

## ABSTRACT

Title of Document: **OBSERVING AND IMPROVING THE RELIABILITY OF INTERNET LAST-MILE LINKS**

Aaron David Schulman, Doctor of Philosophy, 2013

Directed By: **Professor Neil Spring**  
Department of Computer Science

People rely on having persistent Internet connectivity from their homes and mobile devices. However, unlike links in the core of the Internet, the links that connect people's homes and mobile devices, known as "last-mile" links, are not redundant. As a result, the reliability of any given link is of paramount concern: when last-mile links fail, people can be completely disconnected from the Internet.

In addition to lacking redundancy, Internet last-mile links are vulnerable to failure. Such links can fail because the cables and equipment that make up last-mile links are exposed to the elements; for example, weather can cause tree limbs to fall on overhead cables, and flooding can destroy underground equipment. They can also fail, eventually, because cellular last-mile links can drain a smartphone's battery if an application tries to communicate when signal strength is weak.

In this dissertation, I defend the following thesis: *By building on existing infrastructure, it is possible to (1) observe the reliability of Internet last-mile links across different weather conditions and link types; (2) improve the energy efficiency of cellular Internet last-mile links; and (3) provide an incrementally deployable, energy-efficient Internet*

*last-mile downlink that is highly resilient to weather-related failures.* I defend this thesis by designing, implementing, and evaluating systems.

First, I study the reliability of last-mile links during weather events. To observe failures of last-mile links, I develop ThunderPing—a system that monitors a geographically diverse set of last-mile links without participation from providers or customers. So far, ThunderPing has collected 4 billion pings from 3.5 million IP addresses over 400 days of probing from PlanetLab hosts. Because pings may fail to solicit a response even when a last-mile link has not failed, losses must be analyzed to determine if they constitute last-mile link failures. Among other challenges I encountered in this project, I found that determining the connectivity state from noisy pings is similar to finding the edges in a noisy picture. As such, I use algorithmic edge detection to find when a host transitions between connectivity states. By matching these connectivity states with weather reports from weather stations at airports, I observe how weather affects last-mile link failure rate and failure duration.

Second, I improve the reliability of cellular links by reducing wasted energy. To do so, I develop Bartendr, a system that predicts when a moving smartphone will experience high signal strength. A key challenge is to predict high signal strength without consuming more energy than exploiting it would save. I also develop energy-aware scheduling algorithms for different application workloads—syncing and streaming—based on these predictions. I evaluate the scheduling algorithms with a simulation driven by traces obtained during actual drives.

Third, I design a reliable broadcast system that is inexpensive to deploy to many users and is energy-efficient to receive. I adapt reliable FM Radio Data System (RDS) broad-

casts to act as an Internet last-mile link. To accomplish this, I design and implement an over-the-air protocol, receiver software, and a hardware bridge for incremental deployment. I implement the full end-to-end system, deploy it on a 3 kW commercial FM radio station in a metropolitan area, and evaluate the loss rate, energy consumption, and synchronization on either a smartphone or on my new hardware bridge. The results indicate that the full end-to-end system can be reliable, a smartphone receiver can sleep between desired broadcasts, and two receivers tend to deliver the same broadcast within about 5 ms.

OBSERVING AND IMPROVING THE RELIABILITY OF INTERNET  
LAST-MILE LINKS

by

Aaron David Schulman

Dissertation submitted to the Faculty of the Graduate School of the  
University of Maryland, College Park in partial fulfillment  
of the requirements for the degree of  
Doctor of Philosophy  
2013

Advisory Committee:  
Professor Neil Spring, Chair/Advisor  
Professor Bobby Bhattacharjee  
Professor Michael Hicks  
Professor Prabal Dutta  
Professor Mark Shayman

© Copyright by  
Aaron David Schulman  
2013

# Acknowledgments

I am truly grateful to all those who helped me complete this dissertation. If I neglected to mention you, I apologize. Please know that I greatly appreciate your support.

Over the course of undergraduate and graduate school, my advisor, Neil Spring, inspired me to pursue an academic career doing networking research. Along the way, he developed my thinking and gave me the skills to succeed in doing such research. Neil significantly raised my expectations of writing and presentation quality: he helped me realize what I was capable of and gave me a strong set of criteria to evaluate my own work and that of others. He gave me the confidence to investigate hypotheses that are difficult to test. He taught me how to create graphs that present data in a novel way, but are still intuitive and clear. Throughout, Neil was himself willing to learn new skills in order to work with me on the research problems that I was passionate about.

Ever since I met him when I started graduate school, Dave Levin has been one of my closest mentors. Any time I had a question, day or night, I knew Dave would eagerly answer his phone and work through the problem with me. As a collaborator, Dave taught me skills in every area of graduate school. For example, he taught me how to write explicitly, diagram beautifully, and cook deliciously.

Bobby Bhattacharjee has been an inspiring collaborator. He taught me the importance of color selection in my presentations. He also gave me the opportunity to teach embedded systems design to an amazing group of undergrads.

From the first moment I heard him speak, Prabal Dutta always asked the tough questions. Prabal gave me the opportunity to spend a year and a half with him and his excellent students in Michigan learn electrical engineering while spending a year and half of graduate school with my (now) wife. He also introduced me to Thomas Schmid. In the few months I spent with Thomas, he taught basic electrical engineering skills such as soldering circuit boards and advanced skills such as designing embedded systems.

My wife, Ophira Vishkin, taught me how to be precise and never let me doubt my ability. Throughout graduate school, she has been a source of serenity and brilliant editing.

I thank Mark Shayman and Michael Hicks for giving me detailed feedback on this dissertation. Their comments significantly improved the quality of it.

The Lab<sub>1</sub>: Randy Baden, Adam Bender, Dave Levin, Cristian Lumezanu, Rob Sherwood, Bo Han, and Justin McCann patiently taught me the ropes of being a graduate student. *Everybody* knows that I will never forget the lessons they taught me.

The Lab<sub>2</sub>: Yunus Basagalar, Vassilios Lekakis, Matt Lentz, Youndo Lee, and Ramakrishna Padmanabhan have been great friends and even better collaborators. They provided

support and encouragement during my proposal and defense.

One weekend, Michelle Hugue (Meesh) asked me to drive out from Annapolis to College Park so she could convince me to go to graduate school for a PhD (rather than a Master's). She also told me to email the talented young professor who eventually became my advisor.

During my summer at Microsoft Research India, my collaborators, Vishnu Navda, Ramachandran Ramjee, and Venkat Padmanabhan, were great mentors and collaborators. They gave me the opportunity to do research while learning about Indian culture.

Wyn Bennett taught me how to program in Ms. Jellison's computer science class and worked with me on the two science fair projects that whet my appetite for research. Brandi Adams provided me with excellent writing advice and gave me a reason to play with microphones. Katrina LaCurts has been a great collaborator and is the world's best overseas conference roommate. My friends Dan Noble, Aaron Silverman, Chris Testa, Josh Handelman, and Patrick Shoemaker always made sure I was enjoying research. I am continually inspired by their creativity and work ethic.

My parents gave me unconditional support, love, and encouragement. In addition, my father gave me a passion for learning how things work and my mother taught me how to speak in front of an audience.

This dissertation was supported by NSF Awards CNS-0917098, CNS-0643443, and CNS-0626629.

Finally, I would like to explicitly acknowledge the people that I collaborated with on the work in this dissertation.

*Chapter 2:* My collaborators were Youndo Lee, Ramakrishna Padmanabhan, and Neil Spring. Additionally, Patrick Shoemaker and Dave Levin provided helpful discussions. Also thanks to the anonymous reviewers and our shepherd Fabiàn Bustamante for their comments.

*Chapter 3:* My collaborators were Vishnu Navda, Ramachandran Ramjee, Neil Spring, Pralhad Deshpande, Calvin Grunewald, Kamal Jain, and Venkata N. Padmanabhan. Additionally, our shepherd Prasun Sinha, the anonymous reviewers, and the summer 2009 interns of MSR India provided insightful comments and discussion. I extend my gratitude to the cab drivers at Microsoft Research India for their assistance with the data collection.

*Chapter 4:* My collaborators were Dave Levin, Neil Spring. Additionally, Prabal Dutta and Thomas Schmid provided many useful discussions.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Challenges . . . . .	4
1.2	Thesis . . . . .	6
1.3	Insights . . . . .	8
1.4	Contributions . . . . .	10
<b>2</b>	<b>Background and Related Work</b>	<b>13</b>
2.1	Observing last-mile link failures . . . . .	14
2.1.1	With privileged data . . . . .	14
2.1.2	Without privileged data . . . . .	15
2.2	Improving the energy efficiency of smartphone communication . . . . .	17
2.2.1	Radio energy . . . . .	17
2.2.2	Processing and positioning energy . . . . .	22
2.2.3	Energy-efficient cellular data scheduling . . . . .	23
2.3	Improving reliability with data broadcasting . . . . .	24
2.3.1	Datacasting systems . . . . .	25
2.3.2	Broadcast technologies . . . . .	27
2.4	Summary . . . . .	30
<b>3</b>	<b>Weather-related Last-mile Link Failures</b>	<b>31</b>
3.1	Measuring the responsiveness of Internet hosts during weather . . . . .	33
3.1.1	Finding IP addresses subject to weather . . . . .	33
3.1.2	Pinging (last-mile links) in the rain . . . . .	36
3.1.3	Potential sources of error . . . . .	37
3.2	Inferring link-state from pings . . . . .	37
3.2.1	Filtering out intermittent PlanetLab node failures . . . . .	39
3.2.2	Detecting failures with conditional probabilities . . . . .	41
3.2.3	Detecting changes in prolonged loss rate . . . . .	44
3.2.4	Understanding the HOSED state . . . . .	48
3.3	Weather history and link type . . . . .	50
3.3.1	The weather at a host during a ping . . . . .	50
3.3.2	Identifying the link type of an IP address . . . . .	51
3.4	Failure rate . . . . .	52
3.4.1	UP to DOWN failures . . . . .	53
3.4.2	UP to HOSED failures . . . . .	57



3.5	Failure duration . . . . .	58
3.5.1	Does ThunderPing observe the duration of failures? . . . . .	58
3.5.2	How long do <b>UP</b> → <b>DOWN</b> failures last? . . . . .	62
3.6	Summary . . . . .	65
<b>4</b>	<b>Energy-aware Cellular Data Scheduling</b>	<b>67</b>
4.1	Motivation . . . . .	69
4.1.1	Signal varies by location . . . . .	70
4.2	Suitable applications . . . . .	72
4.2.1	Synchronization . . . . .	72
4.2.2	Streaming . . . . .	75
4.3	Architecture . . . . .	77
4.3.1	Predicting signal with signal tracks . . . . .	77
4.3.2	Scheduling sync . . . . .	81
4.3.3	Scheduling streaming . . . . .	82
4.4	Simulation-based evaluation . . . . .	85
4.4.1	Syncing . . . . .	86
4.4.2	Streaming . . . . .	88
4.5	Related work . . . . .	90
4.5.1	Predicting wireless network quality . . . . .	90
4.5.2	Stability of cellular signals . . . . .	91
4.6	Summary . . . . .	91
<b>5</b>	<b>Reliable Broadcast Last-mile Link</b>	<b>93</b>
5.1	The reliability of FM RDS metropolitan radio broadcasting . . . . .	95
5.1.1	FM RDS loss rate is low . . . . .	97
5.1.2	Where there are people, there are FM towers . . . . .	99
5.1.3	Every transmission can be stored . . . . .	102
5.1.4	FM Receivers are small . . . . .	102
5.2	Design and implementation of Abbie . . . . .	103
5.2.1	Tower sharing protocol . . . . .	104
5.2.2	Over-the-air protocol . . . . .	104
5.2.3	RDS-to-LAN bridge . . . . .	107
5.2.4	Receiver software . . . . .	108
5.3	Applications . . . . .	110
5.3.1	Push: DNS cache updates . . . . .	110
5.3.2	Anonymous and synchronous: mass reset . . . . .	112
5.4	Evaluation . . . . .	113
5.4.1	Metropolitan deployment . . . . .	113
5.4.2	Abbie's energy consumption on an Android phone . . . . .	115
5.4.3	Abbie end-to-end synchronization . . . . .	116
5.4.4	RDS receiver cold boot . . . . .	117
5.5	Summary . . . . .	117

<b>6</b>	<b>Conclusions and Open Questions</b>	<b>119</b>
6.1	Longevity . . . . .	121
6.2	Open questions . . . . .	123

# List of Tables

2.1	Smartphones used for power and throughput measurements . . . . .	17
2.2	Candidate datacasting systems for an Internet last-mile link . . . . .	28
3.1	Summary of a small portion of the data collected by ThunderPing . . . . .	53
4.1	Signal strength and energy consumption while playing a YouTube video . . . . .	75
5.1	Abbie prototype receiver locations in the testbed deployment . . . . .	114

# List of Figures

2.1	Signal strength affects power consumption . . . . .	19
2.2	Signal strength affects throughput . . . . .	20
2.3	Cellular radio tail energy . . . . .	21
2.4	Smartphone wakeup and suspend energy . . . . .	22
3.1	Example of an XML weather alert . . . . .	34
3.2	Average loss rate and “dodgy” PlanetLab node loss rates . . . . .	39
3.3	Detail view of a period where PlanetLab nodes were declared “dodgy.” . .	40
3.4	The conditional probability of ping loss for each link type . . . . .	42
3.5	Example of the edge detection algorithm that separates <b>UP</b> and <b>HOSED</b> .	44
3.6	The edge detector’s accuracy of <b>HOSED</b> state detection . . . . .	46
3.7	Example of the moving average failing to separate <b>UP</b> and <b>HOSED</b> . . . .	48
3.8	The distribution of loss rates is related to last-mile link type . . . . .	49
3.9	The distribution of link types for IP addresses pinged by ThunderPing . .	52
3.10	<b>UP</b> → <b>DOWN</b> failure rate for different weather conditions . . . . .	53
3.11	Comparison of the number of five-minute intervals that see failures across multiple ISPs within a US state during a power outage . . . . .	54
3.12	<b>UP</b> → <b>DOWN</b> failure rate excluding suspected power outages . . . . .	57
3.13	<b>UP</b> → <b>HOSED</b> failure rate for different weather conditions . . . . .	57
3.14	Failures and address reassignments in BISMMark . . . . .	63
3.15	Duration <b>DOWN</b> after an <b>UP</b> → <b>DOWN</b> transition . . . . .	63
4.1	Signal strength varies by location . . . . .	70
4.2	Signal strength variations are consistent over 6 drives . . . . .	71
4.3	Average signal strength and energy for email syncs . . . . .	73
4.4	Signal strength prediction error for all 25 m steps in six 17 km tracks . . .	79
4.5	Tracks from opposite directions may not align in signal strength . . . . .	80
4.6	Signal correlation of 25 m steps in all pairs of “from” tracks with “to” tracks	80
4.7	Predicting signal with previous traces can save energy for email syncs . .	86
4.8	Energy savings with signal-based scheduling for data streams . . . . .	89
5.1	Loss rate of RDS messages from a 3 kW FM transmitter . . . . .	98
5.2	The FCC FM signal contours and Census county population estimates . .	99
5.3	The number of people covered by each of the FM transmitters in the US .	100
5.4	Abbie system overview . . . . .	103
5.5	Abbie package structure . . . . .	105

5.6	RDS-to-LAN bridge with integrated antenna . . . . .	107
5.7	Broadcasting packages to four Abbie RDS-to-LAN bridges . . . . .	113
5.8	The energy consumption of the Abbie receiver running on a Samsung Galaxy S smartphone . . . . .	115
5.9	The synchronization of two RDS-to-LAN bridges . . . . .	116
5.10	Time from powered off to receiving first RDS message . . . . .	117

# Chapter 1

## Introduction

The Internet links that connect to homes and mobile devices are called *last-mile links*. When last-mile links are not available, the effectiveness of government services is undermined, the profits of businesses are diminished, and the flexibility of the workforce is lost. However, unlike links in the core of the Internet, last-mile links are typically not redundant. As a result, the reliability of any single link is of paramount concern: when last-mile links fail, people can be completely disconnected from the Internet.

The more last-mile links fail, the less reliable the applications are that depend on them. Yet, governments and businesses increasingly assume that people can access the Internet while they are at home and while they are mobile.

Today, governments rely on last-mile links for emergency communication; in the future, they may rely on these links for online voting. To improve citizen safety, the United States government mandates that home and mobile Internet services provide emergency systems such as the 9-1-1 [32] phone number and the Television and Radio emergency

alert system [31].<sup>1,2</sup> The less time last-mile links are available, the higher the chance citizens miss an emergency alert or are caught in an emergency situation without the ability to contact emergency services. To improve the efficiency and accessibility of voting, governments are developing online voting systems [61]. Online voting benefits from voters being able to cast their ballots while they are at home and mobile. A large enough outage of last-mile links may even result in a voided election [35].

Businesses use last-mile links to advertise to consumers, to provide consumers with streaming media services, and to connect with their workers. A more available Internet connection allows a user to view more ads; conversely, a dead battery or failed link limits ad views. As a result, the reliability of last-mile links may eventually limit the growth of Internet ad revenue. In 2012, overall online advertising revenue grew from \$31.7 to \$36.6 billion, and the percentage of that total from mobile advertising grew even more, from 5% to 9% [41]. Streaming media businesses also benefit from last-mile link reliability. These businesses require users to trade off ownership of media for a subscription to stream it; if consumers cannot access the content when they want it, they may not be willing to make this trade-off. Finally, as more businesses encourage their employees to telework (work from home) [96], they shift their reliance from Internet transit links to their employees' last-mile links.

These observations reveal that last-mile links are increasingly depended upon, but the stark reality is that they are not reliable. Rather, last-mile links are subject to external

---

<sup>1</sup>“Triple play” home Internet services—which also provide telephone and TV—are becoming more popular [89]. Citizens with triple play depend on their Internet link for both 9-1-1 and emergency alerts.

<sup>2</sup>The US Wireless Emergency Alert system started in April 2012. The government does not yet require cellular providers to broadcast emergency alerts. However, all of the major cellular providers voluntarily broadcast some of the alerts [21].

factors that can cause physical failures. last-mile links are made up of multiple technologies that collectively span approximately one mile of transmission media and supporting equipment between the Internet Service Provider's (ISP's) facility and customer's premises—failures may occur at virtually any point along the way.

In turn, there are many different sources of failures, both natural and man-made. *Weather* can attenuate the signals in outdoor transmission media, which can disable the link [8, 40, 44, 90]. In the extreme case, weather can even destroy equipment [3]. Communicating in *poor signal strength* can waste the limited energy in a smartphone's battery [24, 85]. Fixed last-mile links depend on electric power infrastructure to power mid-link equipment such as neighborhood media converters and end-link equipment such as the customer's modem—*power outages* can cause any of these pieces of equipment to fail. ISP technicians can *misconfigure* software parameters such as modulation rate, which can lead to link instability [45]. Also, when there is a *crowded event* smartphone traffic can exceed mobile link resources, leading to instability and outages [88].

Researchers know that weather-related failures and wasted energy can reduce last-mile link reliability. However, we do not know the prominence and prevalence of these harmful effects, nor how to diminish them. If we better understood weather-related failures, we could work around them by deploying new, more resilient types of links. Similarly, a better understanding of energy consumption of mobile links could allow us to diminish wasted energy by changing how applications use the links.



## 1.1 Challenges

It is challenging to observe the reliability of last-mile links; precise measurement studies have, to date, required privileged access to data, and making general inferences about link reliability is complicated by the fact that there are diverse link types deployed across diverse geographic locations. It is also challenging to lessen or work around link failures because the diverse types of links are difficult to modify en masse, and new links are expensive to deploy. Before stating my thesis, I describe these challenges in more detail, and identify several criteria that I believe must be upheld in order to make general observations of and improvements to last-mile link reliability.

It is difficult to observe and improve last-mile link reliability without privileged data, that is, statistics taken directly from last-mile link equipment. Even ISPs themselves find it difficult to observe the physical properties of their own last-mile links; AT&T, for instance, only observes some of the physical link properties of their customers once a week because of resource constraints [45]. Observing last-mile link availability across multiple ISPs, and controlling *when* these observations are collected, requires data that does not come from a privileged source. Fortunately, there exist some popular tools (most notably ICMP “pings” [73]) that allow those without access to privileged data to remotely probe some last-mile links.

Unfortunately, observing the physical operation of cellular radios is far more limited without privileged data. The only physical interface that is available across most phones is a limited set of statistics: signal strength and the identification number of the associated cellular tower. Without privileged access, any improvements to cellular reliability must

only require that interface and no other control over the radio's behavior.

The diversity of last-mile link types, and the relative age of their deployments, make it difficult to observe when a link fails. There is no standard loss rate for all deployed link types; some link types exhibit a persistent, low loss rate, while, for others, switching to a lossy state indicates a failure. These differences arise from the diversity in the transmission media and equipment that comprise last-mile links. For example, some last-mile links are built on repurposed transmission media (e.g., telephone subscriber loops and CATV cables); others have dedicated transmission media (e.g., Fiber To The Home, fixed terrestrial wireless, and satellite links). Any general conclusions about last-mile link reliability must be based on data from a variety of link types.

Environmental conditions vary significantly across different locations. Weather is clearly location-dependent; snow is far less likely in Texas than in Alaska. Operators therefore protect their transmission media and equipment in a manner that considers the weather conditions in their area. For instance, in cold regions, operators commonly deploy snow protection systems such as antenna heaters and falling ice shields. As a result, the same kind of weather event in two different areas may have profoundly different effects on last-mile link reliability. Any general conclusions about last-mile link reliability must include observations from a variety of locations.

Deployment density can also vary across different locations. Cellular ISPs do not place their cellular towers equidistant from one another: they densely deploy in cities and sparsely in rural areas [69]. Improvements to reliability should account for variations in deployment density.

A natural means of improving reliability is to add redundancy. However, it is chal-

lenging to deploy new last-mile links; simply put, new links are expensive. There is a history of ISPs going bankrupt, or giving up, as they try to deploy new last-mile links. One of the earliest cellular last-mile ISPs, Metricom's Ricochet, went into bankruptcy during their deployment phase [12]. Recently, large deployments of Fiber-To-The-Home (FTTH) last-mile links have encountered deployment problems: Verizon's FiOS FTTH deployment took six years and they stopped deploying to new cities in 2010 due to financial constraints, leaving one third of their telephone service area without fiber [93]. Therefore, improving the reliability of last-mile links should not require deploying expensive links.

In summary, making general observations about last-mile link reliability should require no privileged data, and such observations should cover a large number of links that span a variety of link types and locations. Also, improving last-mile link reliability should not require modifying existing infrastructure or deploying new, expensive links.

## **1.2 Thesis**

In this dissertation, I observe and improve the reliability of Internet last-mile links by building on existing infrastructure. Specifically, I build off of unmodified transmission media and equipment, and I collect data only from public measurement infrastructure. Operating under these constraints, my system that infers last-mile link failures is independent of link technology and network topology; my improvements to cellular energy consumption are independent of network technology and smartphone device; and my system that provides exceptional reliability does not require the deployment of new last-mile links.

I defend the following thesis: *By building on existing infrastructure, it is possible to (1) observe the reliability of Internet last-mile links across different weather conditions and link types; (2) improve the energy efficiency of cellular Internet last-mile links; and (3) provide an incrementally deployable, energy-efficient Internet last-mile downlink that is highly resilient to weather-related failures.*

I study two sources of last-mile link failures in particular—weather and wasted energy—and three types of last-mile links—fixed, cellular mobile, and VHF broadcast.

Weather can cause fixed last-mile transmission media to fail [84]. For example, early telephone and Community Antenna Television (CATV) engineering documents describe how to avoid moisture in wires because it impedes signal propagation [44, 90]. Also, rain attenuates satellite signals above 10 GHz [40]. Finally, point-to-point wireless links can experience multipath fading due to objects moving in the wind [8].

Mobile links are less susceptible to link failures since a mobile user could move if necessary. However, they are subject to another, more insidious, reliability constraint: mobile last-mile links drain smartphone batteries when signal strength is low, limiting the availability of the phone for future communication [24, 85]. Communication that could be delayed until the signal strength improves, instead unnecessarily wastes energy.

Such sensitivity to environmental factors is not endemic to all last-mile links. Metropolitan-scale radio broadcast is resilient to weather and consumes little energy, but it must be adapted to function as an Internet last-mile link. The Very High Frequency (VHF) radio transmission medium propagates well and is not significantly affected by weather. Although lightning strikes between the transmitter and receiver adds noise to VHF signals, this effect is mitigated by using FM (Frequency Modulation) [58]. Also, VHF broad-

casts can be received by even the most energy restricted receivers, such as hand crank radios [47]. The robustness of metropolitan scale radio broadcasting is evidenced by the US government's choice of approximately 1000 VHF FM broadcast transmitters to make up NOAA's Weather Radio service [64]. NOAA Weather Radio transmissions have to be receivable during weather, even when there is a power outage and a citizen does not have fresh batteries.

In summary, I focus my study on failures that arise due to weather and wasted energy because I believe that they are unavoidable; weather and limited battery capacity will always threaten the reliability of last-mile links. I study three broad classes of last-mile links: fixed and cellular mobile links because they constitute nearly all Internet last-mile links, and VHF broadcast because its reliability is unlike that of any existing Internet last-mile link.

### **1.3 Insights**

My general insight that allows me to defend my thesis is that *the availability of every last-mile link is associated with the availability of the last-mile links that physically surround them*. At a basic level, last-mile links in the same neighborhood share the same link through a Cable Modem Termination System (CMTS) or Digital Subscriber Line Access Multiplexer (DSLAM). At a deeper level, proximal last-mile links are also subjected to the same environmental effects, even if they are provided by different ISPs. For instance, when it rains in a city, the rain may affect all of the last-mile links in the city. In defending my thesis, I focus on three spatial relationships of last-mile links: fixed last-mile links affected by the same weather system, a mobile device's set of neighboring cellular towers,

and radio broadcast receivers covered by the same transmitter.

**Weather events affect many fixed last-mile links.** Weather systems range in size. They can cover an area ranging from as small as a US county (the area that the US issues weather alerts for [65]) to significant portions of the continental US (e.g., hurricanes). Instead of doing the impractical—continually observing every last-mile link—I focus my observations on regions that may experience adverse weather. Within the affected area, I observe a sample of hosts for all link types and all ISPs that exist in the area.

**Surrounding last-mile links affect mobile energy consumption.** While moving, cellular radios handoff to new towers when their signal strength with the current associated tower drops below a threshold. When the signal strength is low, the energy required by the radio to communicate is high [24, 85]. This drains the battery faster than when the signal strength is high. Instead of communicating when the signal strength is low, I predict when the moving radio will be close to the surrounding towers. Based on these predictions, I delay communication until the signal strength is high.

**Wireless broadcast receivers can increase reliability.** Metropolitan-scale reliable broadcast complements existing fixed and mobile last-mile links; it works when existing last-mile links fail. When broadcast fails, existing last-mile links can repair the error. FM broadcasting is a deployed, low-energy, weather-resistant last-mile link. VHF broadcasts, such as FM radio, are transmitted almost everywhere there are people in the world. When a receiver is close to the transmitter, they are unlikely to get any messages with errors. When a receiver is far from the transmitter, it may experience spontaneous errors. Even if it does, receivers that are near the transmitter can retransmit the correct message to the far receiver.

