UNIVERSITY OF CALIFORNIA, SAN DIEGO

Downclocking WiFi to Improve Energy Efficiency in Mobile Devices

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

in

Electrical Engineering (Computer Engineering)

by

Feng Lu

Committee in charge:

Professor Tara Javidi, Co-Chair
Professor Alex C. Snoeren, Co-Chair
Professor Geoffrey M. Voelker, Co-Chair
Professor Ramesh Rao
Professor Stefan Savage

2014

DEDICATION

To my wife who is always there for me.

EPIGRAPH

The best way to predict the future is to invent it.

Alan Kay

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am deeply indebted to my advisors Professor Alex C. Snoeren and Geoffrey M. Voelker. I would like to thank them for taking a chance on me, and letting me sit in the driver seat while still providing sufficient guidance to steer my research towards the right direction. Their generous help, encouragements, understanding, and patience have constantly been a great source of motivation during my graduate study. I benefit tremendously from their feedback during our fruitful weekly meetings, and am always amazed on how fast they could understand my material and the fresh perspectives they provide. I further thank them for the many editing rounds on our co-authored papers before submission deadlines, no matter how inconvenient the time had been for them. Outside work, they are also extremely supportive and helpful for many of my personal decisions.

I would also like to thank Professor Tara Javidi, Ramesh Rao and Stefan Savage for being part of my committee and their invaluable feedback on this dissertation. I also want to acknowledge Dr. Curt Schurgers for his guidance in my first two years of graduate school and Professor George Porter for sharing his knowledge on networking.

The journey of graduate study would have been much less enjoyable without the companionship from friends and fellow graduate students. I have some awesome office mates, who have made my day-to-day grad life refreshing. And here they are: Alvin AuYoung, Ryan Braud, Peng Huang, Yang Liu, Damon McCoy, Andreas Pitsillidis, Bhanu Vattikonda, Michael Vrable, and David Wang. Many people in UCSD have helped me in different ways: Yuvraj Agarwal, Michael Conley, Danny Huang, Do-kyum Kim, Terry Lam, Xiao Ma, John McCullough, Sivasankar Radhakrishnan, Alex Rasmussen, Daniel Turner, Patrick Verkaik, Dawei Wang, Kevin Webb, Yu Xiang, Tianyin Xu, Ding Yuan and Qing Zhang. Yichao Huang and Wei Guan deserve special mention for being such great friends, and I am always grateful for their help offered in numerous occasions.

I had great fun working with Rishi Kapoor, Patrick Ling, He Liu, Diba Mirza, Malveeka Tewari and Jiaqi Zhang on our co-authored papers. I want to extend my special thanks to Hao Du and Dr. Andreas Terzis for their help, guidance and warmness during my internship in Google. Also greatly appreciate the prompt assistance from our sysnet staff Brian Kantor, Cindy Moore and Jennifer Folkestad.

I am particularly thankful to my parents for their love, care and principles instilled on me. Finally, this dissertation would not be possible without the selfless support from my wife, Lijuan, it is to her this dissertation is dedicated.

Chapter 2, 3, and 5, in part, are a reprint of the material as it appears in Proceedings of the USENIX Symposium on Networked Systems Design and Implementation 2013. Lu, Feng; Voelker, Geoffrey M.; Snoeren, Alex C. The dissertation author was the primary investigator and author of this paper.

Chapter 2, 4, and 5, in part, are a reprint of the material as it appears in Proceedings of the ACM Annual International Conference on Mobile Computing and Networking 2014. Lu, Feng; Ling, Patrick; Voelker, Geoffrey M.; Snoeren, Alex C. The dissertation author was the primary investigator and author of this paper.

VITA

2006	Bachelor of Engineering, Computer Engineering Nanyang Technological University, Singapore
2010	Master of Science Electrical Engineering (Computer Engineering) University of California, San Diego
2014	Doctor of Philosophy Electrical Engineering (Computer Engineering) University of California, San Diego

PUBLICATIONS

Feng Lu, Patrick Ling, Geoffrey M. Voelker, Alex C. Snoeren. “Enfold: Downclocking OFDM in WiFi”. *Proceedings of the* 20th *ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*, Maui, HI, Sep 2014.

Feng Lu, Geoffrey M. Voelker, Alex C. Snoeren. “Managing Contention with Medley”. *To Appear in the IEEE Transactions on Mobile Computing*.

He Liu, Feng Lu, Alex Forencich, Rishi Kapoor, Malveeka Tewari, Geoffrey M. Voelker, George Papen, Alex C. Snoeren, George Porter. “Circuit Switching Under the Radar with REACToR”. *Proceedings of the* 11th *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Seattle, WA, Apr 2014.

Haifeng Lu, Feng Lu, Chuan Heng Foh, Jianfei Cai. “LT-W: Improving LT Decoding with Wiedemann Solver”, *IEEE Transactions on Information Theory*, Volume 59, Issue 12, Pages 7887-7897, Dec 2013.

Feng Lu, Geoffrey M. Voelker, Alex C. Snoeren. “SloMo: Downclocking WiFi Communication”. *Proceedings of the* 10th *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Lombard, IL, Apr 2013.

Feng Lu, Jiaqi Zhang, Stefan Savage. “When Good Services Go Wild: Reassembling Web Services for Unintended Purposes”. *Proceedings of the* 7th *USENIX Workshop on Hot Topics in Security (HotSec)*, Bellevue, WA, Aug 2012.

