for comparison purposes.

We conducted several experiments to try and increase the prediction attempts. We decreased the amount of history used (m), raised the matching tolerance (ϵ), and increased the smoothing by just selecting the dominant tower in each x minute segment. Although these measures sometimes helped, the portion of prediction attempts remain consistently low as shown in Table 6.8.

6.11 Conclusions

In this chapter, we considered how to predict the user's location in the near future. Consistent with the requirements of our primary goal, predicting context, we are primarily interested in the user's approximate location, e.g., the fitness center and not the locker room, in the near future.

We identified four useful features for predicting the user's location: the time of day, the day of the week, the current regime, and the current tower aggregate. We designed and evaluated several algorithms for identifying regimes and found that the resulting performance is not terribly sensitive to the actual algorithm used, however, conditioning on the current regime does result in a significant improvement in the prediction precision. To condition on the current tower, we proposed maintaining transition probability matrices for all interesting prediction offsets, rather than a single transition probability matrix for the smallest unit of time and iterating to make a prediction further in the future. The tradeoff is an increase in the amount of storage space, but a reduction in the uncertainty. Our evaluation showed that the current-tower-aggregate-based predictors performed best, particularly, for predictions less than 4 hours in the future.

We also considered how much data is needed before the predictors make reasonable predictions. We found that even with just an hour of data for the current condition, the prediction precision was nearly maximal. We experimented with aging the data. Although this helped for the plain time-of-day-based predictors, it did not improve the performance of the regime- and current-tower-aggregate-based predictors. These predictors appear to already capture this behavior.

We then turned to maximizing the prediction attempts. For this, we considered chaining multiple predictors similar to PPM. Using this technique, we observed a nearly 90% median prediction accuracy for predictions a half hour into the future with complete coverage (i.e., all prediction trials attempts). For predictions 2.5 hours in the future, which is arguably the most important offset for prefetching applications, we observed a median prediction accuracy that is comfortably over 80%. Using different predictors for different prediction offsets, we obtained 82% prediction accuracy.

Finally, we compared our approach to NextPlace, a well-cited alternative approach based on non-linear time series. We found that although NextPlace has high prediction precision (the authors report a maximum of 80% for predictions half an hour in the future), the portion of attempts was extremely low (7%) for our reimplementation run against our data set. Because this metric was not reported in the paper, we contacted the authors, but they did not remember how high it was. Unfortunately, they weren't able to find their code and we were unable to reproduce their results using the Dartmouth data set, because the authors don't remember what subset of the data they used. Nevertheless, just considering the prediction precision, our predictors—including the baseline predictors—perform significantly better across all prediction trials, not just a small portion.

Chapter 7

Scheduling Opportunistic Data Transfers

So far in this thesis, we have explored how to collect and process cell tower traces in order to identify the places that an individual visits, and to predict her location in the near future. In the introduction, we argued that an important use of this technology is to prefetch data, which provides many small benefits that taken together could result in emergent behavior. In this chapter, we explore the slightly more general question of how change existing applications to support delay-tolerant transfers. Based on our analysis, we conclude that a piece of middleware in the form of an operating system service is in the best position to make decisions and schedule data transfers.

Our central observation in analyzing this problem is that every application should not have to actively monitor the environment, mine the user's actions, and schedule data transfers. Instead, a centralized service should provide an easy-to-use API that enables applications to realize common patterns. First, this infrastructure is, for the most part, not application specific. Thus, application developers should not have to reimplement the same, non-trivial logic. Second, if every application monitors the environment on its own, the same code will be executed by each application simultaneously, which places a higher load on the CPU, memory and the battery than a single centralized service. Third, since the relevant resources are shared by all applications (e.g., the data transfer allowance, energy, and local storage), and co-scheduling data transfers can exploit synergies, scheduling should be coordinated. Finally, a user's privacy can be better protected by providing coarse-grained information to applications rather than access to the user's precise geographic location or even the cell tower trace.

7.1 Related Work

The closest related work is on hoarding [40, 41, 54]. Hoarding tries to improve disconnected operation by caching a subset of the user's files on the mobile device. Work on hoarding assumed that the mobile device has limited storage relative to the user's set of documents, and focused primarily on determining which of the user's documents to cache based on recent use. An essential aspect of this is ensuring that all dependencies are available. Consider hoarding a programming project, for instance: it is not sufficient to cache half of the project's files; all of the files are needed to compile the code.

We are looking at a slightly different problem from hoarding. Whereas hoarding considers what documents the user is actively using, we are trying to predict what Internet-accessible documents the user will consume. Thus, the set of documents under consideration is not only significantly larger, but the data have fundamentally different access patterns. Specifically, we expect most files to be used just once. This means new techniques are required to infer what the user is likely to access. Further, whereas hoarding was concerned with scheduling the local storage, this is only a secondary concern for us: we are first interested in scheduling the user's data allowance, and conserving energy.

We are not aware of any smartphone platforms that implement a general-purpose transmission manager. The closest service that we are aware of is Maemo's heartbeat daemon, which facilitates co-scheduling of network transmissions, in particular, keep-alive packets .[1] This saves power by better amortizing the ramp up and tail energy that is used by an activate wireless interface.

A number of streaming services, such as Google Play and Amazon Prime, allow the user to download content so that she can listen while offline or avoid network transfer fees. Until recently, to take advantage of this service, users needed to manually select the data they wanted to be locally cached. As of the summer of 2016, Amazon introduced *On Deck*, which is a service that uses otherwise unused local storage to cache videos that they believe the user may be interested in. (The author's personal experience with this is that Amazon's algorithms are doing a poor job at identifying interesting content: not only are the selected films completely irrelevant, but the algorithms apparently don't detect that the user is watching a series and should download the next unwatched episode.) Google Maps can transparently cache data, which allows it to work when the device is offline. Although useful, all of these solutions are point solutions: they are designed for a single application, and are not general-purpose operating system services, which is the focus of our work in this chapter.

[1] `http://wiki.maemo.org/Documentation/Maemo_5_Developer_Guide/Architecture/System_Software#Heartbeat_Daemon_.28Heartbeatd.29`

7.2 Design Constraints and Tradeoffs

Before presenting a design, we first take a detailed look at the design space. We start by considering the tradeoffs associated with modifying applications to support delay tolerant transfers versus having a third-party manager provide the functionality. We then examine how to build a system that works completely transparently to the applications, and the ways in which application support can rectify such a system's shortcomings, and substantially simplify its realization.

7.2.1 Dividing Responsibility

One of the first decisions that must be made when designing a systemic solution to managing delay-tolerant data is how to divide responsibility. There are three major aspects: deciding what data to transmit, coordinating the use of shared resources, and monitoring the environment. In each case, the individual applications can be made responsible, or a third-party manager can assume the responsibility.

Deciding What Data to Transmit

In general, the closer a management decision is made to its stakeholders, the better the decision will reflect the stakeholders' concerns. According to this principle, selecting what to prefetch or delete is best decided by the user. Few users, however, want to spend time curating their data: the benefits generally do not justify the time investment. Indeed, given the inconvenience, anecdotal evidence suggests that most users will ignore the problem until a decision is acute. Thus, the end result of making the user responsible for scheduling data transfers is that little data will be prefetched, and data will only be deleted when the available free space is exhausted.

The best proxies of a user's desires are the user's applications: they have detailed, high-level information about the user's behavior, which they can use to estimate the utility of different actions. But, application developers have the same fundamental issue as users: extracting and processing the information needed to model the user's behavior has a high opportunity cost. Anecdotal evidence suggests that application developers want to spend their time implementing new features, not adding infrastructure; they prefer solutions that are quickly realized and good enough to ones that are great, but require a significant time investment. This behavior is rational and appears to be widely practiced as is evident from the use of high-level technologies, such as Python and Qt, to solve high-level problems. These technologies are theoretically less flexible than custom solutions, but are perceived as more than worth the loss of control due to

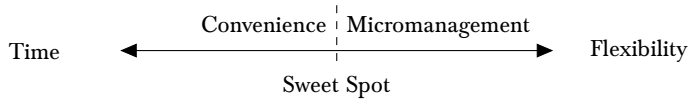


Figure 7.1: Depiction of the time-flexibility tradeoff. More flexibility results in additional overhead and a larger time investment. For most tasks, there is a sweet-spot to the left of which is convenience and to the right of which is micromanagement.

the adequacy of their solutions, the saved time, and the corresponding increase in productivity. Indeed, the solutions are not only adequate, but they strike a good balance between flexibility and micromanagement: at some point, too much flexibility becomes a liability; the point solution becomes too difficult to improve or adapt. This time-flexibility tradeoff is illustrated in Figure 7.1. Based on this line of argumentation, we speculate that most application developers will be happy to delegate scheduling decisions to a third-party manager—if the manager can be made to perform well enough.

The theoretically optimal performance of a third-party manager cannot exceed that of an application-specific solution, and it will often be significantly worse: the application has detailed, semantically meaningful information, which it can directly use, and which is not obscured by having been inferred or translated into some not-completely-appropriate format. Nevertheless, we suspect that a third-party manager will not only perform well enough in practice, it will perform better than most application-specific solutions.

To understand why a general-purpose solution will likely perform better than an application-specific solution, consider the significant amount of research on extensible operating systems in the 1990s [16, 32, 43, 45, 66]. The goal of these extensible systems was to allow applications to directly manage the resources, such as memory, CPU and storage, allocated to them. Although the evaluation of these systems showed that application performance could be dramatically improved, their proposed mechanisms have not been integrated into commodity operating systems. A possible explanation for the success of general-purpose solutions is that modelling behavior and scheduling resources is complex and only general-purpose solutions can attract a sufficient number of developers and users to justify this complexity. More developers and more users firstly results in more testing and more tuning. But, it also means that it is reasonable to consider more sophisticated algorithms. Further, those who work most on a general-purpose solution are more likely to be (or interested in becoming) domain experts.

To intelligently determine whether to schedule a pending transfer or delete an object, the scheduler needs to estimate when the object will be used, if at all. A strong indicator is when related objects have been used and in what context and when this occurred relative to their publication and download. Additional information is certainly useful, however, we suspect that the added value will be increasingly marginal. In a certain sense, this is positive: if applications are to provide the information, then limiting what is collected simplifies their job.

Coordinating the Use of Shared Resources

We want to schedule transmissions to occur when there is good connectivity. But, connectivity is not the only scheduling predicate we need to consider: if we don't take the data transfer allowance, the energy budget, and the available storage space into account, we will negatively impact data availability, the probability that desired data is accessible with acceptable latency without excessive cost. These resources need to be shared not only among the competing applications, but also with the user. This requires that the scheduling mechanism predict and respect the user's activities. Further, to maximize efficiency, the scheduling mechanism should also exploit synergetic scheduling effects.

There are two main ways to allocate resources: applications can use a peer-to-peer scheme and negotiate with each other, or a central manager can mediate access. An example of a well-known peer-to-peer management scheme is TCP's congestion avoidance algorithm. TCP's congestion avoidance algorithm fairly shares the available bandwidth among competing flows. It does not use a server to schedule the bandwidth or require that each agent directly negotiate with other agents. Instead, it uses dropped packets as an indicator of the available bandwidth. This works because bandwidth is used and then released; bandwidth is not consumed. Thus, the agents who are interested in using the bandwidth are actively competing with each other. When a transmission completes, the bandwidth it used is released, and can be immediately used by another flow. When a data transfer occurs via a cellular network, it also consumes some of the user's data allowance. The data allowance is not replenished when a transfer completes, but at the start of the user's next billing cycle: unlike bandwidth, the available data transfer allowance is dependent on past activity. This means that the application needs to determine the operation's importance relative to not only all concurrent operations, but all operations up until the resource is next replenished. This makes coordinating this type of agent significantly harder: unlike for non-consumable resources, not all competing agents are running simultaneously. A central server mitigates this problem: it has a global view of the system—applications must come to it for resources—and it can easily col-

lect historic data to help predict future demand. Further, the main appeal of a peer-to-peer mechanism on a single device is that it is more flexible: there is no central server that imposes some arbitrary policy. In practice, some *de facto* policy is required to facilitate agent negotiations. This mostly adds complexity and impedes the system's ability to agilely react to changing conditions. A solution based on a central server appears to be the dominate strategy in our case.

The scheduler doesn't just need to schedule registered transfers: it also needs to take the user's usage into account. Opportunistic transfers compete for the same resources as those that the user uses: the user needs energy to use her device; she needs to transfer data when spontaneously browsing the web, when forcing an update, and when using unmanaged applications; and, she needs space to save files that she creates or transfers. If the required resources are insufficient, because the scheduler was too aggressive, then the user is inconvenienced. For instance, if the energy is exhausted, she has to wait until she can recharge her device's battery; if the data allowance is exhausted, she needs to decide between incurring overage charges, and waiting until there is free Wi-Fi; and, if storage is exhausted, she needs to manually delete some files. This suggests that the scheduling algorithm needs to be conservative. However, if transfers are scheduled too infrequently, e.g., only when there is power and free Wi-Fi, the user will experience many cache misses, and unacceptably high latency for queued uploads. The scheduling algorithm needs to predict the user's resource usage, and balance the inconvenience of cache misses with the inconvenience of an exhausted resource.

The scheduler's job is made more complicated by the fact that the amount of energy required to transmit a bit of data over a wireless network is only loosely tied to type of network. There are two main reasons for this variability: the first is the large fixed costs, and the second is the quality of the connectivity. To transfer data over the cellular network, the mobile device must first establish a connection. This requires communicating with the base station. To avoid this signalling overhead for every transfer, the device remains in a high-power state for some seconds after the connection becomes idle. See Figure 7.2 for more details. This overhead can be amortized by large transfers, and the co-scheduling of small transfers. The other factor that influences the required energy is the quality of the connection: if the signal-to-noise ratio is low, the transmission power can be reduced, fewer retransmissions are required, and a higher-modulation scheme can be used. Schulman et al. found that transferring data with a weak signal requires up to six times as much energy as when the signal is strong [106]. To reduce the energy cost, transmissions should be scheduled to occur simultaneously to better amortize the fixed costs and to maximize the available bandwidth. Co-scheduling transfers is easiest when all transfers are

scheduled by the same entity. If there is no central scheduler, i.e., each application decides when to transfer data, it is still possible to introduce a small system daemon that can indicate when the network connection becomes active. The problem then is determining whether the connection is being used for interactive data, in which case a background transfer would interfere, or bulk data, in which case additional transfers are helpful.

The fixed costs include energy that is needed to establish the communication channel (the ramp-up energy), and the energy needed to maintain it even if no data is being transferred (the tail energy). The tail energy is incurred, because after a cellular transmission completes, the virtual channel is not released immediately, but retained for several seconds. (Balasubramanian et al. measured values of 6 seconds on AT&T's GSM network and 12 seconds on its UMTS network [14], however, the exact value depends on the network's configuration.) Retaining the channel for a short period of time eliminates the signalling overhead, and the latency of allocating a channel that would otherwise be incurred by subsequent transmissions. This optimization is important, because server and client interactions are often interleaved: the client sends a request (e.g., for a web page), the server replies, the client sends a new request (e.g., for some javascript files and some images), etc. Figure 7.2 shows a simplified version of how virtual channels are managed. Balasubramanian et al. measured the ramp energy to be between 2 and 3 Ws and the tail energy between 7 and 8 Ws [14]. Haverinen et al. measured the energy of a keep-alive packet on a 3G CMDA network (a keep-alive packet brings the mobile device from the idle or pagable state to the shared or active state and incurs the full tail energy) as being between 2 and 13.3 Ws, depending on whether the network supports the pagable state and the length of the timeout (they measured 2, 6 and 10 second timeouts) [46].

This consumption of energy—the ramp-up, tail costs and idle costs—suggests co-scheduling transfers. Co-scheduling transfers better ensures the saturation of the available bandwidth, and the amortization of the ramp and tail energy. Co-scheduling transfers is better than scheduling them back to back, because sometimes a transfer is unable to saturate the link. This can happen if the bottleneck is not the device's Internet connection, or simply due to processing time either on the server or the device itself. The more delay-tolerant transfers there are, the higher the potential degree of co-scheduling. This requires cross-application coordination, which again suggests the use of a third-party manager to schedule transfers.

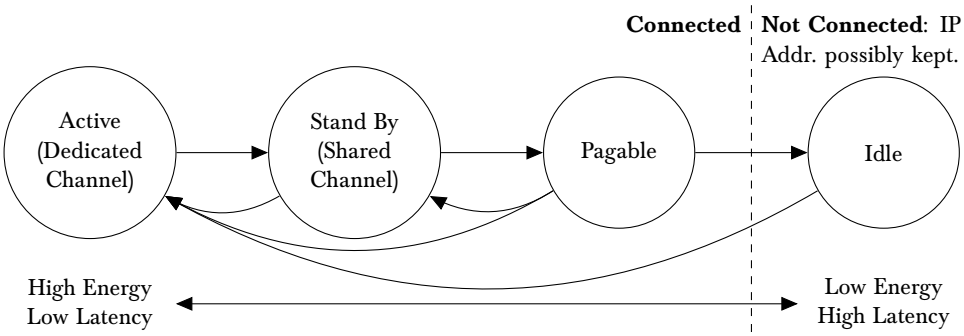


Figure 7.2: Summary of the radio resource control (RRC) protocol used in cellular networks [1, Ch. 7.1]. Transitions to low-power states are primarily controlled by inactivity timers. Many networks do not use the pagable state.

Monitoring the Environment

We want to schedule transfers to occur when conditions are good. A simple definition of good is a static threshold, e.g., Wi-Fi is available, the device has more than 30% of its energy remaining, and the user is idle. A better definition of good is one that considers when data is likely to be needed (i.e., its delay tolerance), and to transfer that data when the best conditions occur before that time. Whatever policy is used to determine good conditions, the scheduling agent needs to monitor the relevant sensors. If predictions are made based on historical data, as is likely required in our second definition, the scheduling agent needs to record the data and process it occasionally.

If every application does its own monitoring, every application needs to remain running to not miss an opportune moment. (If an application only starts periodically to check the current conditions, it cannot agilely adapt.) This wastes memory and CPU cycles if this technique is used by more than a few applications. Further, every application does the same work: similar code for monitoring the environment and processing the data must be developed and tested for every application. Using a central manager means that only a single entity needs to monitor these conditions. A central monitor can then start an application when it should transfer some data. This monitor can either use its own definition of good, or allow applications to describe when they want to be awoken. If desired, the monitor can also export the recorded history, or provide an interface for querying it.

Summary

In this subsection, we considered how to predict what data the user will need, how to coordinate the use of shared resources, and how to monitor the environment to exploit good conditions. In all cases, we argued that using a central manager is significantly better than making the applications solely responsible for the work. Specifically, a central manager appears to simplify the implementation, require less device resources, and result in better scheduling performance.

7.2.2 Transparency vs. Application Support

The most desirable solution to exploit opportunistic connectivity is one in which a transmission manager prefetches and delays uploads without requiring application modifications or user intervention. This can be done by manipulating the applications—reaching in and forcing them to perform updates and transfers. Unfortunately, applying this type of manipulation to an arbitrary program is equivalent to solving the halting problem, i.e., intractable in general.

A more realistic solution, which preserves transparency, is having the transmission manager intercept the application's network communications, and monitor the applications to understand the user's behavior. This approach requires that the transmission manager understand many protocols, circumvent encryption, trick applications into to revealing their authorization credentials, and trace processes to infer what the user accesses. This solution is complex and fragile.

Alternatively, applications can help. The minimum amount of support that a transmission manager requires to predict what to schedule, and to actually schedule it is: information about any pending transfers, when data is used (which is often readily available), and an interface to cause the application to initiate a transfer, and free storage (which should be relatively straightforward, since this functionality is normally already present). We focus on the minimum required support, because any changes are a burden on application developers. In practice, additional features can also be made available, but they should be optional.

Transparent Interposition

To prefetch data using transparent interposition, the transmission manager needs to determine what data is available to arbitrary applications, predict the data the user will likely access, fetch that data when connectivity is good, promptly inject the data into the corresponding application so that the user knows it is available, and manage the application's cache. For it to queue uploads, it must provide a local proxy server, which queues uploads when network connectivity is poor or

not available, make clear to the user that although the program claims the data has been sent, it is really only queued on the device, extract any required login details, and send the data to the real server when connectivity is good.

The first difficulty that an interposing transmission manager must overcome is speaking and understanding each service's protocol. Unfortunately, there are at least a handful of common protocols for each type of service. Blogs and podcasts are typically encoded using RSS or Atom, which is transferred over HTTP. Apple's iTunes, however, uses its own proprietary iTMS protocol. Similarly, there are a few common protocols for transferring Email: fetching mail is usually done with IMAP4, POP3, or MAPI and sending mail is done using SMTP. Supporting social networking services is more difficult: most services use their own protocol. Accessing weather information, and using backup services is analogous.

Supporting such a large number of protocols is a herculean task, and a single vendor cannot provide complete coverage if only because new services with their own protocols are emerging constantly. One possibility is to ship support for some common protocols, and provide a mechanism for third parties to add support for additional protocols. Because the protocol support must be relatively complete—it should work with all applications, and not just target the subset of features that a single application uses—adding support for a new protocol will not be easy, and the number of people who can write such plug-ins will be significantly less than the number of application developers. Another major difficulty is ensuring the correct *de facto* implementation of the protocol. Many application operate on top of HTTP, for instance, but these applications don't always strictly follow HTTP's semantics. In particular, the GET operation may not be an idempotent operation as HTTP specifies it should be: it is not unusual for a link to cause state to be updated, such as deleting an email message in a web mail program [26]. Application-specific solutions will automatically account for these quirks.

Assuming that the transmission manager supports the protocols that an application uses, the next problem is to coerce the application to use the proxy, and to expose its authorization credentials. Capturing the application's traffic can be done by redirecting the client's traffic to the monitor. On a Linux-based system, this is possible using IP tables to redirect network connections to an address on which the manager is listening, or by way of debugging facilities, e.g., ptrace. The manager then needs to determine the protocol in use, and hand the connection off to the appropriate proxy. A serious difficulty arises if the communication in encrypted: in this case, the manager needs to launch a Man-in-the-Middle Attack or interpose on the functions that encrypt and decrypt the data. The latter approach is not a general solution—this is again similar to solving the halting problem—but it is possible for many programs due to their use of a

known library, such as OpenSSL.

To determine what data to prefetch, the manager also needs to gather information about the user's behavior. Since the proxy understands the protocol, it can flag the data that the application fetches, and watch for it to be read from disk later. This can be done using something like taint analysis. This is, however, computationally expensive, and must be applied carefully as, e.g., the use of hash tables can result in false positives [109]. Further, just because data is read into memory does not mean that it is actually used by the user. For instance, data could be read for indexing purposes, to generate a preview, or due to a prefetch operation. Another complication is that the monitor cannot assume that the data will be read by the same program that wrote it. This could be because the application's user interface, and the network agent run in separate processes, or that a program has multiple front ends.

An additional obstacle to realizing transparent prefetching is determining how to inject prefetched data into an application. If the user doesn't know that the data is available, then he is less likely to access it. For instance, if the email proxy finds that there is new email and downloads it, the user will only find out about the email the next time the email client checks for new mail! One option is to place the burden on the user by requiring him to configure the application to check for updates more frequently and thereby minimize the delay. This approach has the disadvantage of increasing resource use, and requiring user intervention. Another possibility is to add application-specific knowledge to the manager to enable it to force the email client to check for new mail. Sometimes there is a way to invoke the program from the command line that causes it to check for new mail, for instance. For a podcatcher that only downloads podcasts on demand, injecting data is even more difficult: there is no way for the user to know whether a podcast is available locally. If he has a limited data transfer budget, he may not want to risk accidentally initiating a transfer.

Finally, the monitor needs a way to manage the application's storage. Applications that are conservative in their use of storage so as to not interfere with other applications and the user, will aggressively delete content. The manager needs to prevent this. On the other hand, applications that do not automatically delete old content will suddenly use more storage due to the inevitable prefetching of data that the user never uses. The manager needs to recognize and constrain this to prevent the application from exhausting the available storage. The problem is that the transmission manager cannot delete arbitrary files: a file may contain the only copy of the data.

Unfortunately, a transparent solution to exploiting opportunistic connectivity—even a non-general one—requires significant effort—many protocols need to be implemented—and it is wrought with uncertainty, because the user's behavior is

not easily inferred. Further, the benefits are sometimes reduced. For instance, users do not realize that data they are interested in is actually available locally; and, applications do not play along, e.g., by prematurely deleting content.

Application Support

Involving the application can significantly simplify the implementation of the transmission manager, ease the acquisition and increase the reliability of its knowledge, and improve the system's usability. This is not simply an exercise in moving complexity elsewhere: the applications can handle these issues in a fundamentally simpler and qualitatively better manner. However, because the application developers must cooperate, any required support needs be as easy as possible to implement.

The most important type of support that an application can provide is exposing hooks that allow the transmission manager to initiate transfers. Such a change alleviates the transmission manager from having to intercept the application's network connections, from having to understand and speak the protocol that the application uses, from having to circumvent any used encryption, and from having to extract the application's authorization credentials. With this change, when the transmission manager decides that the application should transmit some data, it just uses the hook to tell the application to do so. The burden placed on the application developer is relatively small: as the applications in question already transfer data, these hooks should be just a few lines of code, which look up the right data structures and call an existing function. This is significantly simpler than the tens-of-thousands of lines of code to proxy even relatively straightforward protocols. If high-level libraries can be used, the implementation effort required for the application developer to expose the hooks to the transmission manager can be reduced even more.

A second helpful change is providing information to the transmission manager. Applications often have information that is readily available that can help the transmission manager improve its scheduling decisions. This information includes: the pending transfers, their expected transmission size, their publication time, when they are transmitted, when the user uses them, and how they relate to other objects, e.g., emails from the same mailbox. Indicating the user's behavior allows the manager to detect how the data is used, e.g., whether objects in a feed are used serially, or whether only the newest is ever used, and whether data is likely to be fungible, e.g., an email vs. a podcast episode the latter of which can often be replaced by some other episode. Another advantage to having the application provide this information rather than trying to infer it from the application's execution is that the data is significantly more reliable. Although this

is a large list of information that applications could provide, the only essential piece of information is when some data is used.

An important concern with transparent interposition is how to inform the user that new data is available, or when data is actually uploaded. Because the application uses its usual mechanisms to perform the actual transfers, the application's state is always up to date, and the usual notification mechanisms keep the user current.

The final issue that applications can help with is managing storage. This can be facilitated by adding another hook that allows the transmission manager to prompt the application to free storage space. This allows the transmission manager to determine the storage space allocated to each application, and solves the problem of determining whether it is safe to delete a file: the application knows better whether data is precious, or whether it can be retrieved again, if needed. This hook may not be trivial to implement if the application does not already have facilities for purging files. An alternative strategy is to have the application indicate to the transmission manager the files which belong to each object, and whether they can be deleted.

A Hybrid Approach

Neither using a completely transparent manager, nor relying on application support is a completely satisfactory solution: the former is overly complex, and the latter requires support from application developers. The two approaches are fortunately not mutually exclusive: it is possible to provide transparent support for some protocols while also accepting application support. As already noted, some protocols, such as RSS and Atom over HTTP, are widely used and relatively straightforward to proxy. For these protocols, implementing a good proxy with reasonable effort is possible. For overly complex and lesser used protocols, application developers can be expected to help. Of course, it should still be possible to modify applications that use a supported protocol to directly use the transmission manager. This will likely result in better integration and improved scheduling.

7.2.3 Summary

Ideally, an external manager should assume the management responsibility for as much as possible: it has a global point of view, which improves its ability to schedule consumable resources, and because it is used by all applications, there is more interest in its success, which translates to more tuning and testing than specialized code. In practice, the manager needs some application support to

work efficiently. Most importantly, it needs a way to trigger transmissions, which is straightforward for applications, but a herculean task for a transparent manager. The requirements placed on the application should be as simple to fulfill as possible to encourage adoption: anecdotal evidence suggests that application developers do not want to spend time integrating with the platform, they want to implement features.

7.3 Transmission Manager Interface

In the previous section, we argued that a transmission manager should assume as much responsibility for the scheduling and the management of delay-tolerant transfers as possible. We observed that the scheduler is unable to do everything transparently, and argued that four types of application support are particularly useful: the application needs to indicate what transfers are pending, provide a mechanism to initiate transfers, report when data is used, and provide a mechanism to free storage. In this section, we present Woodchuck, an interface for a transmission manager that makes it easy for applications to fulfill these requirements. Note: our focus is on describing how the transmission manager and the applications interact, not the actual scheduling algorithms. We leave these for future work.

7.3.1 Architecture

The transmission manager is firstly a scheduler. It schedules transmissions, or rather, the low-level resources used by a transmission—the data transfer allowance, the energy and the storage—so as to maximize data availability. To do this efficiently, the transmission manager needs to determine when using resources yields a high return. This depends on environmental factors, such as network quality. The transmission manager also needs to estimate the pending transmissions' expected utility: it should invest resources in transmitting data that is likely to be used in the near future, and whose absence causes the most inconvenience. This requires understanding the user's behavior. These aspects are shown in Table 7.1 along with some examples.

7.3.2 Case Studies

To make the use of Woodchuck in existing applications as easy as possible, we studied some applications, which run on Maemo, and which transfer data that could either be prefetched or queued. The programs that we studied are: Modest [95], Maemo's default email client, gPodder [98], a podcast client, Feeding-

Managing Resources	Data Transfer Allowance Energy Storage
Monitoring the Environment	Network Connectivity Location Energy User Activity
Understanding Transmissions	Pending Transmissions Publication Time Time of Use

Table 7.1: The three main aspects of the efficient scheduling of transmissions, and some examples thereof.

It [79], an RSS reader, Khweeteur [49], a Twitter client, OMWeather [121], a weather report program, and Maemo's built-in application manager.

Our investigation revealed that there are generally two types of network transfers: updates of meta-data, and transfers of actual content, which normally doesn't change. When gPodder updates a podcast subscription, for instance, it downloads a list of the available episodes; the actual episodes are downloaded separately. This pattern is also found in the email program, the RSS reader, and the application manager. In contrast, when the Twitter client updates a view (the timeline, direct messages, a standing query, etc.), Khweeteur doesn't just fetch the list of new tweets, but simultaneously downloads the tweets. The reason for this is that tweets are not significantly larger than the metadata describing them. Thus, separating fetching the list of updates from the actual content doesn't make sense. Fetching embedded media, however, could be separated. OMWeather follows a completely different pattern: although a user subscribes to a weather station's forecast, forecasts are not assumed to be published regularly, but updated continuously.

Based on this analysis, a reasonable way to describe pending transmissions is to reify the *content* as *objects*, which are either immutable or updatable, and the *meta-data* as a *stream*, which describes the objects. Modifying programs to support these abstractions should be relatively straightforward. In gPodder, for instance, when the user subscribes to a new podcast, gPodder could register a new stream with a couple of lines of code. Similarly, when it updates a stream, and discovers new episodes, it could register new objects corresponding to the episodes. For OMWeather, a single object, which is marked a mutable, could

be used. The application manager would register a stream for the software updates. It would only register objects corresponding to software updates, not new software: it is most likely that the user does not want most new software.

All of the programs that we examined support automatic updates. Modest and the application manager enable automatic updates by default; the rest require that the user explicitly enable the functionality. All programs provide a way to configure how often updates should occur. The application manager, however, can only be configured programmatically; the rest provide a straightforward GUI configuration dialog. Only one program, Modest, provides an option to not update when using a cellular connection. This makes automatic updates dangerous for users with limited data allowances. This is confirmed by searching the Maemo forums: there are a number of posts[2] from upset users asking how to disable the application manager's automatic update functionality, because it exhausted their data plan.

There are two ways that the examined applications manage automatic updates: either the automatic updates are only performed when the program is running (gPodder), or the program runs constantly in the background using a daemon (Khweeteur) or a desktop widget (the rest). When automatic updates are enabled, the programs set up a timer using, for instance, `g_timeout_add`, if using Gtk+. When the timer fires, a callback function is invoked that iterates over the subscriptions and calls the appropriate update function. Given this structure, modifying these programs to perform updates in response to an upcall should be relatively straightforward: the functionality already exists.

Most of the programs that we examined already track usage information. For instance, FeedingIt shows articles that have been read in a different color from those that have not yet been read. When the user views an article, the front end calls a function that marks the article as read. Modifying this function to also report the usage information to the transmission manager would be straightforward. The other programs that support usage information have a similar function, and could be modified similarly. The exception is gPodder. gPodder provides a set of hooks that allow extensions to receive information about certain events including when the user plays a podcast episode. OMWeather does not track usage information, and, in most cases, it is difficult to extract: a typical interaction is via a desktop widget. Thus, to see the weather forecast, the user simply switches to the desktop. But, just because the user views the desktop does not mean that she has also viewed the weather report. In this case, we cannot easily learn the user's behavior. The application, however, provides an

[2]For instance, `http://talk.maemo.org/showthread.php?t=56099` and `http://talk.maemo.org/showthread.php?p=713619`.

update frequency configuration setting. This information can be used by the transmission manager to control the update frequency.

We observed two approaches to managing storage: the program either ignores the problem, and lets the user deal with it, or the program automatically deletes old data. Some programs do not need to manage storage: the storage they use does not grow with time. This is the case for the application manager, which does not keep information about old updates, and the weather program, which only keeps the most recent forecast. gPodder lets the user deal with the problem. It provides a button associated with each podcast episode that the user can press to delete the data. This mechanism is inconvenient when the goal is to free space—too many interactions are required. Anecdotal evidence suggests that users simply use the file manager to delete all episodes associated with some podcasts when space becomes scarce. Modest, FeedingIt and Khweeteur take the alternate approach: they only keep recently downloaded data. Modest keeps the 250 most recent email messages by default; FeedingIt keeps articles as long as they are referenced in the feed's summary file plus a short period of time; and, Khweeteur takes a hybrid approach: it keeps the status updates from the last, e.g., week or at least, e.g., 100 status updates, whichever is greater.

This analysis suggests two storage management mechanisms. First, the transmission manager could make an upcall to indicate what object to purge. This is good for applications that have no existing storage management mechanism. Second, the transmission manager makes an upcall that triggers existing application facilities to free some space. Strictly speaking, applications that already manage their own storage don't need to be modified: they won't interfere by causing an out-of-error message, but they also won't profit from additional storage space, which could increase their cache hit rate.

7.3.3 Platform

We assume that the transmission manager can run as a daemon in the background, can start applications and can communicate with them using some interprocess communication (IPC) mechanism. Of the commonly used smartphone operating systems in the past decade—Google's Android, Apple's iOS, RIM's Blackberry OS, HP's WebOS, Microsoft's Windows Mobile and Nokia's Maemo/Meego—only iOS fails to fulfill these requirements: iOS does not support application multitasking, which would prevent the transmission manager from starting applications to perform transmissions.

To make the following discussion more concrete, we assume that D-Bus is used for IPC. D-Bus is widely used in the GNU/Linux world: it is the main IPC and scripting mechanism for both GNOME and KDE, and is also used by Nokia's

Maemo/Meego smartphone operating system, which our prototype implementation, Murmeltier, targets. In addition to providing an IPC mechanism, D-Bus starts an application if a message is addressed to it, and it is not running. This requires that applications have persistent, unique system-wide identifiers. D-Bus has a naming convention that ensures this. Systems that don't use D-Bus mostly likely have some IPC communication mechanism that can be used as the basis to fulfill these requirements.

7.3.4 API Overview

To achieve its goal of efficiently scheduling transmissions, Woodchuck requires that applications help the transmission manager in four ways. Applications need to register pending transfers; perform transfers in response to an upcall; report how the user uses the data; and, release storage space in response to an upcall. In this section, we provide a brief overview of the API that we developed based on the case studies. In the remaining sections, we go into more detail.

Registering Transfers: Woodchuck requires that applications register any pending transmissions. It provides two core abstractions: objects, which are registered using **object_register** and streams, which are registered using **stream_register**. An object represents data that is directly used by the user, e.g., a podcast episode or an email; and, a stream represents a subscription, e.g., a podcast subscription or an email account. In addition to providing a schedulable entity, these abstractions reveal how data is structured and related, which allows the transmission manager to more easily apply knowledge about an object or a stream to other objects and streams.

Transferring Data: To decide when to transfer data, the transmission manager monitors the environmental conditions. When it detects that conditions are good relative to the urgency of the pending transmissions, it schedules some pending transfers. By running in the background, and processing data in real-time, the transmission manager is able to agilely react to unexpected conditions.

To actually transfer data, the transmission manager makes an upcall, either **object_transfer** or **stream_update**, to the application. The application is expected to promptly perform the transmission, and report the result to the transmission manager using **transfer_status** or **update_status**. This is needed so that the transmission manager knows whether the transmission was successful. If the application reports an error, the transmission may be retried later.

Woodchuck uses this application interaction as an opportunity to optionally acquire some other useful information, which is likely available. The application can provide some statistics about the transfer, such as the amount of transferred

```
Top-Level:
  manager_register (human_readable_name, cookie, dbus_name) → UUID
  list_managers () → [managers]
  lookup_managers_by_cookie (cookie) → [managers]
Manager Methods:
  unregister ()
  /* FRESHNESS: How often to update the stream. */
  stream_register (human_readable_name, cookie, freshness) → UUID
  list_streams () → [streams]
  lookup_streams_by_cookie (cookie) → [streams]
Stream Methods:
  unregister ()
  /* VERSIONS: Array of object size and utility
      corresponding to each version of the object. */
  object_register (human_readable_name, cookie,
    publication_time, version) → UUID
  list_objects () → [objects]
  lookup_objects_by_cookie (cookie) → [objects]
  /* INDICATORS: Bitmask of indicators shown to user,
     e.g., audio, vibrate. */
  update_status (status, indicators, transfer_size,
    transfer_duration, new_objects)
  viewed (time, duration, use_mask)
Object Methods:
  unregister ()
  /* FILES: Array of files containing the object and
      their respective deletion policy. */
  transfer_status (status, indicators, transfer_size,
    transfer_time, transfer_duration, disk_space, files)
  /* USE_MASK: bitmask of the used parts of object. */
  used (time, duration, use_mask)
  files_deleted ({DELETED, LATER, COMPRESSED}, NEW_SIZE)
Generic:
  property_set (name, new_value)
Upcalls:
  /* Respond with stream.status_update. */
  stream_update (manager, stream)
  /* VERSION: Index into VERSIONS arg of object_register.
     Respond with object.object_transfered. */
  object_transfer (manager, stream, object, version,
    quality_bandwidth_tradeoff)
  /* Respond with object.files_deleted. */
  object_delete_files (manager, stream, object, amount)
```

Figure 7.3: Woodchuck's API

data and the transfer duration. This information allows the transmission manager to better manage the data allowance and energy: it knows how much data was transferred, and can better estimate how much related transmissions are likely to require. The application can also describe the transferred data: it can provide the amount of space the object occupies, the files that contain the data, and whether the transmission manager may delete the files without consulting the application. This information enables the transmission manager to better manage the storage space.

Understanding User Behavior: To learn the user's preferences, Woodchuck has applications report when and how objects and streams are used using **used** and **viewed**, respectively. This mechanism provides detailed, reliable information about how the user uses objects and streams, which the transmission manager can use to better determine whether to retain the data when space becomes scarce, and to better estimate the utility of related pending transmissions.

Reclaiming Storage: When the transmission manager detects that there is an insufficient amount of storage space, it purges some objects. To decide what to discard, it estimates the utility of the objects based on their use and the use of related objects, and removes those with the lowest utility.

Woodchuck provides two ways to reclaim the space. If the application indicated that the transmission manager can discard the associated files when it reported the transfer, then the transmission manager does so. Otherwise, the transmission manager makes the **object_delete_files** upcall to the managing application requesting that it free the object's storage.

Because the transmission manager runs in the background, it can promptly detect when space is running low. This allows the transmission manager to aggressively use the available space without worrying that the user will see an annoying out-of-space error message when, for instance, copying a large amount of data to the device.

7.3.5 Abstractions: Objects, Streams and Managers

Woodchuck uses three abstractions: objects, streams, and managers. Objects are transmissions (either uploads or downloads), streams represent subscriptions; and, managers loosely correspond to applications. These three types of objects form a hierarchy: managers contain streams, and streams contain objects. This relationship is depicted in Figure 7.4

An object represents a transmission. Objects do not disappear when they are transferred: they continue to exist to record used storage space, and to collect use information. By default, an object is assumed to be immutable. This is the usual case for many types of objects: podcasts, blog articles, social

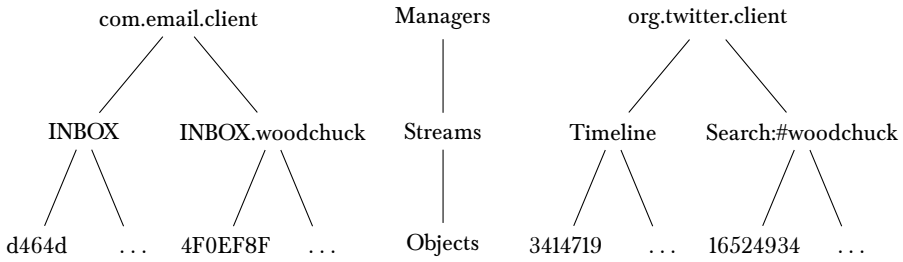


Figure 7.4: Woodchuck's object form a hierarchy. Managers are at the root, streams are in the middle and objects are at the leaves.

network status updates, etc. are typically published in their final form. Should an update become available, an application can indicate that the transmission manager should reschedule the transmission by setting its **NeedUpdate** flag. If an object receives updates regularly, such as an object corresponding to a weather forecast, then the application can cause the object to be transmitted periodically by setting the **TransferFrequency** to a target update interval. This is a hint and the transmission manager can adjust this internally based on when updates arrive, and how the user uses the object.

A stream corresponds to a subscription or a part of a subscription, which can be updated as a single logical entity. A stream update fetches the current stream state, which describes the available objects. The set of available objects is presumed to change with time. As such, a stream, unlike an object, is updated periodically by default. The default update interval is determined by its **Freshness** property, which is set by the manager. This value is internally updated based on how objects arrive and the user's behavior, as observed via its containing objects' use. An email program could use a stream to represent an email account, or use one for each mailbox belonging to that account. A Twitter client could use a stream for the twitter account, or one for the user's timeline and another for each standing query. A finer separation is useful if the entities they correspond to are likely to have different access patterns.

A manager represents an entity that has delay-tolerant transmissions. Typically, a manager corresponds to a program or an application (for expository simplicity, we often assume that this is the case), however, it is conceivable that a single application could use multiple managers, or that multiple programs collectively use a single manager. When a stream update or an object transmission is scheduled, an upcall is sent to a program associated with the manager (as specified by its **DBusName** property) indicating that it should perform the transmission.

Type	Property
Object	Cookie
	HumanReadableName
	TransferFrequency
	NeedUpdate
	PublicationTime
	Versions
Stream	Cookie
	HumanReadableName
	Freshness
Manager	Cookie
	HumanReadableName
	DBusName

Table 7.2: Object types and their principle properties

Table 7.2 lists each type of object's properties. These are described in detail below.

Naming and Designation

When an object, stream or manager is registered, Woodchuck assigns it a unique identifier. The application can use this identifier to refer to the object. However, requiring applications to use Woodchuck assigned names is a burden: applications then need to maintain a mapping between Woodchuck identifiers and local identifiers. Further, this mapping needs to be saved to disk so that the application can continue to address the objects after it is restarted. Woodchuck avoids this problem by allowing applications to use their own naming schemes: when creating an object, the application can set a so-called *cookie*, which can later be used to identify the object.

Choosing a cookie for streams and objects is easy: most applications already have an easy way to identify their objects. For instance, when a podcast is initialized, some directory may be reserved for its episodes. Determining the directory given the podcast is easy. In this case, the directory's name would be an ideal identifier for the stream. If the stream or object is saved in a database, the main entry's primary key could be used. The identifier could also be some real property, such as the stream or object's URL. In this case, the application needs to be careful: if the URL changes, then it needs to update the object's

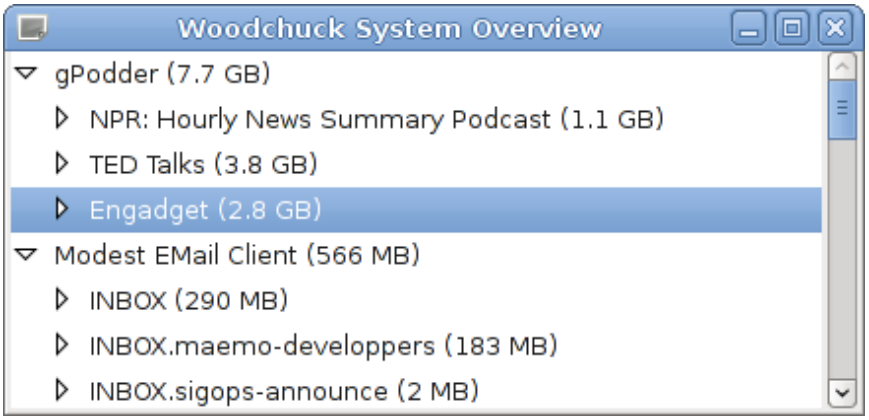


Figure 7.5: A system overview application. The application shows the user what objects are registered and how much space each one uses. This helps improve the system's transparency.

identifier.

Because names only need to be unique with respect to other objects in the containing object, i.e., stream names only need to be unique with respect to other streams in the same manager, the application can easily ensure that its chosen names are unique. This is not the case for managers: the manager's namespace is shared by all applications. To ensure that application-assigned identifiers are unique, a convention is required. Using the program's name is not a very good convention: programs may have generic names, which are prone to namespace collisions. To avoid this problem, D-Bus requires that bus names use a reversed domain name (like Java), e.g., org.woodchuck. Woodchuck uses the same convention. This convention is particularly good, because this information is already required: the application needs to provide its D-Bus name to the transmission manager so that it can receive upcalls.

Human Readable Names

When registering an object, the caller must provide a so-called *human readable name*. The human readable name, as opposed to a machine readable name, is a label that the user recognizes as referring to the object. This information can be used by a management program to show how much disk space each manager uses, and to break it down by stream and objects. This idea is illustrated in Figure 7.5. Such dialogs are not strictly necessary—the transmission manager should automatically handle all of the management without the user's input—

however, some users are wary of systems that are not transparent; they want to understand what the system is doing. This type of program increases the system's transparency, and removes the magic factor.

Choosing a human-readable name for an object is usually easy. Often, an appropriate label is included in the resource itself. For instance, an RSS feed update usually includes the location as well as the title and a description of each referenced object. Similarly, when checking for new email, the summary information includes each new mail's identifier as well as some of the mail's headers, including the subject line, sender and date.

For streams, the human-readable name will often be derived from the configuration data and the stream's meta-data. For instance, when subscribing to an RSS feed, the user typically only provides the feed's URL. A feed update usually includes the feed's title. An appropriate name for a stream corresponding to a mailbox is the email address plus the mailbox's name (e.g., inbox). When configuring an email account, the user provides the email address. The list of mailboxes is downloaded from the mail server.

Update Frequency

When registering a stream, the application needs to indicate how often the transmission manager should schedule an update. Likewise, for objects that are updated, the application needs to specify how often to synchronize the content. An appropriate value is a function of how much the user is willing to tolerate an out-of-date view and how often new updates arrive. These parameters are best learned from historical data. Until this data is available, the transmission manager needs some reasonable default.

A reasonable update interval varies greatly with the type of content. For instance, email typically arrives throughout the day and users want to be informed about new mail promptly. This suggests checking for new mail approximately every half an hour. In contrast, for podcasts accessed via the Zune network, only 30% of podcasts publish a new episode at least weekly, just 10% publish a new podcast at least daily, and only 1% of podcasts are used on the day they are released [39]. Given this, checking for new podcast episodes daily is likely sufficient. Application developers often have a feel for the arrival rates and access patterns of the content that they manage and thus can be expected to provide an acceptable initial estimate of how often to check for updates.

As the transmission manager observes when objects become available, how the user uses the objects and whether the user ever forces any updates, the update interval parameter can be adjusted. The transmission manager can infer the arrival rate from when objects become available: a newly discovered object

must have been published prior to when it was discovered but after the previous check for updates. This estimate can be improved if the transmission manager knows the actual publication time. Often, this information is included in the object's metadata. For instance, RSS feed data often includes when objects were published, and emails contain the time that they were received by the user's server. The upper bound on the update interval is a few days: performing an update when there is power and unlimited Wi-Fi is essentially free, which is likely to occur at least this often.

Handling Multiple Versions of an Object

Many objects have multiple versions. There are two main reasons for this. First, there may be distinct versions of an object, for instance, a video may be encoded with different bit rates or using different formats. This is the case on YouTube, which provides four encodings for many videos: 240p, 360p, 480p and 720p. Second, an object may be divided into multiple parts, each of which improves the object's quality or completeness. For instance, email servers typically support downloading an email's body and attachments separately. Likewise, blog posts sometimes contain images and tweets are sometimes just links to real content, e.g., images or web pages. Even though transfers occur in the background, a lower quality or incomplete version may be preferred to conserve the data transfer allowance or energy. Clearly, an email is less useful without its attachments, however, a lower-quality version is better than no data, and can help the user decide whether explicitly downloading the rest of the object is worth the cost.

By describing the available versions, the transmission manager can better budget the available energy and the data transfer allowance. First, the transmission manager knows the approximate amount of data that will be transferred a priori. Second, it can choose among multiple versions. Each version of the object is described by an array, which includes: the amount of disk space required after the transfer completes (which is negative in the case of an upload), the approximate number of bytes that will be uploaded, and downloaded when transferring that version of the object, the version's utility, and whether the version is a complete version or whether a version with a higher utility should still be considered for transmission if this version is transferred. When the transmission manager schedules an object for transmission, it includes the version of the object that the application should fetch.

If the available versions are difficult or impossible to enumerate a priori, the application can still adapt: when the transmission manager invokes the **object_transfer** upcall, it includes a quality-bandwidth tradeoff parameter. This parameter has a value between 1 and 5: 1 indicates that the lowest quality version

should be downloaded, and 5 indicates that the highest quality version should be downloaded.

Sending Upcalls

When the transmission manager schedules an action, it makes an upcall to the appropriate application. If the application is not running, it first starts it. Using D-Bus, the transmission manager can accomplish both of these things with a single piece of information: the application's D-Bus name. Its value is stored in the manager's **DBusName** property. When the transmission manager has an upcall to send, it directs the message to the responsible manager's D-Bus name. The D-Bus daemon delivers the message to the application that has registered that name. If the application is not running, it first starts the application.

If D-Bus is not used, a more complicated method may be required. For instance, if no stable rendezvous point can be arranged, the application may have to register with the transmission manager when it starts. When an upcall is directed to it and it is not registered, the transmission manager could start it by running some specified executable.

7.3.6 Transmissions

Woodchuck relies on applications to execute and report information about transmissions. When the transmission manager decides that a stream should be updated or an object transferred, it makes a **stream_update** or **object_transfer** upcall to the managing application, as specified in the stream or object's manager. In the case of an object transfer, the transmission manager indicates which version of the object to download and a quality-bandwidth-tradeoff parameter. In most cases, no significant application changes should be required: the application simply looks up the right data structures and calls an existing function.

After attempting the data transfer, the application is expected to report the result using **update_status** or **download_status**, as appropriate. In addition to specifying whether the transfer was successful, the application may indicate how the user was notified, if at all, e.g., by way of an audio sound or a desktop notification. The transmission manager can use this to better estimate when the user likely becomes aware of a new object. The application can also specify the amount of data actually transferred, which improves tracking of the data allowance. This isn't easy to compute externally. For instance, even if the transmission manager interposes on the application's sockets and counts the number of bytes transferred, a connection is often reused for transferring multiple objects. Sometimes, multiple objects are even interleaved. Finally, the application

can indicate what files are associated with the object and their respective deletion policies.

Note: an application should still update a stream or transfer an object when the user indicates it should do so: the transmission manager should not pretend to know better than the user; its goal is to support the user. In such cases, the application should still call **update_status** or **download_status** to indicate that it transfered an object: this enables the transmission manager to improve its model of the user's behavior.

The API encourages, but does not require, programmers to separate transmissions of the stream's meta-data from the transmission of actual content (e.g., updating a mailbox's index vs. downloading new e-mails). This separation better enables the manager to manage the data transmission allowance: if transmitting data is expensive, the manager can decide to synchronize the meta-data, but delay fetching the objects. In this case, the user is still informed about the arrival of new data, e.g., new e-mails, and can choose to explicitly authorize the potential overage charge to download the content. In such cases, the application should still call **transfer_status** to tell the manager that the object was transfered. This not only prevents the manager from scheduling the transfer, but allows the manager to adjust its model of the user's preferences.

7.3.7 User Behavior

To help the transmission manager understand the user's behavior, applications can indicate when and how the user accessed managed data. Woodchuck provides two functions for reporting use: **used** and **viewed**, which are used for indicating that an object or stream was accessed, respectively. These functions allow the application to specify when the access occurred, how long the object or stream was used, and the parts that were used.

Knowing when objects are accessed allows the transmission manager to infer how important it is to have an up-to-date view of a stream: how soon an object is accessed after the user becomes aware of its existence is an indicator of the delay tolerance of its containing stream. (Because it is difficult to determine when the user learns about the existence of an object, the time the object was transmitted can be substituted. Consider an the email program that causes an LED to flash when there is new mail. If the user does not act immediately, is it because the user ignored the indicator, or because the user did not see the flashing LED?) If a stream is not delay tolerant—if it contains many objects that are accessed soon after they are downloaded—it should be updated promptly even if the update is relatively expensive. For many users, this will likely be the case for their inbox: these users want to know about emails as they are received, and will read them

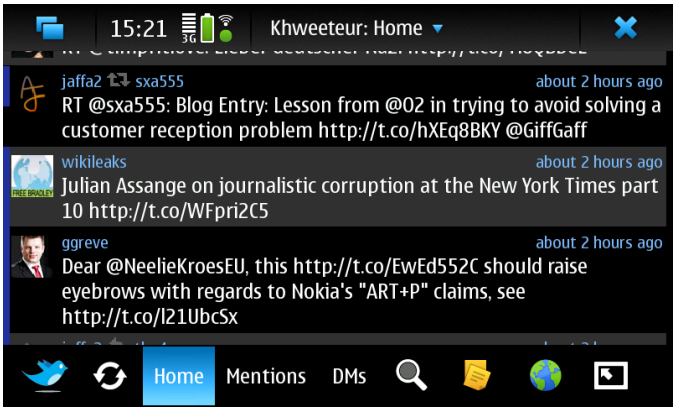


Figure 7.6: A Twitter client usually displays status updates in their entirety, which makes determining whether the user actually used them difficult.

shortly after they learn about their existence. If a stream is delay tolerant—if few objects are accessed shortly after they are downloaded—updates can wait until conditions are good. This is likely the case for data used to pass the time, such as entertainment. For such streams, a slightly stale view is often acceptable.

A stream's access pattern can be inferred from the access times of its containing objects. If the user only ever accesses the newest available object, then old data has little value and need not be transferred. This is likely the case for news-like data: only the most recent hourly news report is of any value. A variation of this is a news feed: if an article is not read within a certain amount of time, it is unlikely to be read at all. If the user accesses a stream's objects in order, then the content might be a series in which case, downloading newer objects should not be a high priority if there are still unused objects.

Applications can specify how an object was used, which helps distinguish multiple uses from a single multi-session use, i.e., the user uses part of the object and then later resumes using the object where she left off. This is important as if an object is used multiple times, the likelihood that it will be used again is high and thus it should be retained. If an object is used in its entirety just once, we know that after the user has completed using the whole object, that the utility of keeping the object is likely low. Applications communicate this information by passing a 64-bit bit mask to the **used** function. The first bit corresponds to the first 64th of the object, the second bit to the second 64th, etc. The bits do not necessarily refer to how the data is laid out on disk, but the likely order in which the parts of the object will be used. This is an important distinction: some formats, such as PDF, are random access.

Applications are not required to indicate when the user accesses data: if providing the information requires so much effort that the application developer chooses not to use the transmission manager, then we have failed. This can happen if it is not obvious whether the user accessed data. For instance, the user does not normally explicitly use a Twitter status update: when the user's timeline is displayed, all status update are displayed in their entirety. This is shown is Figure 7.6. The application could determine whether a status update was actually displayed to the user, however, because the GUI toolkit usually takes care of rendering, getting this information can be difficult. In this case, the application can still use the **viewed** function to indicate that the stream was used. Getting use information can be even more difficult if support for the transmission manager is added externally, perhaps by a user because the developers are not interested. This is perfectly reasonable if the application provides a plug-in interface or a scripting mechanism. These interfaces might not provide use information, however. To work around this limitation, the transmission manager can check when the application is run. This is likely an indicator that the data is being used.

7.3.8 Reclaiming Storage Space

Woodchuck provides two mechanisms to reclaim space: discardable files and an application upcall, **object_delete_files**. A discardable file is a file that the transmission manager can delete without consulting the application. Discardable files are registered when an object is downloaded: when the application calls **transfer_status**, it indicates not only what files correspond to the object, but also a deletion policy for each file. The upcall requests that the application remove the files associated with a specified object.

Discardable files are useful, because the application does not need to be started to reclaim space: when the transmission manager decides that an object should be purged, it simple deletes the associated files if they are discardable. Further, no additional application support is required. We expect discardable files to be common, because applications already need to gracefully handle data files that disappear. This happens, for example, when a user reclaims space using the file manager instead of the application provided interface. This approach doesn't cause a problem, because applications typically keep the metadata separate from the content, e.g., the list of podcast episodes is stored in a database. Should data later be needed, the application can usually redownload it.

Some objects do not lend themselves to being stored as individual files. For instance, an application might keep all data in a single database. In this case, Woodchuck relies on the application to disentangle the objects and free disk

space. An application indicates this by specifying the `deletable` deletion policy. For such files, Woodchuck invokes the **object_delete_files** upcall to request that the application delete the object. As usual, Woodchuck starts the application if it is not running. If the application frees an insufficient amount of space, Woodchuck must accept this; Woodchuck must not delete precious user data. If space is exhausted, the user must intervene.

As already noted, there may be multiple versions of an object or an object may consist of multiple parts. Thus, it may be reasonable to delete just part of an object, for instance, an email's attachments but not the body. In this case, the application responds to a deletion request using **object_delete_files**, but specifying the `compressed` option and the new size.

If the application receives a deletion request, but decides that deleting the files associated with the specified object is not desirable, due, perhaps, to some application-specific knowledge, it can indicate that it refuses to delete the files right now by responding to the transmission manager with the **object_delete_files** and setting the `later` option.

An alternative mechanism to reclaim storage space is to tell the application to free a certain amount of space, but let it decide what to delete. This is useful if application-specific knowledge easily provides substantially better performance than what an external manager can achieve. This appears to be against our argument that application-developers should not be bothered with implementing such functionality, because they don't have the desire to do a good job. In this case, however, some applications have already implemented similar functionality, which may be easier to reuse than to restructure to support Woodchuck's mechanisms. To support this, when making an **object_delete_files** upcall, Woodchuck includes the amount of space it wants the application to free. The application can ignore Woodchuck's suggestion and delete some other object. Of course, for whatever object the application chooses, it should still call **object.files_deleted** so that the transmission manager knows what is deleted, and the application's actual storage space consumption.

7.3.9 Summary

We have presented a transmission manager API call Woodchuck. In developing Woodchuck, we strived to minimize the required changes to applications and to make any required changes as non-invasive as possible. We motivated four application change: registering objects and streams, handling updates and transfer objects on demand, registering object use, and deleting the files associated with an object. Only the first two are strictly required.

7.4 Evaluation

In this section, we examine how much effort is needed to integrate Woodchuck support into an application. To do this, we implemented a Woodchuck server called Murmeltier as well as a C convenience library and Python modules. We adapted four programs to use Murmeltier and wrote one from scratch. We start by describing how an application uses Murmeltier. We then address how an application stays in the background when doing background updates. Finally, we discuss the effort required to use Woodchuck.

7.4.1 Supporting Woodchuck

To fully take advantage of Woodchuck, an application needs to do four things. An application needs to *register streams and objects* with the Woodchuck server so that it can schedule their transmission; implement an upcall to *transmit data* on demand, and report whether it was successful; *register stream and object use*; and, implement an upcall to *delete objects*. The minimum requirements are that the application register its streams and objects, and inform the transmission manager when streams are updated and objects are transferred.