## 1.4 Contributions

Based on these insights, I design and implement systems to observe and improve the reliability of last-mile links. Specifically, I observe the reliability of fixed links in weather, observe and improve the reliability of mobile links that are moving between cellular towers, and increase fixed and mobile reliability by adapting a metropolitan radio broadcast link to the Internet.

**Chapter 2: An introduction to last-mile link reliability.** I describe the specific challenges that each last-mile link technology poses that makes it difficult to observe and improve the reliability of last-mile links. Additionally, I describe how related work addresses these challenges.

**Chapter 3: ThunderPing, a measurement system and analysis techniques for observing weather-related last-mile link failures *without privileged data from ISPs.*** I design and deploy a measurement tool called ThunderPing that measures the responsiveness of residential Internet hosts before, during, and after periods of severe weather forecast by the National Weather Service. ThunderPing uses several PlanetLab [70] hosts as vantage points to attempt to be resilient to routing faults and firewall filtering. ThunderPing collected 4 billion pings from 3.5 million IP addresses over 400 days of probing from PlanetLab hosts. The size and diversity of this data creates an opportunity to discover rare pathologies, but also creates substantial challenges in focusing on the last-mile link despite occasional faults in the broader network.

I analyze this ping data to determine when hosts lose connectivity, completely or partially, and categorize these failures by periods of weather ranging from clear skies to severe storms. Determining the connectivity state from noisy pings is similar to finding the edges in a noisy picture. As such, I use algorithmic edge detection to find when a host transitions between connectivity states.

**Chapter 4: Bartendr, a system for observing and improving the energy efficiency of cellular links *without modifying cellular hardware, firmware or drivers or accessing proprietary cellular data*.** I measure the relationship between signal strength and the energy consumption of popular applications. Then, I show that past observations of signal strength progression along a track can effectively predict signal strength in the future. Finally, I develop energy-aware scheduling algorithms for different workloads—syncing and streaming—and evaluate these via simulation driven by traces obtained during actual drives. I perform my experiments on four cellular networks across two large metropolitan areas, one in India and the other in the US.

**Chapter 5: Abbie, a system that adapts an existing radio broadcast last-mile link to the Internet *without modifying the link's hardware or software*.** I design and implement a reliable broadcast primitive over the FM Radio Data System (RDS) that uses a Distributed Hash Table (DHT) of last resort for retransmission of missed RDS messages, while simultaneously designing the over-the-air protocol to support low-power, embedded devices with no Internet connectivity.

I implement a full end-to-end system, deploy it on a 3 kW commercial radio station in



a metropolitan area, and evaluate it. I implement two types of receivers: software for an FM RDS-equipped mobile phone, and a hardware and firmware RDS-to-LAN bridge. To demonstrate the flexibility of my broadcast Internet last-mile link, I devise a push-based DNS server that does not require any modifications to the protocol, and I describe a large-scale device reset that does not require the sender to maintain any state about the devices.

# Chapter 2

## Background and Related Work

In this chapter, I provide background on prior approaches to observing and improving last-mile link availability. A common theme to most of this prior work is a reliance on privileged data or a willingness to modify links. I discuss some of the results that these assumptions have made possible, the limitations these assumptions impose, and the challenges involved in operating without these assumptions. In particular, I explain how researchers with some privileged data can observe last-mile link failures. These data may contain the precise timing and cause of weather-related failures. Also, I describe how researchers without privileged data can observe failures of core Internet links, but not last-mile links.

I also provide background on some of the technologies I make use of throughout my dissertation. In particular, I describe work that improves smartphone communication and positioning energy consumption. Finally, I review data broadcasting technologies that are potential candidates for being an Internet last-mile link.

## **2.1 Observing last-mile link failures**

Last-mile link failures can be observed either with privileged data, probing out of the network from inside and observing the statistics maintained by the link equipment or without privileged data, probing into the network from the outside. In this section I describe approaches to observing last-mile link failures both with and without access to privileged data.

### **2.1.1 With privileged data**

With privileged data, observers can more easily identify the exact causes of failures, because they can directly query the state of link equipment. However, it is difficult to obtain privileged data from the approximately 1,500 ISPs that exist in the U.S. [27]: let alone, all of the ISPs in the world.

Last-mile link equipment maintains detailed measurement data that can help to directly diagnose failures. Generally the only people with access to this data are employees of the ISP. Although my work does not use these data directly, it is informative to understand them as a goal for my indirect probing. I introduce these by describing some of the measurement data recorded by DSL and Cable last-mile link equipment.

In DSL, the central equipment, the DSLAM, tests last-mile links [45] and in Cable, mid-link transponders test links [4, 14]. Both can observe detailed signal conditions, such as uplink and downlink signal power and noise margin, as well as basic status, like whether or not the customer's modem is powered on. They also report on the state of supporting equipment such as power supplies and battery backups. In summary, there is a wealth of detailed information about the status of last-mile link failures, but this

information can only be obtained by ISP employees.

There are also systems that test last-mile links without the ISP's participation, but with the customer's permission. Their interfaces for collecting these data are restricted to only the customers who chose to participate in the collection. By attaching unbiased measurement devices, such as always-on modified home routers, directly to customer last-mile links, researchers and governments can directly probe the last-mile link to isolate the cause of connection failures [83, 92]. They also can observe bandwidth and latency which are difficult to obtain without customer participation [25]. The main limitation of these types of measurement is in their limited coverage of providers and link types. SamKnows [83], a commercial entity, and BisMark [92], an academic research project, deploy off-the-shelf wireless routers with modified firmware and monitoring software in homes of volunteers. As of January 2013, SamKnows has deployed about 10,000 devices and BisMark has deployed approximately 300. A European organization, RIPE NCC, created their own embedded systems that attach to last-mile links over Ethernet and constantly send pings and traceroutes to well provisioned servers (such as root DNS servers) [82]. As of June 2013, they have deployed about 3,200 of these systems.

### **2.1.2 Without privileged data**

Without privileged data, failures must be inferred indirectly from probes sent from outside the ISP's network. The complicating factor is that a failure observed by one of these external probes may not have been caused by a failure of the last-mile link being probed, but rather any of the other links on the path to and from the last-mile link.

The primary option for observing last-mile link failures without privileged data is to

send pings [73] from outside the ISP's networks to a customer's host. Any Internet host can send a ping to any other Internet host, however pings can be lost for several reasons, only one of which is a failure of the last-mile link. Some hosts' operating systems may have been configured to ignore pings. Additionally, pings could be lost due to failures of any other link between the sender and the receiver, congestion of any of those links, or the host powering off.

Nonetheless, pings have proven to be useful for observing the link properties (e.g., bandwidth, loss, latency) of last-mile links and failures of Internet transit links. Dischinger et al. [25] used pings (and other probes) to observe the bandwidth, latency, and loss of 1,894 last-mile links from 6 DSL and 5 Cable ISPs, but they do not observe last-mile link failures. The most related failure observation system to the one I describe in Chapter 3.1 is Trinocular [75]. Trinocular works by pinging several hosts in a subnet. When all of the hosts in a subnet fail, they infer this is caused by a failure of the subnet's transit links. Because they infer failures from several hosts, they can not find a failure of a single host last-mile link.

In summary, privileged data from ISPs includes fine-grained failure observations, but I can not feasibly get access to them; in-home observation systems can indicate the cause of failures, but they do not have the deployment diversity needed to observe weather-related failures. Without privileged data, related work also uses pings to observe Internet link failures, but they are only able to observe link failures that affect many last-mile links.

I observe last-mile links externally, without privileged data. To do so, I design a probing system, and the analysis of its pings to observe last-mile link failures. In Chapter 3,

Provider	Country	Type	Device	Precision
Sprint	USA	EVDO	Pre	$\geq 80$
Reliance	India	EVDO	EC1260 USB	6
Verizon	USA	EVDO	Omnia	$\geq 80$
AT&T	USA	HSDPA	SGH-i907	$\geq 80$

Table 2.1: Sources of measurement data, including different technologies in different countries. “Precision” represents the number of unique signal strength values the device reports.

I design a tool and analysis technique to monitor many geographically diverse last-mile links across different providers in different weather conditions, without privileged data.

## 2.2 Improving the energy efficiency of smartphone communication

In this section, I identify the energy costs of various features of smartphones. I describe where energy consumption can be improved and describe some of the related work that has improved it. These energy features are representative of the current state of smartphone technology. For concreteness, I provide power measurements from the Palm Pre and Samsung Omnia exemplars; I expect their relative values on other smartphones will be similar. This section comprises two parts: an energy model for the radio and an energy model of the processor and positioning peripherals.

### 2.2.1 Radio energy

Communicating with a strong signal reduces the energy cost by cutting both the power drawn by the cellular radio and the communication time. When the signal is strong the radio uses less power, for both transmission and reception, although my focus is primarily on the reception power. A strong signal also makes it feasible to use advanced modulation schemes that yield higher throughput, thereby cutting the time needed to complete the communication and potentially allowing the device to sleep (enter a low power mode). I

address power and time, in turn.

Intuitively, the energy consumed by cellular communication varies with signal strength, which changes as the phone moves. However, although the extent to which the cellular energy consumption increases as signal strength decreases is known by manufacturers, it is not well understood by researchers. To remedy this, I observed the energy consumption of mobile phones while they communicate in various signal strengths. Table 2.1 lists the mobile devices and networks that I measured. These devices expose signal strength in one of two ways: some provide fine-grained, raw Received Signal Strength Indication (RSSI), others provide only six coarse signal levels, corresponding to the (0–5) “bars” displayed on phones. These reported values lack meaningful units and instead serve as a rough indication of signal strength to users.

Power measurements are typically performed in the lab with an AC powered device [101, 103]. But, measuring the energy consumption of mobile phones in motion requires a power measurement apparatus that is both portable and can be connected between a mobile phone’s battery and the device. My setup consists of a USB oscilloscope that measures current by observing the voltage drop over a  $0.1 \Omega$  precision shunt resistor. The resistor connects the phone’s power lead to its battery.

One reason the radio draws more current to operate in low signal locations is that the power amplifier switches to a high power mode to counter the drop in signal strength [19]. This applies for both transmission and reception since the mobile client continuously reports the received signal strength to the base station, 800 to 1600 times per second (the base station uses this feedback to choose an appropriate modulation and data rate). Figure 2.1 depicts the power consumption of various devices while receiving a packet

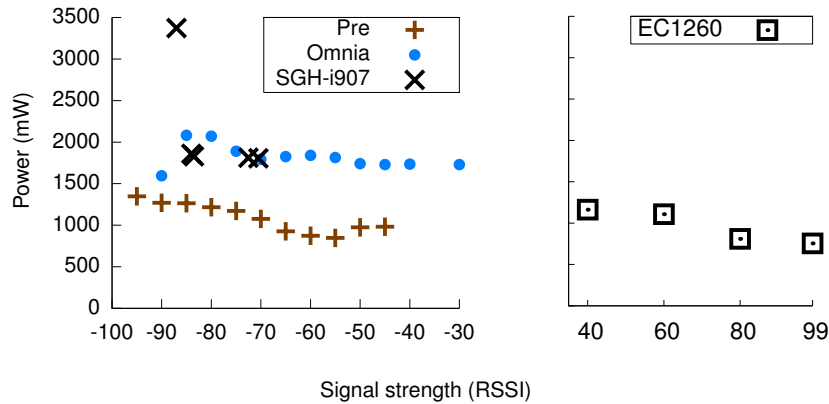


Figure 2.1: Power consumed by mobile devices report fine- (left) or coarse-grained RSSI (right). The EVDO devices are measured while driving; the i907 was measured statically at several locations. The very high i907 power was observed while communicating indoors.

flood, across various signal strengths. Communication in a poor signal location can result in a device power draw that is 50% higher than in strong signal locations.

In locations with high signal strength, there is high throughput, and thus communication takes less time. Specifically, strong signal allows for high modulation rates. For example, EVDO Rev A uses one of 14 rates ranging from 38 Kbps to 3.1 Mbps depending on signal strength. Figure 2.2 depicts cumulative distributions of receive throughput for various signal strengths. The measurements are 2-second (Reliance) and 3-second (Sprint) samples of throughput while receiving a flood of UDP packets.

The median throughput increases dramatically with signal strength: there is a four-fold difference in the median throughput between 60 and 99 RSSI for the Reliance network in India and a similar difference appears between -50 and -110 RSSI for the Sprint network in the US. However, the CDF also shows a wide range of throughputs for each signal quality bin, likely due to variations in sharing of aggregate cell capacity with other users. For the device on Sprint’s network, RSSI values are rounded up to the nearest mul-



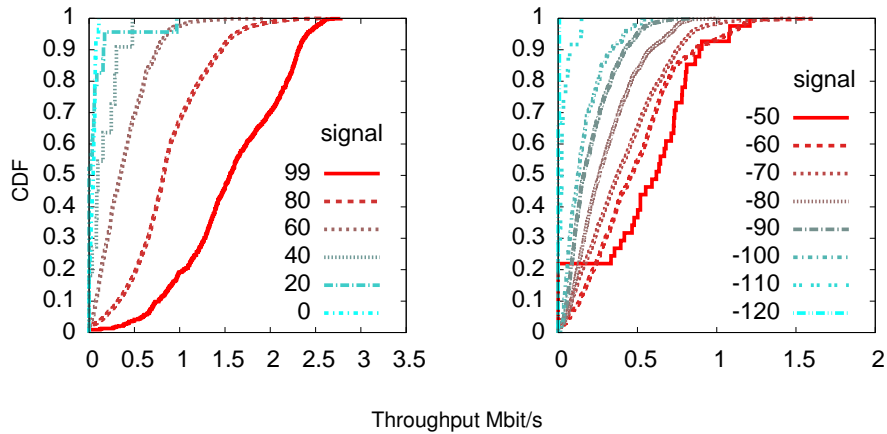


Figure 2.2: Signal strength affects throughput. Throughput over 9 drives on Reliance Telecom’s EVDO network in Bangalore, India (left) and a 4 hour drive using Sprint’s EVDO network on US interstate 95 (right).

tuple of ten, i.e., values between -59 and -50 appear as -50, while the device on Reliance’s network reported only six distinct signal values.

In summary, when the signal is weak, not only does data transfer take longer to complete, but the radio also operates at higher power. These two factors are cumulative, so the overall energy required to transfer a fixed chunk of data, i.e., energy per bit, can be as much as *six times higher* (25% throughput and 150% power) while communicating from poor signal locations compared to strong signal locations.

**Sporadic communication is inefficient.** The cellular radio operates in several power states depending on expected future communication. The radio mostly remains in a low power state, ready to receive incoming phone calls. At an intermediate power state, the radio is ready to transmit and receive data packets. Finally, in the highest power state, the radio is actively transmitting or receiving data. Apart from these states, when the received signal is very poor, the phone may expend energy continuously while searching for a tower with strong signal.

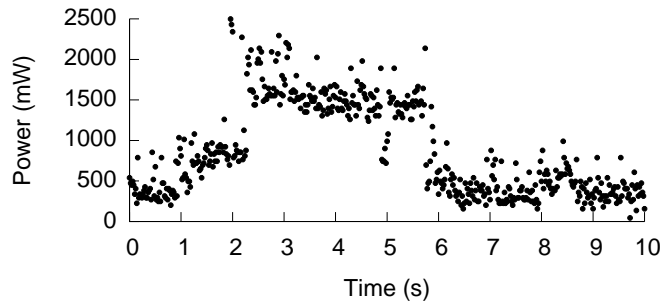


Figure 2.3: The Palm Pre’s tail energy. The Pre initiates an ICMP Ping at 2 s and it completes at 3 s; the tail continues until 6 s. Following the tail, the baseline power of the phone is 400 mW with the display dimmed.

The radio remains active in the intermediate power state for a preconfigured timeout duration after each communication episode, consuming what is known as “tail energy”. Cellular network providers typically control this timeout value, though some mobile devices use a technique called fast dormancy to reduce the duration [74]. The duration of this timeout, which ranges from a few seconds to ten seconds or more, is chosen to balance (1) the cost of signaling that allows a radio to acquire the spectrum resources to operate in the active state (and the resulting latency and energy costs on the device) and (2) the wasted spectrum resources due to maintaining a radio unnecessarily in active state.

Figure 2.3 depicts the progression of the Palm Pre’s radio power states when transmitting a short message. A single ICMP ping message is transmitted around 2 s. Prior to 2 s, the radio remains in a low power state (the phone, dimmed display and radio consume less than 400 mW). When the ping is initiated, signaling messages are exchanged for resource allocation and the radio transitions to the high power state (power drawn goes up to 2000 mW). The ping is sent and a ping response is received between 2 and 3 s. The radio then remains in an intermediate power state, ready to transmit and receive further packets, until about 6 s (power drawn around 1500 mW). Finally, the radio transitions

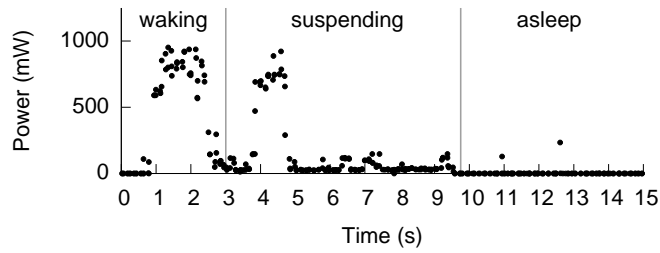


Figure 2.4: Samsung Omnia’s power consumption during wakeup and suspend.

back to the low power state.

It is this tail energy cost that makes sporadic communication a significant energy drain on mobile devices. Prior work solves this problem by consolidating periods of communication and thus reducing the tail energy wasted by sporadic communication [7, 86]. Communicating during strong signal strength can save energy, but it must not make the communication more sporadic, and thus waste the saved energy as tail energy.

## 2.2.2 Processing and positioning energy

These findings reveal a promising opportunity that prior work has yet explored: if one could predict high signal strength then delay tolerant traffic could be sent when the signal strength is high. One way to predict the signal strength is to predict location, but finding the location must be precise enough to detect signal strength without consuming more energy than it saves. In this subsection, I describe the energy consumption of the hardware needed to predict signal strength: processors and positioning peripherals.

**Processors** can sleep in a very low power state. The Samsung Omnia uses extremely little power while sleeping, and when awake it requires approximately 144 mW. Sleep precludes any activity until the processor is awakened by an interrupt. The focus of aggressive power saving is thus to keep the processor in suspended state for as long as

possible.

Unfortunately, the transition between sleep and active states requires more power than an active processor. I show an example transition on the Omnia in Figure 2.4. The peaks before and after the awake state represent the energy cost of restoring state and saving state for peripherals. The energy cost of these transitions is approximately 1 joule to restore from sleep and 0.5 joules to return to sleep. In short, the processor cannot simply be duty cycled to save power.

**GPS** devices can provide precise positioning, but they suffer from high latency to obtain a fix from the satellites and high power to interpret their weak signals. Constandache et al. report 400 mW baseline power for GPS on a Nokia N95 [18]. GPS energy consumption can be improved by offloading intensive GPS processing to a datacenter [53].

**Cellular signal strength** provides an alternative approach to positioning [50] with much lower power requirements. The radio measures signal strength as part of the normal operation of the phone, for example, to receive incoming phone calls and perform registrations. The radio performs this measurement even while the processor is suspended.

**Accelerometer** measurements are nearly free, but can only be used when the processor is powered on. Accelerometer measurements have been previously used for positioning [17].

### **2.2.3 Energy-efficient cellular data scheduling**

In Chapter 4, I present Bartendr, a system that schedules cellular communication during time when signal strength is high. It saves more energy than the scheduling consumes

by efficiently predicting future signal strength with previously collected tracks of signal strength progression.

The proportional fair scheduler [49] used in 3G networks today already uses signal conditions at the mobile nodes to preferentially schedule nodes with higher signal strength. Bartendr uses the same principle of channel state-based scheduling but differs from the proportional fair scheduler in two ways. First, while the proportional fair scheduler operates at a fine granularity of milliseconds, Bartendr schedules traffic over time intervals of tens of seconds to several minutes or more. Thus, while proportional fair is *reactive*, relying on continuous feedback of channel state from the mobile nodes, Bartendr must *predict* future channel state in order to schedule effectively. Second, proportional fair is used today only to schedule traffic on the downlink while Bartendr schedules both uplink and downlink traffic, leveraging a network-based proxy for buffering downlink traffic.

Approaches like TailEnder [7] and Cool-Tether [86] create energy savings by reducing the cellular *tail* overhead; TailEnder performs batching and prefetching for email and web search applications, respectively, while Cool-Tether performs aggregation for web browsing. In contrast to these approaches, Bartendr takes into account both the tail overhead and signal strength information for saving energy.

### **2.3 Improving reliability with data broadcasting**

The final contribution of my dissertation demonstrates how to deploy an additional, reliable, broadcast Internet last-mile link. Specifically, I develop an end-to-end reliable broadcast link that is inexpensive to deploy to many hosts and that permits energy-efficient

receiving. In this section, I describe various broadcast systems which provide blueprints for a broadcast Internet last-mile link. Historically, providers tailor data broadcasting (datacasting) equipment to the requirements of specific applications. For example, datacasting architectures have been created to provide proprietary devices with stock quotes and news [60].

### **2.3.1 Datacasting systems**

Broadcasting digital data to a wide area is known as datacasting. Datacasting systems often transmit digital supplements to analog TV and radio broadcasts. Although they provide interesting services, datacasting systems have come and gone [22, 79]. Most of them required proprietary receivers and comprised providers with tight control over the data that was transmitted. Although some provided Internet-like functionality for example, delivering web pages, typically the content did not come from arbitrary Internet sources. Given that a large number of these systems existed, I discuss only a few notable ones.

**Televisions.** The earliest datacast systems were developed to send closed captioning along with broadcasts for the hearing impaired. The British Broadcasting Corporation developed one of the earliest systems in the 1970s called Teletext [9]. Teletext transmissions were pages of text modulated during the TV vertical blanking interval. The content included news, program guides, and supplemental information about programs. Although the encoding bitrate was 7 Mbits/s, the low frequency and short duration of the vertical blanking interval limited the effective bitrate to just tens of Kbits/s [13]. In the 1990s, Intel used the vertical blanking interval for their datacast system called Intericast. Intericast

broadcast web pages alongside programming to PCs with an InterCast TV tuner card [22].

**Video game consoles.** Video game console makers built subscription services for distributing video games via broadcast. Subscribers would rent a receiver cartridge from their cable company that they would connect to the cable and the console. The cartridge then received new games and demos from a dedicated cable channel. Examples of these services include Mattel's Intellivision PlayCable and Sega's Sega Channel. Video game broadcasts were not limited to cable TV: Nintendo Satellaview transmitted games through Japanese satellite TV providers [43].

**Smart objects.** Microsoft DirectBand (MSN Direct) [60] was a subscription service that provided timely content such as news, weather, and gas prices to battery-powered smart personal objects (typically watches). Microsoft transmitted the data with a proprietary protocol over leased bandwidth from FM radio stations around the US. The devices included a proprietary receiver Integrated Circuit (IC). DirectBand was not limited to only provided content; eventually they allowed subscribers to send locations to their car navigation devices via this channel. The transmission rate was 12 Kbit/s. The service ended on January 1, 2012 [59].

In Chapter 5, I adapt an existing, deployed broadcast technology to provide a reliable broadcast Internet last-mile link. Learning from the failures of the datacasting systems described in this section, I design a broadcast system that stands a chance to last; the receivers are commercially available hardware, and are energy-efficient, so they

can be deployed on phones. Also, I show with a measurement study that most of the transmitters—which are already deployed—cover at least 100,000 people.

### **2.3.2 Broadcast technologies**

Various data broadcast technologies have been used to provide arbitrary data services such as stock quotes and news headlines (Section 2.3.1), but most of those systems failed. One that has succeeded in a wide deployment is the FM Radio Data System (RDS). The advantages of FM RDS in particular are that transmitters are deployed, both the over-the-air protocol [80] the transmitter control protocol [29] are an open standards, and there are commercial off the shelf receiver ICs.

To be a reliable broadcast Internet last-mile link, I believe a broadcast technology should have the following properties: (1) a VHF signal, because VHF is robust to weather, has extensive coverage, and the antenna size is small enough that receivers can be integrated into devices; (2) the ability to transmit arbitrary data, because like other Internet links, the broadcast link should be used for many applications (not just MPEG video); (3) the ability to transmit from arbitrary senders, because like other Internet links many senders should be able to transmit data to receivers that are interested in their data; (4) commercially available receivers, because many devices should be able to receive the data (not just devices authorized to be built with proprietary receivers).

Next I discuss candidate technologies for a radio broadcast Internet last-mile link. Table 2.2 shows the Internet last-mile link properties of each of the candidate broadcast technologies. FM RDS satisfies most of these criteria except for allowing arbitrary senders to transmit data. In Section 5.2 I describe the additions to FM RDS that enable



	VHF	Arbitrary data	Arbitrary senders	Commercially available receivers
White spaces		✓	✓	
Satellite		✓	✓	
Cellular		✓		
HD Radio/TV	✓	✓	✓	
FM RDS	✓	✓	○	✓

✓ = Property of the technology  
○ = New capability discussed in this dissertation

Table 2.2: Candidate datacasting systems broken down by the features needed for an Internet last-mile link.

arbitrary senders to share the broadcast transmission.

**White spaces.** UHF (e.g., 700 MHz) white spaces have two desirable qualities for an Internet broadcast system. The first is small antennas that can fit inside mobile devices: a quarter wavelength monopole antenna for 700 MHz is 10.7 cm. The second is significant bandwidth: UHF TV channels are 6 MHz wide (802.11g channels are 20 MHz).

These advantages come at a cost. UHF signals are attenuated more by buildings than the lower frequency VHF signals. Also, white space transmitters and receivers are complex because they must avoid primaries (licensed transmitters) [6]. Currently, there are no mobile commercially available white space receivers.

**Satellites.** The main advantage of satellites is their coverage: a broadcast can cover a continent. However, for an Internet broadcast last-mile link, satellite networks are not as desirable as terrestrial networks. This is due to the weakness of the signal, a problem often addressed by large antennas with clear line of sight, aimed directly at the satellite. These cumbersome dish setups would inhibit widespread deployment of a broadcast last-mile link. There are satellite links that do not require large antennas, such as GPS, but they limit deployment of a broadcast link because they can not operate indoors.

Adelsbach et al. [2] observe that existing satellite data services can be utilized for

broadcast distribution to all receivers in view of a satellite. The complicating factor is, the receivers must share the private cryptographic keys associated with a designated “broadcast” receiver. These keys can be difficult to access and to change because providers store them in closed satellite receiver hardware.

**Cellular.** There are several cellular broadcast systems that send audio and video from cell towers to smartphones. Examples of these protocols are 3GPP’s Multimedia Broadcast standard and Multicast Services for UMTS MBMS [37]. Cellular broadcast can achieve high bitrates due to the dense deployment of towers. Beyond audio and video, cellular towers can also broadcast SMS messages [1]. However, the ability to broadcast an SMS is restricted to a small number of senders authorized by providers.

**Radios.** The European Broadcasting Union developed the FM RDS in the 1980s [80]. RDS broadcasts station and content identity shown on radios with displays, and provides hidden information about alternate stations. The RDS signal resides in an FM radio sub-carrier and the bit rate is limited to 1.188 Kbit/s. The specification is open and internationally standardized [42]. Today, stations continue to broadcast RDS signals. RDS receivers are also built into Internet-connected devices such as smartphones (e.g., Motorola’s Droid X).