Feng Lu, Geoffrey M. Voelker, Alex C. Snoeren. “Weighted Fair Queuing with Differential Dropping”. *Proceedings of the* 31st *Annual IEEE International Conference on Computer Communications (INFOCOM) Mini-conference*, Orlando, FL, Apr 2012.

Feng Lu, Diba Mirza, Curt Schurgers. “D-Sync: Doppler-Based Time Synchronization for Mobile Underwater Sensor Networks”. *Proceedings of the* 5th *ACM International Workshop on Underwater Networks (WUWNET)*, Woods Hole, MA, Oct 2010.

Feng Lu, Sammy Lee, Jeffrey Mounzer, Curt Schurgers. “Low-cost Medium-Range Optical Underwater Modem”. *Proceedings of the* 4th *ACM International Workshop on Underwater Networks (WUWNET)*, Berkeley, CA, Nov 2009.

Feng Lu, Chuan Heng Foh, Jianfei Cai, Liang-Tien Chia. “LT Codes Decoding: Design and Analysis”. *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, Seoul, Korea, Jun 2009.

Lijuan Geng, Feng Lu, Ying-Chang Liang, Francois Chin. “Secure Multi-path Construction in Wireless Sensor Networks using Network Coding”. *Proceedings of the* 19th *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Cannes, France, Sep 2008.

Feng Lu, Liang-Tien Chia, Kok Leong Tay, Wai Hoe Chong. “NBgossip: an Energy-efficient Gossip Algorithm for Wireless Sensor Networks”. *Springer Journal of Computer Science and Technology*, Volume 23, Issue 3, Pages 426-437, May 2008.

Feng Lu, Lijuan Geng, Liang-Tien Chia, Ying-Chang Liang. “Poster: Secure Multi-path in Sensor Networks”. *Proceedings of the* 5th *ACM Conference on Embedded Network Sensor Systems (Sensys)*, Sydney, Australia, Nov 2007.

Feng Lu, Liang-Tien Chia, Kok Leong Tay. “NBGossip - Neighborhood Gossip with Network Coding Based Message Aggregation”. *Proceedings of the* 4th *IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, Pisa, Italy, Oct 2007.

Feng Lu, Liang-Tien Chia. “Multimedia Web Services for an Object Tracking & Highlighting Application”. *Proceedings of the* 13th *International Multimedia Modeling Conference (MMM)*, Singapore, Jan 2007.

ABSTRACT OF THE DISSERTATION

Downclocking WiFi to Improve Energy Efficiency in Mobile Devices

by

Feng Lu

Doctor of Philosophy in Electrical Engineering (Computer Engineering)

University of California, San Diego, 2014

Professor Tara Javidi, Co-Chair
Professor Alex C. Snoeren, Co-Chair
Professor Geoffrey M. Voelker, Co-Chair

As manufacturers continue to improve the energy efficiency of battery-powered wireless devices, WiFi has become one of the most significant power consuming components on them. Hence, modern devices fastidiously manage their radios, shifting into low-power listening or sleep states whenever possible. Unfortunately, recent studies have shown that a wide variety of popular applications make frequent and persistent use of the network, frustrating attempts to keep the WiFi chipset in a power-efficient state. Even a less chatty application like email could still constitute a significant source of energy consumption. A fundamental limitation with current power-saving approaches is that the radio is incapable of transmitting or receiving unless fully powered.

In this dissertation, I introduce a widely-used technique for energy savings in CMOS, namely dynamic frequency scaling (DFS), to standards-compliant WiFi radios. A fundamental limit that prevents decreasing the clock frequency in WiFi chipsets is the Nyquist sampling theorem, which states that the sampling rate must be twice the signal bandwidth. Leveraging the inherent sparsity in direct sequence spread spectrum (DSSS) modulation in 802.11b, this dissertation proposes a transceiver design based on compressive sensing that allows WiFi devices to operate their radios below the Nyquist rate, thus consuming less power. Recognizing that modern WiFi chipsets are shifting towards orthogonal frequency-division multiplexing (OFDM), I further extend the benefits of DFS to OFDM communication systems. Based on my observations of OFDM signaling, I show that the aliasing effect, resulting from undersampling, essentially transforms the original per-subcarrier quadrature amplitude modulation (QAM) into a more dense, but still decodable, QAM modulation.

I implement both the 802.11b DSSS-based (SloMo) and 802.11a/g OFDM-based design (Enfold) on the Microsoft Sora platform, a full-stack programmable software-defined radio. Both implementations are standards-compliant, and SloMo is also fully backwards compatible with existing WiFi deployments. My experiments and evaluation demonstrate that downclocking WiFi is both practical and beneficial, even when clock rate is reduced by a factor of five. Specifically, my downclocked design can reduce energy consumption by over 30% for many popular smartphone apps even with a deliberately conservative power model.

Chapter 1

Introduction

Smartphones and other battery-powered wireless devices are becoming increasingly popular platforms for all manner of network applications. As a result, the high energy usage of the radios on these devices has become an imperative issue to be addressed. For example, Rozner *et al.* report that the base energy consumption of an HTC Tilt 8900 series phone ranges from 155–475 mW depending on the display backlight intensity. In comparison, for the same phone model, the WiFi radio consumes over 1000 mW when active [54]. Not surprisingly, a large number of techniques have been proposed to help manage the power consumption of both cellular and WiFi radios. Focusing particularly on the WiFi domain, the basic approach has been to implement extremely low-power listening or sleep modes and transition devices into operational mode as little as possible [22, 38, 54]. Unfortunately, recent studies have shown that a wide variety of popular smartphone applications make frequent and persistent use of the network [45, 68], frustrating attempts to keep the WiFi chipset in a power-efficient state.