Accessing Woodchuck

There are two main ways for an application to access a Woodchuck server. First, an application can use the D-Bus interface. This is a relatively low-level approach insofar as the application must manage a number of details, such as explicitly arranging to receive upcalls, and sending and receiving messages. Alternatively, an application can use a library that provides a more convenient interface and is more tightly integrated with the runtime environment. This typically reduces programming and debugging effort. The tradeoff, however, is a loss of flexibility due to the library's assumptions, but this flexibility is rarely needed in practice.

Because high-level libraries are specialized for a particular programming environment, an appropriate library needs to be written for each environment. Although a vendor may support one or a few programming environments, the developer community may choose to use others. Table 7.3 shows the diversity of programming environments used by applications on Maemo: there are five common programming environments (Qt on C++ or Python, Gtk+ on C or Python and SDL). Gtk+ on C is the only officially supported programming environment, and, interestingly, it is not the most popular. To ensure that community-driven development is successful, vendors should ensure that their platforms expose low-level, programming-environment agnostic interfaces for all essential services.

Environment	Packages	Percent
All	1886	100.0
Qt, C++	419	22.2
Python	374	19.8
Gtk+	182	9.6
Qt	125	6.6
Pygame	27	1.4
PySide	19	1.0
Gtk+, C	353	18.7
SDL	148	7.8
Perl	10	0.5
Ruby	4	0.2
Other	638	33.8

Table 7.3: Programming environments used on Maemo (specifically, the main application repository) according to the application's dependencies. Note: some packages list multiple programming environments. "Other" consists primarily of desktop widgets, application plugins, command line programs, and packages that don't correctly list their dependencies.

```
1  from pywoodchuck import PyWoodchuck
2  import woodchuck
3
4  # Connect to Woodchuck. Register the application, if not yet known.
5  wc = PyWoodchuck("podcast client", "org.podcast")
6
7  if wc.available():
8      # List registered streams.
9      for stream_id, stream in wc.items():
10         print wc[stream_id].human_readable_name, stream_id
```

Listing 7.1: Basic Woodchuck initialization and listing registered streams

We've written two high-level convenience libraries: a GLib-based C library (GLib provides Gtk+'s low-level functionality) called libgwoodchuck, and a set of Python modules called PyWoodchuck. The C library integrates with the GLib main loop and uses more convenient data structures. PyWoodchuck is toolkit agnostic and works with Gtk+, Qt and PySide. To facilitate adoption, the Python modules provide an interface that uses common Python idioms. Python is a high-level object-oriented programming language, which makes aggressive use of operator overloading, e.g., hashes are a language feature and hash-like objects are encouraged to mimic their behavior. The examples in this section use the Python modules for accessing a Woodchuck server. Using the C library is comparable, but is more verbose given C's lower-level nature.

To access a Woodchuck server from Python, an application uses the PyWoodchuck class. The constructor takes two arguments: application's human readable name, which is typically the application's name, and its D-Bus name. When the constructor is run for the first time, these are used to register and configure a new manager. The D-Bus name is also used as the application-assigned identifier, which, according to D-Bus's conventions, is guaranteed to be unique.

Consistent with many Python objects, the PyWoodchuck object acts like a hash. It implements Python's dictionary interface and maps stream identifiers to Python objects that represent Woodchuck streams. Similarly, PyWoodchuck's stream objects act like hashes and map object identifiers to Python objects that represent Woodchuck objects.

Listing 7.1 shows how to instantiate a PyWoodchuck object. This form of initialization suppresses upcalls: the application must register callback functions with PyWoodchuck. This will be covered shortly. The code also lists any registered streams. Note that the application only lists the known streams if the Woodchuck server is actually available. Consistent use of this idiom allows Woodchuck to be a soft dependency. This is useful for supporting platforms on which a Woodchuck server is not installed by default.

Registering Streams and Objects

When the user subscribes to a new stream or the application becomes aware of a new object, the application should register it with the Woodchuck server so that it can promptly schedule the stream's update or object's transmission, respectively.

Before an application can register streams or objects, it first needs to decide how to use these abstractions. For an email client, associating an object with each email seems sensible. For streams there are two choices: a stream could be associated with an email account or with a mailbox. Associating a stream with

a mailbox is better as mail within a given mailbox often exhibits similar access properties—the point of a mailbox is to sort the mail according to some features. For instance, a user may promptly check her inbox, but may only sporadically read mailing lists.

Unfortunately, an appropriate mapping to streams and objects is not always as straightforward as for an email client. Consider a program that displays the weather forecast, and a weather service that provides continuous updates. Associating an object with a weather update results in a infiniate number of updates—a new one every instant. The user doesn't need all the updates, just a recent update (and, not necessarily the latest one!). For the Woodchuck server to schedule this sensibly, the application should use a stream that is marked as never receiving updates, and a single object that is marked as updatable.

To register a stream, the application must choose an appropriate human-readable name, assign an identifier, and determine how often the stream should be updated (its target freshness). Registering an object is similar. In addition to providing a human-readable name and assigning it an identifier, the application can provide information about the available versions of the object. If there is more than one version, the application needs to indicate its utility. It can also indicate each version's size, and how much data is likely to be transfered during its transmission. If known, the application should also provide the object's publication time.

Listing 7.2 shows how a podcast-like application might update a stream. The application uses feedparser, a Python module to fetch and parse feed data. If necessary it then registers the feed with Woodchuck. It uses the feed's metadata to choose an appropriate human-readable name (the feed's title) and application-specified identifier (the feed's URL). The freshness parameter is simply assigned a value of one day, because this seems like a reasonable initial update interval for a podcast feed. Recall: the Woodchuck server can update this parameter as it learns when updates arrive and the user's behavior.

The application then iterates over the feed's entries. If the entry does not contain an enclosure (a link to a resource, which, in this case, is presumably a podcast), the entry is skipped. Otherwise, the application checks if the object is registered. If the object is not yet registered, the application extracts useful information from the available metadata, in particular, the publication time and the size of the object. If the size of the object is available, the application uses this to estimate the expected transfer size. This is typically the object's size plus some protocol overhead. In this case, the overhead is due to any DNS lookups and HTTP and TCP/IP overhead, which we assume to be approximately 5% of the object's size (20-byte IPv4 header, 20-byte TCP header, TCP options and ACKs for approximately 1500-byte packets).

```
1  import sys
2  import feedparser
3  from calendar import timegm
4  from pywoodchuck import PyWoodchuck
5  import woodchuck
6
7  wc = PyWoodchuck("podcast client", "org.podcast")
8
9  feed = "http://downloads.bbc.co.uk/podcasts/worldservice/docarchive/rss.xml"
10 d = feedparser.parse(feed)
11 if d.get('status', None) != 200:
12     raise d.bozo_exception
13
14 if wc.available() and feed not in wc:
15     wc.stream_register(
16         stream_identifier=feed,
17         human_readable_name=d.feed.title,
18         freshness=24 * 60 * 60)
19
20 for entry in d.entries:
21     if not entry.enclosures:
22         # There is nothing to download. Ignore this entry.
23         continue
24
25     object_id = entry.link
26     if wc.available() and object_id not in wc[feed]:
27         # Expected transfer amount is a function of the expected object size and
28         # protocol overhead (resulting, in this case, from HTTP and TCP/IP).
29         try:
30             length = int(entry.enclosures[0].length)
31             versions = [{"expected_size": length,
32                          "expected_transfer_up": 4096 + 0.05 * length,
33                          "expected_transfer_down": 1.05 * length}]
34         except (AttributeError, ValueError):
35             versions = None
36
37         wc[feed].object_register(
38             object_identifier=object_id,
39             human_readable_name=entry.get("title", entry.link),
40             publication_time=(timegm(entry.published_parsed)
41                               if hasattr(entry, 'published_parsed') else None),
42             versions=versions)
```

Listing 7.2: Registering streams and objects with Woodchuck

Unregistering: When the user unsubscribes from a feed, or an object becomes unavailable, the corresponding Woodchuck stream or object should be unregistered. In PyWoodchuck, this is done by deleting the object's key, as shown below. Note that when unregistering a stream, any containing objects are also unregistered.)

```
del wc[feed][episode]
del wc[feed]
```

Listing 7.3: Unregistering an object and a stream

Adding Woodchuck Support: When an application is modified to use Woodchuck, it will need to register its streams and objects on upgrade. For many applications, this will be as simple as a double loop over its streams and their containing objects.

Processing Transfer and Deletion Upcalls

A Woodchuck server does not typically act on behalf of applications; its role is to schedule transmissions and deletions. When the Woodchuck server schedules an operation, it sends an upcall to the application. If the application is not running, it is first started.

To start an application, Woodchuck reuses D-Bus's autostart mechanism: when a message is sent to a service that is not running, and that service has a .service, D-Bus queues the message and starts the service. The following listing shows an example .service file:

```
[D−BUS Service]
Name=org.podcast
Exec=/usr/bin/podcast−client
```

Listing 7.4: A D-Bus .service file

For an application to receive messages sent to its name, it needs to claim the name, and arrange to receive messages asynchronously—upcalls can arrive at any time. Most D-Bus libraries already provide support for implementing a service. In the case of the GLib- and Qt-based libraries, this is achieved by using the application's main loop. Our libraries were able to easily reuse this functionality.

A small, yet important detail for the application is that it must claim its name, and start the main loop as quickly as possible: the D-Bus daemon will drop queued messages after 25 seconds. If this happens, the application won't know why it was started. This may require that the application refactor some code.

Many applications already provide a service: a major use of D-Bus is to allow applications to be scripted, a sort of simple extensibility mechanism. A media player, for instance, may provide methods to change the volume, pause or resume playback, etc. Some such interfaces have become *de facto* standards. For media players, there is the `org.freedesktop.MediaPlayer` interface,[3] which allows a script to control any media player that happens to be running. Integrating Woodchuck support into these applications is slightly easier because they already handle the D-Bus related details.

When an application receives an upcall, the high-level library remarshals the arguments, and invokes an application-provided callback. In PyWoodchuck, an application registers functions by subclassing the PyWoodchuck class, and overriding the appropriate virtual methods. The functions are typically not very complicated. Generally, they need to massage the parameters into an appropriate form for some existing function. For a stream update, this could mean finding the data structure that corresponds to the indicated stream, and calling an existing function. In practice, the main difficulty is getting access to the right variables, in particular and ironically, if the code is well modularized.

Listing 7.5 shows how an application initializes the D-Bus library to integrate with the main loop, how it claims a D-Bus name, and how it can process the stream update and object transfer upcalls. The deletion callback is similar, but not shown.

Reporting Transmissions

An application needs to tell the Woodchuck server when a stream update or object transfer *attempt* was made. When the application indicates that an attempt was successful, the Woodchuck server knows that it can stop scheduling the transmission. If the attempt was unsuccessful, there are a few error codes that the application can use to tell the Woodchuck server what the problem was. This helps the Woodchuck server improve its scheduling. For instance, if the problem is network related, it may infer that the resource is not accessible from certain networks. Or, if the resource disappeared, it knows not to schedule the transmission again.

When reporting a successful stream update or object transmission, the application can also provide information about the transmission: it can indicate how the user was told about the transmission (the device vibrated, a desktop notification was shown, etc.), the number of bytes actually transferred and the transfer's time and duration. Unfortunately, determining the number of bytes actually transferred can be difficult: oftentimes, the actual transfer is done by a

[3] `http://xmms2.org/wiki/MPRIS`

```
1  import sys
2  import dbus
3  import dbus.service
4  import gobject
5  from pywoodchuck import PyWoodchuck
6  import woodchuck
7
8  # Tell the D-Bus module to integrate with the mainloop.
9  import dbus.mainloop.glib
10 dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)
11
12 class MyWoodchuck(PyWoodchuck):
13     def __init__(self):
14         PyWoodchuck.__init__(self, "podcast client", "org.podcast")
15
16     def stream_update_cb(self, stream, *args, **kwargs):
17         # Update the specified stream. stream.identifier is the
18         # application-assigned identifier.
19         pass
20
21     def object_transfer_cb(self, stream, object, version, filename, quality,
22                            *args, **kwargs):
23         # Transmit the specified version of the object.
24         # object.identifier is the application-assigned identifier.
25         pass
26
27 def init():
28     # Claim our bus name.
29     try:
30         bus_name = dbus.service.BusName("org.podcast", dbus.SessionBus())
31     except Exception, e:
32         print "Failed to acquire D-Bus name: %s" % e
33         sys.exit(1)
34
35     global wc
36     wc = MyWoodchuck()
37
38     mainloop.quit()
39
40 # Don't delay starting the mainloop: D-Bus only queues messages for a short time.
41 mainloop = gobject.MainLoop()
42 gobject.idle_add(init)
43 mainloop.run()
```

Listing 7.5: Receiving and processing upcalls

```
1  import os
2  import os.path
3  import errno
4  import time
5  import urllib2
6  from pywoodchuck import PyWoodchuck
7  import woodchuck
8
9  wc = PyWoodchuck("podcast client", "org.podcast")
10
11 feed = "http://downloads.bbc.co.uk/podcasts/worldservice/docarchive/rss.xml"
12 podcast = "http://downloads.bbc.co.uk/podcasts/worldservice/docarchive/" \
13     "docarchive_20110106-1104a.mp3"
14
15 directory = "/tmp/podcasts/" + feed.replace("/", "_")
16 if not os.path.isdir(directory):
17     os.makedirs(directory)
18 filename = directory + "/" + podcast.replace("/", "_")
19
20 try:
21     transfer_time = time.time()
22     response = urllib2.urlopen(podcast)
23     data = response.read()
24     transfer_duration = time.time() - transfer_time
25 except urllib2.URLError, e:
26     wc[feed][podcast].transfer_failed(woodchuck.TransferStatus.TransientOther)
27     print str(e)
28     sys.exit(1)
29
30 open(filename, "w").write(data)
31
32 # Use Content-length as an approximation of the amount of data received (the
33 # data may have been compressed). Fallback to the size of the object.
34 size = int(response.info().get('Content-Length', len(data)))
35 downloaded = 1000 + size * 1.05
36 uploaded = 1000 + size * 0.05
37
38 wc[feed][podcast].transferred(
39     indicator=woodchuck.Indicator.DesktopSmallVisual,
40     transferred_up=uploaded, transferred_down=downloaded,
41     transfer_time=transfer_time, transfer_duration=transfer_duration,
42     files=[ [filename, True, # File not shared with other objects.
43             woodchuck.DeletionPolicy.DeleteWithoutConsultation], ])
```

Listing 7.6: Reporting an object transmission

library that does not expose this information. In the case of a stream update, the application can also indicate how many new objects were discovered, whether any objects were updated, and whether objects were delivered inline. For object transfers, the application can also indicate what files are used, whether they are shared with other objects, and their deletion policy.

Listing 7.6 shows how to report an object transmission. The code for indicating that a stream was updated is similar. To fetch the object, the application uses Python's urllib2. urllib2 does not provide a mechanism to easily obtain the number of bytes actually transferred over the TCP connection. To work around this, the application estimates the amount of data transfered using the Content-Length header or, if this is not available, the size of the object. The former is preferred, because this is the size of the object in the data stream, i.e., before it was decoded—it may have been compressed. In the case of urllib2, this is not actually a problem: it doesn't support any encodings and thus the size of the object is close to the amount of data downloaded. The application then uses a heuristic to estimate the TCP/IP and HTTP protocol overhead. HTTP overhead is constant, unless chunked encoding is used, in which case it is linear, but this is primarily used for dynamically generated content.

Our application then indicates that it will display a small visual notification on the desktop. Reporting how (and whether) the user was informed about an object transfer or a stream update is useful: this information allows the Woodchuck server to better gauge the time between when the user learned that the object was available, and the time that the user actually used the object, which is probably a good indicator of how important the data is.

The application also indicates what the object's filename is, that the file is not shared with other objects, and that the Woodchuck server can delete it if it sees fit without consulting the application. This is safe because the application can redownload the data, if necessary.

The application also takes care to tell the Woodchuck server that the transmission failed, when this occurs (although it doesn't bother to determine the exact reason). This allows the Woodchuck server to differentiate an unresponsive or buggy application from a well-behaving application encountering transmission problems.

Reporting Use

An application should tell the Woodchuck server when a stream or object is used. This allows the Woodchuck server to learn the user's behavior with respect to an application or stream. Sometimes, the application may not be able to determine when a stream or object is used. This occurs if the data is displayed in a desktop

```
1  # User clicks play.
2  start = time.time()
3  use_mask = 0
4
5  # Periodically "sample" the stream's position and update use_mask.
6  for pos in (1, 2):
7      use_mask |= 1 << int (64 * (pos / length_of_podcast) - 1)
8
9  # User clicks stop after a minute. use_mask is now
10 # 0x3, i.e., the least two significant bits are set.
11 end = time.time()
12
13 wc[feed][podcast].used(start, end - start, use_mask)
```

Listing 7.7: Reporting object use

widget. In this case, there is no easy way to distinguish the data being visible from the data being looked at (and thus used). Determining whether data is used is also hard if the Woodchuck code scripts the application (as opposed to directly modifying it to include support for Woodchuck) and the application does not expose use information to scripts.

In addition to reporting that a stream or object was used, an application can report how long the object was used and which parts. Including how long an object was used allows Woodchuck to infer whether the user actually used the object or just passed over it. By indicating what parts of an object are used, it is possible to determine whether the user completely used the object or, e.g., just viewed the first five minutes of a video. Using historical data, it may be possible to infer whether the user is likely to use the rest at a later time.

Listing 7.7 shows how to report that the user used an object. The code records when the user clicked play. It periodically samples the position of the stream and updates `use_mask`, which is a 64-bit bitmask in which each bit corresponds to approximately 1/64th of the stream. If the stream were 32 minutes long, the first half minute would correspond to the least-significant bit, the second half minute to the second least-significant bit, etc. When the user clicks stop, the application reports the use information to the Woodchuck server.

Oftentimes, updates are run in a separate thread. This is because it is easier to use blocking network operations and blocking operations must not run in the same thread as the GUI code—the application will appear to freeze during long-running operations. urllib2, Python's main http module, is structured in this way, for instance. In fact, there is no easy way to integrate it with the

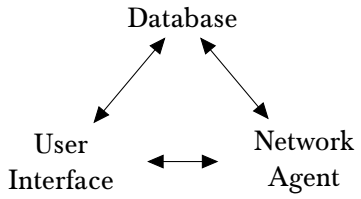


Figure 7.7: An application's main components: the user interface, the network agent and the database.

application's main loop. This raises an issue for a Woodchuck-enabled program: most transfers and their processing are performed in another thread, but the low-level D-Bus library, which PyWoodchuck uses, is not thread safe. Fortunately, the application can execute any Woodchuck calls in the main thread by queueing an idle callback in the main loop. This is thread safe and straightforward, as the following code shows:

```
def e():
   wc[feed][podcast].used(start, end − start, use_mask)

gobject.idle_add(e)
```

Listing 7.8: Using PyWoodchuck from a thread

7.4.2 Background Updates

To provide the best user experience, an application should ensure that it performs *transmissions in the background.* This means that when the Woodchuck server schedules a transmission, only the network agent should run; the user interface should not be shown. The reason for this is that if the user interface is shown when the user did not start the application, it may distract or surprise the user. Showing the window also raises the non-trivial question of whether the application should exit after all transmissions have completed. If the user started interacting with the application, the application clearly shouldn't just disappear. But, if the user never interacted with the application, it may confuse the user (why does this application keep starting on its own!). Further, leaving the application unnecessarily running consumes resources. Unfortunately, not showing the window is sometimes more complicated than it may first appear.

Woodchuck-using applications generally consist of three main components, which are shown in Figure 7.7: the user interface, the network agent, and the

	Separation	Colocation
Background Updates	**Easy**	Hard
Coherency Issues	Yes	**Some**
Software Engineering	**Sound**	Variable
Short-term Effort	Hard	**Easy**
Long-term Maintainability	**Medium**	Variable

Table 7.4: A summary of the major tradeoffs of either separating the user agent and the network interface, or colocating them in a single process (bold is better).

backend database. Our experience modifying several applications to use Woodchuck is that how easy it is to perform background updates strongly depends on how these components, in particular, the user interface and the network interface, are organized.

There are two main ways to organize these components: they can run in different processes, or they can be colocated in a single process. In those applications that we examined where the two ran in separate processes, it was easy to perform updates without showing the user interface; in those applications where the two were colocated, it was significantly more difficult.

Separating components has a number of long-term maintenance advantages, but these come at the cost of increased short-term development effort. Separating components is seen as a good software engineering approach: it enforces modularity, because the interfaces between the components must be formalized, which improves maintainability. This formalization takes time, however. Further, because data is shared between the components, a coherency protocol must also be developed. These trade-offs are summarized in Table 7.4 and detailed below.

Separating Components

Separating the user interface from the network agent means having the user interface, and the network agent execute in separate processes. This requires a clear separation of concerns, and a formalization of the components' interactions—because there are no shared data structures, one component cannot directly access the internal state of the other, and all interactions must occur using some form of IPC.

This type of separation generally results in a straightforward solution to the background update problem: when scheduling a transmission, only the network agent is started. This loose coupling has another benefit: resources can be saved;

the user interface and network agent only need to run when their functionality is actually required.

Realizing this architecture requires more engineering effort upfront: the components' interfaces need to be formalized, and a data coherency protocol is required. Formalizing the components' interfaces is necessary, because all interactions must use them; cheating and violating module boundaries is now often harder than following good software engineering practices. Depending on the IPC mechanism used, transferring data, particularly complex data structures, such as structures and hashes, may also require some complicated marshalling and demarshalling routines. Additional effort is also needed to ensure data coherency: because the user interface and network agent use the same data, and because the data structures cannot be shared, a coherency protocol is needed to make sure in-memory copies of the data are not (too) stale, and that updates are not lost. Unfortunately, it is not sufficient to use shared memory as there are many derived data structures, particularly in the user interface, which also need to be updated, namely display widgets. Ensuring that changes are propagated is easily overlooked if data structures are modified directly. Further, summarizing changes so that the other process knows what to invalidate can be complicated. In practice, however, it will often be sufficient for a component to indicate that something (without specifying in detail what it is) has changed, and any other components simply reload all possibly modified state.

Colocated Components

Components are colocated when they run in the same process. This architecture makes it easier for one component to use another component: the interfaces need not be so rigorously defined, and the language-provided function call mechanism can be used, which simplifies argument marshalling and demarshalling. This architecture also simplifies the data coherency problems: because all components run in a single process, there only needs to be a single copy of data in memory, which all components can share. This does not completely eliminate the coherency problem, however: it is still necessary to ensure that derived data structures such as display widgets are updated when a data structure is changed.

These simplifications save time in the short-term. This is particularly true when developing an application using any of the agile development processes, such as the iterative and incremental development (IID) process in which planning and implementation work are interleaved. (This contrasts with the waterfall approach in which the problem is fully described, and the interfaces are formalized before coding is even started.) When using an agile development process barriers to change, such as formalizing interfaces and marshalling, are imped-

Background Approach	Development Effort	Usability
Separate Componenets	High	**Good**
Show User Interface on Demand	Medium	**Good**
Check for User Interaction	**Easy**	Acceptable

Table 7.5: Ways to improve the usability of background updates in monolithic applications.

iments. IID-style development methods appear to be common. The programs that we examined in our case study (see Section 7.3.2) all appear to be developed using an IID methodology. This may, however, be an artifact of the open development model in which anyone who is interested can participate and influence the project.

Although this architecture can quicken development, it does not have a straightforward solution to the background update problem: because the components are colocated, when a transmission is scheduled, and the application is started, the user interface is also initialized and displayed. If it is undesirable to convert the program to run the components in different address spaces—even if the program is highly modular, this approach may not be easy due to the need to convert function calls into IPC calls, and develop a cache coherency protocol—there are still two approaches to improve the usability of background updates. The first is to show the windows based on a command-line argument, or by automatically detecting whether the program was started by the transmission manager based on the presence of any queued IPCs. Unfortunately, some code depends on the windows being realized. The other is to ignore the problem of the suddenly appearing window, and automatically exit after any updates, if the user never interacted with the program. If this is not the case after an update has completed, then it is safe to quit. These approaches are summarized in Table 7.5.

Initializing the components on demand enables the application to only display the user interface if the user explicitly requests it. When an update is done, the network agent can check if the user interface is displayed and, if not, it can cause the application to exit. When starting the application, a command-line argument can be used to indicate what functionality is desired. Once started, the application first needs to check if another instance in already running and, if so, forward its command-line arguments to that process, which then interprets them as appropriate. Alternatively, the application can automatically detect whether it was started by the transmission manager, but this latter approach is error prone, and includes a race condition. Although the user interface need not be shown, it

likely still needs to be initialized: the network agent often calls the user interface when changes are made. In fact, some changes may be required if any code relies on the widgets being realized. Thus, although hiding the window seems easy, it includes a number of gotchas, which can't easily be hidden in a library.

Instead of not showing the user interface, the application can track whether the user has interacted with it. If this is not the case after the network agent has finished an update, the application can safely quit. This approach is less desirable than not showing the user interface at all, because a new window will suddenly appear when an update is started. But, in many cases, the user won't notice: if the Woodchuck server tries to schedule transmissions when the system is idle, the user will never see that the program was started. The advantage of this approach is that it is easier to implement. At least on X11-based systems, determining whether the user interacted with the user interface is easy: X11 tracks the last user interaction with each window, and both Gtk+ and Qt expose this information via relatively convenient interfaces.[4] If this information is not available, an alternative is whether the user is idle. This heuristic will never result in false positives—if the user never interacts with the system, then the user certainly did not interact with the application's user interface—but it may result in false negatives—just because the user interacted with the system does not imply that the user used the application.

7.4.3 Applications

In this section, we describe the changes required to modify three applications to use Woodchuck, to script another application, and to use Woodchuck in a new application. Each application required approximately three weeks of effort. We needed to make the changes, test them, and either work with the upstream author to integrate the changes, which sometimes requires revisions, or publish the application. We expect that these changes could have been done more quickly by someone already familiar with the application: a lot of our time was spent understand how the applications work, and how to best integrate the required changes. In the end, the three applications that we directly modified now include our changes.

[4] X11 windows have a `_NET_WM_USER_TIME` property, which X11 sets to the current time after each user interaction. Gtk+ exposes this to the application via `gdk_x11_display_get_user_time` and Qt via `QX11Info::appUserTime`.

gPodder

gPodder [98] is a podcast manager written in Python. It runs on Nokia's Maemo- and Meego-based smart phones as well as on desktop-based GNU/Linux, Windows and Mac OS X systems. Although gPodder is highly modular, it uses a single monolithic process.

gPodder provides an extensibility mechanism called hooks. Hooks allow code to modify, and inspect the behavior of gPodder. Any configured extensions are loaded automatically on start-up. An example of a third-party extension is one that listens for new download events (by adding a function to a so-called *hook*) and normalizes the episode's volume. Due to the modular nature of gPodder, and the resulting clean interfaces, extensions can access much of gPodder's core functionality.

gPodder implements time-based automatic updates, however, the application must be running. gPodder can be configured to either only fetch the feed's metadata or to also automatically download new episodes; it does not provide a mechanism to tune this according to the type of connection. This makes time-based updates impractical for users who do not have large data allowances.

Changes Before starting to integrate Woodchuck support into gPodder, we asked the main developer, Thomas Perl, what the best approach is in order to make the code acceptable for inclusion in gPodder. He suggested writing an extension and, where the interfaces were inadequate, to improve them. Several changes were required to the core code to fully integrate Woodchuck. Most of them were straightforward. The end result is that the Woodchuck support is completely separate from the core code, and that the added functionality can be used by other extensions.

gPodder lacked hooks for several events relevant to Woodchuck support—when the user subscribes to a podcast, when a podcast update fails, when a podcast or episode is deleted, when a podcast episode that was not downloaded becomes unavailable, when the application start-up is complete, and when the download queue is empty. With one exception, adding support for a new hook was trivial: each new hook required two lines of code for its declaration, and one more to trigger it—due to the well-factored code there was just a single place in the code where each event occurs. The exception was the start up event, which needs to be called by each frontend. Further, the start up event also needed to provide callbacks to trigger a podcast update and an episode download. These were just thin wrappers around the front-end specific code with a uniform interface. This hook required a dozen lines of new, straightforward code in each front end.

Although gPodder indicates when a podcast or episode data structure is changed, it did not indicate what was changed. This is due to the way that gPodder manages changes: gPodder directly modifies podcast and episode data structures, and then calls a function to synchronize the in-memory copy with the copy on disk. It is in this synchronization routine that the save hook is called, and the Woodchuck module finds out that an episode or podcast was updated. Among other things, this data structure includes where the user paused playback or if an episode was played to the end, and whether an episode's meta-data was modified. The Woodchuck module needed a way to determine what changed to decide what to do, e.g., indicate that the episode was used or schedule a new download. We modified the data structure's class to track the fields that were changed since the data structure was last synchronized to disk. This modification was isolated to three pieces of code, all in the database model module, and resulted in approximately 20 new lines of code.

We next modified gPodder to support built-in extensions. gPodder only supported loading extensions from a special directory under the user's home directory; support for built-in extensions was a pending change. This change was easy: a function was provided for any module to receive hook notifications, and a list of built-in extensions is loaded start up. This list has the sole reference to Woodchuck in the core code.

When gPodder updated a podcast and new episodes were discovered, it displayed a dialog prompting the user to select podcasts to download. This functionality is distracting for non-user-triggered updates. We inhibited this behavior by providing a flag to the update function indicating whether the update was user triggered. This required three new lines of code: one to add the new argument to the update function's signature; one to examine the argument; and, one to change the caller to set the argument appropriately.

When gPodder downloads an episode or updates a podcast, the operation is performed in a thread. When the transfer is complete, the Woodchuck code is invoked via a hook, and it registers the transfer with the Woodchuck server. This requires sending a message over D-Bus, which is not thread-safe. A simple workaround is to queue a callback in the main loop, as shown in Listing 7.8. Because gPodder supports both Gtk+ and Qt, code needed to be added for both toolkits. Perl decided that the support for this should be in the generic utility module to keep the Woodchuck extension toolkit agnostic, and due to the code's general utility. This resulted in a new 44-line function.

When the Woodchuck server makes an application upcall, the application's D-Bus library passes control to the Woodchuck module, which forwards the call to an application-provided stub function. The stub function acts as glue code between the Woodchuck server and the application: it translates the server's

commands into the right sequence of internal function calls. The right functions to call depend on the front end in use. When the front end indicates that start-up is complete, it calls a hook function, and passes a callback function that updates a specified podcasts, and a second one that downloads a particular episode. These functions are passed the podcast to update, or the episode to download, respectively. The stub function needs to look these up by mapping the application-assigned identifier to the corresponding data structure.

Given the application-assigned identifier, the data structure can be read from the database. This, however, returns a new instance of the data structure, which creates a potential coherency problem: the front end also has an instance. To prevent changes from being lost and ensure that changes are correctly propagated, the data structure needs to be shared. To do this, we introduced an additional level of indirection to the database model module: instead of calling a function to look up the data structure, we introduced a class that manages the data structures in memory. When a podcast or episode is looked up, it first checks if there is a corresponding data structure in its cache. If so, it returns that; otherwise, it reads the data from the database. The change was not difficult, but required about a dozen one-line changes to the front ends.

The final change was to support background updates. Perl rejected changing gPodder to use a multi-process architecture. He also didn't want to change gPodder to only initialize the frontend on demand. He argued that both were too complicated. Thus, we modified gPodder to check whether the user has interacted with the application when an update is finished, and, if not, to quit. This required adding a new toolkit agnostic function to the generic utility module. This function consisted of 19 lines of code.

The Woodchuck module consists of 323 physical source lines of code, as computed by SLOCCount, which, using the constructive cost model (COCOMO), would require approximately 3 person-weeks of development effort [129]. The code can be divided into four parts: utility functions, which are used by the module, but not Woodchuck specific; helper functions, which are used by the Woodchuck infrastructure and are Woodchuck specific; the Woodchuck infrastructure, which handles Woodchuck's upcalls and gPodder's hooks; and, the startup and initialization code. The code is generally straightforward. One reason that it is so large, is that it is careful to check for error conditions. The code is broken down in Table 7.6.

Issues *Background Updates:* Although gPodder fully supports Woodchuck, there is one problem: background updates are not transparent. When the Woodchuck server schedules a podcast update, or an object download, it doesn't just

Category / Function	SLoC	Comment
Utility Functions	56	
execute_in_main_thread	23	Schedule a function to run in the main thread.
coroutine	14	Break up a long running function to avoid blocking the main loop for too long.
last_user_interaction	19	Time of the last user interaction.
Helper Functions	35	
autodownload	13	For new subscriptions, only schedule the two newest episodes.
stream_to_podcast	11	Find the internal data structure associated with the specified stream.
object_to_episode	11	Like stream_to_podcast.
Woodchuck Support	107	
stream_update_cb	6	
object_transfer_cb	6	
on_podcast_subscribe	5	
on_podcast_delete	1	
on_podcast_save	19	Podcast metadata changed.
on_podcast_updated	1	
on_podcast_update_failed	12	
on_episode_save	47	Episode metadata changed.
on_episode_downloaded	2	
on_episode_delete	4	
on_episode_removed_from_podcast	4	Episode not downloaded and no longer available.

Table 7.6: Breakdown of the Woodchuck extension in gPodder. SLoC of the function bodies, excluding blank lines, comments, assertions and debugging output.

Category / Function	SLoC	Comment
Startup and Initialization	63	
Module preamble	22	Module imports; recover from missing PyWoodchuck.
__init__	9	Save module parameters.
check_subscriptions	14	Register podcasts with Woodchuck, if necessary.
on_ui_initialized	18	Check if Woodchuck server is available and initialize library.

Table 7.6 (Continued): Breakdown of the Woodchuck extension in gPodder. SLoC of the function bodies, excluding blank lines, comments, assertions and debugging output.

start an update or download, it starts the whole application. If the user is interacting with the system, the new window, which suddenly appears, can be distracting. If the user is not interacting with the system, then when the user again uses the system, gPodder will sometimes still be running. This can be quite surprising especially if the user doesn't understand why this is happening.

Latency: gPodder uses a thread to perform an update or a download in the background. Use of threads is Python is not recommended: the interpreter uses a global lock, which only permits a single thread to run in the Python interpreter at a time, and results in significant overhead [15]. This overhead is painfully obvious when gPodder is performing an update, and the user attempts to interact with the application: we measured a delay between 5 and 45 seconds on the N900 when tapping a widget and the widget reacting. When updates are not being performed delays are minimal. The increased CPU usage also negatively impacts other applications runnning on the system.

This problem occurs more frequently when using Woodchuck, because even though Woodchuck tries to schedule downloads when the system is idle, the downloads take time (audio and videos can be large, and take a long time to download), and the Woodchuck server prefetches a lot of content. This problem could be mitigated if gPodder paused downloads when the user becomes active.

Amount Transferred: The Woodchuck server can better allocate the data allowance and energy, if it knows approximately how much data a transfer will likely transfer, and how much data it actually transferred. The podcast data typ-

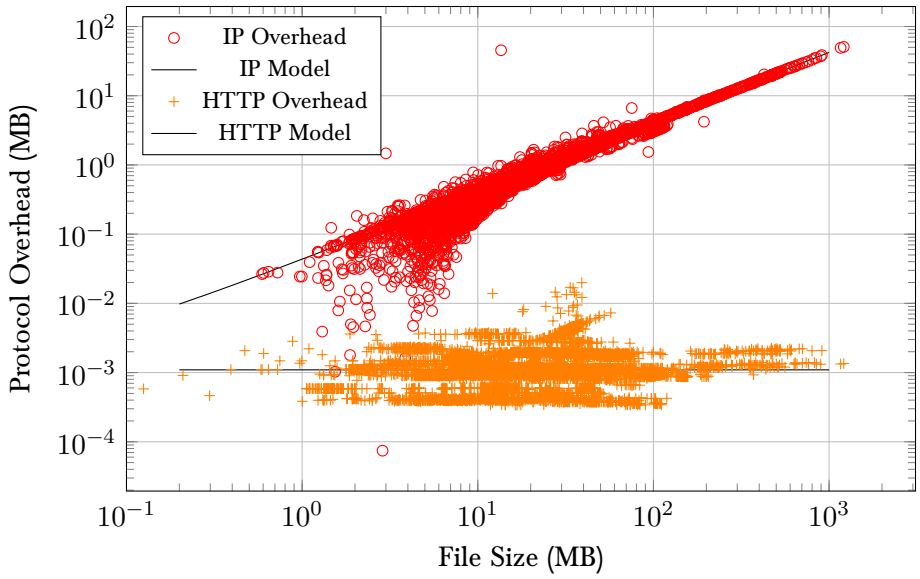


Figure 7.8: HTTP and IP overhead for gPodder transfers.

ically includes the expected size. The amount of data actually transferred is a function of the object's size, any content or transport encoding and the protocol overhead, which, in this case, is due to TCP/IP and HTTP. Interposing at the socket interface makes it possible to count the bytes passed over the TCP flow. Obtaining the number of bytes transferred over a TCP flow would be better, but requires a special kernel module and root privilege on Linux. Obtaining this information is not strictly necessary, however, given that TCP/IP's overhead is fairly predictable.

We confirmed this by fetching the list of the 100 most popular podcasts on `http://gpodder.net`,[5] gPodder's web service, and measuring the amount of overhead due to HTTP and to IP for over 5300 episodes. To measure the HTTP overhead, we counted the number of bytes sent and received over the socket used to communicate with the HTTP server, and subtracted the size of the file; to measure the IP overhead, we used Linux's ip_conntrack module to determine the number of bytes sent and received via the corresponding TCP flows, and subtracted the amount measured at the HTTP layer.

Figure 7.8 shows the results as well as a model based on a least-squares regression. The HTTP overhead is independent of the file size, and approximately 1 KB (it depends primarily on the number of HTTP redirects: half of podcasts are

[5] `http://gpodder.net/toplist/100.opml`

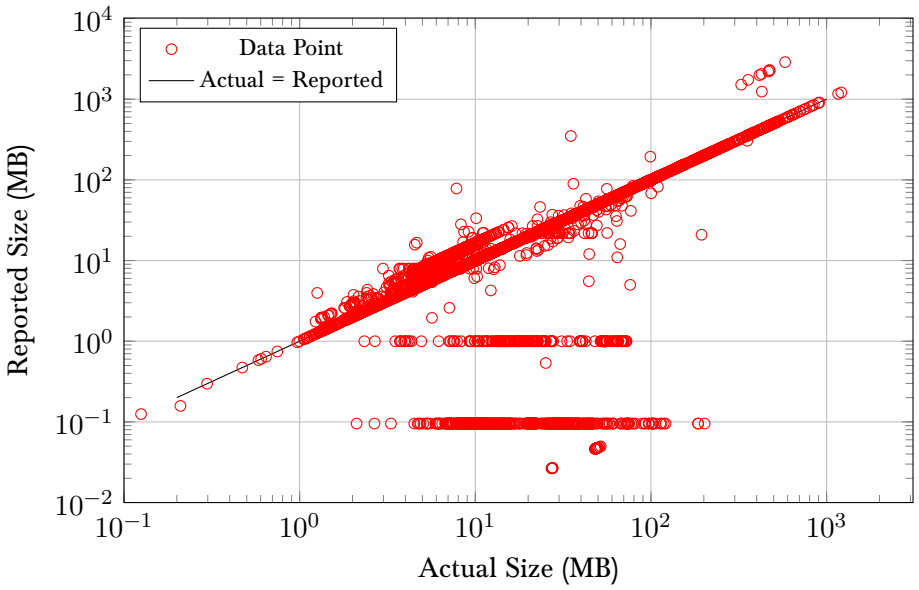


Figure 7.9: The actual size vs. the reported size (as indicated in the RSS or Atom feed) of episodes.

hidden behind at least one HTTP redirect; a quarter behind at least two). This is expected, because gPodder does not support any content encodings, such as deflate. It does support chunked encoding, whose overhead is linear in the size of the data transferred, however, it appears that chunked encoding was never used. This makes sense, because chunked encoding is primarily for dynamic content whose size is not known beforehand. The IP overhead is 1 KB plus 4.2% of the data transferred, which is expected given TCP and IP's setup cost and header sizes. This will vary depending on the actual packet sizes that the network supports, and the number of packet retransmissions. Given how insignificant the HTTP overhead is relative to the actual file sizes (most are over 5 MB), and how accurate we can model the overhead, more accurately determining the HTTP overhead was deemed unnecessary.

We also collected the size of an episode as reported by the episode's metadata. We found that 6 podcasts did always include a length and that 49 misreported the length at least once. Of those, 34 misreported the size by more than 10% at least once. In terms of episodes: of the 5306 episode, 331 episodes (6%) had no enclosure length, 1202 (23%) episodes had an incorrect enclosure length, and 662 episodes (12%) had an enclosure length that was more off by more than 10% of the actual size. Figure 7.9 shows the actual size of the file vs. the re-

ported size. A number of episodes are reported to be 1 MB, 10 MB or 20 MB in size. This suggests that some podcast authors simply choose a "reasonable" size rather than take the time to incorporate real values.

This suggests a mechanism in gPodder for performing an HTTP HEAD on episodes that have no length, or whose reported length is likely to be incorrect based on historical data. This should be relatively inexpensive if this is done when the podcast is updated, and the episodes are on the same server as the podcast's data: the same HTTP connection can be reused. Adding this support is non-trivial, but not difficult.

FeedingIt

FeedingIt [79] is an RSS reader written in Python. It runs only on Maemo. FeedingIt supports time-based updates. Like gPodder, it does not incorporate information about the type of network connection an update would use, in particular, whether performing the update would use the cellular network. This limitation makes time-based updates reasonable only for those users who have large data allowances.

FeedingIt has three front ends: a Gtk+-based GUI, a desktop widget, and a command-line interface. The desktop widget serves two roles: it provides the user with a quick overview of the feeds with unread articles and, because it is always running, it is responsible for scheduling time-based updates. Both the GUI and the command-line interface can perform updates. When an update is triggered via the widget, the widget sends a D-Bus message. This message is processed by the GUI if it is running. If not, the command-line interface is started, and it handles the message. The idea is to make updates transparent—the GUI implements a progress bar—without interrupting the user by having a window pop up. The components synchronize their state by broadcasting D-Bus messages. There is one type of message: something has changed. When a process sees such a message, it reloads the database.

Changes Before adding Woodchuck support to FeedingIt, we first contacted FeedingIt's author, Yves Marcoz, and asked for recommendations regarding the best approach. He suggested integrating Woodchuck in place, i.e., he saw no reason to try too hard to abstract the functionality, and place it in a separate module.

The most time-consuming change to FeedingIt was scheduling Woodchuck-triggered updates. The GUI and the command-line client both implement an update manager (a dispatcher and thread-pool manager). We could have added similar code for handling Woodchuck upcalls to each of these implementations,

however, we decided to refactor the code so that the GUI does not perform downloads directly, but uses the command-line client instead. In addition to reducing the amount of duplicated code (both in FeedingIt, and that required to add Woodchuck support), this made the GUI usable while an update was occurring: as already noted in the context of gPodder (page 281), multi-threaded applications perform very poorly in Python [15]. Because Woodchuck can schedule updates at any time, this change helped reduce the degree to which Woodchuck-scheduled updates interfere with the user's use of FeedingIt. Although the change was large in terms of the number of lines modified, it consisted primarily of shuffling and tweaking existing code. The major addition was a new D-Bus signal to indicate the status of an ongoing update. This was necessary so that the GUI could display, and update the progress bar.

The remaining Woodchuck-related changes were made to the FeedingIt database module. As recommended by Marcoz, we did not abstract the Woodchuck functionality. Nevertheless, some support code was added to a Woodchuck module. This decision simplified the required changes as the Woodchuck-related code could directly access the preexisting data structures; there was no need to introduce new interfaces so that Woodchuck could access data, and become aware of relevant events. Further, because the changes were made in the database module itself, there was no need to make the front ends aware of Woodchuck. The Woodchuck-related changes can be divided into roughly three categories: the initialization, the download process, and the reporting of Woodchuck-related events.

Initialization: The Woodchuck-related initialization code was added directly to the database module's initialization routine. In addition to initializing the Woodchuck module, the code compares the subscribed-to feeds with the registered streams, and ensures that only subscribed-to feeds have a corresponding stream, and that these are up to date with respect to FeedingIt's state. This change is essential to migrate existing users of FeedingIt to a Woodchuck-enabled FeedingIt. The code also ensures that Woodchuck's configuration and FeedingIt's configuration do not drift out of sync. Due to the cross-cutting nature of Woodchuck support, a change to the application may forget to update some Woodchuck parameter. It is also possible that the Woodchuck-server crashes, and fails to record some information. Alternatively, the user may directly modify the database, perhaps using a script, and not update the Woodchuck configuration. This change gracefully handles such inconsistencies by correcting them the next time the application is started. This code consisted of 37 physical source lines of code.

Downloads: The code to update a feed is in the database module. To update

a feed, a front end invokes the `updateFeed` method of appropriate feed object. This function either forwards the request to the command-line client, or processes the update if the current process is the command-line client. In the latter case, the code creates and enqueues a job, which the thread pool manager eventually runs. An update consists of downloading the feed, processing any new articles and, if enabled, downloading any referenced images.

We decided not to modify the code to have Woodchuck explicitly schedule image downloads, but to instead download them during a stream update. FeedingIt already downloads new articles when a feed is updated: it can't defer this download, because articles are delivered inline. To have the Woodchuck server schedule image downloads would have required registering two version of the object corresponding to the article: a low-quality version, which corresponds to just the text, and a high-quality version, which corresponds to the text and the referenced images. When FeedingIt downloads a stream, it would register the object, and mark the low-quality version as downloaded. The Woodchuck server could then schedule the download of the higher-quality version when appropriate. It is essential to register a Woodchuck object and mark the lower-quality version as transferred so that the Woodchuck server knows when the object became available and so that should the user read the article before the images are downloaded, it is possible to mark the object as read. Implementing this would have required splitting the image download code into a new function and registering multiple versions of the object. Both of these are relatively straightforward changes, however, we decided not to perform this change during the initial port primarily, because Murmeltier did not fully support objects with multiple versions.

Our first change to the update code was to detect whether an error occurred in transferring the feed data. This required 21 lines of code to examine the HTTP status code to determine what happened, and to act on that information, i.e., to register a failed update with Woodchuck or to continue. The original code did not check for failure, but relied on exception handling to catch and ignore unusual cases.

We next collected transfer statistics. Rather than just use the object's file size to estimate how much data was transferred, we implemented a transfer handler for urllib2, the HTTP library that FeedingIt uses to fetch data. The handler counts the number of bytes sent over the socket, which includes HTTP overhead, and accounts for any changes in size due to the encoding used, but does not consider overhead due to TCP/IP. The code was implemented as a separate module and consists of 101 lines of code, which is not FeedingIt specific. Given the results from gPodder (see the discussion on page 281), this change is unnecessary, because FeedingIt also uses urllib and HTTP, however, it demonstrates

what is likely required in situations where the file size is not a reliable basis for determining the amount of transfered data.

The urllib2 handler was straightforward to integrate: when updating a feed, an additional line of code was needed to initialize the handler, and another to pass it to the call to download the feed or image. Approximately 17 lines of code were needed to compute the transfer statistics for each article.

The download code also needs to register new objects, mark them as downloaded, and report the result of the stream update. This required 42 lines of code. In addition to reporting the amount of data transferred, we also reported the transfer time and duration, the publication time of each article (when available), the amount of disk space used by each object, and the application indicators used to show that an update occured.

A remaining issue is that a stream update is done from a thread, however, making Woodchuck calls is not thread safe (due to the low-level D-Bus library). To handle this, the Woodchuck-related code was placed in a function, which was queued as an idle callback in the main loop. We used a small module to facilitate this. The module consists of 50 lines of application-independent code. We could have used a sinlge line of code to enqueue the function (as shown in Listing 7.8), however, this module optimizes calls and simplifies argument passing. As many Woodchuck-using applications are likely to need this module, it makes sense to have PyWoodchuck include it, rather than have every application include a copy in their source tree.

Reporting Events: Acting on the remaining events, such as the user reading an article, or subscribing to a new feed, was relatively easy and non-invasive. For instance, the FeedingIt GUI calls a database-provided function to mark an article as read. Modifying these types of functions generally required two lines of code to call the appropriate Woodchuck function and a few lines of code for doing error checking.

Khweeteur

Khweeteur [49] is a Twitter and Identi.ca client for Maemo. A screenshot is shown on page 258. It is written in Python and uses multiple processes: the GUI runs in one process, and a daemon process performs updates. To maintain coherency, when the update daemon downloads new status updates, it broadcasts a D-Bus message indicating which view was updated. When the GUI process sees such messages, it reloads the indicated view from the database if it is being displayed.

Khweeteur supports time-based updates. When enabled, the daemon runs in the background, and periodically polls for new status updates. Like gPodder and

FeedingIt, it does not provide a mechanism to modify its behavior when only a cellular connection is available for updates. This makes the use of time-based updates feasible only for those who have large data allowances.

Before modifying Khweeteur to support Woodchuck, we first contacted Khweeteur's author, Benoît Hervier. He confirmed that adding support directly in the code, and not abstracting the functionality was a good approach.

Changes The changes to Khweeteur were mostly straightforward. The biggest obstacle was understanding Khweeteur's code: instead of using functions to abstract functionality, Khweeteur simply repeats the same code. Oftentimes, the code has small variations. This was sometimes due to the relevant corner cases. Other times, it was because of a bug fix that was not applied everywhere.

The Khweeteur daemon initializes Woodchuck support on start up. It first initializes the Woodchuck module. If a Woodchuck server is detected, it ensures that the set of streams known to the Woodchuck server are consistent with the set of views, and it disables time-based automatic updates. These changes consisted of 50 lines of new code.

We mapped a stream to each so-called *view* rather than to an account. A view is a collection of status updates, which Khweeteur displays in a scrolled window. There is a separate view for the user's timeline, for her direct messages, and for each of her standing queries. The Twitter protocol supports updating each view independently. Mapping streams to views rather than to an account is advantageous, because different views are likely to have different access patterns. For instance, we expect the user to use the timeline the most, to read direct messages promptly, and to have fleeting interest in standing queries. Frequently updating standing queries after the user's interest has waned is a waste of resources. This is particularly true for popular queries, which have a lot of new content. Khweeteur merges the same views from different accounts into a single view. Thus, the timeline view includes both the Twitter account's timeline as well as the Identi.ca account's timeline. It is thus not possible to differentiate the use of the Twitter account from the Identi.ca account. Nevertheless, we represent each account and view separately so that Woodchuck can learn the arrival rate of the underlying process.

Originally, Khweeteur had a single function that updated all views. We split this code into two functions: one that queues an update for a particular view, and one that iterates over all views, and queues updates using the first function. The stream update upcall was then written to use the first function.

We represent each status update as a single object. Like FeedingIt's articles, status updates don't need to be explicitly downloaded; they are delivered inline.

Also like FeedingIt's articles, status updates can reference external objects. Some status updates reference web pages. Some others reference images. Khweeteur does not specially handle these objects: if the user visits a link, Khweeteur just opens the link in the user's web browser. If Khweeteur did prefetch these objects, it would be useful to schedule their transmission using Woodchuck. Most messages do, however, reference the sender's avatar, which Khweeteur does manage. When Khweeteur displays a status update, it shows the sender's avatar next to the update's text. We could have used multiple object versions—a lower-quality version consisting of just the text and a higher-quality version, which includes the image—to allow Woodchuck to schedule avatar downloads. We decided not to do this. The main issue is that a single avatar is often shared by many status updates: a user normally receives messages from the same users over and over again. Thus, an avatar is not really part of an object even though it is referenced by the object. A better approach would be to introduce a separate avatar stream, and represent avatars as objects in that stream. This simplifies the clean up of old avatars: Woodchuck will automatically figure out when they have not been used for a long time and delete them. This approach is not difficult, but we did not consider the added value worth the effort in the initial port.

After attempting to update a stream, we register whether the update was successful. If the update failed, we report the failure to Woodchuck. Otherwise, we register the new objects and the fact that they have been transfered. We generate the objects' human readable name using the sender's moniker, and the first 25 characters of the status update. When reporting that an object was transferred, we report the amount of disk space that it used. A typically status update is about a kilobyte in size: although a status update contains at most 140 characters of user-generated text, a status update also includes its global identifier, the sender, any location information as well as another twenty or so fields. Unfortunately, we cannot report how much data was actually transferred. Status updates are encoded as JSON objects. The Twitter module converts these to use Python's data structures, and returns that representation to the application. The application never directly handles the JSON data, nor does the twitter library provide an interface to determine the amount of data actually transferred, or a straightforward way to interpose on the network socket. Given that the Python data is a reasonable first-order approximation of the actual data transferred, we decided that providing just the file size was sufficient. Registering stream updates and object transfers required 57 lines of code.

We were unable to report object-use information to the Woodchuck server. When Khweeteur displays a view, it doesn't display a summary of each status update: it displays the status updates in full. It doesn't make sense to display a summary of a status update: they are so short. This makes determining whether

a status update was used difficult: using a status update does not require an explicit user interaction. A possible heuristic is to mark a status update as used when the user performs some action on it, e.g., sends a reply, opens a containing URL, or marks it as a favorite. This heuristic captures just a small portion of the actual uses: most status updates are read, and not acted on. Another heuristic is to consider those status updates that are displayed as used. Because only a few status updates are visible on the screen at once, and the user must scroll to see more, the user has likely read any visible status updates. The problem with this approach is that Khweeteur doesn't do the actual drawing: this is delegated to the GUI toolkit. It would be possible to sample the position of the scroll bar and compute the status updates that are in view, however, this requires a significant amount of effort, which did not appear to justify the benefit: individual objects are never fetched, only a view is ever updated. Khweeteur can, however, easily and reliably determine what streams are used and report that. Reporting stream use when the user selects the view takes just a few lines of code in the GUI.

Unlike gPodder and FeedingIt, Khweeteur manages uploads as well as downloads. Twitter is an interactive medium: a user publishes status updates, shares other users' status updates ("retweets"), and sends private messages to other users. Khweeteur collectively refers to these as *posts*. When the user creates a post, Khweeteur saves it in a staging area, and starts the update daemon. The update daemon tries to send the post immediately. It can fail if, for instance, there is no network connectivity. In this case, Khweeteur doesn't display an error message and give up: it tries again later. This functionality improves the usability of Khweeteur when operating in disconnected mode. To allow Woodchuck to schedule the uploads, we represent the staging area as a stream, which never receives updates, and the posts as objects in that stream, whose transmission Woodchuck schedules.

Khweeteur implemented sending all posts as a single function: it iterated over each pending post and sent it. It would have been reasonable to call this function when any post was scheduled for upload: in most cases, if Woodchuck schedules sending one post, it will schedule all pending posts. Nevertheless, we split this function in two: one that sends a particular post, and another that iterates over each post and calls the first function. The object transfer upcall uses the first function.

After attempting to upload a post, we register whether the result with the Woodchuck server. If the post was also deleted, we indicate that as well, and then unregister the object. Messages are deleted if they are successfully sent or generate a server error. This required 28 lines of code.

Assigning stream and object identifiers required a small trick. Khweeteur supports multiple accounts, which is especially useful for users who have both

a Twitter and an Identi.ca account. This means that it is not sufficient to use a view's description (e.g., "timeline" or "search:khweeteur") as the stream name: each account has the same set of views. Instead, we used the account's key plus the description. This required encoding the name when addressing a stream, and decoding it on Woodchuck upcalls. This resulted in a few lines of code for two new helper functions, but was otherwise straightforward.

The Khweeteur daemon performs updates in a separate thread. This is necessary because the update daemon needs to process D-Bus messages asynchronously, and the library that it uses for accessing Twitter does not integrate with the main loop: it blocks on network operations. Because Woodchuck calls can only be made from the main thread, we used the same 50-line module as in FeedingIt for executing Woodchuck-related functions in the context of the main loop (see page 287 for details). This module required no modifications even though Khweeteur is Qt-based and FeedingIt uses Gtk+.

We didn't use Woodchuck's storage management facilities in Khweeteur. Khweeteur has its own clean up mechanism and policy for purging old messages. We initially removed this, however, we found that as the number of messages in a view exceeded a few hundred, the time it took to load the view was unacceptable. This is because Khweeteur does not load status updates on demand, but loads all status updates when the view is selected. This is partially due to the requirement to determine the amount of vertical space needed by each message to properly size the scroll bar. This can be worked around, but requires a fair amount of effort. Given that Khweeteur doesn't support searching in messages, and most messages are just read once, we came to the conclusion that supporting more than a few hundred messages offers little benefit to the user anyway.

APT Woodchuck

APT Woodchuck is a small program, which fixes a big problem with Maemo's built-in application manager, HAM. By default, when the user connects to a network, and a software update hasn't been attempted in the past 24 hours, HAM performs an update even if the user's connectivity is via the cellular network.[6] As an update is typically a few megabytes large, this is a serious inconvenience for users with a small data allowance. This problem can be fixed by disabling updates (albeit not in a user-friendly manner). This solution introduces a new problem: most users still want to install new software, and know when updates for their installed software is available. APT Woodchuck modifies HAM's behavior to use a Woodchuck server to schedule updates. As an extension, APT

[6] See, for instance, `https://bugs.maemo.org/show_bug.cgi?id=5137`.

Woodchuck also prefetches updates for installed packages (but does not install them).

We didn't modify HAM directly. Although the source code to HAM is available, it is not possible to provide an update for it because it is a system component. Instead, we wrote a script called APT Woodchuck, which disables HAM's update functionality, and uses Woodchuck to schedule updates. This is possible, because HAM has a setting that controls how often it should perform updates, which APT Woodchuck uses to disable automatic updates, and updates are performed by a command-line driver program, which the script can call.

APT Woodchuck maps the updates to a stream and each package update to an object. Although Woodchuck can cause HAM to perform updates and fetch packages, HAM does not provide a way for it to determine when a package was installed. APT Woodchuck does not lie: it simply reports nothing. Woodchuck is still able to properly schedule updates, because it knows the stream's application-specified freshness—it is just unable to update this value based on usage. In this case, however, adapting the update frequency is unlikely to provide a benefit.

APT Woodchuck is able to determine the size of a package update, however, it does not have a mechanism to estimate the amount of data that checking for updates will use. It also is unable to report the amount of data actually transferred when checking for an update. This is a problem, because the size of an update is variable (i.e., there is no reasonable guess), and this information is helpful when determining whether to schedule an update. The main issue is that the driver program does not expose this information. It is possible to track the amount of data transferred by proxying the network calls, however, this requires a significant implementation effort.

APT Woodchuck is written in Python and, according to SLOCCount, consists of 449 physical source lines of code, and required just over one person month to develop [129]. This estimate is high: we invested two weeks of time to implement, test and publish the program. Approximately half of the code is concerned with manipulating HAM's driver program to fetch updates and prefetch packages, a quarter with Woodchuck-related support, and the remaining quarter with various minor tasks, such as command-line option parsing, logging, and having the script quit when it has been idle for a few minutes.

VCS Sync

VCS Sync is a program to synchronize version control repositories. It can either fetch or push change sets. It determines what to do based on a small configuration file in which the user specifies the location of each repository, and the branches to fetch or push. VCS Sync supports both Git and Mercurial.

VCS Sync is useful for tracking a remote repository, and for keeping a copy of a repository up to date on another machine. Although not very useful on a mobile phone, VCS Sync is an example of how to write backup software that uses Woodchuck. We would have modified an existing program, however, we were not aware of any backup software that automatically performs network syncs for Maemo.

VCS Sync uses a stream and an updatable object for each branch it synchronizes. This is similar to the way that we recommended to add Woodchuck support to the weather forecast program that receives continuous updates on page 245. Like APT Woodchuck, VCS Sync is not able to report use information to Woodchuck or the amount transferred: it does not have a way to determine this information. Nevertheless, Woodchuck is able to schedule the updates based on the streams' freshness parameter.

VCS Sync is written in Python, and consists of 440 physical source lines of code. According to SLOCCount [129], it required one person-month of development effort. In reality, it took approximately two weeks of full-time effort to write, debug and release. The code is divided among processing the configuration file, managing the VCS worker programs and managing Woodchuck.