Rahmati et al. [78] demonstrate that it is feasible to construct large, repairable messages out of RDS’s eight byte messages. They also describe some of the higher-layer challenges in deploying a general data RDS broadcast system on existing FM radio stations. However, they do not address these challenges. Instead of using a naming sys-

tem, Rahmati et al. use application identifier and chunk identifier integers. They present several classes of applications, including wide-area queries with few responses and the distribution of agricultural prices.

The RadioDNS [76] standard gives Internet-connected radios a way to look up the IP address of Internet services for a radio station, based on the RDS broadcast identifier of a radio station. A demo of RadioDNS shows a receiver switching from FM to Internet streaming when reception is poor. I take from RadioDNS the expectation that devices will increasingly embed both FM receivers and Internet connectivity.

Digital radio such as iBiquity HD Radio and European Digital Audio Broadcasting (DAB) can transmit arbitrary data at bitrates significantly higher than RDS. The primary limitation of these systems is their deployment: HD radio is used only in the United States, while DAB deployments are mostly in Europe. Also, the receivers are not as small, inexpensive, or low power as RDS permits.

## **2.4 Summary**

In this chapter I demonstrated how techniques from prior work do not observe and improve the reliability of Internet last-mile links without modifying transmission media and equipment or accessing privileged data. Additionally, I provided background on last-mile link technologies that I make use of throughout the remainder of my dissertation. This includes my own measurements of smartphone communication energy consumption, which showed that communication in poor signal locations can result in a device power draw that is 50% higher than in strong locations. Finally, I described why I chose to use FM RDS for a reliable broadcast Internet last-mile link.

# Chapter 3

## Weather-related Last-mile Link

### Failures

In this chapter, I observe how weather affects the failure rate and failure length of fixed Internet last-mile links *by collecting data only from public measurement infrastructure*. Residential links are vulnerable to all types of weather, including wind, heat, and rain. This is because residential equipment and wiring are often installed outdoors: wind can blow trees onto overhead wires, heat can cause equipment to fail, and rain can seep into deteriorating cables.

Without privileged data (e.g., ISP equipment monitoring data [4, 14, 45] or data collected from residential routers [82, 83, 92]), it is possible to observe the responsiveness of the diverse set of last-mile link types, across a variety of geographic locations and various weather conditions. To observe last-mile link failures without privileged access, I send pings [73] from outside the ISPs' networks over their customers' last-mile links to their hosts. One difficulty of using pings to observe the diversity of link types, locations,

and weather conditions, is that the rate at which pings can be sent is limited either by server bandwidth or by last-mile ISPs. Because I cannot ping all last-mile links all the time, I design a system called ThunderPing (Section 3.1) that focuses pings on a sample of last-mile links in regions that are forecast to experience weather soon. ThunderPing follows weather alerts from the US National Weather Service (NWS) and pings a sample of last-mile link IP addresses from ISPs in the affected areas.

I ping conservatively. To avoid ISP complaints and rate limits, I ping a single host from each server once every eleven minutes. To ensure observed failures are of last-mile links and not other links on the path from my servers, I ping each last-mile link from ten different servers. To increase the likelihood that weather is the cause of observed failures, I ping beginning six hours before the alert begins and to observe time to recovery, I ping until six hours after it expires.

Pings, unlike privileged data from ISPs and researchers, indirectly observe the last-mile link. Therefore, I must analyze the pings to determine when lost pings likely indicate a failure of the last-mile link. Since different link types have different loss rates, I use a link's long-term loss rate history (on the scale of days) to determine if a sequence of lost pings indicates the link is completely, rather than partially, unresponsive (Section 3.2.2). Some links can fail by becoming partially, rather than completely, unresponsive. To determine when the link is partially responsive, I use an edge detection algorithm (Section 3.2.3) to find short-term transitions between loss rates (on the scale of minutes). Since the servers I am using can fail, resulting in lost pings, I exclude time intervals where many of the links a single server pings appear to fail simultaneously (Section 3.2.1).

Even after I perform these steps to find apparent last-mile link failures in pings, my

resulting set of failures may not be caused by weather-induced faults with the last-mile link, and my resulting set of link recoveries may not be recoveries of the same last-mile link. To separate weather-induced power failures from weather-induced last-mile link failures, I first characterize power failures in my data by correlating a known list of power failures with the number of ISPs that had last-mile links simultaneously fail in my data (Section 3.4.1). Then, I exclude all failures that appear as power failures, because last-mile links from several ISPs failed at the same time.

Determining the length of a failure until recovery with pings might be impossible because the last-mile link might be assigned a new IP address after a failure. I validate my assumption that I can observe the duration of failures with privileged data. This data indicates that for many ISPs in many locations, ISPs reassign the same IP address to last-mile links, even after the link has been disconnected for hours; so I may be able to observe failure length with pings (Section 3.5.1). In summary, I show how to observe the failure rate and duration of last-mile link failures *without privileged data*, and instead by pinging.

### **3.1 Measuring the responsiveness of Internet hosts during weather**

I developed ThunderPing to observe how adverse weather affects the failure rate and failure duration of Internet last-mile links. In this section, I describe the design of ThunderPing.

#### **3.1.1 Finding IP addresses subject to weather**

The first problem to address is to find last-mile link IP addresses in a geographic region that can be matched to a US National Weather Service alert. I select IP addresses by a scan

```

<title>Severe Weather Statement issued May 12 at 4:46PM CDT
  expiring May 12 at 5:15PM CDT by NWS GreenBay
  http://www.crh.noaa.gov/grb/</title>
<summary>...A SEVERE THUNDERSTORM WARNING REMAINS IN EFFECT
  FOR CENTRAL WAUPACA AND NORTHWESTERN OUTAGAMIE COUNTIES
  UNTIL 515 PM CDT... AT 443 PM CDT...NATIONAL WEATHER
  SERVICE DOPPLER RADAR INDICATED A SEVERE THUNDERSTORM
  CAPABLE OF PRODUCING QUARTER SIZE HAIL...AND DAMAGING
  WINDS IN EXCESS OF 60 MPH. THIS STORM WAS LOCATED 7 MILES
  NORTH OF NEW LONDON...OR 20 MILES NORTHEAST OF
  WAUPACA...MOVING</summary>
<cap:effective>2011-05-12T16:46:00-05:00</cap:effective>
<cap:expires>2011-05-12T17:15:00-05:00</cap:expires>
<cap:urgency>Immediate</cap:urgency>
<cap:severity>Severe</cap:severity>
<cap:certainity>Observed</cap:certainity>
<cap:geocode><valueName>FIPS6</valueName>
<value>055087 055135</value></cap:geocode>
...

```

Figure 3.1: Example XML entry for a weather alert for two counties in Wisconsin. Some XML entries omitted for brevity.

of the reverse DNS space, classify each IP address as a last-mile link by DNS suffix (domain), and determine their approximate location by the MaxMind GeoIP database [56].

The focused scan of reverse DNS records proceeds as follows. First I choose three IP addresses, ending in .1, .44, or .133 from every possible /24 block, and query for the name of each. If any of the three have a name matching an ISP in the DSLReports.com US ISP list [27], such as comcast.net or verizon.net, I determine all the names of all the IP addresses in the block and include the addresses with matching names. This approach is comparable to that used to study Internet last-mile links in prior work [25, 100]. I performed this scan once and I discovered 100,799,297 possible last-mile link IP addresses in the US.

The US National Weather Service provides an XML feed of the latest severe weather alerts for regions in the US [65]. An example alert appears in Figure 3.1. The regions un-

der alert are listed by FIPS code, which is a numeric code for each county in the US. The FIPS code for Los Angeles, for example, is 06037. I consider all weather alerts including “watches,” which indicate conditions conducive to severe weather, and “warnings,” which indicate that severe weather has been observed.

To map IP addresses to the FIPS codes used in weather alerts requires IP geolocation. I used MaxMind’s GeoIP [56] database to determine an estimate of the latitude and longitude of each IP address, and the US Census Bureau’s county border data file<sup>1</sup> to determine the FIPS county location for any IP address.

I use MaxMind’s database because of its availability and the potential to determine the location of every possible last-mile IP address. Researchers have questioned its accuracy [71], and have developed probing-based methods for positioning Internet hosts [98, 99] that seem impractical for locating 100 million hosts. Clearly, improved IP geolocation methods would yield more precision to the location and might lend more accuracy to my analysis. I expect, however, that precision in geolocation would have limited benefit because weather alerts are provided on the scale of a county and because weather does not respect city or county boundaries.

After an alert comes in, I pick 100 IP addresses from every provider and link type (when embedded in the DNS name) in each FIPS-coded region in the alert. I identify a provider and link type by the DNS name without numbers (e.g., pool----.sangtx.dsl-w.verizon.net).

---

<sup>1</sup>[http://www.census.gov/geo/cob/bdy/co/co00ascii/co99\\_d00\\_ascii.zip](http://www.census.gov/geo/cob/bdy/co/co00ascii/co99_d00_ascii.zip)



### **3.1.2 Pinging (last-mile links) in the rain**

Testing my hypothesis that weather affects the Internet is difficult because weather's effect on the connectivity of Internet hosts may be hidden by congestion, outages at the source, or other network events.

#### **Ping infrequently**

Internet measurement traffic has a tendency to generate reports of network abuse from recipients of unsolicited traffic. I send typical ICMP echo messages with an identifying payload as infrequently as possible.

I follow the inter-ping interval chosen by Heidemann et al. in their Internet census. [38]. They surveyed the occupancy of IP addresses on the Internet on the scale of tens of minutes, and reported that they could send pings at an interval of 5 minutes without generating any abuse reports. For their surveys, they pinged IP addresses for several weeks at an 11 minute interval without generating many abuse reports, so I do the same.

#### **Omit needless pings**

In addition to sending more pings to determine if a host is down, ThunderPing must cull the set of observed hosts during a weather alert to include only those that respond to pings. Otherwise, the pinger would waste time pinging addresses that either are not assigned to a host or have a host that is not awake for the weather event. I implement a simple timeout: If after an hour (five pings from each of ten vantage points) a response is not heard from the host, then it is no longer pinged for that weather alert.

### **One vantage point is not enough**

ThunderPing distinguishes between faults in the middle of the Internet and faults at the endpoint, the observed host, by simultaneously pinging from several vantage points. The responsiveness of the host is not determined by any individual vantage point, but by agreement between the vantage points. For the experiments in this chapter, I used a dynamic set of up to ten PlanetLab machines as vantage points. Ten permitted some vantage points to fail occasionally while still giving each IP address a good chance of demonstrating availability even during low loss.

### **3.1.3 Potential sources of error**

A source of error for my probing would be when a host appears to have failed, but in reality, its Dynamic Host Configuration Protocol (DHCP) lease simply ran out and it was given a new address. From correlating Hotmail identities to IP addresses for one month of data, Xie et al. [100] report that for SBC, one of the largest DSL providers in the US, most users retained the same IP address for one to three days. For Comcast, one of the largest cable providers in the US, they report that 70% of IP addresses do not change for users within a month-long trace. This stability suggests that the addresses of responsive hosts will not be reassigned in a way that would suggest failure during weather events. However, these studies do not make clear whether hosts that fail hold on to their IP addresses after recovery. I investigate this in Section 3.5.1.

## **3.2 Inferring link-state from pings**

Observing last-mile link failures caused by weather requires measurements of the link's state at the same timescale as weather: minutes. In minute-timescale link-state observa-

tions, a failure during weather will appear as a transition from a link-state of low loss (**UP**) to a state of persistent high loss, where either the link is entirely unresponsive (**DOWN**) or partially responsive (**HOSED**).

I assume that the behavior of ping responses at minute-timescale, and across several vantage points, reflects the state of the host's last-mile link. It is challenging to infer link-state from instantaneous pings because of general loss on the Internet and ambiguity due to aliasing. For any set of ping responses, there could be several link-states that created those response patterns. The following are examples of last-mile link-states that will be misclassified as other states based on minute-timescale sets of pings.

- The last-mile link is responsive (**UP**), but a few pings are lost due to general Internet loss, or failure of an individual Internet link (looks like **HOSED**).
- The last-mile link is unresponsive (**DOWN**), but due to slow vantage points, ThunderPing only sent a few pings during that time (looks like **HOSED**).
- The last-mile link has a persistent 50% loss rate (**HOSED**), but due to chance, several successful pings occur sequentially (looks like **UP**).

I contribute a method that attempts to resolve these ambiguities by considering history of link-state. The goals of this analysis method are two-fold: (1) Discount lost pings due to faults with the measurement platform (PlanetLab), not fault of the last-mile link, and (2) only consider a link as **DOWN** when it is very unlikely that a sequence of failed pings would occur due to random loss.

I contribute the following analysis components that reduce pings to minute-timescale link-states while keeping with the goals listed above. In the following sections, I describe

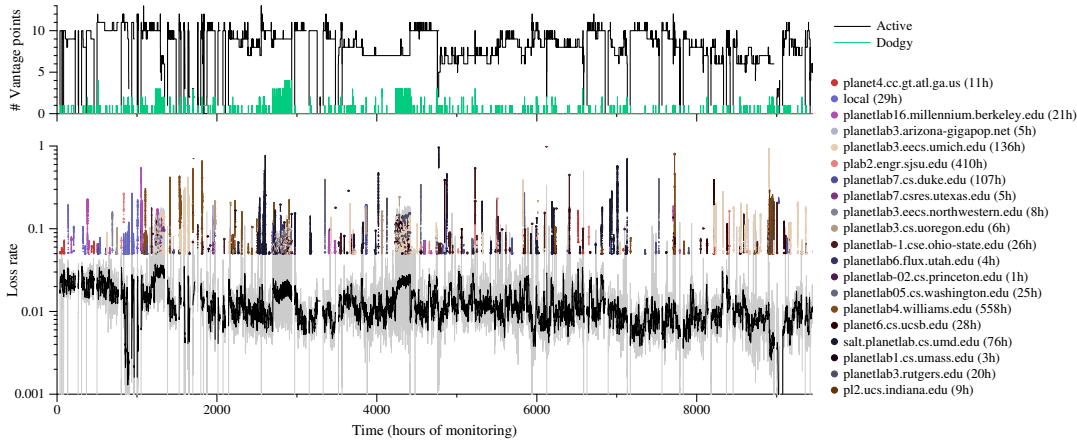


Figure 3.2: Average observed loss rate and excluded “dodgy” PlanetLab node loss rates. Different PlanetLab hosts become dodgy at different times and with different severity. *Lower graph:* Overall loss rate (gray), smoothed loss rate (black), loss rate of dodgy vantage points (various heavy lines above 0.05). *Upper graph:* Concurrently-dodgy vantage points and total active vantage points; spikes downward are typical of system restarts.

the details of each step:

**Section 3.2.1** Discard failed pings that are sent from a temporarily dodgy vantage point.

**Section 3.2.2** Compute conditional probabilities of pings. Find failures (**DOWN**) with conditional probability based anchors, and expand them.

**Section 3.2.3** Find edges in remaining active intervals between continuous lossy (**HOSED**) almost no loss (**UP**).

### 3.2.1 Filtering out intermittent PlanetLab node failures

I use PlanetLab to provide vantage points for ThunderPing because it provides a reasonably diverse platform in terms of geography and commercial network connectivity [91]. This diversity provides the potential to identify and ignore faults that are better identified as network routing faults or errors that occur at an individual site.

In observing the sequence of successful and failed pings for individual IP addresses, one feature often repeated: a single PlanetLab vantage point failed repeatedly while the

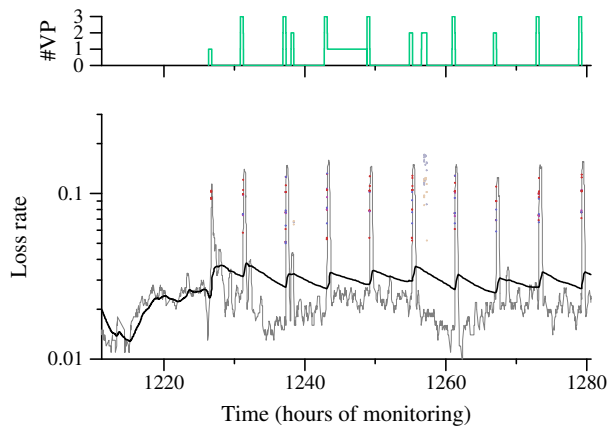


Figure 3.3: Detail view of a period where different PlanetLab nodes were declared “dodgy.” The periodic spikes in overall loss rate coincide with periodic management activity on my coordination machine, and the “dodgy” nodes are ignored relatively briefly.

rest received nearly flawless responses. This would appear to be 10% loss, which is substantial. This raised the question: is it a problem specific to that vantage point and destination, such as a routing failure or filtering? Or, is it a broader problem with the PlanetLab node itself, such as overloading, or filtering at the source? If the repeated loss is a problem at the PlanetLab host, can I believe that PlanetLab as a whole does not fail in this way all at once?

In this analysis, I estimate the overall loss rate for each vantage point—out of all pings it sent to destinations that have responded, what fraction of those pings timed out without a response—and compare that loss rate to the average of the loss rates of all other active vantage points. I call a vantage point “dodgy” when its loss rate is greater than twice the average of the rest and greater than 5%.

Figure 3.2 shows the basic calculation and its effect. The upper graph contains the number of active vantage points and the number of vantage points classified as “dodgy” at any time. The lower graph is a representation of the overall loss rate excluding “dodgy”

sources, using a log scale to capture both the typical 2% loss and the exceptional loss rate: gray is the raw loss rate across non-dodgy vantage points and the black line is a smoothed approximation. The overall loss rate fluctuates but generally approaches 2%. Above this line are the loss rates of the individual “dodgy” vantage points when so classified.

This error is not limited to PlanetLab: Notice that the figure includes “local,” a non-PlanetLab machine I pinged from, but which I manually excluded after learning not to trust its results. The analysis of “dodgy” nodes found this problem with my local node, as well.

There are three periods on the left half of the graph in which more than two PlanetLab nodes appear “dodgy.” Unfortunately, this appears to be a result of contention at my measurement coordinator, consisting of temporary spikes of apparent loss that correlate with a six-hour period for my measurement log rotation and compression script. One of these time periods appears in Figure 3.3.

### **3.2.2 Detecting failures with conditional probabilities**

Because the occasional ping may be lost, a single lost ping is not sufficient evidence that a last-mile link has failed. A subsequent lost ping is not particularly convincing either, since the probability of a loss, given that a loss just occurred, can in fact be rather high. Further, for a lossy link a sequence of lost pings is less indicative of failure than for non-lossy links. In this section, I describe how I use per-IP-address conditional probabilities to identify when hosts enter a **DOWN** state with a one-in-a-billion chance of being by random chance.

The task is to develop a method that provides approximately the same level of cer-

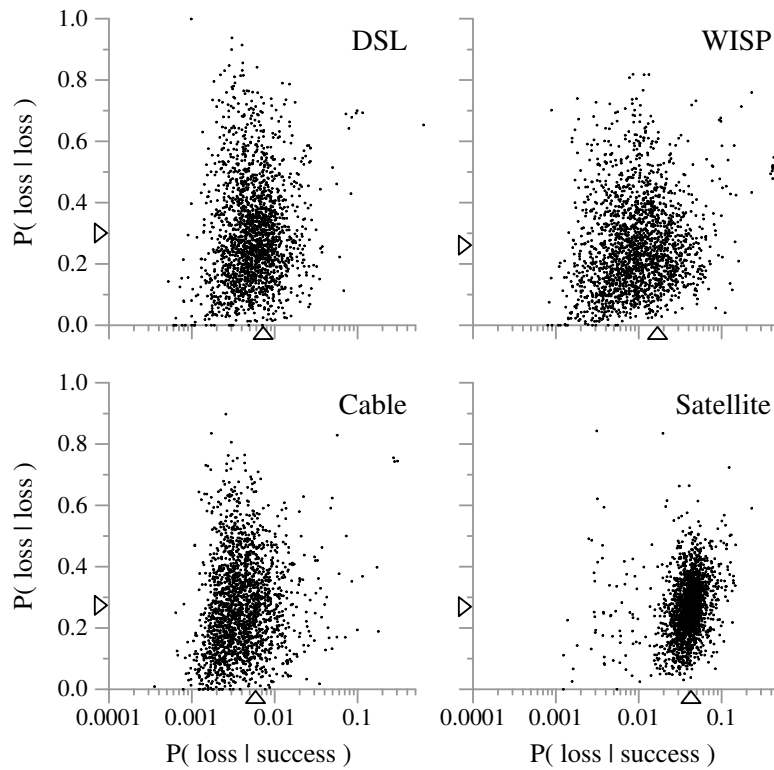


Figure 3.4: The conditional probability of ping loss over all pings (at least 10,000) sent to a sample of 2,000 IP addresses of each link type (each dot is an IP address). The conditional probability of a vantage point seeing a lost ping after another lost ping (the y-axis) is greater than the probability of a lost ping after a successful ping (the log-scale x-axis), even after removing intervals when a destination is likely **DOWN**. Averages of each are indicated by triangles on the axes.

tainty that an IP address is **DOWN**, regardless of how many vantage points are active at a time or how lossy the last-mile link happens to be. I choose to model each IP address using two conditional probabilities of loss: one given that the prior ping from the same vantage point was also lost, the other given that the prior ping from the same vantage point was successful. I leverage the assumption that a ping will never succeed (receive a reply) across a failed link, and expect that sufficiently many consecutive lost pings provide confidence that the destination is unreachable.

Figure 3.4 shows, for a random sample of 2,000 IP addresses thought to be of a given link type by reverse DNS name, the conditional loss rate given a prior success (the shorter x-axis with range 0 to 0.5) or loss (y-axis). I omit ping sequences where a more liberal method declares that the host is down. This ensures that the probability of loss given a preceding loss is an estimate taken when the host is functioning. Points high along the y-axis have a high correlated loss rate: for these points, a loss predicts another, even though the IP address does not appear to be **DOWN**. Points along the diagonal indicate uncorrelated losses that occur regardless of context. High ping loss may be caused by various factors; these graphs summarize those factors to provide a dot for each of 2,000 IP addresses of each type of link.

In my analysis, I deem a host **DOWN** when the pattern of lost pings attain a probability of occurring by random chance below one in a billion. Each vantage point has an opportunity to contribute one loss given a prior success, with probability typically below 0.01. It can then contribute only losses given prior losses, typically with probability above 0.2. For IP addresses that consistently have high loss, these conditional probabilities of loss are higher and more lost pings in a row are needed to reach the one-in-a-billion threshold.



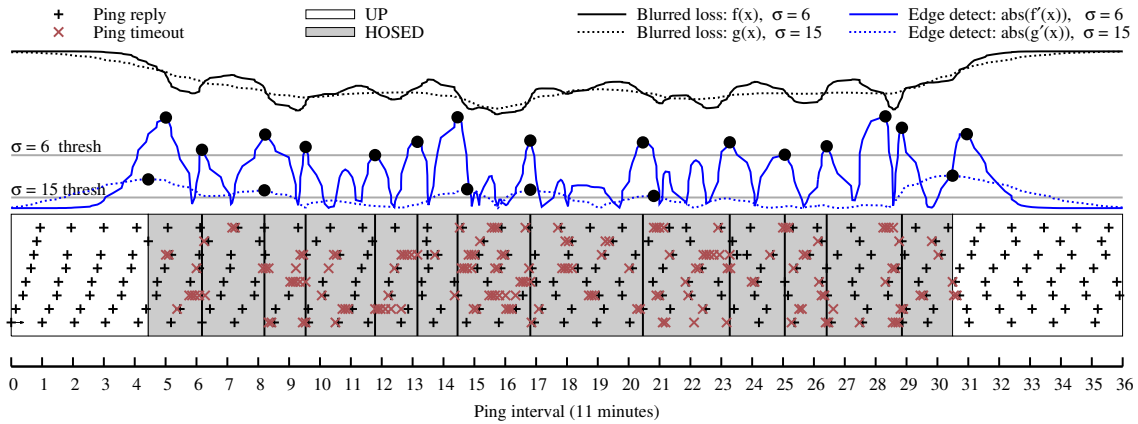


Figure 3.5: Determining the boundaries for states of responsiveness for a Speakeasy DSL customer in Virginia. The edge detection algorithm treats each ping from any vantage point as 1 or 0 observation (+ and × bottom), then it Gaussian blurs the pings to produce the blurred loss rate (top). Finally, local maxima (●) above the thresholds (*thresh*) in the derivative (middle) of the blurred loss indicate the state boundaries.

Similarly, when few vantage points are active, failed pings only increase the likelihood of a **DOWN** using the higher loss-given-loss probability, and thus more pings are needed to declare the host **DOWN**.

With the **DOWN** intervals identified by conditional probabilities, I now turn to separating **HOSED** from **UP**.

### 3.2.3 Detecting changes in prolonged loss rate

After identifying the **DOWN** states, the final challenge is to find the boundaries between the remaining pings that will be classified as one of two active link states: **UP** and **HOSED**. In essence, the goal is to find the ping that is at the boundary between two different persistent loss rate regimes. In my preliminary work, I attempted to find these boundaries with a moving average of loss rate. This approach found approximate boundaries between some instances of **UP** and **HOSED**, but it was sensitive to the loss rate threshold that defined **UP** and **HOSED** (Section 3.2.3).

I observe that finding the changes in ping loss rates, without knowing what the loss rates are, is similar to finding the edges in an image without knowing anything about the objects in the image. By assigning a 1 to a successful ping and a 0 to a failed ping, and by placing the pings in a list indexed by the number of pings sent, I can apply the classic Canny edge detection algorithm [10] on the one-dimensional list of pings. The detected edges are the boundaries between loss regimes.

### **Detecting edges**

Figure 3.5 demonstrates the edge detector on a Speakeasy DSL customer that transitions from **UP** to **HOSED** then back to **UP**. Given the array of ping responses described above, I smooth the responses by convolving with a Gaussian kernel (applying Gaussian blur). After the blurring, some pings will have fractional values (blurred loss). I then take the absolute value of the derivative of the smoothed pings (edge detect). The local maxima of these derivatives indicate the pings when there is a change in the loss rate. To avoid spurious edges, for each Gaussian kernel  $\sigma$  I select a threshold (*thresh*) that the local maxima of the derivative must be greater then to declare an edge. I find **HOSED** states consisting of short high-loss rate and longer low-loss rates by simultaneously detecting edges with a small and large Gaussian kernel  $\sigma$  values.