Transitioning in and out of sleep mode adds significant overhead, both in terms of time and energy. In particular, in addition to the costs associated with powering up the transceiver, once awake the WiFi chipset still needs to participate in the carrier sense multiple access (CSMA) channel-access scheme which frequently results in the device spending significant time in idle listening mode waiting for its turn to access the channel [38, 70]. Moreover, once a device is done transmitting or receiving, it will remain in a tail state for some period of time in anticipation of subsequent transmissions [38, 45]. In the case of WiFi, idle power consumption is almost identical to that in transmission (Tx) and reception (Rx) modes.

To amortize the cost of frequent state transitions, the 802.11 power save mode (PSM) specification allows WiFi devices to save power by only waking up on the granularity of the 100-ms access point (AP) beacon interval to check for incoming traffic. Hence, while useful for bulk data transfers [22] or situations where traffic patterns can be predicted precisely [51], PSM-style power saving approaches are often ineffective for applications that need to send or receive data frequently [70]. Finally, even otherwise-effective power saving mechanisms implemented by the WiFi chipset can be overridden by applications in many popular frameworks [11, 12]: some apps prevent the WiFi device from entering PSM mode, forcing the WiFi card to stay awake in an effort to improve performance [17, 64].

Consequently, given how popular apps interact with the network and manage power states, the WiFi radio spends a large fraction of time in high-power states (idle state in particular). Furthermore, I observe that the data rates for many popular apps are small—ranging from 10s to 100s of Kbps— compared to the channel capacity offered by modern WiFi networks (e.g., $\geq$ 1 Gbps in 802.11ac). Traditionally, when faced with low-demand clients, system designers have used excess channel capacity to improve reception rates by introducing redundant coding and/or reducing transmission power. For example, 802.11n specifies a wide variety of link rates, ranging from 1 to 150 Mbps and beyond. The lower link rates use more robust encoding and signaling schemes that can be decoded at lower signal-to-noise ratios (SNRs). These schemes translate into longer range or the ability to decrease transmission power which, along with the potential for power savings at the *sender*, can increase spatial reuse. Unfortunately, the amount of energy saved by transmission power control is rather limited, primarily due to the large asymmetry of WiFi traffic (dominated by the downlink traffic from access point to clients [27]). Even for balanced traffic patterns such as those generated by voice over Internet Protocol (VoIP) applications, transmission often consumes only a small fraction of the overall energy for WiFi.

1.1 Downclocking

In this dissertation, I exploit a widely-used technique, namely dynamic frequency scaling (DFS), to reduce the power consumption of WiFi chipsets. Given that the power consumption of a complementary metal-oxide-semiconductor (CMOS) computing device is proportional to its clock rate, modern central processing units (CPUs) dynamically adjust their clock frequency to conserve power or to reduce the amount of heat generated when the workload is light. Unlike CPUs, where the entire pipeline only handles digital data, WiFi chipsets (or communication devices in general) need to interface with real-world signals. A fundamental limit that prevents lowering the clock frequency is the Nyquist(-Shannon) sampling theory, which dictates that the sampling rate must be at least twice the occupied bandwidth of the signal. Therefore, the clock frequency of a WiFi chipset is essentially gated by the Nyquist sampling rate, which suggests a minimum clock frequency of 40 (80) Mhz for WiFi 802.11a/b/g(n).

After exploring the properties of WiFi signaling, this dissertation proposes a number of techniques to bypass the Nyquist sampling rate constraint for WiFi chipsets. Specifically, I observe that the information rate is eleven times smaller than the occupied bandwidth in 802.11b under the direct sequence spread spectrum (DSSS) modulation scheme. By leveraging this inherent information sparsity, I can decode the transmitted information bits with fewer samples than the sampling theory requires using compressive sensing. Unlike the DSSS signal, however, orthogonal frequency-division multiplexing

(OFDM) signals (used in 802.11a/g/n/ac) are extremely dense and the compressive sensing technique is no longer applicable. I instead recognize that there exists well-defined structure in terms of how the subcarriers fold up under aliasing due to undersampling. As a result, aliasing essentially transforms the original per-subcarrier quadrature amplitude modulation (QAM) into a more dense—but still decodable—QAM modulation.

To ensure correct WiFi medium access control (MAC) protocol interactions, the WiFi transceiver needs to transmit packets at a reduced clock rate as well. Given that signal spectrum is proportional to clock rate, a signal transmitted while operating in downclocked state unavoidably yields a narrower bandwidth than a standard WiFi signal. Fortunately, I find that there exists large design headroom in commercial WiFi chipsets while decoding DSSS modulated WiFi packets. By carefully crafting the downclocked signals given sufficient SNR, the excess design margin allows us to successfully convince standards-compliant WiFi receivers that the incoming non-standard-conforming signals are valid DSSS modulated signals. Therefore, I believe that it is feasible to realize downclocked packet transmission and reception while being fully standards-compliant. My thesis is that it is possible to save energy in mobile devices by applying dynamic frequency scaling to the WiFi transceiver.

1.2 Contribution

Practical communications systems often involve more complex operations than merely frame reception and transmissions. There are many other system-level components, such as timing synchronization and phase compensation, that are critical for proper packet decoding and encoding. For example, timing synchronization helps the receiver to determine the correct symbol boundaries. Most of these functions involve direct manipulation of raw channel samples. Given that the WiFi chipset was not designed to operate in the downclocked mode, it remains questionable whether these components would continue to work with reduced channel samples. In this dissertation, I systematically study the impact of downclocking on these components and suggest alternative implementations if necessary.