Conclusion

We presented the results of adapting four programs to use Woodchuck and writing one new program. Each program required approximately 500 lines of Woodchuck-related code. Most modifications were minor, and no disruptive changes were required. Each program required less than a month of time. This includes the time to modify, test and collaborate with upstream to integrate the changes or publish the result, as appropriate. Of the three programs we directly modified, all three integrated our changes into the main line.

We think that given good documentation, application developers should be able to add Woodchuck support in just a week: they are already familiar with their code, and most of the Woodchuck changes are either straightforward, or can be ignored in an initial port. For instance, reporting the amount of data that is actually transferred is useful, but proved difficult to determine and can be safely ignored, particularly when the content's size is a good first order approximation of the transfer size, which we observed is often the case.

7.4.4 Summary

In this section, we evaluated Woodchuck by describing in detail the types of changes that applications need to make, examining application architectures to

understand the difficulty of certain changes, in particular, seamless background updates, and presenting the results of adopting fours programs to use Woodchuck and writing one program from scratch. We concluded that an application developer should be able to integrate Woodchuck support in their application in about a week, given good documentation.

7.5 Conclusion

In this chapter, we presented the design and implementation of middleware for scheduling opportunistic data transfers on a mobile device. We argued that a centralized service saves application developer time, saves resources, facilitates scheduling and exploiting synergies, and potentially increases privacy. We then presented a simple and easy-to-use interface for scheduling opportunistic data transfers. We evaluated this interface by modifying three existing applications, writing a new application, and implementing a helper application that enables opportunistic data transfers for a proprietary application. We used two metrics: whether the proposed changes were acceptable to upstream; and, the amount of required changes and their invasiveness. For all three modified applications, our changes were accepted by the upstream developers. In terms of the required changes, typically just a few hundred lines of code were needed to provide basic or intermediate support for opportunistic transfers. This small number is because most of the required infrastructure already exists, and the implementation primarily linked the transmission manager's callbacks to the right functions. In the end over, $54\,000$ users installed our middleware.[7] Unfortunately, despite extensive documentation, no developers integrated support for our middleware on their own.

[7] http://maemo.org/downloads/product/Maemo5/murmeltier/#

Chapter 8

Conclusion

In the introduction, we provided an example of a context-aware application, a podcatcher, that prefetches the latest news report before the user leaves work based on the user's past behavior on her commute home. This thesis doesn't provide all of the components to realize this scenario, but we make some small steps in that direction.

Using the algorithms that we developed to identify places in chapter 5, the application can be confident, without any expenditure of energy, that the user is still at work. The application can also be confident that high-speed Wi-Fi is available. Indeed, it can conclude this without expending energy to scan for available Wi-Fi access points: in the evaluation of these algorithms, we found that places are good indicators of available resources, including network connectivity.

In chapter 6, we developed algorithms to predict the the user's location in the near future. Our evaluation revealed that we were able to predict a user's location over the next 24 hours with 82% accuracy, on average.

Using this knowledge, the application is able to predict with high likelihood when the user will approximately leave for work, and it can schedule some data transfers accordingly. Of course, having all applications implement the required logic not only places a high burden on application developers, but it makes coordinating access to shared resources difficult. Hence, in practice, the actual download would be scheduled by a transmission manager, such as the one we describe in chapter 7, and the application developer would only have to implement a small amount of support.

The two main missing pieces of the puzzle are: determining what data the user wants; and, scheduling the resources to avoid exhausting the device's energy, and any data allowance.

We note that our algorithms function on the device itself; we did not need

to send the user's data to a third-party to process the data. This thesis provides strong evidence that doing computations locally, contrary to the current popular trend of offload to the cloud, is a reasonable alternative.

8.1 Broader Lessons

During the course of this work, we learned that reproducability is not a given. While developing algorithms to aggregate the cell towers and predict the user's location, we attempted to reimplement several algorithms. The experience surprised us. Laasonen et al.'s tower aggregation algorithm [68] as described was so computationally expensive that it failed to complete even on a very small data set with days of CPU time. After describing our problem to the authors, one replied that they had indeed used a variant of the algorithm, but he couldn't remember exactly what they had changed. We also tried to implement Chon et al.'s tower aggregation algorithm [135]. In this case, the algorithm was radically underspecified and a number of educated guesses about the authors' intent was not enough to come up with a reasonable approximation. Further, when we asked the authors for help, our email was ignored. Yadav et al. made their data set publicly available [70]. Unfortunately, it was just the raw data. When we asked if we could have access to the version they used to evaluate their algorithms, they indicated that they no longer had it, and that the scripts to clean the data had also been lost. When attempting to reproduce the NextPlace results on the Dartmouth data set [105], we discovered that they had only used a subset of the data set, but they hadn't indicated what subset in the paper. We asked the authors for help, and they replied that hadn't recorded what data they used, and that the relevant code was lost.

Another surprising result is a lack of baselines. In the evaluation of our prediction algorithms, we provided two baselines. These baselines performed surprisingly well on our data set: they managed to correctly predict about two-thirds of the prediction trials. Most other papers did not provide a baseline. In particular, it would have been interesting to see a baseline in the NextPlace paper: comparing just the *prediction precision* (i.e., attempted predictions) of our implementation NextPlace on our data set, it is only barely better than our baselines.

When developing our transmission manager, we attempted to convince several developers to integrate support for it. Their response was that they didn't want to have a hard dependency on third-party middleware, and a soft depedency would be too much work for too little benefit, because no one would install middleware that they didn't know about it. (Unlike Android and iOS app stores,

the N900 app store allowed middleware.) This provides a serious conundrum for middleware authors: the only way to see much adoption is to convince the vendor to include the middleware. Note: having application developers just include a copy of the middleware doesn't help: when the purpose of the middleware is to manage a shared resource and there are n instances of the middle running on behalf of n applications, the purpose of the middleware is defeated! This is unfortunate as it prevents innovation in this space.

8.2 Future Work

It would be interesting to do a more thorough evaluation of our oscillation heuristic, cell tower aggregation algorithms, and our prediction algorithms using other data sets. In particular, if the data set included GPS traces, then for the oscillation heuristic and the cell tower aggregation algorithms, we could also check how compact the resulting locations are, to what degree do they overlap, and whether they obscure user movement.

The work in this thesis addresses a small part of a much larger multivariate optimization problem. Specifically, our ideal transmission manager should optimize: data availability (i.e., ensuring the data the use wants is available when the user wants it), energy consumption, use of any data tranfer allowances, and available storage. This thesis has focused on identifying available resources in the near future, which is useful for determining what type of network connectivity will be available in the next x hours, and when the next charging opportunity is likely to occur. We haven't examined predicting what data the user is likely to want, or how to balance the different dimensions of this optimization problem.

As just described, the transmission manager takes a rather myopic view: it only considers the user's behavior on the device itself. However, some behavior may be correlated across devices. For instance, if the user starts watching a new television series, it may be useful to automatically download the next, unwatched episode on the mobile device. Similarly, large emails that have already been replied to may not need to be downloaded at all on the mobile device.

Given how overloaded cellular networks are [12], it would be interesting to work with the carriers to explore how to provide information and incentives to mobile devices to schedule delay-tolerant transmissions to reduce congestion.

Assuming a transmission manager is widely deployed, it would be interesting to run a study to understand whether user's still worry about overage charges and throttle their use, as discussed in Trestian et al. [118].

Intended to be blank.

Appendix A

User Study Materials

This appendix contains a copy of the recruitment email and the consent form. chapter 2 contains details about the user study.

A.1 Recruitment Email

```
Hello world!

We at the Hopkins Storage Systems Lab (HSSL) are trying to understand
how you, the tech elite, access and use data and how you access the
Internet on mobile devices.  We are interested in you, because we
think that your data-use habits may be indicative of what might become
common in a few years time.

We want to understand how you access and use data on mobile devices to
improve the user experience on mobile devices.  Specifically, we want
to:

  - Improve disconnected operation;
  - Make accessing data faster;
  - Increase battery life;
  - Reduce network connectivity costs; and,
  - Simplify data management.

We suspect that significant amounts of data that you use are
downloaded on demand and that this data could be effectively
prefetched.  Although prefetching sounds easy enough, there are a
number of issues that need to be considered: when should data be
prefetched? what data should be prefetched? how do we avoid exhausting
free space? how do we enable applications to coordinate the use of
```

shared resources?

To this end, we are conducting a user study. We'd like you to participate by running our data collection software, which gathers information about the data you use, your network connectivity, and your battery use.

To help by running the data collection software, which should take about 10 minutes to install and not require any further interactions on your part, please visit:

http://hssl.cs.jhu.edu/~neal/woodchuck/smart-storage-logger.install

or:

http://tinyurl.com/wcssl

Much of the data that we collect will be anonymized. No personally identifying data will be published. Data collection will last for approximately one year.

Anyone with a compatible device may run the data collection software. Your participation in this experiment is entirely voluntary. Should you choose to participate, your data will be kept confidential to the extent possible by law. Only researchers involved in this study will see collected data. Published data will not include identifying artifacts (i.e., we will make every effort to prevent the identify of participants from being determined from the data we publish). Encryption will be used to transfer collected data and to verify the server to which that data is uploaded.

If you have any questions or concerns, feel free to contact me at neal@cs.jhu.edu or Randal Burns, the principle investigator, at randal@cs.jhu.edu. Your assistance in helping us meet our research goals would be greatly appreciated.

Thanks for your help!

Neal Walfield

P.S. This study is research. You will not receive any direct benefits from participating in this study. This study may benefit society if the results lead to a better understanding of how data is used on

mobile devices. The study is taking place at the Whiting School of Engineering at the Johns Hopkins University in the United States. The principle investigator is Randal Burns:

Email: randal(at)cs.jhu.edu
Phone: 410.516.7708
Mailing Address:
Department of Computer Science
The Johns Hopkins University
222 New Engineering Building
Baltimore, MD 21218
USA

Approved by HIRB on XXX
HIRB Study number: 2010112

A.2 Consent Form

INFORMED CONSENT FORM

TITLE OF RESEARCH PROJECT:

Smart Storage: Improving the User Experience on Mobile Devices by More Intelligently Managing Data

Principal Investigator: Randal Burns, the John Hopkins University (USA) Whiting School of Engineering

Date: XXX, 2010

WARNING:

This study includes the COLLECTION OF PERSONALLY IDENTIFYING INFORMATION, including the names of files on your mobile computer, and web sites that you visit. Before installing the software, you should carefully read this document to understand what information is collected and the potential risks. IF YOU DO NOT COMPLETELY UNDERSTAND THE RISKS, which are explained in detail below, DO NOT INSTALL THE SOFTWARE.

PURPOSE OF RESEARCH STUDY:

The purpose of this user study is to understand what types of data people use on mobile computers, such as smart phones and laptops, how they use that data, and when network connectivity and power are available. The overarching goal of the research project is to improve the user experience on mobile computers by better automating the management of data--automatically downloading data the user is likely to use, and deleting (or suggesting for deletion) data that the user is likely to no longer need when free storage space becomes scarce.

PARTICIPATION:

Anyone 18 years of age or older may participate in the study. A further requirement to install the logging software is that the participant have access to a compatible device and operating system.

We expect to have hundreds of participants who fill out the questionnaire and/or install the software.

PROCEDURES:

This experiment consists of two optional parts.

The first part is a questionnaire, which asks you about how you use computers and the Internet and some specific technologies including RSS feeds and PodCasts. The questions do not request data that can be used to identify you individually. We estimate that filling out the questionnaire will take approximately 20 minutes.

In the second part, you run a logging program on one or more computers for approximately one year. Installing the program will take approximately 10 minutes. After installing the program, you will not need to interact with it again.

(If you don't have a smart phone and would like to borrow one, we have a limited number available. In this case, you need to fill out the questionnaire and indicate that you would like to borrow one. Based on the responses, we will select a number of people.)

The logging software collects data about the files you use, the programs you run, your network connectivity and battery status. Specifically, the collected data are:

1. File accesses (the filename of the accessed file, the type of access, create, delete, read, write, and the time of the access; the contents of the file will not be recorded);

2. Programs run (their names, the time at which they were started, and the time at which they exited);

3. Battery status (the available power as reported by the operating system and the time the sample was taken, when the device is attached to a power source and when it is detached);

4. Network connection statistics (the name of the network, anonymized using a one-way hash (explained in the section CONFIDENTIALITY, below), e.g., the cell phone provider's common name or the WiFi access point's SSID, when a connection is established, when a connection is disconnected, the number of bytes transferred, the connection's signal strength, and the time of the sample);

5. Available networks (the identity of networks, anonymized using a one way hash, available in the vicinity of the device, but which the device has not connected to, and the time stamp at which the sample was made);

6. Connected cell tower (time at which the device connect to the cell tower and the tower's identification including its location area code, the cell identifier, the network code and the country identifier, anonymized using a one-way hash, except the country identifier);

7. User activity (when the user starts interacting with the system, and when the user stops interacting with the system, e.g., when the screen saver is deactivated or activated);

8. System boot, shutdown, suspend and resume (the time when the system boots, when the time when the system is turned off, when the system is suspended and when it is resumed);

9. Web sites that the user visits, anonymized using a one-way hash for each URL component except for the name of the file, e.g., http://porn.com/transvestites/pic1.jpg would be reported as something along the lines of http://a1040356.3275c2/0016fff24/pic1.jpg; and,

10. Warnings and error messages emitted by the logging program (the time at which the message was emitted and the message itself; for debugging purposes).

The program is designed to be lightweight so as to minimally interfere with you. Moreover, the program will be run with a low priority.

Approximately once per day, the logging program will upload the data over an encrypted connection to hssl.cs.jhu.edu, a server under PI Burn's control. This will only be done if:

1. There is an ethernet or WiFi connection available; a cellular connection will never be used to avoid the chance of the logging program incurring monetary charges.
2. The user appears to not be interacting with the system (e.g., the screen saver is active).
3. The server's reported identity matches the expected identity (this is done using cryptographic techniques and ensures that the data is not accidentally reveal data to the wrong server).

If the data is successfully uploaded, the synchronized data will be deleted from the device thereby reducing the storage space used by the logging program on the device and avoiding the case that someone who later obtains the device gets access to the logged data.

The data will be identified by a random, secret identifier that is generated on the device when the program is first run.

When the experiment is completed, an updated version of the program will be made available, which should automatically be made available to you using the operating system's usual software update mechanism. This version of the program will delete any remaining logged data, disable logging, and remove the logging software.

RISKS/DISCOMFORTS:

The risks associated with participation in this study are minor; we try to be aware of, and carefully avoid, any problems that could arise for participants as a result of taking part in this research. Disclosure of data collected about you (detailed above in the section PROCEDURES) is the primary risk. The data that may have personally identifying information includes:

1. File names (the standard email client on the Nokia N900 includes the email address of the email account in the file name when saving email messages);
2. URLs of web pages the user visited; and,
3. Location information (names of WiFi access points and cellular

towers).

If collected data is exposed, you may be embarassed: the data may include URLs of web sites that embarass you, e.g., pornography.

A person who accesses your device (e.g., a thief) is able to access recently logged data. As data is anonymized prior to it being saved, the amount of information an attacker could gain is reduced. Further, this is limited to data logged since the last upload, which typically occurs every 24 hours, since logged data is deleted from the device. Some information an attacker could learn include: The names of files that you have recently deleted. If the person were interested in knowing whether you were at a particular location and knew the identity of cell phone towers in that area, that person could verify whether you were there at a particular time.

CONFIDENTIALITY:

Any study records that identify you will be kept confidential to the extent possible by law. The records from your participation may be reviewed by people responsible for making sure that research is done properly, including members of the Homewood Institutional Review Board of the Johns Hopkins University, or officials from government agencies such as the National Institutes of Health or the Office for Human Research Protections. (All of these people are required to keep your identity confidential.) Otherwise, records that identify you will be available only to people working on the study, unless you give permission for other people to see the records.

To protect your privacy, we partially anonymize the URLs and location information using a one-way hash (as described in the section PROCEDURES). A one-way hash is a deterministic procedure for transforming data. It has two important properties: given the hashed data, it is impractical to recover the original data; and, the same data always results in the same transformation. These properties allow us to determine how often participants frequency a particular location or web site without being able to identify the web site or location. However, because the hash transforms the same data in the same way, it is possible to verify the existence of particular data. For instance, determining what corresponds to "2q34lkjgfd.9808" is difficult. However, if someone with access to the data wants to determine whether a trace includes particular data, e.g., "porn.com," it is only necessary to compute the hash corresponding to "porn.com," perhaps, "2q34lkjgfd.9808," and checking if that data appears in the

trace.

Data collected from you will be stored on password protected computers. The data collected by the logging program will be transferred using an encrypted connection. The identity of the server will also be cryptographically verified. Ten years after the study has been completed, the data will be deleted.

BENEFITS:

You will not receive any direct benefits from participating in this study. This study may benefit society if the results lead to a better understanding of how data is used on mobile devices. Specifically, this study tries to increase data availability, improve the performance of data access, improve battery life, decrease internet connectivity costs, and facilitate managing data by determining which files to prefetch and which files are least valuable.

VOLUNTARY PARTICIPATION AND RIGHT TO WITHDRAW:

Your participation in this study is entirely voluntary: you choose whether to participate. If you decide not to participate, there are no penalties, and you will not lose any benefits to which you would otherwise be entitled.

If you choose to participate in the study, you can stop your participation at any time, without any penalty or loss of benefits. If you want to withdraw from the study, you need either stop filling out the questionnaire, or uninstall the software on your devices.

COMPENSATION:

You will not receive any payment or other compensation for participating in this study.

IF YOU HAVE QUESTIONS OR CONCERNS:

You can ask questions about this research study now or at any time during the study, by talking to the researcher(s) working with you, or by emailing Randal Burns (<mailto:randal@cs.jhu.edu>), the principal investigator for this study. If you have questions about your rights

as a research subject, or feel that you have not been treated fairly, please call the Homewood IRB at Johns Hopkins University at (410) 516-6580.

Intended to be blank.

Appendix B

Right Censored Power Laws

In this chapter, we start by reviewing heavy tailed distributions with a focus on power law distributions, a reoccurring concept in the chapter 3. This discussion is drawn heavily from [6, 55, 71, 72, 81]. We observe that the state of the art does not deal well with a downward deviation in the tail, which can occur due to an external dampening process. We then extend Clauset et al.'s methodology [6] for fitting and testing power law distributions to deal with this case.

B.1 Understanding Heavy Tailed Distributions

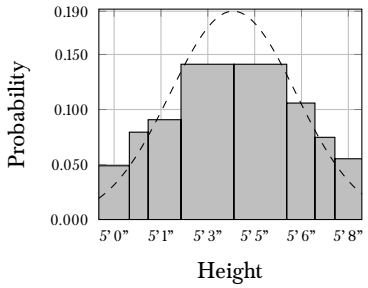
Informally, a heavy tailed distribution is a distribution that has no typical value. Whereas data drawn from a normal distribution cluster around a central value, data drawn from a heavy tailed distribution span multiple orders of magnitude.

To better grasp heavy tailed distributions, consider the distribution of city populations in contrast to the distribution of human heights. These two phenomena have been used in this way in Clauset et al. [6], Newman [55, 81] and Adamic and Huberman [71], for instance.

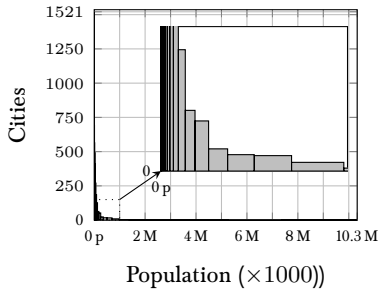
The height of adult humans is normally distributed and is typically between five and six feet. There are a few extremely short people who are less than three feet tall; and, there are a few extremely tall people who are over eight feet tall. Nevertheless, five to six feet is a good characterization of human height. A histogram of the height of American adult females as well as a fitted normal curve is shown in Figure B.1.

In contrast, the population of cities follows a power law distribution. According to the 2011 US census, the average American city has 9963 inhabitants.[1] This

[1]The census data only includes incorporated cities. One third of the US population (100 000 000 people) lives in unincorporated communities. The defining feature of an incorpo-



(a) Heights of US females over the age of 20 [85, Table 9]. Source data was provided in selected percentiles. The normal curve is a fit to that data.



(b) Histogram of the population of incorporated American Cities in 2011 [17]. The size of the histogram's bins grow exponentially. There are 9 cities with at least a million inhabitants. The largest city has a population of 8 244 910.

Figure B.1: Heights of American females and populations of American cities.

number doesn't do a good job of describing a typical American city: in the US, there are a few big cities with millions of inhabitants (where most people live), and many villages and towns, each of which has just a few thousand inhabitants. Unlike human heights, which are all the same order of magnitude, only 20.2% of the cities have a population that is the same order of magnitude as the mean!

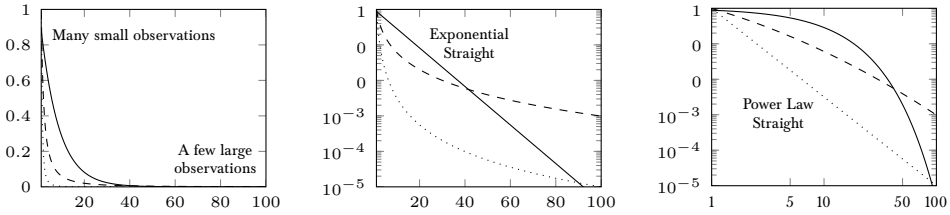
If human height were distributed according to a similar power law with the same average height as before, most people would be just a foot tall and there would be a few people who are over 100 000 feet tall! This is a factor of over 100 000 between the extremes. In reality, the extremes are a factor of four apart.

Although such a large spread is typical for a heavy tailed distribution, this is not what primarily differentiates it from a normal distribution. Unlike a normal distribution, a heavy tailed distribution is highly unbalanced: there are many observations with very small values and a few observations with very large values. The latter occur sufficiently often that they cannot be considered outliers.

rated city is local governance, whose overhead isn't justified in small communities. This explains the small number of very small cities in the data. The practical result is that the power law is left-truncated and appears reasonable for cities with a population above approximately 50 000. We conjecture that 50 000 is the point where local governance becomes advantageous. Including unincorporated communities would likely reduce the mean population by an order of magnitude.

Distribution	PDF: $P(X = x)$	CCDF: $P(X \geq x)$	Legend
Exponential	$\lambda \cdot e^{-\lambda x}$	$e^{-\lambda x}$	———
Log-Normal	$\frac{1}{x\sqrt{2\pi\sigma^2}} \cdot \exp\left(\frac{-(\ln(x)-\mu)^2}{2\cdot\sigma^2}\right)$	$0.5 - 0.5 \cdot \text{erf}\left(\frac{\ln x-\mu}{\sqrt{2\sigma^2}}\right)$	- - - - -
Power Law	$\frac{\alpha-1}{x_{\min}} \cdot \left(\frac{x}{x_{\min}}\right)^{-\alpha}$	$\left(\frac{x}{x_{\min}}\right)^{-\alpha+1}$	.....
Discrete	$\frac{x^{-\alpha}}{\zeta(\alpha,x_{\min})}$	$\frac{\zeta(\alpha,x)}{\zeta(\alpha,x_{\min})}$	*not shown*

(a) Three heavy tailed distributions.



(b) Linear axes. (c) Semi-log plot. (d) Log-log plot.

Figure B.2: Plots of the exponential, log-normal and power law heavy tailed functions. The exponential function appears as a straight line when plotted on a semi-log plot. The power law function appears as a straight line on a log-log plot. The log-normal function remains curved in all of them.

B.2 Example Heavy Tailed Distributions

There are many different heavy tailed distributions. Figure B.2 shows examples of three common right heavy tailed distributions—an exponential distribution, a log-normal distribution and a power law distribution.

In all three cases, the curve follows an L-shape when plotted on linear axes. This is illustrated in Figure B.2b. In practice, when plotting observations drawn from a heavy tailed distribution, the lines will appear to even more closely follow the x axis due to the near inevitable presence of large observations. We saw this in the plot of city populations in Figure B.1b. In Figure B.2, we chose the parameters and scaled the data so that it is easier to distinguish the curves.

The first plot, Figure B.2b, makes visually clear that these distributions have many small observations. We can easily confirm this analytically. Consider a power law distribution with $\alpha = 2$. (For $\alpha = 2$, $C = 1/\int x^{-2}dx = 1$.) 90% of

the samples drawn from this distribution will not exceed 10:

$$\begin{aligned} P(x \leq 10; \alpha = 2) &= \int_1^{10} x^{-2} dx \\ &= \left[-x^{-1} \right]_1^{10} \\ &= \left(-\frac{1}{10} + \frac{1}{1} \right) \\ &= 0.9 \end{aligned}$$

Because we constrained the x axis, it is harder to see that the tails are, in fact, heavy. Consider the probability that a sample will exceed 1000, which, as we have just established, is far away from the bulk of the samples:

$$\begin{aligned} P(x > 1000; \alpha = 2) &= \int_{1000}^{\infty} x^{-2} dx \\ &= 1 - \int_1^{1000} x^{-2} dx \\ &= 1 - \left[-x^{-1} \right]_1^{1000} \\ &= 1 - \left(-\frac{1}{1000} + \frac{1}{1} \right) \\ &= 0.001 \end{aligned}$$

This may seem like an unlikely event, however, if we have a thousand data points, the probability that at least one data point exceeds 1000 is $1 - P(X \leq 1000)^{1000} = 0.63$, which is quite likely. This is perhaps surprising given that 90% of the observations do not exceed ten.

Although the L-shape allows us to recognize a right heavy tailed distribution, the details are obscured. Plotting the data on logarithmic axes not only reveals more detail, but also makes it possible to visually distinguish several different types of heavy-tailed distributions.

In practice, the exponential and the power law distribution are the easiest to recognize by looking at the plot. When plotted on semi-logarithmic axes, the exponential distribution appears as a straight line. Similarly, when plotted on logarithmic axes, a power law distribution appears as a straight line. This can be seen in Figure B.2c and Figure B.2d, respectively.

Although both the exponential and power law distributions appear to intersect the x axis when drawn as a straight line, there is no horizontal cutoff: they are just asymptotically approaching the x axis; it is the y axis that is cutoff before reaching zero; the smaller y becomes, the further to the right the line extends.

Note: Clauset et al. warn that data that appears straight in a log-log plot is not necessarily drawn from a power law distribution. In particular, the fact that data forms a roughly straight line is a necessary, but *not* a sufficient, condition to conclude that the data is drawn from a power law distribution. To gain confidence that a power-law distribution is a plausible model, it is imperative to perform a goodness of fit test [6]. We discuss how to perform this goodness of fit test shortly.

B.3 Visualizing Power Law Data

We first consider how to effectively visualize data that appears to be drawn from a power law distribution. Our decision to focus on the power law distribution stems from the fact that much of the data that we examine appears to be drawn from a power law distribution.

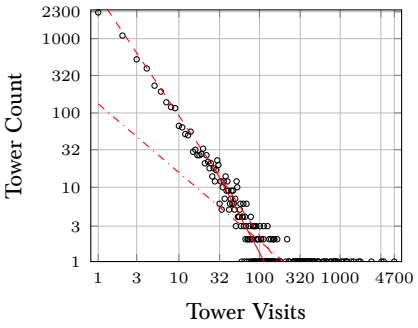
Figure B.3 shows three plots of the same data—the number of times that a user connects to different cell towers. The three plots show the same relationship. The data is just displayed differently.

The first plot is a Pareto plot in which the x axis is size and the y axis is frequency. In this plot, the x axis is the number of times a tower was visited and the y axis is the number of towers for which that is true. To find the number of towers that the user visited exactly 100 times, we find the y that corresponds to $x = 100$. If we were instead plotting city sizes, as in Figure B.1b, population would be on the x axis and city count would be on the y axis.

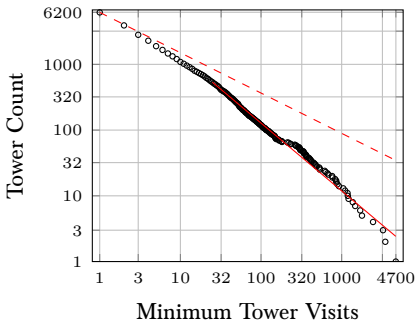
The second plot is a complementary cumulative Pareto plot. This differs from the previous plot in that the x axis is now the minimum size. Thus, $y(x = 1)$ is the number of towers that were visited at least once (rather than exactly once as in the Pareto plot), which is the total number of towers seen over the course of the trace. To find the number of towers that were visited exactly once, we would need to compute $y(2) - y(1)$. If we are interested in the number of visits to the 10 most visited towers, then we would look up $y^{-1}(10)$, i.e., we would find the point for which $y = 10$ and then look up its x value.

The last plot in the figure is a Zipf plot. A Zipf plot is a Pareto plot with the x and y axes flipped and instead of the x axis being frequency, it is rank. Thus, $x = 10$ refers to the 10^{th} most visited tower and the corresponding y value is the number of times the user visited that tower.

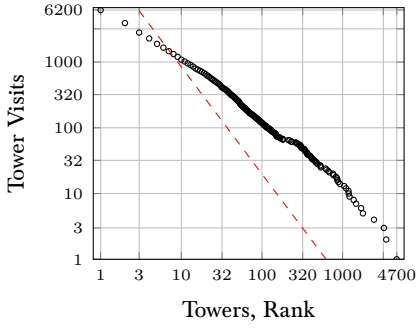
A Pareto plot is a frequency plot: x is some attribute, typically a size, and y is a count. On the left-hand side of a Pareto plot, it is easy to make out a linear relationship between size and frequency. The tail, however, appears to be very noisy. This is due to the inevitable sparsity of data points in that region: because



(a) Pareto: Size vs. Frequency ($P(X = x)$).



(b) Complementary cumulative Pareto: Minimum Size vs. Frequency ($P(X \leq x)$).



(c) Zipf: Rank vs. Size.

Key	Method	α	$x_{\min}$
	MLE	2.03	25
	MLE, Fixed $x_{\min}$	1.61	1
	OLS	0.92	1

(d) Legend.

Figure B.3: Three ways to visualize data drawn from a power law distribution.

non-integral counts are impossible, we shift from a regime of rapidly decreasing counts on the left, which nicely follow a straight line, to a regime of rapidly increasing stretches of zeros separated by singletons on the right, which appear not to follow a straight line. The issue is that the number of observations of any particular large value of x approaches zero for finite sample sizes. For instance, the expected number of observations for which $x = 100$ for 1000 samples from

a discrete power law distribution with $\alpha = 2$ is:

$$\begin{aligned} n &= 1000 \\ p = P(X = 100; \alpha = 2) &= 0.0000608 \\ E(x = 100; n, p) &= \sum_{k=0}^{n} k \cdot \left[\binom{n}{k} p^k \cdot (1-p)^{n-k} \right] \\ &= np \\ &= 0.0608 \end{aligned}$$

where n is the number of samples and p is the probability that $X = 100$. However, as we have already seen, for a range of large x values, the probability that at least one will be observed is likely. In our example, we expect to observe more than 6 values that are at least 100 ($P(X \geq 100) = 0.00611 \implies E(X \geq 100) = 6.1$). In these cases, the singleton observation significantly exceeds the expected value (which is near zero) and thus appears as a spike in the plot. This phenomenon is clearly seen in Figure B.3: between 1000 and 4684, the largest observation, there are just 13 observations.

Plotting the data as a complementary cumulative Pareto plot smooths out the bumps in the tail as seen in Figure B.3b. Unlike a histogram in which data is binned, this transformation does not lose any information. The smoothing occurs because we display the cumulative frequency rather than the frequency of individual items. For data that approximately fits a power law, the cumulative frequency is much closer to the expected cumulative frequency than the frequency is to the expected frequency.

Whereas the complementary cumulative Pareto plot emphasizes the common small events, the Zipf plot, also referred to as a rank-frequency plot, emphasizes the rare large events. As seen in Figure B.3c, a Zipf plot makes it easier to discern the details of large events on the left-hand side of the plot, but the common events are now squished together on the right-hand side. In our experience, Pareto plot's are often easier to interpret: it seems more natural to think about the size of the xth largest observation than the number of observations with a particular size. Mathematically, it doesn't matter: the two plots are equivalent [72].

In this thesis, we use the Pareto complementary CDF plot to display data that appears to follow a power law. We opt not to use the Pareto plot due to the visual noise in the tail. And, although we find the Zipf plot easier to interpret, we reject it, because it appears to be less standard. This is perhaps because the axes match the variables in the power law equation.

B.4 Fitting Data to a Power Law

Once we've observed that a data set roughly follows a straight line on a log-log plot, our next step is to identify the best fit. There are three parameters that we need to identify: the power law's steepness (α), the lower bound ($x_{\min}$) and the upper bound ($x_{\max}$).

B.4.1 Estimating the Scaling Parameter

An obvious choice for estimating α is to use ordinary least squares (OLS) to fit a straight line to the transformed data. We can recast the power law equation to be in the form of a straight line as follows:

$$\begin{aligned} y &= Cx^{-\alpha} \\ \log(y) &= \log(Cx^{-\alpha}) \\ \log(y) &= \log(C) + \log(x^{-\alpha}) \\ \log(y) &= \log(C) - \alpha \log(x) \end{aligned}$$

The slope of the line is $-\alpha$ and $\log(C)$ is the y intercept.

Unfortunately, OLS can result in a bad fit, which can be seen in Figure B.3a. To understand why OLS is inappropriate, consider the overly simplified case where we have two data points on our plot: $(1, 1000)$ and $(1000, 1)$, i.e., we have 1000 observations with size 1 and 1 observation with size 1000. OLS will fit a line through the two points. A better fit, though, is not a line that passes through the two points, but one that is nearly vertical at $x = 1$! This is because $(1, 1000)$ isn't a single data point, but a 1000 data points—y is frequency. Due to our formulation, OLS doesn't take this into account.

A better approach is to use maximum likelihood estimation (MLE). In MLE, we select the parameters (θ) that are most likely given the data ($\mathbf{x}$). That is, we compute $\operatorname{argmax}_\theta \mathcal{L}(\theta \mid \mathbf{x})$. $\mathcal{L}(\theta \mid \mathbf{x})$ is the likelihood function and is defined as $P(\mathbf{x} \mid \theta)$. Note that $\mathbf{x}$ is a vector of observations. Thus, we are computing the joint probability of all of the observations, i.e., $P(x_1, x_2, \ldots, x_{|0\mathbf{x}|0} \mid \theta)$. If the observations are independent, as they usually are, then we can factor the joint distribution to be the product of the probability of the individual events:

$$\mathcal{L}(\theta \mid \mathbf{x}) = \prod_{i=1}^{|0\mathbf{x}|0} P(X = x_i \mid \theta) \tag{B.1}$$

Typically, we don't deal directly with the likelihood function, but with the log

likelihood function:

$$\ell(\theta \mid \mathbf{x}) = \sum_{i=1}^{|0\mathbf{x}|0} \log P(X = x_i \mid \theta) \tag{B.2}$$

This form is easier to deal with mathematically and has fewer numeric problems—we don't need to potentially multiply hundreds of numbers between 0 and 1, which can result in an underflow. Consider computing the likelihood of 1000 flips of a fair coin: $\mathcal{L}(H = 0.5 \mid 1000 \text{ flips}) = 0.5^{1000} \approx 10^{-302}$. Although this sample isn't terribly large, this value already poses problems for floating point numbers. For instance, if we try to compute the complement, i.e., $1 - 10^{-302}$, we get exactly 1! This problem occurs when dealing with numbers whose magnitudes differ by just 10^{16}.

To understand MLE, consider estimating the probability that flipping a potentially biased coin comes up heads (H). Say that we flip the coin five times and it comes up heads every time. To find H using MLE, we compute $\operatorname{argmax}_H \mathcal{L}(H \mid hhhhh)$. The likelihood that flipping a fair coin results in five heads is $\mathcal{L}(H = 0.5 \mid hhhhh) = 0.5^5 = 0.031$, which is not very likely, but also not impossible. The likelihood that a coin with two heads generates five heads is $\mathcal{L}(H = 1 \mid hhhhh) = 1^5 = 1$, which is, in fact, the maximum likelihood for this data.

The MLE can appear overly confident when the sample size is small (do we really believe with absolute certainty that the coin has two heads after just five trials?). As the sample size grows, however, this problem becomes less significant.

We can systematically determine the most likely parameters by either finding a closed form expression for θ or by using numeric optimization. Both of these exploit the fact that the likelihood function is concave downward. To find a closed form expression, we set the derivative of the likelihood function equal to zero and solve for θ. Using numeric optimization, we evaluate $\operatorname{argmax}_\theta \mathcal{L}(\theta \mid \mathbf{x})$ for the possible values of θ. Because the function is concave, we don't need to check all possible values. Instead, we can use the robust and trivial to implement golden section method, which is essentially a simple binary search (although more advanced techniques are known: refer to any book on numeric optimization, e.g., [42, Chapter 10]). If a variable is continuous, then we stop when the region containing the most likely value is sufficiently small.

To find the likelihood function, we simply substituted the relevant probability function (see Figure B.2) into the general likelihood function (Equation B.1) (see also [6, Section 3]). For a continuous power law the resulting likelihood function

is:

$$\mathcal{L}(\alpha \mid \mathbf{x}) = \prod_{i=1}^{|0\mathbf{x}|0} \frac{\alpha - 1}{x_{\min}} \left(\frac{x_i}{x_{\min}} \right)^{-\alpha} \tag{B.3}$$

And, for a discrete power law the likelihood function is:

$$\mathcal{L}(\alpha \mid \mathbf{x}) = \prod_{i=1}^{|0\mathbf{x}|0} \frac{x^{-\alpha}}{\zeta(\alpha, x_{\min})} \tag{B.4}$$

A closed form expression for α exists for the continuous case, however, none is known for the discrete case.

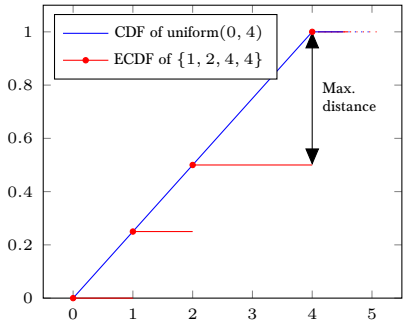
B.4.2 Estimating the Lower Bound

The likelihood functions in Equation B.3 and Equation B.4 contain an unbound variable, $x_{\min}$. $x_{\min}$ is the minimum value of x for which the power law holds. For a pure power law distribution, this is 1. However, it is often the case that the data only follow a power law after x exceeds some threshold. Indeed, even for data drawn from a power law, statistical fluctuations can mean that the best estimate of $x_{\min}$ is larger than 1. Drawing 1000 samples from a continuous power law with $\alpha = 2$ and fitting both $x_{\min}$ (using the method described below) and α results in values of $x_{\min}$ that discard approximately 13% of the data, on average.

We already saw an example of a power law with $x_{\min} \gg 1$. In the city population data [17], which we discussed in Section B.2, we observed that the power law only holds for cities with populations greater than approximately $50\,000$: for smaller communities, the overhead of managing a city is perhaps less than its benefits.

Another, more relevant example, is how long a user stays connected to a cell tower. Switching to a new tower happens not only when there is a tower with a better signal, but when there is a tower with a ***significantly*** better signal. This threshold limits the signalling that is incurred by a tower handoff. The practical result is that stationary users tend to stay connected to a tower for some minimum amount of time. There is also a physical lower limit on the amount of time that the user can stay connected to a tower: signal propagation and processing takes time.

Clauset et al. warn that it is important to choose $x_{\min}$ appropriately [6, Section 3.3]. If we don't, we run the risk of including data that is drawn from a different distribution if $x_{\min}$ is too small or throwing away relevant data if $x_{\min}$ is too large. This can lead to a poor fit.



(a) Plots of the CDFs.

x_i	$\lvert F(x_i) - G(x_i)\rvert$	$\lvert F(x_i) - G(x_{i-1})\rvert$
1	$1/4 - 1/4$	$1/4 - 0$
2	$1/2 - 1/2$	$1/2 - 1/4$
4	$1 - 1$	$1 - 1/2$
4	$1 - 1$	$1 - 1$

(b) Calculation of the KS statistic.

Figure B.4: Computing the KS statistic of the data set $\{1, 2, 4, 4\}$ and the continuous uniform distribution over 0 through 4. Just comparing $F(x_i)$ and $G(x_i)$ would yield $D = 0$. We also need to compare $\lim_{x \to x_i} F(x)$ and $G(x_{i-1})$ to find the actual maximum distance of 0.5.

We can't use MLE to find $x_{\min}$: varying $x_{\min}$ changes the amount of data that is used and we can't meaningfully compare likelihoods based on different amounts of data. To choose $x_{\min}$, Clauset et al. propose finding the best α for each candidate $x_{\min}$ and then choosing the set of parameters with the smallest Kolmogorov-Smirnov (KS) statistic relative to the data [6, Section 3.3].

The KS statistic is the maximum distance between two CDFs:

$$D = \sup\left(\lvert F(x) - G(x)\rvert\right) \tag{B.5}$$

Note: $\sup(x)$ is the supremum function. It returns the smallest value that is larger than x. It is also referred to as the least upper bound (LUB). For real numbers, $\sup(0)$ denotes the smallest number that is larger than 0, which we can't write down. For integers, $\sup(x)$ is just $x + 1$.

To compute the supremum between the best fit distribution (F) and the empirical distribution (G), which is discrete, we need to find the maximum distance between $F(x_i)$ and $G(x_i)$ and $\lim_{x \to x_i} F(x)$ and $G(x_{i-1})$:

$$D = \max_i \left[\max\left(\lvert F(x_i) - G(x_i)\rvert, \lvert \lim_{x \to x_i} F(x) - G(x_{i-1})\rvert\right)\right] \tag{B.6}$$

When F is continuous, $\lim_{x \to x_i} F(x) = F(x_i)$ and

$$D = \max_i \left[\max\left(\lvert F(x_i) - G(x_i)\rvert, \lvert F(x_i) - G(x_{i-1})\rvert\right)\right] \tag{B.7}$$

Figure B.4 shows a simple scenario illustrating the necessity of incorporating the comparison of $\lim_{x \to x_i} F(x)$ and $G(x_{i-1})$. This point is subtle and was missed

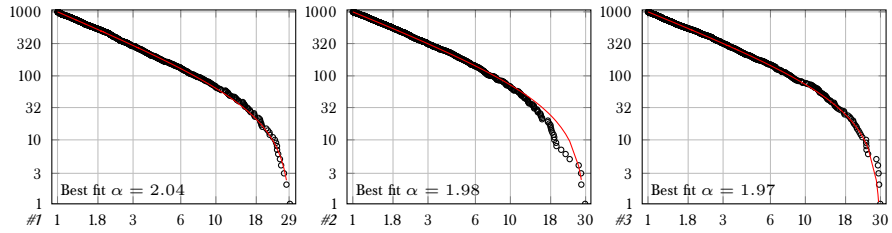


Figure B.5: Examples of data drawn from a right truncated power law distribution with $\alpha = 2$, $x_{\min} = 1$ and $x_{\max} = 30$. The regression was fit using the right truncated likelihood function, which we present in Equation B.11.

by Clauset et al. in their implementation [22]. Unfortunately, most textbooks only provide the definition of the KS statistic (i.e., Equation B.5) and don't show a form that is appropriate for numeric evaluation. Nevertheless, we are confident that this expansion is correct: Arnold and Emerson [3], Trivedi [104, Page 718] and Press et al. [42, Page 737] all agree on this form. In cases where we use Clauset et al.'s implementation, we use a version in which this error is corrected.

Arnold and Emerson note that if the theoretical distribution F is also discrete, then $\lim_{x \to x_i} F(x) \neq F(x_i)$. Instead, we need to replace $\lim_{x \to x_i} F(x)$ with $F(x_i - \epsilon)$ for some value of ϵ that does not exceed the smallest step [3]. For a discrete power law, the step size is always 1. So:

$$D = \max_i \left[\max \left(|F(x_i) - G(x_i)|, |F(x_i - 1) - G(x_{i-1})| \right) \right] \tag{B.8}$$

Unfortunately, unlike the likelihood function, D is not necessarily concave so we can't easily optimize the search for the best value of $x_{\min}$; we need to check all possibilities to find the optimal value.

B.4.3 Incorporating an Upper Bound

Just as there is sometimes a lower bound on the power law, there can also be an upper bound. This occurs if there is some natural upper limit on the process. For instance, diurnal effects, such as opening and closing times, can limit the maximum dwell time at a particular location. The presence of an upper bound manifests itself as a downward deviation in the tail. This deviation can be seen in the plots of some synthetic data sets drawn from a power law with $\alpha = 2$ and truncated at $x = 30$ along with their regressions to a right truncated power law in Figure B.5.

We extend Clauset et al.'s methodology to work with power law data with an upper limit. We consider two methods for dealing with an upper bound:

truncating the data at the upper bound, i.e., ignoring all of the data above $x_{\max}$; and, treating the data above the upper bound as if it were right censored, i.e., viewing the data above $x_{\max}$ as a lower bound on the true value. Both cases require modifying the likelihood function and estimating $x_{\max}$.

Right Truncation

Aban et al. have studied right truncated power laws [13]. A right truncated power law is a power law in which events larger than a particular threshold are suppressed. To better intuit the behavior of a right truncated power law consider that we can synthesize a data set by drawing a sample from an untruncated power law and throwing away all of the data above the upper limit.

The CCDF for a right truncated power law is:

$$P'(X \geq x; \alpha, x_{\min}, x_{\max}) = \frac{P(X \geq x) - P(X > x_{\max})}{1 - P(X > x_{\max})} \tag{B.9}$$

and its density function is:

$$P'(X = x; \alpha, x_{\min}, x_{\max}) = \frac{P(X = x)}{1 - P(X > x_{\max})} \tag{B.10}$$

with the constraint that $x_{\min} \leq x \leq x_{\max}$.[2] $P(\cdot)$ is the probability of the expression in the untruncated model (see Figure B.2a).

This formulation makes intuitive sense. Values above $x_{\max}$ are impossible. Thus, if we are interested in, say, $P'(X > x)$, we find $P(X > x)$ (the probability in the untruncated model) and exclude the probability mass above $x_{\max}$. Of course, since we want a probability distribution, we need to normalize the result so that the integral is 1. This is the purpose of the denominator.

The likelihood function is again a straightforward combination of the general likelihood function (Equation B.1) and the density function (Equation B.10):

$$\mathcal{L}(\theta \mid \mathbf{x}) = \prod_{i=1}^{|0\mathbf{x}|0} \frac{P(X = x_i)}{1 - P(X > x_{\max})} \tag{B.11}$$

Unfortunately, even in the continuous case, there is no exact closed form solution.

Aban et al. consider how to estimate the parameters when the data is pure and when there is a deviation on the left. In the latter case, we need to remove this data from the sample. They recommend identifying the end of the deviation

[2] We rewrote the equation using our notation. Aban et al.'s parameters in terms of our parameters are $\gamma = x_{\min}$, $\nu = x_{\max}$ and $\alpha = \alpha - 1$.

"on the basis of a log-log plot ... so that [the amount of used data] is as large as possible as long as the model fit is adequate" [13, Section 4]. Unfortunately, they don't provide any advice on how to determine whether a model fit is adequate.

Aban et al. don't consider varying $x_{\max}$. If $x_{\max}$ is not known a priori, they set it to the largest observed value, which, they note, is its maximum likelihood estimate. We observe that it can be useful to remove this restriction and select a value of $x_{\max}$ that excludes some of the data at the very end of the tail. For instance, if we think that data above a particular threshold is generated by a mixture, we can simply set the upper bound to that threshold and fit the data below the threshold to a right truncated power law. We can then use the estimated parameters to better decompose the mixture.

We can easily adapt Clauset et al.'s use of the KS statistic for estimating $x_{\min}$ to estimate both $x_{\min}$ and $x_{\max}$: nothing special is required to compute the empirical CDF for the data or the CDF of a right truncated power law. We check the effectiveness of this test below.

Right Censoring

An alternative way to view the downward deviation in the tail is as the presence of an external dampening process. That is, there is some external process that is working to prevent the most extreme values from taking their "true" value. In fact, Aban et al. use this type of language to describe the cause of the truncation in at least two of the three examples that they present. In particular, they explain that the most extreme values of a stock's price are likely suppressed by automatic safe guards implemented by the stock exchange to slow trading when extreme price fluctuations occur and that in hydraulic conductivity, "high-flow channels can be occluded with sedimentation" thereby inhibiting very large flows [13, Section 4]. The dampening explanation also makes more intuitive sense than the truncated process: in the truncated case, the events that lead to the largest values are suppressed before they occur; in the dampened case, they still occur, but an external process limits them once they reach a certain size. Aban et al. briefly mention this explanation at the end of their paper, but don't study it in depth.

Given this realization, we can instead view the upper threshold as a censoring point. That is, we treat the data points above the threshold as representing a minimum value rather than the true value. (Note that this threshold is different from the upper threshold in the truncated case. In the truncated case, the upper threshold is the maximum value that we expect to observe; in the censored case, the upper threshold is the point at which the external process begins to take effect, i.e., the start of the downward deviation.) This approach appears

promising: van der Vaart notes that even though censored data contains much less information than non-censored data, taking it into account can result in a considerable improvement in the estimation of a distribution's parameters [119]. This approach appears to also automatically handle mixture distributions, such as power laws with an exponential cutoff, in a nonparametric manner.

Klein and Moeschberger [58, Section 3.5] and Patti et al. [97] describe a framework for dealing with right censored data. They observe that observations of censored data are determined by two processes. The first process is the one that determines the data's true value. We model this process using random variable T (for true). The second process determines the censoring point. We model this process using the random variable C (for censoring point). When we make an observation, we don't directly observe T and C. Instead, we observe the minimum of the two, which we denote by X, as well as whether the data was censored, which we denote by Δ.[3] Δ is a binary variable and, by convention, 0 means that the observation was censored and 1 means that it was not censored. Thus:

$$X = \min(T, C) \tag{B.12}$$

$$\Delta = \begin{cases} 0 & \text{if } T > C \quad \text{Censored (true value exceeds censoring point)} \\ 1 & \text{if } C \geq T \quad \text{Not censored (censoring point exceeds true value)} \end{cases} \tag{B.13}$$

Given this formulation, the probability of an observation is $P(X, \Delta)$. We can factor the joint probability as follows:

$$P(X = x, \Delta = \delta) = \underbrace{P(X = x, \Delta = 0)}_{\text{censored}}{}^{1-\delta} \cdot \underbrace{P(X = x, \Delta = 1)}_{\text{not censored}}{}^{\delta} \tag{B.14}$$

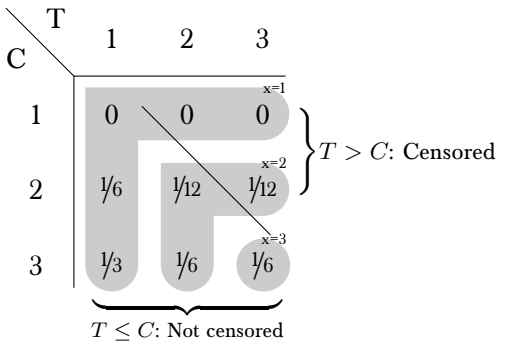
Δ acts as a switch that selects $P(X = x, \Delta = 0)$ if $\Delta = 0$ (censored) and $P(X = x, \Delta = 1)$ if $\Delta = 1$ (not censored).

These two factors are easy to define in terms of what we know. First, consider $P(X = x, \Delta = 1)$, i.e., an uncensored observation. Since the observation is not censored ($\Delta = 1$), the true value does not exceed the censoring point, i.e., $C \geq T$ (Equation B.13). Further, since the observed value is the minimum of the true value and the censoring point, i.e., $X = \min(T, C)$ (Equation B.12) and, as just noted, the true value does not exceed the censoring point, the observed

[3] We use different variable names from Patti et al. to provide more intuitive names and to be consistent with the notation we've used so far. In particular, our T (true value) corresponds to their U, our C (censoring point) to their V and our X (observed value) to their T. Klein and Moeschberger swap T and X.

	1	2	3
$P(T)$	½	¼	¼
$P(C)$	0	⅓	⅔

(a) Example $P(T)$ and $P(C)$.



(b) $P(T, C) = P(T) \cdot P(C)$. The upper triangle, $T > C$, corresponds to censored observations ($\Delta = 0$); the lower triangle and main diagonal, $T \leq C$, correspond to non-censored observations ($\Delta = 1$). The grey bands correspond to the different values of $X = \min(T, C)$.

Figure B.6: Illustration of how the observed right censored variables X and Δ are related to the underlying processes T and C for a toy data set.

value corresponds to the true value, i.e., $X = T$. Plugging these into the initial expression yields:

$$\begin{aligned} P(X = x, \Delta = 1) &= P(T = x, C \geq T) \\ &= \int_{c=x}^{\infty} P(T = x, C = c)\, dc \end{aligned}$$

This is $P(T = x)$ excluding those points that are censored. Figure B.6 illustrates this using a toy data set.

If T and C are independent, which is likely the case if C corresponds to an external dampening process, then we can factor the above as follows:

$$\begin{aligned} \int_{c=x}^{\infty} P(T = x, C = c)\, dc &= P(T = x) \cdot \int_{c=x}^{\infty} P(C = c)\, dc \\ &= P(T = x) \cdot P(C \geq x) \end{aligned}$$

The last term is the CCDF of C.

The censored factor in Equation B.14, $P(X = x, \Delta = 0)$, is defined similarly.

Thus:

$$P(X, \Delta) = \underbrace{[P(C = x)\, P(T > x)]}_{\text{censored}}^{1-\delta} \cdot \underbrace{[P(T = x)\, P(C \geq x)]}_{\text{not censored}}^{\delta} \tag{B.15}$$

To better intuit why these scaling factors are necessary, consider a situation in which values of x that exceed $x_{\max}$ are censored and consider a point x, such that $x > x_{\max}$, i.e., a point that is necessarily censored. If $P(T)$ corresponds to an untruncated power law, then $P(T)$ is non-zero for all values of x greater than $x_{\min}$, which includes those values that exceed $x_{\max}$. However, we know that $P(X = x, \Delta = 0) = 0$ since $x > x_{\max}$ and all points that exceed $x_{\max}$ are censored. Multiplying $P(T = x)$ by $P(C \geq x) = P(x_{\max} \geq x) = 0$ corrects this.

Patti et al. use a slightly different proof to arrive at the same conclusion. Unfortunately, they make two mistakes, which fortuitously cancel each other out. In particular, they factor $P(X = x, \Delta = 1)$ as $P(X = x \mid \Delta = 1)\, P(\Delta = 1)$. This is fine. However, they then precede to equate $P(X = x \mid \Delta = 1)$ with $P(T = x)$ (their Equation 26), but $P(X = x \mid \Delta = 1) = P(T = x)\, P(x \leq C)/P(\Delta = 1)$. This error is cancelled out by their definition of $P(\Delta = 1) = P(x \leq C)$ (their Equation 27). But, $P(\Delta = 1) = P(C \geq T)$, by definition. Their equality incorrectly assumes knowledge of X; $P(\Delta = 1 \mid X = x) = P(x \leq C) \neq P(\Delta = 1)$. They make the same mistakes in the censored case, $P(X = x, \Delta = 0)$.

We improved our confidence that our proof (and Klein and Moeschberger's proof [58]) is correct and that Patti et al.'s mistakes are genuine using a Monte Carlo simulation. We sampled a million values of T and C for different distributions of T and C and used these to compute various empirical distributions. We compare these to their theoretical counterparts, as derived above. In particular, we checked:

$$\begin{aligned} P(X = x, \Delta = 1) &= P(T = x)\, P(x \leq C) \\ P(X = x, \Delta = 0) &= P(C = x)\, P(x < T) \\ P(X = x \mid \Delta = 1) &\neq P(T = x) \\ P(X = x \mid \Delta = 0) &\neq P(C = x) \end{aligned}$$

Example output and the simulation's code is provided in Appendix C.

Plugging the joint probability of the observed values (Equation B.15) into the

general likelihood function (Equation B.1), yields:

$$\mathcal{L}(\theta \mid \{\mathbf{x}, \boldsymbol{\delta}\}) = \prod_{i=1}^{|0\mathbf{x}|0} [\underbrace{P(C = x_i)\, P(x_i < T)}_{\text{censored}}]^{1-\delta_i} \cdot [\underbrace{P(T = x_i)\, P(x_i \leq C)}_{\text{not censored}}]^{\delta_i} \tag{B.16}$$

We know T's distribution: that's the untruncated power law distribution. Further, based on our problem description, we know whether a point is censored or not: if it greater than $x_{\max}$ it is censored, otherwise, it is not censored. The remaining piece is C, the censoring point's distribution.

We've observed that the censoring is due to a dampening process that comes into effect above a particular threshold. Thus, C does not share any parameters with T; it is not a function of α. As such, when computing the likelihood of α, terms based on C will appear as a constant and won't influence the optimization process's selection of α. Accordingly, we can drop these terms. This results in the following likelihood function:

$$\mathcal{L}(\alpha \mid \{\mathbf{x}, \boldsymbol{\delta}\}) \propto \prod_{i=1}^{|0\mathbf{x}|0} \underbrace{P(x_i < T)}_{\text{censored}}^{1-\delta_i} \cdot \underbrace{P(T = x_i)}_{\text{not censored}}^{\delta_i} \tag{B.17}$$

Where, again, $P(\cdot)$ refers to the probability of the expression for an untruncated power law.

Van der Vaart warns that "the fact that the maximum likelihood estimator based on the 'good' observations ... behaves very well, does not guard against bad behavior of the maximum likelihood estimator in the model with both 'good' and 'bad' observations" [119]. Based on this observation, we also consider a weaker version of the likelihood function.

To weaken Equation B.17, we only consider the number of points that exceed the threshold; we don't view that data as a lower bound. To do this, we simply cap all observations at $\sup(x_{\max})$. This formulation effectively results in all of the probability mass in C being concentrated at a single point:

$$P(C = x) = \begin{cases} 0 & \text{if } x = \sup(x_{\max}) \\ 1 & \text{if } x \leq x_{\max} \end{cases} \tag{B.18}$$

$$\mathcal{L}(\alpha \mid \{\mathbf{x}, \boldsymbol{\delta}\}) \propto \prod_{i=1}^{|0\mathbf{x}|0} \underbrace{P(x_{\max}) < T}_{\text{censored}}^{1-\delta_i} \cdot \underbrace{P(T = x_i)}_{\text{not censored}}^{\delta_i} \tag{B.19}$$

Note that x_i does not enter the censored term; it has been replaced by $x_{\max}$.

$x_{\min}$	$x_{\max}$	α	D	oom
5	22	1.05	0.074	0.64
23	154	1.05	0.030	0.83
241	1379	1.32	0.041	0.76
1266	10 113	1.53	0.025	0.90
9131	55 427	1.83	0.039	0.78
65 895	∞	2.40	0.025	2.1

Table B.1: The best, mostly non-overlapping fits for the population data. Several fits have comparably small values of D, however, the last fit in the table is clearly the best fit: it spans over two orders of magnitudes; the other fits span less than an order of magnitude.

We now consider how to estimate $x_{\min}$ and $x_{\max}$. Recall that Clauset et al. identify the best $x_{\min}$ by first estimating α for each candidate $x_{\min}$ and then taking the $x_{\min}$ whose KS statistic with respect to the observed data is smallest (see Equation B.5). That is, $\operatorname{argmin}_{x_{\min}} \mathrm{KS}(P(X \leq x; \alpha_{x_{\min}}, x_{\min}), \text{data})$, where $\alpha_{x_{\min}}$ is $x_{\min}$'s most likely α.

In the truncated case, we observed that we can use this method essentially without modification: we just need to compute the theoretical and empirical CDFs, which is straightforward. Now, however, our observations consist of two types of data: the true values and the censored values. Fleming et al. have generalized the KS test to work with data sets consisting of a mix of uncensored and right censored data [100]. They note, however, that "if particularly heavy censorship beyond some [point] τ is expected, one has the capability only to test [the values] $0 \leq t \leq T'$, where $T' \equiv \min([\max(\text{data})], \tau)$" [100, Section 2.4]. In our case, the values beyond $x_{\max}$ are not only heavily censored, they are completely censored. Moreover, there are no censored data points less than or equal to $x_{\max}$. Thus, to compute the KS statistic for censored data, we just compute the empirical CDF though $x_{\max}$ (being careful to weigh the probability by all of the observations and not just those through $x_{\max}$) and compare this to the best fit power law through $x_{\max}$.