### **Parameterization**

I perform an experiment to test parameters for the edge detector ( $\sigma$  and the derivative threshold). Figure 3.6 shows the accuracy of detecting **HOSED** states with a range of loss rates and of durations. The edge detector parameters are the  $\sigma$  for the Gaussian kernel and the threshold of the derivative that a local maximum must reside above to call an

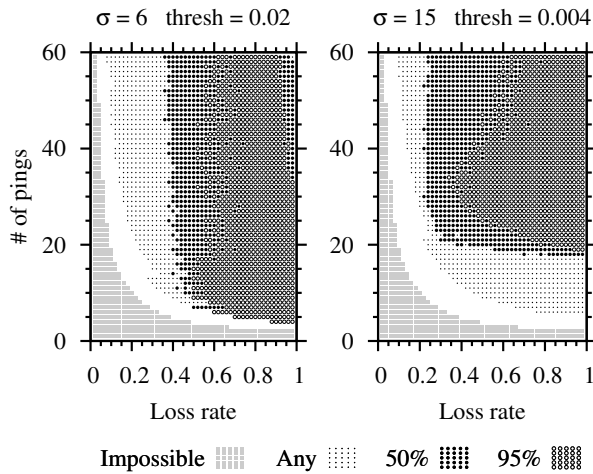


Figure 3.6: The edge detector’s accuracy of **HOSED** state detection for a transition from **UP** to **HOSED** then back to **UP**. I run the edge detector 100 times with random ordering of the failed pings in the **HOSED** state.

edge detected. I test the ability of the edge detector to detect an **UP**  $\rightarrow$  **HOSED**  $\rightarrow$  **UP** transition where I control the length and loss rate of the **HOSED** state. The **HOSED** state must start and end with a failed ping. To test the ability of the detector to detect the edges I test 100 random permutations of the failed pings that make up the **HOSED** for each pair of loss rate and length.

Figure 3.6 shows the results of the experiment for the final set of  $\sigma$  and *thresh* that I chose. For each **HOSED** loss rate [0 - 1] and number of pings [1 - 60] with random permutations of failed pings. The dots show the percentage [50%, 95%] of the trials where the edge detector found the start and end ping within five pings from each edge, as well as the detection of any edge, even if is inaccurate. I also plot small dots to show the **HOSED** states where any edge was detected, even if it was not accurate. It is very important that my edge detector does not fire at all when there is a state that is mostly **UP** for a long time, but has either a low persistent loss rate (upper left), or very small clusters of high loss (lower right). The gray blocked off area of the graph indicates impossible

parameterizations: the loss rates that are not possible given that my experiment requires at least one failed pings at the start and end of the **HOSED**.

A value of  $\sigma = 6$  accurately finds short **HOSED** with greater than 60% loss.  $\sigma = 15$  finds states with 30% loss rate for shorter lengths. I run two edge detectors with these two parameters simultaneously. When either of the edge detectors finds an edge, I initially treat it as true edge. However, for **HOSED** states longer than 60 pings, the detector has a high probability of firing more than once within a state. This is acceptable though, because states can be merged if they contain similar loss rates.

### **Placing the detected edge**

The edge detector detects edges, but as the edges come from a smoothed loss rate, the edge detector does not necessarily find the exact failed ping at the edge of the state. To remedy this, for each detected edge, I compute the loss rate for the state that ends with the edge. If the loss rate is greater than zero, then I move the edge to its closest down ping. When the edge is the end of a state with 0 loss, then I move the edge forward until the first successful ping followed by a failed ping. This small edge correction ensures **UP** states start and end with a successful ping, and **HOSED** states start and end with a failed ping.

### **Why not use a moving average of loss rate?**

Moving averages do not accurately find the edges of **HOSED** states (changes in loss rate), because there is no clear loss rate that defines **HOSED**. In the moving average example shown in the Figure 3.7, a Speakeasy DSL host enters a prolonged high loss state at ping interval 4. The windowed loss rate shown as the line hugs the 50% loss rate, but it does

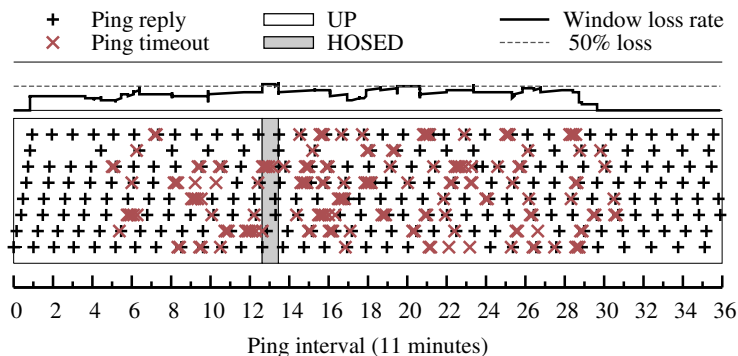


Figure 3.7: **HOSED** states may not exhibit a steady loss rate. So, the detection of a **HOSED** state with a moving average may start late and end early as it did in this example.

not cross the 50% threshold until interval 12, well into the **HOSED** state. The interval also ends early, with the loss rate dropping slightly below 50%.

### 3.2.4 Understanding the **HOSED** state

After detecting the edges between loss rate conditions, I calculate the loss rate of the pings between the detected edges, and use that loss rate to determine if the link is **UP** or **HOSED** in that interval. I define **HOSED** as greater than 5% loss rate, what I believe is unacceptable loss for most applications. If the loss rate is less than 5%, then the state is **UP**.

There are two categories of **HOSED** states that I discovered in this study: **HOSED** where loss rate related to link type and **HOSED** where only one PlanetLab vantage point receives ping replies can be due to by my PlanetLab vantage points being blacklisted by a WISP with a honeypot.

#### Loss rate by link type

Figure 3.8 shows the cumulative distribution of loss rates for **HOSED** states for the five primary last-mile link types I study (DSL, Cable, Fiber, WISP, Satellite). The first ob-

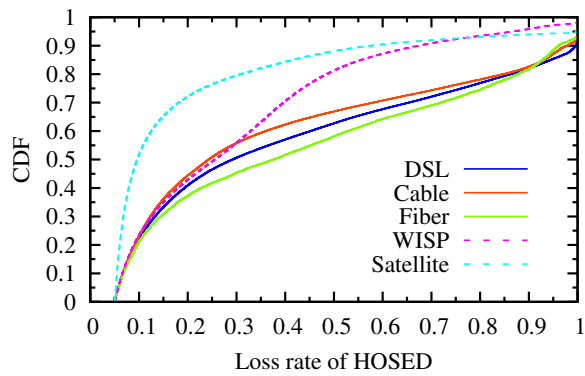


Figure 3.8: The distribution of loss rates for long (>50 pings) **HOSED** states is related to last-mile link type.

ervation is the low loss rate of satellite link **HOSED** states. For the wired links (DSL, Cable, and Fiber) and WISPs, 30% of the **HOSED** states were less than 15% loss rate. In contrast, 65% of the satellite links' states were less than 15% loss rate. This may indicate that satellite link loss rate is all or nothing: either there is very little loss, or the loss rate is so high that the link would not be classified as **HOSED** and instead be **DOWN**.

The second observation is that WISPs have the same distribution as the wired links for less than 30% loss rate, but above 30% loss there appears to be high loss that is WISP specific.

The final observation is the prevalence of 95% loss for Fiber and Cable links. I manually investigated a few of the links that experienced 95% loss, and it appears the customer is running software that filters or rate limits ping messages because the vantage point that receives the ping reply is not consistent.

### Honeypots

The most surprising **HOSED** behavior I found is when one vantage point pinging a link experiences almost no loss (less than 5% loss rate), while all others (up to 9) experience

100% loss. My investigation into this behavior revealed most of the IP addresses that exhibit this behavior belong to a rural WISP that uses a honeypot.

I contacted the WISP's administrator to determine the cause of this behavior. The administrator reported that the WISP had several unassigned IP addresses in their subnets set up as Honeypots. When any traffic comes to these IP addresses, their router blacklists the source IP address for three days. I observe only one vantage point getting through when all others are on the blacklist and the working vantage point has not sent a ping to a Honeypot in three days. It turns out that this behavior is not uncommon, I also discovered four other smaller WISPs that exhibit the same behavior. I suppose wireless ISPs blacklist aggressively to preserve their limited bandwidth. These WISPs' do not skew my results because I excluded their IP addresses in the other analyses.

### **3.3 Weather history and link type**

#### **3.3.1 The weather at a host during a ping**

The NWS and Federal Aviation Administration (FAA) administer approximately 900 Automated Surface Observing System (ASOS)<sup>2</sup> weather stations at airports in the US. These stations provide hourly weather measurements (primarily for pilots) in METAR format. Beyond the basic wind, pressure, and rainfall sensors, ASOS stations include a Light Emitting Diode Weather Indicator (LEDWI) that measures the type (Rain, Hail, Snow) and density of precipitation. Most stations also have antennas that measure lightning strikes. There are some weather events that are not detected by the automatic system, such as Tornados. These less common events are manually recorded by a weather ob-

---

<sup>2</sup><http://www.weather.gov/asos/>

server at the airport. The US National Oceanic and Atmospheric Administration (NOAA) provides yearly archives of the hourly METAR readings from many airports in its Integrated Surface Data<sup>3</sup> (ISD). ASOS stations. I do not use the larger network of older AWOS weather stations because some do not have the LEDWI sensor, which provides data I consider important.

Of the 24 fields that make up each hour of weather history, this study focuses on three weather conditions that I believe could cause a link failure: precipitation, wind, and temperature. The NWS categorizes precipitation precisely; they categorize 12 types of Rain. In this study I focus only on the type of weather, not its intensity, as the intensity that an airport experiences at some point in time may not be the same intensity experienced by the last-mile links outside the airport.

I group precipitation into categories: Rain, Freezing Rain, Drizzle, Snow, Thunderstorm, Fog, General precipitation, Smoke, Hail, Tornado, Dust, Haze, and Mist. I categorize wind speed using the Beaufort Wind Scale<sup>4</sup>. This scale ranks wind speed by its effects (e.g., “whistling in overhead wires”). Finally, the temperature readings are either: Hot (greater than 80 °F), Mild (between 80 °F and 32 °F), or Cold (less than 32 °F).

### **3.3.2 Identifying the link type of an IP address**

In order to test how the different link types fail during weather, I must identify the IP addresses’ link types. Figure 3.9 shows the number of IP addresses mapped to each link type. The different link types may fail differently in weather because of the different media types (coaxial, fiber, telephone, wireless), frequencies for wireless links, and MAC

---

<sup>3</sup><ftp://ftp.ncdc.noaa.gov/pub/data/noaa/>

<sup>4</sup><http://www.spc.noaa.gov/faq/tornado/beaufort.html>



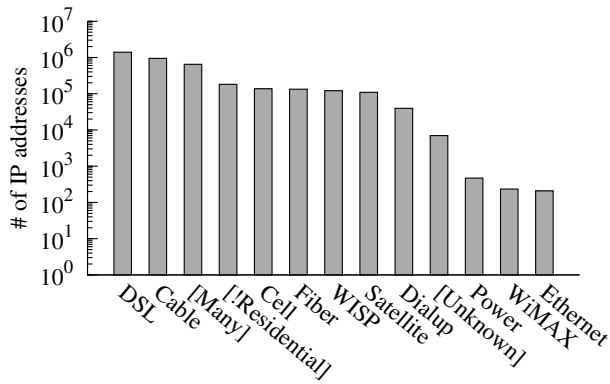


Figure 3.9: The distribution of link types for IP addresses pinged by ThunderPing.

protocols (retransmissions).

First, I identify the link type of IP addresses by searching for strings in the reverse DNS name that identify the link type (e.g., cable, dsl, and fiber).

For all of the remaining IP addresses, I determined their link types by manually inspecting the web sites of the 1,186 ISPs ThunderPing pinged.<sup>5</sup> From this manual inspection, I could determine the link type of 672 ISPs because they listed only one link type on their web site; all other ISPs listed at least two link types.

### 3.4 Failure rate

This section presents the results of my study on the effects of weather on residential link failures. I studied different link types and providers, particularly the largest providers of each link type. Table 3.1 summarizes how much time each provider spent in each weather condition as well as the total number of failures I observed, including and excluding suspected power outages.

The figures that follow show the failure rate in different weather conditions.

Precipitation can be accompanied by wind and high temperatures, but I only selected

<sup>5</sup>I also found 193 domains I obtained from the dslreports list that were not residential ISPs

	Cable			DSL					Satellite		Fiber			WISP			
	chtr	c'cast	cox	ameri.	cen'tel	megapath	speak.	w'stream	verizon	wildblue	starband	verizon	daktel	drtel	skyb.	digis	airband
Alive	244k	448k	61k	21k	172k	36k	12k	279k	274k	105k	2k	142k	1k	2k	11k	4k	4k
Airports	308	333	138	132	208	285	237	193	302	433	189	155	5	5	22	13	32
UP→DOWN	131k	178k	38k	17k	284k	32k	8k	236k	182k	263k	20k	17k	1k	2k	9k	3k	2k
—power	126k	172k	36k	16k	277k	30k	7k	225k	172k	252k	20k	15k	1k	2k	8k	3k	2k
UP→HOSED	84k	87k	40k	8k	183k	43k	17k	164k	116k	316k	51k	13k	1k	1k	12k	3k	3k
Clear	76.4	54.9	213.6	80.5	111.7	265.4	192.7	74.6	47.1	69.1	358.5	38.3	211.7	102.3	92.0	120.6	150.5
Snow	4.5	2.8	4.1	7.3	5.9	8.5	5.3	2.6	3.0	2.9	28.9	1.5	22.9	7.1	8.8	8.8	2.3
Fog	2.3	1.7	3.9	1.7	2.9	6.8	4.5	1.5	1.3	1.5	16.1	1.1	3.6	1.1	1.6	1.7	3.2
Rain	7.7	6.5	17.7	7.9	10.7	27.7	18.5	7.0	6.3	5.3	36.3	4.6	10.4	2.9	2.4	6.8	11.9
Tstorm	0.7	0.5	2.2	1.0	1.3	2.5	1.9	0.9	0.5	0.6	1.5	0.3	0.8	0.1	0.7	0.5	1.6

Table 3.1: Summary of a small portion of the data collected by ThunderPing. For a sample of providers, this table shows: the number of IP addresses that are ever seen alive, the number of airports that the alive IP addresses map to as well as the number of IP addresses that transition at least once from **UP** to **DOWN** and **UP** to **HOSED**. The final five rows are the average time (in hours) ThunderPing pings an IP address during each weather condition.

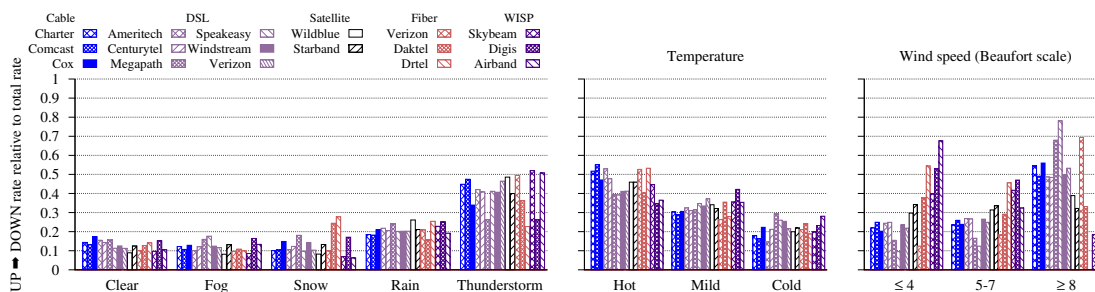


Figure 3.10: **UP** → **DOWN** failure rate for different weather conditions.

temperature and wind observations that are not accompanied by precipitation to observe the isolated effect of these conditions.

I compute normalized failure rate per provider as follows. For each provider, I sum the number of **UP** to **DOWN** and **UP** to **HOSED** transitions and divide by the time spent **UP** in each weather condition. Then, I normalize the failure rates for that provider so I can compare the failure rate across providers and link types.

### 3.4.1 UP to DOWN failures

Figure 3.10 shows the normalized rate of **UP** to **DOWN** transitions per provider *including suspected power failures* for five classes of precipitation (including clear), three categories of temperature, and three categories of wind speed.

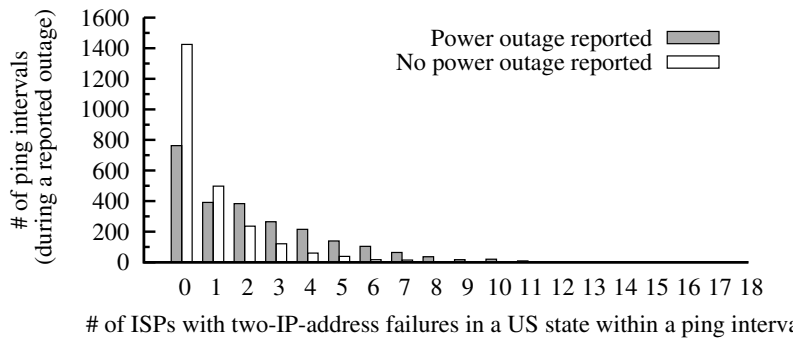


Figure 3.11: Comparison of the number of five-minute intervals that see failures across multiple ISPs within a US state during a power outage. Only failures of more than one IP address in an ISP are counted. The gray bars are shown for each interval. The white bar represents the number of concurrent ISP failures in a randomly chosen comparably sized state during matching intervals. I consider outages to be a result of power outage when the number of ISPs with two IP address failures exceeds 4. (Random selection repeated ten times; error bars omitted because too small for illustration.)

I observe the rate of **UP** to **DOWN** transitions in rain is  $2\times$  the rate in clear and the rate in thunderstorms is  $4\times$  the rate in clear. Snow does not appear to have a significant effect on failure rate; except in the case of the two fiber providers in North Dakota, Daktel experiences more Snow than any other provider in these observations.

Temperature and wind speed also appear to have clear effects on failure rates across all providers. The failure rate in high temperatures is  $2\times$  the failure rate in below freezing temperatures.

Cable and DSL are particularly affected by wind. Surprisingly, Satellite providers are not as affected by wind; although they could experience antenna misalignment in high winds.

## **Finding power outages**

Power outages are confounding factors for my analysis: they can be caused by weather, but they also can appear to be a network outage. At a high level, I expect that power failures are indifferent to the ISP providing residential service, and that significant power failures are likely to affect several ISPs at once. In contrast, a network failure would most likely affect only one ISP at a time, or only a few customers. (Of course, if a WISP uses a cable modem for backhaul, both could easily fail together.)

I have correlated my failure observations with two power reliability datasets. Most prominent is the OE-417 data maintained by the Department of Energy.<sup>6</sup> These data includes “electric emergency incidents and disturbances” including instances of vandalism and weather-related power outages, along with the affected state, time of occurrence, and number of customers affected. Utility companies record these by hand, and as a result comprise imprecise locations and rarely represent the precise timing of outages. In particular, the hundreds of small outages in a severe storm are typically captured in a single summary record.

Another dataset I considered was generated by scraping the outage reports from a utility web site (<http://pepcotracker.com>). Although I can exploit the apparent precision of the timing data (scrapes were repeated every ten minutes), these data seems to be more precise about the timing of events, the data set does not cover enough time to provide reliable conclusions.

Using the OE-417 data, I considered the following question: is there a severity of an outage across different ISPs large enough to indicate a power outage as the root cause with

---

<sup>6</sup>[http://www.oe.netl.doe.gov/OE417\\_annual\\_summary.aspx](http://www.oe.netl.doe.gov/OE417_annual_summary.aspx)

high probability? I experimented with several options to define severity and settled upon a failure that included at least two IP addresses from each of at least four ISPs. Higher thresholds (e.g., eight ISP failures) are more exclusively power failures, but include a very small fraction of all failures; lower thresholds capture a substantial number of non-power-related failures.

I show an aspect of this analysis in Figure 3.11. Given each five-minute interval within an OE-417-reported power outage, I choose each state within the outage and report the number of distinct ISPs that had two IP addresses fail during that interval. This interval is represented by an entry in a gray bar. To compare, I choose another state which, at the same time, had approximately (within 20% of) the same number of concurrently-probed ISPs. I then record the number of ISPs that experience a concurrent two-address failure with a white bar.

This analysis is particularly conservative. In order to find a comparative “no power outage” entry, I seek a state where I probe approximately the same number of ISPs, which suggests that the comparison state may also be undergoing some interesting weather. Further, the “power outage reported” intervals may include several in which no “new” power outages occur and few to no new **UP** to **DOWN** transitions occur. This is inherent in the imprecision of the OE-417 data. As such, I am optimistic that the actual pattern of failures induced by power outages is even more distinct from other failures.

### **Excluding power outages**

Now that I have identified correlated failures of four providers and two customers per provider as being suspected power outages, I revisit the **UP** to **DOWN** failure rate in each

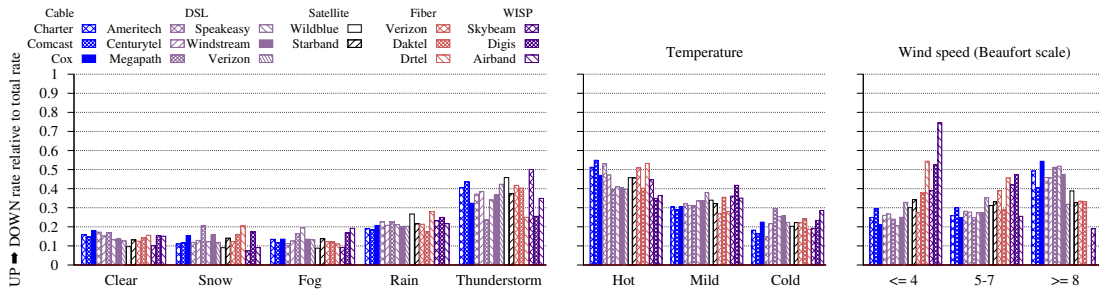


Figure 3.12: UP →DOWN failure rate excluding suspected power outages.

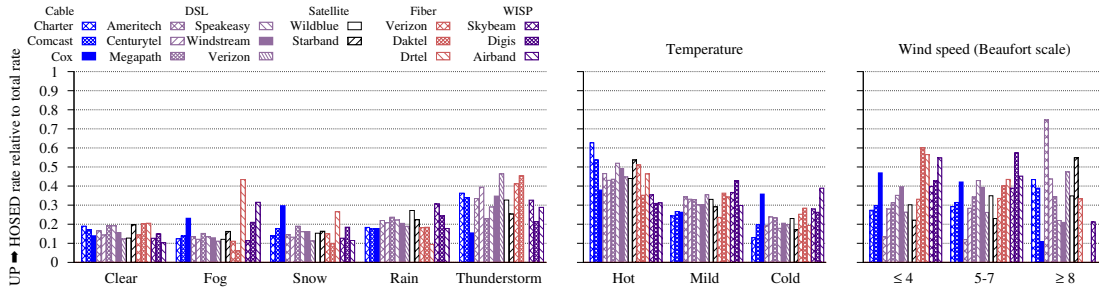


Figure 3.13: UP →HOSED failure rate for different weather conditions.

weather condition excluding these outages.

Figure 3.12 shows the recalculated failure rate without power outages. As expected, the failure rates decrease for rain and thunderstorms, but they remain higher than in clear conditions, indicating that thunderstorms and rain may cause link failures as well as power outages. The failure rates in different temperatures and wind speeds also decrease, but remain similar to the failure rate include power outages.

### 3.4.2 UP to HOSED failures

Figure 3.13 shows the prevalence of UP to HOSED transitions in different weather conditions. In rain and thunderstorms: Wildblue satellite and Skybeam WISP have more than double UP to HOSED transitions in rain and thunderstorms. The UP to HOSED rate in rain in thunderstorms is also similar.

## **3.5 Failure duration**

This section examines how long weather-related failures last. When observing failures externally, the observed length of failures may be incorrect because of dynamic IP address reassignment after the failure. This is because providers assign IP addresses to last-mile links dynamically, with DHCP. When a last-mile link fails, and subsequently recovers, the customer's modem will request an IP address, and the provider may return a different address than the one assigned before the failure. As such, before I present my observations of failure length, I defend ThunderPing's ability to observe failure duration by investigating the mechanics of DHCP, using BISMark home routers [92] to observe how providers parameterize DHCP in existing last-mile deployments.

### **3.5.1 Does ThunderPing observe the duration of failures?**

I am interested in observing the duration of outages with ThunderPing, but can only do so if the host I see fail returns with the same IP address. In this section, I estimate the likelihood of retaining an IP address during outage. On one hand, DHCP includes several features to keep addresses stable. On the other hand, one might suspect that residential ISPs intentionally inject churn into address assignment to manage their addresses or to discourage customers from running servers.

In this section, I discuss the DHCP RFC [26] mechanisms and recommendations for IP address reassignment. Since DHCP servers and ISPs do not necessarily implement the recommendations, I also study some residential hosts to observe how often hosts keep the same IP address after a failure.

## **DHCP mechanisms**

DHCP includes mechanisms that help clients keep the same IP address, both in the standard [26] and in common implementations [15, 51]. The DHCP RFC specifies a lease for an IP address. A lease is a promise that the client can use an address for at least a specified amount of time. If the client fails and recovers before the lease expires, it will retain the same address. If the DHCP server does not renew a lease, it may provide a new address after the old one expires.

What happens, though, when there is a power or link failure and the lease expires during the failure? Section 1 of the DHCP RFC rules states that a server should reissue the same IP address to a client whenever possible.

In practice, DHCP servers use *grace periods* to ensure that expired leases are reissued to the same client. Even after the lease for an address expires, the DHCP server will not reassign the address to a different client until the grace period also expires. The grace period parameter, like the lease duration, is configurable by the ISP and their durations vary. For instance, Cisco Network Registrar, the DHCP server software for many large ISPs such as Comcast [15], has a default grace period of five minutes and the Windows DHCP server has a default grace period of four hours. The Internet Systems Consortium DHCP server, the oldest and most prominent Open Source DHCP server [51], has an infinite grace period. This is not the case for some implementations, like the basic Cisco DHCP server included in IOS which immediately forgets about leases when they expire. However, these are mostly minimal implementations and are thus unlikely to be used by residential Internet providers.



## **DHCP reassignment in practice**

Since I do not know how long leases and grace periods last, I observe dynamic IP address reassignment after a power or link failure in residential links. There are several probing systems that send generic probes out from the customers' networks for Internet measurement, such as BISMark [92], Atlas [82], Samknows [83] and DIMES [87]. I look at the BISMark dataset for my study.

The BISMark dataset comprises reports of the IP addresses assigned to 104 BISMark gateways measured every ten minutes, between September 1, 2011 and March 30, 2012. The dataset contains both public and private (RFC 1918) IP addresses. Among the public addresses, some belong to Georgia Tech; the BISMark researchers reported these to be addresses used for testing purposes. I am interested only in the persistence of public, residential IP addresses, so I ignore those that were private or belonged to Georgia Tech.

Of the 104 BISMark gateways, 43 reported at least one residential IP address. The addresses belonged to a diverse set of ISPs. It is possible that some of these 43 BISMark gateways did not change IP addresses because they were installed at residences that paid for static IP addresses. Information about the service plan was not available, but I suspect static IP addresses to be both unlikely in BISMark and no more likely here than for the sample of addresses ThunderPing probes.

Despite the small size of the current BISMark dataset, it was the best available. Atlas does not reveal the public IP addresses for its gateways due to privacy concerns. DIMES does not reveal the public IP address of its gateways in the public data sets. SamKnows records the last address for each gateway every month, however, these data are not fine-

grained enough for me to determine the effect of short duration (on the order of a few hours) power outages. Also, Samknows only recorded the first 24 bits of every IP address once during their FCC study.

I believe that the question of IP address permanence is important, and research initiatives like Atlas, RIPE and DIMES would benefit from including source IP address details as part of their dataset.

### **IP address reassignment in the BISMark dataset**

I analyze the 43 gateways with at least one public IP address from the BISMark dataset to see how often IP addresses change after a connectivity or power disruption. I also observe the relationship between the duration that a gateway loses power or connectivity, and IP address reassignment.