Even if it were possible to implement downclocked WiFi transceivers, the effective SNR under downclocking is likely to be smaller than that at full clock rate due to reduced channel samples. As a result, my downclocked design explicitly trades SNR (and throughput indirectly) for energy savings. To understand whether such a trade-off is worthwhile, I sample a wide range of popular smartphone apps and thoroughly evaluate their energy-consumption behavior under downclocking.

In summary, this dissertation makes the following contributions:

- Although the signal spectrum is fully occupied, I observe that the information rate is substantially smaller than the signaling rate for DSSS modulation in 802.11b. I further notice that the DSSS modulation process is very similar to the decoding of a sparse signal based on compressive-sensing techniques. Therefore, leveraging the information sparsity in DSSS, I propose a compressive-sensing-based DSSS reception pipeline inspired by a previous design [62]. My downclocked DSSS design allows 802.11b WiFi devices to reduce their clock rates by a factor of five or more while still being able to decode regular DSSS modulated packets.

- I extend downclocked reception to OFDM signals in 802.11a/g/n/ac by exploiting the aliasing effect. I first identify the essential building blocks of an OFDM-based modulation scheme: fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT). Under downclocking, the FFT component remains intact while the IFFT component only receives half (a quarter) of the original time-domain samples at 50 (25)% clock rate. I then perform the IFFT operation on the reduced samples and mathematically show that there exists well-defined structure in terms of how the subcarriers fold up as reflected in the IFFT output. In particular, by folding the subcarriers on top of each other, I decode frames transmitted with a form of QAM as if they were encoded with a more dense QAM modulation.

- I find that commercial WiFi chipsets typically employ a matched filter in their reception pipeline, where the output of the matched filter is compared with a pre-determined threshold to decode the received data. In other words, the decoding decision is entirely based on the matched filter output but not the shape of the input signal. Based on this observation, I propose several waveforms that could successfully pass the matched filter to yield a correctly-decoded output but with lower signaling rates than DSSS. These proposed waveforms effectively allow a downclocked WiFi radio to transmit non-standard-conforming signals that are decodable by commercial WiFi chipsets.

- I carefully analyze the impact on the entire WiFi transceiver pipeline due to downclocking. First, I discern the various functional units that form the reception and transmission pipelines. For each functional unit, I then determine its role and implementation details, and evaluate whether this functional unit continues to operate under downclocking. In case it fails, I provide an alternative implementation that achieves the same purpose and remains fully functional when downclocked. Otherwise, I study the performance impact due to reduced clock rate through theoretical analysis or extensive experimentation.

- I discuss the modifications needed for existing deployed WiFi infrastructure to support downclocked operations. Specifically, I show how I could re-use the existing (re)association frames, which are defined in the 802.11 specification, to notify the AP that a node is going to enter (leave) downclocked state. For my OFDM-based design, I also explain the necessary modifications at the AP to permit downclocked OFDM modulated WiFi packet reception, beacon listening, and AP scanning.

- I implement my solutions on the Microsoft Sora platform (an open-source software-defined radio platform), and experimentally verify the standards compatibility of my design. I further report the downclocked transmission and reception performance that I am able to achieve under different SNR conditions and packet sizes. Finally, I evaluate the potential energy-consumption behavior of a downclocked WiFi chipset based on network traces from a wide range of popular smartphone apps. Even with a deliberately conservative WiFi power model, my findings suggest that my approach could reduce power consumption by up to 34% compared to the default power-save mode in 802.11.

1.3 Organization

The remainder of this dissertation is organized in the following manner.

Chapter 2 provides a general overview of the WiFi chipset architecture, previous work on designing energy-efficient WiFi systems, and the motivation for introducing dynamic frequency scaling to WiFi radios.

Chapter 3 discusses my downclocked 802.11b transceiver design that is able to transmit and receive DSSS WiFi packets while downclocked. Further, I evaluate the communication performance of my design and demonstrate that it is possible to run real applications on a downclocked WiFi radio.

Chapter 4 contains a detailed analysis on the per-subcarrier channel response under aliasing. Based on the findings, I then present my downclocked 802.11a/g receiver design, and investigate the impact on OFDM-communicaiton performance due to downclocking.

Chapter 5 studies the potential impact on energy consumption of my downclocked design. I collect network traces from a wide range of popular smartphone apps and evaluate the corresponding energy consumption based on a conservative WiFi power model.

Chapter 6 concludes the dissertation and points out how this dissertation could be extended.

Chapter 2

Background and Motivation

This chapter provides background information about the technologies discussed in the rest of this dissertation. In addition, we present an overview of existing research literature on energy-efficient WiFi design. Finally, we discuss our motivation for introducing dynamic frequency scaling to WiFi chipsets.

2.1 A Communication System Primer

Shannon introduces the fundamental law of data compression and transmission (also known as Shannon theory) in his “A Mathematical Theory of Communication” [57], which later on becomes the foundation of modern day communication systems. Shannon theory establishes the maximum data rate at which information can be reliably transmitted over a communication channel of a particular bandwidth given some signal-to-noise ratio (SNR) value. More specifically, the channel capacity C is defined as:

$$C = B\log_2(1+\frac{S}{N}) \tag{2.1}$$

where B is the bandwidth of the channel in Hertz, S is average received power and N is the average noise or interference power (both in Watts). The Shannon theory does not specify a particular data encoding (modulation) scheme that can be used to achieve the channel capacity in practical communication systems. It is up to system designers to determine which modulation and coding technique to apply. As a general guideline, the achievable throughput under any modulation scheme in a communication system is bounded by the Shannon channel capacity.