B.4.4 Multiple Fits

When we allow both $x_{\min}$ and $x_{\max}$ to vary, it is possible that multiple, non-overlapping pieces of a data set will appear to be distributed according to a power law. This is the case for the population data, which we looked at in Sec-

tion B.2. Table B.1 shows the best mostly non-overlapping fits for that data set. We see that there are several, non-overlapping fits with comparably small values of D. One fit, however, spans over two orders of magnitude and is clearly the best fit.

Based on this observation, when fitting data to a power law, we select the best mostly non-overlapping fits according to the KS statistic and then return the fit that has the largest dynamic range.

In particular, we first find the optimal fit according to the KS statistic. We then keep just those candidate $x_{\min}/x_{\max}$ pairs whose value of $x_{\max}$ is at most 1.1 times the best fit's $x_{\min}$ or whose value of $x_{\min}$ exceeds 0.9 times the best fit's $x_{\max}$. We repeat this process until the search space is empty. Given the best fits, we then eliminate any fits whose KS statistic is larger than twice the best fit's KS statistic and return the fit that covers the most orders of magnitude as defined by the following function:

$$\text{oom} = \log_{10}\left(\frac{\max(\textbf{data})}{\min(\textbf{data})}\right) \qquad x_{\min} \leq \textbf{data} \leq x_{\max} \tag{B.20}$$

B.4.5 Comparing the Likelihood Functions

We've formulated three approaches to dealing with an upper bound. In this section, we compare their ability to recover α.

To test the likelihood functions, we generated 100 data sets each consisting of 1000 data points drawn from a particular power law. We considered both continuous and discrete power laws with $\alpha = 1.7$ and $\alpha = 2.3$. When fitting the data, we fixed the values of $x_{\min}$ and $x_{\max}$ to avoid confounding variables. When truncating or censoring the data, we truncated or censored approximately 5% of the data (the details are presented below). We chose relatively small, but still typical, values of α to ensure a fair amount of spread in the data given the amount of truncating and censoring. For instance, whereas the 0.95 quantile of a discrete power law with $\alpha = 1.7$ is 43, it is just 6 for a discrete power law with $\alpha = 2.3$. Table B.2 lists the summary statistics; Figure B.7 shows histograms of the estimated αs for $\alpha = 1.7$.

To establish a reference point, we first fit the untruncated data using the untruncated likelihood function (Equation B.3). This does an excellent job of recovering α; it is both precise and accurate.

We next truncated the data by discarding any data that fell above the 0.95 quantile. We first used the untruncated likelihood function to estimate α. Although the results are nearly as precise as in the untruncated case (the standard deviation is a bit larger, but this can be attributed to the reduced amount of data used to compute the fit), it's precision is poor: it significantly

Scenario	$\alpha = 1.7$			$\alpha = 2.3$		
	μ	σ	$\overline{\lvert\hat{\alpha} - \alpha\rvert}$	μ	σ	$\overline{\lvert\hat{\alpha} - \alpha\rvert}$
Untruncated Continuous	1.70	0.023	0.019	2.30	0.042	0.032
Discrete	1.70	0.025	0.020	2.30	0.046	0.037
Naïve Continuous	1.83	0.022	0.13	2.54	0.042	0.24
Discrete	1.84	0.026	0.14	2.59	0.046	0.29
Truncated Continuous	1.70	0.031	0.025	2.30	0.059	0.045
Discrete	1.69	0.037	0.030	2.30	0.064	0.052
Exponential Tail Continuous	1.69	0.024	0.022	2.25	0.043	0.057
Discrete	1.68	0.025	0.025	2.25	0.048	0.058
Exponential Continuous	1.65	0.020	0.053	2.22	0.037	0.082
Discrete	1.64	0.022	0.057	2.24	0.045	0.061
Capped Continuous	1.70	0.023	0.019	2.30	0.041	0.032
Discrete	1.70	0.025	0.020	2.30	0.045	0.037

Table B.2: A comparison of the likelihood functions. For each parameterization, we synthesized 100 data sets consisting of 1000 samples each. In the untruncated scenarios, we test the untruncated likelihood function (Equation B.3) on the unmodified data; in the naïve scenarios, we test the untruncated likelihood function excluding values falling above the 0.95 quantile ($x_{\max} = \infty$); in the truncated scenarios, we test the truncated likelihood function (Equation B.11) excluding values falling above the 0.95 quantile ($x_{\max} = P^{-1}(0.95)$); in the exponential tail scenarios, we test the full censored likelihood function (Equation B.17) replacing values falling above the 0.95 quantile with data from an exponential distribution shifted to $\sup(x_{\max})$ ($x_{\max} = P^{-1}(0.95)$); in the exponential scenarios, we test the full censored likelihood function replacing 5% of the data with data drawn from an exponential distribution shifted to 1; and, in the capped scenario, we test the capped likelihood function (Equation B.19) with the data effectively capped at $\sup(P^{-1}(0.95))$ ($x_{\max} = P^{-1}(0.95)$).

overestimated the true value of α. This result shows the importance of directly addressing deviations in the tail.

We then applied Aban et al.'s likelihood function to the right truncated data (Equation B.11). This did a good job of recovering the true value of α. The standard deviation of the estimated value of α, however, is about 50% larger than in the untruncated case.

We then censored the data. To censor the data, we replaced the data falling above the 0.95 quantile with random values from an exponential distribution.

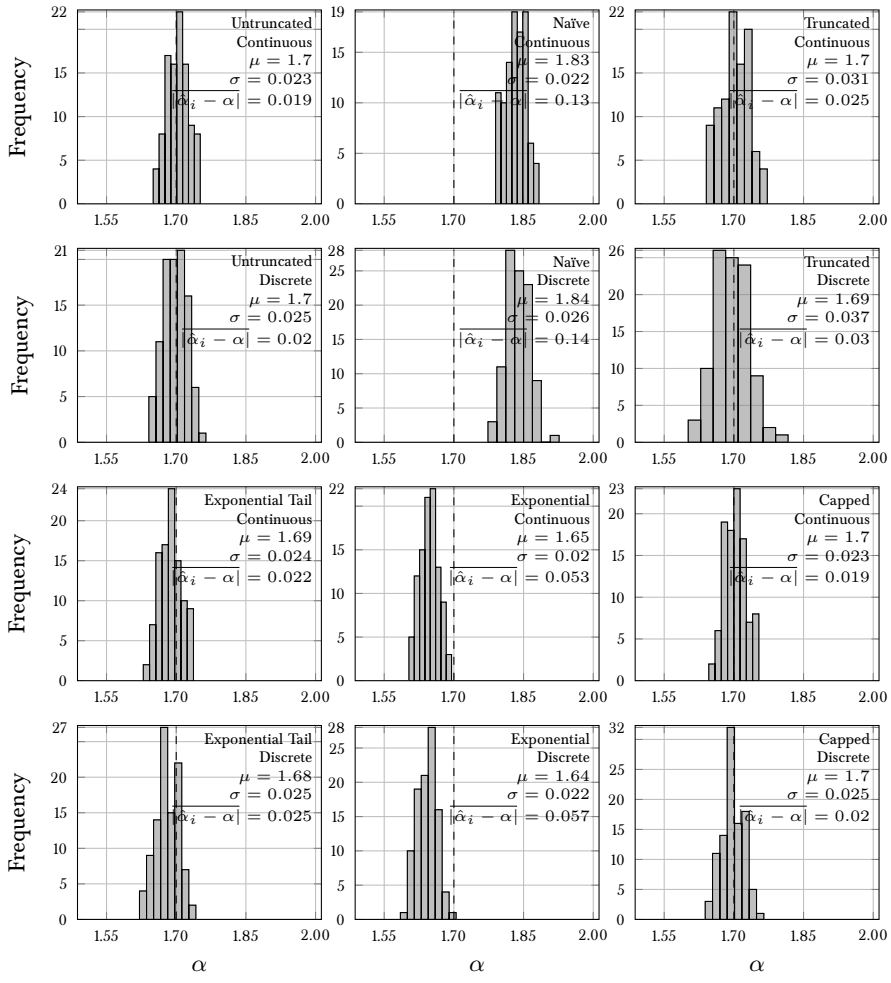


Figure B.7: Plots of the comparison of the performance of the likelihood functions at estimating α for both continuous data (odd rows) and discrete data (even rows) for $\alpha = 1.7$. See Table B.2 for a description of the scenarios.

We scaled the samples from the exponential distribution such that the 0.99 quantile is $5 \cdot x_{\max}$ and we then shifted them to $\sup(x_{\max})$. Examples of these data sets are shown in Figure B.8. Although the exponential distribution is also a heavy tailed distribution, it falls to 0 much faster than a power law does. Thus, a value drawn from such an exponential distribution is stochastically smaller than a value drawn from a power law. Note that we couldn't have simply moved the values above $x_{\max}$ towards $x_{\max}$: the censored data needs to be drawn from

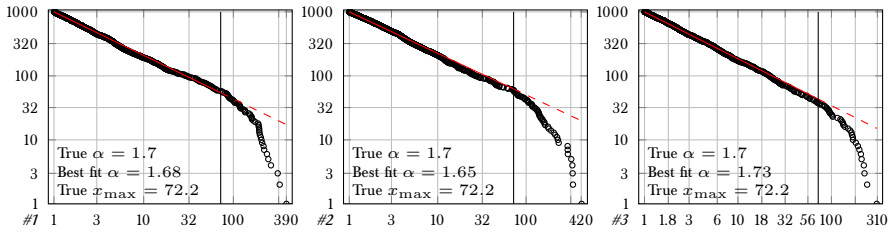


Figure B.8: Example of the synthesized power law data sets with an exponential tail. To synthesize a data set, we drew an appropriately sized sample from an untruncated power law and then replaced the data above $x_{\max}$ with data drawn from a scaled and shifted exponential distribution. The fits were done using the full right censored likelihood function.

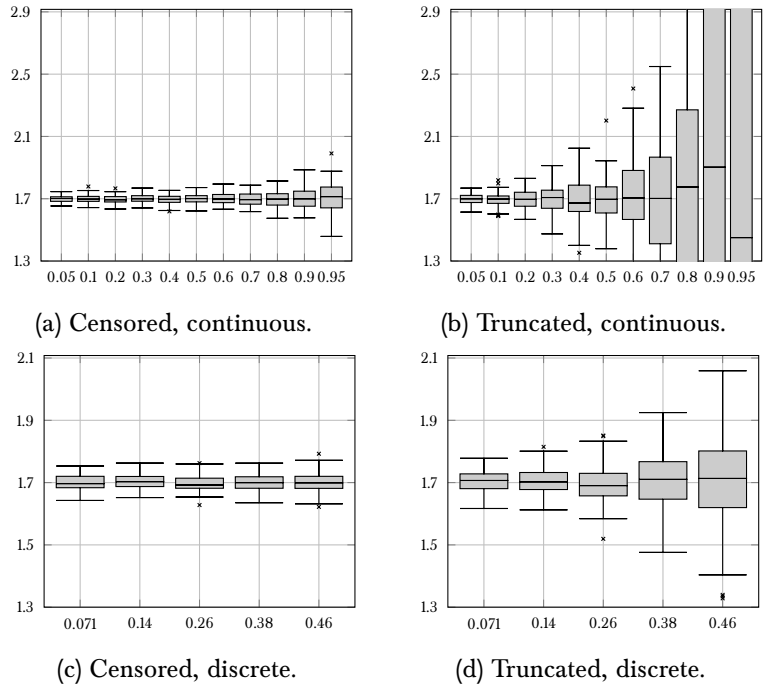
a different distribution; Equation B.17 assumes that the distribution of the true values (T) and that of the censored values (C) are independent. Surprisingly, the full likelihood function (Equation B.17) consistently underestimated α on this data and performed worse than the truncated likelihood function even though it had more information.

To determine if the underestimation arose from just censoring the tail, we randomly selected 5% of the data (independent of their value) and replaced them them with data drawn from a standard exponential distribution scaled such that the 0.99 quantile is $x_{\max}$ and shifted to 1. Surprisingly, the results were even worse. Van der Vaart's warning that incorporating censored data does not always lead to improvements in the estimate of the parameters appears justified.

In the last scenario, we capped the data at $x_{\max}$ and used the capped likelihood function for right censored data (Equation B.19). That is, we effectively set all values that exceed $x_{\max}$ to $\sup(x_{\max})$. This did an excellent job of recovering the true value of α. In fact, it did nearly as good a job of recovering the true value of α as the untruncated approach!

In conclusion, it seems that Aban et al.'s approach to estimating α using a right truncated power law does a very good job of approximating the true value of α. For undetermined reasons, using the right censored approach with the full likelihood function for right censored data performs poorly. However, if we use the capped likelihood function, which effectively just considers the number of points that exceed $x_{\max}$ and otherwise ignores the censoring point, we are able to recover α nearly as well as when using the untruncated likelihood function to estimate α on the untruncated data.

Although the capped likelihood function performs best in this test, this doesn't mean that the capped likelihood function should be strictly preferred



(a) Censored, continuous.

(b) Truncated, continuous.

(c) Censored, discrete.

(d) Truncated, discrete.

Figure B.9: The performance degradation of the capped likelihood function for censored data (Equation B.19) and of the truncated likelihood function (Equation B.11) as the portion of censored or truncated data, respectively, increases. The x values correspond to the portion of censored or truncated data. In the discrete case, there are fewer values, because $P(x = 1)$ is so large. It appears that the likelihood function does an excellent job of recovering α even if approximately the top 80% of data is censored. The truncated censored function degrades significantly faster—after about the top quarter of the data is truncated it no longer provides a good estimate.

when the data exhibits an upper bound. The capped likelihood function assumes that the amount of data that exceeds the threshold is consistent with a power law; the truncated likelihood function does not make this assumption and thus can handle more types of deviations in the tail. That is, if the tail exhibits a deviation, but the data in that region does not appear to be dampened, then the truncated approach should be used.

We now briefly consider how well the truncated and capped likelihood functions perform on different amounts of truncated or censored data. Figure B.9

shows the performance degradation as the amount of censored or truncated data increases. We again considered 100 continuous and 100 discrete data sets consisting of 1000 data points drawn from a power law distribution with $\alpha = 1.7$. As expected, as the portion of truncated or censored data increases, the estimate becomes worse. However, whereas the estimate for the truncated likelihood function becomes unusable after approximately 20% of the data have been censored, the censored likelihood function remains robust even when 80% of the data have been censored. The increased performance likely has to do with the additional information that the censored likelihood function exploits, namely, the number of points that exceed the upper bound. In practice, however, it seems unlikely that more than a few percent of the data will be dampened.

B.4.6 Impact on Estimating $x_{\min}$

We now evaluate how the different likelihood functions impact the estimation of $x_{\min}$ using the KS statistic as described by Clauset et al. [6, Section 3.3]. Since we haven't modified how $x_{\min}$ is selected and it doesn't directly use the likelihood functions, we expect the performance to remain unchanged. Note, however, that since the truncated and censored approaches use less information than the untruncated approach, they will probably perform a bit worse.

We evaluated the impact of the likelihood functions for both continuous and discrete data. We produced 100 continuous data sets and 100 discrete data sets consisting of 1000 data points drawn from a power law with $\alpha = 2$. We truncated or censored the data falling above the distribution's 0.9 quantile, i.e., approximately the top 10% of the data. Figure B.10 shows histograms of the estimated $x_{\min}$ for each likelihood function and for both the continuous data sets (top row) and the discrete data sets (bottom row).

As seen from the plots in the first column of Figure B.10, when using the untruncated likelihood function, the KS statistic does an excellent job of recovering the true value of $x_{\min}$. This is also the case when using the truncated likelihood function on the continuous data. The remaining plots exhibit a bowl shape: although the KS statistic sometimes selects a small value of $x_{\min}$, it often selects a value of $x_{\min}$ that approaches $x_{\max}$. This turns out to be a consequence of the small amount of data that is used in these cases: for values of $x_{\min}$ that approach $x_{\max}$, α overfits the data.

To understand why this overfitting occurs, consider how the KS statistic is computed for censored data. In particular, consider a value of $x_{\min}$ such that we have just 3 uncensored data points and assume that there are 97 data points that exceed the censoring threshold. In this example, the uncensored data points account for just 3% of the data. It's not difficult to imagine a value of α that does

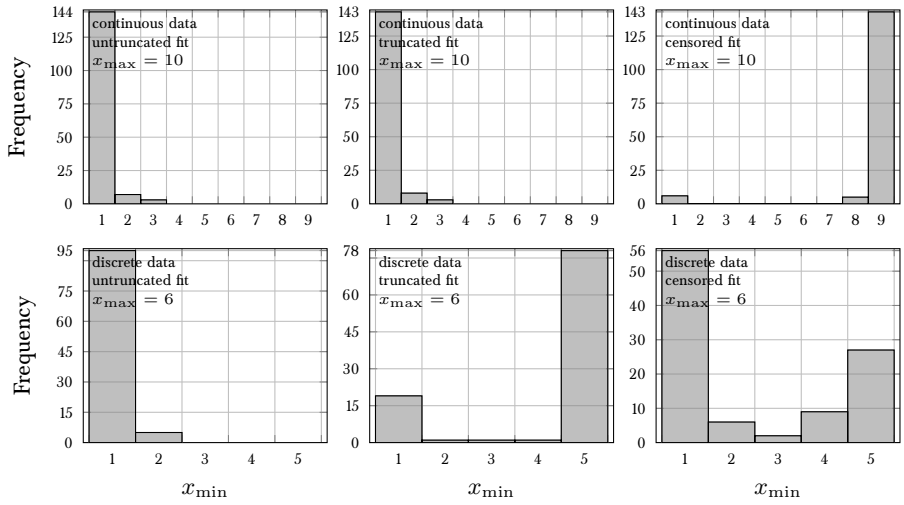


Figure B.10: The value of $x_{\min}$ chosen by the KS statistic when the untruncated, truncated and capped censored likelihood functions are used to estimate α on both continuous and discrete data. We performed the fits on 100 samples consisting of 1000 data points drawn from an untruncated power law with $\alpha = 2$. As appropriate, we truncated or censored the data that exceeded the 0.9 quantile. With the exception of the truncated approach on continuous data, when truncated or censoring the data, the KS test often chooses an $x_{\min}$ that is close to $x_{\max}$. In these cases, α overfits the available data.

a fair job of matching these 3 data points and perfectly matching the amount of censored data. Since the censored data accounts for 97% of the probability mass, the KS statistic will necessarily be smaller than 0.03. Further, since the value of α does a reasonable job of matching the uncensored data, it will be even smaller. As we will see in Table B.3, this is rather small relative to a sample with 100 data points.

Overfitting the discrete right truncated data happens for similar reasons. Again, consider how the KS statistic is computed as $x_{\min}$ approaches $x_{\max}$. Concretely, consider $x_{\min} = 5$ and $x_{\max} = 6$. In this case, the empirical and theoretical distributions consist of just two values—$P(X \leq 5)$ and $P(X \leq 6)$. The second value is necessarily 1: $P(X \leq x_{\max}) = 1$. Thus, there is really just a single value to fit. In this case, overfitting is easy. Note: although there may be multiple observations whose value is 5, these appear as a single data point in the empirical CDF.

The reason that the continuous right truncated data is not overfit has to do

with a bit of luck. For $\alpha = 2$, $x_{\max} = P^{-1}(0.9) = 10$. By default, our algorithm only considers values of $x_{\min}$ that are whole numbers less than the maximum observed value. This will be 9. $1000 \cdot P(9 \leq X \leq 10) = 11.1$. Thus, we will typically observe 11 data points that exceed $x_{\max}$. Since the data is continuous, these won't be equal to each other and the CDFs will consist of about 11 values. This is enough to generally inhibit overfitting. Testing other values of α and other amount of censoring demonstrates (not shown), however, that overfitting is a problem in general.

Note that if there is enough data that overfitting is not a problem then, in general, smaller values of $x_{\min}$ will be preferred, because more data reduces the sampling error as per the law of large numbers.

B.4.7 Avoiding Overfitting

To avoid overfitting, we propose adding a penalty to the KS statistic.

A commonly used penalty when estimating parameters is the Bayesian information criterion (BIC). The BIC is based on the amount of data that is used and the number of parameters. In our case, the number of parameters is constant, but the amount of data used varies according to the values of $x_{\min}$ and $x_{\max}$.

Unfortunately, a penalty based on the amount of data that is used does not solve the overfitting problem. Consider the case that $x_{\min} = 1$ and $x_{\max} = 2$. For data drawn from a power law with a typical value of α, this region accounts for a significant portion of the probability mass and thus a significant portion of the data. To be concrete, let $\alpha = 2$. $P(1 \leq X \leq 2) = 0.50$ in the continuous case and $P(1 \leq X \leq 2) = 0.61$ in the discrete case. The probability mass in this region increases as α increases. Compare this with $x_{\min} = 10$ and $x_{\max} = 100$, which spans an order of magnitude. For $\alpha = 2$, this region accounts for just 0.09 of the probability mass in the continuous case and 0.06 in the discrete case. If we were to apply a penalty based on the portion or amount of data that is used, we would penalize the latter case significantly more than the former case, which is the exact opposite of what we want.

This analysis suggests an alternate approach: penalize fits with narrow boundaries. This makes sense: as Newman notes, a power law is characterized less by its typical value than that it "var[ies] over an enormous dynamic range, sometimes many orders of magnitude" [55]. Thus, we should penalize values of $x_{\min}$ and $x_{\max}$ that only capture a small range of values.

Given Newman's characterization of a power law, it seems reasonable that the penalty should start to take effect when the boundaries span less than approximately 1.5 orders of magnitude. The penalty should initially be modest and increase slowly as the boundaries become increasingly tighter. Once the span of

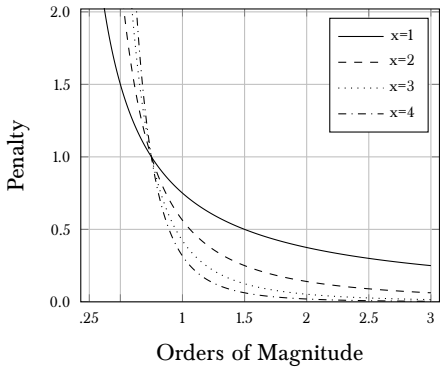


Figure B.11: Plot of the penalty function $\text{penalty}(\text{oom}) = \left(\text{oom}/0.75\right)^{-x}$ for different values of x. (oom stands for *orders of magnitude*.)

the boundaries drops below a critical point, the penalty should rapidly increase to avoid overfitting. This should be set at about 0.75 orders of magnitude based again on the characterization that power laws normally span multiple orders of magnitude.

The following family of functions meets these requirements:

$$\text{penalty}(\text{oom}; c, x, s) = s\left(\frac{\text{oom}}{c}\right)^{-x} \qquad c, x, s > 0$$

$$\text{oom} = \log_{10}\left(\frac{\max(\mathbf{data})}{\min(\mathbf{data})}\right) \qquad x_{\min} \leq \mathbf{data} \leq x_{\max}$$

oom is the orders of magnitude spanned by the data between $x_{\min}$ and $x_{\max}$. c, x and s correspond to the changeover point, the exponent, and the penalty's scale, respectively.

The proposed family of functions hugs the axes in the upper right quadrant. This provides the required minimal penalty when the boundaries are wide. Starting from $x = \infty$, the penalty gently increases as the boundaries become tighter until $\text{oom} = c$ at which point the penalty increases rapidly as oom decreases. Since we want the penalty to rapidly increase when the data spans less than 0.75 orders of magnitude, we set $c = 0.75$.

Figure B.11 shows plots of penalty functions that show the effect of varying the exponent. The x axis is the orders of magnitude spanned by the data and the y axis is the corresponding penalty. We fixed the scale (s) at 1. As expected, all of the curves hug the axes. The larger the exponent, the more this is the case. Values of x that are at least 2 appear to provide the desired minimal penalty for

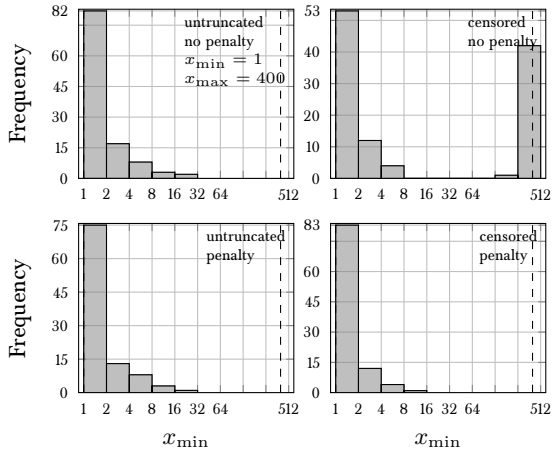
	$\alpha = 1.7$		$\alpha = 2.3$		$\alpha = 2.9$	
N	μ	σ	μ	σ	μ	σ
32	0.12	0.032	0.13	0.037	0.13	0.03
100	0.073	0.022	0.071	0.016	0.072	0.021
320	0.042	0.012	0.041	0.011	0.039	0.0093
1000	0.023	0.0066	0.023	0.0067	0.023	0.0065
3200	0.012	0.0032	0.013	0.0035	0.013	0.0036
10 000	0.0077	0.0021	0.0073	0.0021	0.0074	0.0018

Table B.3: Typical values of KS statistic for different amounts of randomly drawn data (N) from power law distributions with different values of α. For each α and N, we averaged the results of fits to 100 synthesized data sets with $x_{\min} = 1$ and $x_{\max} = \infty$. D appears to be inversely proportional to the number of samples and independent of α.

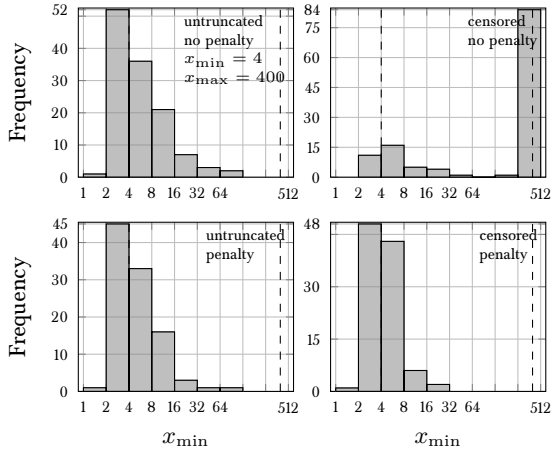
oom > 1.5. For values of x larger than about 3, it appears that the penalty no longer gently increases, but suddenly transitions from no penalty to a rapidly increasing penalty. This effectively creates a hard threshold on the minimum span. Based on these observation, we set $x = 2$. There is certainly some room for adjustment here, however, it is unclear whether this will make a significant difference.

The remaining issue to consider is the size of the penalty, s. Recall from Equation B.5 that D is the absolute difference between two CDFs. As such, its range is 0 to 1. Table B.3 shows the typical value of the KS statistic for fits to synthetic data sets drawn from an untruncated power law distribution with $x_{\min}$ fixed to 1 and $x_{\max}$ fixed to ∞. D appears to be inversely proportional to the number of samples and independent of α. This makes sense: again, according to the law of large numbers, the more data we have the better the fit. Given this observation, we propose to scale the penalty by 0.03, a value that is comparable to the typical value of the KS statistic for modest sample sizes. Thus, if the data really follows a power law, this will effectively prune boundaries that fall below c.

Example values of the penalty function using the selected parameters are shown in Table B.4.

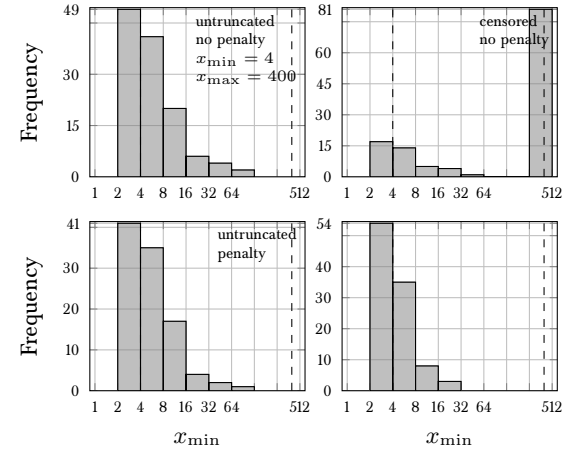


(a) No noise.

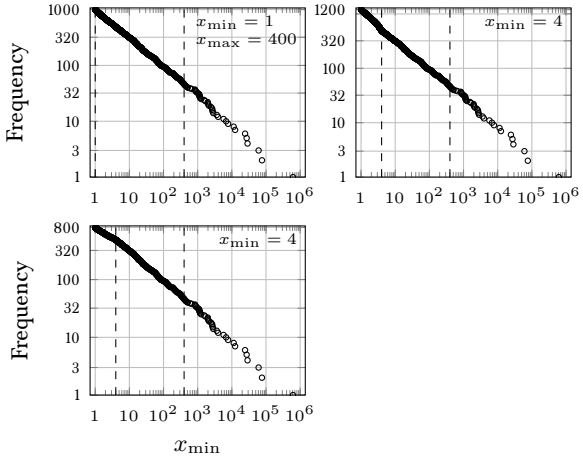


(b) Added Noise.

Figure B.12: The effect of the penalty on choosing $x_{\min}$. Each group of plots shows how $x_{\min}$ is chosen in the presence of a different type of noise. In the first group, no noise is added to the sample; in the second group, 200 uniformly distributed data points are added to the sample below the median (4); and, in the third group, 40% of the data below the median are removed. Each group shows the performance for the untruncated and the censored approaches with and without use of the penalty. The last group shows an example data set from each group. The vertical dashed lines show the true $x_{\min}$ and the censor threshold.



(c) Subtracted signal.



(d) Complementary cumulative Pareto plot of example data sets.

Figure B.12 (Continued): The effect of the penalty on choosing $x_{\min}$. Each group of plots shows how $x_{\min}$ is chosen in the presence of a different type of noise. In the first group, no noise is added to the sample; in the second group, 200 uniformly distributed data points are added to the sample below the median (4); and, in the third group, 40% of the data below the median are removed. Each group shows the performance for the untruncated and the censored approaches with and without use of the penalty. The last group shows an example data set from each group. The vertical dashed lines show the true $x_{\min}$ and the censor threshold.

Orders of Magnitude	Penalty
0.5	0.068
0.75	0.030
1	0.017
1.5	0.0075
2	0.0042
3	0.0019

Table B.4: Example values of the penalty function for $c = 0.75$, $x = 2$, and $s = 0.03$.

B.4.8 Effectiveness of the Penalty

We now consider the effectiveness of the penalty on synthetic data sets. Figure B.12 shows the performance of the untruncated and censored approaches in the absence of noise and in the presence of two different types of noise. For each scenario, we started with 100 synthetic data sets consisting of 1000 data points drawn from a continuous power law distribution with $\alpha = 1.5$. To create the censored data, we censored the data falling in the top 5% of the distribution, i.e., those values exceeding $P^{-1}(0.95)$.

We set $\alpha = 1.5$ to ensure a fair amount of spread in the data despite the relatively large amount of censoring. This allows us to better observe the influence of the penalty. If $x_{\max}$ were too small, then the penalty would be large and we would automatically choose the most extreme boundaries. For instance, for $\alpha = 1.7$, $P^{-1}(0.95) = 72$ and for $\alpha = 2.3$, this is just 10.

For the first group of plots, we didn't add any noise; in the second group of plots, we added 200 uniformly distributed data points through the median of the theoretical distribution ($P^{-1}(0.5) = 4$) resulting in 1200 total data points; and, in the third group of plots, we removed 40% of the data through the median leaving approximately 800 data points. If the noise is significant, we expect the algorithm to select $x_{\min} = 4$ in the latter two cases. The last group of plots shows an example data set modified for each scenario.

The left column of each group corresponds to the untruncated approach with (bottom) and without (top) the use of the penalty. We see that the penalty doesn't have a noticeable effect on the selection of $x_{\min}$. This is expected since the untruncated approach rarely chooses a very large value of $x_{\min}$ and the penalty, by design, only comes into effect for such values.

Each group's right column shows the censored approach with (bottom) and

		Untruncated				Censored			
Noise	Penalty	$\overline{\alpha}$	σ	$\overline{x_{\min}}$	σ	$\overline{\alpha}$	σ	$\overline{x_{\min}}$	σ
None	no	1.50	0.023	2.38	3.4	1.59	0.31	135	170
None	yes	1.50	0.023	2.22	2.9	1.50	0.022	1.59	1.2
Added	no	1.51	0.035	8.10	10	1.80	0.75	251	160
Added	yes	1.51	0.032	6.64	7.9	1.50	0.029	5.20	3.3
Subtracted	no	1.51	0.035	8.15	11	1.80	0.75	241	170
Subtracted	yes	1.51	0.033	7.23	8.8	1.50	0.029	5.23	3.8

Table B.5: Summary statistics of the effect of the penalty on choosing $x_{\min}$ in the presence of noise for the experiments presented in Figure B.12.

without (top) the use of the penalty. Consistent with Figure B.10, when we don't have a penalty, the censored approach often chooses a value of $x_{\min}$ that approaches $x_{\max}$. With the penalty, however, the censored approach performs similarly to untruncated case.

Table B.5 shows summary statistics of the experiments presented in Figure B.12. We again receive confirmation that the untruncated approach consistently does an excellent job of recovering α and a good job of recovering $x_{\min}$ independent of the noise and whether we use a penalty.

The censored approach does a good job of recovering the true parameters when we use the penalty. As expected, if we don't use the penalty, we overfit the data and we get large values of $x_{\min}$. When we use the penalty, however, the censored approach does as well as the untruncated approach at recovering α and $x_{\min}$. This is quite impressive given that the censored approach has less information to work with.

These results appear promising: using the penalty with the censored approach eliminates overfitting. Unfortunately, we do not have proof that the penalty is optimal. In particular, we only looked at two scenarios involving noise. The scenarios correspond to the two main types of deviations on the left: the presence of an additional process and the suppression of the power law below some threshold. The city data was an example of the latter: there are fewer cities with less than $50\,000$ people than predicted due, we theorized, to the benefits of incorporation not outweighing the overhead for small communities. To prove the general validity of the penalty function, we need to consider the effect of the penalty for different amounts of data and different amounts and types of noise. We leave this as future work.

B.4.9 Evaluation of the Estimation of $x_{\max}$

We've described how to extend Clauset et al.'s fitting algorithm to estimate $x_{\max}$ in Section B.4.3. We now evaluate its effectiveness.

To test the performance of the KS statistic on estimating $x_{\max}$, we used the same procedure to generate the data as when testing the full likelihood function for censored data: we synthesized 100 data sets consisting of 1000 samples from both continuous and discrete power laws with $\alpha = 1.7$. We replaced the values falling above the 0.95 quantile with random values from an exponential distribution. We scaled the samples from the exponential distribution such that the 0.99 quantile is $f \cdot x_{\max}$ and we then shifted them to $\sup(x_{\max})$. f stands for *factor*. We considered values of $f = 5$, $f = 3$ and $f = 1$. A smaller value of f corresponds to a steeper tail and more aggressive truncation. Examples data sets for $f = 5$ were shown in Figure B.8.

Table B.6 shows summary statistics (the relevant data are the rows for which "Bias" is "Yes") and Figure B.13 shows histograms of the recovered $x_{\max}$ in each situation. The data reveals that the KS statistic appears to overestimate the value of $x_{\max}$. This is particularly true for larger values of f. Looking at the plots in Figure B.8, which show example data sets for $f = 5$, we see that the beginning of the exponential tail appears to continue the straight line implied by the power law data until $x \approx 120$ and only then begins to deviate downward. Thus, we shouldn't be surprised that the test indicates that $x_{\max}$ is typically over 100 in this case; the data is still consistent with a power law. Indeed, this demonstrates that the test is effective and we do a reasonable job of recovering $x_{\max}$. As f becomes smaller, the deviation becomes more abrupt and the test does a better job of detecting the actual transition point.

Another reason for the greater variance in the estimation of $x_{\max}$ than in the estimation of $x_{\min}$ is the small amount of data in the tail relative to the amount of data on the left. The result is that increasing the upper boundary a bit typically adds just a few data points, which has little impact on the CDF. Decreasing $x_{\min}$ a bit, however, can sometimes double the amount of data, which can have a large impact. For instance, $P(X \geq 2; \alpha = 2) = 0.5$ for a continuous power law. Thus $x_{\min} = 2$ considers just half as much data as $x_{\min} = 1$.

Clauset et al. note that this could be an issue in the context of estimating $x_{\min}$ and propose the use of an unbiased version of the KS statistic [6, Equation 3.11]:

$$D^* = \sup \left(\frac{|F(x) - G(x)|}{\sqrt{F(x)\ (1 - F(x))}} \right) \tag{B.21}$$

where $F(\cdot)$ corresponds to the theoretical distribution. They observe that the results are very similar to the original, biased estimator and don't consider the

			$\hat{\alpha}$				$\widehat{x_{\max}}$		
	f	Biased	μ	σ	$\overline{\lvert\hat{\alpha}-\alpha\rvert}$	$x_{\max}$	$\mu_{1/2}$	MAD	$\overline{\lvert\widehat{x_m}-x_m\rvert}_{1/2}$
Cont.	5	Yes	1.71	0.023	0.021	72.2	96.5	36	29
Cont.	5	No	1.71	0.024	0.021	72.2	122	70	55
Discr.	5	Yes	1.70	0.025	0.020	43.0	54.5	14	14
Discr.	5	No	1.70	0.025	0.020	43.0	28.5	26	23
Cont.	3	Yes	1.70	0.027	0.021	72.2	79.7	21	15
Cont.	3	No	1.70	0.024	0.020	72.2	83.1	35	22
Discr.	3	Yes	1.70	0.024	0.018	43.0	43.5	5.2	4.0
Discr.	3	No	1.70	0.025	0.019	43.0	32.0	21	14
Cont.	1	Yes	1.70	0.025	0.021	72.2	72.4	8.7	6.1
Cont.	1	No	1.71	0.028	0.024	72.2	73.7	8.6	7.1
Discr.	1	Yes	1.70	0.026	0.020	43.0	44.0	3.0	2.0
Discr.	1	No	1.70	0.027	0.022	43.0	18.0	13	25

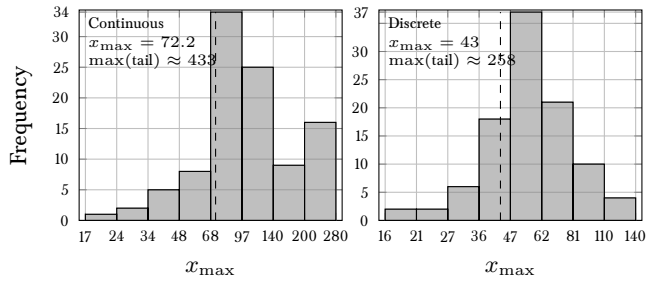
Table B.6: Summary statistics of estimating $x_{\max}$ using the KS statistic for a power law distribution with an exponential tail. We synthesized 100 data set from a power law with $\alpha = 1.7$ and replaced the values that exceeded the 0.95 quantile with data drawn from an exponential distribution. We fixed $x_{\min} = 1$. The width of the exponential tail is $f \cdot x_{\max}$, where f means *factor*. Note: the $1/2$ subscript means the median. The untruncated likelihood function estimates $\overline{\hat{\alpha}} = \{1.73, 1.72, 1.73\}$ in the continuous case, and equal $\overline{\hat{\alpha}} = \{1.72, 1.73, 1.74\}$ in the discrete case, for $f = \{5, 3, 1\}$, respectively. We used both a biased estimator (the original KS test) and an unbiased variant to find the best $x_{\max}$.

unbiased estimator further.

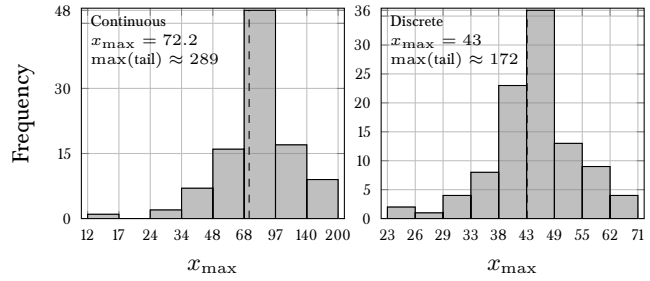
We used this unbiased version of the KS statistic to estimate $x_{\max}$ on the same data sets. The results are also shown in Table B.6. We see that the estimate of $x_{\max}$ significantly underestimates the true value. Further, this estimator tends to have significantly more variance.

Correctly identifying where the deviation becomes significant helps when estimating α. In all six scenarios, the estimated value of α is very close to the true value of α. For comparison, we also estimated the value of α using the untruncated likelihood function. This overestimated the value of α. As expected, this estimation became worse as the tail became increasingly compressed and the deviation increased.

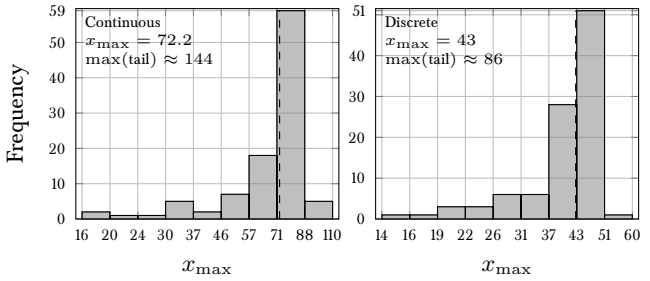
One minor problem remains. Initially, we considered all values that appeared



(a) Exponential tail's span: $5 \cdot x_{\max}$



(b) Exponential tail's span: $3 \cdot x_{\max}$



(c) Exponential tail's span: $1 \cdot x_{\max}$

Figure B.13: Plots of the performance of using the KS statistic to estimate $x_{\max}$ for a power law distribution with different exponential tails. The dashed line corresponds to the beginning of the exponential tail, i.e., the expected $x_{\max}$.

in the data as $x_{\max}$ candidates. This occasionally resulted in choosing a value of $x_{\max}$ near $\max(\mathbf{data})$. Although this resulted in a fit with a smaller value of D, it wasn't appropriate: when we set $x_{\max}$ to a value less than ∞, we assert that the tail has a different distribution. If the alleged tail only includes a handful of data points, then we are probably overfitting the data rather than identifying the start of a different process.

We used a simple mitigation strategy to prevent overfitting the tail: when searching for $x_{\max}$, don't consider the values of the 10 largest observations. This ensures that the tail contains at least 10 values, which is arguably below the lower limit for confidently identifying a transition to a new distribution. A better approach would be to use a gradual penalty as we did to prevent choosing $x_{\min} \approx x_{\max}$ in Section B.4.7, however, in practice, 10 observations appears to be sufficient to ensure that the KS test compares the distance between the distributions and not the noise due to statistical fluctuations.

B.4.10 Pruning the Search Space

When estimating α, we can sometimes use a closed form expression. When this is not possible, we can still use numeric optimization to quickly bracket the solution within a tight interval in $O(log(n))$ time where $n = {}^{\text{upper}-\text{lower}}/_{\text{tolerance}}$ and upper and lower are the initial bounds on the search space. (For details, see a text on optimization, such as, Press et al. [42, Chapter 10].) Unfortunately, neither of these techniques are applicable to the estimation of $x_{\min}$ and $x_{\max}$: the KS statistic is not smooth as we vary $x_{\min}$ and $x_{\max}$. This means that we need to do an exhaustive search in order to find the best estimates. Unfortunately, if we estimate both $x_{\min}$ and $x_{\max}$, the search space is two-dimensional and is impractical to exhaustively search for moderate sample sizes. To deal with this, we need to prune the search space. To determine how much pruning is necessary, we first examine the conditions under which an exhaustive search really is intractable.

Table B.7 shows an evaluation of our pruning strategy. Of interest at this point is the approximate number of $x_{\min}/x_{\max}$ combinations for different parameterizations and sample sizes. The empirical data confirms that the search grows quadratically. In the continuous case, since we don't see duplicate observations, we need to test ${}^{n^2}/_2$ combinations. The ${}^1/_2$ comes into play since we can immediately eliminate any $x_{\min}/x_{\max}$ combinations for which $x_{\min} \geq x_{\max}$. Thus, for $n = 1000$, an exhaustive search would need to check approximately half a million combinations. For $n = 3200$, the search space consists of over 4 million combinations. Unfortunately, $n = 3200$ is not a terribly large sample. In the discrete case, the situation is not so bad, since most of the observations take one of a few values. For instance, $P(X = 1; \alpha = 2) = 0.61$ and, for data drawn from such a distribution, we expect nearly two thirds of the observations to have the value 1. Nevertheless, the search space still grows quadratically with n and, as we will see shortly, thousands of combinations is the most that we can typically afford to check, which practically limits an exhaustive search in the discrete case to $n \lesssim 5000$.

					$x_{\min}/x_{\max}$		
		$\lvert D - \hat{D}\rvert$			Total		Tested %
α	n	$\mu_{1/2}$	MAD	> 0.001	$\mu_{1/2}$	MAD	$\mu_{1/2}$
1.7	100	$<10^{-15}$	$<10^{-15}$	40%	4100	0	18%
1.7	320	$<10^{-15}$	$<10^{-15}$	33%	47 900	458	5%
1.7	1000	$<10^{-15}$	$<10^{-15}$	30%	481 000	2910	0.9%
1.7	3200	$<10^{-15}$	$<10^{-15}$	7%	4 760 000	11 400	0.3%
1.7	10 000	$<10^{-15}$	$<10^{-15}$	0%	41 000 000	168 000	0.1%
2.3	100	$<10^{-15}$	$<10^{-15}$	13%	4100	0	17%
2.3	320	$<10^{-15}$	$<10^{-15}$	3%	47 600	457	5%
2.3	1000	$<10^{-15}$	$<10^{-16}$	0%	470 000	4310	0.9%
2.3	3200	$<10^{-15}$	$<10^{-15}$	0%	4 410 000	48 400	0.3%
2.3	10 000	$<10^{-15}$	$<10^{-15}$	0%	33 200 000	313 000	0.1%
2.9	100	$<10^{-15}$	$<10^{-15}$	3%	4100	0	11%
2.9	320	$<10^{-15}$	$<10^{-15}$	10%	47 300	457	5%
2.9	1000	$<10^{-15}$	$<10^{-16}$	3%	455 000	4960	0.9%
2.9	3200	$<10^{-15}$	$<10^{-15}$	3%	4 040 000	48 500	0.3%
2.9	10 000	$<10^{-15}$	$<10^{-15}$	0%	26 700 000	254 000	0.1%

Table B.7: Evaluation of the $x_{\min}/x_{\max}$ pruning strategy for continuous power laws. For each set of parameters and sample size, we compared the value of the KS statistic for the best fit as determined by Clauset et al.'s implementation and our implementation on 30 synthetic data sets. The table shows the median deviation between the best fits' KS statistics as well as the median number of $x_{\min}/x_{\max}$ combinations and the median portion of combinations that were actually tested by our implementation.

To get a feeling of the actual cost of estimating $x_{\min}$ and $x_{\max}$, consider again the city population data from Figure B.1b. This data contains 19 515 data points with 7973 unique values (the data is discrete). Using an Intel Xeon E5520 (released in 2009) clocked at 2.27 GHz, it took approximately 24 milliseconds to find the best estimate of α and compute the KS statistic for each candidate $x_{\min}/x_{\max}$, on average. In total, it took 190 seconds to find the best estimate of $x_{\min}$ when fixing $x_{\max}$ to ∞. If we had also searched for $x_{\max}$, we would have had to consider 32 million combinations. This would have taken approximately 200 hours to find the best fit! This is sometimes acceptable, however, to do a goodness of fit test, as we will discuss in the next section, we compare the data's best fit to the best fits of 1000 synthesized data sets. Estimating $x_{\min}$ and $x_{\max}$

					$x_{\min}/x_{\max}$		
		$\lvert D - \hat{D}\rvert$			Total		Tested %
α	n	$\mu_{1/2}$	MAD	> 0.001	$\mu_{1/2}$	MAD	$\mu_{1/2}$
1.7	100	0.0010	0.0008	50%	74	41	39%
1.7	320	0.0016	0.0018	67%	474	87	31%
1.7	1000	0.0011	0.0010	53%	2560	609	11%
1.7	3200	0.0007	0.0006	27%	11 900	1520	3%
1.7	10 000	0.0004	0.0005	23%	46 500	3350	0.8%
2.3	100	0.0005	0.0005	27%	12	5	65%
2.3	320	0.0005	0.0003	7%	43	13	48%
2.3	1000	0.0003	0.0004	0%	162	52	36%
2.3	3200	0.0004	0.0004	10%	570	122	27%
2.3	10 000	0.0004	0.0003	13%	2090	426	10%
2.9	100	0.0001	0.0001	3%	5	1	80%
2.9	320	0.0002	0.0002	0%	11	3	64%
2.9	1000	0.0002	0.0002	0%	31	6	54%
2.9	3200	0.0002	0.0002	0%	74	17	45%
2.9	10 000	0.0002	0.0002	0%	219	46	36%

Table B.7 (Continued): Evaluation of the $x_{\min}/x_{\max}$ pruning strategy for discrete power laws. For each set of parameters and sample size, we compared the value of the KS statistic for the best fit as determined by Clauset et al.'s implementation and our implementation on 30 synthetic data sets. The table shows the median deviation between the best fits' KS statistics as well as the median number of $x_{\min}/x_{\max}$ combinations and the median portion of combinations that were actually tested by our implementation.

for these would take over 20 CPU *years*. This is effectively intractable for all but the most important problems.

Hope, however, is not lost: although the KS statistic is not smooth, it will be approximately smooth in the region around the best estimates of the boundaries. In particular, we expect the value of the KS statistic to initially slowly increase to the right of the best estimate of $x_{\min}$ and to the left of the best estimate of $x_{\max}$. The reason for this is simple. Consider the region $x_{\min} + \delta_1$ through $x_{\max} - \delta_2$, for $\delta \ll x_{\max} - x_{\min}$. This region is part of the region spanned by $x_{\min}$ thorough $x_{\max}$, our best fit. If the data between $x_{\min} + \delta_1$ through $x_{\max} - \delta_2$ were inconsistent with a power law, it is unlikely that expanding this region a little bit would result in a significantly better fit: any sub-region of a

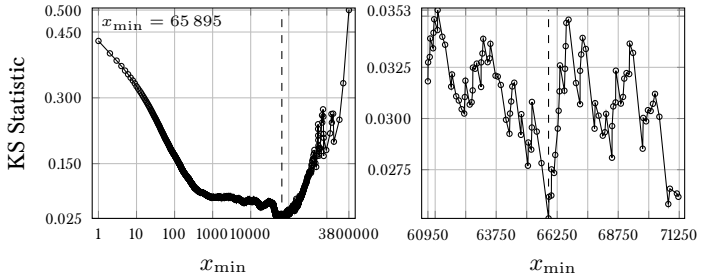


Figure B.14: Plot of the candidate values of $x_{\min}$ for $x_{\max} = \infty$ vs. the KS statistic for the city population data. The right plot is a zoomed in view around the best estimate of $x_{\min}$.

power law is also a power law with the same scaling parameter. Thus, as per the law of large numbers, the smaller region's KS statistic should only be a bit larger due to the slight increase in influence of the statistical fluctuations arising from the sample's smaller size. Figure B.14 shows that this is the case for the city population data. This local smoothness enables us to sample the search space and be confident that we find a fit that is similar to the best fit.

Clauset et al. tacitly acknowledge that exhaustively searching for $x_{\min}$ can be a problem. In their implementation, they provide an option to sample the set of candidate $x_{\min}$s. In particular, they sort the data, remove duplicates and index the remaining data as follows:

$$\text{indices} = \left\{ \text{round}\left(i \cdot \frac{|0\text{unique}(\mathbf{data})|0}{\text{samples} - 1} \right) : 0 \leq i < \text{samples}, i \in \mathbb{Z} \right\}$$

where samples is the number of candidate $x_{\min}$ to consider. This is a nice nonparametric approach. By having each sample cover approximately the same amount of data, each sample covers approximately the same amount of probability mass. In other words, this approach automatically adjusts to the underlying distribution and allocates more samples to higher density regions and fewer samples to lower density regions. This is particularly important for heavy tailed distributions. By comparison, consider a linear binning pruning strategy in which we round the observations to obtain a set of candidates. For a power law with $\alpha = 2$, approximately half of the data is between 1 and 2: $P(1 < X < 2) = 0.5$! Not considering any values between 1 and 2 would be too coarse: if there is just a bit of noise on the very left, we may end up throwing away half of the data.

Clauset et al.'s pruning strategy is reasonable when most of the data is unique. If it is not, then the samples will not cover the same amount of probability mass. This is particularly problematic when sampling discrete power laws:

most of the observations will be drawn from a few distinct, yet adjacent values. Indeed, most of the values will be $x_{\min}$. Consider a discrete power law with $\alpha = 2$. If we draw $100\,000$ samples, we will typically observe about 436 unique values. Of these, we expect about 61% to be 1 and about 15% to be 2. If we were to sample every fifth value (i.e., draw about a 100 samples), then we would consider 1, 6, 11, etc. Unfortunately, the first sample would cover nearly 90% of the observations! If the real best fit were $x_{\min} = 2$, and we chose $x_{\min} = 6$ instead, our best fit would span just a quarter of the data of the actual best fit. Throwing away this data increases the effects of the statistical fluctuations. This suggests taking every x^{th} observation. Unfortunately, this approach doesn't work either: continuing with the previous example, most of our samples would be 1!

Based on these observations, we propose the following algorithm to sample the data. Assume we want s samples and we have n observations. Cap the number of observations for any value at n/s. Thus, if we have 100 observations with value 1 and $n/s = 10$, throw away 90 of the observations whose value is 1. Repeat this using the resulting data set until all values have at most n/s observations. Note: each iteration except for the last decreases the number of effective observations (i.e., n), which causes n/s to increase and is why we need to iterate. Using the final data set, distribute the n/s samples evenly across the sorted observations. Since each observation occurs at most n/s times, no value will be chosen multiple times. But, because we allow multiple observations with the same value, adjacent values may be chosen.

The next problem is to consider how to chose the sample size. Clauset et al. don't provide any guidance on this. We observe that if we can't afford to exhaustively search the search space, then we clearly don't want to just reduce the computation time by a factor of, say, 2. We need to seriously reduce the search space to make the computation feasible.

Based on this, we propose considering $O(n)$ combinations by setting the number of samples to ceiling$(\sqrt{\max(n, 1000)})$, where n is the number of observations. This results in approximately $\sqrt{n}^2 = n\ x_{\min}/x_{\max}$ combinations.

In practice, choosing $s \approx \sqrt{n}$ samples for each of $x_{\min}$ and $x_{\max}$ appears to do a reasonable job of estimating the best fit. Nevertheless, we can do even better without a significant cost. Using the best fit as an anchor, we zoom in on its end points and repeat the process. We zoom in by again sampling $x_{\min}$ and $x_{\max}$ s times each, but this time instead of considering all of the data, we only consider the observations in the area around the best fit. In particular, we sample from the $n' = 0.5^i \cdot n$ observations around each end of the best fit where i is the number of times that we've zoomed so far. Because we expect the best fit to underestimate the true best fit, we emphasize the data away from the best fit by

considering the $2/3 \cdot n'$ observations away from best fit and the $1/3 \cdot n'$ towards the center of the best fit. We repeat this process until the zoomed-in region doesn't include any unexamined $x_{\min}/x_{\max}$ combinations. This increases the number of evaluated $x_{\min}/x_{\max}$ combinations by a factor $\log_2(n)$ to $O(n \log(n))$.

Recall from Section B.4.4 that when we allow both $x_{\min}$ and $x_{\max}$ to vary, there may be multiple non-overlapping portions of the data that appear to follow a power law. To deal with this, after identifying the best fit, we exclude the portion of the search space that overlaps with the fit and repeat the procedure. If we only consider fits that span some minimum orders of magnitude, the data set can be fairly quickly partitioned.

The question now is how many orders of magnitude a fit should span to be considered. As already noted in the context of the penalty function, power laws are characterized by their large dynamic range. Based on this observation, we designed the penalty function to highly penalize $x_{\min}/x_{\max}$ combinations that span less than 0.75 orders of magnitude. We now turn this into a hard threshold. We eliminate any $x_{\min}/x_{\max}$ combinations that span less than 0.75 orders of magnitude or a quarter of the number of orders of magnitude spanned by the data, whichever is less. The latter condition ensures that even for very compact data, we still consider some $x_{\min}/x_{\max}$ combinations. Applying this pruning strategy doesn't mean that the penalty now has no use: 0.75 orders of magnitude is a conservative lower bound on the span of an authentic power law; and, the penalty remains moderate until the span exceeds approximately 1.5 orders of magnitude.

We now evaluate the pruning strategy that we've developed. There are two aspects that we need to consider. First, we want to know whether the results with pruning are similar to the results obtained when no pruning is used. Second, we want to determine whether the computation cost is acceptable.

Table B.7 shows a comparison of our implementation using the pruning strategy with Clauset et al.'s implementation (with the correction to the computation of the KS statistic, as mentioned in Section B.4.2). For each set of parameters and sample size, we compared the value of the KS statistic on 30 synthetic data sets. The table shows the median difference between the values of D computed by each implementation. The column labeled "> 0.001" is the portion of test data sets on which the difference between the KS statistics was greater than 0.001. The table also shows the total number of $x_{\min}/x_{\max}$ combinations and the portion actually considered by our implementation.

Looking at the table, we see that the differences between the best fits' KS statistics is very small relative to the best fit's typical value (recall Table B.3, which shows the typical value of D for different parameters and sample sizes). In fact, in many of the cases, the estimated values of $x_{\min}$ are identical. For

α, the main source of difference is numerical error in the continuous case; in the discrete case, we use a different search technique to estimate α resulting in a typical difference in the estimated value of α of 0.005 from Clauset et al.'s implementation. In conclusion, the best fit identified using our pruning strategy is very close to the true best fit.

In terms of the number of $x_{\min}/x_{\max}$ pairs that we actually consider, we see that it is a small fraction of the total space. Clearly, the pruning strategy has saved a lot of work.

B.4.11 Evaluation on Real Data

We now compare the use of the right censored likelihood function and the use of KS test to estimate $x_{\min}$ and $x_{\max}$ on real data. In particular, we look at the length of alternating sequences of two towers in cell tower traces. This data is presented in Figure 4.16. See that figure and its context for details. We fit both an untruncated power law ($x_{\max} = \infty$) and a dampened power law using the capped likelihood function.

Looking at Figure B.15, we can divide the differences in the untruncated and dampened fits into three categories: completely different, similar and nearly identical.

In plots 1 and 11, the untruncated and dampened fits are completely different. In these cases, there is a significant tail that appears to follow a straight line. Whereas the untruncated power law fits the tail and sets $x_{\min}$ to the approximate end of the deviation, the truncated power law matches the flat area on the left and sets $x_{\max}$ to the start of the deviation. It is difficult to determine whether one fit is better than the other, whether they are both reasonable, or even whether either is reasonable and some other distribution should be considered.

For some plots, the fits are similar, but the truncated approach identifies a tail. This is the case in plots 4, 7, 8, 9, 13 and 15. The untruncated approach appears to do a reasonable job of identifying the start of the deviation, which appears authentic. In these cases, the truncated fit recovers a larger value of α than the untruncated fit. A larger value of α means a steeper curve, which is due to the tail pulling the curve down. In terms of fitting the data on the left, the truncated fit appears to do a better job. In these cases, the untruncated fit sometimes seems to fit neither parts of the curve very well, as is the case in plots 7 and 13.