When a gateway fails to report to the central BISMark server, a transition occurs, with or without change in IP address. Since the measurements were performed every ten minutes, I expect to observe a duration between reports longer than ten minutes. However, BISMark researchers reported a measurement scheduling issue that could also cause a gateway to miss sending a report, meaning that some outages shorter than 25 minutes may be spurious. I show all the data, but trust only the longer failures to be genuine. I refer to the duration that a node fails to report as the downtime of a transition.

Figure 3.14 details the downtimes of the 43 gateways and also shows the duration over which the gateways were measured. In 5,362 of the 5,407 transitions that I recorded, I observe that a gateway retained its IP address. For 29 of the 43 gateways, the address *never* changed. The remaining 14 gateways changed their address at least once, with or

without downtime.

Changes in IP addresses for a given gateway could take place for two reasons. First, the leases of hosts could expire resulting in assignment of new IP addresses which may or may not be from the same prefix. Second, ISPs could be performing IP address renumbering in which case the prefixes of the new IP address are likely to change.

In Figure 3.14, there are a few transitions in the top left of the figure where IP address changes occur with downtimes less than 25 minutes. Since there is little to no downtime, this may be due to IP address renumbering. For downtimes greater than 25 minutes but less than 1000 minutes (top center of the figure), all but one of those IP address transitions were across prefixes, indicating that there is a high likelihood that these address changes were triggered by renumbering. For downtimes greater than 1000 minutes (top right of the figure), I notice address changes both across different prefixes and within the same prefix; these may well be caused by DHCP lease expiration.

Importantly, I observe that even for gateways for which change in addresses occurred, in 1,482 of 1,527 total transitions for these gateways, they retained their addresses. Also, for every gateway that changed addresses for a downtime less than 1000 minutes, there was at least one occasion when the downtime was greater than 1000 minutes for which the address did not change. Since failed hosts return with the same IP address in most cases, I can use ThunderPing to observe the duration of outages.

### **3.5.2 How long do UP →DOWN failures last?**

I now investigate how, when a link fails, the weather conditions at the time of the failure affect the length of time to recover from the failure. After a failure (UP →DOWN transi-

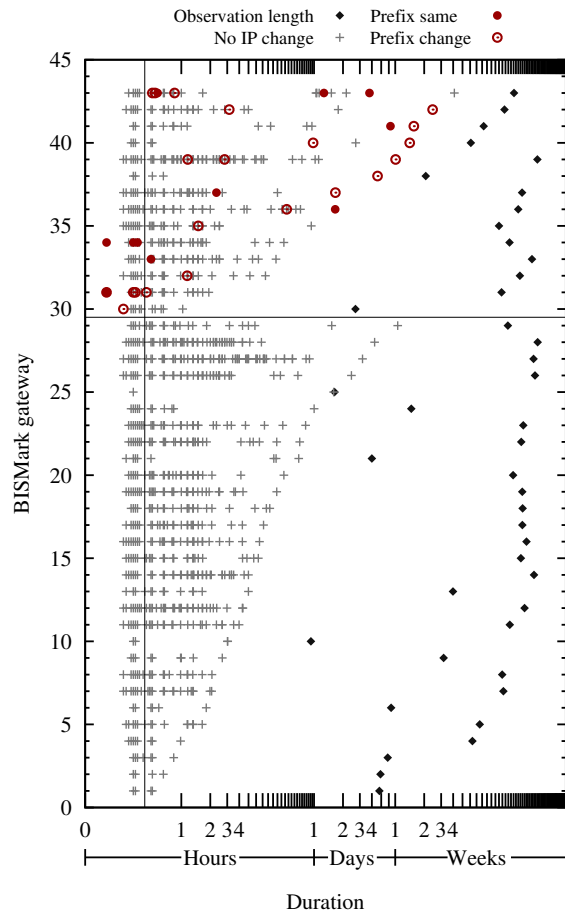


Figure 3.14: Failures and address reassignments in BISMart. Gateways are separated into two categories: ones that experienced IP address reassignment and ones that retained addresses throughout the observed duration. Within each category, the gateways are further sorted by the maximum experienced downtime. The durations are plotted in log scale with the base set to 10. Diamonds indicate the total duration for which each probe was observed. Pluses indicate transitions where a gateway failed to report and retained its IP address when it reported again. Unfilled circles indicate transitions where the address changed across different prefixes. Filled circles indicate transitions where the address changed across the same prefix. The vertical line at 25 minutes on the x-axis delineates possibly incorrect downtime measurements from correct downtime measurements.

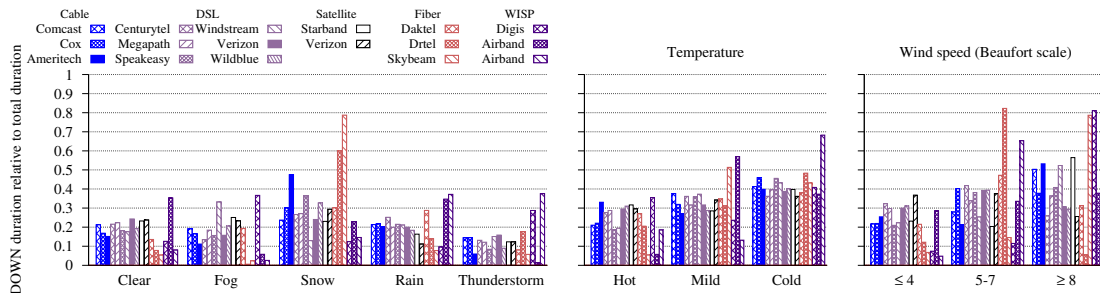


Figure 3.15: Duration **DOWN** after an **UP** → **DOWN** transition. Relative median failure duration for different weather conditions.

tion), the host remains in the **DOWN** state until it eventually recovers by entering an **UP** state. The duration of the **DOWN** state defines the duration of the failure.

The experiment is to take all **UP** → **DOWN** → **UP** transitions and categorize the **DOWN** states by the weather conditions that the link experienced at the beginning of the failure. As described in Section 3.4.1, I do not include failures that look like power failures. I compute the median failure duration per provider and condition.

To compare the durations of failures between providers, I perform the same normalization as the failure rate: for each ISP and category of weather (Precipitation, Temperature, and Wind) I sum all of the median durations and divide them by the total.

Figure 3.15 shows these relative median durations of failures. For precipitation, almost all ISPs have similar median failure duration in Clear, Fog, and Rain. Except for satellite providers that experience about 100% shorter median failures in Rain and Thunderstorm. This is consistent with short satellite signal drops caused by clouds and moisture in the air.

The differences are in Snow and Thunderstorms. Median Snow failure durations are about 50% longer across ISPs than those in Clear. This could be because failures in snow take a long time for the repair vehicles to travel through the snow to repair the damage. Median Thunderstorm failure durations are about 50% shorter than Clear. This could be because of lightning causing transient faults in power, causing equipment such as modems and possibly power equipment to reset. This would not be captured by my model of power outages, which requires a large outage where many providers fail at once.

The explanation of time to reach the repair site could also explain why failures in cold temperatures (where there may be snow or ice on the ground) take about 33% longer to

repair than mild or hot conditions. Also, compared to calm wind, when the wind is at least a 8 on the Beaufort scale, where trees start to break, I observe 50% longer failure durations. This could be because Beaufort 8 winds are likely to be part of large storms that take a long time to attend to.

### 3.6 Summary

In this chapter, I showed that weather conditions affect the failure rate and failure duration of last-mile links. The observations are extensive: over the course of 400 days, Thunder-Ping, my last-mile link pinging system, sent 4 billion pings to 3.5 million IPs. I applied various techniques in order to observe last-mile link failures *with data collected only from public measurement infrastructure*. I removed failures of my servers that appeared in the data as last-mile link failures. I determined failures by link type because ping loss rates differ across link types. Using an edge detection algorithm, I found states of partial responsiveness apparently caused by the nature of different transmission media or by ISP defense systems that blocked my servers. Finally, I determined whether IP reassignment after failures limits my ability to measure failure duration with pings.

Excluding suspected power outages, I observed the following correlation between weather and failure rate: compared to Clear, last-mile links appear to fail about  $1.5\times$  more often in Rain, and  $2.3\times$  more often in Thunderstorms. Surprisingly, Snow did not correlate with an increased failure rate over Clear. High temperature showed  $1.3\times$  the mild temperature failure rate, and cold temperature showed  $0.7\times$  the mild temperature failure rate. High winds resulted in  $2.5\times$  the failure rate of low wind conditions, but only for Cable and DSL links.

I observed the median failure durations in Clear, Fog and Rain are similar. However, the median duration in snow failures was about  $1.5\times$  the median Clear duration. Surprisingly, the median duration of Thunderstorm failures were shorter,  $0.5\times$ , than Clear.

## Chapter 4

# Energy-aware Cellular Data Scheduling

In this chapter, I describe a system that increases the reliability of cellular last-mile links on smartphones by reducing their wasted energy. One reason smartphones waste energy is because applications communicate whenever, regardless of how far away the cellular tower is. In this case, the cellular radio consumes more energy than it would if the smartphone were close to a tower. I show how to remedy this problem without modifying the cellular radio's proprietary hardware, firmware, or even drivers. The only modification required is scheduling applications to communicate when the radio operates efficiently. This means the only OS programming interface that applications will use to control the radio is the OS timer to suspend and wakeup the smartphone.

This seemingly simple strategy requires predicting when a smartphone will be near a tower without consuming more energy than scheduling the communication will save. I show how to predict the efficiency of future communication without using privileged data, such as the radio's proprietary statistics (only available to the radio manufacturer and its partners) or the exact locations of cellular towers (only available to providers). I



only require two statistics that are publicly available via the OS's programming interface to the radio: signal strength and cell tower identification number.

My system, called Bartendr, shows that it is possible to reduce wasted energy from cellular communication. Bartendr can be applied to any smartphone because it does not require *modifications to transmission media or equipment* and it does not require *access to privileged data*.

This chapter makes the following contributions:

First, I show that location alone is not sufficient to predict signal strength because of the hysteresis built into cellular handoff decisions, which results in affinity to the current point of attachment [72]. On the other hand, I show that the pattern of variation in signal strength is quite stable when *location is coupled with direction of travel*. Bartendr, thus, leverages the notion of a track [5], e.g., the route from a user's home to workplace, for its signal prediction. However, using GPS on the phone for identifying its position on a track can be prohibitive in terms of energy cost. Bartendr sidesteps this difficulty by representing a track solely in terms of the identifies of the base stations encountered, together with the corresponding cellular signal strength values. The device's current position and future signal values are identified by *matching against the recorded tracks of signal strength*, thereby completely avoiding the need, and hence the energy cost, of determining the device's physical location.

Second, I develop energy-aware scheduling algorithms for different application workloads. For a syncing workload, where little data is transmitted or received, scheduling is based on finding intervals in which the signal strength exceeds a threshold. For a streaming workload, on the other hand, I develop a dynamic-programming-based algorithm to

find the optimal data download schedule. This algorithm incorporates the predictions of signal quality, and thereby energy per bit, and also the tail energy cost. Note that in this chapter I focus on data download, although I believe energy-aware scheduling is equally applicable to the case of data upload.

Finally, I evaluate Bartendr using extensive simulations based on signal and throughput measurements obtained during actual drives. My experiments have been performed on four cellular networks across two large metropolitan areas, Bangalore in India and Washington, D.C. in the US, and includes 3G networks based on both EVDO and HSDPA. My evaluation demonstrates energy savings of up to 10% for the email sync application, even when the sync operation results in no email being downloaded, implying that the energy savings in this case results only from the lowering of the radio power when the signal is strong. In contrast, my evaluation shows energy savings of up to 60% for the streaming application, where energy-aware scheduling helps save energy by both lowering the radio power used and by cutting the duration of radio activity owing to the higher throughput enabled by a strong signal.

## **4.1 Motivation**

In this section, I argue, based on measurements, that exploiting variation in signal strength can yield energy savings. Table 2.1 lists the mobile devices and networks that I measured. These devices expose signal strength in one of two ways: some provide fine-grained, raw received signal strength indication (RSSI), others provide only six coarse signal levels, corresponding to the (0–5) “bars” displayed on phones. These reported values lack meaningful units.

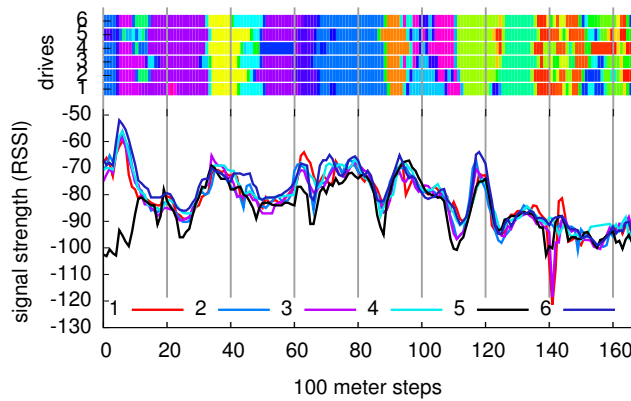


Figure 4.1: Signal varies by location for 6 drives over 17 km. Colors on top indicate base station id.

I focus on saving energy for applications that mostly download because these are most prevalent on mobile devices (e.g., email, news, streaming audio and video). While there are applications that primarily upload (e.g., photo sharing), I do not discuss these here or present measurements of the upload power consumption on mobile phones.

I show that signal strength varies in practice, and that this variation is consistent. Also, I describe how a few typical applications may be sufficiently flexible in scheduling their communication to match the periods of good signal strength.

#### 4.1.1 Signal varies by location

Cellular signal strength varies depending on location because of the physics of wireless signal propagation. Signal strength degrades with distance from the base station and is impeded by obstructions such as trees and buildings. Although the “bars” displayed on phones have made people aware that cellular signal varies across locations, this variation needs to be both significant and consistent for signal strength based scheduling to be effective.

Figure 4.1 plots the signal strength reported by the Palm Pre on each of five drives

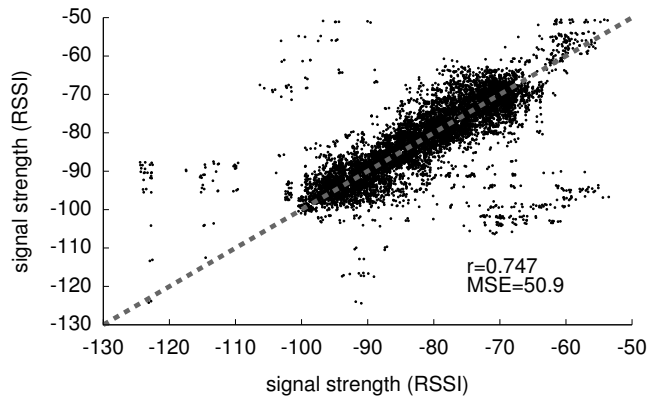


Figure 4.2: Signal variations are consistent for 6 drives over 17 km. Signal correlation for 25 m steps in all drive-pairs. To better illustrate the density of integral signal strength points, I add random noise of -0.5 to 0.5 RSSI to each pair.

along a 17 km highway path from Washington, DC to College Park, MD collected on different days. The colored bars above the graph represent which base station the device is associated with during each track. In presenting these repeated drives of approximately the same path, I take for granted that humans are creatures of habit and that their paths are predictable [48]. Figure 4.1 shows graphically that, despite potential changes in hand-off behavior or environmental effects, recorded traces of signal variation may be able to predict future signal variation along the same path. A majority of the signal variation across drives are small ( $< 5$  RSSI) while a few variations are significant, for example, at the start of drive 5. For this particular drive, unlike the others, the Pre does not keep a steady association to the base station represented by blue. Instead it switches between base stations until close to step 15 when it steadily associates with the violet (dark gray) base station.

For each 25 meter step, I present a scatter plot of signal strength values across all pairs of drives in Figure 4.2. Perfect linear correlation is represented by the 45-degree line; one can see that most of the points in the figure are clustered around this line. The overall

correlation coefficient is 0.75, which indicates that there is significant linear correlation of signal strength values across drives. This validates my hypothesis that signal variation along a path is consistent between drives.

Figure 4.1 shows that the variation of the signal strength along the drive is also significant. The highest and lowest strength values are -50 and -120 RSSI and there are frequent signal strength variations between -90 and -70 RSSI. The cost of communicating at -90 instead of at -70 RSSI entails the use of about 20% additional power (Figure 2.1) and a median throughput that is 50% lower (Figure 2.2). This results in an energy per bit on the Pre that is 2.4 times higher at -90 RSSI compared to at -70 RSSI. Thus, if applications were to preferentially communicate at -70 RSSI instead of at -90 RSSI, the potential communication energy savings are 60%.

## **4.2 Suitable applications**

The approach to saving energy in Bartendr is to defer communication, where possible, until the device moves into a location with better signal strength, or conversely, to prefetch information before the signal degrades. However, not all communication is amenable to adjustments in timing. I now look at two types of common applications that can make use of this approach.

### **4.2.1 Synchronization**

The first application class that I consider is background synchronization, where the device probes a server periodically to check for new mail, updated news, or similar pending messages. Within each periodic interval, the syncing operation can be scheduled whenever the signal is strong. Many devices perform this background synchronization whenever pow-

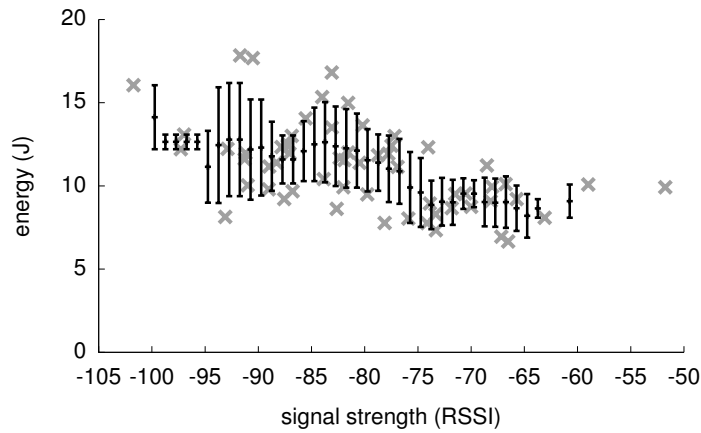


Figure 4.3: Average signal and energy for 60 SSL email syncs on the Palm Pre. Error bars represent  $\mu$  and  $\sigma$  for windows of  $\pm 2.5$  RSSI. Above  $-75$  RSSI, the sync energy is consistently low. Below this threshold, the sync energy is higher and variable.

ered on, meaning that even small savings in energy consumption for each background synchronization operation has the potential to yield large cumulative energy savings.

In order to schedule individual synchronization attempts, I assume that the synchronization interval is flexible. For example, while the application may be configured to check for new mail every five minutes, I allow each individual sync to be spaced, say, four or six minutes apart, while ensuring that the average sync interval remains five minutes. In my evaluation, I focus on the common case of a sync operation that does not result in any new data (e.g., email) being fetched. This represents a lower bound on the potential energy savings. I expect that, by choosing to perform sync more often when the signal is good, actual data communication (e.g., downloading of emails and attachments), if any, would also likely happen during periods of good signal, resulting in additional energy savings.

Figure 4.3 shows the energy consumed for 60 syncs while driving on main roads in two states in the U.S. (20 in Michigan, 40 in Maryland). The signal shown is the average

signal over the sync. The median sync time is 10 seconds, the maximum is 13.2 s and the minimum is 7.4 s. Although the signal strength during a given sync operation is typically stable (median variation is 3 RSSI), it can also be highly variable at times (maximum variation is 22 RSSI). The error bars represent average and standard deviation for  $\pm 2.5$  RSSI windows. Nevertheless, in Figure 4.3, I can set a threshold of approximately -75 RSSI, such that syncs performed when the signal is above the threshold consistently consume less energy than when below the threshold. The average sync at a location with signal above -75 requires 75.3% of the energy when the signal is below -75. The decrease in average sync energy as signal increases matches the power profile of the Pre in Figure 2.1.

Although the differences in energy cost are significant, designing a system to realize these savings is challenging. In the extreme case, when there are no other applications executing on the device, the entire device could be asleep in between each sync operation, so the device must predict when to sync before it goes to sleep, and while it sleeps the prediction can not be updated. A simple syncing schedule that checks precisely every five minutes might miss opportunities to sync in strong signal locations, but for those five minutes, the device uses very little power. Further, any energy expended as part of the computation to predict where the good signal strength location five minutes in the future detracts from the potential savings, if any.

The alternative of push-based notification is not necessarily more energy-efficient than a pull-based approach. Push notification requires the device to be in a (higher-energy) state where it can receive such notifications and also expend energy to maintain a connection to the server as the device traverses many cells. In contrast, a pull-based approach can keep the device in a very low-power suspended state between each sync. Finally,

Run	Power (mW)	Time (s)	Energy (J)
-93 RSSI			
1	1969	85	167
2	1983	83	164
3	1904	82	156
-73 RSSI			
4	1655	86	142
5	1539	68	104
6	1532	187	286
7	1309	85	111
8	1400	76	106
9	1403	71	99

Table 4.1: Even while playing a YouTube video on the Palm Pre, efficient communication saves significant energy. Energy consumed while playing a one minute YouTube video in low (-93 RSSI) and high (-73 RSSI) signal.

even in the case of a device that uses push notifications, Bartendr could be used to decide when to schedule the downloading of large messages, the availability of which is learned through the push notification mechanism.

## 4.2.2 Streaming

Streaming applications such as Internet radio and YouTube are another class of applications that permit flexible communication scheduling, so long as application playout deadlines are met. Streaming sites typically transmit pre-generated content over HTTP. In addition, some of these sites throttle the rate at which the data is streamed to the client, while keeping the client-side buffer non-empty to avoid playout disruptions. I can save energy for these applications by modulating the traffic stream to match the radio energy characteristics: downloading more data in good signal conditions and avoiding communication at poor signal locations. The challenge, however, is to ensure that every packet is delivered to the client before its playout deadline, to avoid any disruption being experienced by the user.



One might question whether the variation in power due to signal strength is significant relative to the baseline power consumed by display and processor during video playback. In fact, video playing is an important worst-case example because it exercises the most energy consuming hardware on the device. To address this question, Table 4.1 presents the total energy cost of downloading and playing a one minute YouTube clip on the Pre at two locations with different signal strengths. In general, energy consumed at -93 RSSI is about 50% higher than the energy consumed at -73 RSSI. In other words, communication energy savings are significant, even while most of the phone's hardware is active. However, in run 6, energy consumed at -73 RSSI is higher than energy consumed at -93 RSSI, because of lower bandwidth at -73 RSSI for this experiment (perhaps due to competing users). This bandwidth variation can pose a significant challenge in delivering the full energy savings of scheduling at good signal locations.

Given the above findings that indicate cellular power consumption is significant relative to processor and display power consumption, in later experiments where I evaluate Bartendr, I will ignore these devices, and focus the measurement solely on the cellular energy. Doing so also helps me avoid noise due to fluctuations in the energy consumed by the other components.

Although my focus is on streaming, with its intermediate deadlines for maintaining uninterrupted playback, bulk transfers could be equally amenable to scheduling. Consider the tasks of uploading photographs and downloading podcasts: although these tasks should be completed within a reasonable time, a user might appreciate if the transfers also placed a minimal cost on battery life.

## 4.3 Architecture

In this section, I introduce Bartendr. Bartendr strives for energy efficiency by scheduling communication during periods of strong signal. To accomplish this, it predicts signal strength minutes into the future. For example, Bartendr can predict efficient times to wake up and sync email, and intervals when data should be downloaded. First, I describe how Bartendr uses prior tracks of signal strength to predict future signal strength. Then, I compare tracks to alternate methods of signal prediction based on location and history. Finally, I present algorithms that use the predicted signal strength to efficiently schedule syncs and streaming media.

### 4.3.1 Predicting signal with signal tracks

Bartendr predicts signal strength for a phone moving along a path. As I saw in Figures 4.1 and 4.2, signal strength is consistent at the granularity of 25 and 100 m steps along a path. This consistency means that Bartendr could, in principle, predict signal strength along a path using previous signal measurements, captured while traveling along the same path. Before discussing the challenges involved in accomplishing this, I lay out my assumptions. I assume that users will, in general, store several of these *signal tracks* on their phone, corresponding to the paths that they frequently travel on. I further assume that Bartendr will be able to infer the current track of the mobile phone. The track could be identified with high probability using mobility prediction techniques [48].

Predicting signal strength on a signal track requires two steps: finding the current position of the phone on the track, and predicting the signal in the future starting from that position. GPS could be used to locate a phone on a track, but doing so would drain

considerable energy and would detract from the energy savings sought by Bartendr. Instead, Bartendr locates itself on a signal track by finding the measurement in the track that is *closest* to its current signal measurement. Signal measurements come at no extra energy cost because the phone's cellular protocol needs them for handoff [95]. Of course, there may be several points on a signal track with the same signal strength, so each signal strength sample in the track also include a neighbor list: a list of all the phone's neighboring base stations sorted by signal strength. Bartendr's current position in the track is the one that has the most matching neighbors (in order) and the closest signal strength. While this approach of signal tuple-based matching in Bartendr is similar to that used for localization in prior work [50], all of the computation in Bartendr is confined to signal space, without any reference to physical location.

I find that the closest match heuristic works well for Bartendr's needs, but errors are possible. The signal strength approaching and leaving a cell may be similar, and the closest neighbor list might not disambiguate these two positions. Further, if the signal is not changing often, perhaps in an environment of sparsely populated terrain, the closest match may be ambiguous. I observed very few occurrences of such errors in my testing.

Once Bartendr determines the phone's location on the track, it predicts signal strength minutes into the future. With signal tracks this means simply looking ahead in time from the current location. Although signal is consistent for locations along a path, the time to travel to a location along the path is not. For example, if the phone is traveling along a road it may travel at different speeds or stop at different times. This problem can be mitigated by constantly updating the phone's location on the track. For the evaluation of Bartendr in Section 4.4, Bartendr skips over stops in the signal tracks, although this

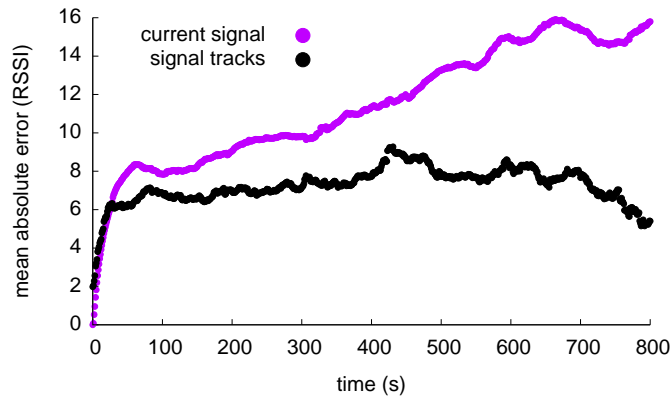


Figure 4.4: Average prediction error from all 25 m steps in six 17 km tracks. Signal tracks predict signal strength with lower error than predicting future signal to be the same as the current signal. Signal tracks can also be used for long term predictions.

provides minimal benefit.

I now compare Bartendr’s signal track predictor with a simple predictor that only uses the current observed signal strength, which is averaged over a few seconds, to predict future signal strength. I choose a position every 25 m on each of the six 17 km tracks as the starting position for the prediction. Using the two methods, I computed the error of signal strength predictions up to 800 s in the future. I ran both predictors on all of the starting positions in all six tracks. I also used all six tracks as previous tracks for the signal track predictor.