A typical communication system consists of the following three elements: the sender, the channel, and the receiver. Now suppose some modulation and coding scheme is selected at the sender to encode and transmit information on the channel, how could the receiver decode the transmitted signal and recover the original information? Most (if not all) modern communication systems are based on digital circuits. Given that the signal transmitted on the physical channel is analog in nature, this analog signal needs to be first converted to digital samples (the sampling process) before the decoding process can be started. To perfectly reconstruct the analog signal, the sampling rate, which determines how frequently a signal sample should be acquired, must be at least twice the highest frequency (bandwidth) contained in the signal as mandated by Nyquist sampling theory [58]. When sampling below the Nyquist rate, aliasing occurs where the signal spectrum folds up, and it is generally impossible to discern the original signal.

However, it is possible to bypass the Nyquist sampling constraint and recover an analog signal with a lower sampling rate under certain circumstances. For example, when the signal is sparse, compressive sensing techniques can be used to efficiently sample and reconstruct a signal with much smaller sampling rate than Nyquist sampling theory requires [19]. At a high level, compressive sensing is a signal-processing methodology for dealing with fewer samples that leverages the presence of sparsity and compressibility in a signal. Mathematically, compressive sensing recovers the original signal by finding solutions to a set of underdetermined linear equations. Since its inception, compressive sensing has been widely applied in numerous application domains such as medical imaging and statistics.

2.2 WiFi Overview

Wireless local-area networking (WLAN) technologies have become pervasive, and their increasing prevalence has raised users' expectations for performance and availability. Today's gadget-oriented consumer expects high-quality Internet access not only in their homes and workplaces, but also in businesses, public areas, and increasingly even buses and airplanes. Moreover, an ever-expanding variety of consumer electronics have come to rely on wireless networking technologies in order to operate. Due to its cheap infrastructure cost, WiFi has also been used by wireless carriers and their customers to offload cellular traffic [36]. To cope with the ever-increasing demand placed on WiFi networks, advances in physical layer technologies are delivering additional channel capacity.

The first generation of WiFi products is based on 802.11b [32], which delivers a peak data rate of 11 Mbps. The 802.11b specification defines two physical layer modulation schemes: direct sequence spread spectrum (DSSS) and complementary code keying (CCK). When channel condition (SNR) is poor, the more robust DSSS modulation is used yielding a 1 or 2-Mbps data rate; as SNR improves, the compact CCK modulation can be used to achieve higher data rates (5.5 and 11 Mbps). Realizing the speed limitation of 802.11b products, the second WiFi generation (i.e., 802.11g [33]) employs the popular orthogonal frequency-division multiplexing (OFDM) modulation technique, which significantly boosts the maximum data rate supported to 54 Mbps.[1] Due to its high speed and backwards compatibility with 802.11b, 802.11g-based WiFi products enjoyed great market success and were widely deployed in home and enterprise environments. With advances in multiple-input-multiple-out (MIMO) technology [47], more recent WiFi generations, e.g., 802.11n [33], use multiple antennas and a wider

[1]The 5-Ghz counterpart, 802.11a [31], is almost identical to 802.11g but has been less successful due to cost and compatibility issues.

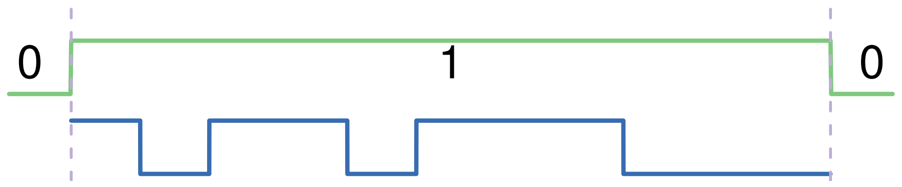


Figure 2.1. DSSS modulation waveform.

channel bandwidth (40 MHz) to increase data rates, which leads to an order-of-magnitude speed improvement from 54 Mbps to 600 Mbps. Finally, 802.11ac [5], which was just approved by the IEEE standardization body early this year, promises speeds of over one gigabits per second for the next generation of WiFi devices.

In general, each newer WiFi generation is entirely backwards compatible with the older ones. For example, although 802.11b has been released for over a decade, it remains widely supported in both deployed WiFi networks and the newest 802.11ac chipsets (e.g., Broadcom 4335) and routers (e.g., Cisco EA6500). Interestingly, due to its robust communication range and low cost, 802.11b is the only supported WiFi mode in some special-purpose devices [24, 25, 67].

2.2.1 Physical Layer

The WiFi physical layer defines the means of transmitting and receiving data frames over the shared wireless link. Over the years, WiFi has specified three modulation standards (DSSS, CCK and OFDM) for encoding data on the wireless medium. All WiFi generations, except the first generation which only supports DSSS and CCK, can communicate with any of the three schemes. In this dissertation, our efforts are primarily focused on DSSS and OFDM.

Direct Sequence Spread Spectrum

In DSSS, instead of transmitting an information bit directly, each bit is first combined with a high-speed pseudo-random numerical sequence (also known as “chips”) via a modulo-2 adder (XOR function). The output of the modulo-2 adder is then used for transmission. As a result, the signaling rate (chip rate) is much higher than the information bit rate as shown in Figure 2.1. Under the 802.11b specification, the eleven-chip Barker sequence $(+1, -1, +1, +1, -1, +1, +1, +1, -1, -1, -1)$ serves as the pseudo-random sequence in the modulo-2 adder. At the receiver side, the incoming signal is correlated with the eleven-chip Barker sequence, and the correlation output helps the receiver to lock onto the received signal and start the decoding process. Depending on whether the in phase component is used or not, the resulted data rate with DSSS is 1 or 2 Mbps.