In the remaining plots, the fits are nearly the same. In plots 2, 5, 6, 10 and 14, this seems completely reasonable. These plots do have a bit of noise in the tail, however, this is more likely due to statistical fluctuations or outliers than evidence of a new process. In the last two plots, plots 3 and 10, the data

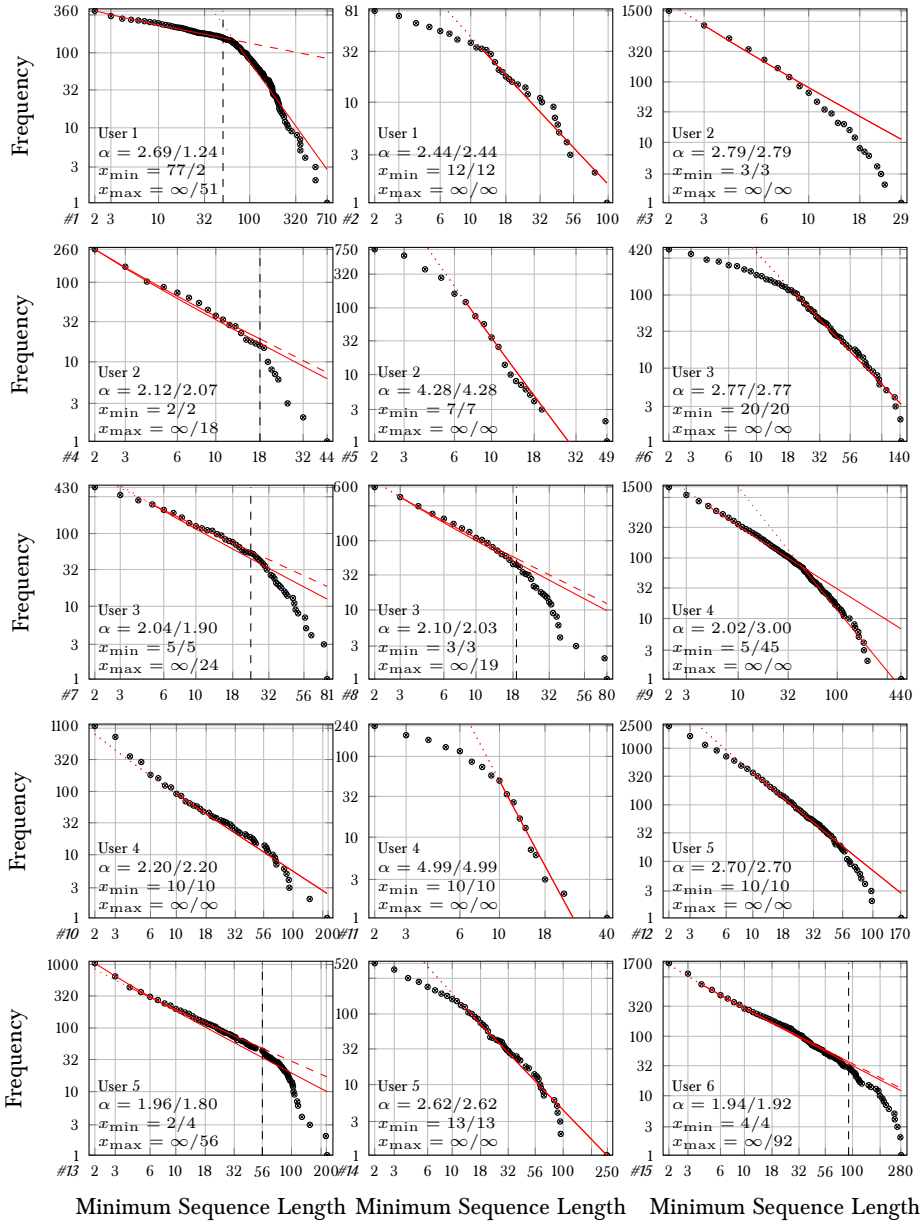


Figure B.15: Fitting α, $x_{\min}$ and $x_{\max}$ on real data. We show both an untruncated fit and a fit using the censored approach. In the latter case, we use the KS test to estimate $x_{\max}$. The vertical, dashed line depicts $x_{\max}$.

suggests a curved line. A power law seems to be a poor fit. Given the shape, the log-normal distribution might be better.

Based on this small sample of data, we saw that the dampened power law can do a better job of fitting data than an untruncated power law. In a certain sense, this is expected: it has an additional degree of freedom. Nevertheless, the added flexibility appears useful, at least for the data under consideration.

B.5 Goodness of Fit

Given a set of parameters, the next step is to determine how well they model the data, the model's goodness of fit.

B.5.1 Background

The idea behind a goodness of fit test is relatively straightforward. Unfortunately, we can rarely be certain that data was *not* generated by a particular process. For instance, a normal distribution has a non-zero value for all values of x although most of the probability mass is concentrated near the mean. Thus, all values are possible even if most are unlikely. As such, a sample drawn from a normal distribution could appear to be consistent with a power law. What we can do is determine the probability that data was generated by a theorized distribution: $P(\textbf{data} \mid \theta)$. The theorized distribution is the so-called *null hypothesis*. If the probability that the data came from the distribution is sufficiently small, we reject the null hypothesis and declare the model to not be a good fit. Otherwise, we retain the null hypothesis as a possible explanation.

The probability that a model would generate a sample, the so-called p *value*, is the probability that the model would generate data as extreme as the observed data. A point is typically considered more extreme if its absolute distance from the mean is larger than the observation's absolute distance (d). That is, $p = \int_{x,|x-\mu|\geq d} P(x)\,dx$.

A consequence of this approach is that since we reject the null hypothesis for observations that have a non-zero probability of actually occurring if the model were true, we will occasionally reject the null hypothesis based on data that really was generated by the model. Thus, the threshold we adopt (the p value below which we reject the null hypothesis) represents the false positive rate. In practice, a p value of 0.05 is often used. This p value means that we are willing to incorrectly reject the null hypothesis (the proposed model) 5% of the time. If we use a smaller p value, then the probability that we incorrectly reject the null hypothesis decreases, but the probability that we incorrectly retain the null hypothesis increases.

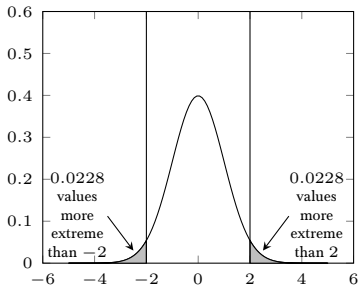


Figure B.16: A plot of $\mathcal{N}(0, 1)$ and a sample with the value of 2. The probability of observing a value that is more extreme than 2 away from the mean (0) is 0.0456. Using a p value of 0.05, we would reject the null hypothesis that this sample was drawn from $\mathcal{N}(0, 1)$ at the $p = 0.05$ level and fail to reject it at the $p = 0.01$ level.

When we retain the null hypothesis, we don't claim that the null hypothesis is correct, i.e., that the data actually arose from a process consistent with the model. Instead, we say that the model does a reasonable job of describing the data. For instance, if we want to know whether it rained and our test is whether the sidewalk is wet, then if the data says the sidewalk is wet, we can't conclude that it rained: it could just be that the neighbor's sprinkler was on.

Consider the following scenario. Say that we think a standard normal distribution (i.e., a normal distribution with a mean of 0 and a standard deviation of 1) describes a process and that we make an observation and find that its value is 2. To determine the goodness of fit, we compute the probability that samples that are actually drawn from the model are at least as extreme as the actual observation. In this case, 0.192 of samples drawn from a standard normal distribution will be at least as extreme as our sample. In other words, if we generate many samples from the standard normal distribution, approximately 0.192 of the them will have values that are further from the mean. This is illustrated in Figure B.16. Using a p value of 0.05, we would retain the null hypothesis.

B.5.2 Power Laws

Clauset et al.'s goodness of fit test compares how often samples synthesized from the data's best fit parameters are more extreme than the data itself [6, Section 4.1].

The procedure is as follows. Determine the best fit parameters (α and $x_{\min}$) for the data and compute the portion of the data (p) that is at least $x_{\min}$. Now, synthesize a data set with the same number of observations as the original data

set such that $u \sim \text{binom}(|0\text{data}|0, p)$ observations are at least $x_{\min}$ and the remaining are less than $x_{\min}$. For the data that is at least $x_{\min}$, draw u samples from the best fit. For the data less than $x_{\min}$, draw $n - u$ samples (with replacement) from the original data that is less than $x_{\min}$. Now, find the best fit parameters for the synthetic data set and compare the KS statistic of the original data and its best fit with the KS statistic of the synthetic data set and its best fit. Repeat. The portion of times that the original data's KS statistic is less than the synthesized data's KS statistic is the p-value. Thus, a *larger* p-value indicates a better fit, i.e., the data is consistent with a power law.

The accuracy of the p-value is:

$$\epsilon = \frac{1}{2\sqrt{n}} \tag{B.22}$$

where n is the number of trials. Thus, a thousand trials will yield a p-value that is accurate to within approximately 0.016 of the true value.

Clauset et al. recommend being conservative and ruling out a power law if $p \leq 0.10$. They note, however, that using a p-value of 0.05 is more common [6]. We follow this latter convention in this thesis.

B.5.3 Dampened Power Laws

We now extend Clauset et al.'s goodness of fit test to deal with dampened power laws. The main challenge is synthesizing the data above $x_{\max}$. To synthesize data below $x_{\min}$, Clauset et al. use a two step process: first, they determine the number of observations that should be below $x_{\min}$ and then they sample the original data. We use the same basic framework with minor modifications to better handle the (typically) sparse data in the tail. A second complication is ensuring that the model comparisons are between similar fits. As mentioned in Section B.4.4, since we allow both $x_{\min}$ and $x_{\max}$ to vary, there may be multiple, mostly non-overlapping fits. We want to ensure that the synthesized data's model is similar to the data's best fit. A final issue is determining how to compare the data's fit and the synthesized data's fits.

Synthesizing the Tail

To determine the number of observations below $x_{\min}$, Clauset et al. draw n samples from a binomial distribution where n is the number of observations and where the probability of success (p) is the portion of data below $x_{\min}$ in the original data set. We denote the number of successes by b (for *below*). This method is also appropriate for determining the amount of data that should appear in

the tail when using the right truncated model. When we assume that the data is dampened, however, the best fit explicitly models the probability that an observation will occur above $x_{\max}$: $P(X > x_{\max})$, where $P(\cdot)$ is the probability of the expression in an untruncated power law. Thus, we use this probability instead of the portion of the data in the tail for the binomial's p parameter. If the null hypothesis is true (i.e., the data is drawn from a dampened power law), these probabilities will be close. If, however, the null hypothesis is false, they can be significantly different. The choice of p in the latter case can significantly influence the test's result, since the data in the tail is not ignored, but helps determine the model's parameters. Having determined an appropriate value for p, we then draw $n - b$ samples from the binomial distribution. We denote the number of successes by a (for ***above***).

We now consider how to actually synthesize the data for each of the three parts of the synthesized data set. For the data below $x_{\min}$, we reuse Clauset et al.'s procedure: we draw b samples with replacement from the data that is less than $x_{\min}$. For the main body, we simply draw $n - a - b$ samples from a right truncated power law. This is appropriate for both the right truncated model and the dampened model: in both cases, we want exactly $n - a - b$ samples from a power law with definite lower and upper bounds. This leaves the data above $x_{\max}$. Unfortunately, sampling the data above $x_{\max}$ does not appear to be completely appropriate in this case: if the data in the tail is sparse, duplicated values are unlikely and unexpected. Instead, we estimate the tail's density and draw samples from that distribution. Note, however, that if the user supplies a single fixed value for $x_{\max}$, we are done: a is the number of points that are dampened, which is all that the fitting procedure needs; it doesn't look at the actual values of the data in this case. We only need to synthesize the data if there are multiple candidates for $x_{\max}$.

Sampling data is equivalent to sampling from a uniform distribution and transforming the samples using the inverse of the empirical CDF (the sample quantile). The empirical CDF is a step function that puts a probability mass of $1/n$ at each data point. Although the empirical CDF converges in probability to the distribution function (see, e.g., [4, Theorem 7.3]), sampling from it is really only appropriate when the number of required samples is significantly less than the amount of data used to construct the empirical CDF.

There are a number of commonly used variations of the sample quantile function that turn it into a continuous function. See, for instance, Hyndman and Fan [53]. These approximations are crude—they just evenly spread out the probability mass between two data points.

Instead, we use a kernel density estimator (KDE) to estimate the dampening process. A KDE places a kernel, K, at each data point. The kernel is typically

a Gaussian, however, any symmetric PDF is fine. Indeed, the selection of the kernel is not critical; the crucial parameter is the bandwidth, the kernels' spread (see, e.g., Wasserman [4, Page 422]). To estimate the density of some value x, we simply compute the average probability of x for each of the kernels:

$$f(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

where n is the number of observations and h is the bandwidth. See, for instance, Wasserman's books for more background [4, 5].

There are two basic ways to estimate the bandwidth: the plug-in approach and the cross validation approach. The main plug-in algorithm is the Sheather-Jones (SJ) algorithm [56]; the main cross validation approach is least squares cross validation (LSCV) [122]. There is general consensus that the SJ method results in more visually appealing fits and that LSCV appears to undersmooth. Loader demonstrates, however, that appearances are deceiving and that the plug-in methods are generally incapable of identifying smaller bandwidths when these are correct [101, Page 422]. Wasserman notes that a reason that the plug-in methods perform poorly is that they require estimating f'', the second derivative of the distribution, "which is harder than estimating f," because "we need to make stronger assumptions about f to estimate f'' [5, Section 6.3]. Based on these arguments and the fact that we would prefer to undersmooth rather than oversmooth, since we want to synthesize a sample that is similar to the original sample, we elect to use LSCV. In our implementation, we use Duong's implementation of LSCV, `hlscv` [29].

Wasserman notes that LSCV doesn't deal well with data that has been rounded [4, Page 317]. The problem is not so much that the data are rounded, but that rounding can result in multiple observations with the same value, which LSCV doesn't handle well. Repetitions are particularly a problem when dealing with discrete data. To work around this, Wasserman adds some random Gaussian noise to the data. We do the same. To avoid getting into a discussion about the best variance of the normal distribution, we use a standard normal distribution and repeat the process 10 times. We then set h to the median of the estimated bandwidths.

Venables and Ripley note that "[m]ost density estimators will not work well when the density is non-zero at an end of its support" [123]. This is clearly the case for the tail's distribution: the density drops to zero to the left of $x_{\max}$. To deal with this problem, they recommend reflecting the data about the end point and then ignoring the density estimate beyond the support (in our case, below the reflection point) and doubling the estimated density within the support (in

```
1  library(ks)
2
3  boundary = xmax + if (discrete) 0.5 else 0
4
5  tail.data = data[data > boundary]
6  # Reflect the data.
7  tail.data = c(boundary − (tail.data − boundary), tail.data)
8
9  duplicated.mask = duplicated(tail.data)
10 duplicated.count = sum(duplicated.mask)
11 h = median(sapply(1:(if (duplicated.count > 0) 10 else 1),
12   function (.) {
13     tail.data[duplicated.mask] = tail.data[duplicated.mask] + rnorm(duplicated.count)
14     return (hlscv(tail.data))
15   }))
16
17 rtail <− function(n) {
18   # Draw a random sample from the tail's density estimate.
19   q = rnorm(n, mean=sample(tail.data, n, replace=TRUE), sd=h)
20
21   # Reflect the data below the boundary.
22   reflect = q < boundary
23   q[reflect] = boundary + (boundary − q[reflect])
24   if (discrete)
25     q = round(q)
26   return (q)
27 }
```

Listing B.1: Estimating and sampling from the tail

our case, above the reflection point) to normalize the resulting PDF. We take this approach. When dealing with discrete data, we reflect the data about $x_{\max} + 0.5$ rather than $x_{\max}$, since we make the data discrete by rounding and rounding a value $x \in [x_{\max}, x_{\max} + 0.5]$ rounds x down to $x_{\max}$, which is not in the tail.

Sampling from the estimated distribution is straightforward. Observe that each kernel contributes the same amount of probability mass to the density estimate. Further, generating a random number from a normal distribution is often a standard function. Thus, we just need to sample the kernels (i.e., the data) and then sample the kernel.

Listing B.1 shows R code to estimate the tail and draw samples from it.

Finding a Similar Fit

In Section B.4.4, we observed that when we allow both $x_{\min}$ and $x_{\max}$ to vary, there may be multiple, mostly non-overlapping fits. To choose the best fit, we proposed finding the best mostly non-overlapping fits according to their KS statistics, dropping any whose KS statistic was more than twice the best fit, and then taking the fit that spanned the most orders of magnitude.

When fitting the synthesized data, a different approach is more appropriate. We want to compare the data's best fit with a similar fit in the synthesized data set. This may be different from the best fit. To ensure that we compare a similar fit, we only consider $x_{\min}/x_{\max}$ pairs that overlap with the data's best fit. In particular, we only consider $x_{\min}/x_{\max}$ pairs whose span overlaps with at least a quarter of the span (in log space) of the data's best fit or at least 0.75 orders of magnitude, whichever is less. The upper threshold is primarily there to deal with the case that $x_{max} = \infty$ in which case the span is ∞. The threshold also helps when there is a single very large outlier included in the data's fit.

Comparing Fits

A final point that needs clarification is how to compute the p value. The p value is the portion of synthetic data sets' whose best fit is worse than the data's best fit. To compare two data sets and their respective best fits, Clauset et al. use the KS statistic: they compute the KS statistic of the data and its best fit and compare that to the KS statistic of the synthetic fit and its respective best fit. If the data's KS statistic is smaller, its best fit is considered better:

$$|0\text{KS}(\text{data}, \text{fit}(\text{data})) < \text{KS}(\text{data}_i, \text{fit}(\text{data}_i))|0, i \in \{1, 2, \ldots, n\}$$

In Section B.4.7, we developed a penalty that we used to avoid overfitting $x_{\min}/x_{\max}$, which is chosen based primarily on the value of the KS statistic. This raises the question of whether we should use this penalty here as well. It turns out this is not appropriate. The reason for this is that the KS statistics are often very close and the maximum value in the data set, which can vary greatly, plays the deciding factor, which is undesirable. Consider drawing 1000 samples from an untruncated power law with $\alpha = 2$. Repeating this a 1000 times resulted in maximum values ranging from 163.9 to $975\,242$—over 3 orders of magnitude!

B.5.4 Evaluation

To improve our confidence that our extensions to Clauset's methodology are valid, we first synthesized a number of data sets from different untruncated power laws and examined their p values as determined by our methodology

and Clauset et al.'s methodology. Although the models are different—Clauset et al.'s model is a special case of our three parameter model with $x_{\max} = \infty$—we generally expect to see similar p values for data drawn from a power law with $x_{\max} = \infty$. Nevertheless, our model has more capacity [4, 76], and large differences will arise when the synthesized data is inconsistent with an untruncated power law, but is consistent with a dampened power law. This happens when there is a deviation in the tail, which will occur due to normal statistical fluctuations. In these cases, the best fit for our model will have a small KS statistic (i.e., be a good fit), since our model accommodates the deviation, and the best fit for the two parameter model will have a large KS statistic (i.e., be a bad fit), since it does not.

For the comparison, we synthesized data from power laws with $x_{\min} = 1$ and $x_{\max} = \infty$. We considered different values of α, both the continuous and the discrete case and different sample sizes. For each parameterization, we synthesized 100 data sets and computed their p values using both our methodology and Clauset et al.'s methodology (using their implementation with our correction to the computation of the KS statistic; see Section B.4.2).

For each parameterization, we first compared the difference between the computed p values. This is summarized in Table B.8 and shown in more detail in Figure B.17.

Looking at the table, we see that the average difference is generally small, but that the variance is non-negligible. Since the goodness of fit test considers 1000 synthesized data sets, according to Equation B.22, we expect the computed p values to be within about $1/2\sqrt{1000} = 0.016$ of their true values. The standard deviation often exceeds this suggesting that there is another reason for the differences between the p values than just statistical fluctuation. As mentioned above, another source of variance comes from the use of different models: although our model is a generalization of the model used by Clauset et al., the models have different capacities and thus accommodate statistical fluctuations differently.

Figure B.17 shows the computed p values plotted against each other for the parameterizations with 1000 samples. There are some significant outliers. These tend to be in the upper left quadrant and mean that the data is unlikely to come from the 2 parameter model, but is consistent with the 3 parameter model. In most cases, however, the plots reveal that the p values are comparable.

We now consider the distribution of p values. We expect the p values to be distributed according to a uniform distribution when the data sets are drawn from the null hypothesis. See, e.g., Wasserman, for an explanation [4, Theorem 10.14]. For each parameterization, we compute the KS statistic of the p values and the uniform distribution. The table shows that for both models, the p values are generally consistent with a uniform distribution ($p > 0.05$). This

			$x_{\min}/x_{\max}$ − Clauset		KS(ECDF, uniform)	
α		n	μ	σ	$x_{\min}/x_{\max}$	Clauset
1.7	Continuous	320	0.014	0.22	0.058	0.0025
1.7	Continuous	1000	0.01	0.092	0.82	0.90
1.7	Discrete	320	0.0086	0.20	0.45	0.55
1.7	Discrete	1000	0.0035	0.14	0.58	0.24
2.3	Continuous	320	0.0069	0.12	0.76	0.56
2.3	Continuous	1000	0.027	0.12	0.14	0.012
2.3	Discrete	320	$<10^{-3}$	0.043	0.29	0.24
2.3	Discrete	1000	0.011	0.063	0.61	0.66
2.9	Continuous	320	0.011	0.074	0.17	0.14
2.9	Continuous	1000	0.0083	0.087	0.45	0.20
2.9	Discrete	320	0.0053	0.072	0.45	0.35
2.9	Discrete	1000	0.00094	0.057	0.88	0.42
3.5	Continuous	320	0.032	0.13	0.049	0.0050
3.5	Continuous	1000	0.007	0.082	$<10^{-4}$	$<10^{-3}$
3.5	Discrete	320	0.0019	0.023	0.48	0.56
3.5	Discrete	1000	$<10^{-3}$	0.030	0.86	0.98

Table B.8: Comparison of the p values computed using the goodness of fit tests. The first result shows the mean difference between the computed values and their standard deviation. On average, there is no difference, however, the variance is not negligible. The second result shows how consistent the distribution of the p values is with a uniform distribution. We computed this using the KS statistic. The distribution of the p values computed by each methodology for each scenario is generally consistent with a uniform distribution, which is what we expect for an unbiased goodness of fit test.

is visually confirmed by the plots in Figure B.18, which show the uniform CDF vs. the empirical CDF of the p values. When a line goes above the $y = x$ diagonal, the methodology appear pessimistic (we have more small p values than expected); when it falls below the line, the methodology appears optimistic (we have more large p values than expected). Generally, the empirical distributions follow the diagonal well. Where they diverge, they tend to diverge in the same manner.

We now briefly consider the performance of our methodology on dampened data. As in Section B.4.5, when synthesizing data, we replace the data exceeding the distribution's 0.95 quantile with data drawn from a shifted exponential. See Figure B.8 for some example data sets. Note: we again only consider $\alpha = 1.7$.

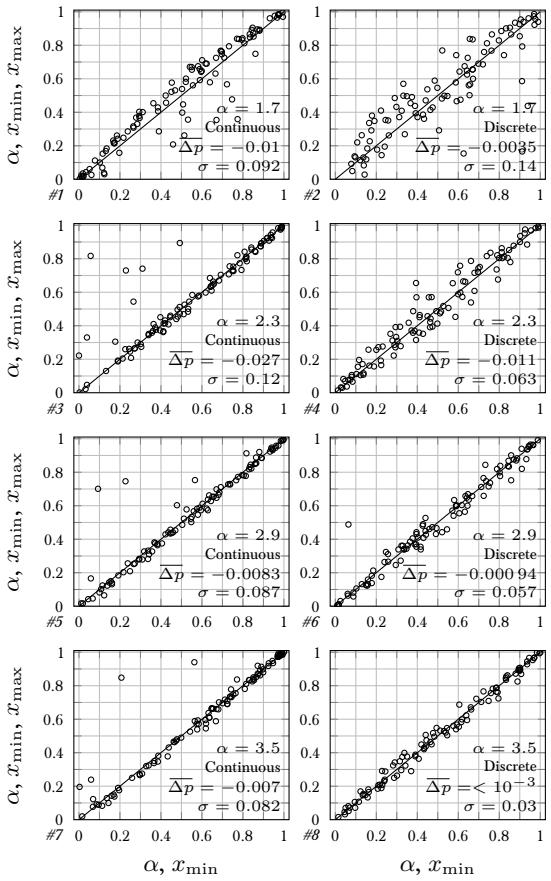


Figure B.17: Plots of the p values of the synthesized data sets with 1000 samples computed by Clauset et al.'s methodology vs. the p values computed by our methodology. When the data follows a straight line, the computed p values are comparable. Points above the straight line correspond to data sets to which our model assigns higher p values (i.e., the reported fits are better) than that computed by Clauset et al's model.

For $f = 5$, this tail distribution is not appropriate for values of $\alpha \gtrsim 2$: in that case, the generated values are typically greater than the original values, i.e., the generated data is not actually dampened.

Table B.9 and Figure B.19 shows the results. As before the p values appear to be distributed fairly consistently with a uniform distribution.

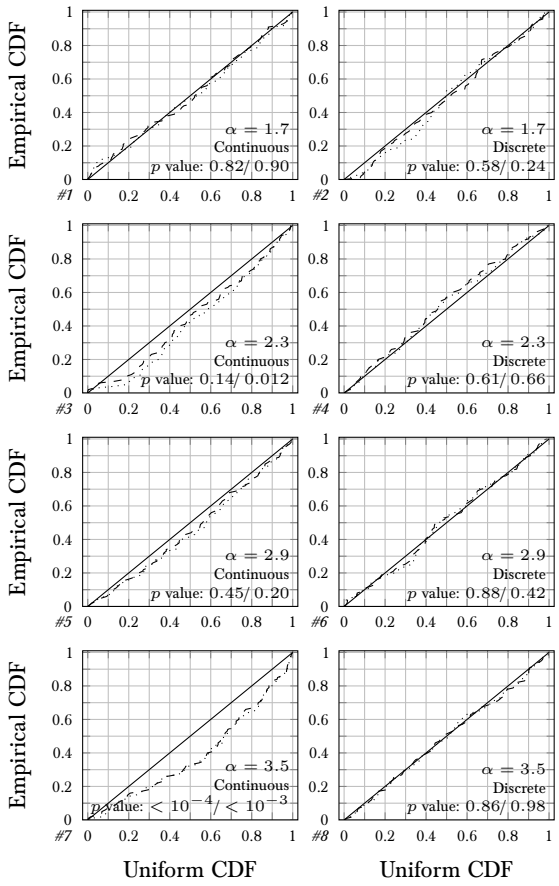


Figure B.18: Plots of the uniform CDF vs. the empirical CDFs of the estimated p values of the synthesized data sets with 1000 samples computed using our methodology (dashed line) and Clauset et al.'s methodology (dotted line). The solid line is the uniform CDF plotted against itself, i.e., $y = x$. Inset in each plot is the p value of the distribution of the p values vs. the uniform distribution using the KS statistic. When the ECDF falls above the $y = x$ line, the methodology appears pessimistic; when it falls below the line, the methodology appears optimistic.

α		n	KS(ECDF, uniform)
1.7	Continuous	320	0.015
1.7	Continuous	1000	0.058
1.7	Discrete	320	0.48
1.7	Discrete	1000	0.92

Table B.9: Performance of the goodness of fit test on dampened data. We used the KS statistic to compare the distribution of the computed p values with the uniform distribution. The p values appear to be generally consistent with a uniform distribution, which is what we expect for an unbiased goodness of fit test.

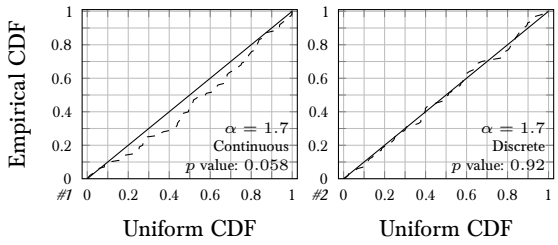


Figure B.19: Plots of the uniform CDF vs. the empirical CDFs of the estimated p values of the synthesized dampened data sets with 1000 samples. The solid line is the uniform CDF plotted against itself, i.e., $y = x$; the dashed line is the CDF of the p values. We used the KS statistic to compare the distribution of the p values with the uniform distribution. This p value is inset in the plot.

B.6 Conclusion

In this chapter, we reviewed heavy tailed distributions with a particular emphasis on power law distributions, which are a major theme in subsequent chapters. We observed that the current state of the art for fitting data to a power law doesn't handle data with a strong downward deviation in the tail. Based on the observation that this feature can arise from an external dampening process, we proposed treating such data as if it were right censored, i.e., treating the data above the threshold as a lower bound on the data's true, but suppressed, value. We then extended Clauset et al.'s methodology for fitting and performing goodness of fits on untruncated power laws to handle this case [6].

Extending Clauset et al.'s methodology to deal with dampened data required

identifying an appropriate likelihood function for estimating α, developing a method to estimate $x_{\max}$ and ensuring that the fitting methodology and goodness of fit test are sound. During this development, we encountered several complications. For instance, when we allow both $x_{\min}$ and $x_{\max}$ to vary, overfitting can occur. We solved this by penalizing fits with a narrow dynamic range. Also, when both $x_{\min}$ and $x_{\max}$ vary, the $x_{\min}/x_{\max}$ search space is $O(n^2)$, where n is the number of samples in the data set. This is too expensive to search exhaustively for moderate sample sizes. To deal with this we developed heuristics to prune the search space, which require checking just $O(n \log(n))$ $x_{\min}/x_{\max}$ combinations, which is reasonable. Our evaluation found that our heuristics did an excellent job of identifying the best fit. Another significant problem is how to synthesize data sets when we don't know the tail's distribution. We developed a nonparametric approach based on kernel density estimates.

Our evaluation showed that our methodology performs similarly to Clauset et al.'s on data that is drawn from an untruncated power law. We also showed that we are able to do a good job of estimating the parameters when there is a fair amount of noise. Future work is to improve the evaluation by considering real data sets.

Intended to be blank.

Appendix C

Simulation of Equalities used in the Right Censored Distribution Proof

In this appendix, we present the Monte Carlo simulation that we used to verify different equalities employed in the proof of:

$$P(X, \Delta) = \underbrace{[P(C = x)\, P(T > x)]}_{\text{censored}}^{1-\delta} \cdot \underbrace{[P(T = x)\, P(C \geq x)]}_{\text{not censored}}^{\delta}$$

We consider both the equalities that we identified as problematic in Patti et al.'s proof [97] as well as our corrections, which we presented in Section B.4. In particular, we examine:

$$\begin{aligned} P(X = x, \Delta = 1) &= P(T = x)\, P(x \leq C) \\ P(X = x, \Delta = 0) &= P(C = x)\, P(x < T) \\ P(X = x \mid \Delta = 1) &\neq P(T = x) \\ P(X = x \mid \Delta = 0) &\neq P(C = x) \end{aligned}$$

The simulation generates a million pairs of true and censored values based on discrete distributions of T and C. It then computes the empirical distribution of the relevant equalities and compares them to the theoretical results. If the results are within 1 percentage point of each other, we conclude, at least for the scenario in question, that the equalities and inequalities hold.

The output of a simple simulation is below. This is followed by the R code.

```
P(T) (true value):
```

```
  T=1   T=2   T=3
"1/2" "1/4" "1/4"

P(C) (censoring point):
  C=1   C=2   C=3
  "0" "1/3" "2/3"

p(T, C): empirical matches theoretical.
    T=1   T=2    T=3
C=1 "0"   "0"    "0"    "0"
C=2 "1/6" "1/12" "1/12" "1/3"
C=3 "1/3" "1/6"  "1/6"  "2/3"
    "1/2" "1/4"  "1/4"  "1"

P(X) = P((T=x, x <= C) | (C=x, x <= T)) (observed value):
 empirical matches theoretical.
  X=1   X=2   X=3
"1/2" "1/3" "1/6"

P(D) = {0: P(T > C), 1: P(C >= T)} (censor decision):
 empirical matches theoretical.
    D=0     D=1
 "1/12" "11/12"

P(C <= t): empirical matches theoretical.
  C=1   C=2   C=3
  "1"   "1" "2/3"

P(T <= t): empirical matches theoretical.
  T=1   T=2   T=3
"1/2" "3/4"   "1"

P(X=x, D=d) = [P(T=x) P(C >= x)]^(1-d) * [P(C=x) P(T > x)]^d:
 empirical matches theoretical.
    D=0    D=1
X=1 "0"    "1/2"   "1/2"
X=2 "1/12" "1/4"   "1/3"
X=3 "0"    "1/6"   "1/6"
    "1/12" "11/12" "1"
```

```
P(X | D) != P(C) if D=0 (Patti eqn 22), P(T) if D=1 (Patti eqn 26):
 empirical matches theoretical.
    D=0 D=1
X=1 "0" "6/11"
X=2 "1" "3/11"
X=3 "0" "2/11"
```

```
# A monte carlo simulation of the proof presented in section 5 of
# "Review of the Maximum Likelihood FUnctions for Right Censored Data"
# by Patti, Biganzoli and Boracchi.

library(plyr)

# sink(file="right-censored-proof-monte-carlo.txt")

if (TRUE) {
  p.T = c(1/2, 1/4, 1/4)
  p.C = c(0, 1/3, 2/3)
  T = C = 1:length(p.T)
} else {
  # Some random values.
  T = C = 1:10
  p.T = runif(length(T))
  p.T = p.T / sum(p.T)
  p.C = runif(length(T))
  p.C = p.C / sum(p.C)
}

stopifnot(all(T == C) && length(T) == length(p.T) && length(C) == length(p.C))
stopifnot(abs(sum(p.T) - 1) < 1e-4 && abs(sum(p.C) - 1) < 1e-4)

frac <- function (x) {
  # Print a number as a fraction.
  x2 = sapply(x, function (x) {
    if (x %in% c(0, 1))
      return (as.character(x))

    d = which.min(sapply(1:40, function (i) abs((x * i) - round(x * i))))
    if (d == 1)
      return (sprintf("%d", round(x)))
    return (sprintf("%d/%d", round(d * x), d))
  })

  if (is.matrix(x)) {
```

```
    dim(x2) = dim(x)
    rownames(x2) = rownames(x)
    colnames(x2) = colnames(x)
  }

  return (x2)
}

dump <- function(s=NULL, x, truth=NULL) {
  if (missing(x) && !is.character(s)) {
    x = s
    s = NULL
  }

  if (!is.null(s)) {
    cat(sprintf("\n%s:", s))
    if (nchar(s) > 30)
      cat("\n")
  }

  if (!is.null(truth)) {
    ok = all(abs(x - truth) < 1e-2)
    if (ok)
      cat(" empirical matches theoretical.")
    else {
      cat(" empirical DOES NOT match theoretical.")
      stop("Mismatch.")
    }
  }
  cat("\n")

  if (is.matrix(x)) {
    # Display the marginals if we have a joint distribution.
    given.cols = all(abs(colSums(x) - 1) < 1e-2)
    given.rows = all(abs(rowSums(x) - 1) < 1e-2)
    if (!(given.cols || given.rows)) {
      x = rbind(x, colSums(x))
      x = cbind(x, rowSums(x))
    }
  }
  x = frac(x)
  print(x)
}

names(T) = names(p.T) = sprintf("T=%d", T)
```

```
dump("P(T) (true value)", p.T)

names(C) = names(p.C) = sprintf("C=%d", C)
dump("P(C) (censoring point)", p.C)

samples = 1000000
Ts = sample(T, samples, replace=TRUE, prob=p.T)
Cs = sample(C, samples, replace=TRUE, prob=p.C)

# P(T, C)
p.T.C = outer(p.C, p.T, '*')
p.T.C.empirical = matrix(nrow=length(C), ncol=length(T), data=0)
result = ddply(expand.grid(C=1:length(C), T=1:length(T)), .(C, T),
  function (df) {
    data.frame(count=sum(Ts == T[df$T] & Cs == C[df$C]))
  })

p.T.C.empirical[cbind(result$C, result$T)] = result$count / length(Ts)

rownames(p.T.C.empirical) = names(C)
colnames(p.T.C.empirical) = names(T)
dump("p(T, C)", p.T.C.empirical, p.T.C)

# X = min(T, C)
X = T
Xs = pmin(Ts, Cs)

result = ddply(expand.grid(X=1:length(X), T=1:length(T), C=1:length(C)),
  .(X, T, C),
  function (df) {
    p = 0
    if ((T[df$T] == X[df$X] && X[df$X] <= C[df$C])
        || (C[df$C] == X[df$X] && X[df$X] <= T[df$T]))
      p = (p.T.C[df$C, df$T])
    return (data.frame(p=p))
  })
p.X = sapply(X, function (x) sum(result$p[X[result$X] == x]))

p.X.empirical = sapply(X, function (X) sum(Xs == X) / length(Xs))

names(X) = names(p.X) = names(p.X.empirical) = sprintf("X=%d", X)
dump("P(X) = P((T=x, x <= C) | (C=x, x <= T)) (observed value)",
      p.X, p.X.empirical)
```

```
# P(D) = 0 = censored = T > C
# 1 = not censored = C >= T
D = 0:1
Ds = ifelse(Ts > Cs, 0, 1)

result = expand.grid(T=1:length(T), C=1:length(C))
mask = T[result$T] > C[result$C]
p.D = c(sum(p.T.C[cbind(result$C[mask], result$T[mask])]),
   sum(p.T.C[cbind(result$C[!mask], result$T[!mask])]))

p.D.empirical = c(0, 0)
p.D.empirical[which(D == 0)] = sum(Ds == 0) / length(Ds)
p.D.empirical[which(D == 1)] = sum(Ds == 1) / length(Ds)

names(D) = names(p.D) = names(p.D.empirical) = sprintf("D=%d", D)
dump("P(D) = {0: P(T > C), 1: P(C >= T)} (censor decision)",
       p.D, p.D.empirical)


# P(C >= t)
ccdf.C = sapply(C, function (c) sum(p.C[C >= c]))
ccdf.C.empirical = sapply(C, function (c) sum(Cs >= c) / length(Cs))
dump("P(C >= t)", ccdf.C, ccdf.C.empirical)


# P(T <= t)
# (Note: in the discrete case, P(T <= t) != P(T < t)!)
cdf.T = sapply(T, function (t) sum(p.T[T <= t]))
cdf.T.empirical = sapply(T, function (t) sum(Ts <= t) / length(Ts))
dump("P(T <= t)", cdf.T, cdf.T.empirical)


# P(X, D)
# Recall:
# D = 0 = censored
# D = 1 = not censored
#
# P(X, D=0) = P(C=x) * P(T > x) (censored)
# = P(C=x) * (1 - P(T <= x))
# P(X, D=1) = P(T=x) * P(C >= x) (not censored)
p.X.D = matrix(nrow=length(T), ncol=length(D),
   data=c(p.C * (1 - cdf.T), p.T * ccdf.C))

result = ddply(expand.grid(X=1:length(X), D=1:length(D)), .(X, D),
   function (df) {
```

```
    data.frame(count=sum(Xs == X[df$X] & Ds == D[df$D]))
  })
p.X.D.empirical = matrix(nrow=length(X), ncol=length(D), data=0)
rownames(p.X.D.empirical) = names(X)
colnames(p.X.D.empirical) = names(D)
p.X.D.empirical[cbind(result$X, result$D)] = result$count / length(Xs)
p.X.D.empirical = p.X.D.empirical / sum(p.X.D.empirical)

rownames(p.X.D) = names(X)
colnames(p.X.D) = names(D)
dump("P(X=x, D=d) = [P(T=x) P(C >= x)]^(1-d) * [P(C=x) P(T > x)]^d",
     p.X.D, p.X.D.empirical)


# P(X | D) = P(X, D) / P(D) (and not P(C) if D = 0 and P(T) if D = 1)
p.X.given.D = t(t(p.X.D) / p.D)

p.X.given.D.empirical = matrix(nrow=length(X), ncol=length(D), data=0)
result = ddply(expand.grid(X=1:length(X), D=1:length(D)), .(X, D),
  function (df) {
    data.frame(counts=sum(Xs == X[df$X] & Ds == D[df$D]))
  })
p.X.given.D.empirical[cbind(result$X, result$D)] = result$counts
p.X.given.D.empirical =
  t(t(p.X.given.D.empirical) / colSums(p.X.given.D.empirical))

rownames(p.X.given.D) = names(X)
colnames(p.X.given.D) = names(D)
dump("P(X | D) != P(C) if D=0 (Patti eqn 22), P(T) if D=1 (Patti eqn 26)",
     p.X.given.D, p.X.given.D.empirical)

sink(NULL)
```

Listing C.1: Simulation of some equalities in the right censored proof

Intended to be blank.

Appendix D

Predictor Performance

This appendix lists the performance (correct prediction attempts by user, portion of prediction attempts, and correct prediction trials) of the predictors presented in chapter 6 and several more for different weight thresholds and aging parameters. The simpler predictors come first, and the more complex predictors come later.

The listed predictors are:

- $P(t)$: Table D.1
- $P(t \mid 30m)$: Table D.2
- $P(t \mid h)$: Table D.3
- $P(t \mid d)$: Table D.4
- $P(t \mid w)$, western weekend : Table D.5
- $P(t \mid w)$, local weekend : Table D.6
- $P(t \mid 30m, d)$: Table D.7
- $P(t \mid h, d)$: Table D.8
- $P(t \mid 30m, w)$, western weekend : Table D.9
- $P(t \mid h, w)$, western weekend : Table D.10
- $P(t \mid 30m, w)$, local weekend : Table D.11
- $P(t \mid h, w)$, local weekend : Table D.12

- $P(t \mid r)$: Table D.13
- $P(t \mid r, 30m)$: Table D.14
- $P(t \mid r, h)$: Table D.15
- $P(t \mid r, 30m, w)$, western weekend : Table D.16
- $P(t \mid r, h, w)$, western weekend : Table D.17
- $P(t \mid r, 30m, w)$, local weekend : Table D.18
- $P(t \mid r, h, w)$, local weekend : Table D.19
- $P(t \mid r, 30m, d)$: Table D.20
- $P(t \mid r, h, d)$: Table D.21
- $P(t \mid h, c)$: Table D.22
- $P(t \mid h, w, c)$, western weekend : Table D.24
- $P(t \mid h, w, c)$, local weekend : Table D.24
- $P(t \mid h, d, c)$: Table D.25
- $P(t \mid r, h), P(t \mid r)$: Table D.26
- $P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$: Table D.27
- $P(t \mid r, h, d), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$: Table D.28
- $P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$, western weekend : Table D.29
- $P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$: Table D.30
- $P(t \mid h, d, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$: Table D.31
- $P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$: Table D.32
- $P(t \mid h, d, c, \Delta), P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$: Table D.33

	Correct Attempts		Attempts		Correct Trials	
Weight	Median	MAD	μ	σ	Median	MAD
60 m	61%	30%	100%	0.0%	61%	30%
2 h	61%	30%	100%	0.0%	61%	30%
4 h	61%	30%	100%	0.0%	61%	30%
8 h	61%	30%	100%	0.0%	61%	30%
16 h	61%	30%	100%	0.0%	61%	30%

Table D.1: $P(t)$

		Correct Attempts		Attempts		Correct Trials	
↓Hour	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	71%	24%	100%	1%	71%	24%
∞	2 h	71%	24%	100%	2%	71%	24%
∞	4 h	71%	23%	92%	13%	66%	24%
∞	8 h	65%	33%	67%	36%	40%	39%
∞	16 h	41%	56%	48%	39%	29%	43%
21 d	60 m	72%	17%	100%	1%	72%	17%
21 d	2 h	72%	17%	100%	2%	72%	17%
21 d	4 h	72%	17%	92%	13%	68%	19%
21 d	8 h	69%	24%	66%	36%	51%	29%
21 d	16 h	0%	0%	0%	0%	0%	0%
14 d	60 m	72%	16%	100%	1%	72%	16%
14 d	2 h	72%	16%	100%	2%	72%	17%
14 d	4 h	72%	17%	92%	13%	68%	18%
14 d	8 h	0%	0%	0%	0%	0%	0%
14 d	16 h	0%	0%	0%	0%	0%	0%
7 d	60 m	73%	17%	100%	1%	73%	17%
7 d	2 h	73%	17%	100%	2%	73%	18%
7 d	4 h	0%	0%	0%	0%	0%	0%
7 d	8 h	0%	0%	0%	0%	0%	0%
7 d	16 h	0%	0%	0%	0%	0%	0%
4 d	60 m	71%	18%	100%	1%	71%	18%
4 d	2 h	74%	18%	78%	15%	58%	18%
4 d	4 h	0%	0%	0%	0%	0%	0%
4 d	8 h	0%	0%	0%	0%	0%	0%
4 d	16 h	0%	0%	0%	0%	0%	0%

Table D.2: $P(t \mid 30m)$

		Correct Attempts		Attempts		Correct Trials	
↓Hour	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	70%	24%	100%	0%	70%	24%
∞	2 h	70%	24%	100%	1%	70%	25%
∞	4 h	70%	25%	100%	2%	70%	25%
∞	8 h	71%	22%	92%	14%	66%	26%
∞	16 h	66%	32%	66%	36%	40%	39%
21 d	60 m	71%	17%	100%	0%	71%	17%
21 d	2 h	71%	17%	100%	1%	71%	17%
21 d	4 h	71%	17%	100%	2%	71%	17%
21 d	8 h	72%	17%	92%	14%	67%	20%
21 d	16 h	68%	24%	66%	36%	51%	28%
14 d	60 m	72%	16%	100%	0%	72%	16%
14 d	2 h	72%	16%	100%	1%	72%	16%
14 d	4 h	72%	16%	100%	2%	72%	16%
14 d	8 h	72%	17%	92%	14%	68%	18%
14 d	16 h	0%	0%	0%	0%	0%	0%
7 d	60 m	73%	17%	100%	0%	73%	17%
7 d	2 h	73%	17%	100%	1%	73%	17%
7 d	4 h	73%	17%	100%	2%	73%	17%
7 d	8 h	0%	0%	0%	0%	0%	0%
7 d	16 h	0%	0%	0%	0%	0%	0%
4 d	60 m	72%	17%	100%	0%	72%	17%
4 d	2 h	72%	17%	100%	1%	72%	17%
4 d	4 h	74%	16%	66%	19%	48%	22%
4 d	8 h	0%	0%	0%	0%	0%	0%
4 d	16 h	0%	0%	0%	0%	0%	0%

Table D.3: $P(t \mid h)$

	Correct Attempts		Attempts		Correct Trials	
Weight	Median	MAD	μ	σ	Median	MAD
60 m	57%	26%	100%	1%	57%	26%
2 h	57%	26%	100%	1%	57%	26%
4 h	57%	26%	100%	2%	57%	26%
8 h	57%	27%	99%	4%	57%	29%
16 h	57%	27%	98%	7%	57%	29%

Table D.4: $P(t \mid d)$

	Correct Attempts		Attempts		Correct Trials	
Weight	Median	MAD	μ	σ	Median	MAD
60 m	59%	30%	100%	0.0%	59%	30%
60 m	59%	31%	100%	0.0%	59%	31%
2 h	59%	30%	100%	0.0%	59%	30%
2 h	59%	31%	100%	0.0%	59%	31%
4 h	59%	30%	100%	0.0%	59%	30%
4 h	59%	31%	100%	0.0%	59%	31%
8 h	59%	30%	100%	0.0%	59%	30%
8 h	59%	31%	100%	0.0%	59%	31%
16 h	59%	30%	100%	0.0%	59%	30%
16 h	59%	31%	100%	0.0%	59%	31%

Table D.5: $P(t \mid w)$, western weekend

	Correct Attempts		Attempts		Correct Trials	
Weight	Median	MAD	μ	σ	Median	MAD
60 m	59%	30%	100%	0.0%	59%	30%
60 m	59%	31%	100%	0.0%	59%	31%
2 h	59%	30%	100%	0.0%	59%	30%
2 h	59%	31%	100%	0.0%	59%	31%
4 h	59%	30%	100%	0.0%	59%	30%
4 h	59%	31%	100%	0.0%	59%	31%
8 h	59%	30%	100%	0.0%	59%	30%
8 h	59%	31%	100%	0.0%	59%	31%
16 h	59%	30%	100%	0.0%	59%	30%
16 h	59%	31%	100%	0.0%	59%	31%

Table D.6: $P(t \mid w)$, local weekend

		Correct Attempts		Attempts		Correct Trials	
↓Hour	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	66%	30%	71%	32%	45%	34%
∞	2 h	45%	53%	51%	39%	31%	46%
∞	4 h	38%	56%	34%	35%	10%	16%
∞	8 h	0%	0%	17%	28%	0%	0%
∞	16 h	0%	0%	7%	16%	0%	0%
21 d	60 m	67%	27%	71%	32%	52%	35%
21 d	2 h	53%	45%	51%	39%	31%	46%
21 d	4 h	50%	59%	34%	35%	10%	16%
21 d	8 h	0%	0%	17%	28%	0%	0%
21 d	16 h	0%	0%	0%	0%	0%	0%
14 d	60 m	67%	27%	71%	32%	52%	35%
14 d	2 h	56%	43%	51%	39%	32%	48%
14 d	4 h	50%	59%	34%	35%	10%	16%
14 d	8 h	0%	0%	0%	0%	0%	0%
14 d	16 h	0%	0%	0%	0%	0%	0%
7 d	60 m	69%	26%	71%	32%	52%	34%
7 d	2 h	56%	45%	51%	39%	37%	48%
7 d	4 h	0%	0%	0%	0%	0%	0%
7 d	8 h	0%	0%	0%	0%	0%	0%
7 d	16 h	0%	0%	0%	0%	0%	0%
4 d	60 m	70%	23%	71%	32%	53%	33%
4 d	2 h	59%	38%	43%	35%	26%	38%
4 d	4 h	0%	0%	0%	0%	0%	0%
4 d	8 h	0%	0%	0%	0%	0%	0%
4 d	16 h	0%	0%	0%	0%	0%	0%

Table D.7: $P(t \mid 30m, d)$

		Correct Attempts		Attempts		Correct Trials	
↓Hour	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	72%	20%	94%	11%	66%	28%
∞	2 h	67%	29%	70%	32%	45%	34%
∞	4 h	45%	54%	50%	39%	31%	46%
∞	8 h	38%	57%	34%	35%	10%	16%
∞	16 h	0%	0%	17%	28%	0%	0%
21 d	60 m	72%	18%	94%	11%	66%	22%
21 d	2 h	67%	27%	70%	32%	52%	36%
21 d	4 h	53%	45%	50%	39%	31%	46%
21 d	8 h	52%	59%	34%	35%	10%	16%
21 d	16 h	0%	0%	17%	28%	0%	0%
14 d	60 m	72%	18%	94%	11%	66%	22%
14 d	2 h	67%	26%	70%	32%	52%	36%
14 d	4 h	53%	45%	50%	39%	32%	48%
14 d	8 h	52%	60%	34%	35%	10%	16%
14 d	16 h	0%	0%	0%	0%	0%	0%
7 d	60 m	72%	17%	94%	11%	67%	23%
7 d	2 h	71%	24%	70%	32%	52%	38%
7 d	4 h	53%	46%	50%	39%	36%	48%
7 d	8 h	0%	0%	0%	0%	0%	0%
7 d	16 h	0%	0%	0%	0%	0%	0%
4 d	60 m	72%	17%	94%	11%	66%	20%
4 d	2 h	71%	23%	70%	32%	53%	34%
4 d	4 h	59%	38%	39%	32%	21%	30%
4 d	8 h	0%	0%	0%	0%	0%	0%
4 d	16 h	0%	0%	0%	0%	0%	0%

Table D.8: $P(t \mid h, d)$

		Correct Attempts		Attempts		Correct Trials	
↓Hour	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	73%	21%	98%	3%	72%	20%
∞	60 m	73%	20%	98%	3%	72%	19%
∞	2 h	72%	18%	91%	11%	65%	22%
∞	2 h	73%	21%	90%	12%	65%	23%
∞	4 h	72%	21%	71%	30%	47%	30%
∞	4 h	74%	23%	71%	30%	47%	30%
∞	8 h	58%	37%	50%	37%	31%	43%
∞	8 h	56%	39%	51%	37%	28%	42%
∞	16 h	32%	47%	34%	33%	14%	21%
∞	16 h	38%	56%	34%	33%	14%	20%
21 d	60 m	73%	18%	98%	3%	73%	16%
21 d	2 h	73%	16%	91%	11%	68%	18%
21 d	4 h	73%	18%	71%	30%	50%	31%
21 d	8 h	64%	27%	50%	37%	35%	46%
21 d	16 h	0%	0%	0%	0%	0%	0%
14 d	60 m	75%	18%	98%	3%	74%	15%
14 d	2 h	75%	14%	91%	11%	68%	17%
14 d	4 h	75%	15%	71%	30%	53%	29%
14 d	8 h	0%	0%	0%	0%	0%	0%
14 d	16 h	0%	0%	0%	0%	0%	0%
7 d	60 m	77%	15%	98%	3%	75%	15%
7 d	2 h	77%	15%	91%	11%	68%	19%
7 d	4 h	0%	0%	0%	0%	0%	0%
7 d	8 h	0%	0%	0%	0%	0%	0%
7 d	16 h	0%	0%	0%	0%	0%	0%
4 d	60 m	76%	14%	98%	3%	75%	15%
4 d	2 h	77%	14%	72%	18%	52%	21%
4 d	4 h	0%	0%	0%	0%	0%	0%
4 d	8 h	0%	0%	0%	0%	0%	0%
4 d	16 h	0%	0%	0%	0%	0%	0%

Table D.9: $P(t \mid 30m, w)$, western weekend

		Correct Attempts		Attempts		Correct Trials	
↓Hour	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	73%	21%	100%	1%	73%	21%
∞	60 m	73%	21%	100%	1%	73%	20%
∞	2 h	73%	21%	98%	4%	72%	18%
∞	2 h	73%	21%	98%	4%	72%	19%
∞	4 h	72%	18%	90%	12%	64%	23%
∞	4 h	73%	20%	90%	12%	64%	22%
∞	8 h	73%	21%	70%	30%	46%	32%
∞	8 h	74%	23%	70%	31%	47%	30%
∞	16 h	56%	38%	50%	37%	31%	43%
∞	16 h	55%	38%	50%	37%	28%	42%
21 d	60 m	74%	17%	100%	1%	74%	17%
21 d	2 h	73%	17%	98%	4%	73%	15%
21 d	4 h	73%	15%	90%	12%	67%	18%
21 d	8 h	73%	18%	70%	30%	50%	31%
21 d	16 h	65%	28%	50%	37%	35%	46%
14 d	60 m	75%	18%	100%	1%	75%	18%
14 d	2 h	75%	18%	98%	4%	73%	16%
14 d	4 h	75%	14%	90%	12%	68%	17%
14 d	8 h	74%	16%	70%	30%	52%	29%
14 d	16 h	0%	0%	0%	0%	0%	0%
7 d	60 m	76%	17%	100%	1%	76%	16%
7 d	2 h	76%	16%	98%	4%	75%	14%
7 d	4 h	77%	14%	90%	12%	67%	18%
7 d	8 h	0%	0%	0%	0%	0%	0%
7 d	16 h	0%	0%	0%	0%	0%	0%
4 d	60 m	76%	16%	100%	1%	76%	15%
4 d	2 h	76%	15%	98%	4%	75%	14%
4 d	4 h	77%	16%	62%	21%	44%	23%
4 d	8 h	0%	0%	0%	0%	0%	0%
4 d	16 h	0%	0%	0%	0%	0%	0%

Table D.10: $P(t \mid h, w)$, western weekend

		Correct Attempts		Attempts		Correct Trials	
↓Hour	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	73%	21%	98%	3%	72%	20%
∞	60 m	73%	20%	98%	3%	72%	19%
∞	2 h	72%	18%	91%	11%	65%	22%
∞	2 h	73%	21%	90%	12%	65%	23%
∞	4 h	72%	21%	71%	30%	47%	30%
∞	4 h	74%	23%	71%	30%	47%	30%
∞	8 h	58%	37%	50%	37%	31%	43%
∞	8 h	56%	39%	51%	37%	28%	42%
∞	16 h	32%	47%	34%	33%	14%	21%
∞	16 h	38%	56%	34%	33%	14%	20%
21 d	60 m	73%	18%	98%	3%	73%	16%
21 d	2 h	73%	16%	91%	11%	68%	18%
21 d	4 h	73%	18%	71%	30%	50%	31%
21 d	8 h	64%	27%	50%	37%	35%	46%
21 d	16 h	0%	0%	0%	0%	0%	0%
14 d	60 m	75%	18%	98%	3%	74%	15%
14 d	2 h	75%	14%	91%	11%	68%	17%
14 d	4 h	75%	15%	71%	30%	53%	29%
14 d	8 h	0%	0%	0%	0%	0%	0%
14 d	16 h	0%	0%	0%	0%	0%	0%
7 d	60 m	77%	15%	98%	3%	75%	15%
7 d	2 h	77%	15%	91%	11%	68%	19%
7 d	4 h	0%	0%	0%	0%	0%	0%
7 d	8 h	0%	0%	0%	0%	0%	0%
7 d	16 h	0%	0%	0%	0%	0%	0%
4 d	60 m	76%	14%	98%	3%	75%	15%
4 d	2 h	77%	14%	72%	18%	52%	21%
4 d	4 h	0%	0%	0%	0%	0%	0%
4 d	8 h	0%	0%	0%	0%	0%	0%
4 d	16 h	0%	0%	0%	0%	0%	0%

Table D.11: $P(t \mid 30m, w)$, local weekend

		Correct Attempts		Attempts		Correct Trials	
↓Hour	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	73%	21%	100%	1%	73%	21%
∞	60 m	73%	21%	100%	1%	73%	20%
∞	2 h	73%	21%	98%	4%	72%	18%
∞	2 h	73%	21%	98%	4%	72%	19%
∞	4 h	72%	18%	90%	12%	64%	23%
∞	4 h	73%	20%	90%	12%	64%	22%
∞	8 h	73%	21%	70%	30%	46%	32%
∞	8 h	74%	23%	70%	31%	47%	30%
∞	16 h	56%	38%	50%	37%	31%	43%
∞	16 h	55%	38%	50%	37%	28%	42%
21 d	60 m	74%	17%	100%	1%	74%	17%
21 d	2 h	73%	17%	98%	4%	73%	15%
21 d	4 h	73%	15%	90%	12%	67%	18%
21 d	8 h	73%	18%	70%	30%	50%	31%
21 d	16 h	65%	28%	50%	37%	35%	46%
14 d	60 m	75%	18%	100%	1%	75%	18%
14 d	2 h	75%	18%	98%	4%	73%	16%
14 d	4 h	75%	14%	90%	12%	68%	17%
14 d	8 h	74%	16%	70%	30%	52%	29%
14 d	16 h	0%	0%	0%	0%	0%	0%
7 d	60 m	76%	17%	100%	1%	76%	16%
7 d	2 h	76%	16%	98%	4%	75%	14%
7 d	4 h	77%	14%	90%	12%	67%	18%
7 d	8 h	0%	0%	0%	0%	0%	0%
7 d	16 h	0%	0%	0%	0%	0%	0%
4 d	60 m	76%	16%	100%	1%	76%	15%
4 d	2 h	76%	15%	98%	4%	75%	14%
4 d	4 h	77%	16%	62%	21%	44%	23%
4 d	8 h	0%	0%	0%	0%	0%	0%
4 d	16 h	0%	0%	0%	0%	0%	0%

Table D.12: $P(t \mid h, w)$, local weekend

		Correct Attempts		Attempts		Correct Trials	
↓Regime	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	74%	19%	100%	1%	74%	20%
∞	2 h	74%	19%	99%	1%	74%	20%
∞	4 h	74%	18%	99%	2%	74%	20%
∞	8 h	74%	18%	97%	5%	73%	20%
∞	16 h	75%	17%	95%	8%	73%	21%
42 d	60 m	74%	19%	100%	1%	74%	20%
42 d	2 h	74%	19%	99%	1%	74%	20%
42 d	4 h	74%	18%	99%	2%	74%	20%
42 d	8 h	74%	18%	97%	5%	73%	20%
42 d	16 h	75%	17%	95%	8%	73%	21%
28 d	60 m	74%	19%	100%	1%	74%	20%
28 d	2 h	74%	19%	99%	1%	74%	20%
28 d	4 h	74%	18%	99%	2%	74%	20%
28 d	8 h	74%	18%	97%	5%	73%	20%
28 d	16 h	75%	17%	95%	8%	73%	20%
21 d	60 m	74%	19%	100%	1%	74%	20%
21 d	2 h	74%	19%	99%	1%	74%	20%
21 d	4 h	74%	18%	99%	2%	74%	20%
21 d	8 h	74%	18%	97%	5%	73%	20%
21 d	16 h	75%	17%	95%	8%	73%	20%
14 d	60 m	74%	19%	100%	1%	74%	20%
14 d	2 h	74%	19%	99%	1%	74%	20%
14 d	4 h	74%	18%	99%	2%	73%	20%
14 d	8 h	74%	18%	97%	5%	73%	20%
14 d	16 h	75%	17%	95%	8%	73%	20%
7 d	60 m	74%	19%	100%	1%	74%	20%
7 d	2 h	74%	19%	99%	1%	74%	20%
7 d	4 h	74%	18%	98%	2%	74%	20%
7 d	8 h	74%	17%	97%	5%	73%	20%
7 d	16 h	75%	17%	95%	8%	73%	21%