Figure 4.4 shows the average absolute error (y-axis) of all signal predictions 0 s to 800 s in the future (x-axis). Signal appears to vary about 6 RSSI over short intervals (< 20 s). I find that predicting past 20 s in the future, the signal track based predictor has lower average error than the current signal based predictor. Signal tracks can also predict signal strength 800 s in the future without a significant increase in error.

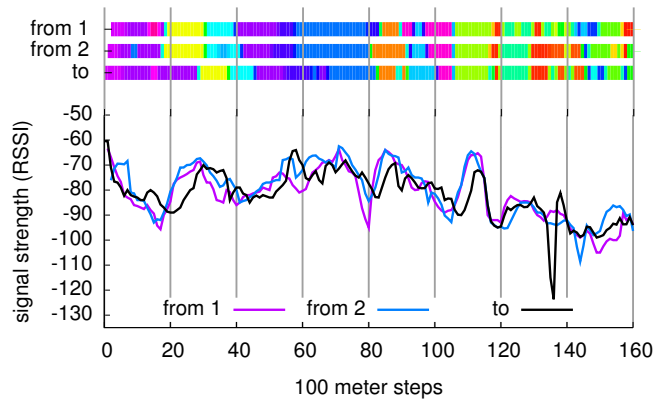


Figure 4.5: Tracks from opposite directions may not align in signal strength. Two “from” tracks compared to a representative “to” track from Figure 4.1.

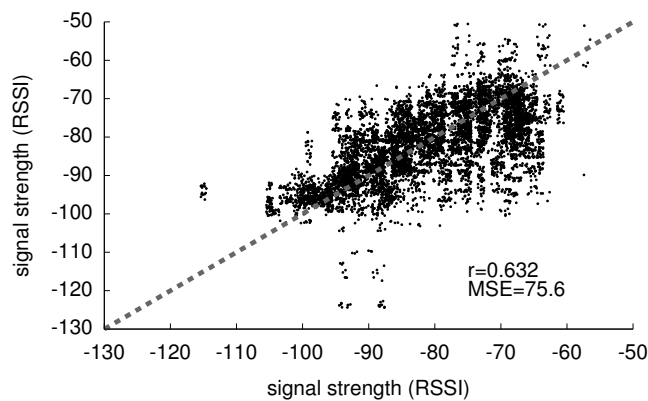


Figure 4.6: Signal correlation of 25 m steps in all pairs of the two “from” tracks with the six “to” tracks. The dashed line represents the ideal correlation.

### Location alone is not sufficient

One might expect that the precise location of GPS, if made for “free” in terms of energy cost, would yield better estimates of signal than would be possible with Bartendr’s signal track predictor. However, my measurements show that signal strength can be significantly different at a location based on how the device arrived there, for example, the direction of arrival. Figure 4.5 depicts the average signal strength for each 100 m step of tracks collected while traveling in *opposite* directions (“from” and “to”). Compared to signal strength values over a track when traveling in the same direction (Figure 4.1), it is clear

that there is less correlation when traveling in opposite directions. The lower correlation values of a signal at a location when traveling in opposite directions (Figure 4.6) compared to traveling in the same direction (Figure 4.2) provides further evidence that location alone is not sufficient for signal strength prediction.

The identity of the cellular base station that the phone is attached to at a given location (color-coded at the top of the graph) does not match at many steps across tracks in opposite directions. I believe that the hysteresis in the cellular handoff process explains this effect. A phone switches from its current attached base station only when its received signal strength dips below the signal strength from the next base station by more than a threshold [72]. Thus, phones traveling in different directions may be attached to different base stations at a given location. This observation implies that incorporating direction of travel with location, i.e., a track [5], is necessary for accurately predicting signal strength on a path.

### **4.3.2 Scheduling sync**

After locating the device on a stored signal track, I must determine when to schedule the next sync. Recall that for sync, my goal is to put the processor to sleep for a calculated interval, so that the phone wakes up when it is at a strong signal location. If the prediction is incorrect, perhaps because the device traversed a track more quickly or more slowly than expected, the device may sync in a location of poor signal or attempt to defer for a few seconds to catch the good signal if it is soon to come.

I propose two threshold-based techniques to find the location to sync given my current location in the track, *first above threshold* and *widest above threshold*. The threshold used

in both techniques represents significant power savings: in Figure 4.3, reasonable threshold values span -76 to -74 RSSI, from which I picked -75 RSSI. The *first* approach waits to sync until the first time -75 RSSI is crossed in the stored track; the *widest* approach waits (potentially longer) for the time at which the stored track exceeds -75 RSSI for the widest interval. *First* could limit prediction mistakes if it is easier to predict near-term events; *widest* could limit prediction mistakes if the wide intervals represent easy targets.

Once the future time for sync is predicted by one of these techniques, the device is suspended to a very low power state until the predicted time. If the user on the track travels at a speed significantly different from that of the historical tracks, the wakeup could occur at a different signal strength value compared to prediction, possibly resulting in diminished energy savings.

### 4.3.3 Scheduling streaming

I next look at scheduling communication for the streaming application. When streaming, the device remains powered on and continuously determines position, so that unlike syncs, errors due to speed variations can be compensated for dynamically.

I now look at how to efficiently schedule a data stream of size  $S$  over certain duration of time  $T$  with minimal energy. To make the problem tractable, I divide the input stream into fixed size chunks of  $N$  frames, and time  $T$  is divided into slots. A slot is defined as the period of time where a single frame can be transmitted. Since data rates are not fixed, each slot can be of variable width depending on the expected datarate at that time. The power consumed to transmit a frame in a slot is also variable. I use the predicted signal strengths and median observed throughput values for the scheduling interval  $T$  to

estimate the slot widths and average power consumption for each slot. Given a predicted  $signal_\ell$  in slot  $\ell$ , I calculate the communication energy as follows:

$$\frac{\text{Signal\_to\_Power}(signal_\ell) * \frac{S}{N}}{\text{Signal\_to\_Throughput}(signal_\ell)}$$

The two functions in this expression map a signal value to the corresponding power value and median throughput value. The mapping is done based on empirical measurements as described in Section 4.1.

Given  $N$  frames and  $M$  slots, where  $N \leq M$ , the optimal scheduling problem seeks to find an assignment for each frame to one of the slots, such that the total energy required to transmit  $N$  frames is the minimum of all possible assignments. One approach is to greedily schedule the frames in the best  $N$  slots which incur the least energy for communication. However, this approach ignores the cost of tail energy incurred every time there is a communication. When multiple frames are scheduled in consecutive slots, the entire batch of frames incur only one tail, as opposed to a tail for each frame if they are spaced out in time. A greedy approach that ignores the radio tail behavior can be very inefficient.

Thus, the scheduling algorithm should take into account both the energy required for communication and the tail energy incurred for the schedule. Let us look at how to compute the tail energy overhead for a schedule. When there are no frames scheduled for a certain period of time prior to the current slot, the radio is in an idle state at the beginning of the slot. If a frame is sent during this slot, the radio switches to its active state, and remains in the active state for the duration of at least one tail period. However, if a frame



is scheduled in a slot when the radio is already in active state due to some transmission in the prior slots, none or only a fraction of the tail energy needs to be accounted. I now describe a dynamic programming formulation that computes the minimum energy schedule given the energy cost of transmission in each slot, accounting for these various tail overheads.

Let  $E_{k,t}$  be the minimum energy required to transmit  $k$  frames in  $t$  timeslots. Corresponding to this minimum energy schedule, the variable  $Last_{k,t}$  stores the slot number where the  $k$ th frame is scheduled. Let  $ESlot_\ell$  be the sum of the communication energy required to transmit a frame in slot  $\ell$  and the incremental tail energy cost, given the transmissions that occurred in the previous slots. The dynamic programming algorithm that computes the minimum energy schedule is as follows:

**Initialization**

**for**  $t = 1$  to  $M$  **do**

$$E_{0,t} = 0$$

**end for**

**Computing optimal schedules**

**for**  $k = 1$  to  $N$  **do**

**for**  $t = k$  to  $M$  **do**

$$E_{k,t} = \min_{\ell=k-1}^{t-1} ( E_{k-1,\ell} + ESlot_{\ell+1} )$$

$Last_{k,t} = \ell$  value for which the previous quantity was minimized

**end for**

**end for**

The intuition behind the dynamic programming algorithm is that the minimum energy to transfer  $k$  frames in time  $t$ ,  $E_{k,t}$ , is simply the minimum of sum of transferring  $k - 1$  frames in time  $(k - 1$  to  $t - 1)$  and the cost of transferring the  $k'$ th frame in the time remaining, including incurred tail costs, if any. Thus, the optimal substructure property holds and the solution to the dynamic program is the same as the optimal solution.

Additional timing constraints for a frame can be easily incorporated in this algorithm by restricting the search for minimum value within the arrival and deadline slots for the frame as follows:

$$E_{k,t} = \min_{\ell=Arrival(k)-1}^{Deadline(k)-1} ( E_{k-1,\ell} + ESlot_{\ell+1} )$$

The value  $E_{N,M}$  is the minimum energy for the predicted schedule, which can be computed by tracing backwards from  $Last_{N,M}$ . The order of the above algorithm is  $O(M^2 \times N)$ .

The algorithm could suffer from two kinds of errors: 1) the speed of the current track being different from speed of the previous track, and 2) the expected throughput at a slot being different from the median throughput in the track. Fortunately, since the device continues to remain powered on for running this application, I can simply re-run the dynamic programming algorithm from the point of discrepancy and recompute the optimal schedule. In my evaluations, this recomputation helped avoid significant deterioration in energy savings on account of the above errors.

#### 4.4 Simulation-based evaluation

In this section, I simulate Bartendr with the 17 km tracks shown in Figure 4.1. While driving these tracks I collected the Palm Pre's signal strength and throughput. I model the phone's power with the measurements shown in Figure 2.1. The advantage of using a simulator is that I can compare the performance of different approaches against each other as well as compare their performance against an optimal algorithm that has full knowledge of future signal.

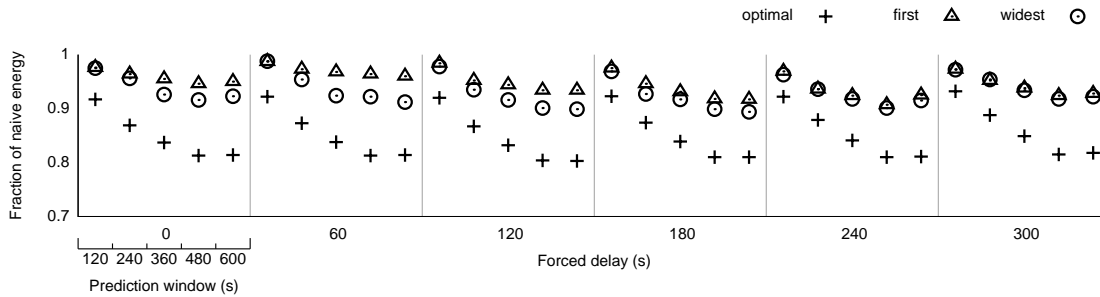


Figure 4.7: Based on the Pre’s radio power profile, predicting signal with previous traces can save energy for email syncs. Median energy for 42 pairs of experiment and training traces with a maximum ten minute scheduling window. Y-axis starts at my expected best savings.

### 4.4.1 Syncing

In this section, I show that the variation of signal is amenable to energy savings by an optimal algorithm and that the prediction algorithms (first and widest, Section 4.3.2) are able to approach that optimal reduction. I run the simulation on all pairs of seven 17 km tracks (training and experiment). Although likely valuable, I do not yet have a scheme for combining different tracks to form a refined track model.

At every ten second interval of the experiment track, I execute the prediction given two constraints: the *forced delay* represents the minimum time that must be slept before performing a sync, while the *prediction window* represents the interval of time after the forced delay where a sync may occur. The decomposition into these two parameters allows me to model various application-level choices about how to constrain the schedule, and provides information about how much latitude is required and whether prediction accuracy degrades with time.

I assume all syncs take a fixed time of 10 seconds (the median observed in Section 4.2.1). It is possible that syncs can take more or less time because of latency variations.

Figure 4.7 presents the total sync energy for *widest*, *first*, and optimal. It shows the sync energy for these techniques relative the naive approach (always sync immediately after the end of the forced delay period, equivalent to a prediction window of zero). Optimal scheduling can provide up to 20% sync energy savings, when it is possible to choose over prediction windows longer than six minutes. Optimal uses future knowledge of the current track to make decisions, and will always choose a low-energy period if one is available. I note from this graph the potential energy savings resulting from increased flexibility in scheduling communication events.

The *widest* scheduling approach generally outperforms *first*, offering up to 10% reduction in sync energy. I suppose that the advantage of widest is due to short-term variations in mobility that do not accumulate over long enough intervals. That is, the variations may cancel each other out.

While these savings are modest, they represent savings for the case where the sync operation does not result in downloading of any content. Thus, the energy savings are only due to reduced transmission power from good signal locations and does not benefit from better data rates available at these locations. When the sync operations result in downloading of updated content, the higher data rates available at good signal locations should substantially improve energy savings (see next section on scheduling streaming).

Finally, I expect that the effectiveness of these scheduling approaches could be improved by better aggregation of training data (omitting track 5 seen in Figure 4.1 in particular alters the results substantially). Further, by explicitly considering variations in mobility I would have further potential to increase scheduling accuracy.

## 4.4.2 Streaming

I compare the performance of a naive approach to Bartendr’s signal-based scheduling algorithm for the Streaming application through data driven simulations. Again, I use the data collected from real driving experiments, which consist of signal strength, instantaneous throughput and power consumption measurements while receiving TCP streams over several drives, to drive the custom simulator. I analyze the energy consumed to download streams of varying bitrates (corresponding to popular audio/video encoding rates) and varying stream lengths (120 s to 600 s of play length). The bitrates play a role in how fast the application consumes the downloaded data, and thus impacts the how long a data frame in the stream can be delayed. The stream length determines the total number of data frames in the stream that need to be downloaded.

In the naive case, all data frames in the stream are downloaded in one shot from the beginning of the stream until completion. In the signal-based scheduling algorithm, I plug in the real signal values from the track. I then map these signal values to median throughput and power consumption numbers from prior tracks corresponding to the same drive, which are used in the computation of  $E\text{Slot}_\ell$  values. For each frame in the input stream, I also compute the deadline before which it needs to be scheduled for transmission based on the stream bitrate. I illustrate how the deadlines are calculated through an example. Consider a 5 min audio stream encoded at 128 Kbps (total size  $\approx$  5 MB), and assume the number of data frames  $N$  to be 25 (frame size  $\approx$  200 KB). Since the application consumes data at an average rate of 128 Kbps, at most one data frame is needed every 12 seconds to ensure that the playout buffer doesn’t run out of data. Thus, in this example the deadlines

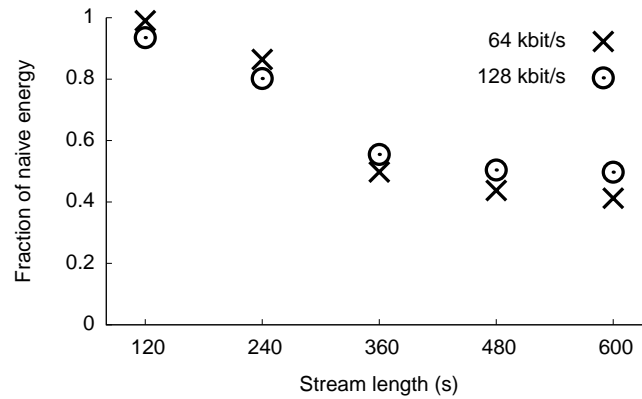


Figure 4.8: Energy savings with signal-based scheduling for 64 and 128Kbps data streams.

for each consecutive frame occurs every 12 s within the 5 min window. After computing the deadlines, I then run the dynamic programming algorithm to compute the schedule for downloading each frame. During the execution of the schedule, if I find that there is a deviation in the actual and expected throughput, I rerun the dynamic programming algorithm with an updated number of frames to be scheduled in the remainder of the interval along with their corresponding deadlines.

To implement the signal-based approach, I need to be able to start and stop the stream download based on the schedule computed by the dynamic programming solution. I achieved this using a network proxy that starts to download the data from the streaming server as soon as the client initiates a request. The schedule basically consists of start and stop times for downloading each frame from the proxy.

I stagger the start to arbitrary times in the tracks and present the average results for over hundred runs. Figure 4.8 plots the energy savings of the signal-based scheduling schemes compared to the naive case. As the stream length increases from 120 s to 600 s, there are more number of data frames that are farther away from the start of the stream.

These frames have longer deadlines and provide more opportunities to search for energy-efficient time slots to download them within their specified deadlines. I see that energy savings of up to 60% are achievable by scheduling using Bartendr.

## **4.5 Related work**

Several studies have results that are relevant to the Bartendr theme, including: mobile prediction of wireless network quality, stability of ubiquitous wireless network quality measurements, scheduling and other approaches for mobile energy savings.

### **4.5.1 Predicting wireless network quality**

The goals of Breadcrumbs [66] closely resemble those of Bartendr: predict network quality at a physical location, and use this knowledge to make applications use the network more efficiently. However, the two systems differ in many ways.

Bartendr seeks to provide energy savings on ubiquitous cellular networks, while Breadcrumbs is tailored to wireless LANs. While Breadcrumbs indexes WiFi bandwidth availability by GPS location (similar to the work of [77] where location is a key context used in determining whether to scan for WiFi availability), I find that for cellular networks, *location coupled with direction of arrival* is necessary for predicting signal strength, and thus available bandwidth.

Furthermore, because of the dynamic nature of wireless LANs, Breadcrumbs requires an energy consuming measurement framework that periodically scans for WiFi bandwidth availability to predict network quality at a location. Bartendr leverages the fact that cellular signal strength information can be obtained very inexpensively on the mobile (since the cellular radio must remain on to receive phone calls) to schedule communication at

appropriate locations.

## **4.5.2 Stability of cellular signals**

Recent studies have quantified the service-level stability of ubiquitous wireless networks [54, 94]. They found that bandwidth changes over time, even though signal strength measurements remain stable. In the earliest study, Tan et al. discovered that over three weeks, even though signal strength measurements were stable at a location, network bandwidth measurements vary by 54% [94]. Later, Liu et al. [54], using more detailed measurements of an EVDO network, found that signal strength and the downlink data rate used by the provider were highly correlated over long timescales. However, they also noticed that over about a months time, the rates fluctuate between 500Kbps and 3Mbps at a given location.

Given this information, can an approach like Bartendr still be effective? First, since I find that energy efficiency is dependent on signal strength, relative signal strength stability and high long term correlation with data rates is certainly helpful. Second, while some bandwidth availability variation is to be expected on a commercial network with competing users, applications like email or RSS feeds that I expect to benefit from Bartendr do not require the maximum throughput of the link.

## **4.6 Summary**

In this chapter, I showed how to increase the reliability of cellular last-mile links by reducing wasted energy. Energy measurements from an example smartphone showed that email syncing in high signal strength areas requires on average 75% of the energy required for email syncing in low strength areas. The measurements also show that streaming a



video in high signal strength areas requires on average 50% of the energy required in low signal strength areas.

I demonstrated that scheduling syncing and streaming for periods of predicted high signal strength can reduce wasted energy. Simulations based on signal strength traces obtained while driving show that scheduling syncing for periods of high signal strength reduces the energy required for syncing by up to 10%. Streaming simulations based on the same signal strength traces indicate that scheduling streaming reduces the energy required for streaming by up to 40%. I also demonstrated that reducing wasted energy does not require *modifying the cellular hardware, firmware, or drivers*, nor does it require *access to privileged cellular data*.

# Chapter 5

## Reliable Broadcast Last-mile Link

In this chapter, I describe how to adapt a widely-deployed, reliable broadcast last-mile link to the Internet by *building off of unmodified transmission media and equipment*. This new link is necessary because existing Internet last-mile links cannot be relied upon to widely distribute critical data. In previous chapters, I showed that fixed links can fail during weather (Chapter 3) and mobile links can (eventually) fail due to wasted energy when their signal strength is low (Chapter 4).

Unlike existing Internet last-mile links, metropolitan VHF radio broadcast systems are inherently reliable. The reliability of VHF broadcast links comes from their ability to be received even if a user is moving in and out of buildings and from their resilience to weather. A single broadcast tower can cover an entire metropolitan area because VHF signals propagate through buildings [58]. Also, atmospheric noise (e.g., lightning) does not significantly affect the VHF spectrum [68]. Worldwide use of the VHF spectrum for critical services such as weather radio, air traffic control, and public safety (e.g., in the US [33], Europe [28], and India [23]), provides evidence of its ability to propagate

through buildings and its lack of interference.

Not only are VHF radio broadcasts inherently reliable, but FM RDS adds even more reliability. FM provides additional robustness because it filters out the noise that line-of-sight lightning may cause in the VHF spectrum [58]. The RDS protocol adds even more robustness: transmissions include 10 error-correcting bits for every 16 data bits.

In Section 2.3.2, I showed that FM RDS has the potential to be an Internet last-mile downlink. In this chapter, I show how to adapt FM RDS to reliably transmit Internet traffic without modifying existing RDS transmitters and receivers.

I describe and implement a server that enables multiple Internet senders to transmit over a single RDS transmitter. This is possible because there is a standard protocol to control commercially available RDS transmitters: the Universal Encoder Communication Protocol (UECP) [29].

In order to make RDS a reliable Internet last-mile downlink, I implement a system on top of Open Data Access (ODA) [80], RDS's arbitrary data protocol. My system, called *Abbie*, provides hierarchical energy-efficient schedules for multiple Internet senders, a receiver-driven Distributed Hash Table (DHT) for last-resort retransmissions, and lightweight digital signatures for verifying authenticity of transmissions. I also implement and evaluate receiver software that can operate on smartphones, low-power microcontrollers, and PCs. To demonstrate incremental deployment to residential networks, I design an RDS-to-LAN bridge that connects over USB to residential routers. In summary, I show that by adapting the FM RDS system to the Internet, it is possible to create a reliable Internet last-mile downlink *without modifying link transmission media and equipment*.

I evaluate both the underlying RDS transmission medium and the Abbie system I built on top of it to the extent possible, using FCC and Census data, benchmarks from hardware prototype receivers and mobile phones, and real-life deployment on a commercial 3 kW station sending to my USB prototype devices attached to home routers. (Section 5.4)

## 5.1 The reliability of FM RDS metropolitan radio broadcasting

I propose Abbie, a broadcast system that is not just best-effort broadcast, it is reliable. If Abbie receivers miss a message transmitted over radio broadcast, they can obtain re-transmissions of messages over Internet last-mile links. For the applications that work well with the deployment of FM RDS transmitters (Section 5.3), this broadcast system is mostly reliable by its nature. However, to make it completely reliable, I also add a capability for last-resort retransmissions. According to Chang and Maxemchuk, the properties I must achieve as a terminating reliable broadcast system are formally defined as follows [11]:

**Validity:** If the transmitter broadcasts a message, then all correct receivers must process the message.

**Integrity:** Receivers only deliver each message one time, and those messages must have been broadcast by the transmitter at some point in the past.

**Agreement:** If a correct receiver processes a message, then all correct receivers must process that message.

**Termination:** If the transmitter that broadcasts a message stays up, then all processes that remain up eventually process that message.

This definition of reliable broadcast does not allow for receivers that are correct, but do not desire the message being broadcast. The reason is that reliable broadcast systems generally assume they are being implemented for data that all are interested in or for use by systems that all hosts participate in. Additionally, they assume broadcasts are sent over unicast links. My goal is to envelop as many devices as possible with the possibility of obtaining the broadcast messages—no last-mile link is left out. To allow for these receivers, I slightly modify the definition of validity, agreement, and termination.

**Validity:** If the transmitter broadcasts a message, then all correct receivers *who are interested* must deliver the message.

**Agreement:** If a correct receiver delivers a message, then all correct receivers *who are interested* must deliver that message.

**Termination:** If the process that delivers a message stays up, then all processes that remain up *and desire the message* eventually will deliver that message.

For validity, most receivers will get this property due to the extremely low loss nature of the FM RDS (Section 5.1.1). However, in the unlikely circumstance that there is a loss, every broadcast transmission is available in a DHT (Section 5.2.4). The applications that will use this broadcast system provide content that is relevant to some subset of receivers covered by the transmitter's signal. These broadcast towers are located in populated areas (Section 5.1.2), so almost any location is an option to receive locally relevant content. Finally, Abbie's always on USB-to-LAN bridge hardware provides a middle ground for

receivers that are not always-on, and yet they still can receive broadcasts without accessing the DHT (Section 5.2.3).

For integrity, the transmitter digitally signs every message, and every message includes a transmitter-assigned sequence number (Section 5.2.2). However, the digital signature required to verify the signatures cannot be too large, because broadcast links are severely bandwidth constrained. They also cannot require heavy processing, as some receivers may be power- and processing-constrained (Section 5.2.3). The properties of validity and integrity combine to give agreement.

For termination, there is very little cost to store every transmission (Section 5.1.3). This means that as long as a host is online, it has no reason to delete transmissions that may even be several years old. Eventually consistent protocols typically require truncation, but given that the set of intended receivers is not known to Abbie, this is not possible.

In the following subsections, I present observations that provide support for the reliability and wide deployment of the FM RDS broadcasting system.

### **5.1.1 FM RDS loss rate is low**

I tested the RDS reception distance with my prototype receiver (Section 5.2.3) by observing the RDS message loss rate while driving. I tuned the receiver to a moderate power 3 kW FM station in the middle of the city. (Typical campus radio transmission power is approximately 10 W; popular radio stations may transmit at 50 kW.) Then I drove on the main highways around the transmitter. Figure 5.1 shows the observations from this drive. I computed the RDS loss rate in the following way: RDS messages come in four 16 bit groups, so I consider a message successfully received when all four 16 bit groups

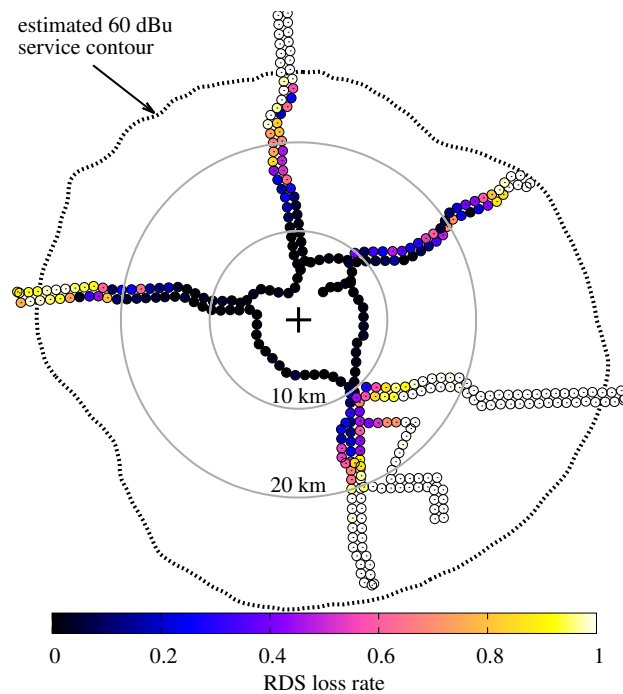


Figure 5.1: Loss rate of RDS messages from a 3 kW FM transmitter (+) to my prototype receiver board. Observed while driving the major highways of a metropolitan area. The dots show the loss rate computed every 1 km. Forward and return trip separated by 1 km for clarity.

are received without errors, or when all of the group error protection bits repaired the errors. RDS error protection can detect one- and two-bit errors in a 16 bit block, as well as any burst spanning 10 bits or less. It can correct bursts of 5 bits or less. Radio stations broadcast RDS messages at a fixed rate of 11.4 messages per second [46]. I infer losses from gaps in the timestamps of received messages.