Orthogonal Frequency-Division Multiplexing

OFDM is the dominant modulation scheme in WiFi, and has also been embraced by many other communication systems. It divides the entire spectrum band into many small and partially overlapping, but orthogonal, frequency bands known as subcarriers. These subcarriers are then used to carry data without inter-carrier interference, eliminating the need for guard bands. Each subcarrier could be independently modulated with a different modulation scheme (e.g., phase-shift keying, quadrature amplitude modulation, etc.), although all subcarriers are modulated using the same scheme under WiFi. As the signal-to-noise ratio (SNR) gets higher, the per-subcarrier modulation scheme becomes more dense. For example, 802.11a/g defines rates of 6, 9, 12, 18, 24, 36, 48 and 54 Mbps where the modulation scheme on each subcarrier changes from binary phase shift keying (BPSK) to 64 quadrature amplitude modulation (64-QAM).

2.2.2 Rate Selection

Given the wide range of transmission data rates, it is entirely up to the sending node to determine which rate to use for transmitting a packet. Since the WiFi channel condition varies across time and space, a smart WiFi sender will dynamically adapt its modulation scheme to decide the bitrate employed. The ultimate purpose of rate selection is to improve the network goodput between the sender and the receiver. In general, as the SNR gets worse, for example when the node moves away from the access point, the transmission data rate decreases. Although the selected data rate could vary significantly from 1 to 54 Mbps, the transmission power remains approximately constant across the entire range [28] assuming only a single antenna is used. Rate adaptation gets more complicated for recent WiFi generations, such as 802.11n and 802.11ac, as more variants are introduced in the selection process. For example, the sender now needs to decide how many antennas to be used even for the same bitrate. In this dissertation, we focus our discussion on the single-input-single-output (SISO) configuration of MIMO and assume that there is only one active antenna at both the sender and the receiver side.

2.2.3 Media Access Control

Since the wireless channel is a shared medium, if multiple nodes transmit at the same time, their transmissions will interfere with each other and lead to packet collision where no transmission is successful. To prevent collisions, a media access control (MAC) protocol is introduced to coordinate the channel access. In particular, WiFi uses the carrier sense multiple access with collision avoidance (CSMA/CA) mechanism to mediate access to the channel. If a node needs to send a frame, it will first sense the wireless medium to determine whether there is any ongoing transmission, which is also known as clear channel assessment (CCA). If the channel is sensed busy, the transmission attempt is aborted and the node will wait until the channel is free again. Otherwise, if the channel has been free for some period of time, the node transmits the frame. After receiving this frame successfully, the intended receiver will reply back with an acknowledgment (ACK) frame. If the sender receives the acknowledgment, it will remove the frame from its transmission queue and conclude that this transmission is a success. On the other hand, if the sender fails to receive an acknowledgment, for example when the transmission is not successful due to insufficient SNR, the frame is assumed lost. The sending node will back off and restart the transmission once the channel becomes available. Finally, the packet is discarded after multiple failed attempts.

2.3 The Potential of Downclocking

Given that wireless link speeds continue to increase, mobile devices are more and more likely to use only a small fraction of the channel capacity. With current WLAN technologies, however, use of the network is an all-or-nothing affair in terms of power. Said another way, technologies like WiFi are not power proportional with respect to bandwidth for applications that require continuous, low-rate connectivity. While applications that need only to transmit can simply power their radios down when not in use, applications that must receive small but frequent packets are forced to leave the radio on at all times, consuming essentially the same power as an application that was receiving at full link rate.

In the past, system designers have used excess channel capacity to improve reception rates by introducing redundant coding and/or reducing transmission power, known as transmission power control. Lower link rates use more robust encoding and signaling schemes that can be decoded at lower signal-to-noise ratios (SNRs), which translate into longer range or the ability to decrease transmission power which, along with the potential for power savings at the *sender*, can increase spatial reuse. We argue that the time has come for WLAN designers to additionally consider the power required to successfully decode the signal: in particular, one can instead turn excess channel capacity into an opportunity to save power at the *receiver*.

The power consumption of a complementary metal-oxide-semiconductor (CMOS) computing device is proportional to its clock rate [52]. In general, CMOS logic gates use current when switching states. As the clock rate increases, gates switch more frequently, which leads to increased power consumption. More specifically, the power consumption P of CMOS circuits is directly related to the core operating voltage v and the clocking frequency f based on the following relationship:

$$P \propto f \times v^2.$$

Not surprisingly, dynamic frequency scaling (DFS) has long been used as a technique to save power in a variety of computing domains [26, 66]. Indeed, DFS has been part of the *de facto* feature set offered by modern microprocessors, i.e., to scale the central processing unit (CPU) frequency up or down to save power, for decades in commercial and industrial settings. For example, CPU frequency scaling is implemented in the Linux kernel, and the *cpufreq* module is loaded automatically when the system starts up. Fundamentally, the same rules apply to wireless transceivers: downclocking the radio hardware can result in significant power savings. The challenge in downclocking radio equipment, however, is that the receiver must sample the channel at least twice the bandwidth of the signal [58]. As a practical matter, today's WiFi devices are designed in such a way that the frequency of the entire radio pipeline is gated by the sampling rate.