Table D.13: $P(t \mid r)$

		Correct Attempts		Attempts		Correct Trials	
↓Regime	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	78%	11%	92%	13%	74%	15%
∞	2 h	78%	10%	88%	18%	73%	18%
∞	4 h	78%	12%	76%	27%	68%	16%
∞	8 h	77%	16%	52%	36%	48%	34%
∞	16 h	70%	26%	35%	36%	15%	23%

Table D.14: $P(t \mid r, 30m)$

		Correct Attempts		Attempts		Correct Trials	
↓Regime	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	77%	11%	95%	8%	75%	14%
∞	2 h	78%	11%	92%	13%	74%	16%
∞	4 h	78%	11%	88%	18%	73%	18%
∞	8 h	77%	12%	75%	27%	68%	17%
∞	16 h	76%	15%	52%	36%	47%	35%
42 d	60 m	77%	11%	95%	8%	76%	16%
42 d	2 h	78%	11%	92%	13%	74%	16%
42 d	4 h	78%	11%	88%	18%	73%	18%
42 d	8 h	78%	12%	75%	27%	68%	18%
42 d	16 h	77%	14%	51%	36%	46%	37%
28 d	60 m	77%	12%	95%	8%	76%	14%
28 d	2 h	77%	11%	92%	14%	74%	16%
28 d	4 h	78%	11%	88%	19%	73%	18%
28 d	8 h	77%	12%	75%	27%	68%	18%
28 d	16 h	77%	14%	50%	36%	37%	40%
21 d	60 m	77%	12%	95%	8%	76%	14%
21 d	2 h	77%	11%	91%	14%	74%	16%
21 d	4 h	78%	11%	87%	19%	73%	19%
21 d	8 h	77%	11%	74%	27%	67%	18%
21 d	16 h	78%	12%	42%	36%	23%	34%
14 d	60 m	78%	11%	95%	9%	76%	14%
14 d	2 h	78%	11%	91%	14%	74%	16%
14 d	4 h	78%	11%	87%	19%	73%	19%
14 d	8 h	78%	11%	70%	28%	65%	19%
14 d	16 h	0%	0%	0%	0%	0%	0%
7 d	60 m	77%	12%	94%	10%	75%	15%
7 d	2 h	78%	12%	90%	16%	73%	16%
7 d	4 h	78%	10%	79%	24%	70%	22%
7 d	8 h	0%	0%	0%	0%	0%	0%
7 d	16 h	0%	0%	0%	0%	0%	0%

Table D.15: $P(t \mid r, h)$

		Correct Attempts		Attempts		Correct Trials	
↓Regime	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	81%	9%	87%	17%	76%	12%
∞	60 m	81%	9%	87%	17%	76%	12%
∞	2 h	81%	9%	76%	23%	68%	18%
∞	2 h	81%	9%	76%	24%	68%	18%
∞	4 h	79%	11%	56%	32%	49%	33%
∞	4 h	80%	12%	56%	32%	49%	33%
∞	8 h	75%	18%	38%	34%	25%	36%
∞	8 h	75%	19%	38%	34%	24%	36%
∞	16 h	69%	39%	24%	29%	2%	4%
∞	16 h	69%	43%	24%	29%	2%	2%

Table D.16: $P(t \mid r, 30m, w)$, western weekend

		Correct Attempts		Attempts		Correct Trials	
↓Regime	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	81%	10%	92%	11%	78%	13%
∞	60 m	81%	10%	93%	11%	78%	12%
∞	2 h	81%	9%	87%	18%	76%	12%
∞	2 h	81%	10%	87%	18%	76%	12%
∞	4 h	81%	10%	76%	24%	68%	18%
∞	4 h	81%	9%	76%	24%	68%	18%
∞	8 h	79%	12%	56%	32%	48%	34%
∞	8 h	80%	12%	56%	32%	48%	32%
∞	16 h	74%	19%	38%	34%	24%	36%
∞	16 h	74%	19%	38%	34%	24%	35%
42 d	60 m	81%	9%	92%	11%	78%	13%
42 d	2 h	82%	8%	87%	18%	76%	13%
42 d	4 h	82%	9%	76%	24%	68%	18%
42 d	8 h	79%	12%	55%	32%	47%	33%
42 d	16 h	75%	20%	30%	26%	22%	32%
28 d	60 m	81%	10%	92%	12%	78%	12%
28 d	2 h	82%	8%	86%	18%	76%	13%
28 d	4 h	82%	9%	75%	24%	68%	18%
28 d	8 h	81%	11%	47%	26%	43%	23%
28 d	16 h	74%	21%	25%	26%	12%	17%
21 d	60 m	81%	10%	92%	12%	78%	12%
21 d	2 h	82%	9%	86%	18%	76%	13%
21 d	4 h	82%	8%	73%	24%	67%	17%
21 d	8 h	82%	12%	43%	24%	39%	24%
21 d	16 h	0%	0%	0%	1%	0%	0%
14 d	60 m	81%	10%	92%	12%	78%	12%
14 d	2 h	82%	9%	85%	18%	75%	14%
14 d	4 h	83%	8%	64%	20%	58%	14%
14 d	8 h	83%	13%	34%	23%	29%	29%
14 d	16 h	0%	0%	0%	0%	0%	0%
7 d	60 m	79%	10%	88%	13%	74%	13%
7 d	2 h	81%	9%	70%	16%	61%	13%
7 d	4 h	84%	8%	40%	20%	35%	22%
7 d	8 h	0%	0%	0%	0%	0%	0%
7 d	16 h	0%	0%	0%	0%	0%	0%

Table D.17: $P(t \mid r, h, w)$, western weekend

		Correct Attempts		Attempts		Correct Trials	
↓Regime	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	81%	9%	87%	17%	76%	12%
∞	60 m	81%	9%	87%	17%	76%	12%
∞	2 h	81%	9%	76%	23%	68%	18%
∞	2 h	81%	9%	76%	24%	68%	18%
∞	4 h	79%	11%	56%	32%	49%	33%
∞	4 h	80%	12%	56%	32%	49%	33%
∞	8 h	75%	18%	38%	34%	25%	36%
∞	8 h	75%	19%	38%	34%	24%	36%
∞	16 h	69%	39%	24%	29%	2%	4%
∞	16 h	69%	43%	24%	29%	2%	2%

Table D.18: $P(t \mid r, 30m, w)$, local weekend

		Correct Attempts		Attempts		Correct Trials	
↓Regime	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	81%	10%	92%	11%	78%	13%
∞	60 m	81%	10%	93%	11%	78%	12%
∞	2 h	81%	9%	87%	18%	76%	12%
∞	2 h	81%	10%	87%	18%	76%	12%
∞	4 h	81%	10%	76%	24%	68%	18%
∞	4 h	81%	9%	76%	24%	68%	18%
∞	8 h	79%	12%	56%	32%	48%	34%
∞	8 h	80%	12%	56%	32%	48%	32%
∞	16 h	74%	19%	38%	34%	24%	36%
∞	16 h	74%	19%	38%	34%	24%	35%
42 d	60 m	81%	9%	92%	11%	78%	13%
42 d	2 h	82%	8%	87%	18%	76%	13%
42 d	4 h	82%	9%	76%	24%	68%	18%
42 d	8 h	79%	12%	55%	32%	47%	33%
42 d	16 h	75%	20%	30%	26%	22%	32%
28 d	60 m	81%	10%	92%	12%	78%	12%
28 d	2 h	82%	8%	86%	18%	76%	13%
28 d	4 h	82%	9%	75%	24%	68%	18%
28 d	8 h	81%	11%	47%	26%	43%	23%
28 d	16 h	74%	21%	25%	26%	12%	17%
21 d	60 m	81%	10%	92%	12%	78%	12%
21 d	2 h	82%	9%	86%	18%	76%	13%
21 d	4 h	82%	8%	73%	24%	67%	17%
21 d	8 h	82%	12%	43%	24%	39%	24%
21 d	16 h	0%	0%	0%	1%	0%	0%
14 d	60 m	81%	10%	92%	12%	78%	12%
14 d	2 h	82%	9%	85%	18%	75%	14%
14 d	4 h	83%	8%	64%	20%	58%	14%
14 d	8 h	83%	13%	34%	23%	29%	29%
14 d	16 h	0%	0%	0%	0%	0%	0%
7 d	60 m	79%	10%	88%	13%	74%	13%
7 d	2 h	81%	9%	70%	16%	61%	13%
7 d	4 h	84%	8%	40%	20%	35%	22%
7 d	8 h	0%	0%	0%	0%	0%	0%
7 d	16 h	0%	0%	0%	0%	0%	0%

Table D.19: $P(t \mid r, h, w)$, local weekend

		Correct Attempts		Attempts		Correct Trials	
↓Regime	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	77%	13%	57%	34%	52%	33%
∞	2 h	74%	22%	38%	36%	24%	36%
∞	4 h	66%	49%	25%	31%	0%	1%
∞	8 h	0%	0%	12%	22%	0%	0%
∞	16 h	0%	0%	4%	10%	0%	0%

Table D.20: $P(t \mid r, 30m, d)$

		Correct Attempts		Attempts		Correct Trials	
↓Regime	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	78%	12%	78%	25%	69%	18%
∞	2 h	77%	12%	56%	34%	51%	37%
∞	4 h	73%	22%	38%	36%	23%	34%
∞	8 h	62%	55%	25%	31%	0%	0%
∞	16 h	0%	0%	12%	22%	0%	0%
42 d	60 m	78%	11%	78%	25%	71%	21%
42 d	2 h	77%	11%	55%	34%	46%	37%
42 d	4 h	73%	22%	35%	35%	20%	29%
42 d	8 h	0%	0%	0%	0%	0%	0%
42 d	16 h	0%	0%	0%	0%	0%	0%
28 d	60 m	78%	11%	77%	25%	70%	20%
28 d	2 h	78%	12%	53%	33%	44%	37%
28 d	4 h	77%	22%	13%	14%	6%	8%
28 d	8 h	0%	0%	0%	0%	0%	0%
28 d	16 h	0%	0%	0%	0%	0%	0%
21 d	60 m	78%	11%	76%	25%	68%	19%
21 d	2 h	79%	12%	47%	30%	36%	36%
21 d	4 h	0%	0%	0%	1%	0%	0%
21 d	8 h	0%	0%	0%	0%	0%	0%
21 d	16 h	0%	0%	0%	0%	0%	0%
14 d	60 m	77%	11%	72%	24%	60%	20%
14 d	2 h	82%	10%	24%	15%	21%	15%
14 d	4 h	0%	0%	0%	0%	0%	0%
14 d	8 h	0%	0%	0%	0%	0%	0%
14 d	16 h	0%	0%	0%	0%	0%	0%
7 d	60 m	79%	9%	36%	13%	31%	10%
7 d	2 h	0%	0%	0%	0%	0%	0%
7 d	4 h	0%	0%	0%	0%	0%	0%
7 d	8 h	0%	0%	0%	0%	0%	0%
7 d	16 h	0%	0%	0%	0%	0%	0%

Table D.21: $P(t \mid r, h, d)$

		Correct Attempts		Attempts		Correct Trials	
↓Tower	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	79%	10%	80%	14%	64%	18%
∞	2 h	81%	10%	74%	19%	60%	19%
∞	4 h	82%	10%	67%	23%	54%	21%
∞	8 h	82%	10%	52%	27%	46%	25%
∞	16 h	79%	14%	34%	30%	26%	37%
28 d	60 m	79%	11%	79%	14%	64%	14%
28 d	2 h	81%	11%	73%	19%	60%	18%
28 d	4 h	82%	10%	66%	23%	54%	22%
28 d	8 h	83%	10%	50%	27%	40%	24%
28 d	16 h	79%	14%	28%	27%	19%	26%
21 d	60 m	79%	11%	79%	14%	64%	15%
21 d	2 h	80%	11%	73%	19%	60%	17%
21 d	4 h	81%	11%	65%	23%	54%	21%
21 d	8 h	82%	10%	48%	26%	38%	24%
21 d	16 h	78%	13%	23%	26%	10%	15%
14 d	60 m	79%	12%	79%	14%	63%	15%
14 d	2 h	80%	11%	72%	19%	60%	17%
14 d	4 h	81%	12%	64%	23%	52%	22%
14 d	8 h	81%	10%	43%	26%	33%	22%
14 d	16 h	62%	39%	10%	12%	3%	4%
7 d	60 m	78%	12%	78%	15%	59%	16%
7 d	2 h	79%	11%	70%	20%	56%	20%
7 d	4 h	80%	11%	55%	23%	45%	21%
7 d	8 h	70%	18%	18%	16%	9%	11%
7 d	16 h	0%	0%	3%	5%	0%	0%
4 d	60 m	74%	13%	75%	16%	55%	17%
4 d	2 h	77%	12%	63%	21%	49%	18%
4 d	4 h	75%	14%	36%	21%	26%	16%
4 d	8 h	47%	46%	7%	9%	1%	2%
4 d	16 h	0%	0%	1%	3%	0%	0%

Table D.22: $P(t \mid h, c)$

		Correct Attempts		Attempts		Correct Trials	
↓ Tower	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	82%	9%	75%	18%	62%	17%
∞	60 m	82%	9%	75%	18%	62%	16%
∞	2 h	83%	8%	66%	22%	57%	18%
∞	2 h	83%	8%	66%	22%	57%	18%
∞	4 h	84%	8%	54%	25%	46%	21%
∞	4 h	84%	8%	54%	25%	46%	23%
∞	8 h	83%	12%	38%	28%	29%	33%
∞	8 h	83%	11%	38%	28%	29%	31%
∞	16 h	79%	17%	24%	27%	13%	19%
∞	16 h	80%	17%	24%	27%	9%	14%
28 d	60 m	82%	9%	74%	18%	62%	15%
28 d	2 h	83%	8%	65%	22%	56%	17%
28 d	4 h	84%	7%	51%	24%	43%	21%
28 d	8 h	82%	12%	31%	23%	23%	22%
28 d	16 h	77%	24%	14%	17%	4%	6%
21 d	60 m	81%	10%	74%	18%	61%	15%
21 d	2 h	82%	8%	64%	22%	55%	17%
21 d	4 h	83%	8%	50%	23%	42%	20%
21 d	8 h	82%	12%	27%	20%	22%	19%
21 d	16 h	66%	35%	7%	9%	1%	2%
14 d	60 m	81%	9%	73%	18%	61%	16%
14 d	2 h	82%	10%	63%	22%	53%	18%
14 d	4 h	82%	9%	45%	22%	38%	16%
14 d	8 h	80%	14%	21%	18%	16%	15%
14 d	16 h	38%	56%	2%	4%	0%	0%
7 d	60 m	78%	10%	69%	18%	55%	18%
7 d	2 h	80%	9%	55%	20%	46%	16%
7 d	4 h	79%	12%	31%	18%	25%	13%
7 d	8 h	58%	36%	5%	7%	2%	3%
7 d	16 h	0%	0%	1%	2%	0%	0%
4 d	60 m	68%	12%	57%	16%	40%	11%
4 d	2 h	72%	13%	39%	16%	29%	11%
4 d	4 h	66%	22%	12%	9%	7%	6%
4 d	8 h	32%	47%	2%	4%	0%	0%
4 d	16 h	0%	0%	0%	1%	0%	0%

Table D.23: $P(t \mid h, w, c)$, western weekend

		Correct Attempts		Attempts		Correct Trials	
↓Tower	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	82%	9%	75%	18%	62%	17%
∞	60 m	82%	9%	75%	18%	62%	16%
∞	2 h	83%	8%	66%	22%	57%	18%
∞	2 h	83%	8%	66%	22%	57%	18%
∞	4 h	84%	8%	54%	25%	46%	21%
∞	4 h	84%	8%	54%	25%	46%	23%
∞	8 h	83%	12%	38%	28%	29%	33%
∞	8 h	83%	11%	38%	28%	29%	31%
∞	16 h	79%	17%	24%	27%	13%	19%
∞	16 h	80%	17%	24%	27%	9%	14%
28 d	60 m	82%	9%	74%	18%	62%	15%
28 d	2 h	83%	8%	65%	22%	56%	17%
28 d	4 h	84%	7%	51%	24%	43%	21%
28 d	8 h	82%	12%	31%	23%	23%	22%
28 d	16 h	77%	24%	14%	17%	4%	6%
21 d	60 m	81%	10%	74%	18%	61%	15%
21 d	2 h	82%	8%	64%	22%	55%	17%
21 d	4 h	83%	8%	50%	23%	42%	20%
21 d	8 h	82%	12%	27%	20%	22%	19%
21 d	16 h	66%	35%	7%	9%	1%	2%
14 d	60 m	81%	9%	73%	18%	61%	16%
14 d	2 h	82%	10%	63%	22%	53%	18%
14 d	4 h	82%	9%	45%	22%	38%	16%
14 d	8 h	80%	14%	21%	18%	16%	15%
14 d	16 h	38%	56%	2%	4%	0%	0%
7 d	60 m	78%	10%	69%	18%	55%	18%
7 d	2 h	80%	9%	55%	20%	46%	16%
7 d	4 h	79%	12%	31%	18%	25%	13%
7 d	8 h	58%	36%	5%	7%	2%	3%
7 d	16 h	0%	0%	1%	2%	0%	0%
4 d	60 m	68%	12%	57%	16%	40%	11%
4 d	2 h	72%	13%	39%	16%	29%	11%
4 d	4 h	66%	22%	12%	9%	7%	6%
4 d	8 h	32%	47%	2%	4%	0%	0%
4 d	16 h	0%	0%	0%	1%	0%	0%

Table D.24: $P(t \mid h, w, c)$, local weekend

		Correct Attempts		Attempts		Correct Trials	
↓ Tower	Weight	Median	MAD	μ	σ	Median	MAD
∞	60 m	81%	11%	57%	24%	47%	24%
∞	2 h	81%	12%	38%	28%	31%	30%
∞	4 h	78%	20%	25%	28%	11%	17%
∞	8 h	65%	52%	15%	22%	0%	0%
∞	16 h	0%	0%	7%	16%	0%	0%
28 d	60 m	80%	10%	55%	23%	42%	20%
28 d	2 h	80%	12%	34%	26%	26%	24%
28 d	4 h	71%	34%	14%	18%	3%	5%
28 d	8 h	0%	0%	1%	2%	0%	0%
28 d	16 h	0%	0%	0%	0%	0%	0%
21 d	60 m	79%	10%	54%	23%	42%	19%
21 d	2 h	78%	11%	32%	25%	22%	20%
21 d	4 h	53%	57%	5%	7%	1%	2%
21 d	8 h	0%	0%	0%	1%	0%	0%
21 d	16 h	0%	0%	0%	0%	0%	0%
14 d	60 m	77%	12%	51%	22%	41%	17%
14 d	2 h	76%	11%	25%	21%	15%	14%
14 d	4 h	37%	54%	2%	4%	0%	0%
14 d	8 h	0%	0%	0%	1%	0%	0%
14 d	16 h	0%	0%	0%	0%	0%	0%
7 d	60 m	74%	11%	43%	20%	33%	13%
7 d	2 h	44%	26%	6%	7%	2%	3%
7 d	4 h	0%	0%	1%	2%	0%	0%
7 d	8 h	0%	0%	0%	0%	0%	0%
7 d	16 h	0%	0%	0%	0%	0%	0%
4 d	60 m	19%	22%	11%	7%	2%	2%
4 d	2 h	3%	4%	2%	3%	0%	0%
4 d	4 h	0%	0%	0%	1%	0%	0%
4 d	8 h	0%	0%	0%	0%	0%	0%
4 d	16 h	0%	0%	0%	0%	0%	0%

Table D.25: $P(t \mid h, d, c)$

			Correct Attempts		Attempts		Correct Trials	
↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
∞	60 m	F	76%	14%	100%	1%	76%	14%
∞	60 m	T	76%	14%	100%	1%	76%	13%
∞	2 h	F	76%	14%	99%	1%	76%	14%
∞	2 h	T	77%	13%	99%	1%	76%	14%
∞	4 h	F	77%	14%	99%	2%	76%	14%
∞	4 h	T	77%	13%	99%	2%	76%	14%
∞	8 h	F	77%	13%	97%	4%	76%	14%
∞	8 h	T	77%	14%	97%	5%	76%	15%
∞	16 h	F	77%	14%	96%	8%	75%	17%
∞	16 h	T	76%	13%	95%	8%	74%	17%
42 d	60 m	F	77%	14%	100%	1%	77%	14%
42 d	60 m	T	77%	14%	100%	1%	77%	14%
42 d	2 h	F	77%	13%	99%	1%	77%	14%
42 d	2 h	T	77%	13%	99%	1%	77%	14%
42 d	4 h	F	77%	14%	99%	2%	77%	14%
42 d	4 h	T	77%	13%	99%	2%	77%	14%
42 d	8 h	F	77%	13%	97%	4%	76%	14%
42 d	8 h	T	77%	14%	97%	5%	76%	15%
42 d	16 h	F	77%	14%	96%	8%	76%	17%
42 d	16 h	T	76%	13%	95%	8%	74%	17%
28 d	60 m	F	77%	14%	100%	1%	77%	14%
28 d	60 m	T	77%	14%	100%	1%	77%	14%
28 d	2 h	F	77%	14%	99%	1%	77%	14%
28 d	2 h	T	77%	13%	99%	1%	77%	14%
28 d	4 h	F	77%	14%	99%	2%	77%	14%
28 d	4 h	T	77%	14%	99%	2%	77%	14%
28 d	8 h	F	77%	14%	97%	4%	76%	14%
28 d	8 h	T	77%	14%	97%	5%	76%	14%
28 d	16 h	F	77%	13%	96%	8%	76%	17%
28 d	16 h	T	77%	13%	95%	8%	74%	17%
21 d	60 m	F	77%	14%	100%	1%	77%	14%
21 d	60 m	T	77%	14%	100%	1%	77%	14%
21 d	2 h	F	77%	13%	99%	1%	77%	14%
21 d	2 h	T	77%	13%	99%	1%	77%	14%
21 d	4 h	F	77%	13%	99%	2%	76%	14%

Table D.26: $P(t \mid r, h), P(t \mid r)$

			Correct Attempts		Attempts		Correct Trials	
↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
21 d	4 h	T	77%	14%	99%	2%	76%	14%
21 d	8 h	F	77%	13%	97%	4%	76%	14%
21 d	8 h	T	77%	14%	97%	5%	76%	14%
21 d	16 h	F	77%	13%	96%	8%	75%	17%
21 d	16 h	T	76%	13%	95%	8%	74%	17%
14 d	60 m	F	77%	14%	100%	1%	76%	14%
14 d	60 m	T	77%	13%	100%	1%	76%	14%
14 d	2 h	F	77%	13%	99%	1%	76%	14%
14 d	2 h	T	77%	13%	99%	1%	77%	13%
14 d	4 h	F	77%	12%	99%	2%	77%	14%
14 d	4 h	T	77%	12%	99%	2%	77%	13%
14 d	8 h	F	78%	12%	97%	4%	76%	14%
14 d	8 h	T	77%	13%	97%	5%	76%	15%
14 d	16 h	F	77%	14%	96%	8%	76%	17%
14 d	16 h	T	75%	17%	95%	8%	73%	20%
7 d	60 m	F	76%	14%	100%	1%	76%	14%
7 d	60 m	T	76%	13%	100%	1%	76%	14%
7 d	2 h	F	76%	12%	99%	1%	76%	14%
7 d	2 h	T	76%	12%	99%	1%	76%	14%
7 d	4 h	F	76%	12%	99%	2%	76%	14%
7 d	4 h	T	76%	12%	98%	2%	76%	14%
7 d	8 h	F	77%	13%	97%	4%	76%	16%
7 d	8 h	T	74%	17%	97%	5%	73%	20%
7 d	16 h	F	76%	15%	95%	8%	74%	19%
7 d	16 h	T	75%	17%	95%	8%	73%	21%

Table D.26 (Continued): $P(t \mid r, h), P(t \mid r)$

			Correct Attempts		Attempts		Correct Trials	
↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
∞	60 m	F	76%	13%	100%	1%	76%	13%
∞	60 m	T	78%	12%	100%	1%	78%	12%
∞	2 h	F	79%	12%	99%	1%	79%	12%
∞	2 h	T	78%	12%	99%	1%	78%	12%
∞	4 h	F	79%	12%	99%	2%	79%	12%
∞	4 h	T	79%	12%	99%	2%	78%	13%
∞	8 h	F	79%	12%	97%	4%	79%	12%
∞	8 h	T	78%	12%	97%	5%	78%	14%
∞	16 h	F	79%	12%	96%	8%	78%	14%
∞	16 h	T	77%	14%	95%	8%	74%	18%
42 d	60 m	F	77%	13%	100%	1%	77%	13%
42 d	60 m	T	78%	12%	100%	1%	78%	12%
42 d	2 h	F	79%	13%	99%	1%	78%	12%
42 d	2 h	T	78%	12%	99%	1%	78%	13%
42 d	4 h	F	80%	12%	99%	2%	79%	13%
42 d	4 h	T	79%	12%	99%	2%	78%	13%
42 d	8 h	F	79%	12%	97%	4%	79%	12%
42 d	8 h	T	77%	14%	97%	5%	76%	15%
42 d	16 h	F	78%	13%	96%	8%	77%	15%
42 d	16 h	T	76%	13%	95%	8%	74%	17%
28 d	60 m	F	77%	12%	100%	1%	77%	13%
28 d	60 m	T	78%	12%	100%	1%	78%	12%
28 d	2 h	F	78%	12%	99%	1%	78%	13%
28 d	2 h	T	78%	13%	99%	1%	78%	13%
28 d	4 h	F	80%	12%	99%	2%	79%	12%
28 d	4 h	T	78%	12%	99%	2%	77%	14%
28 d	8 h	F	79%	12%	97%	4%	78%	14%
28 d	8 h	T	77%	14%	97%	5%	76%	14%
28 d	16 h	F	78%	13%	96%	8%	77%	15%
28 d	16 h	T	77%	13%	95%	8%	74%	17%
21 d	60 m	F	76%	13%	100%	1%	76%	13%
21 d	60 m	T	76%	13%	100%	1%	76%	13%
21 d	2 h	F	77%	12%	99%	1%	77%	13%
21 d	2 h	T	78%	14%	99%	1%	78%	14%
21 d	4 h	F	79%	13%	99%	2%	79%	13%

Table D.27: $P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

			Correct Attempts		Attempts		Correct Trials	
↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
21 d	4 h	T	77%	14%	99%	2%	76%	14%
21 d	8 h	F	78%	12%	97%	4%	78%	14%
21 d	8 h	T	77%	14%	97%	5%	76%	14%
21 d	16 h	F	78%	13%	96%	8%	76%	16%
21 d	16 h	T	76%	13%	95%	8%	74%	17%
14 d	60 m	F	74%	12%	100%	1%	74%	13%
14 d	60 m	T	75%	12%	100%	1%	75%	12%
14 d	2 h	F	76%	12%	99%	1%	76%	12%
14 d	2 h	T	78%	14%	99%	1%	78%	14%
14 d	4 h	F	78%	13%	99%	2%	78%	13%
14 d	4 h	T	77%	12%	99%	2%	77%	13%
14 d	8 h	F	78%	12%	97%	4%	77%	14%
14 d	8 h	T	77%	13%	97%	5%	76%	15%
14 d	16 h	F	78%	13%	96%	8%	77%	15%
14 d	16 h	T	75%	17%	95%	8%	73%	20%
7 d	60 m	F	75%	14%	100%	1%	75%	14%
7 d	60 m	T	75%	12%	100%	1%	75%	14%
7 d	2 h	F	75%	12%	99%	1%	75%	15%
7 d	2 h	T	76%	12%	99%	1%	76%	14%
7 d	4 h	F	76%	12%	99%	2%	76%	13%
7 d	4 h	T	76%	12%	98%	2%	76%	14%
7 d	8 h	F	77%	12%	97%	4%	76%	14%
7 d	8 h	T	74%	17%	97%	5%	73%	20%
7 d	16 h	F	76%	15%	95%	8%	74%	19%
7 d	16 h	T	75%	17%	95%	8%	73%	21%

Table D.27 (Continued): $P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

			Correct Attempts		Attempts		Correct Trials	
↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
∞	60 m	F	76%	13%	100%	1%	76%	13%
∞	60 m	F	76%	13%	100%	1%	76%	13%
∞	60 m	T	77%	12%	100%	1%	77%	13%
∞	60 m	T	77%	13%	100%	1%	77%	13%
∞	2 h	F	77%	13%	99%	1%	77%	13%
∞	2 h	F	77%	13%	99%	1%	77%	13%
∞	2 h	T	80%	11%	99%	1%	80%	12%
∞	2 h	T	80%	11%	99%	1%	80%	12%
∞	4 h	F	80%	11%	99%	2%	80%	11%
∞	4 h	F	80%	12%	99%	2%	80%	12%
∞	4 h	T	79%	12%	99%	2%	79%	12%
∞	4 h	T	79%	12%	99%	2%	78%	12%
∞	8 h	F	80%	10%	98%	4%	80%	12%
∞	8 h	F	80%	10%	98%	4%	80%	12%
∞	8 h	T	78%	12%	97%	5%	78%	13%
∞	8 h	T	78%	12%	97%	5%	78%	13%
∞	16 h	F	80%	11%	96%	8%	79%	12%
∞	16 h	F	80%	11%	96%	8%	79%	12%
∞	16 h	T	78%	14%	95%	8%	76%	16%
∞	16 h	T	78%	14%	95%	8%	76%	15%
42 d	60 m	F	77%	13%	100%	1%	77%	13%
42 d	60 m	T	77%	12%	100%	1%	77%	13%
42 d	2 h	F	77%	13%	99%	1%	77%	13%
42 d	2 h	T	80%	12%	99%	1%	80%	12%
42 d	4 h	F	80%	10%	99%	2%	80%	12%
42 d	4 h	T	79%	11%	99%	2%	79%	12%
42 d	8 h	F	81%	10%	98%	4%	80%	11%
42 d	8 h	T	79%	12%	97%	5%	77%	12%
42 d	16 h	F	80%	11%	96%	8%	79%	12%
42 d	16 h	T	77%	14%	95%	8%	74%	17%
28 d	60 m	F	77%	12%	100%	1%	77%	12%
28 d	60 m	T	78%	12%	100%	1%	77%	12%
28 d	2 h	F	78%	13%	99%	1%	77%	13%
28 d	2 h	T	80%	12%	99%	1%	79%	13%
28 d	4 h	F	80%	10%	99%	2%	80%	11%

Table D.28: $P(t \mid r, h, d), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

			Correct Attempts		Attempts		Correct Trials	
↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
28 d	4 h	T	79%	12%	99%	2%	78%	13%
28 d	8 h	F	81%	10%	98%	4%	80%	11%
28 d	8 h	T	77%	12%	97%	5%	76%	15%
28 d	16 h	F	80%	11%	96%	8%	78%	13%
28 d	16 h	T	77%	13%	95%	8%	74%	17%
21 d	60 m	F	76%	13%	100%	1%	76%	12%
21 d	60 m	T	76%	12%	100%	1%	76%	12%
21 d	2 h	F	77%	13%	99%	1%	77%	13%
21 d	2 h	T	79%	12%	99%	1%	79%	13%
21 d	4 h	F	80%	11%	99%	2%	80%	11%
21 d	4 h	T	79%	12%	99%	2%	79%	13%
21 d	8 h	F	81%	10%	98%	4%	80%	11%
21 d	8 h	T	77%	13%	97%	5%	76%	15%
21 d	16 h	F	80%	11%	96%	8%	79%	13%
21 d	16 h	T	76%	13%	95%	8%	74%	17%
14 d	60 m	F	75%	14%	100%	1%	74%	14%
14 d	60 m	T	75%	12%	100%	1%	75%	12%
14 d	2 h	F	76%	12%	99%	1%	76%	12%
14 d	2 h	T	79%	12%	99%	1%	79%	12%
14 d	4 h	F	80%	11%	99%	2%	80%	12%
14 d	4 h	T	78%	12%	99%	2%	78%	13%
14 d	8 h	F	80%	11%	98%	4%	80%	12%
14 d	8 h	T	77%	12%	97%	5%	76%	15%
14 d	16 h	F	79%	12%	96%	8%	78%	14%
14 d	16 h	T	75%	17%	95%	8%	73%	20%
7 d	60 m	F	76%	13%	100%	1%	76%	13%
7 d	60 m	T	76%	13%	100%	1%	76%	13%
7 d	2 h	F	77%	13%	99%	1%	77%	13%
7 d	2 h	T	78%	12%	99%	1%	77%	13%
7 d	4 h	F	78%	13%	99%	2%	78%	13%
7 d	4 h	T	76%	12%	98%	2%	76%	12%
7 d	8 h	F	78%	12%	97%	4%	78%	14%
7 d	8 h	T	74%	17%	97%	5%	73%	20%
7 d	16 h	F	78%	12%	95%	8%	77%	14%
7 d	16 h	T	75%	17%	95%	8%	73%	21%

Table D.28 (Continued): $P(t \mid r, h, d), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

			Correct Attempts		Attempts		Correct Trials	
↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
∞	60 m	F	80%	12%	100%	1%	80%	12%
∞	60 m	F	80%	11%	100%	1%	80%	11%
∞	60 m	T	80%	12%	100%	1%	80%	12%
∞	60 m	T	80%	11%	100%	1%	80%	11%
∞	2 h	F	80%	12%	99%	1%	80%	12%
∞	2 h	F	80%	11%	99%	1%	80%	11%
∞	2 h	T	80%	12%	99%	1%	80%	12%
∞	2 h	T	80%	12%	99%	1%	80%	12%
∞	4 h	F	80%	11%	99%	2%	80%	12%
∞	4 h	F	80%	11%	99%	2%	80%	12%
∞	4 h	T	79%	12%	99%	2%	79%	13%
∞	4 h	T	79%	12%	99%	2%	79%	12%
∞	8 h	F	80%	11%	98%	4%	80%	12%
∞	8 h	F	80%	11%	98%	4%	80%	12%
∞	8 h	T	79%	12%	97%	5%	77%	12%
∞	8 h	T	79%	12%	97%	5%	77%	14%
∞	16 h	F	80%	11%	96%	8%	78%	14%
∞	16 h	F	80%	11%	96%	8%	78%	13%
∞	16 h	T	78%	14%	95%	8%	76%	16%
∞	16 h	T	78%	14%	95%	8%	76%	15%
42 d	60 m	F	80%	11%	100%	1%	80%	12%
42 d	60 m	T	80%	11%	100%	1%	80%	11%
42 d	2 h	F	80%	11%	99%	1%	80%	12%
42 d	2 h	T	80%	11%	99%	1%	80%	12%
42 d	4 h	F	80%	11%	99%	2%	80%	12%
42 d	4 h	T	79%	12%	99%	2%	79%	13%
42 d	8 h	F	80%	11%	98%	4%	80%	12%
42 d	8 h	T	79%	12%	97%	5%	77%	12%
42 d	16 h	F	80%	11%	96%	8%	78%	14%
42 d	16 h	T	77%	14%	95%	8%	74%	17%
28 d	60 m	F	80%	11%	100%	1%	80%	12%
28 d	60 m	T	80%	11%	100%	1%	80%	12%
28 d	2 h	F	80%	11%	99%	1%	80%	12%
28 d	2 h	T	80%	11%	99%	1%	80%	12%
28 d	4 h	F	81%	11%	99%	2%	80%	12%

Table D.29: $P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$, western weekend

			Correct Attempts		Attempts		Correct Trials	
↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
28 d	4 h	T	79%	12%	99%	2%	78%	13%
28 d	8 h	F	80%	11%	98%	4%	80%	12%
28 d	8 h	T	77%	12%	97%	5%	76%	15%
28 d	16 h	F	79%	12%	96%	8%	78%	14%
28 d	16 h	T	77%	13%	95%	8%	74%	17%
21 d	60 m	F	80%	12%	100%	1%	80%	12%
21 d	60 m	T	80%	12%	100%	1%	80%	12%
21 d	2 h	F	80%	11%	99%	1%	80%	12%
21 d	2 h	T	80%	11%	99%	1%	80%	12%
21 d	4 h	F	81%	11%	99%	2%	80%	12%
21 d	4 h	T	79%	12%	99%	2%	79%	13%
21 d	8 h	F	80%	11%	98%	4%	80%	12%
21 d	8 h	T	77%	13%	97%	5%	76%	15%
21 d	16 h	F	79%	12%	96%	8%	78%	14%
21 d	16 h	T	76%	13%	95%	8%	74%	17%
14 d	60 m	F	80%	12%	100%	1%	80%	12%
14 d	60 m	T	80%	12%	100%	1%	80%	12%
14 d	2 h	F	80%	12%	99%	1%	80%	12%
14 d	2 h	T	80%	12%	99%	1%	80%	12%
14 d	4 h	F	80%	11%	99%	2%	80%	12%
14 d	4 h	T	78%	12%	99%	2%	78%	13%
14 d	8 h	F	79%	11%	98%	4%	79%	12%
14 d	8 h	T	77%	12%	97%	5%	76%	15%
14 d	16 h	F	78%	12%	96%	8%	77%	14%
14 d	16 h	T	75%	17%	95%	8%	73%	20%
7 d	60 m	F	78%	14%	100%	1%	78%	14%
7 d	60 m	T	79%	13%	100%	1%	79%	13%
7 d	2 h	F	79%	13%	99%	1%	78%	14%
7 d	2 h	T	78%	12%	99%	1%	77%	13%
7 d	4 h	F	78%	12%	99%	2%	78%	13%
7 d	4 h	T	76%	12%	98%	2%	76%	12%
7 d	8 h	F	78%	12%	97%	4%	77%	13%
7 d	8 h	T	74%	17%	97%	5%	73%	20%
7 d	16 h	F	78%	12%	95%	8%	77%	15%
7 d	16 h	T	75%	17%	95%	8%	73%	21%

Table D.29 (Continued): $P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$, western weekend

				Correct Attempts		Attempts		Correct Trials	
↓ Tower	↓ Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
∞	∞	60 m	F	76%	15%	100%	1%	76%	15%
∞	∞	60 m	T	76%	14%	100%	1%	76%	15%
∞	∞	2 h	F	77%	14%	99%	1%	77%	14%
∞	∞	2 h	T	77%	14%	99%	1%	77%	14%
∞	∞	4 h	F	77%	12%	99%	2%	77%	14%
∞	∞	4 h	T	77%	13%	99%	2%	77%	14%
∞	∞	8 h	F	78%	12%	98%	4%	77%	15%
∞	∞	8 h	T	77%	14%	97%	5%	76%	16%
∞	∞	16 h	F	78%	14%	96%	8%	77%	15%
∞	∞	16 h	T	76%	14%	95%	8%	73%	17%
∞	42 d	60 m	F	76%	15%	100%	1%	76%	15%
∞	42 d	60 m	T	76%	14%	100%	1%	76%	15%
∞	42 d	2 h	F	77%	14%	99%	1%	77%	14%
∞	42 d	2 h	T	77%	13%	99%	1%	77%	14%
∞	42 d	4 h	F	77%	12%	99%	2%	77%	14%
∞	42 d	4 h	T	78%	13%	99%	2%	77%	14%
∞	42 d	8 h	F	78%	12%	98%	4%	77%	14%
∞	42 d	8 h	T	76%	14%	97%	5%	76%	15%
∞	42 d	16 h	F	78%	14%	96%	8%	77%	15%
∞	42 d	16 h	T	76%	13%	95%	8%	73%	18%
∞	28 d	60 m	F	76%	15%	100%	1%	76%	15%
∞	28 d	60 m	T	76%	14%	100%	1%	76%	15%
∞	28 d	2 h	F	77%	14%	99%	1%	77%	14%
∞	28 d	2 h	T	77%	13%	99%	1%	77%	14%
∞	28 d	4 h	F	77%	12%	99%	2%	77%	14%
∞	28 d	4 h	T	77%	13%	99%	2%	77%	14%
∞	28 d	8 h	F	78%	12%	98%	4%	77%	15%
∞	28 d	8 h	T	77%	14%	97%	5%	76%	15%
∞	28 d	16 h	F	78%	14%	96%	8%	77%	15%
∞	28 d	16 h	T	76%	13%	95%	8%	73%	17%
∞	21 d	60 m	F	76%	15%	100%	1%	76%	15%
∞	21 d	60 m	T	76%	14%	100%	1%	76%	15%
∞	21 d	2 h	F	77%	14%	99%	1%	77%	14%
∞	21 d	2 h	T	77%	14%	99%	1%	77%	14%
∞	21 d	4 h	F	77%	12%	99%	2%	77%	14%

Table D.30: $P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
∞	21 d	4 h	T	77%	14%	99%	2%	77%	14%
∞	21 d	8 h	F	78%	13%	98%	4%	77%	15%
∞	21 d	8 h	T	77%	14%	97%	5%	76%	15%
∞	21 d	16 h	F	78%	14%	96%	8%	77%	15%
∞	21 d	16 h	T	76%	14%	95%	8%	73%	17%
∞	14 d	60 m	F	76%	15%	100%	1%	76%	15%
∞	14 d	60 m	T	76%	14%	100%	1%	76%	14%
∞	14 d	2 h	F	77%	14%	99%	1%	77%	14%
∞	14 d	2 h	T	77%	13%	99%	1%	77%	14%
∞	14 d	4 h	F	77%	12%	99%	2%	77%	14%
∞	14 d	4 h	T	77%	14%	99%	2%	77%	14%
∞	14 d	8 h	F	78%	13%	98%	4%	77%	15%
∞	14 d	8 h	T	76%	14%	97%	5%	76%	14%
∞	14 d	16 h	F	78%	14%	96%	8%	76%	16%
∞	14 d	16 h	T	76%	15%	95%	8%	73%	18%
∞	7 d	60 m	F	77%	14%	100%	1%	77%	14%
∞	7 d	60 m	T	77%	14%	100%	1%	77%	14%
∞	7 d	2 h	F	77%	13%	99%	1%	77%	14%
∞	7 d	2 h	T	77%	12%	99%	1%	76%	14%
∞	7 d	4 h	F	77%	12%	99%	2%	77%	14%
∞	7 d	4 h	T	77%	13%	99%	2%	76%	14%
∞	7 d	8 h	F	77%	13%	98%	4%	76%	14%
∞	7 d	8 h	T	75%	16%	97%	5%	73%	17%
∞	7 d	16 h	F	77%	12%	96%	8%	76%	16%
∞	7 d	16 h	T	76%	15%	95%	8%	73%	19%
28 d	∞	60 m	F	76%	15%	100%	1%	76%	15%
28 d	∞	60 m	T	76%	14%	100%	1%	76%	15%
28 d	∞	2 h	F	77%	14%	99%	1%	77%	15%
28 d	∞	2 h	T	77%	14%	99%	1%	77%	14%
28 d	∞	4 h	F	77%	12%	99%	2%	77%	14%
28 d	∞	4 h	T	78%	13%	99%	2%	77%	14%
28 d	∞	8 h	F	78%	12%	98%	4%	77%	14%
28 d	∞	8 h	T	77%	14%	97%	5%	76%	15%
28 d	∞	16 h	F	78%	13%	96%	8%	77%	15%
28 d	∞	16 h	T	77%	14%	95%	8%	74%	16%

Table D.30 (Continued): $P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
28 d	42 d	60 m	F	76%	15%	100%	1%	76%	15%
28 d	42 d	60 m	T	76%	14%	100%	1%	76%	15%
28 d	42 d	2 h	F	77%	14%	99%	1%	76%	15%
28 d	42 d	2 h	T	77%	14%	99%	1%	77%	14%
28 d	42 d	4 h	F	77%	12%	99%	2%	77%	14%
28 d	42 d	4 h	T	78%	13%	99%	2%	77%	14%
28 d	42 d	8 h	F	78%	12%	98%	4%	77%	14%
28 d	42 d	8 h	T	76%	14%	97%	5%	75%	15%
28 d	42 d	16 h	F	78%	14%	96%	8%	77%	15%
28 d	42 d	16 h	T	76%	14%	95%	8%	74%	17%
28 d	28 d	60 m	F	76%	15%	100%	1%	76%	15%
28 d	28 d	60 m	T	76%	13%	100%	1%	76%	15%
28 d	28 d	2 h	F	76%	14%	99%	1%	76%	15%
28 d	28 d	2 h	T	77%	14%	99%	1%	76%	15%
28 d	28 d	4 h	F	77%	12%	99%	2%	77%	14%
28 d	28 d	4 h	T	77%	13%	99%	2%	76%	14%
28 d	28 d	8 h	F	78%	12%	98%	4%	77%	14%
28 d	28 d	8 h	T	76%	14%	97%	5%	76%	14%
28 d	28 d	16 h	F	78%	14%	96%	8%	77%	15%
28 d	28 d	16 h	T	76%	13%	95%	8%	74%	17%
28 d	21 d	60 m	F	76%	15%	100%	1%	76%	16%
28 d	21 d	60 m	T	76%	13%	100%	1%	76%	14%
28 d	21 d	2 h	F	76%	13%	99%	1%	76%	15%
28 d	21 d	2 h	T	77%	14%	99%	1%	76%	15%
28 d	21 d	4 h	F	77%	12%	99%	2%	77%	14%
28 d	21 d	4 h	T	77%	14%	99%	2%	76%	14%
28 d	21 d	8 h	F	78%	12%	98%	4%	77%	14%
28 d	21 d	8 h	T	76%	14%	97%	5%	75%	15%
28 d	21 d	16 h	F	78%	14%	96%	8%	76%	16%
28 d	21 d	16 h	T	76%	14%	95%	8%	74%	16%
28 d	14 d	60 m	F	76%	15%	100%	1%	76%	16%
28 d	14 d	60 m	T	76%	13%	100%	1%	76%	14%
28 d	14 d	2 h	F	76%	13%	99%	1%	76%	15%
28 d	14 d	2 h	T	77%	14%	99%	1%	76%	14%
28 d	14 d	4 h	F	77%	12%	99%	2%	77%	14%

Table D.30 (Continued): $P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
28 d	14 d	4 h	T	77%	14%	99%	2%	76%	14%
28 d	14 d	8 h	F	78%	13%	98%	4%	77%	14%
28 d	14 d	8 h	T	76%	14%	97%	5%	75%	15%
28 d	14 d	16 h	F	78%	13%	96%	8%	76%	16%
28 d	14 d	16 h	T	76%	15%	95%	8%	73%	18%
28 d	7 d	60 m	F	76%	15%	100%	1%	76%	15%
28 d	7 d	60 m	T	76%	14%	100%	1%	76%	15%
28 d	7 d	2 h	F	76%	14%	99%	1%	76%	15%
28 d	7 d	2 h	T	77%	12%	99%	1%	76%	14%
28 d	7 d	4 h	F	77%	12%	99%	2%	76%	14%
28 d	7 d	4 h	T	76%	13%	99%	2%	76%	14%
28 d	7 d	8 h	F	77%	12%	98%	4%	76%	14%
28 d	7 d	8 h	T	75%	16%	97%	5%	73%	16%
28 d	7 d	16 h	F	77%	12%	96%	8%	76%	16%
28 d	7 d	16 h	T	76%	16%	95%	8%	73%	19%
21 d	∞	60 m	F	76%	15%	100%	1%	76%	15%
21 d	∞	60 m	T	76%	13%	100%	1%	76%	14%
21 d	∞	2 h	F	76%	13%	99%	1%	76%	15%
21 d	∞	2 h	T	77%	14%	99%	1%	77%	14%
21 d	∞	4 h	F	77%	12%	99%	2%	77%	14%
21 d	∞	4 h	T	78%	13%	99%	2%	77%	14%
21 d	∞	8 h	F	78%	13%	98%	4%	77%	15%
21 d	∞	8 h	T	77%	14%	97%	5%	76%	15%
21 d	∞	16 h	F	78%	13%	96%	8%	77%	15%
21 d	∞	16 h	T	77%	14%	95%	8%	74%	16%
21 d	42 d	60 m	F	76%	15%	100%	1%	76%	15%
21 d	42 d	60 m	T	76%	13%	100%	1%	76%	14%
21 d	42 d	2 h	F	76%	13%	99%	1%	76%	15%
21 d	42 d	2 h	T	77%	14%	99%	1%	76%	14%
21 d	42 d	4 h	F	77%	13%	99%	2%	77%	14%
21 d	42 d	4 h	T	78%	13%	99%	2%	77%	14%
21 d	42 d	8 h	F	78%	13%	98%	4%	77%	15%
21 d	42 d	8 h	T	77%	14%	97%	5%	75%	14%
21 d	42 d	16 h	F	78%	14%	96%	8%	77%	15%
21 d	42 d	16 h	T	76%	13%	95%	8%	74%	16%

Table D.30 (Continued): $P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓ Tower	↓ Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
21 d	28 d	60 m	F	76%	15%	100%	1%	76%	15%
21 d	28 d	60 m	T	76%	12%	100%	1%	76%	14%
21 d	28 d	2 h	F	76%	13%	99%	1%	76%	15%
21 d	28 d	2 h	T	77%	14%	99%	1%	76%	14%
21 d	28 d	4 h	F	77%	13%	99%	2%	77%	14%
21 d	28 d	4 h	T	77%	13%	99%	2%	76%	14%
21 d	28 d	8 h	F	78%	13%	98%	4%	77%	14%
21 d	28 d	8 h	T	76%	14%	97%	5%	75%	14%
21 d	28 d	16 h	F	78%	14%	96%	8%	77%	15%
21 d	28 d	16 h	T	76%	13%	95%	8%	74%	16%
21 d	21 d	60 m	F	75%	15%	100%	1%	75%	16%
21 d	21 d	60 m	T	76%	13%	100%	1%	75%	14%
21 d	21 d	2 h	F	76%	13%	99%	1%	76%	15%
21 d	21 d	2 h	T	77%	14%	99%	1%	76%	15%
21 d	21 d	4 h	F	77%	13%	99%	2%	77%	14%
21 d	21 d	4 h	T	77%	13%	99%	2%	76%	14%
21 d	21 d	8 h	F	78%	13%	98%	4%	77%	14%
21 d	21 d	8 h	T	76%	14%	97%	5%	75%	14%
21 d	21 d	16 h	F	78%	14%	96%	8%	76%	16%
21 d	21 d	16 h	T	76%	13%	95%	8%	74%	16%
21 d	14 d	60 m	F	75%	15%	100%	1%	75%	16%
21 d	14 d	60 m	T	76%	13%	100%	1%	75%	14%
21 d	14 d	2 h	F	76%	13%	99%	1%	76%	15%
21 d	14 d	2 h	T	76%	13%	99%	1%	76%	14%
21 d	14 d	4 h	F	77%	12%	99%	2%	77%	14%
21 d	14 d	4 h	T	77%	13%	99%	2%	76%	14%
21 d	14 d	8 h	F	78%	13%	98%	4%	76%	15%
21 d	14 d	8 h	T	76%	14%	97%	5%	75%	14%
21 d	14 d	16 h	F	78%	12%	96%	8%	76%	16%
21 d	14 d	16 h	T	75%	15%	95%	8%	73%	19%
21 d	7 d	60 m	F	75%	14%	100%	1%	75%	15%
21 d	7 d	60 m	T	76%	13%	100%	1%	76%	14%
21 d	7 d	2 h	F	76%	13%	99%	1%	76%	15%
21 d	7 d	2 h	T	76%	13%	99%	1%	76%	14%
21 d	7 d	4 h	F	76%	13%	99%	2%	76%	14%

Table D.30 (Continued): $P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
21 d	7 d	4 h	T	76%	13%	99%	2%	76%	14%
21 d	7 d	8 h	F	77%	12%	98%	4%	76%	15%
21 d	7 d	8 h	T	75%	16%	97%	5%	73%	17%
21 d	7 d	16 h	F	76%	12%	96%	8%	76%	16%
21 d	7 d	16 h	T	75%	16%	95%	8%	73%	19%
14 d	∞	60 m	F	75%	15%	100%	1%	75%	15%
14 d	∞	60 m	T	76%	13%	100%	1%	76%	14%
14 d	∞	2 h	F	76%	13%	99%	1%	76%	15%
14 d	∞	2 h	T	77%	14%	99%	1%	76%	15%
14 d	∞	4 h	F	77%	12%	99%	2%	76%	14%
14 d	∞	4 h	T	77%	14%	99%	2%	77%	14%
14 d	∞	8 h	F	78%	14%	98%	4%	77%	14%
14 d	∞	8 h	T	76%	13%	97%	5%	75%	14%
14 d	∞	16 h	F	78%	14%	96%	8%	77%	15%
14 d	∞	16 h	T	76%	13%	95%	8%	74%	17%
14 d	42 d	60 m	F	75%	15%	100%	1%	75%	15%
14 d	42 d	60 m	T	76%	14%	100%	1%	76%	14%
14 d	42 d	2 h	F	76%	14%	99%	1%	76%	14%
14 d	42 d	2 h	T	77%	14%	99%	1%	76%	15%
14 d	42 d	4 h	F	77%	12%	99%	2%	76%	14%
14 d	42 d	4 h	T	77%	14%	99%	2%	77%	14%
14 d	42 d	8 h	F	78%	14%	98%	4%	77%	14%
14 d	42 d	8 h	T	76%	13%	97%	5%	74%	15%
14 d	42 d	16 h	F	78%	14%	96%	8%	77%	15%
14 d	42 d	16 h	T	76%	13%	95%	8%	74%	16%
14 d	28 d	60 m	F	75%	15%	100%	1%	75%	15%
14 d	28 d	60 m	T	76%	14%	100%	1%	75%	14%
14 d	28 d	2 h	F	76%	14%	99%	1%	76%	14%
14 d	28 d	2 h	T	77%	14%	99%	1%	76%	14%
14 d	28 d	4 h	F	77%	12%	99%	2%	76%	14%
14 d	28 d	4 h	T	77%	13%	99%	2%	76%	14%
14 d	28 d	8 h	F	78%	14%	98%	4%	77%	14%
14 d	28 d	8 h	T	76%	13%	97%	5%	74%	15%
14 d	28 d	16 h	F	78%	14%	96%	8%	76%	16%
14 d	28 d	16 h	T	76%	12%	95%	8%	74%	16%

Table D.30 (Continued): $P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
14 d	21 d	60 m	F	75%	15%	100%	1%	75%	15%
14 d	21 d	60 m	T	75%	14%	100%	1%	75%	14%
14 d	21 d	2 h	F	76%	14%	99%	1%	75%	14%
14 d	21 d	2 h	T	76%	14%	99%	1%	76%	15%
14 d	21 d	4 h	F	77%	12%	99%	2%	76%	14%
14 d	21 d	4 h	T	77%	13%	99%	2%	76%	14%
14 d	21 d	8 h	F	78%	14%	98%	4%	77%	14%
14 d	21 d	8 h	T	76%	13%	97%	5%	74%	15%
14 d	21 d	16 h	F	78%	14%	96%	8%	76%	16%
14 d	21 d	16 h	T	76%	13%	95%	8%	74%	16%
14 d	14 d	60 m	F	75%	14%	100%	1%	74%	14%
14 d	14 d	60 m	T	75%	14%	100%	1%	75%	14%
14 d	14 d	2 h	F	75%	14%	99%	1%	75%	14%
14 d	14 d	2 h	T	76%	13%	99%	1%	76%	14%
14 d	14 d	4 h	F	77%	13%	99%	2%	76%	14%
14 d	14 d	4 h	T	77%	13%	99%	2%	76%	14%
14 d	14 d	8 h	F	77%	13%	98%	4%	77%	14%
14 d	14 d	8 h	T	76%	13%	97%	5%	74%	15%
14 d	14 d	16 h	F	77%	13%	96%	8%	76%	16%
14 d	14 d	16 h	T	74%	16%	95%	8%	72%	18%
14 d	7 d	60 m	F	74%	14%	100%	1%	74%	14%
14 d	7 d	60 m	T	75%	14%	100%	1%	75%	14%
14 d	7 d	2 h	F	75%	14%	99%	1%	75%	14%
14 d	7 d	2 h	T	76%	13%	99%	1%	75%	14%
14 d	7 d	4 h	F	76%	13%	99%	2%	76%	14%
14 d	7 d	4 h	T	76%	13%	99%	2%	76%	13%
14 d	7 d	8 h	F	77%	13%	98%	4%	76%	15%
14 d	7 d	8 h	T	74%	16%	97%	5%	73%	16%
14 d	7 d	16 h	F	76%	12%	96%	8%	75%	16%
14 d	7 d	16 h	T	74%	16%	95%	8%	72%	20%
7 d	∞	60 m	F	74%	15%	100%	1%	74%	15%
7 d	∞	60 m	T	74%	15%	100%	1%	74%	15%
7 d	∞	2 h	F	74%	15%	99%	1%	74%	15%
7 d	∞	2 h	T	75%	14%	99%	1%	74%	14%
7 d	∞	4 h	F	75%	12%	99%	2%	75%	13%

Table D.30 (Continued): $P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
7 d	∞	4 h	T	76%	12%	99%	2%	75%	14%
7 d	∞	8 h	F	77%	11%	98%	4%	76%	14%
7 d	∞	8 h	T	75%	13%	97%	5%	75%	15%
7 d	∞	16 h	F	78%	12%	96%	8%	77%	15%
7 d	∞	16 h	T	76%	13%	95%	8%	74%	18%
7 d	42 d	60 m	F	74%	15%	100%	1%	74%	15%
7 d	42 d	60 m	T	74%	15%	100%	1%	74%	15%
7 d	42 d	2 h	F	75%	14%	99%	1%	74%	14%
7 d	42 d	2 h	T	75%	13%	99%	1%	75%	13%
7 d	42 d	4 h	F	76%	12%	99%	2%	76%	14%
7 d	42 d	4 h	T	76%	12%	99%	2%	75%	14%
7 d	42 d	8 h	F	77%	11%	98%	4%	76%	14%
7 d	42 d	8 h	T	75%	13%	97%	5%	75%	15%
7 d	42 d	16 h	F	78%	13%	96%	8%	77%	15%
7 d	42 d	16 h	T	76%	13%	95%	8%	74%	16%
7 d	28 d	60 m	F	74%	15%	100%	1%	74%	15%
7 d	28 d	60 m	T	74%	15%	100%	1%	74%	15%
7 d	28 d	2 h	F	75%	14%	99%	1%	74%	14%
7 d	28 d	2 h	T	75%	13%	99%	1%	75%	13%
7 d	28 d	4 h	F	76%	12%	99%	2%	76%	14%
7 d	28 d	4 h	T	76%	13%	99%	2%	75%	14%
7 d	28 d	8 h	F	77%	11%	98%	4%	76%	14%
7 d	28 d	8 h	T	75%	13%	97%	5%	75%	15%
7 d	28 d	16 h	F	78%	14%	96%	8%	77%	15%
7 d	28 d	16 h	T	76%	13%	95%	8%	74%	16%
7 d	21 d	60 m	F	74%	16%	100%	1%	74%	16%
7 d	21 d	60 m	T	74%	15%	100%	1%	74%	15%
7 d	21 d	2 h	F	75%	14%	99%	1%	74%	14%
7 d	21 d	2 h	T	75%	13%	99%	1%	75%	13%
7 d	21 d	4 h	F	76%	12%	99%	2%	75%	14%
7 d	21 d	4 h	T	76%	13%	99%	2%	75%	14%
7 d	21 d	8 h	F	77%	12%	98%	4%	76%	14%
7 d	21 d	8 h	T	75%	13%	97%	5%	75%	15%
7 d	21 d	16 h	F	78%	14%	96%	8%	77%	15%
7 d	21 d	16 h	T	76%	13%	95%	8%	73%	16%