Figure 5.1 shows that very few messages are lost within a 10 km radius from the transmitter. Also, at fairly high loss rates, the prototype’s short antenna picks up RDS messages while close to most of the estimated service contour from the FCC. The FCC provides geographic service contour estimates for licensed FM transmitters [30]. The loss measurements are consistent with a BBC RDS propagation study performed in 1987 with a large antenna mounted on a truck [55]. A remaining question is: how are errors

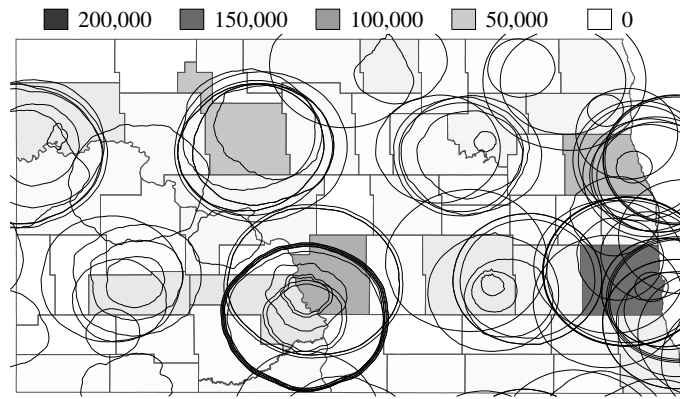


Figure 5.2: A map showing the overlapping FCC FM signal contours and Census county population estimates (shading gradient) for North Dakota. Due to non-uniform horizontal gain of the antennas, the signal contours are not simply discs. This map indicates that wherever there are people, there are several overlapping FM transmitters.

dispersed? For 20 hours I recorded RDS messages from a non-mobile receiver that had a 4.3% average loss rate. The time between errors has a mean of 2.04 and a standard deviation of 2.85 seconds. This indicates errors are not evenly spaced in time. Even if there are errors, some entire messages may get through.

The RDS signal is typically mixed in at 5% of the transmitted power of the rest of the transmission, an imbalance that protects the audio signal from interference from this extra service. This imbalance means that audio can sometimes be received and played at the edges of the transmission range where RDS can no longer be decoded.

### 5.1.2 Where there are people, there are FM towers

One benefit of broadcast is receiver scalability: a single transmission reaches many. I next try to understand just how many people can receive a transmission and whether there could be enough recipients to make it cost-effective. To that end, I study the distribution of people covered by FM stations in the US. Assuming that every person has a receiver (a modest assumption if FM receivers are mandated in cell phones [63]) I can compute



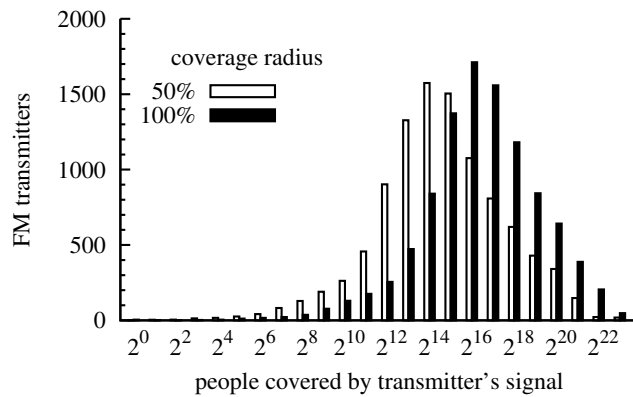


Figure 5.3: Histogram of the number of people covered (x-axis) by each of the FM transmitters in the US (y-axis). Computed from the FCC's estimated signal contours and population data from the 2009 US Census county population estimates. Most FM transmitters licensed by the FCC in the US cover at least 100,000 people. When limited to 50% of the coverage radius (one fourth of the coverage area) the towers cover one fourth as many people.

the equivalent unicast bandwidth that would be needed to send the message to that many people.

To estimate the number of people covered by licensed FM transmitters, I intersect signal coverage data (FCC service contours) with population data. These contours are based on antenna location, height, and when available, design and orientation. They also consider topographic features, such as mountains, that may block the signal. The RDS signal may not extend to the edge of the contour, therefore I also investigate a more conservative, half radius contour. Figure 5.2 provides an example of what these signal contours look like.

I estimate the population covered by FM stations based on regional population data from the US Census Bureau [97]. I intersect the FCC signal contours with the geographic borders of the population regions (counties). For simplicity, I assume the people in the population regions are evenly distributed: when a signal covers part of a county I assume

the fraction of the population covered is equal to the fraction of the area covered. This assumption likely causes an underestimate of how many people are within range of a tower: I expect that people are in fact distributed unevenly in counties, with more people closer to population centers and FM towers, and fewer people farther away, but my analysis conservatively does not consider this expectation (Figure 5.2).

Figure 5.3 shows a histogram of population covered by every FM station in the US. According to the full radius contour, most of the 10,075 FM stations in the US cover at least 100,000 people, and 13% cover more than 1 million people. The coverage distribution for the half radius contour is a factor of four smaller than the full radius, consistent with their difference in area. With such large populations covered, the unicast equivalent bandwidth of FM RDS can be significant. Assuming all 100,000 people have receivers, and the RDS goodput is 421.8 bps, the effective bandwidth of a broadcast receivers is 42.18 Mbps. For the 43 stations that cover at least 10 million people (New York City and Los Angeles), the effective bandwidth of one RDS stream is 4.22 Gbit/s. Although one might wonder whether a deployment could possibly include every man, woman, and child in Los Angeles, there are efforts to mandate FM receivers in cell phones [63] and according to a 2011 survey of cell companies in the US, there are more cell subscriptions (322 million) than people (304 million) [20]. I omit stations that appear to serve fewer than 16 people; they are all in Alaska and reflect how my analysis underestimates the coverage for large counties with tiny populations.

It should be noted that the FM service contours are an estimate of coverage. For example, the southeastern corner of the contour from my drive shown earlier in Figure 5.1 shows significantly greater RDS loss rate than the rest of the map. A conversation with

the radio station's engineer revealed the antenna is oriented northwest, and the design and mount of the antenna makes it less than omni-directional. Also, the altitude drops off in the southeastern direction, so the antenna is below the strongest vertical gain field (the strongest is directly parallel to the ground from the antenna). The FCC's license for this station does not contain the station's antenna horizontal gain in each direction. This investigation reveals a problem with the FCC service contour estimates: they are only as good as the information about the antenna that the stations provide the FCC.

### **FM RDS transmitters are pervasive and inexpensive**

Transmitting RDS data requires a special-purpose encoder and a generic PC server, connected to each other typically by serial or Ethernet. An encoder typically costs approximately \$2,000, making them widely deployed: In 2008, several major US broadcast companies reported that at least 450 of their stations broadcast RDS [16]. Once the data are encoded, the RDS signal is mixed with the audio, FM modulated, then transmitted.

### **5.1.3 Every transmission can be stored**

The limited transmission rate of RDS means a receiver can store indiscriminately and indefinitely. The raw bit rate of RDS is only 1187.5 bits per second; after error correction and header information, this raw bit rate is significantly reduced. I use the standard RDS Open Data Application (ODA) framework to embed my data, which leaves 421.8 bps of goodput for Abbie. A 16 GB MicroSD card can store ten years of continuous broadcasts.

### **5.1.4 FM Receivers are small**

Installing a broadcast receiver in networked devices and embedded systems that are not traditionally networked, necessitates a receiver IC (Integrated Circuit) that is inexpensive,

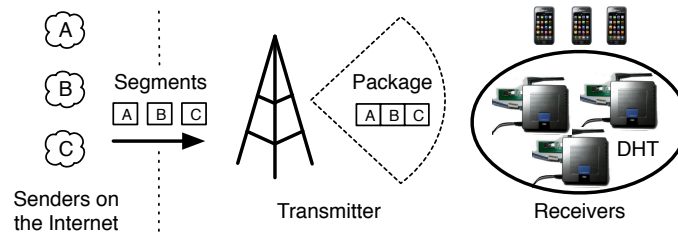


Figure 5.4: Abbie system overview

small, and power friendly. Silicon Labs (si47xx) and Philips (SAA6588) provide FM RDS ICs with these properties. The Silicon Labs si4705 IC is 3 mm<sup>2</sup> and requires only a few external components. The IC consumes only 64 mW while receiving RDS.

It is challenging to design a compact antenna for the FM band (100 MHz). The FM wavelength is 3 meters. A quarter wavelength antenna, which would allow the antenna to resonate and collect a stronger signal, would be 75 cm long. This requirement is why portable FM receivers in smartphones and music players often use the headphone wire as an antenna. Yu et al. measure the effectiveness of headphone antenna cable lengths to estimate the range of FM broadcasts [102].

A short monopole antenna with a loading coil can resonate in frequencies with wavelengths much longer than the antenna itself. My prototype (Section 5.2.3) features 1/30th wavelength monopole antenna trace on the PCB. The drawback of a short embedded PCB antenna is extra board space: There must be space between the components and the antenna.

## 5.2 Design and implementation of Abbie

Abbie provides a mechanism to share a broadcast link that consists of an over-the-air protocol, software running at the tower, receiver software, and RDS-to-LAN bridge hard-

ware (Figure 5.4). The over-the-air protocol operates within the RDS specification. It is data efficient, and it supports sleeping for low power receivers. The receiver software provides an interface for applications, that is platform independent and includes a DHT for searching for retransmissions of broadcasts.

The protocol I built on top of FM RDS to adapt it to be an Internet last-mile link operates as follows: The tower transmits schedules that contain hierarchical names that allow a receiver to subscribe to a provider, or specific content, and this allow energy inefficient receivers to sleep. The system can be incrementally deployed with an RDS-to-LAN bridge USB hardware. In the rare case that a receiver fails to demodulate a transmission, I adapt a DHT to provide a last-resort place to look for retransmissions and I show that the transmissions can be authenticated with digital signatures, even on low power processors.

### **5.2.1 Tower sharing protocol**

Installed at the radio station, the Abbie transmitter software runs on an Internet-connected computer that can send data to the RDS encoder with Ethernet or serial connection. A sender anywhere on the Internet sends the tower the segment name, and segment that it would like to transmit. The transmitter can authenticate, authorize, and charge senders, but this is outside of the scope of this work.

### **5.2.2 Over-the-air protocol**

To allow for low power receivers to sleep, the tower should commit to a schedule of segments for a short period of time. This commitment is the highest level of framing from a tower, I call it a *package*. A package contains a sequence number and schedule which

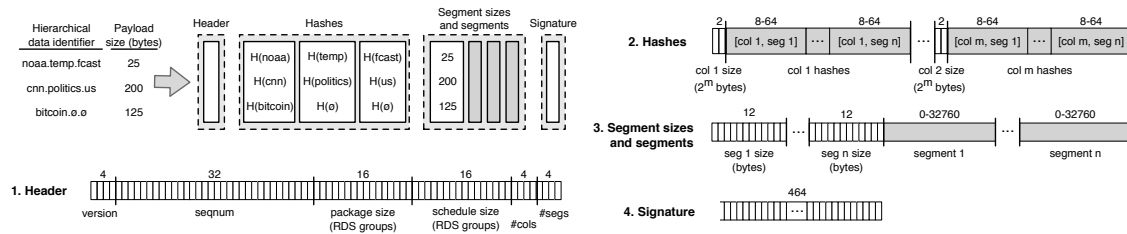


Figure 5.5: Abbie package structure

maps from hierarchical content names to the time when their data (called *segments*) will be broadcast. In a package, after the schedule is the *segments* advertised in the schedule. The final data in the package is a *signature* that binds the package to the tower which allows receivers to verify future retransmissions of the package.

*Framing* The beginning of a package, the segments within a package, and the signature are delimited. This is driven by the principle that a radio's wake-up time varies and a receiver should be able to detect the start of the schedule or segment they want to receive. I use the RDS B group as a delimiter (16 modifiable bits) then I follow it with the data in RDS A groups (37 modifiable bits).

*Naming* To allow receivers to know what segments are coming in a package, I must provide a naming scheme that is expressive, but compact enough to work on low-bandwidth broadcast links. For expressiveness, Abbie supports hierarchical namespaces. As an example, noaa.temp.fcast can be used to represent a temperature forecast message from NOAA. Abbie imposes no restrictions on the namespace; applications may prefer to use self-certifying instead of human-readable names.

For conciseness, Abbie constructs SHA256 hashes of each level of the namespace hierarchy. In my example, noaa.temp.fcast would result in  $H(\text{noaa})$ ,  $H(\text{temp})$ , and  $H(\text{fcast})$ . More specific levels have a lower probability of collision within the higher

level since they inherit the specificity of the higher levels. So I propose decreasing hash size for more specific levels in the name. For example  $H(\text{noaa})$  is 8 bytes,  $H(\text{temp})$  is 4 bytes,  $H(\text{fcast})$  is 2 bytes.

This is similar to naming schemes proposed in information-centric networking [34]; except for compactness I do not include the principal who owns the data; the notion of requiring data ownership does not directly translate to broadcast, but applications in Abbie may include sender verification in their payloads. Also, information-centric networking does not require compactness to the same extent.

*Power saving* The order of the schedule (Figure 5.5) is such that low power receivers can shut off in the middle of the schedule, as soon as they determine that the package does not contain any segments they are interested in. Low power receivers can also turn off between segments they are subscribed to. The package and schedule sizes appear early in the schedule header so that, at any point, the receiver can turn off and skip over the rest of the schedule or the rest of the package. Skipping a small segment or schedule may not save any energy because of the wake and sleep time and power consumption of the RDS hardware and the device it is running, but skipping longer segments will save energy.

In the schedule, I represent the segment name hashes with a matrix (Figure 5.5 top left), where each row is the full name of segment and each column corresponds to a level in the naming hierarchy. If there are  $m$  items and the longest name captures  $n$  levels in the namespace hierarchy, then there is an  $(m + 1) \times (n + 1)$  matrix, with shorter names containing null entries. Transmitting the schedule therefore reduces to transmitting this matrix. I provide an example in Figure 5.5. For ease of exposition, I use an adjacency

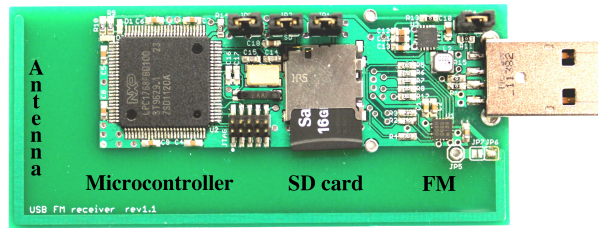


Figure 5.6: RDS-to-LAN bridge with integrated antenna. The SD card stores all broadcasts for long term access and retransmission.

matrix representation, but large discrepancies in the sizes of the names (the number of columns) could merit more compact, sparse matrix representations.

A primary goal with the scheduling format is to allow receivers to identify as early as possible whether they are interested in the upcoming transmissions, so that, if none are of interest, they may quickly go to sleep. To this end, the matrix of data names is delivered column-by-column. Consider the example in Figure 5.5; a receiver who is not interested in transmissions pertaining to noaa, cnn, or bitcoin has sufficient information upon receiving the first column to realize it should go to sleep, and to determine when to wake up to obtain the transmitter’s next schedule.

*Verification* The tower signs each package. This assists retransmission by allowing receivers to verify retransmissions they receive were originally sent by the tower. Due to the throughput constraint signatures need to be as small as possible. Therefore I used the 233-bit Koblitz curve, elliptic curve signatures.

### 5.2.3 RDS-to-LAN bridge

To realize my vision of a pervasive deployment of Abbie receivers in Internet-connected devices, I designed the RDS-to-LAN bridge pictured in Figure 5.6. The bridge connects to a USB port on a PC or OpenWRT home router. The bridge’s microcontroller runs the



full receiver software. It even verifies ECC signatures by running OpenECC [67], an ECC implementation for embedded systems, on its 24 MHz CPU. It also stores the packages on its on-board MicroSD card. The bridge also forwards every RDS group to the router so it can multicast them on the LAN to other receivers.

## 5.2.4 Receiver software

I implemented a receiver in C that runs on the RDS-to-LAN bridge embedded system, on a mobile phone, and on laptops. My design takes advantage of the continuous stream nature of broadcasts; a receiver can easily detect that it missed a transmission by observing a missing sequence number. Receivers without power constraints can participate in a receiver-driven retransmission system. They receive constantly, store packages locally, and can serve them to other receivers via a DHT. Receivers with power constraints listen only when there is a schedule or something interesting in the current schedule. The receiver can receive broadcasts from a local receiver (as is the case with an FM-enabled mobile phone) or over LAN multicast from an RDS-to-LAN bridge.

*Subscription* Applications ask the receiver to subscribe to Abbie segments. The receiver's subscription service is a basic TCP server that takes subscriptions as input and outputs segments to applications. The subscription message is simply the hierarchical name of the segment, such as `dns.google.www` and permits wildcards such as `dns.*.*`. Recall that each fragment of this name is hashed, and only a prefix of that hash sufficient to be unique, is transmitted by the tower.

The client application can specify whether it would like the last segment the receiver stored or the next segment it receives. The last segment is useful when, e.g., a laptop goes

to sleep and would like to determine whether it missed any broadcasts when it wakes.

The client application can specify if it wants the segments reliably and in-order. When a receiver providing a reliable subscription misses a package, the receiver eagerly searches in the retransmission DHT for another receiver that can provide the missing package.

The process of storing data keyed on a variable-length name hash prefix, then retrieving it by any matching prefix, is comparable to longest-prefix matching in IP routing. Although I currently store subscriptions and recently received data in simple arrays, one possible alternative is to develop a more scalable data structure consisting of a hierarchy of patricia tries [62]. Hash prefixes (as transmitted in the schedule) are represented by nodes in a top-level trie. Each of these nodes then references a trie for the next level in the hierarchy. At the top “default” prefix, is a list of all subscribers to messages matching the name so far and the recent messages for that complete name. With this structure, a wide variety of received names should be compactly stored and accessed.

*Retransmission* Although RDS receivers with a strong signal have extremely low loss rates, some applications may require reliability. Abbie has a DHT-based, receiver-driven retransmission system. The tower facilitates package retransmissions by broadcasting a signature for every package, which allows receivers to verify that the tower sent the package they retrieve from the DHT. The tower also inserts every package it broadcasts into the DHT, so it is guaranteed to be available even if no receiver is able to insert it into the DHT.

I modified Chroboczek’s Kademila [57] DHT from BitTorrent to lookup addresses of other receivers that can retransmit packages, instead of lists of peers downloading a

file. The name of a package is the 16 bit RDS station ID (PI code) appended with the package sequence number. For example, the retransmitters store and the tower signs the raw RDS groups that make up a package. Within these groups is the schedule header with the package's sequence number so a retransmission requester can not be fooled into thinking that any package signed by the transmitter is the package sequence number they are looking for.

Some always-on retransmitters with good signal strength are helpful to keep the DHT populated with every package ever transmitted by a tower. The always-on RDS-to-LAN bridges may serve this purpose. Though, there has been extensive work in developing DHTs that are resilient to high amounts of churn [36, 81].

When a client wants a retransmission, it looks up the relevant package in the DHT by its package name. The node that responds, serves the retransmission from its basic Abbie TCP server. The a client only stores a package if it can verify the signature.

## **5.3 Applications**

In this section I describe how Abbie can change the way we think about distributed applications. Broadcast can eliminate the need for adaptive load balancing with short DNS TTLs. Also, broadcast allows us to reset a large number of hosts without knowing about them.

### **5.3.1 Push: DNS cache updates**

Large content providers commonly perform load balancing across their servers by setting short TTLs in their authoritative DNS responses. For instance, Google and Akamai commonly use TTLs of just a few minutes. Such low TTLs provide some assurance that

if a provider changes its load balancing strategy—for instance in response to a surge in demand or a failure—then users will *pull* updates before long. On the other hand, with this strategy, a provider  $p$  must *always* set its TTLs to no greater than the amount of time  $p$  would be willing to allow its users to be out of sync. This can result in superfluous queries and increased network congestion.

When using DNS TTLs in this manner, the trade-off between responsiveness and communication overhead is unavoidable because *it is in essence a polling system*. Providers force users to effectively poll their servers in case there is a new load balancing strategy.

Here, I describe a system that I have implemented that uses Abbie to instead *push* DNS changes via broadcast. Receivers subscribe to segments from *dns.\**; to support power-efficient receiver-side filtering, the names are hierarchical, of the form *dns.google.www*. When the router receives such a segment, it checks whether the DNS name matches one of its client's subscriptions, and if so, receives and stores the corresponding segments.

Round-robin DNS load balancing allows a resolver to cycle through a list of IP addresses that map to a domain in order to spread the load to those various servers. To distributed load while having a single broadcast message, I instead broadcast all of the IP addresses, resulting in a segment of the form: [DNS name,  $IP_1, \dots, IP_n$ ], transmitted in binary format. Were receivers to simply use this list in a round-robin fashion, there may be spikes of usage at  $IP_1$ ; after all, the broadcast synchronizes this message across all receivers in a geographic region. Ideally, each receiver would get its own random sample of this list, but broadcast messages are equivocation-resilient. While both of these properties—synchronized and equivocation-resilient communication—are useful in other applications, they present a minor challenge here. Instead, receivers are expected to per-

mute the list before using it.

In my implementation, the router permutes the list of IP addresses, adds it to `/etc/hosts`, and sends `-HUP` to `dnsmasq`, causing it to locally update to this new state. My router sets the TTL to zero, keeping most<sup>1</sup> clients from caching the responses, and to instead go to its router each time it wishes to access a given domain, ensuring the most up-to-date entries. Note that this local polling is precisely what large providers try to do today but *in the wide area*; in my setting, if there is any redundant traffic, it is isolated to the user's home network.

### **5.3.2 Anonymous and synchronous: mass reset**

Reconfiguring or, in the extreme case, rebooting nodes in a large distributed system can be challenging. Traditional approaches complicate either the task of the administrator—operators often must maintain updated contact information like IP addresses or phone numbers of all of the devices they manage—or the protocol designer—reconfiguring routing in a network can cause cascading consistency issues.

The inherent properties of broadcast make it nicely suited for supporting large scale reconfiguration. Broadcasts are receiver anonymous, the administrator only needs to know the frequency that the receivers will be listening on, not the identifiers of the machines themselves. Further, Broadcast also is an inherent source of synchronization. Resetting a large number of devices in a metropolitan area can occur within milliseconds. Synchronized reset can also be achieved via multi-unicast with clock synchronization, but in a sense, broadcast simultaneously provides distribution and synchronization.

---

<sup>1</sup>I have observed that some browsers interpret TTL=0 differently; Firefox, for instance, caches for several minutes, while Safari does not appear to cache at all.

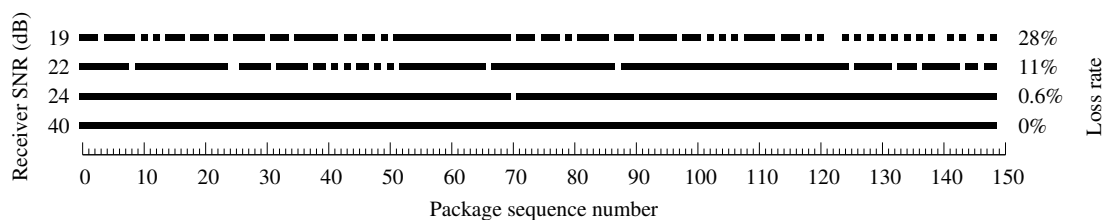


Figure 5.7: Broadcasting 148 packages (x-axis), each 160 bytes, from a 3 kW tower to four Abbie RDS-to-LAN bridges (y-axis). A black box indicates the package’s signature verified, a white box indicates either the signature did not verify or there were missing RDS groups. RDS error correction is disabled. The high SNR receivers capture > 99% of the packages. The low SNR receivers miss many, but not all, and they only miss three of the same packages.

This example, along with the DNS example in this section, demonstrates that Abbie can facilitate protocol design and administration of large-scale distributed systems.

## 5.4 Evaluation

In this section, I describe my “end-to-end” experiments with a metropolitan area deployment of four Abbie USB receivers listening to a 3kW tower. The Abbie transmitter interfaces with the station’s Audemat FMB80 RDS encoder via the UECP protocol. The goals of the experiments are to determine whether messages transit the gateways in identical time to permit synchronized delivery, and to provide microbenchmarks of the FM receiver IC’s startup time, which affects the ability to sleep to conserve power.

### 5.4.1 Metropolitan deployment

I evaluated Abbie end-to-end across a metropolitan area with a 3 kW commercial FM radio station located in the center of Ann Arbor, Michigan, and four RDS-to-LAN bridges.

The station agreed to let me send Abbie’s RDS messages 50% of the time with their artist and track title RDS messages occupying the other 50%. Because Abbie transmits with a different RDS message identifier than the station, it should not interfere, and I am

Location	SNR	Distance from tower
Apartment building	40	0.85 km
Apartment building	24	5.5 km
University building interior	22	2.7 km
Apartment building	19	8.0 km

Table 5.1: Abbie prototype receiver locations in the testbed deployment

not aware of any complaints of radio problems from listeners of the station.

In my end-to-end experiment, I transmitted 148 Abbie packages each containing one 100 byte segment simultaneously to the four receivers and observed if the signatures verified successfully. Figure 5.7 shows the results. Two receivers are in a high signal strength area, while two are in a low signal strength area, as shown in Table 5.1. Consistent with the mobile experiment shown in Figure 5.1, the high signal strength receivers received over 99% of the packages. The 24 SNR receiver operates on the edge of this high reception probability region, and it only had errors in one package.

The low SNR receivers lost several packages, but only three of the same packages, confirming my hypothesis that losses occur independently at receivers. (Those caused by line-of-sight lightning may be correlated; the weather was calm during this experiment.) That losses are infrequent for high signal strength is consistent with my low-level RDS bit error experiment, from Section 5.1.1, that indicated errors are not evenly spaced throughout the transmission: Even distribution of errors would cause the receivers to lose many, if not all, of the packages. This is an encouraging result. It indicates that receivers in high signal strength areas may have a very low loss rate, and that all receivers in low signal strength areas do not necessarily lose the same packages.

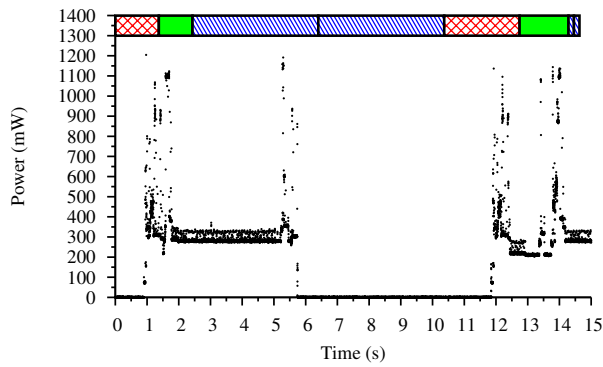


Figure 5.8: The energy consumption of the RDS-equipped Samsung Galaxy S while skipping two packages because their schedule does not match then finally receiving the short segments at the end. The boxes at the top show the broadcast. Green is the schedule, blue are the segments and red is signature.