Figure 2.2 shows a typical WiFi transceiver architecture [13, 70]. The analog baseband signal is first processed by a baseband filter to confine the signal to the desired band. It is then sampled by an analog-to-digital converter (ADC) and data samples are passed to the baseband processor, which decodes the signal and uploads the recovered

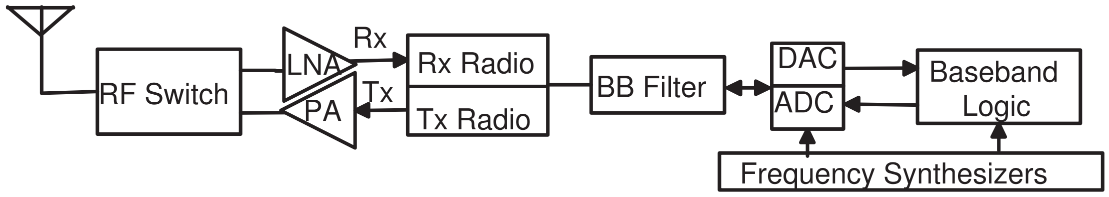


Figure 2.2. A simplified WiFi card architecture.

frame to the host. The entire chipset is driven by a common crystal oscillator, which feeds the frequency synthesizer and the phase-locked loop (PLL). The frequency synthesizer generates the center frequency for radio frequency (RF) operation while the PLL serves as the clock source for the ADC and baseband processor. For a 22-MHz 802.11b channel, the radio runs at 44 MHz (or faster).

As a result, the channel sampling rate directly determines the permissible clocking rate—and power consumption—of the WiFi card. Studies have shown that the power consumption of popular WiFi chipsets (e.g., from Atheros and Netgear) does indeed vary with frequency [13, 70], although the precise relationship depends on what the device is doing (sending frames, receiving frames, or idling) and differs across chipsets. Table 2.1 shows the reported energy consumption of a popular WiFi chipset while operating at various clock rates [70]. These measurements were obtained by varying the channel width for a 6-Mbps OFDM 802.11g encoding, so the device was operating on channel bandwidths of 5, 10, and 20 MHz, respectively.

The savings are sub-linear (e.g., 36% savings for a 50% clock rate reduction on the Atheros 5414 chipset [70]), but current devices were not designed to be downclocked, hence it is unlikely that they are optimized to be power-efficient at frequencies other than their target operating point. Power savings can even be obtained on research-grade software radio devices; the universal software radio peripheral (USRP) [23] achieves a 22% power savings when downclocked by half [70]. Hence, we believe that future designs can be refined to take greater advantage of downclocking opportunities.

Table 2.1. Power draw of the Atheros 5414 WiFi chipset in the LinkSys network interface controller (WPC55AG) at various clock rates [70].

Clock rate	25%	50%	Full rate
Idle	640 mW	780 mW	1200 mW
Reception (Rx)	980 mW	1440 mW	1600 mW
Transmission (Tx)	1210 mW	1460 mW	1710 mW

2.4 Downclocked Transmission

It is not obvious that downclocking a radio would be beneficial when transmitting data: the lower the data rate, the longer the transmission takes. Hence, with existing chipsets one should transmit as fast as possible and place the radio back into low-power mode as soon as transmission is complete. Alternatively, one could realize similar savings by transmitting at a low data rate and scaling back the transmission power. These approaches, however, presume that the frequency and/or power of the transceiver can be adjusted efficiently.

Moreover, even if a device only receives data, the 802.11 specification requires that it transmit an ACK frame to confirm receipt of each data frame—and the ACK frame must be sent within a strict, 20-μs inter-frame time (SIFS). As with reception, Nyquist requires that the transceiver operate at twice the signal bandwidth to transmit the standard Barker sequence. While some chipsets, such as the MAXIM 2831, are able to switch back to full clock rate in time to transmit an ACK frame, others take substantially longer (e.g., an Atheros 5414 takes roughly 125 μs to switch clock rates [70]). In such cases, to realize the benefits of downclocked reception, the transceiver needs to transmit an ACK frame at a slower clock rate than a standard-compliant WiFi transmitter will accept. (The reception power draws in Table 2.1 assume the device remains downclocked for ACK transmissions.)

The potential benefits of downclocked transmission go even further when considering the energy spent on clear channel assessment (CCA) when a node attempts to gain access to the channel. Previous studies have shown that CCA is the dominant power drain when there is a high contention level in the network [38, 70]. Most commercial WiFi chipsets implement the carrier sensing component of CCA (i.e., determining whether the channel is free) using energy detection, which can be conducted at virtually any clock rate. Moreover, modern WiFi cards seem to be more power proportional when in this so-called idle listening state. As shown in Table 2.1, the measured Atheros chipset consumes 47% less power in idle listening mode when downclocked by a factor of four.

However, once the channel is detected to be idle, a WiFi station needs to attempt to transmit a frame within a very short order (as little as 50 μs depending on its current back-off interval). Given the switching times of commodity chipsets, these timing requirements suggest that WiFi devices are likely to need to perform carrier sensing and frame transmission at the same clock rate. In other words, to perform CCA while downclocked, the WiFi device must be prepared to transmit while downclocked as well.

Finally, Table 2.1 also indicates that the power savings from downclocked transmissions themselves may be slightly less significant than reception (e.g., 29% for a 4$\times$ clock rate reduction), but, when combined with the ability to remain in idle listening states for longer periods of time, the savings can be dramatic.