Table D.30 (Continued): $P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓ Tower	↓ Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
7 d	14 d	60 m	F	73%	16%	100%	1%	73%	16%
7 d	14 d	60 m	T	74%	16%	100%	1%	74%	16%
7 d	14 d	2 h	F	74%	15%	99%	1%	74%	15%
7 d	14 d	2 h	T	75%	14%	99%	1%	74%	14%
7 d	14 d	4 h	F	76%	12%	99%	2%	75%	15%
7 d	14 d	4 h	T	76%	13%	99%	2%	75%	14%
7 d	14 d	8 h	F	77%	12%	98%	4%	76%	15%
7 d	14 d	8 h	T	75%	13%	97%	5%	75%	15%
7 d	14 d	16 h	F	78%	13%	96%	8%	76%	16%
7 d	14 d	16 h	T	74%	16%	95%	8%	73%	21%
7 d	7 d	60 m	F	74%	16%	100%	1%	74%	16%
7 d	7 d	60 m	T	74%	16%	100%	1%	74%	16%
7 d	7 d	2 h	F	74%	15%	99%	1%	74%	15%
7 d	7 d	2 h	T	75%	14%	99%	1%	74%	14%
7 d	7 d	4 h	F	75%	13%	99%	2%	75%	14%
7 d	7 d	4 h	T	76%	13%	98%	2%	75%	14%
7 d	7 d	8 h	F	76%	11%	97%	4%	75%	15%
7 d	7 d	8 h	T	73%	14%	97%	5%	72%	15%
7 d	7 d	16 h	F	76%	13%	95%	8%	75%	17%
7 d	7 d	16 h	T	74%	16%	95%	8%	73%	21%
4 d	∞	60 m	F	70%	16%	100%	1%	70%	16%
4 d	∞	60 m	T	71%	15%	100%	1%	71%	15%
4 d	∞	2 h	F	72%	15%	99%	1%	72%	16%
4 d	∞	2 h	T	73%	13%	99%	1%	72%	14%
4 d	∞	4 h	F	74%	12%	99%	2%	74%	13%
4 d	∞	4 h	T	75%	11%	99%	2%	74%	12%
4 d	∞	8 h	F	77%	11%	98%	4%	76%	13%
4 d	∞	8 h	T	76%	13%	97%	5%	76%	14%
4 d	∞	16 h	F	79%	12%	96%	8%	77%	14%
4 d	∞	16 h	T	77%	14%	95%	8%	74%	18%
4 d	42 d	60 m	F	70%	16%	100%	1%	70%	16%
4 d	42 d	60 m	T	71%	15%	100%	1%	71%	15%
4 d	42 d	2 h	F	72%	15%	99%	1%	72%	16%
4 d	42 d	2 h	T	73%	13%	99%	1%	72%	14%
4 d	42 d	4 h	F	74%	12%	99%	2%	74%	13%

Table D.30 (Continued): $P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
4 d	42 d	4 h	T	74%	12%	99%	2%	74%	12%
4 d	42 d	8 h	F	77%	11%	98%	4%	76%	13%
4 d	42 d	8 h	T	75%	13%	97%	5%	75%	14%
4 d	42 d	16 h	F	78%	13%	96%	8%	77%	15%
4 d	42 d	16 h	T	76%	14%	95%	8%	74%	16%
4 d	28 d	60 m	F	70%	16%	100%	1%	70%	16%
4 d	28 d	60 m	T	71%	15%	100%	1%	71%	15%
4 d	28 d	2 h	F	72%	15%	99%	1%	72%	15%
4 d	28 d	2 h	T	73%	13%	99%	1%	72%	14%
4 d	28 d	4 h	F	74%	12%	99%	2%	74%	12%
4 d	28 d	4 h	T	75%	12%	99%	2%	74%	13%
4 d	28 d	8 h	F	77%	11%	98%	4%	76%	13%
4 d	28 d	8 h	T	75%	13%	97%	5%	75%	14%
4 d	28 d	16 h	F	78%	13%	96%	8%	77%	15%
4 d	28 d	16 h	T	77%	14%	95%	8%	74%	17%
4 d	21 d	60 m	F	70%	16%	100%	1%	70%	16%
4 d	21 d	60 m	T	71%	15%	100%	1%	71%	15%
4 d	21 d	2 h	F	72%	15%	99%	1%	72%	16%
4 d	21 d	2 h	T	73%	13%	99%	1%	72%	14%
4 d	21 d	4 h	F	74%	12%	99%	2%	74%	12%
4 d	21 d	4 h	T	75%	12%	99%	2%	74%	13%
4 d	21 d	8 h	F	76%	11%	98%	4%	76%	14%
4 d	21 d	8 h	T	75%	14%	97%	5%	75%	14%
4 d	21 d	16 h	F	78%	13%	96%	8%	76%	15%
4 d	21 d	16 h	T	76%	14%	95%	8%	74%	17%
4 d	14 d	60 m	F	70%	16%	100%	1%	70%	16%
4 d	14 d	60 m	T	70%	15%	100%	1%	70%	16%
4 d	14 d	2 h	F	72%	15%	99%	1%	71%	15%
4 d	14 d	2 h	T	73%	13%	99%	1%	72%	14%
4 d	14 d	4 h	F	74%	12%	99%	2%	74%	13%
4 d	14 d	4 h	T	74%	11%	99%	2%	74%	13%
4 d	14 d	8 h	F	76%	12%	98%	4%	76%	14%
4 d	14 d	8 h	T	76%	13%	97%	5%	75%	14%
4 d	14 d	16 h	F	78%	13%	96%	8%	76%	15%
4 d	14 d	16 h	T	75%	17%	95%	8%	73%	20%

Table D.30 (Continued): $P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓ Tower	↓ Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
4 d	7 d	60 m	F	70%	12%	100%	1%	70%	14%
4 d	7 d	60 m	T	70%	15%	100%	1%	70%	15%
4 d	7 d	2 h	F	71%	12%	99%	1%	70%	15%
4 d	7 d	2 h	T	72%	13%	99%	1%	72%	13%
4 d	7 d	4 h	F	73%	11%	99%	2%	73%	12%
4 d	7 d	4 h	T	74%	12%	98%	2%	73%	13%
4 d	7 d	8 h	F	75%	13%	97%	4%	75%	14%
4 d	7 d	8 h	T	73%	15%	97%	5%	72%	17%
4 d	7 d	16 h	F	76%	13%	95%	8%	75%	17%
4 d	7 d	16 h	T	75%	17%	95%	8%	73%	21%

Table D.30 (Continued): $P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
∞	∞	60 m	F	77%	12%	100%	1%	77%	12%
∞	∞	60 m	T	78%	12%	100%	1%	77%	12%
∞	∞	2 h	F	78%	12%	99%	1%	78%	12%
∞	∞	2 h	T	78%	13%	99%	1%	78%	13%
∞	∞	4 h	F	79%	13%	99%	2%	79%	12%
∞	∞	4 h	T	78%	12%	99%	2%	78%	13%
∞	∞	8 h	F	79%	12%	98%	4%	78%	13%
∞	∞	8 h	T	78%	13%	97%	5%	77%	16%
∞	∞	16 h	F	79%	12%	96%	8%	78%	14%
∞	∞	16 h	T	76%	14%	95%	8%	74%	17%
∞	42 d	60 m	F	77%	12%	100%	1%	77%	12%
∞	42 d	60 m	T	78%	12%	100%	1%	78%	12%
∞	42 d	2 h	F	78%	12%	99%	1%	78%	12%
∞	42 d	2 h	T	78%	13%	99%	1%	78%	13%
∞	42 d	4 h	F	79%	12%	99%	2%	79%	13%
∞	42 d	4 h	T	78%	12%	99%	2%	78%	13%
∞	42 d	8 h	F	79%	12%	98%	4%	78%	13%
∞	42 d	8 h	T	78%	13%	97%	5%	76%	16%
∞	42 d	16 h	F	79%	12%	96%	8%	78%	14%
∞	42 d	16 h	T	76%	13%	95%	8%	74%	16%
∞	28 d	60 m	F	77%	13%	100%	1%	77%	13%
∞	28 d	60 m	T	77%	12%	100%	1%	77%	12%
∞	28 d	2 h	F	78%	12%	99%	1%	78%	12%
∞	28 d	2 h	T	78%	13%	99%	1%	78%	13%
∞	28 d	4 h	F	79%	13%	99%	2%	79%	13%
∞	28 d	4 h	T	78%	12%	99%	2%	78%	13%
∞	28 d	8 h	F	79%	12%	98%	4%	78%	13%
∞	28 d	8 h	T	78%	14%	97%	5%	76%	15%
∞	28 d	16 h	F	78%	13%	96%	8%	78%	14%
∞	28 d	16 h	T	76%	13%	95%	8%	74%	16%
∞	21 d	60 m	F	77%	13%	100%	1%	77%	13%
∞	21 d	60 m	T	77%	12%	100%	1%	77%	12%
∞	21 d	2 h	F	78%	12%	99%	1%	78%	12%
∞	21 d	2 h	T	78%	13%	99%	1%	78%	13%
∞	21 d	4 h	F	79%	13%	99%	2%	79%	13%

Table D.31: $P(t \mid h, d, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
∞	21 d	4 h	T	78%	12%	99%	2%	78%	13%
∞	21 d	8 h	F	79%	12%	98%	4%	78%	13%
∞	21 d	8 h	T	77%	14%	97%	5%	77%	15%
∞	21 d	16 h	F	78%	12%	96%	8%	77%	14%
∞	21 d	16 h	T	76%	14%	95%	8%	74%	16%
∞	14 d	60 m	F	76%	13%	100%	1%	76%	13%
∞	14 d	60 m	T	77%	12%	100%	1%	77%	12%
∞	14 d	2 h	F	78%	12%	99%	1%	78%	12%
∞	14 d	2 h	T	78%	13%	99%	1%	78%	13%
∞	14 d	4 h	F	79%	13%	99%	2%	78%	13%
∞	14 d	4 h	T	78%	13%	99%	2%	78%	13%
∞	14 d	8 h	F	79%	12%	98%	4%	78%	14%
∞	14 d	8 h	T	77%	14%	97%	5%	77%	15%
∞	14 d	16 h	F	78%	12%	96%	8%	77%	15%
∞	14 d	16 h	T	76%	14%	95%	8%	73%	18%
∞	7 d	60 m	F	76%	13%	100%	1%	76%	13%
∞	7 d	60 m	T	77%	12%	100%	1%	77%	12%
∞	7 d	2 h	F	78%	12%	99%	1%	77%	12%
∞	7 d	2 h	T	78%	13%	99%	1%	78%	13%
∞	7 d	4 h	F	78%	13%	99%	2%	78%	14%
∞	7 d	4 h	T	78%	12%	99%	2%	78%	13%
∞	7 d	8 h	F	78%	12%	98%	4%	78%	13%
∞	7 d	8 h	T	76%	16%	97%	5%	74%	17%
∞	7 d	16 h	F	78%	14%	96%	8%	77%	15%
∞	7 d	16 h	T	77%	14%	95%	8%	73%	18%
28 d	∞	60 m	F	76%	12%	100%	1%	76%	12%
28 d	∞	60 m	T	76%	12%	100%	1%	76%	12%
28 d	∞	2 h	F	77%	12%	99%	1%	77%	12%
28 d	∞	2 h	T	78%	14%	99%	1%	77%	14%
28 d	∞	4 h	F	79%	12%	99%	2%	78%	13%
28 d	∞	4 h	T	78%	12%	99%	2%	78%	14%
28 d	∞	8 h	F	79%	12%	98%	4%	78%	13%
28 d	∞	8 h	T	77%	14%	97%	5%	76%	15%
28 d	∞	16 h	F	78%	13%	96%	8%	78%	14%
28 d	∞	16 h	T	77%	14%	95%	8%	74%	16%

Table D.31 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
28 d	42 d	60 m	F	76%	12%	100%	1%	76%	12%
28 d	42 d	60 m	T	76%	12%	100%	1%	76%	12%
28 d	42 d	2 h	F	77%	12%	99%	1%	77%	12%
28 d	42 d	2 h	T	78%	13%	99%	1%	77%	14%
28 d	42 d	4 h	F	79%	12%	99%	2%	78%	13%
28 d	42 d	4 h	T	78%	12%	99%	2%	78%	14%
28 d	42 d	8 h	F	79%	12%	98%	4%	78%	13%
28 d	42 d	8 h	T	76%	14%	97%	5%	75%	14%
28 d	42 d	16 h	F	78%	13%	96%	8%	77%	14%
28 d	42 d	16 h	T	76%	14%	95%	8%	74%	17%
28 d	28 d	60 m	F	76%	12%	100%	1%	76%	12%
28 d	28 d	60 m	T	76%	12%	100%	1%	76%	12%
28 d	28 d	2 h	F	77%	12%	99%	1%	76%	12%
28 d	28 d	2 h	T	78%	14%	99%	1%	77%	14%
28 d	28 d	4 h	F	79%	12%	99%	2%	78%	13%
28 d	28 d	4 h	T	78%	12%	99%	2%	78%	14%
28 d	28 d	8 h	F	79%	12%	98%	4%	78%	13%
28 d	28 d	8 h	T	76%	14%	97%	5%	75%	14%
28 d	28 d	16 h	F	78%	14%	96%	8%	77%	15%
28 d	28 d	16 h	T	76%	13%	95%	8%	74%	17%
28 d	21 d	60 m	F	76%	12%	100%	1%	76%	12%
28 d	21 d	60 m	T	76%	12%	100%	1%	76%	12%
28 d	21 d	2 h	F	77%	12%	99%	1%	76%	12%
28 d	21 d	2 h	T	77%	14%	99%	1%	77%	14%
28 d	21 d	4 h	F	79%	12%	99%	2%	78%	13%
28 d	21 d	4 h	T	78%	12%	99%	2%	77%	14%
28 d	21 d	8 h	F	79%	12%	98%	4%	78%	13%
28 d	21 d	8 h	T	76%	14%	97%	5%	75%	14%
28 d	21 d	16 h	F	78%	14%	96%	8%	77%	15%
28 d	21 d	16 h	T	76%	14%	95%	8%	74%	16%
28 d	14 d	60 m	F	75%	12%	100%	1%	75%	12%
28 d	14 d	60 m	T	76%	13%	100%	1%	76%	12%
28 d	14 d	2 h	F	76%	13%	99%	1%	76%	13%
28 d	14 d	2 h	T	77%	14%	99%	1%	77%	14%
28 d	14 d	4 h	F	79%	13%	99%	2%	78%	13%

Table D.31 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
28 d	14 d	4 h	T	78%	12%	99%	2%	77%	14%
28 d	14 d	8 h	F	79%	12%	98%	4%	78%	14%
28 d	14 d	8 h	T	76%	13%	97%	5%	75%	14%
28 d	14 d	16 h	F	78%	13%	96%	8%	77%	15%
28 d	14 d	16 h	T	76%	15%	95%	8%	73%	18%
28 d	7 d	60 m	F	76%	12%	100%	1%	76%	12%
28 d	7 d	60 m	T	76%	12%	100%	1%	76%	12%
28 d	7 d	2 h	F	76%	13%	99%	1%	76%	13%
28 d	7 d	2 h	T	77%	13%	99%	1%	77%	14%
28 d	7 d	4 h	F	78%	13%	99%	2%	78%	13%
28 d	7 d	4 h	T	77%	13%	99%	2%	77%	14%
28 d	7 d	8 h	F	78%	13%	98%	4%	77%	14%
28 d	7 d	8 h	T	75%	16%	97%	5%	73%	16%
28 d	7 d	16 h	F	77%	13%	96%	8%	76%	16%
28 d	7 d	16 h	T	76%	16%	95%	8%	73%	19%
21 d	∞	60 m	F	75%	12%	100%	1%	75%	12%
21 d	∞	60 m	T	76%	12%	100%	1%	76%	12%
21 d	∞	2 h	F	77%	12%	99%	1%	76%	12%
21 d	∞	2 h	T	77%	13%	99%	1%	77%	14%
21 d	∞	4 h	F	79%	13%	99%	2%	78%	13%
21 d	∞	4 h	T	77%	13%	99%	2%	77%	14%
21 d	∞	8 h	F	78%	12%	98%	4%	78%	14%
21 d	∞	8 h	T	77%	14%	97%	5%	76%	15%
21 d	∞	16 h	F	78%	13%	96%	8%	77%	14%
21 d	∞	16 h	T	77%	14%	95%	8%	74%	16%
21 d	42 d	60 m	F	75%	12%	100%	1%	75%	12%
21 d	42 d	60 m	T	76%	12%	100%	1%	76%	12%
21 d	42 d	2 h	F	77%	12%	99%	1%	76%	12%
21 d	42 d	2 h	T	77%	13%	99%	1%	77%	14%
21 d	42 d	4 h	F	79%	13%	99%	2%	78%	13%
21 d	42 d	4 h	T	77%	13%	99%	2%	77%	14%
21 d	42 d	8 h	F	78%	12%	98%	4%	78%	13%
21 d	42 d	8 h	T	77%	14%	97%	5%	75%	14%
21 d	42 d	16 h	F	78%	14%	96%	8%	77%	15%
21 d	42 d	16 h	T	76%	13%	95%	8%	74%	16%

Table D.31 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
21 d	28 d	60 m	F	75%	12%	100%	1%	75%	12%
21 d	28 d	60 m	T	76%	12%	100%	1%	76%	12%
21 d	28 d	2 h	F	76%	13%	99%	1%	76%	13%
21 d	28 d	2 h	T	77%	13%	99%	1%	77%	14%
21 d	28 d	4 h	F	79%	13%	99%	2%	78%	13%
21 d	28 d	4 h	T	77%	13%	99%	2%	77%	14%
21 d	28 d	8 h	F	78%	13%	98%	4%	78%	13%
21 d	28 d	8 h	T	76%	14%	97%	5%	75%	14%
21 d	28 d	16 h	F	78%	14%	96%	8%	77%	15%
21 d	28 d	16 h	T	76%	13%	95%	8%	74%	16%
21 d	21 d	60 m	F	75%	12%	100%	1%	75%	12%
21 d	21 d	60 m	T	76%	12%	100%	1%	76%	12%
21 d	21 d	2 h	F	76%	13%	99%	1%	76%	13%
21 d	21 d	2 h	T	77%	14%	99%	1%	77%	14%
21 d	21 d	4 h	F	79%	13%	99%	2%	78%	13%
21 d	21 d	4 h	T	77%	13%	99%	2%	77%	14%
21 d	21 d	8 h	F	79%	13%	98%	4%	78%	14%
21 d	21 d	8 h	T	76%	14%	97%	5%	75%	14%
21 d	21 d	16 h	F	78%	14%	96%	8%	77%	15%
21 d	21 d	16 h	T	76%	13%	95%	8%	74%	16%
21 d	14 d	60 m	F	75%	12%	100%	1%	75%	12%
21 d	14 d	60 m	T	75%	12%	100%	1%	75%	12%
21 d	14 d	2 h	F	76%	13%	99%	1%	76%	14%
21 d	14 d	2 h	T	77%	14%	99%	1%	77%	14%
21 d	14 d	4 h	F	78%	13%	99%	2%	78%	13%
21 d	14 d	4 h	T	77%	13%	99%	2%	77%	14%
21 d	14 d	8 h	F	79%	12%	98%	4%	78%	14%
21 d	14 d	8 h	T	76%	14%	97%	5%	75%	14%
21 d	14 d	16 h	F	78%	14%	96%	8%	76%	16%
21 d	14 d	16 h	T	75%	15%	95%	8%	73%	19%
21 d	7 d	60 m	F	75%	12%	100%	1%	75%	12%
21 d	7 d	60 m	T	75%	12%	100%	1%	75%	13%
21 d	7 d	2 h	F	76%	13%	99%	1%	76%	14%
21 d	7 d	2 h	T	77%	12%	99%	1%	76%	14%
21 d	7 d	4 h	F	78%	13%	99%	2%	78%	13%

Table D.31 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
21 d	7 d	4 h	T	77%	13%	99%	2%	76%	14%
21 d	7 d	8 h	F	78%	13%	98%	4%	77%	14%
21 d	7 d	8 h	T	75%	16%	97%	5%	73%	17%
21 d	7 d	16 h	F	77%	13%	96%	8%	76%	16%
21 d	7 d	16 h	T	75%	16%	95%	8%	73%	19%
14 d	∞	60 m	F	74%	12%	100%	1%	74%	12%
14 d	∞	60 m	T	75%	12%	100%	1%	75%	12%
14 d	∞	2 h	F	75%	13%	99%	1%	75%	14%
14 d	∞	2 h	T	77%	13%	99%	1%	76%	14%
14 d	∞	4 h	F	77%	12%	99%	2%	77%	14%
14 d	∞	4 h	T	77%	12%	99%	2%	77%	14%
14 d	∞	8 h	F	78%	13%	98%	4%	78%	14%
14 d	∞	8 h	T	76%	13%	97%	5%	75%	14%
14 d	∞	16 h	F	78%	13%	96%	8%	77%	15%
14 d	∞	16 h	T	76%	13%	95%	8%	74%	17%
14 d	42 d	60 m	F	74%	12%	100%	1%	74%	12%
14 d	42 d	60 m	T	75%	12%	100%	1%	75%	12%
14 d	42 d	2 h	F	75%	13%	99%	1%	75%	14%
14 d	42 d	2 h	T	76%	13%	99%	1%	76%	14%
14 d	42 d	4 h	F	77%	12%	99%	2%	77%	13%
14 d	42 d	4 h	T	77%	12%	99%	2%	77%	14%
14 d	42 d	8 h	F	78%	13%	98%	4%	78%	14%
14 d	42 d	8 h	T	76%	13%	97%	5%	74%	15%
14 d	42 d	16 h	F	78%	14%	96%	8%	77%	15%
14 d	42 d	16 h	T	76%	13%	95%	8%	74%	16%
14 d	28 d	60 m	F	74%	12%	100%	1%	74%	12%
14 d	28 d	60 m	T	75%	12%	100%	1%	75%	13%
14 d	28 d	2 h	F	75%	12%	99%	1%	75%	14%
14 d	28 d	2 h	T	76%	13%	99%	1%	76%	14%
14 d	28 d	4 h	F	77%	12%	99%	2%	77%	13%
14 d	28 d	4 h	T	77%	13%	99%	2%	77%	14%
14 d	28 d	8 h	F	78%	13%	98%	4%	78%	14%
14 d	28 d	8 h	T	76%	13%	97%	5%	74%	15%
14 d	28 d	16 h	F	78%	14%	96%	8%	77%	15%
14 d	28 d	16 h	T	76%	12%	95%	8%	74%	16%

Table D.31 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
14 d	21 d	60 m	F	74%	12%	100%	1%	74%	12%
14 d	21 d	60 m	T	75%	12%	100%	1%	74%	13%
14 d	21 d	2 h	F	75%	13%	99%	1%	75%	14%
14 d	21 d	2 h	T	76%	13%	99%	1%	76%	14%
14 d	21 d	4 h	F	77%	12%	99%	2%	77%	14%
14 d	21 d	4 h	T	77%	13%	99%	2%	76%	14%
14 d	21 d	8 h	F	78%	13%	98%	4%	78%	14%
14 d	21 d	8 h	T	76%	13%	97%	5%	74%	15%
14 d	21 d	16 h	F	78%	14%	96%	8%	77%	15%
14 d	21 d	16 h	T	76%	13%	95%	8%	74%	16%
14 d	14 d	60 m	F	73%	13%	100%	1%	73%	13%
14 d	14 d	60 m	T	74%	13%	100%	1%	74%	13%
14 d	14 d	2 h	F	75%	13%	99%	1%	75%	14%
14 d	14 d	2 h	T	76%	13%	99%	1%	76%	13%
14 d	14 d	4 h	F	77%	12%	99%	2%	77%	14%
14 d	14 d	4 h	T	77%	13%	99%	2%	76%	14%
14 d	14 d	8 h	F	78%	13%	98%	4%	77%	14%
14 d	14 d	8 h	T	76%	13%	97%	5%	74%	15%
14 d	14 d	16 h	F	78%	14%	96%	8%	76%	16%
14 d	14 d	16 h	T	74%	16%	95%	8%	72%	18%
14 d	7 d	60 m	F	74%	12%	100%	1%	73%	13%
14 d	7 d	60 m	T	74%	12%	100%	1%	74%	13%
14 d	7 d	2 h	F	75%	13%	99%	1%	75%	14%
14 d	7 d	2 h	T	76%	12%	99%	1%	76%	14%
14 d	7 d	4 h	F	77%	12%	99%	2%	77%	14%
14 d	7 d	4 h	T	77%	13%	99%	2%	76%	14%
14 d	7 d	8 h	F	78%	12%	98%	4%	76%	15%
14 d	7 d	8 h	T	74%	16%	97%	5%	73%	16%
14 d	7 d	16 h	F	76%	13%	96%	8%	75%	16%
14 d	7 d	16 h	T	74%	16%	95%	8%	72%	20%
7 d	∞	60 m	F	70%	12%	100%	1%	70%	12%
7 d	∞	60 m	T	72%	11%	100%	1%	72%	11%
7 d	∞	2 h	F	72%	11%	99%	1%	72%	11%
7 d	∞	2 h	T	74%	12%	99%	1%	74%	12%
7 d	∞	4 h	F	75%	11%	99%	2%	75%	13%

Table D.31 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
7 d	∞	4 h	T	76%	12%	99%	2%	75%	14%
7 d	∞	8 h	F	77%	11%	98%	4%	76%	14%
7 d	∞	8 h	T	75%	13%	97%	5%	75%	15%
7 d	∞	16 h	F	78%	11%	96%	8%	77%	15%
7 d	∞	16 h	T	76%	13%	95%	8%	74%	18%
7 d	42 d	60 m	F	70%	12%	100%	1%	70%	12%
7 d	42 d	60 m	T	72%	11%	100%	1%	72%	11%
7 d	42 d	2 h	F	72%	11%	99%	1%	72%	11%
7 d	42 d	2 h	T	74%	12%	99%	1%	74%	12%
7 d	42 d	4 h	F	75%	11%	99%	2%	75%	14%
7 d	42 d	4 h	T	76%	12%	99%	2%	75%	13%
7 d	42 d	8 h	F	77%	11%	98%	4%	76%	13%
7 d	42 d	8 h	T	75%	13%	97%	5%	75%	15%
7 d	42 d	16 h	F	78%	13%	96%	8%	77%	15%
7 d	42 d	16 h	T	76%	13%	95%	8%	74%	16%
7 d	28 d	60 m	F	70%	13%	100%	1%	70%	13%
7 d	28 d	60 m	T	72%	11%	100%	1%	72%	11%
7 d	28 d	2 h	F	72%	12%	99%	1%	72%	12%
7 d	28 d	2 h	T	74%	12%	99%	1%	74%	12%
7 d	28 d	4 h	F	75%	11%	99%	2%	75%	13%
7 d	28 d	4 h	T	76%	13%	99%	2%	75%	14%
7 d	28 d	8 h	F	77%	11%	98%	4%	76%	13%
7 d	28 d	8 h	T	75%	13%	97%	5%	75%	15%
7 d	28 d	16 h	F	78%	13%	96%	8%	77%	15%
7 d	28 d	16 h	T	76%	13%	95%	8%	74%	16%
7 d	21 d	60 m	F	70%	13%	100%	1%	70%	13%
7 d	21 d	60 m	T	72%	11%	100%	1%	72%	11%
7 d	21 d	2 h	F	72%	12%	99%	1%	72%	12%
7 d	21 d	2 h	T	74%	12%	99%	1%	74%	12%
7 d	21 d	4 h	F	75%	11%	99%	2%	75%	13%
7 d	21 d	4 h	T	75%	13%	99%	2%	75%	14%
7 d	21 d	8 h	F	77%	12%	98%	4%	76%	14%
7 d	21 d	8 h	T	75%	13%	97%	5%	75%	15%
7 d	21 d	16 h	F	78%	13%	96%	8%	77%	15%
7 d	21 d	16 h	T	76%	13%	95%	8%	73%	16%

Table D.31 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
7 d	14 d	60 m	F	70%	12%	100%	1%	70%	14%
7 d	14 d	60 m	T	72%	11%	100%	1%	72%	11%
7 d	14 d	2 h	F	72%	13%	99%	1%	72%	13%
7 d	14 d	2 h	T	74%	12%	99%	1%	74%	13%
7 d	14 d	4 h	F	75%	12%	99%	2%	75%	13%
7 d	14 d	4 h	T	75%	13%	99%	2%	75%	14%
7 d	14 d	8 h	F	77%	12%	98%	4%	76%	15%
7 d	14 d	8 h	T	75%	13%	97%	5%	75%	16%
7 d	14 d	16 h	F	78%	13%	96%	8%	76%	16%
7 d	14 d	16 h	T	74%	16%	95%	8%	73%	21%
7 d	7 d	60 m	F	70%	12%	100%	1%	70%	12%
7 d	7 d	60 m	T	72%	11%	100%	1%	72%	11%
7 d	7 d	2 h	F	72%	11%	99%	1%	72%	12%
7 d	7 d	2 h	T	74%	12%	99%	1%	74%	13%
7 d	7 d	4 h	F	75%	11%	99%	2%	75%	13%
7 d	7 d	4 h	T	75%	13%	98%	2%	75%	14%
7 d	7 d	8 h	F	76%	11%	97%	4%	75%	14%
7 d	7 d	8 h	T	73%	14%	97%	5%	72%	15%
7 d	7 d	16 h	F	76%	13%	95%	8%	75%	17%
7 d	7 d	16 h	T	74%	16%	95%	8%	73%	21%
4 d	∞	60 m	F	63%	10%	100%	1%	63%	10%
4 d	∞	60 m	T	68%	11%	100%	1%	68%	11%
4 d	∞	2 h	F	67%	11%	99%	1%	67%	12%
4 d	∞	2 h	T	72%	12%	99%	1%	71%	12%
4 d	∞	4 h	F	72%	12%	99%	2%	71%	13%
4 d	∞	4 h	T	74%	12%	99%	2%	74%	12%
4 d	∞	8 h	F	75%	11%	98%	4%	74%	12%
4 d	∞	8 h	T	76%	13%	97%	5%	76%	14%
4 d	∞	16 h	F	78%	11%	96%	8%	77%	14%
4 d	∞	16 h	T	77%	14%	95%	8%	74%	18%
4 d	42 d	60 m	F	63%	10%	100%	1%	63%	10%
4 d	42 d	60 m	T	68%	11%	100%	1%	68%	11%
4 d	42 d	2 h	F	67%	11%	99%	1%	67%	11%
4 d	42 d	2 h	T	72%	12%	99%	1%	71%	13%
4 d	42 d	4 h	F	72%	12%	99%	2%	71%	13%

Table D.31 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
4 d	42 d	4 h	T	74%	12%	99%	2%	74%	12%
4 d	42 d	8 h	F	75%	11%	98%	4%	74%	12%
4 d	42 d	8 h	T	75%	13%	97%	5%	75%	14%
4 d	42 d	16 h	F	78%	12%	96%	8%	77%	15%
4 d	42 d	16 h	T	76%	14%	95%	8%	74%	16%
4 d	28 d	60 m	F	63%	10%	100%	1%	63%	10%
4 d	28 d	60 m	T	68%	11%	100%	1%	68%	12%
4 d	28 d	2 h	F	67%	11%	99%	1%	67%	12%
4 d	28 d	2 h	T	71%	12%	99%	1%	71%	12%
4 d	28 d	4 h	F	72%	12%	99%	2%	71%	13%
4 d	28 d	4 h	T	74%	12%	99%	2%	74%	13%
4 d	28 d	8 h	F	75%	10%	98%	4%	74%	12%
4 d	28 d	8 h	T	75%	13%	97%	5%	75%	14%
4 d	28 d	16 h	F	78%	13%	96%	8%	77%	15%
4 d	28 d	16 h	T	77%	14%	95%	8%	74%	17%
4 d	21 d	60 m	F	63%	10%	100%	1%	63%	10%
4 d	21 d	60 m	T	68%	11%	100%	1%	68%	12%
4 d	21 d	2 h	F	67%	11%	99%	1%	67%	12%
4 d	21 d	2 h	T	71%	12%	99%	1%	71%	13%
4 d	21 d	4 h	F	72%	12%	99%	2%	71%	13%
4 d	21 d	4 h	T	74%	12%	99%	2%	74%	13%
4 d	21 d	8 h	F	74%	11%	98%	4%	74%	12%
4 d	21 d	8 h	T	75%	14%	97%	5%	75%	14%
4 d	21 d	16 h	F	78%	13%	96%	8%	76%	16%
4 d	21 d	16 h	T	76%	14%	95%	8%	74%	17%
4 d	14 d	60 m	F	63%	11%	100%	1%	63%	11%
4 d	14 d	60 m	T	68%	11%	100%	1%	67%	12%
4 d	14 d	2 h	F	67%	12%	99%	1%	66%	12%
4 d	14 d	2 h	T	71%	13%	99%	1%	71%	12%
4 d	14 d	4 h	F	72%	12%	99%	2%	71%	13%
4 d	14 d	4 h	T	74%	11%	99%	2%	74%	13%
4 d	14 d	8 h	F	74%	12%	98%	4%	74%	12%
4 d	14 d	8 h	T	76%	13%	97%	5%	75%	14%
4 d	14 d	16 h	F	77%	12%	96%	8%	76%	16%
4 d	14 d	16 h	T	75%	17%	95%	8%	73%	20%

Table D.31 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
4 d	7 d	60 m	F	63%	10%	100%	1%	63%	10%
4 d	7 d	60 m	T	67%	10%	100%	1%	67%	11%
4 d	7 d	2 h	F	67%	12%	99%	1%	67%	12%
4 d	7 d	2 h	T	71%	12%	99%	1%	71%	12%
4 d	7 d	4 h	F	71%	11%	99%	2%	70%	12%
4 d	7 d	4 h	T	74%	12%	98%	2%	73%	13%
4 d	7 d	8 h	F	74%	12%	97%	4%	73%	12%
4 d	7 d	8 h	T	73%	15%	97%	5%	72%	17%
4 d	7 d	16 h	F	76%	13%	95%	8%	75%	18%
4 d	7 d	16 h	T	75%	17%	95%	8%	73%	21%

Table D.31 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, d), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓ Tower	↓ Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
∞	∞	60 m	F	79%	12%	100%	1%	79%	12%
∞	∞	60 m	F	79%	11%	100%	1%	79%	12%
∞	∞	60 m	T	79%	12%	100%	1%	79%	12%
∞	∞	60 m	T	79%	12%	100%	1%	79%	12%
∞	∞	2 h	F	79%	12%	99%	1%	79%	13%
∞	∞	2 h	F	80%	11%	99%	1%	79%	12%
∞	∞	2 h	T	80%	12%	99%	1%	79%	12%
∞	∞	2 h	T	80%	11%	99%	1%	79%	12%
∞	∞	4 h	F	80%	12%	99%	2%	79%	12%
∞	∞	4 h	F	80%	12%	99%	2%	80%	12%
∞	∞	4 h	T	79%	12%	99%	2%	78%	13%
∞	∞	4 h	T	79%	12%	99%	2%	78%	13%
∞	∞	8 h	F	80%	11%	98%	4%	79%	13%
∞	∞	8 h	F	80%	11%	98%	4%	79%	12%
∞	∞	8 h	T	78%	14%	97%	5%	77%	15%
∞	∞	8 h	T	78%	14%	97%	5%	77%	15%
∞	∞	16 h	F	80%	11%	96%	8%	79%	13%
∞	∞	16 h	F	80%	11%	96%	8%	79%	12%
∞	∞	16 h	T	78%	14%	95%	8%	76%	16%
∞	∞	16 h	T	78%	14%	95%	8%	76%	16%
∞	42 d	60 m	F	79%	12%	100%	1%	79%	12%
∞	42 d	60 m	T	79%	12%	100%	1%	79%	12%
∞	42 d	2 h	F	79%	12%	99%	1%	79%	13%
∞	42 d	2 h	T	79%	12%	99%	1%	79%	12%
∞	42 d	4 h	F	80%	12%	99%	2%	79%	12%
∞	42 d	4 h	T	79%	12%	99%	2%	78%	13%
∞	42 d	8 h	F	80%	11%	98%	4%	79%	13%
∞	42 d	8 h	T	78%	14%	97%	5%	77%	15%
∞	42 d	16 h	F	80%	11%	96%	8%	79%	13%
∞	42 d	16 h	T	78%	14%	95%	8%	75%	17%
∞	28 d	60 m	F	79%	12%	100%	1%	79%	12%
∞	28 d	60 m	T	79%	12%	100%	1%	79%	12%
∞	28 d	2 h	F	79%	12%	99%	1%	79%	12%
∞	28 d	2 h	T	80%	12%	99%	1%	79%	12%
∞	28 d	4 h	F	80%	12%	99%	2%	79%	12%

Table D.32: $P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
∞	28 d	4 h	T	79%	12%	99%	2%	78%	13%
∞	28 d	8 h	F	80%	11%	98%	4%	79%	13%
∞	28 d	8 h	T	78%	14%	97%	5%	77%	15%
∞	28 d	16 h	F	80%	11%	96%	8%	79%	13%
∞	28 d	16 h	T	78%	14%	95%	8%	76%	17%
∞	21 d	60 m	F	79%	12%	100%	1%	79%	12%
∞	21 d	60 m	T	79%	12%	100%	1%	79%	12%
∞	21 d	2 h	F	79%	12%	99%	1%	79%	12%
∞	21 d	2 h	T	80%	12%	99%	1%	79%	12%
∞	21 d	4 h	F	80%	12%	99%	2%	79%	12%
∞	21 d	4 h	T	79%	12%	99%	2%	78%	13%
∞	21 d	8 h	F	80%	11%	98%	4%	79%	13%
∞	21 d	8 h	T	78%	14%	97%	5%	77%	15%
∞	21 d	16 h	F	80%	11%	96%	8%	79%	13%
∞	21 d	16 h	T	78%	14%	95%	8%	75%	18%
∞	14 d	60 m	F	79%	12%	100%	1%	79%	12%
∞	14 d	60 m	T	79%	12%	100%	1%	79%	12%
∞	14 d	2 h	F	79%	12%	99%	1%	79%	13%
∞	14 d	2 h	T	79%	12%	99%	1%	79%	12%
∞	14 d	4 h	F	80%	12%	99%	2%	79%	12%
∞	14 d	4 h	T	79%	12%	99%	2%	78%	13%
∞	14 d	8 h	F	80%	12%	98%	4%	79%	13%
∞	14 d	8 h	T	78%	14%	97%	5%	77%	15%
∞	14 d	16 h	F	80%	11%	96%	8%	79%	13%
∞	14 d	16 h	T	76%	14%	95%	8%	73%	19%
∞	7 d	60 m	F	78%	12%	100%	1%	78%	13%
∞	7 d	60 m	T	79%	13%	100%	1%	78%	13%
∞	7 d	2 h	F	79%	13%	99%	1%	78%	13%
∞	7 d	2 h	T	79%	12%	99%	1%	79%	13%
∞	7 d	4 h	F	79%	12%	99%	2%	79%	13%
∞	7 d	4 h	T	79%	13%	99%	2%	78%	13%
∞	7 d	8 h	F	80%	11%	98%	4%	79%	13%
∞	7 d	8 h	T	76%	16%	97%	5%	75%	17%
∞	7 d	16 h	F	80%	11%	96%	8%	78%	14%
∞	7 d	16 h	T	77%	14%	95%	8%	73%	18%

Table D.32 (Continued): $P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
28 d	∞	60 m	F	79%	12%	100%	1%	78%	12%
28 d	∞	60 m	T	79%	12%	100%	1%	79%	13%
28 d	∞	2 h	F	79%	12%	99%	1%	79%	13%
28 d	∞	2 h	T	79%	12%	99%	1%	79%	12%
28 d	∞	4 h	F	80%	12%	99%	2%	79%	12%
28 d	∞	4 h	T	79%	12%	99%	2%	78%	13%
28 d	∞	8 h	F	80%	11%	98%	4%	79%	13%
28 d	∞	8 h	T	78%	13%	97%	5%	77%	15%
28 d	∞	16 h	F	80%	11%	96%	8%	79%	13%
28 d	∞	16 h	T	77%	14%	95%	8%	75%	18%
28 d	42 d	60 m	F	79%	12%	100%	1%	78%	12%
28 d	42 d	60 m	T	79%	12%	100%	1%	79%	13%
28 d	42 d	2 h	F	79%	12%	99%	1%	79%	13%
28 d	42 d	2 h	T	79%	12%	99%	1%	79%	12%
28 d	42 d	4 h	F	80%	12%	99%	2%	79%	12%
28 d	42 d	4 h	T	79%	12%	99%	2%	78%	13%
28 d	42 d	8 h	F	80%	11%	98%	4%	79%	13%
28 d	42 d	8 h	T	78%	13%	97%	5%	77%	15%
28 d	42 d	16 h	F	80%	11%	96%	8%	79%	13%
28 d	42 d	16 h	T	76%	14%	95%	8%	74%	17%
28 d	28 d	60 m	F	79%	12%	100%	1%	79%	12%
28 d	28 d	60 m	T	79%	12%	100%	1%	79%	12%
28 d	28 d	2 h	F	79%	12%	99%	1%	79%	13%
28 d	28 d	2 h	T	79%	12%	99%	1%	79%	12%
28 d	28 d	4 h	F	80%	12%	99%	2%	79%	12%
28 d	28 d	4 h	T	79%	12%	99%	2%	78%	13%
28 d	28 d	8 h	F	80%	11%	98%	4%	79%	13%
28 d	28 d	8 h	T	77%	14%	97%	5%	76%	16%
28 d	28 d	16 h	F	80%	11%	96%	8%	79%	13%
28 d	28 d	16 h	T	76%	13%	95%	8%	74%	17%
28 d	21 d	60 m	F	79%	12%	100%	1%	78%	12%
28 d	21 d	60 m	T	79%	12%	100%	1%	79%	12%
28 d	21 d	2 h	F	79%	12%	99%	1%	79%	13%
28 d	21 d	2 h	T	79%	12%	99%	1%	79%	12%
28 d	21 d	4 h	F	80%	12%	99%	2%	79%	12%

Table D.32 (Continued): $P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
28 d	21 d	4 h	T	79%	12%	99%	2%	78%	14%
28 d	21 d	8 h	F	80%	11%	98%	4%	79%	13%
28 d	21 d	8 h	T	77%	13%	97%	5%	76%	15%
28 d	21 d	16 h	F	80%	11%	96%	8%	79%	13%
28 d	21 d	16 h	T	76%	14%	95%	8%	74%	17%
28 d	14 d	60 m	F	79%	12%	100%	1%	78%	12%
28 d	14 d	60 m	T	79%	13%	100%	1%	79%	13%
28 d	14 d	2 h	F	79%	12%	99%	1%	79%	13%
28 d	14 d	2 h	T	79%	12%	99%	1%	79%	12%
28 d	14 d	4 h	F	80%	12%	99%	2%	79%	12%
28 d	14 d	4 h	T	79%	12%	99%	2%	78%	13%
28 d	14 d	8 h	F	80%	12%	98%	4%	79%	13%
28 d	14 d	8 h	T	77%	13%	97%	5%	76%	15%
28 d	14 d	16 h	F	80%	11%	96%	8%	78%	14%
28 d	14 d	16 h	T	76%	16%	95%	8%	73%	18%
28 d	7 d	60 m	F	78%	13%	100%	1%	78%	13%
28 d	7 d	60 m	T	78%	13%	100%	1%	78%	13%
28 d	7 d	2 h	F	79%	13%	99%	1%	78%	13%
28 d	7 d	2 h	T	79%	13%	99%	1%	79%	13%
28 d	7 d	4 h	F	79%	12%	99%	2%	79%	13%
28 d	7 d	4 h	T	78%	12%	99%	2%	78%	14%
28 d	7 d	8 h	F	80%	11%	98%	4%	79%	13%
28 d	7 d	8 h	T	75%	16%	97%	5%	74%	17%
28 d	7 d	16 h	F	79%	12%	96%	8%	78%	14%
28 d	7 d	16 h	T	76%	16%	95%	8%	73%	19%
21 d	∞	60 m	F	78%	13%	100%	1%	78%	13%
21 d	∞	60 m	T	78%	13%	100%	1%	78%	13%
21 d	∞	2 h	F	78%	13%	99%	1%	78%	13%
21 d	∞	2 h	T	79%	12%	99%	1%	78%	13%
21 d	∞	4 h	F	79%	12%	99%	2%	79%	13%
21 d	∞	4 h	T	79%	13%	99%	2%	78%	13%
21 d	∞	8 h	F	80%	11%	98%	4%	79%	13%
21 d	∞	8 h	T	76%	12%	97%	5%	75%	14%
21 d	∞	16 h	F	79%	12%	96%	8%	79%	13%
21 d	∞	16 h	T	76%	13%	95%	8%	75%	18%

Table D.32 (Continued): $P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
21 d	42 d	60 m	F	78%	13%	100%	1%	78%	13%
21 d	42 d	60 m	T	78%	13%	100%	1%	78%	13%
21 d	42 d	2 h	F	78%	13%	99%	1%	78%	13%
21 d	42 d	2 h	T	79%	12%	99%	1%	78%	13%
21 d	42 d	4 h	F	79%	12%	99%	2%	79%	13%
21 d	42 d	4 h	T	79%	13%	99%	2%	78%	13%
21 d	42 d	8 h	F	80%	11%	98%	4%	79%	13%
21 d	42 d	8 h	T	76%	12%	97%	5%	75%	14%
21 d	42 d	16 h	F	79%	12%	96%	8%	79%	13%
21 d	42 d	16 h	T	76%	13%	95%	8%	74%	17%
21 d	28 d	60 m	F	78%	13%	100%	1%	78%	13%
21 d	28 d	60 m	T	78%	13%	100%	1%	78%	13%
21 d	28 d	2 h	F	78%	13%	99%	1%	78%	13%
21 d	28 d	2 h	T	79%	12%	99%	1%	79%	13%
21 d	28 d	4 h	F	79%	12%	99%	2%	79%	13%
21 d	28 d	4 h	T	79%	13%	99%	2%	78%	13%
21 d	28 d	8 h	F	80%	11%	98%	4%	79%	13%
21 d	28 d	8 h	T	76%	13%	97%	5%	75%	15%
21 d	28 d	16 h	F	79%	12%	96%	8%	79%	13%
21 d	28 d	16 h	T	76%	12%	95%	8%	74%	17%
21 d	21 d	60 m	F	78%	13%	100%	1%	78%	13%
21 d	21 d	60 m	T	78%	13%	100%	1%	78%	13%
21 d	21 d	2 h	F	78%	13%	99%	1%	78%	13%
21 d	21 d	2 h	T	79%	12%	99%	1%	78%	13%
21 d	21 d	4 h	F	79%	12%	99%	2%	79%	13%
21 d	21 d	4 h	T	79%	13%	99%	2%	78%	13%
21 d	21 d	8 h	F	80%	11%	98%	4%	79%	13%
21 d	21 d	8 h	T	76%	13%	97%	5%	75%	15%
21 d	21 d	16 h	F	79%	12%	96%	8%	78%	13%
21 d	21 d	16 h	T	76%	13%	95%	8%	74%	17%
21 d	14 d	60 m	F	78%	13%	100%	1%	78%	13%
21 d	14 d	60 m	T	78%	13%	100%	1%	78%	13%
21 d	14 d	2 h	F	78%	13%	99%	1%	78%	13%
21 d	14 d	2 h	T	79%	13%	99%	1%	78%	13%
21 d	14 d	4 h	F	79%	12%	99%	2%	79%	13%

Table D.32 (Continued): $P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
21 d	14 d	4 h	T	78%	12%	99%	2%	78%	13%
21 d	14 d	8 h	F	79%	12%	98%	4%	79%	13%
21 d	14 d	8 h	T	76%	13%	97%	5%	75%	15%
21 d	14 d	16 h	F	79%	12%	96%	8%	78%	13%
21 d	14 d	16 h	T	75%	16%	95%	8%	73%	19%
21 d	7 d	60 m	F	78%	13%	100%	1%	78%	14%
21 d	7 d	60 m	T	78%	14%	100%	1%	78%	14%
21 d	7 d	2 h	F	78%	13%	99%	1%	78%	14%
21 d	7 d	2 h	T	78%	13%	99%	1%	78%	13%
21 d	7 d	4 h	F	79%	13%	99%	2%	78%	14%
21 d	7 d	4 h	T	78%	12%	99%	2%	77%	14%
21 d	7 d	8 h	F	79%	12%	98%	4%	78%	13%
21 d	7 d	8 h	T	75%	16%	97%	5%	73%	17%
21 d	7 d	16 h	F	78%	13%	96%	8%	78%	14%
21 d	7 d	16 h	T	75%	16%	95%	8%	73%	19%
14 d	∞	60 m	F	77%	14%	100%	1%	77%	14%
14 d	∞	60 m	T	77%	14%	100%	1%	77%	14%
14 d	∞	2 h	F	78%	14%	99%	1%	78%	14%
14 d	∞	2 h	T	78%	13%	99%	1%	78%	13%
14 d	∞	4 h	F	79%	13%	99%	2%	78%	14%
14 d	∞	4 h	T	78%	12%	99%	2%	78%	13%
14 d	∞	8 h	F	79%	12%	98%	4%	78%	13%
14 d	∞	8 h	T	76%	13%	97%	5%	74%	14%
14 d	∞	16 h	F	79%	12%	96%	8%	78%	14%
14 d	∞	16 h	T	76%	13%	95%	8%	74%	18%
14 d	42 d	60 m	F	77%	14%	100%	1%	77%	14%
14 d	42 d	60 m	T	77%	14%	100%	1%	77%	14%
14 d	42 d	2 h	F	78%	14%	99%	1%	78%	14%
14 d	42 d	2 h	T	78%	13%	99%	1%	78%	13%
14 d	42 d	4 h	F	79%	13%	99%	2%	78%	14%
14 d	42 d	4 h	T	78%	12%	99%	2%	78%	13%
14 d	42 d	8 h	F	79%	12%	98%	4%	78%	13%
14 d	42 d	8 h	T	76%	13%	97%	5%	74%	15%
14 d	42 d	16 h	F	79%	12%	96%	8%	78%	14%
14 d	42 d	16 h	T	76%	13%	95%	8%	73%	17%

Table D.32 (Continued): $P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
14 d	28 d	60 m	F	77%	14%	100%	1%	77%	14%
14 d	28 d	60 m	T	77%	14%	100%	1%	77%	14%
14 d	28 d	2 h	F	78%	14%	99%	1%	78%	14%
14 d	28 d	2 h	T	78%	13%	99%	1%	78%	13%
14 d	28 d	4 h	F	79%	13%	99%	2%	78%	14%
14 d	28 d	4 h	T	78%	12%	99%	2%	78%	14%
14 d	28 d	8 h	F	79%	12%	98%	4%	78%	14%
14 d	28 d	8 h	T	75%	12%	97%	5%	74%	15%
14 d	28 d	16 h	F	79%	12%	96%	8%	78%	14%
14 d	28 d	16 h	T	76%	12%	95%	8%	73%	17%
14 d	21 d	60 m	F	77%	14%	100%	1%	77%	14%
14 d	21 d	60 m	T	77%	14%	100%	1%	77%	14%
14 d	21 d	2 h	F	78%	14%	99%	1%	78%	14%
14 d	21 d	2 h	T	78%	13%	99%	1%	78%	13%
14 d	21 d	4 h	F	78%	13%	99%	2%	78%	14%
14 d	21 d	4 h	T	78%	12%	99%	2%	78%	14%
14 d	21 d	8 h	F	79%	12%	98%	4%	78%	13%
14 d	21 d	8 h	T	75%	13%	97%	5%	74%	15%
14 d	21 d	16 h	F	79%	12%	96%	8%	78%	14%
14 d	21 d	16 h	T	76%	12%	95%	8%	74%	16%
14 d	14 d	60 m	F	77%	14%	100%	1%	77%	14%
14 d	14 d	60 m	T	77%	14%	100%	1%	77%	14%
14 d	14 d	2 h	F	78%	14%	99%	1%	77%	14%
14 d	14 d	2 h	T	78%	13%	99%	1%	78%	13%
14 d	14 d	4 h	F	78%	13%	99%	2%	78%	14%
14 d	14 d	4 h	T	78%	12%	99%	2%	77%	13%
14 d	14 d	8 h	F	79%	12%	98%	4%	78%	13%
14 d	14 d	8 h	T	75%	13%	97%	5%	74%	16%
14 d	14 d	16 h	F	78%	12%	96%	8%	78%	14%
14 d	14 d	16 h	T	74%	16%	95%	8%	71%	18%
14 d	7 d	60 m	F	76%	14%	100%	1%	76%	14%
14 d	7 d	60 m	T	77%	14%	100%	1%	77%	14%
14 d	7 d	2 h	F	77%	14%	99%	1%	77%	14%
14 d	7 d	2 h	T	78%	14%	99%	1%	78%	14%
14 d	7 d	4 h	F	78%	14%	99%	2%	78%	14%

Table D.32 (Continued): $P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
14 d	7 d	4 h	T	76%	13%	99%	2%	76%	14%
14 d	7 d	8 h	F	78%	13%	98%	4%	78%	14%
14 d	7 d	8 h	T	74%	16%	97%	5%	72%	16%
14 d	7 d	16 h	F	78%	12%	96%	8%	77%	14%
14 d	7 d	16 h	T	74%	16%	95%	8%	71%	18%
7 d	∞	60 m	F	74%	14%	100%	1%	74%	14%
7 d	∞	60 m	T	75%	13%	100%	1%	74%	13%
7 d	∞	2 h	F	75%	13%	99%	1%	75%	14%
7 d	∞	2 h	T	76%	12%	99%	1%	76%	14%
7 d	∞	4 h	F	76%	13%	99%	2%	76%	14%
7 d	∞	4 h	T	76%	11%	99%	2%	76%	12%
7 d	∞	8 h	F	78%	11%	98%	4%	77%	13%
7 d	∞	8 h	T	76%	14%	97%	5%	74%	16%
7 d	∞	16 h	F	79%	11%	96%	8%	78%	13%
7 d	∞	16 h	T	77%	14%	95%	8%	76%	16%
7 d	42 d	60 m	F	74%	14%	100%	1%	74%	14%
7 d	42 d	60 m	T	75%	13%	100%	1%	75%	13%
7 d	42 d	2 h	F	75%	13%	99%	1%	75%	14%
7 d	42 d	2 h	T	76%	12%	99%	1%	76%	14%
7 d	42 d	4 h	F	76%	13%	99%	2%	76%	14%
7 d	42 d	4 h	T	76%	11%	99%	2%	76%	12%
7 d	42 d	8 h	F	78%	11%	98%	4%	77%	13%
7 d	42 d	8 h	T	76%	14%	97%	5%	74%	16%
7 d	42 d	16 h	F	79%	10%	96%	8%	78%	13%
7 d	42 d	16 h	T	76%	13%	95%	8%	74%	16%
7 d	28 d	60 m	F	74%	14%	100%	1%	74%	14%
7 d	28 d	60 m	T	75%	13%	100%	1%	74%	13%
7 d	28 d	2 h	F	75%	13%	99%	1%	75%	14%
7 d	28 d	2 h	T	76%	12%	99%	1%	76%	14%
7 d	28 d	4 h	F	76%	13%	99%	2%	76%	14%
7 d	28 d	4 h	T	76%	11%	99%	2%	76%	12%
7 d	28 d	8 h	F	78%	11%	98%	4%	77%	13%
7 d	28 d	8 h	T	75%	13%	97%	5%	73%	15%
7 d	28 d	16 h	F	79%	10%	96%	8%	78%	14%
7 d	28 d	16 h	T	76%	13%	95%	8%	74%	16%

Table D.32 (Continued): $P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
7 d	21 d	60 m	F	74%	14%	100%	1%	74%	14%
7 d	21 d	60 m	T	75%	13%	100%	1%	74%	13%
7 d	21 d	2 h	F	75%	13%	99%	1%	75%	14%
7 d	21 d	2 h	T	76%	12%	99%	1%	76%	14%
7 d	21 d	4 h	F	76%	13%	99%	2%	76%	14%
7 d	21 d	4 h	T	76%	11%	99%	2%	76%	12%
7 d	21 d	8 h	F	78%	12%	98%	4%	77%	13%
7 d	21 d	8 h	T	76%	13%	97%	5%	74%	16%
7 d	21 d	16 h	F	79%	11%	96%	8%	78%	14%
7 d	21 d	16 h	T	76%	13%	95%	8%	73%	16%
7 d	14 d	60 m	F	74%	14%	100%	1%	74%	14%
7 d	14 d	60 m	T	74%	13%	100%	1%	74%	13%
7 d	14 d	2 h	F	75%	13%	99%	1%	75%	14%
7 d	14 d	2 h	T	76%	13%	99%	1%	76%	14%
7 d	14 d	4 h	F	76%	13%	99%	2%	76%	14%
7 d	14 d	4 h	T	76%	11%	99%	2%	76%	12%
7 d	14 d	8 h	F	78%	12%	98%	4%	77%	14%
7 d	14 d	8 h	T	75%	13%	97%	5%	74%	16%
7 d	14 d	16 h	F	78%	11%	96%	8%	78%	14%
7 d	14 d	16 h	T	74%	16%	95%	8%	73%	20%
7 d	7 d	60 m	F	74%	14%	100%	1%	73%	14%
7 d	7 d	60 m	T	74%	13%	100%	1%	74%	14%
7 d	7 d	2 h	F	75%	13%	99%	1%	74%	14%
7 d	7 d	2 h	T	75%	12%	99%	1%	75%	13%
7 d	7 d	4 h	F	76%	13%	99%	2%	76%	14%
7 d	7 d	4 h	T	76%	12%	98%	2%	76%	13%
7 d	7 d	8 h	F	77%	12%	98%	4%	77%	13%
7 d	7 d	8 h	T	73%	13%	97%	5%	71%	15%
7 d	7 d	16 h	F	77%	12%	96%	8%	76%	14%
7 d	7 d	16 h	T	74%	16%	95%	8%	73%	20%
4 d	∞	60 m	F	66%	11%	100%	1%	65%	11%
4 d	∞	60 m	T	68%	11%	100%	1%	68%	11%
4 d	∞	2 h	F	67%	11%	99%	1%	67%	11%
4 d	∞	2 h	T	72%	12%	99%	1%	72%	12%
4 d	∞	4 h	F	72%	12%	99%	2%	71%	12%

Table D.32 (Continued): $P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
4 d	∞	4 h	T	74%	10%	99%	2%	74%	11%
4 d	∞	8 h	F	76%	11%	98%	4%	74%	12%
4 d	∞	8 h	T	76%	10%	97%	5%	75%	14%
4 d	∞	16 h	F	78%	11%	96%	8%	78%	13%
4 d	∞	16 h	T	77%	14%	95%	8%	76%	16%
4 d	42 d	60 m	F	66%	11%	100%	1%	65%	11%
4 d	42 d	60 m	T	68%	11%	100%	1%	68%	11%
4 d	42 d	2 h	F	67%	11%	99%	1%	67%	11%
4 d	42 d	2 h	T	72%	12%	99%	1%	72%	12%
4 d	42 d	4 h	F	72%	11%	99%	2%	71%	12%
4 d	42 d	4 h	T	74%	10%	99%	2%	74%	11%
4 d	42 d	8 h	F	76%	11%	98%	4%	74%	11%
4 d	42 d	8 h	T	76%	12%	97%	5%	75%	14%
4 d	42 d	16 h	F	79%	11%	96%	8%	78%	13%
4 d	42 d	16 h	T	76%	14%	95%	8%	74%	17%
4 d	28 d	60 m	F	66%	11%	100%	1%	65%	11%
4 d	28 d	60 m	T	68%	11%	100%	1%	68%	11%
4 d	28 d	2 h	F	68%	11%	99%	1%	67%	11%
4 d	28 d	2 h	T	72%	12%	99%	1%	72%	12%
4 d	28 d	4 h	F	72%	11%	99%	2%	72%	12%
4 d	28 d	4 h	T	74%	10%	99%	2%	74%	11%
4 d	28 d	8 h	F	76%	11%	98%	4%	74%	12%
4 d	28 d	8 h	T	75%	12%	97%	5%	74%	14%
4 d	28 d	16 h	F	79%	11%	96%	8%	78%	13%
4 d	28 d	16 h	T	76%	14%	95%	8%	74%	17%
4 d	21 d	60 m	F	66%	11%	100%	1%	65%	11%
4 d	21 d	60 m	T	68%	11%	100%	1%	68%	12%
4 d	21 d	2 h	F	68%	11%	99%	1%	67%	11%
4 d	21 d	2 h	T	72%	12%	99%	1%	72%	11%
4 d	21 d	4 h	F	72%	11%	99%	2%	72%	12%
4 d	21 d	4 h	T	74%	10%	99%	2%	74%	11%
4 d	21 d	8 h	F	76%	12%	98%	4%	74%	12%
4 d	21 d	8 h	T	75%	12%	97%	5%	74%	14%
4 d	21 d	16 h	F	78%	11%	96%	8%	78%	14%
4 d	21 d	16 h	T	76%	14%	95%	8%	74%	17%