## 5.4.2 Abbie’s energy consumption on an Android phone

A small number of smartphones already include FM RDS receivers, so I can evaluate the energy consumption of receiving a desired segment on a device where energy is a primary concern. To test the energy consumption of Abbie, I ran the receiver software on a Samsung Galaxy S Android phone. This phone has a built-in si4709 FM receiver IC.

Figure 5.8 shows the power of the phone as it receives a package header (1<sup>st</sup> green) and determines that the desired segment is not coming in this package. So it skips over the rest of the package’s segments (1<sup>st</sup> and 2<sup>nd</sup> blue) and signature (2<sup>nd</sup> red). Then, the smartphone wakes up to receive the next schedule (2<sup>nd</sup> green). It sees this schedule contains two desired segments that immediately follow the schedule (2<sup>nd</sup> and 3<sup>rd</sup> blue) so it stays awake to receive them.

The phone stays awake five seconds longer than it should during the first segment (1<sup>st</sup> blue) because the Android operating system does not suspend as aggressively as possible. The receiver software holds a wake lock when it wants to receive a part of a package and releases it when it is done. I measured the power consumption of the phone when it sees



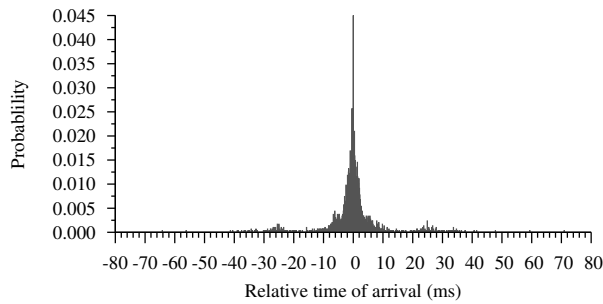


Figure 5.9: Measurements of the synchronization of two RDS-to-LAN bridges.

a schedule with two long segments it does not want and one with segments that it does. It does skip over most of the two segments that it does not care about.

### 5.4.3 Abbie end-to-end synchronization

One of the advantages of a broadcast system is the inherent synchronization of receivers. Applications can assume that within a certain period of time when receiver delivers a message, all other receivers do so as well. However in Abbie there are several factors that decrease the synchronization of the receivers. These factors include: the synchronizer in the FM RDS receiver IC, the verification of digital signatures, and the USB bridge to the residential routers.

I evaluate the end-to-end synchronization of two Abbie RDS-to-LAN bridges. To do so, I measure the relative time of arrival of an Abbie package received simultaneously by RDS-to-LAN bridges. The bridges send an Ethernet packet when the package signature verifies to a laptop running tcpdump (Figure 5.9). For most of the transmissions the packages arrive within 5 ms of each other. Li et al. [52] observe that there is a small but unpredictable time between receiving an RDS group and the receiver IC raising an interrupt.

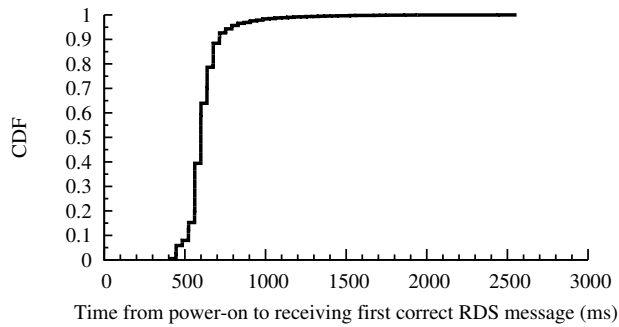


Figure 5.10: CDF of time from powered off to receiving first RDS message for the si4705 IC.

#### 5.4.4 RDS receiver cold boot

RDS receiver ICs are not designed for rapid startup. To determine the feasibility of cycling the RDS receiver IC on and off based on schedules, I must measure the time it takes for the IC I used on the RDS-to-FM bridge to startup and receive the first RDS group. Figure 5.10 shows the results of this study. Surprisingly, the median startup time is 599 ms. The longer startup times may be due to failures of the automatic tuning mechanism or the RDS synchronization.

### 5.5 Summary

In this chapter, I described adapting FM RDS to become a reliable broadcast Internet last-mile link by *building on FM RDS's transmission media and equipment*. I drove around a metropolitan area to evaluate the inherent reliability of FM RDS: even with a low power 3 kW transmitter, there were no losses within a 5 km radius from the transmitter. I combined FCC service contours and US census estimates to determine the coverage of deployed FM transmitters in the US: most of the 10,075 FM stations cover at least 100,000 people and 13% cover over 1 million people. I described how receivers can be small devices, and how they can store all transmissions they receive for a virtually

unlimited time which enables receivers to provide their own retransmission system.

I built a system on top of FM RDS to adapt it to be an Internet last-mile link. I also described DNS and mass reset applications that would benefit from this reliable service with the parameters.

I evaluated the full end-to-end system on a 3 kW commercial radio station and demonstrated the reliability of FM RDS transmission when signal strength is high: even when it is low, failures are not across all receivers. I implemented receiver software, ran it on a smartphone with a built-in FM RDS receiver, and measured the phone's power consumption. These results demonstrated that the smartphone can sleep between transmissions that it does not subscribe to. I sent broadcasts to two receivers to demonstrate that two independent receivers running the end-to-end system mostly deliver a message within 5 ms of each other. Although adapting FM RDS to be an Internet last-mile link required creating software and some hardware, *I did not modify FM RDS's transmitters and receivers.*

# Chapter 6

## Conclusions and Open Questions

In this dissertation, I observed and improved the reliability of Internet last-mile links by building off of unmodified transmission media and equipment, and collecting data only from public measurement infrastructure. These restrictions allowed me to make conclusions about general Internet last-mile link reliability, to improve the reliability of existing last-mile links, and to build a reliable Internet last-mile downlink. I defended the following thesis: *By building on existing infrastructure, it is possible to (1) observe the reliability of Internet last-mile links across different weather conditions and link types; (2) improve the energy efficiency of cellular Internet last-mile links; and (3) provide an incrementally deployable, energy-efficient Internet last-mile downlink that is highly resilient to weather-related failures.*

***Without privileged data from ISPs, I observed the how weather affects the reliability of last-mile links .*** In Chapter 3, I correlated weather with observations of last-mile link failures. I observed link failures *without privileged data* by pinging last-mile links before, during, and after forecasted weather. Then I showed how to sift through the noisy

pings to find probable last-mile link failures. To accomplish this, I introduced a history-based probabilistic method that can find intervals when a last-mile link is likely failed. Also, I used an edge-detection algorithm to detect changes in persistent loss rates. Together, these two tools provided an accurate external observation of time and duration of failures.

Using my observation and analysis tools, I observed failures induced by wind, rain, thunderstorms and high temperatures, *even when I excluded suspected power outages*. My observations characterize small timescale failures (on the order of days); I believe my techniques could be used to classify longer timescale failures as well.

***Without modifying cellular hardware, firmware, or drivers, or accessing proprietary cellular data, I improved the reliability of cellular communication by reducing wasted energy.*** In the background Chapter 2, I showed how signal strength has a direct impact on cellular radio energy consumption, which by far dominates the base energy consumption of mobile devices. Then in Chapter 4, I showed that the variations in cellular signal strength as a user drives around—when coupled with the presence of flexible applications, such as email syncing, photo sharing, and on-demand streaming—presents a significant opportunity to save energy. I designed Bartendr, a practical framework for scheduling application communication to be aligned with periods of good signal. Bartendr addresses a number of challenges and makes novel contributions, including: track-based, energy-efficient prediction of signal strength and a dynamic programming-based procedure for computing the optimal communication schedule. My simulations of drives demonstrated significant energy savings of up to 10% for email sync and up to 60% for

on-demand streaming.

*Without modifying a widely-deployed broadcast link, I adapted it to provide the first reliable broadcast Internet last-mile link.* In Chapter 5, I described Abbie, a reliable data broadcast system designed to simultaneously support diverse applications on inexpensive hardware, supporting both Internet-connected and energy-limited devices. Abbie uses FM RDS because of its extremely low loss rate and extensive coverage. These properties allows Abbie to operate when other Internet last-mile links cannot. I measured these properties using FCC and Census data. Using FM RDS as the underlying physical link introduced several technical challenges such as designing a reliable protocol that could operate at such a low bitrate, and interfacing with existing RDS transmitters and receivers. I deployed Abbie and measured its energy consumption and delivery probability using: a testbed consisting of custom-designed boards, wireless access points, a real commercial tower, evaluation boards, and mobile phones.

The primary finding underlying all of this work is that the reliability of Internet last-mile links can be observed and improved across many different link types and sources of failures without modifying transmission media and equipment or accessing privileged data.

## **6.1 Longevity**

The findings in this dissertation will have longevity for the following reasons:

So long as transmission media and equipment are exposed to the elements, weather

will continue to be a fundamental source of failures for last-mile links. Moreover, re-deploying exposed transmission media and equipment is an endeavor so costly and expansive that it is unlikely to ever happen. As such, I believe there will always be a need to observe Internet last-mile link failures in the presence of weather, which ThunderPing does at scale. Finally, my techniques for identifying last-mile link failures from pings are not restricted to weather, and can be applied to other sources of failure.

So long as smartphones have limited battery capacity, and the radio consumes a significant portion of this capacity, energy-efficient communication will continue to be an important factor in improving reliability. Radios are more energy-efficient when they have high signal strength, and thus determining when best to send delay-tolerant traffic benefits from being able to predict future signal strengths. Bartendr requires little information to predict future signal strength (cell IDs and current signal strength), and is therefore likely to be applicable even as mobile devices and cellular technologies evolve. Recent work advocates having applications directly inform the radio of their deadlines and priorities [39]. Even then, there will continue to be a need to predict signal strength to determine the most energy efficient time to send.

Although FM RDS broadcasting may not last forever, the part of the spectrum it uses, VHF, is minimally affected by weather (except for line-of-sight lightning), penetrates walls, and permits compact receiver antennas. As such, VHF is fundamentally one of the most reliable broadcast transmission media. Abbie demonstrates how to build a broadcast Internet last-mile link that benefits from the VHF spectrum's reliability. The contributions of this thesis suggest rethinking the use of the VHF spectrum as a reliable last-mile link for arbitrary data from the Internet (though, hopefully with a much higher bitrate than

FM RDS). Abbie's broadcast scheduling and hierarchical naming, which allow receivers to ignore transmissions they are not interested in, are not bound to VHF in particular, and could be applied to any broadcast transmission medium.

## 6.2 Open questions

I now close my dissertation with a few significant open questions that my work raises:

Although with ThunderPing, I can observe weather-related last-mile link failures, an open question is: can we predict weather-related failures? If so, how would we use the predictions? An extremely accurate predictor of weather coupled with a precise understanding of how weather affects any given last-mile link, would make this problem straightforward. However, weather forecasts themselves are predictions, and ThunderPing only observes the behavior of a sample of last-mile links. It remains to be seen whether the error in the forecasts and sampling precludes predicting failures for any given last-mile link. But perhaps even a moderately accurate prediction, could improve the reliability of some applications. Identifying which applications could benefit from predicting weather and how they should react to weather forecasts is another potential area to investigate.

Bartendr periodically predicts signal strength in order to find the most energy efficient periods to communicate. However, because computing these predictions requires the smartphone to enter and leave a fully active state quickly, I had to trade off between prediction accuracy and the energy consumed in making the prediction. This raises the following question: Would it be possible to reduce the energy consumed making the prediction by entering and leaving a *partially* active state? Is it possible to run only one program and only its required peripherals, and would this result in energy savings?



Perhaps smartphone operating systems can only suspend and wakeup a few tasks and peripherals that they are needed for the prediction.

With Abbie, I demonstrated that FM RDS can be adapted to be an Internet last-mile link, but an important open question is: what assumptions about FM RDS would not carry over to a broadcast system with a much higher bitrate. FM RDS's bitrate is sufficient to provide for interesting applications such as a mass reset, but a higher bitrate would allow for more senders, and more data-intensive applications. How would the assumptions about reliability change? How would the receiver energy consumption change? How would the coverage change? How would the cost of receiver ICs change? Could low power microcontroller devices keep up with the pace of broadcasts?

# Bibliography

- [1] 3GPP. Technical realization of Cell Broadcast Service (CBS), 2011.
- [2] A. Adelsbach, U. Greveler, and S. Löschner. Anonymous data broadcasting by misuse of satellite ISPs. In *Proceedings of the Chaos Communication Congress (CCC)*, 2005.
- [3] D. Aguayo, J. Bicket, S. Biswas, D. S. J. D. Couto, and R. Morris. MIT Roofnet implementation, 2003.
- [4] Alpha Technologies. Installation and technical manual DOCSYS HMS embedded transponder, 2004.
- [5] G. Ananthanarayanan, M. Haridasan, I. Mohomed, D. Terry, and C. A. Thekkath. Star-Track: A framework for enabling track-based applications. In *Proceedings of the ACM Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2009.
- [6] P. Bahl, R. Chandra, T. Moscibroda, R. Murty, and M. Welsh. White space networking with Wi-Fi like connectivity. In *Proceedings of the SIGCOMM Conference on Data Communication*, 2009.
- [7] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile smartphones: A measurement study and implications for network applications. In *Proceedings of the Internet Measurement Conference (IMC)*, 2009.
- [8] H. Bölcskei, A. J. Paulraj, K. V. S. Hari, R. U. Nabar, and W. W. Lu. Fixed broadband wireless access: State of the art, challenges, and future directions. *IEEE Communications Magazine*, 2001.
- [9] British Broadcasting Corporation, Independent Broadcasting Authority, and British Radio Equipment Manufacturers Association. *Broadcast Teletext Specification*, 1978.
- [10] J. Canny. A computational approach to edge detection. In *IEEE Pattern Analysis and Machine Intelligence*, 1986.
- [11] J.-M. Chang and N. F. Maxemchuk. Reliable broadcast protocols. *ACM Transactions on Computer Systems (TOCS)*, 1984.
- [12] B. Charny. Metricom files for bankruptcy protection. *CNET*, 2001.

- [13] W. Ciciora, G. Sgrignoli, and W. Thomas. An introduction to Teletext and Viewdata with comments on compatibility. *IEEE Transactions on Consumer Electronics*, 1979.
- [14] Cisco. GS7000 DOCSYS status monitor transponder installation and operation guide, 2011.
- [15] Cisco Systems. Comcast successfully deploys Cisco Network Registrar for DHCP and DNS services for IPv4 and IPv6. <http://blogs.cisco.com/sp/comcast-successfully-deploys-cisco-network-registrar-for-dhcp-and-dns-services-for-ipv4-and-ipv6/>.
- [16] ClearChannel. Leading radio broadcasters unite to bring FM song tagging to everyone, 2008.
- [17] I. Constandache, R. R. Choudhury, and I. Rhee. Towards mobile phone localization without war-driving. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2010.
- [18] I. Constandache, S. Gaonkar, M. Sayler, R. R. Choudhury, and L. Cox. EnLoc: Energy-efficient localization for mobile phones. In *IEEE INFOCOM Mini-conference*, 2009.
- [19] S. C. Cripps. *RF Power Amplifiers for Wireless Communications*. Artech House, 2006.
- [20] CTIA - The Wireless Association. Semi-annual wireless industry survey, 2011.
- [21] CTIA-The Wireless Association. Wireless emergency alerts on your mobile device. <http://www.ctia.org/wea>.
- [22] J. Davis. Intercast dying of neglect. *CNET*, 1997.
- [23] Department of Telecommunications. National frequency allocation table, 2011. Draft.
- [24] N. Ding, D. Wagner, X. Chen, Y. C. Hu, and A. Rice. Characterizing and modeling the impact of wireless signal strength on smartphone battery drain. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2013.
- [25] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu. Characterizing residential broadband networks. In *Proceedings of the Internet Measurement Conference (IMC)*, 2007.
- [26] R. Droms. RFC 2131: Dynamic Host Configuration Protocol, 1997.
- [27] dslreports.com. ISP Rolodex. <http://www.dslreports.com/isplist?c=us>, 2011.
- [28] Electronic Communications Committee. The European Table of Frequency Allocations and Applications in the Frequency Range 8.3 kHz to 3000 GHz, 2013.
- [29] European Broadcasting Union / RDS Forum. RDS Universal Encoder Communication Protocol, 1997.

- [30] FCC transmitter contour database. [http://transition.fcc.gov/ftp/Bureaus/MB/Databases/fm\\_service\\_contour\\_data](http://transition.fcc.gov/ftp/Bureaus/MB/Databases/fm_service_contour_data).
- [31] Federal Communications Commission. Emergency Alert System 2007 TV (including digital TV) handbook.
- [32] Federal Communications Commission. FCC rules and regulations for VoIP 911, 2005.
- [33] Federal Communications Commission Office of Engineering and Technology Policy and Rules Division. FCC Online table of frequency allocations, 2013.
- [34] A. Ghodsi, T. Koponen, J. Rajahalme, P. Sarolahti, and S. Shenker. Naming in content-oriented architectures. In *Proceedings of the ACM SIGCOMM Workshop on Information Centric Networking (ICN)*, 2011.
- [35] R. Gibson. Elections online: Assessing Internet voting in light of the Arizona Democratic Party. *Political Science Quarterly*, 2001.
- [36] P. B. Godfrey, S. Shenker, and I. Stoica. Minimizing churn in distributed systems. In *Proceedings of the SIGCOMM Conference on Data Communication*, 2006.
- [37] F. Hartung, U. Horn, J. Huschke, M. Kampmann, T. Lohmar, and M. Lundevall. Delivery of broadcast services in 3G networks. *IEEE Transactions on Broadcasting*, 2007.
- [38] J. Heidemann, Y. Pradkin, R. Govindan, C. Papadopoulos, G. Bartlett, and J. Bannister. Census and survey of the visible Internet. In *Proceedings of the Internet Measurement Conference (IMC)*, 2008.
- [39] B. D. Higgins, J. Flinn, T. Giuli, B. Noble, C. Peplin, and D. Watson. Informed mobile prefetching. In *Proceedings of the ACM Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2012.
- [40] D. Hogg and T.-S. Chu. The role of rain in satellite communications. *Proceedings of the IEEE*, 1975.
- [41] Interactive Advertising Bureau (IAB). Internet advertising revenue report - 2012 full year results, 2013.
- [42] International Electrotechnical Commission (IEC). IEC 62106, 2009.
- [43] International Game Developers Association. 2008 - 2009 Casual Games White Paper.
- [44] F. B. Jewett. The modern telephone cable. In *Proceedings of 26th annual convention of the American Institute of Electrical Engineers*, 1909.
- [45] Y. Jin, N. Duffield, A. Gerber, P. Haffner, S. Sen, and Z.-L. Zhang. NEVERMIND, the problem is already fixed: Proactively detecting and troubleshooting customer DSL problems. In *Proceedings of the International Conference on emerging Networking Experiments and Technologies (CoNEXT)*, 2010.

- [46] D. Kopitz and B. Marks. *RDS: The Radio Data System*, 1999.
- [47] J. Krikke. Sunrise for energy harvesting products. *IEEE Pervasive Computing*, 2005.
- [48] J. Krumm and E. Horvitz. Predestination: Where do you want to go today? *IEEE Computer Magazine*, 2007.
- [49] H. Kushner and P. Whiting. Convergence of proportional-fair sharing algorithms under general conditions. *IEEE Transactions on Wireless Communications*, 2004.
- [50] A. LaMarca et al. Place Lab: Device positioning using radio beacons in the wild. In *IEEE Pervasive Computing*, 2005.
- [51] T. Lemon. ISC's DHCP distribution. In *Proceedings of the USENIX Annual Technical Conference Freenix Track*, 1998.
- [52] L. Li, G. Xing, L. Sun, W. Huangfu, R. Zhou, and H. Zhu. Exploiting FM Radio Data System for adaptive clock calibration in sensor networks. In *Proceedings of the ACM Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2011.
- [53] J. Liu, B. Priyantha, T. Hart, H. S. Ramos, A. A. F. Loureiro, and Q. Wang. Energy efficient GPS sensing with cloud offloading. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2012.
- [54] X. Liu, A. Sridharan, S. Machiraju, M. Seshadri, and H. Zang. Experiences in a 3G network: Interplay between the wireless channel and applications. In *Proceedings of the ACM Conference on Mobile Computing and Networking (MobiCom)*, 2008.
- [55] A. Lyner. Experimental Radio Data System (RDS): A survey of reception reliability in the UK, 1987.
- [56] MaxMind, Inc. GeoIP city database. <http://www.maxmind.com/app/city>, 2011.
- [57] P. Maymounkov and D. Mazieres. Kademia: A Peer-to-peer information system based on the XOR Metric. In *Proceedings of the Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.
- [58] G. N. Mendenhall and R. J. Fry. FM broadcast transmitters. *NAB Engineering Handbook, 9th edition*, 1999.
- [59] Microsoft Corp. MSN Direct. <http://www.msndirect.com/>.
- [60] Microsoft Corp. MSN Direct services overview. <http://msdn.microsoft.com/en-us/library/cc510527.aspx>.
- [61] J. Mohen and J. Glidden. The case for Internet voting. *Communications of the ACM (CACM)*, 2001.
- [62] D. R. Morrison. PATRICIA—practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM*, 15(4):514–534, 1968.

- [63] National Association of Broadcasters. Radio rocks my phone. <http://www.radiorocksmyphone.com>.
- [64] National Weather Service. NOAA weather radio all hazards. <http://www.nws.noaa.gov/nwr/>.
- [65] National Weather Service. NWS public alerts in XML/CAP v1.1 and ATOM formats. <http://alerts.weather.gov/>.
- [66] A. J. Nicholson and B. D. Noble. BreadCrumbs: Forecasting mobile connectivity. In *Proceedings of the ACM Conference on Mobile Computing and Networking (MobiCom)*, 2008.
- [67] OpenECC. An open source library for elliptic curve cryptosystem. <http://openecc.org/>.
- [68] J. D. Parsons. *The Mobile Radio Propagation Channel*. John Wiley & Sons Ltd, 2000.
- [69] C. Peng, S.-B. Lee, S. Lu, H. Luo, and H. Li. Traffic-driven power saving in operational 3G cellular networks. In *Proceedings of the ACM Conference on Mobile Computing and Networking (MobiCom)*, 2011.
- [70] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the Internet. In *Proceedings of the Workshop on Hot Topics in Networks (HotNets)*, 2002.
- [71] I. Poese, S. Uhlig, M. A. Kaafa, B. Donnet, and B. Gueye. IP geolocation databases: Unreliable? In *Computer Communication Review (CCR)*, 2011.
- [72] G. P. Pollini. Trends in handover design. *IEEE Communications Magazine*, 1996.
- [73] J. Postel. RFC 792: Internet Control Message Protocol, 1981.
- [74] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. TOP: Tail optimization protocol for cellular radio resource allocation. In *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*, 2010.
- [75] L. Quan, J. Heidemann, and Y. Pradkin. Trinocular: Understanding internet reliability through adaptive probing. In *Proceedings of the SIGCOMM Conference on Data Communication*, 2013.
- [76] RadioDNS. <http://www.radiodns.org>.
- [77] A. Rahmati and L. Zhong. Context-for-wireless: Context-sensitive energy-efficient wireless data transfer. In *Proceedings of the ACM Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2007.
- [78] A. Rahmati, L. Zhong, V. Vasudevan, J. Wickramasuriya, and D. Stewart. Enabling pervasive mobile applications with the FM radio broadcast data system. In *Proceedings of the Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2010.

- [79] B. Ray. Microsoft ditches MSN Direct. *The Register*, 2009.
- [80] RDS Forum. *RDS Technical Specification*, 2009.
- [81] S. Rhea, D. Geels, T. Roscoe, and J. Kubiawicz. Handling churn in a DHT. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2004.
- [82] RIPE NCC. RIPE Atlas. <http://atlas.ripe.net>, 2012.
- [83] SamKnows. Accurate broadband performance information for consumers, governments and ISPs. <http://www.samknows.com>, 2013.
- [84] A. Schulman and N. Spring. Pingin' in the rain. In *Proceedings of the Internet Measurement Conference (IMC)*, 2011.
- [85] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V. N. Padmanabhan. Bartendr: A practical approach to energy-aware cellular data scheduling. In *Proceedings of the ACM Conference on Mobile Computing and Networking (MobiCom)*, 2010.
- [86] A. Sharma, V. Navda, R. Ramjee, V. Padmanabhan, and E. Belding. Cool-Tether: Energy efficient on-the-fly WiFi hot-spots using mobile phones. In *Proceedings of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2009.
- [87] Y. Shavitt and E. Shir. DIMES: Let the Internet measure itself. *Computer Communication Review (CCR)*, 2005.
- [88] M. Z. Shaq, L. Ji, A. X. Liu, J. Pang, S. Venkataraman, and J. Wang. A first look at cellular network performance during crowded events. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2013.
- [89] L. Shi, C. Liu, and B. Liu. Network utility maximization for triple-play services. *Computer Communications*, 2008.
- [90] W. T. Smith and W. L. Roberts. Design and characteristics of coaxial cables for Community Antenna Television. *IEEE Transactions on Communication Technology*, 1966.
- [91] N. Spring, L. Peterson, A. Bavier, and V. Pai. Using PlanetLab for network research: Myths, realities, and best practices. *ACM SIGOPS Operating Systems Review*, 2006.
- [92] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè. Broadband Internet performance: A view from the gateway. In *Proceedings of the SIGCOMM Conference on Data Communication*, 2011.
- [93] P. Svensson. Verizon winds down expensive FiOS expansion. *Associated Press*, 2010.
- [94] W. L. Tan, F. Lam, and W. C. Lau. An empirical study on 3G network capacity and performance. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2007.

- [95] N. D. Tripathi, J. H. Reed, and H. F. VanLandingham. Handoff in cellular systems. *IEEE Personal Communications*, 1998.
- [96] United States Office of Personnel Management. Guide to telework in the federal government, 2011.
- [97] US Census data. <http://www.census.gov/popest/data/datasets.html>.
- [98] Y. Wang, D. Burgener, M. Flores, A. Kuzmanovic, and C. Huang. Towards street-level client-independent IP geolocation. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.
- [99] B. Wong, I. Stoyanov, and E. G. Sirer. Octant: A comprehensive framework for the geolocalization of Internet hosts. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)*, 2007.
- [100] Y. Xie, F. Yu, K. Achan, E. Gillum, M. Goldszmidt, and T. Wobber. How dynamic are IP addresses? In *Proceedings of the SIGCOMM Conference on Data Communication*, 2007.
- [101] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha. Appscope: Application energy metering framework for android smartphones using kernel activity monitoring. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2012.
- [102] H. Yu, A. Rahmati, A. A. Sani, L. Zhong, J. Wickramasuriya, and V. Vasudevan. Data broadcasting using mobile FM radio: Design, realization and application. In *Proceedings of the International Conference on Ubiquitous Computing (Ubicomp)*, 2011.
- [103] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (CODES+ISS)*, 2010.