Table D.32 (Continued): $P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
4 d	14 d	60 m	F	66%	11%	100%	1%	65%	11%
4 d	14 d	60 m	T	68%	11%	100%	1%	68%	11%
4 d	14 d	2 h	F	68%	11%	99%	1%	67%	11%
4 d	14 d	2 h	T	72%	12%	99%	1%	72%	12%
4 d	14 d	4 h	F	72%	11%	99%	2%	71%	12%
4 d	14 d	4 h	T	74%	11%	99%	2%	74%	11%
4 d	14 d	8 h	F	75%	12%	98%	4%	74%	12%
4 d	14 d	8 h	T	75%	12%	97%	5%	75%	15%
4 d	14 d	16 h	F	78%	12%	96%	8%	77%	14%
4 d	14 d	16 h	T	75%	17%	95%	8%	73%	20%
4 d	7 d	60 m	F	65%	11%	100%	1%	65%	11%
4 d	7 d	60 m	T	68%	11%	100%	1%	67%	11%
4 d	7 d	2 h	F	67%	11%	99%	1%	67%	11%
4 d	7 d	2 h	T	71%	12%	99%	1%	71%	12%
4 d	7 d	4 h	F	71%	11%	99%	2%	71%	12%
4 d	7 d	4 h	T	73%	12%	98%	2%	73%	12%
4 d	7 d	8 h	F	74%	11%	98%	4%	73%	13%
4 d	7 d	8 h	T	73%	15%	97%	5%	72%	16%
4 d	7 d	16 h	F	77%	12%	96%	8%	75%	14%
4 d	7 d	16 h	T	75%	17%	95%	8%	73%	21%

Table D.32 (Continued): $P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓ Tower	↓ Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
∞	∞	60 m	F	77%	13%	100%	1%	77%	13%
∞	∞	60 m	F	77%	13%	100%	1%	77%	13%
∞	∞	60 m	T	78%	12%	100%	1%	78%	12%
∞	∞	60 m	T	78%	12%	100%	1%	78%	12%
∞	∞	2 h	F	78%	12%	99%	1%	78%	12%
∞	∞	2 h	F	78%	12%	99%	1%	78%	12%
∞	∞	2 h	T	79%	12%	99%	1%	79%	12%
∞	∞	2 h	T	79%	12%	99%	1%	79%	12%
∞	∞	4 h	F	79%	12%	99%	2%	79%	12%
∞	∞	4 h	F	80%	11%	99%	2%	79%	12%
∞	∞	4 h	T	79%	12%	99%	2%	78%	13%
∞	∞	4 h	T	79%	12%	99%	2%	78%	13%
∞	∞	8 h	F	80%	12%	98%	4%	79%	12%
∞	∞	8 h	F	80%	11%	98%	4%	79%	12%
∞	∞	8 h	T	78%	14%	97%	5%	77%	15%
∞	∞	8 h	T	78%	14%	97%	5%	77%	15%
∞	∞	16 h	F	80%	11%	96%	8%	79%	13%
∞	∞	16 h	F	80%	11%	96%	8%	79%	12%
∞	∞	16 h	T	78%	14%	95%	8%	76%	16%
∞	∞	16 h	T	78%	14%	95%	8%	76%	16%
∞	42 d	60 m	F	77%	13%	100%	1%	77%	13%
∞	42 d	60 m	T	78%	12%	100%	1%	78%	12%
∞	42 d	2 h	F	78%	12%	99%	1%	78%	12%
∞	42 d	2 h	T	79%	12%	99%	1%	79%	13%
∞	42 d	4 h	F	79%	12%	99%	2%	79%	12%
∞	42 d	4 h	T	79%	12%	99%	2%	78%	13%
∞	42 d	8 h	F	80%	11%	98%	4%	79%	12%
∞	42 d	8 h	T	78%	14%	97%	5%	77%	15%
∞	42 d	16 h	F	80%	11%	96%	8%	79%	13%
∞	42 d	16 h	T	78%	14%	95%	8%	75%	17%
∞	28 d	60 m	F	77%	13%	100%	1%	77%	13%
∞	28 d	60 m	T	78%	12%	100%	1%	78%	12%
∞	28 d	2 h	F	78%	12%	99%	1%	78%	12%
∞	28 d	2 h	T	79%	12%	99%	1%	79%	12%
∞	28 d	4 h	F	79%	12%	99%	2%	79%	12%

Table D.33: $P(t \mid h, d, c, \Delta), P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
∞	28 d	4 h	T	79%	12%	99%	2%	78%	13%
∞	28 d	8 h	F	80%	11%	98%	4%	79%	12%
∞	28 d	8 h	T	78%	14%	97%	5%	77%	15%
∞	28 d	16 h	F	80%	11%	96%	8%	79%	13%
∞	28 d	16 h	T	78%	14%	95%	8%	76%	17%
∞	21 d	60 m	F	77%	13%	100%	1%	77%	13%
∞	21 d	60 m	T	78%	12%	100%	1%	78%	12%
∞	21 d	2 h	F	78%	12%	99%	1%	78%	12%
∞	21 d	2 h	T	79%	12%	99%	1%	79%	13%
∞	21 d	4 h	F	79%	12%	99%	2%	79%	12%
∞	21 d	4 h	T	79%	12%	99%	2%	78%	13%
∞	21 d	8 h	F	80%	11%	98%	4%	79%	12%
∞	21 d	8 h	T	78%	14%	97%	5%	77%	15%
∞	21 d	16 h	F	80%	11%	96%	8%	79%	13%
∞	21 d	16 h	T	77%	14%	95%	8%	75%	18%
∞	14 d	60 m	F	77%	13%	100%	1%	77%	13%
∞	14 d	60 m	T	78%	12%	100%	1%	78%	12%
∞	14 d	2 h	F	78%	12%	99%	1%	78%	12%
∞	14 d	2 h	T	79%	12%	99%	1%	79%	13%
∞	14 d	4 h	F	79%	12%	99%	2%	79%	12%
∞	14 d	4 h	T	79%	12%	99%	2%	78%	13%
∞	14 d	8 h	F	80%	12%	98%	4%	79%	12%
∞	14 d	8 h	T	78%	14%	97%	5%	77%	15%
∞	14 d	16 h	F	80%	11%	96%	8%	79%	13%
∞	14 d	16 h	T	76%	14%	95%	8%	73%	19%
∞	7 d	60 m	F	76%	13%	100%	1%	76%	13%
∞	7 d	60 m	T	78%	12%	100%	1%	78%	12%
∞	7 d	2 h	F	78%	12%	99%	1%	78%	12%
∞	7 d	2 h	T	78%	12%	99%	1%	78%	13%
∞	7 d	4 h	F	79%	12%	99%	2%	79%	12%
∞	7 d	4 h	T	78%	12%	99%	2%	78%	13%
∞	7 d	8 h	F	80%	12%	98%	4%	79%	13%
∞	7 d	8 h	T	77%	16%	97%	5%	74%	16%
∞	7 d	16 h	F	80%	12%	96%	8%	78%	14%
∞	7 d	16 h	T	77%	14%	95%	8%	73%	18%

Table D.33 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
28 d	∞	60 m	F	76%	12%	100%	1%	76%	12%
28 d	∞	60 m	T	77%	12%	100%	1%	77%	12%
28 d	∞	2 h	F	77%	11%	99%	1%	77%	11%
28 d	∞	2 h	T	78%	12%	99%	1%	78%	13%
28 d	∞	4 h	F	79%	12%	99%	2%	78%	12%
28 d	∞	4 h	T	79%	12%	99%	2%	78%	13%
28 d	∞	8 h	F	80%	11%	98%	4%	79%	13%
28 d	∞	8 h	T	78%	13%	97%	5%	77%	15%
28 d	∞	16 h	F	80%	11%	96%	8%	79%	13%
28 d	∞	16 h	T	77%	14%	95%	8%	75%	18%
28 d	42 d	60 m	F	76%	12%	100%	1%	76%	12%
28 d	42 d	60 m	T	77%	12%	100%	1%	77%	12%
28 d	42 d	2 h	F	77%	11%	99%	1%	77%	11%
28 d	42 d	2 h	T	78%	12%	99%	1%	78%	13%
28 d	42 d	4 h	F	79%	12%	99%	2%	78%	12%
28 d	42 d	4 h	T	79%	12%	99%	2%	78%	13%
28 d	42 d	8 h	F	80%	11%	98%	4%	79%	13%
28 d	42 d	8 h	T	78%	13%	97%	5%	77%	15%
28 d	42 d	16 h	F	80%	11%	96%	8%	79%	13%
28 d	42 d	16 h	T	76%	14%	95%	8%	74%	17%
28 d	28 d	60 m	F	76%	12%	100%	1%	76%	12%
28 d	28 d	60 m	T	77%	12%	100%	1%	77%	12%
28 d	28 d	2 h	F	77%	11%	99%	1%	77%	11%
28 d	28 d	2 h	T	78%	12%	99%	1%	78%	13%
28 d	28 d	4 h	F	79%	12%	99%	2%	78%	12%
28 d	28 d	4 h	T	79%	12%	99%	2%	78%	13%
28 d	28 d	8 h	F	80%	12%	98%	4%	79%	13%
28 d	28 d	8 h	T	77%	13%	97%	5%	76%	16%
28 d	28 d	16 h	F	80%	11%	96%	8%	79%	13%
28 d	28 d	16 h	T	76%	13%	95%	8%	74%	17%
28 d	21 d	60 m	F	76%	12%	100%	1%	76%	12%
28 d	21 d	60 m	T	77%	12%	100%	1%	77%	12%
28 d	21 d	2 h	F	77%	11%	99%	1%	77%	11%
28 d	21 d	2 h	T	78%	12%	99%	1%	78%	13%
28 d	21 d	4 h	F	79%	12%	99%	2%	78%	12%

Table D.33 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
28 d	21 d	4 h	T	79%	12%	99%	2%	78%	13%
28 d	21 d	8 h	F	80%	12%	98%	4%	79%	13%
28 d	21 d	8 h	T	77%	13%	97%	5%	76%	15%
28 d	21 d	16 h	F	80%	11%	96%	8%	79%	13%
28 d	21 d	16 h	T	76%	14%	95%	8%	74%	17%
28 d	14 d	60 m	F	76%	12%	100%	1%	76%	12%
28 d	14 d	60 m	T	77%	12%	100%	1%	77%	12%
28 d	14 d	2 h	F	77%	11%	99%	1%	77%	11%
28 d	14 d	2 h	T	78%	12%	99%	1%	78%	13%
28 d	14 d	4 h	F	79%	12%	99%	2%	78%	13%
28 d	14 d	4 h	T	78%	12%	99%	2%	78%	13%
28 d	14 d	8 h	F	80%	12%	98%	4%	79%	13%
28 d	14 d	8 h	T	77%	13%	97%	5%	76%	15%
28 d	14 d	16 h	F	80%	11%	96%	8%	79%	13%
28 d	14 d	16 h	T	76%	16%	95%	8%	73%	18%
28 d	7 d	60 m	F	76%	13%	100%	1%	76%	12%
28 d	7 d	60 m	T	77%	12%	100%	1%	77%	12%
28 d	7 d	2 h	F	77%	12%	99%	1%	77%	12%
28 d	7 d	2 h	T	78%	13%	99%	1%	78%	13%
28 d	7 d	4 h	F	79%	12%	99%	2%	78%	13%
28 d	7 d	4 h	T	78%	12%	99%	2%	77%	13%
28 d	7 d	8 h	F	79%	12%	98%	4%	78%	14%
28 d	7 d	8 h	T	75%	16%	97%	5%	74%	17%
28 d	7 d	16 h	F	80%	12%	96%	8%	78%	14%
28 d	7 d	16 h	T	76%	16%	95%	8%	73%	19%
21 d	∞	60 m	F	76%	12%	100%	1%	76%	12%
21 d	∞	60 m	T	77%	12%	100%	1%	77%	12%
21 d	∞	2 h	F	77%	11%	99%	1%	77%	11%
21 d	∞	2 h	T	78%	12%	99%	1%	78%	13%
21 d	∞	4 h	F	78%	12%	99%	2%	78%	13%
21 d	∞	4 h	T	79%	12%	99%	2%	78%	13%
21 d	∞	8 h	F	79%	12%	98%	4%	79%	14%
21 d	∞	8 h	T	76%	12%	97%	5%	75%	14%
21 d	∞	16 h	F	80%	11%	96%	8%	79%	13%
21 d	∞	16 h	T	76%	13%	95%	8%	75%	18%

Table D.33 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
21 d	42 d	60 m	F	76%	12%	100%	1%	76%	12%
21 d	42 d	60 m	T	77%	12%	100%	1%	77%	12%
21 d	42 d	2 h	F	77%	11%	99%	1%	77%	11%
21 d	42 d	2 h	T	78%	12%	99%	1%	78%	13%
21 d	42 d	4 h	F	78%	12%	99%	2%	78%	13%
21 d	42 d	4 h	T	79%	12%	99%	2%	78%	13%
21 d	42 d	8 h	F	79%	12%	98%	4%	79%	14%
21 d	42 d	8 h	T	76%	12%	97%	5%	75%	14%
21 d	42 d	16 h	F	80%	11%	96%	8%	79%	13%
21 d	42 d	16 h	T	76%	13%	95%	8%	74%	17%
21 d	28 d	60 m	F	76%	12%	100%	1%	76%	12%
21 d	28 d	60 m	T	77%	12%	100%	1%	77%	12%
21 d	28 d	2 h	F	77%	11%	99%	1%	77%	11%
21 d	28 d	2 h	T	78%	12%	99%	1%	78%	13%
21 d	28 d	4 h	F	78%	12%	99%	2%	78%	13%
21 d	28 d	4 h	T	79%	12%	99%	2%	78%	13%
21 d	28 d	8 h	F	79%	12%	98%	4%	79%	13%
21 d	28 d	8 h	T	76%	13%	97%	5%	75%	14%
21 d	28 d	16 h	F	80%	11%	96%	8%	79%	13%
21 d	28 d	16 h	T	76%	12%	95%	8%	74%	17%
21 d	21 d	60 m	F	76%	12%	100%	1%	76%	12%
21 d	21 d	60 m	T	77%	12%	100%	1%	77%	12%
21 d	21 d	2 h	F	77%	11%	99%	1%	77%	11%
21 d	21 d	2 h	T	78%	12%	99%	1%	78%	13%
21 d	21 d	4 h	F	78%	12%	99%	2%	78%	13%
21 d	21 d	4 h	T	79%	12%	99%	2%	78%	13%
21 d	21 d	8 h	F	79%	12%	98%	4%	79%	14%
21 d	21 d	8 h	T	76%	13%	97%	5%	75%	15%
21 d	21 d	16 h	F	80%	11%	96%	8%	79%	14%
21 d	21 d	16 h	T	76%	13%	95%	8%	74%	17%
21 d	14 d	60 m	F	76%	12%	100%	1%	76%	12%
21 d	14 d	60 m	T	77%	12%	100%	1%	77%	12%
21 d	14 d	2 h	F	77%	11%	99%	1%	77%	12%
21 d	14 d	2 h	T	78%	12%	99%	1%	78%	13%
21 d	14 d	4 h	F	78%	13%	99%	2%	78%	13%

Table D.33 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
21 d	14 d	4 h	T	78%	12%	99%	2%	78%	13%
21 d	14 d	8 h	F	79%	12%	98%	4%	78%	13%
21 d	14 d	8 h	T	76%	13%	97%	5%	75%	15%
21 d	14 d	16 h	F	80%	11%	96%	8%	78%	14%
21 d	14 d	16 h	T	75%	16%	95%	8%	73%	19%
21 d	7 d	60 m	F	76%	12%	100%	1%	76%	12%
21 d	7 d	60 m	T	76%	12%	100%	1%	76%	12%
21 d	7 d	2 h	F	77%	12%	99%	1%	77%	12%
21 d	7 d	2 h	T	78%	13%	99%	1%	78%	13%
21 d	7 d	4 h	F	78%	13%	99%	2%	78%	14%
21 d	7 d	4 h	T	77%	12%	99%	2%	77%	13%
21 d	7 d	8 h	F	79%	12%	98%	4%	78%	13%
21 d	7 d	8 h	T	75%	16%	97%	5%	73%	17%
21 d	7 d	16 h	F	79%	12%	96%	8%	78%	14%
21 d	7 d	16 h	T	75%	16%	95%	8%	73%	19%
14 d	∞	60 m	F	74%	12%	100%	1%	74%	12%
14 d	∞	60 m	T	75%	12%	100%	1%	75%	13%
14 d	∞	2 h	F	75%	13%	99%	1%	75%	13%
14 d	∞	2 h	T	77%	13%	99%	1%	77%	14%
14 d	∞	4 h	F	78%	14%	99%	2%	77%	14%
14 d	∞	4 h	T	78%	12%	99%	2%	77%	13%
14 d	∞	8 h	F	79%	12%	98%	4%	78%	13%
14 d	∞	8 h	T	76%	13%	97%	5%	74%	15%
14 d	∞	16 h	F	79%	12%	96%	8%	78%	13%
14 d	∞	16 h	T	76%	13%	95%	8%	74%	18%
14 d	42 d	60 m	F	74%	12%	100%	1%	74%	12%
14 d	42 d	60 m	T	75%	12%	100%	1%	75%	13%
14 d	42 d	2 h	F	75%	13%	99%	1%	75%	13%
14 d	42 d	2 h	T	77%	14%	99%	1%	77%	14%
14 d	42 d	4 h	F	78%	14%	99%	2%	77%	14%
14 d	42 d	4 h	T	78%	12%	99%	2%	78%	13%
14 d	42 d	8 h	F	79%	12%	98%	4%	78%	13%
14 d	42 d	8 h	T	76%	13%	97%	5%	74%	15%
14 d	42 d	16 h	F	79%	12%	96%	8%	78%	13%
14 d	42 d	16 h	T	76%	13%	95%	8%	73%	17%

Table D.33 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
14 d	28 d	60 m	F	74%	12%	100%	1%	74%	11%
14 d	28 d	60 m	T	75%	12%	100%	1%	75%	13%
14 d	28 d	2 h	F	75%	13%	99%	1%	75%	13%
14 d	28 d	2 h	T	77%	14%	99%	1%	77%	14%
14 d	28 d	4 h	F	78%	14%	99%	2%	77%	14%
14 d	28 d	4 h	T	78%	12%	99%	2%	77%	13%
14 d	28 d	8 h	F	79%	12%	98%	4%	78%	13%
14 d	28 d	8 h	T	75%	13%	97%	5%	74%	15%
14 d	28 d	16 h	F	79%	12%	96%	8%	78%	14%
14 d	28 d	16 h	T	76%	12%	95%	8%	73%	17%
14 d	21 d	60 m	F	74%	12%	100%	1%	74%	11%
14 d	21 d	60 m	T	75%	12%	100%	1%	75%	13%
14 d	21 d	2 h	F	75%	13%	99%	1%	75%	13%
14 d	21 d	2 h	T	77%	14%	99%	1%	77%	14%
14 d	21 d	4 h	F	78%	14%	99%	2%	77%	14%
14 d	21 d	4 h	T	78%	12%	99%	2%	77%	14%
14 d	21 d	8 h	F	79%	12%	98%	4%	78%	13%
14 d	21 d	8 h	T	75%	13%	97%	5%	74%	15%
14 d	21 d	16 h	F	79%	12%	96%	8%	78%	13%
14 d	21 d	16 h	T	76%	12%	95%	8%	74%	16%
14 d	14 d	60 m	F	74%	12%	100%	1%	74%	12%
14 d	14 d	60 m	T	75%	12%	100%	1%	75%	13%
14 d	14 d	2 h	F	75%	13%	99%	1%	75%	13%
14 d	14 d	2 h	T	77%	14%	99%	1%	77%	14%
14 d	14 d	4 h	F	78%	14%	99%	2%	77%	14%
14 d	14 d	4 h	T	78%	11%	99%	2%	77%	13%
14 d	14 d	8 h	F	79%	12%	98%	4%	78%	13%
14 d	14 d	8 h	T	75%	13%	97%	5%	74%	16%
14 d	14 d	16 h	F	79%	11%	96%	8%	78%	14%
14 d	14 d	16 h	T	74%	16%	95%	8%	71%	18%
14 d	7 d	60 m	F	74%	12%	100%	1%	74%	11%
14 d	7 d	60 m	T	75%	13%	100%	1%	75%	13%
14 d	7 d	2 h	F	75%	13%	99%	1%	75%	13%
14 d	7 d	2 h	T	77%	13%	99%	1%	77%	14%
14 d	7 d	4 h	F	77%	13%	99%	2%	77%	14%

Table D.33 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
14 d	7 d	4 h	T	76%	13%	99%	2%	76%	14%
14 d	7 d	8 h	F	78%	12%	98%	4%	78%	14%
14 d	7 d	8 h	T	74%	16%	97%	5%	72%	16%
14 d	7 d	16 h	F	78%	11%	96%	8%	78%	14%
14 d	7 d	16 h	T	74%	16%	95%	8%	71%	18%
7 d	∞	60 m	F	70%	12%	100%	1%	70%	12%
7 d	∞	60 m	T	71%	12%	100%	1%	71%	12%
7 d	∞	2 h	F	71%	12%	99%	1%	71%	12%
7 d	∞	2 h	T	75%	12%	99%	1%	75%	13%
7 d	∞	4 h	F	74%	12%	99%	2%	74%	13%
7 d	∞	4 h	T	76%	11%	99%	2%	76%	12%
7 d	∞	8 h	F	77%	10%	98%	4%	76%	13%
7 d	∞	8 h	T	76%	14%	97%	5%	74%	16%
7 d	∞	16 h	F	79%	10%	96%	8%	78%	13%
7 d	∞	16 h	T	77%	14%	95%	8%	76%	16%
7 d	42 d	60 m	F	70%	12%	100%	1%	70%	12%
7 d	42 d	60 m	T	71%	12%	100%	1%	71%	12%
7 d	42 d	2 h	F	71%	11%	99%	1%	71%	12%
7 d	42 d	2 h	T	75%	12%	99%	1%	75%	13%
7 d	42 d	4 h	F	74%	11%	99%	2%	74%	13%
7 d	42 d	4 h	T	76%	11%	99%	2%	76%	12%
7 d	42 d	8 h	F	77%	11%	98%	4%	76%	13%
7 d	42 d	8 h	T	76%	14%	97%	5%	74%	16%
7 d	42 d	16 h	F	79%	10%	96%	8%	78%	13%
7 d	42 d	16 h	T	76%	13%	95%	8%	74%	16%
7 d	28 d	60 m	F	70%	12%	100%	1%	70%	12%
7 d	28 d	60 m	T	71%	12%	100%	1%	71%	12%
7 d	28 d	2 h	F	71%	11%	99%	1%	71%	12%
7 d	28 d	2 h	T	75%	12%	99%	1%	75%	13%
7 d	28 d	4 h	F	74%	11%	99%	2%	74%	13%
7 d	28 d	4 h	T	76%	11%	99%	2%	76%	12%
7 d	28 d	8 h	F	77%	11%	98%	4%	76%	13%
7 d	28 d	8 h	T	75%	13%	97%	5%	73%	15%
7 d	28 d	16 h	F	79%	11%	96%	8%	78%	14%
7 d	28 d	16 h	T	76%	13%	95%	8%	74%	16%

Table D.33 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
7 d	21 d	60 m	F	70%	12%	100%	1%	70%	12%
7 d	21 d	60 m	T	71%	12%	100%	1%	71%	12%
7 d	21 d	2 h	F	71%	11%	99%	1%	71%	12%
7 d	21 d	2 h	T	75%	12%	99%	1%	75%	13%
7 d	21 d	4 h	F	74%	11%	99%	2%	74%	13%
7 d	21 d	4 h	T	76%	11%	99%	2%	76%	12%
7 d	21 d	8 h	F	77%	11%	98%	4%	76%	13%
7 d	21 d	8 h	T	76%	13%	97%	5%	74%	16%
7 d	21 d	16 h	F	79%	11%	96%	8%	78%	14%
7 d	21 d	16 h	T	76%	13%	95%	8%	73%	16%
7 d	14 d	60 m	F	70%	12%	100%	1%	70%	12%
7 d	14 d	60 m	T	71%	12%	100%	1%	71%	12%
7 d	14 d	2 h	F	71%	12%	99%	1%	71%	12%
7 d	14 d	2 h	T	75%	13%	99%	1%	75%	13%
7 d	14 d	4 h	F	74%	11%	99%	2%	74%	13%
7 d	14 d	4 h	T	76%	11%	99%	2%	76%	12%
7 d	14 d	8 h	F	77%	11%	98%	4%	76%	13%
7 d	14 d	8 h	T	75%	13%	97%	5%	74%	16%
7 d	14 d	16 h	F	78%	11%	96%	8%	78%	14%
7 d	14 d	16 h	T	74%	16%	95%	8%	73%	20%
7 d	7 d	60 m	F	70%	12%	100%	1%	70%	12%
7 d	7 d	60 m	T	71%	12%	100%	1%	71%	12%
7 d	7 d	2 h	F	71%	12%	99%	1%	71%	12%
7 d	7 d	2 h	T	75%	13%	99%	1%	74%	13%
7 d	7 d	4 h	F	74%	11%	99%	2%	74%	13%
7 d	7 d	4 h	T	76%	12%	98%	2%	76%	12%
7 d	7 d	8 h	F	77%	12%	98%	4%	76%	13%
7 d	7 d	8 h	T	73%	13%	97%	5%	71%	15%
7 d	7 d	16 h	F	77%	11%	96%	8%	76%	12%
7 d	7 d	16 h	T	74%	16%	95%	8%	73%	20%
4 d	∞	60 m	F	60%	10%	100%	1%	60%	9%
4 d	∞	60 m	T	65%	10%	100%	1%	65%	10%
4 d	∞	2 h	F	65%	11%	99%	1%	64%	10%
4 d	∞	2 h	T	71%	12%	99%	1%	71%	12%
4 d	∞	4 h	F	70%	11%	99%	2%	68%	12%

Table D.33 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
4 d	∞	4 h	T	74%	10%	99%	2%	74%	11%
4 d	∞	8 h	F	73%	10%	98%	4%	73%	12%
4 d	∞	8 h	T	76%	10%	97%	5%	75%	14%
4 d	∞	16 h	F	78%	11%	96%	8%	77%	13%
4 d	∞	16 h	T	77%	14%	95%	8%	76%	16%
4 d	42 d	60 m	F	60%	10%	100%	1%	60%	9%
4 d	42 d	60 m	T	65%	10%	100%	1%	65%	10%
4 d	42 d	2 h	F	65%	10%	99%	1%	64%	11%
4 d	42 d	2 h	T	71%	12%	99%	1%	71%	12%
4 d	42 d	4 h	F	70%	11%	99%	2%	68%	12%
4 d	42 d	4 h	T	74%	10%	99%	2%	74%	11%
4 d	42 d	8 h	F	73%	11%	98%	4%	73%	12%
4 d	42 d	8 h	T	76%	12%	97%	5%	75%	14%
4 d	42 d	16 h	F	78%	12%	96%	8%	77%	13%
4 d	42 d	16 h	T	76%	14%	95%	8%	74%	17%
4 d	28 d	60 m	F	60%	10%	100%	1%	60%	10%
4 d	28 d	60 m	T	65%	11%	100%	1%	65%	11%
4 d	28 d	2 h	F	65%	11%	99%	1%	64%	11%
4 d	28 d	2 h	T	71%	12%	99%	1%	71%	12%
4 d	28 d	4 h	F	70%	11%	99%	2%	68%	12%
4 d	28 d	4 h	T	74%	10%	99%	2%	74%	11%
4 d	28 d	8 h	F	73%	11%	98%	4%	73%	12%
4 d	28 d	8 h	T	75%	12%	97%	5%	74%	14%
4 d	28 d	16 h	F	78%	12%	96%	8%	77%	13%
4 d	28 d	16 h	T	76%	14%	95%	8%	74%	17%
4 d	21 d	60 m	F	60%	10%	100%	1%	60%	10%
4 d	21 d	60 m	T	65%	11%	100%	1%	65%	11%
4 d	21 d	2 h	F	65%	11%	99%	1%	64%	11%
4 d	21 d	2 h	T	71%	12%	99%	1%	71%	13%
4 d	21 d	4 h	F	70%	11%	99%	2%	68%	12%
4 d	21 d	4 h	T	74%	10%	99%	2%	74%	11%
4 d	21 d	8 h	F	73%	11%	98%	4%	73%	12%
4 d	21 d	8 h	T	75%	12%	97%	5%	74%	14%
4 d	21 d	16 h	F	78%	11%	96%	8%	77%	14%
4 d	21 d	16 h	T	76%	14%	95%	8%	74%	17%

Table D.33 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

				Correct Attempts		Attempts		Correct Trials	
↓Tower	↓Regime	Weight	Hard	Median	MAD	μ	σ	Median	MAD
4 d	14 d	60 m	F	60%	10%	100%	1%	60%	10%
4 d	14 d	60 m	T	65%	11%	100%	1%	65%	11%
4 d	14 d	2 h	F	65%	11%	99%	1%	64%	11%
4 d	14 d	2 h	T	71%	12%	99%	1%	71%	12%
4 d	14 d	4 h	F	70%	11%	99%	2%	68%	12%
4 d	14 d	4 h	T	74%	11%	99%	2%	74%	11%
4 d	14 d	8 h	F	73%	11%	98%	4%	72%	12%
4 d	14 d	8 h	T	75%	12%	97%	5%	75%	15%
4 d	14 d	16 h	F	77%	11%	96%	8%	76%	15%
4 d	14 d	16 h	T	75%	17%	95%	8%	73%	20%
4 d	7 d	60 m	F	60%	10%	100%	1%	59%	10%
4 d	7 d	60 m	T	65%	10%	100%	1%	64%	10%
4 d	7 d	2 h	F	64%	11%	99%	1%	64%	11%
4 d	7 d	2 h	T	71%	12%	99%	1%	70%	13%
4 d	7 d	4 h	F	68%	11%	99%	2%	68%	11%
4 d	7 d	4 h	T	73%	12%	98%	2%	73%	12%
4 d	7 d	8 h	F	72%	10%	98%	4%	71%	12%
4 d	7 d	8 h	T	73%	15%	97%	5%	72%	16%
4 d	7 d	16 h	F	76%	11%	96%	8%	75%	14%
4 d	7 d	16 h	T	75%	17%	95%	8%	73%	21%

Table D.33 (Continued): $P(t \mid h, d, c, \Delta), P(t \mid h, w, c, \Delta), P(t \mid h, c, \Delta), P(t \mid r, h, w), P(t \mid r, h), P(t \mid r)$

Intended to be blank.

Bibliography

[1] 3GPP. Radio Resource Control (rrc); Protocol Specification: 3gpp ts 25.331. http://www.3gpp.org/ftp/Specs/html-info/25331.htm, 2011.

[2] 3GPP. ts 23.003: Numbering, Addressing And Identification. http://www.3gpp.org/DynaReport/23003.htm, October 2014. Version 12.4.1.

[3] Arnold, Taylor A. and Emerson, John W. Nonparametric Goodness-of-fit Tests For Discrete Null Distributions. *The R Journal*, 3(2):34–39, 2011.

[4] Wasserman, Larry A. *All Of Statistics: A Concise Course In Statistical Inference.* Springer Science+Business Media, 2004.

[5] Wasserman, Larry A. *All Of Nonparametric Statistics.* Springer Science+Business Media, 2006.

[6] Clauset, Aaron, Shalizi, Cosma Rohilla, and Newman, M. E. J. Power-law Distributions In Empirical Data. *SIAM Rev.*, 51(4):661–703, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, November 2009.

[7] Primo, Abena, Phoha, Vir V, Kumar, Rajesh, and Serwadda, Abdul. Context-aware Active Authentication Using Smartphone Accelerometer Measurements. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 98–105, 2014.

[8] Rahmati, Ahmad and Zhong, Lin. Context-for-wireless: Context-sensitive Energy-efficient Wireless Data Transfer. *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services*, pages 165–178, ACM, New York, NY, USA, 2007.

[9] Rahmati, Ahmad and Zhong, Lin. Context-based Network Estimation For Energy-efficient Ubiquitous Wireless Connectivity. *IEEE Transactions on*

Mobile Computing, page 54-66, IEEE Computer Society, Los Alamitos, CA, USA, 2011.

[10] Kirmse, Andrew, Udeshi, Tushar, Bellver, Pablo, and Shuma, Jim. Extracting Patterns From Location History. *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 397–400, 2011.

[11] LaMarca, Anthony, Chawathe, Yatin, Consolvo, Sunny, Hightower, Jeffrey, Smith, Ian, Scott, James, Sohn, Timothy, Howard, James, Hughes, Jeff, Potter, Fred, Tabert, Jason, Powledge, Pauline, Borriello, Gaetano, and Schilit, Bill. Place Lab: Device Positioning Using Radio Beacons In The Wild. *Proceedings of the Third International Conference on Pervasive Computing*, pages 116–133, Springer-Verlag, Berlin, Heidelberg, 2005.

[12] AT&T. An Update For Our Smartphone Customers With Unlimited Data Plans. `http://www.att.com/gen/press-room?pid=20535`, October 2011.

[13] Aban, Inmaculada B., Meerschaert, Mark M., and Panorska, Anna K. Parameter Estimation For The Truncated Pareto Distribution. *Journal of the American Statistical Association*, 101:270-277, 2006.

[14] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy Consumption In Mobile Phones: A Measurement Study And Implications For Network Applications. *ACM SIGCOMM Internet Measurement Conference*, pages 280–293, ACM, New York, NY, USA, November 2009.

[15] David Beazley. Understanding The python gil. http://www.dabeaz.com/GIL/, February 2010.

[16] Brian N. Bershad, Stefan Savage, Przemysław Pardyak, Emin Gün Sirer, Marc E. Fiuczynski, David Becker, Craig Chambers, and Susan Eggers. Extensibility, Safety And Performance In The Spin Operating System. *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP)*, December 1995.

[17] United States Census Bureau. Vintage 2011 City And Town (incorporated Place And Minor Civil Division) Population Datasets—Population Estimates Of Incorporated Places: April 1, 2010 To July 1, 2011. `http://www.census.gov/popest/data/datasets.html`, `http://www.census.gov/popest/data/cities/totals/2011/files/SUB-EST2011-IP.csv`, 2011.

[18] Gonzalez, Marta C., Hidalgo, Cesar A., and Barabasi, Albert-Laszlo. Understanding Individual Human Mobility Patterns. *Nature*, 453(7196):779–782, Nature Publishing Group, June 2008.

[19] Noulas, Anastasios AND Scellato, Salvatore AND Lambiotte, Renaud AND Pontil, Massimiliano AND Mascolo, Cecilia. A Tale Of Many Cities: Universal Patterns In Human Urban Mobility. *PLoS ONE*, 7(5):1-10, Public Library of Science, 05 2012.

[20] William Chedsey. Greenwald: nsa Creates Blackmail Risk For Women Who Get Abortions. `http://www.newsmax.com/Newsfront/edward-snowden-glenn-greenwald-nsa-abortion/2013/12/20/id/543086/`, December 2013.

[21] S. Cheshire and M. Krochmal. Special-use Domain Names. RFC 6761 (Proposed Standard), 6761, IETF, February 2013.

[22] Aaron Clauset. plfit.m: Possible Bug In Computation Of ks Statistic. Personal Communication, August 2013.

[23] Consumer Reports. Bill Shock Is Common. `http://www.consumerreports.org/cro/magazine-archive/2011/january/electronics/best-cell-plans-and-providers/cell-phone-bills/index.htm`, January 2011.

[24] Costa, Costandinos, Laoudias, Christos, Zeinalipour-Yazti, Demetrios, and Gunopulos, Dimitrios. Smarttrace: Finding Similar Trajectories In Smartphone Networks Without Disclosing The Traces. *2011 IEEE 27th International Conference on Data Engineering*, pages 1288–1291, 2011.

[25] Zhang, Daqiang, Zhang, Daqing, Xiong, Haoyi, Yang, Laurence T, and Gauthier, Vincent. Nextcell: Predicting Location Using Social Interplay From Cell Phone Traces. *IEEE Transactions on Computers*, 64(2):452–463, IEEE, 2015.

[26] Brian D. Davison. Assertion: Prefetching With Get Is Not Good. *6th International Workshop on Web Caching and Content Distribution (WCW)*, pages 203–215, 2001.

[27] Choujaa, Driss and Dulay, Naranker. Predicting Human Behaviour From Selected Mobile Phone Data Points. *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, pages 105–108, ACM, New York, NY, USA, 2010.

[28] Charles Duhigg. How Companies Learn Your Secrets. http://www.nytimes.com/2012/02/19/magazine/shopping-habits.html, February 2012.

[29] Tarn Duong. Ks: Kernel Smoothing. R package version 1.8.13, 2013.

[30] Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis: Probabilistic Models Of Proteins And Nucleic Acids*. Cambridge University Press, 1998.

[31] Anderson, Edgar. The Irises Of The Gaspe Peninsula. *Bulletin of the American Iris Society*, 59:2–5, 1935.

[32] Dawson R. Engler, M. Frans Kaashoek, and O'Toole Jr., James. Exokernel: An Operating System Architecture For Application-level Resource Management. *15th ACM SOSP*, pages 251–266, December 1995.

[33] Ericsson AB. What Consumers Want From tv/video Solutions. http://www.ericsson.com/news/090626_what_consumers_wants_254740099_c, June 2009.

[34] Hossein Falaki, Ratul Mahajan, Srikanth Kandula, Dimitrios Lymberopoulos, Ramesh Govindan, and Deborah Estrin. Diversity In Smartphone Usage. *International Conference on Mobile Systems, Applications, and Services*, pages 179–194, ACM, New York, NY, USA, June 2010.

[35] Erbas, Fazli, Steuer, Jan, Eggesieker, Dirk, Kyamakya, Kyandoghere, and Jobinann, K. A Regular Path Recognition Method And Prediction Of User Movements In Wireless Networks. *Vehicular Technology Conference, 2001. VTC 2001 Fall. IEEE VTS 54th*, volume 4, pages 2672–2676, 2001.

[36] Federal Communications Commission. Mobility Fund Phase I Auction Scheduled For September 27, 2012. http://wireless.fcc.gov/auctions/default.htm?job=auction_summary&id=901, February 2012. AU Docket No. 12-25.

[37] Edward W. Felten. Written Testimony Of Edward W. Felten Professor Of Computer Science And Public Affairs, Princeton University United States Senate, Committee On The Judiciary Hearing On Continued Oversight Of The Foreign Intelligence Surveillance Act. http://www.cs.princeton.edu/~felten/testimony-2013-10-02.pdf, October 2013.

[38] Chris Foresman. New At&t Data Plans Milk Data Gluttons, Lower Costs For Most. `http://arstechnica.com/telecom/news/2010/06/new-att-data-plans-milk-data-gluttons-lower-costs-for-most.ars`, June 2010.

[39] Dinan Gunawardena, Thomas Karagiannis, Alexandre Proutiere, and Milan Vojnovic. Characterizing Podcast Services: Publishing, Usage, And Dissemination. *ACM SIGCOMM Internet Measurement Conference*, pages 209–222, ACM, New York, NY, USA, November 2009.

[40] Kuenning, Geoffrey H. and Popek, Gerald J. Automated Hoarding For Mobile Computers. *SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 264–275, ACM, New York, NY, USA, 1997.

[41] Kuenning, Geoffrey H., Ma, Wilkie, Reiher, Peter, and Popek, Gerald J. Simplifying Automated Hoarding Methods. *MSWiM '02: Proceedings of the 5th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems*, pages 15–21, ACM, New York, NY, USA, 2002.

[42] Press, William H., Teukolsky, Saul A., Vetterling, William T., and Flannery, Brian P. *Numerical Recipes 3rd Edition: The Art Of Scientific Computing*. Cambridge University Press, 2007.

[43] Steven M. Hand. Self-paging In The Nemesis Operating System. *3rd OSDI*, pages 73–86, 1999.

[44] Xiong, Haoyi, Zhang, Daqing, Zhang, Daqiang, and Gauthier, Vincent. Predicting Mobile Phone User Locations By Exploiting Collective Behavioral Patterns. *Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC/ATC), 2012 9th International Conference on*, pages 164–171, 2012.

[45] Kieran Harty and David R. Cheriton. Application-controlled Physical Memory Using External Page-cache Management. *ASPLOS-V*, October 1992.

[46] Henry Haverinen, Jonne Siren, and Pasi Eronen. Energy Consumption Of Always-on Applications In Wcdma Networks. *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*, pages 964–968, April 2007.

[47] Kang, Jong Hee, Welbourne, William, Stewart, Benjamin, and Borriello, Gaetano. Extracting Places From Traces Of Locations. *Proceedings of the*

2nd ACM international workshop on Wireless mobile applications and services on WLAN hotspots, pages 110–118, 2004.

[48] Lieberman, Henry and Selker, Ted. Out Of Context: Computer Systems That Adapt To, And Learn From, Context. *IBM systems journal*, 39(3.4):617–632, IBM, 2000.

[49] Benoît Hervier. Khweeteur. `http://khertan.net/khweeteur`. Source code: `https://gitorious.org/khweeteur`.

[50] Furukawa, Hiroshi and Akaiwa, Yoshihiko. A Microcell Overlaid With Umbrella Cell System. *Vehicular Technology Conference, 1994 IEEE 44th*, volume 3, pages 1455–1459, jun 1994.

[51] Kantz, Holger and Schreiber, Thomas. *Nonlinear Time Series Analysis*. Cambridge university press, 2004.

[52] Burbey, Ingrid and Martin, Thomas L. When Will You Be At The Office? Predicting Future Locations And Times. *International Conference on Mobile Computing, Applications, and Services*, pages 156–175, 2010.

[53] Hyndman, Rob J. and Fan, Yanan. Sample Quantiles In Statistical Packages. *The American Statistician*, 50(4):361–365, American Statistical Association, November 1996.

[54] Kistler, James J. and Satyanarayanan, M. Disconnected Operation In The Coda File System. *ACM Trans. Comput. Syst.*, 10(1):3–25, ACM, New York, NY, USA, 1992.

[55] Newman, M. E. J. Power Laws, Pareto Distributions And Zipf's Law. *Contemporary Physics*, 46:323–351, December 2005.

[56] Sheather, S. J. and Jones, M. C. A Reliable Data-based Bandwidth Selection Method For Kernel Density Estimation. *Journal of the Royal Statistical Society. Series B (Methodological)*, 53(3):683–690, Blackwell Publishing for the Royal Statistical Society, 1991.

[57] Onnela, J.-P., Saramk̈i, J., Hyvönen, J., Szabó, G., Lazer, D., Kaski, K., Kertśz, J., and Barabśi, A.-L. Structure And Tie Strengths In Mobile Communication Networks. *Proceedings of the National Academy of Sciences*, 104(18):7332-7336, 2007.

[58] Klein, J.P. and Moeschberger, M.L. *Survival Analysis: Techniques For Censored And Truncated Data*. Springer, 2003.

[59] Flinn, Jason and Satyanarayanan, M. Energy-aware Adaptation For Mobile Applications. *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, pages 48–63, ACM, New York, NY, USA, 1999.

[60] François, Jean-Marc, Leduc, Guy, and Martin, Sylvain. Learning Movement Patterns In Mobile Networks: A Generic Method. *European Wireless 2004*, page 128–134, 2004.

[61] Pattillo, Jeffrey, Youssef, Nataly, and Butenko, Sergiy. Clique Relaxation Models In Social Network Analysis. *Handbook of Optimization in Complex Networks*, pages 143–162, Springer, 2012.

[62] Yang, Jie, Varshavsky, Alexander, Liu, Hongbo, Chen, Yingying, and Gruteser, Marco. Accuracy Characterization Of Cell Tower Localization. *Proceedings of the 12th ACM international conference on Ubiquitous computing*, pages 223–226, 2010.

[63] Cleary, John and Witten, Ian. Data Compression Using Adaptive Coding And Partial String Matching. *IEEE transactions on Communications*, 32(4):396–402, IEEE, 1984.

[64] Casey Johnston. Bye-bye Unlimited 4g: Sprint Institutes Data Usage Caps. `http://arstechnica.com/gadgets/news/2011/10/bye-bye-unlimited-4g-sprint-institutes-data-usage-caps.ars`, October 2011.

[65] Dey, Anind K and Abowd, Gregory D. Cybreminder: A Context-aware System For Supporting Reminders. *Handheld and Ubiquitous Computing*, pages 172–186, 2000.

[66] M. Frans Kaashoek, Dawson R. Engler, Gregory R. Ganger, Héctor M. Briceño, Russell Hunt, David Mazières, Thomas Pinckney, Robert Grimm, John Jannotti, and Kenneth Mackenzie. Application Performance And Flexibility On Exokernel Systems. *Proceedings of the 16th Symposium on Operating Systems Principles*, 1997.

[67] Laasonen, Kari. Route Prediction From Cellular Data. *Workshop on Context-Awareness for Proactive Systems (CAPS), Helsinki, Finland*, volume 1617, 2005.

[68] Laasonen, Kari, Raento, Mika, and Toivonen, Hannu. Adaptive On-device Location Recognition. *Pervasive Computing*, volume 3001 of *Lecture Notes in Computer Science*, pages 287-304, Ferscha, Alois and Mattern, Friedemann, eds., Springer Berlin Heidelberg, 2004.

[69] James Kendrick. Verizon Tiered Data Plans Charge For Bits, Not Speed. http://www.zdnet.com/blog/mobile-news/verizon-tiered-data-plans-charge-for-bits-not-speed/2975, June 2011.

[70] Yadav, Kuldeep, Naik, Vinayak, Kumar, Abhishek, and Jassal, Prateek. Placemap: Discovering Human Places Of Interest Using Low-energy Location Interfaces On Mobile Phones. *Proceedings of the Fifth ACM Symposium on Computing for Development*, pages 93–102, 2014.

[71] Adamic, L. and Huberman, B. Zipf's Law And The Internet. *Glottometrics*, 3(1):143–150, 2002.

[72] Adamic, Lada. Zipf, Power-laws, And Pareto – A Ranking Tutorial. http://www.hpl.hp.com/research/idl/papers/ranking, 2000.

[73] Song, Libo, Kotz, David, Jain, Ravi, and He, Xiaoning. Evaluating Location Predictors With Extensive Wi-fi Mobility Data. *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 1414–1424, 2004.

[74] Yang Liu, Armin Sarabi, Jing Zhang, Parinaz Naghizadeh, Manish Karir, Michael Bailey, and Mingyan Liu. Cloudy With A Chance Of Breach: Forecasting Cyber Security Incidents. *24th USENIX Security Symposium (USENIX Security 15)*, pages 1009–1024, USENIX Association, Washington, D.C., August 2015.

[75] Stephen Lovely. 73% Of Subscribers Would Download Netflix Content. https://www.allflicks.net/73-of-subscribers-would-download-netflix-content/, July 2016.

[76] Bishop, Christopher M. *Pattern Recognition And Machine Learning*. Springer-Verlag New York, Inc., 2006.

[77] Gregor Maier, Fabian Schneider, and Anja Feldmann. A First Look At Mobile Hand-held Device Traffic. *International Conference on Passive and Active Measurement*, pages 161–170, Springer-Verlag, Berlin, Heidelberg, April 2010.

[78] Gupta, Manu, Intille, Stephen S, and Larson, Kent. Adding Gps-control To Traditional Thermostats: An Exploration Of Potential Energy Savings And Design Challenges. *Pervasive Computing*, pages 95–114, Springer, 2009.

[79] Yves Marcoz. Feedingit. http://feedingit.marcoz.org/. Source code: https://garage.maemo.org/projects/feedingit.

[80] Korkea-Aho, Mari. Context-aware Applications Survey. *Department of Computer Science, Helsinki University of Technology*, 2000.

[81] Newman, Mark. *Networks: An Introduction*. Oxford University Press, Inc., New York, NY, USA, 2010.

[82] Baldauf, Matthias, Dustdar, Schahram, and Rosenberg, Florian. A Survey On Context-aware Systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, Inderscience Publishers, 2007.

[83] Jonathan Mayer and Patrick Mutchler. metaphone: The Sensitivity Of Telephone Metadata. http://webpolicy.org/2014/03/12/metaphone-the-sensitivity-of-telephone-metadata/, March 2014.

[84] Rodriguez-Sanchez, MC, Martinez-Romo, J, Borromeo, Susana, and Hernandez-Tamames, JA. Gat: Platform For Automatic Context-aware Mobile Services For M-tourism. *Expert Systems with Applications*, 40(10):4154–4163, Elsevier, 2013.

[85] Margaret A. McDowell, Cheryl D. Fryar, Cynthia L. Ogden, and Katherine M. Flegal. Anthropometric Reference Data For Children And Adults: United States, 2003–2006. http://www.cdc.gov/nchs/fastats/bodymeas.htm, http://www.cdc.gov/nchs/data/nhsr/nhsr010.pdf, *National Health Statistics Reports*, (10), October 2008.

[86] Kim, Minkyong, Kotz, David, and Kim, Songkuk. Extracting A Mobility Model From Real User Traces. *INFOCOM*, volume 6, pages 1–13, 2006.

[87] Schwartz, Mischa. *Mobile Wireless Communications*. Cambridge University Press, 2005.

[88] Tripathi, N.D., Reed, J.H., and VanLandingham, H.F. *Radio Resource Management In Cellular Systems*. Springer, 2001.

[89] Eagle, Nathan and Pentland, Alex S. Reality Mining: Sensing Complex Social Systems. *Personal Ubiquitous Comput.*, 10(4):255–268, Springer-Verlag, London, UK, UK, March 2006.

[90] Eagle, Nathan, Quinn, John A, and Clauset, Aaron. Methodologies For Continuous Cellular Tower Data Analysis. *Pervasive computing*, pages 342–353, Springer, 2009.

[91] Nethercote, Nicholas and Seward, Julian. Valgrind: A Framework For Heavyweight Dynamic Binary Instrumentation. *Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, pages 89–100, ACM, New York, NY, USA, 2007.

[92] Jakob Nielsen. *Usability Engineering.* Morgan Kaufmann, San Francisco, 1993.

[93] Banerjee, Nilanjan, Rahmati, Ahmad, Corner, Mark D., Rollins, Sami, and Zhong, Lin. Users And Batteries: Interactions And Adaptive Energy Management In Mobile Systems. *Proceedings of the 9th International Conference on Ubiquitous Computing*, pages 217–234, Springer-Verlag, Berlin, Heidelberg, 2007.

[94] Ravi, Nishkam, Scott, James, Han, Lu, and Iftode, Liviu. Context-aware Battery Management For Mobile Phones. *IEEE International Conference on Pervasive Computing and Communications*, pages 224-233, March 2008.

[95] Nokia. Modest. `http://modest.garage.maemo.org/`. Source code: `http://gitorious.org/modest/`.

[96] Paddy O'Donnell and Stephen W. Draper. How Machine Delays Change User Strategies. *SIGCHI Bull.*, 28:39–42, ACM, New York, NY, USA, April 1996.

[97] Stefano Patti, Elia Biganzoli, and Patrizia Boracchi. Review Of The Maximum Likelihood Functions For Right Censored Data: A New Elementary Derivation. *COBRA Preprint Series*, page 1–10, Berkeley Electronic Press, May 2007.

[98] Thomas Perl. Gpodder. `http://gpodder.org/`. Source code: `https://github.com/gpodder/gpodder`.

[99] Zeng, Qing-An and Agrawal, Dharma P. Handoff In Wireless Mobile Networks. *Handbook of Wireless Networks and Mobile Computing*, pages 1–25, John Wiley & Sons, Inc., New York, NY, USA, 2002.

[100] Fleming, Thomas R., O'Fallon, Judith R., O'Brien, Peter C., and Harrington, David P. Modified Kolmogorov-smirnov Test Procedures With Application To Arbitrarily Right-censored Data. *Biometrics*, 36(4):607–625, December 1980.

[101] Loader, Clive R. Bandwidth Selection: Classical Or Plug-in? *The Annals of Statistics*, 27(2):415–438, 1999.

[102] Peter Romirer-Maierhofer, Fabio Ricciato, Alessandro D'Alconzo, Robert Franzan, and Wolfgang Karner. Network-wide Measurements Of Tcp Rtt In 3g. *Proceedings of the First International Workshop on Traffic Monitoring and Analysis*, pages 17–25, Springer-Verlag, Berlin, Heidelberg, 2009.

[103] Boslaugh, S. and Watters, P. *Statistics In A Nutshell: A Desktop Quick Reference*. O'Reilly Media, Incorporated, 2008.

[104] Trivedi, Kishor S. *Probability And Statistics With Reliability, Queuing And Computer Science Applications*. John Wiley & Sons, 2002.

[105] Scellato, Salvatore, Musolesi, Mirco, Mascolo, Cecilia, Latora, Vito, and Campbell, Andrew T. nextplace: A Spatio-temporal Prediction Framework For Pervasive Systems. *Pervasive computing*, pages 152–169, Springer, 2011.

[106] Aaron Schulman, Vishnu Navda, Ramachandran Ramjee, Neil Spring, Pralhad Deshpande, Calvin Grunewald, Kamal Jain, and Venkata N. Padmanabhan. Bartendr: A Practical Approach To Energy-aware Cellular Data Scheduling. *International Conference on Mobile Computing and Networking*, pages 85–96, ACM, New York, NY, USA, September 2010.

[107] Yun, Se-Young, Lelarge, Marc, and Proutiere, Alexandre. Streaming, Memory Limited Algorithms For Community Detection. *NIPS*, pages 3167–3175, 2014.

[108] Isaacman, Sibren, Becker, Richard, Cáceres, Ramón, Kobourov, Stephen, Rowland, James, and Varshavsky, Alexander. A Tale Of Two Cities. *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, pages 19–24, 2010.

[109] Asia Slowinska and Herbert Bos. Pointless Tainting? Evaluating The Practicality Of Pointer Tainting. *Proceedings of ACM SIGOPS EUROSYS*, Nuremberg, Germany, March-April 2009.

[110] SQLite. Frequently Asked Questions - insert Is Really Slow - i Can Only Do Few Dozen inserts Per Second. `https://sqlite.org/faq.html#q19`.

[111] U.S. Bureau of Labor Statistics. Table 7. Beginning And Ending Hours: Full-time Wage And Salary Workers, May 2004. 2016. [Online; accessed 28-July-2016].

[112] Wagner, Daniel T, Rice, Andrew, and Beresford, Alastair R. Device Analyzer: Understanding Smartphone Usage. *Mobile and Ubiquitous Systems*, pages 195–208, Springer, 2013.

[113] Wagner, Daniel T, Rice, Andrew, and Beresford, Alastair R. Device Analyzer: Large-scale Mobile Data Collection. *ACM SIGMETRICS Performance Evaluation Review*, 41(4):53–56, ACM, 2014.

[114] T-Mobile. T-mobile Unveils Affordable And Worry-free Unlimited Data Plans. `http://newsroom.t-mobile.com/articles/t-mobile-unveils-unlimited-data-plans`, October 2011.

[115] Singal, T.L. *Wireless Communications*. McGraw-Hill Education, 2010.

[116] Sohn, Timothy, Varshavsky, Alex, LaMarca, Anthony, Chen, Mike Y, Choudhury, Tanzeem, Smith, Ian, Consolvo, Sunny, Hightower, Jeffrey, Griswold, William G, and De Lara, Eyal. Mobility Detection Using Everyday gsm Traces. *International Conference on Ubiquitous Computing*, pages 212–224, 2006.

[117] De Pessemier, Toon, Dooms, Simon, and Martens, Luc. Context-aware Recommendations Through Context And Activity Recognition In A Mobile Environment. *Multimedia Tools and Applications*, 72(3):2925–2948, 2014.

[118] Ionut Trestian, Supranamaya Ranjan, Aleksandar Kuzmanovic, and Antonio Nucci. Measuring Serendipity: Connecting People, Locations And Interests In A Mobile 3g Network. *ACM SIGCOMM Internet Measurement Conference*, pages 267–279, ACM, New York, NY, USA, November 2009.

[119] van der Vaart, Aad W. Maximum Likelihood Estimation With Partially Censored Observations. *Annals of Statistics*, 22(4):1896–1916, 1994.

[120] Etter, Vincent, Kafsi, Mohamed, Kazemi, Ehsan, Grossglauser, Matthias, and Thiran, Patrick. Where To Go From Here? Mobility Prediction From Instantaneous Information. *Pervasive and Mobile Computing*, 9(6):784–797, Elsevier, 2013.

[121] Vasiliev, Vlad and Fialko, Pavel. Omweather. `https://garage.maemo.org/projects/omweather/`.

[122] Bowman, Adrian W. An Alternative Method Of Cross-validation For The Smoothing Of Density Estimates. *Biometrika*, 71(2):353–360, 1984.

[123] Venables, W.N. and Ripley, B.D. *Modern Applied Statistics With S*. Springer Verlag, 2002.

[124] Neal H. Walfield. An Architecture For Opportunistic Data Transfers. Johns Hopkins University, http://hssl.cs.jhu.edu/~neal/walfield16mobicom-rejection.pdf, 2016.

[125] Neal H. Walfield. n900 Smartphone Traces. Johns Hopkins University, http://hssl.cs.jhu.edu/~neal/woodchuck/user-study.pdf, 2016.

[126] Neal H. Walfield and Christian Grothoff. Aggregating Cell Towers For Location-based Analysis. Johns Hopkins University, http://hssl.cs.jhu.edu/~neal/walfield16mobicase-rejection.pdf, 2016.

[127] Neal H. Walfield and Christian Grothoff. tomorrowtoday: gsm-based Location Prediction. Johns Hopkins University, http://hssl.cs.jhu.edu/~neal/walfield16percom-submission.pdf, 2016.

[128] Neal H. Walfield, John Linwood Griffin, and Christian Grothoff. A Quantitative Analysis Of Cell Tower Trace Data For Understanding Human Mobility And Mobile Networks. *Proceedings of the 6th International Workshop on Mobile Entity Localization, Tracking and Analysis*, 2016.

[129] David A. Wheeler. Sloccount. http://www.dwheeler.com/sloccount/.

[130] Wikipedia. nitz — Wikipedia, The Free Encyclopedia. 2014. [Online; accessed 24-November-2014].

[131] Wikipedia. Workweek And Weekend — Wikipedia, The Free Encyclopedia. 2016. [Online; accessed 20-July-2016].

[132] Wang, Xinxi, Rosenblum, David, and Wang, Ye. Context-aware Mobile Music Recommendation For Daily Activities. *Proceedings of the 20th ACM international conference on Multimedia*, pages 99–108, 2012.

[133] Chen, Mike Y and others. Practical Metropolitan-scale Positioning For gsm Phones. *UbiComp 2006: Ubiquitous Computing*, pages 225–242, Springer, 2006.

[134] Chon, Yohan, Shin, Hyojeong, Talipov, Elmurod, and Cha, Hojung. Evaluating Mobility Models For Temporal Prediction With High-granularity Mobility Data. *Pervasive Computing and Communications (PerCom), 2012 IEEE International Conference on*, pages 206–212, 2012.

[135] Chon, Yohan, Ryu, Wanchang, and Cha, Hojung. Predicting Smartphone Battery Usage Using Cell Tower id Monitoring. *Pervasive and Mobile Computing*, 13:99–110, Elsevier, 2014.

Index

Biography

Neal H. Walfield was born in 1980 in the USA.

Neal did his undergraduate work at the University of Massachusetts, Lowell where he majored in Computer Science and English Literature, and minored in Mathematics and French. During his undergraduate studies, he spent a year at the Sorbonne studying French, and a summer at the University of Karlsruhe working on a port of the Hurd to L4. His paper "A Critique of the Hurd" (with Marcus Brinkmann) was partially a result of that visit. Neal also taught the first semester Computer Science Recitation course several times, and advised a French independent study.

In 2006, Neal joined Hermann Härtig's OS group at TU Dresden. During this time he work on his experimental microkernel Viengoos, and founded and led the local paper reading group.

In 2008, Neal began his PhD at Johns Hopkins University. He was a teaching assistant for Randal Burns' Parallel Computing class.