2.5 Related Work

Energy management is crucial for battery-powered mobile devices, where network activities can consume a significant portion of the available energy. For example, one recent study showed that a popular smartphone consumes almost two orders of magnitude (1114 vs 16 mW) more power when its WiFi device is active than when it is idle [8]. While vendors continue to improve the power efficiency of their WiFi chipsets, the fact remains that they frequently account for a significant portion of the entire device's energy budget. Of course, most devices do not need to use WiFi all the time, and modern devices actively manage their radios, shifting into low-power listening modes or even sleep states whenever possible. There has been a great deal of work on improving the energy efficiency of WiFi devices. These efforts can be broadly classified into three categories: 1) improvements to 802.11 power save mode (PSM), 2) systems that duty cycle the WiFi device, and 3) attempts to decrease transmit power.

2.5.1 Efficient Power Save Modes

Most approaches rely on placing the device in a low-power sleep mode whenever possible. The two basic alternatives are to coordinate these periods of sleep between the access point and the device, either through periodic polling (as with the 802.11 PSM standard) or deliberate scheduling [54]. Others have proposed dynamically adjusting sleep periods based upon a client's traffic pattern [9, 34]. Researchers have previously noted the disparity between modern 802.11 link speeds and the traffic demands of many clients. μPM suggests powering down low-demand WiFi clients between individual frame transmissions [37], relying upon 802.11 devices retransmitting unacknowledged frames to limit losses. Catnap [22] extends this approach by estimating bottleneck throughput and scheduling client wake-ups based upon the predicted availability of data from the wide-area network.

One challenge with these approaches is that, when awake, a WiFi device must participate in the channel contention process to gain channel access. Studies have shown that this process can consume considerable amounts of energy, especially in dense deployments where nodes are in range of multiple APs. SleepWell coordinates sleep cycles among neighboring APs to decrease contention during wake-ups, thereby increasing client power efficiency [38].

Finally, even a well-defined sleep scheduling or policy implemented on the WiFi chipset could be overridden by a user-specified WiFi sleep policy intentionally or un-intentionally. For example, the Android framework (version 3.1+) exposes a particular type of WiFi lock (*WIFI_MODE_FULL_HIGH_PERF*), which will keep a WiFi device from entering PSM mode [11, 12]. Therefore, apps could force the WiFi card to stay in constant awake mode (CAM) to reduce network response time for performance improvements [17, 64]. Perhaps more devastating, these WiFi lock services might be misused (e.g., bugs) and lead to WiFi energy wastage [4, 46].

2.5.2 Device Duty Cycling

Others take a more drastic approach: rather than entering low-power sleep modes, they identify times when it is possible to simply turn a WiFi device off entirely. One early system, SPAN [14], turns off entire nodes in multi-hop ad hoc wireless networks if the connectivity of the network can be preserved without them.

In more general environments, systems have been designed to keep WiFi powered down by default, and use an out-of-band signal to asynchronously alert the device of pending data [8, 59]. Since smartphones may frequently be outside the coverage area of a WiFi AP, the only reason to keep the WiFi transceiver powered is to determine when coverage returns. Many systems have attempted to reclaim this energy by instead duty cycling WiFi radios based upon predictions of WiFi availability. These predictions are variously based upon the detection of nearby Bluetooth devices [10] or cell towers [53], or historical device movement patterns [42].

2.5.3 Transmission Power Control

Finally, a direct approach to decreasing WiFi power draw while transmitting is to reduce radiated energy. WiFi transceivers can leverage transmit power control to emit signals using sub-mW energy when the SNR is high. Unfortunately, despite the obvious attractiveness of such an approach, studies have repeatedly shown that adjusting transmit power has little impact on the total power draw of commercial 802.11 devices due to the limited power consumption of the power amplifier relative to the rest of the electronics [28, 48].

2.5.4 Downclocking

Researchers have argued that devices could dynamically adjust their sampling rate based upon the frequencies contained within the observed signal [20], but their approach is not directly applicable to the encoding schemes employed by WiFi. Recently, compressive sensing and sparse recovery have been applied to bring the sampling rate below Nyquist. As a result, the clock frequency and hardware complexity can be reduced substantially. For example, compressive sensing has been widely used in the field of spectrum sensing, which could potentially lower the analog-to-digital converter (ADC) sampling rate by orders of magnitude [49]. Hassanieh *et al.* [30] showed how to reduce the runtime of Global Positioning System (GPS) synchronization by exploiting the sparse nature of synchronization.

In the context of WiFi, recent proposals argue that next-generation systems should support multiple channel widths and adapt their instantaneous channel width based on the offered load [13],[2] and develop mechanisms to detect packet arrivals in a downclocked state [70]. Downclocking a receiver through dynamic frequency scaling has been applied in the wireline context in the past [56], but we are not aware of any similar schemes in the wireless domain.

[2]Although stations operating in different bandwidths cannot decode each other's transmission and the 17-ms switching overhead makes co-existence challenging.

2.6 Summary

Given the importance of WiFi, not surprisingly there exists a rich body of existing literature on WiFi energy efficiency. Broadly speaking, these efforts can be divided into three areas: energy-efficient power save mode, duty cycling WiFi radio, and transmission power control. We take a dramatically different approach to reduce WiFi energy consumption by downclocking WiFi. A fundamental challenge with downclocking WiFi, however, is the Nyquist sampling rate constraint. In the next two chapters, we will present the techniques we developed to address the Nyquist challenge.

Chapter 2, in part, is a reprint of the material as it appears in Proceedings of the USENIX Symposium on Networked Systems Design and Implementation 2013. Lu, Feng; Voelker, Geoffrey M.; Snoeren, Alex C. The dissertation author was the primary investigator and author of this paper.

Chapter 2, in part, is a reprint of the material as it appears in Proceedings of the ACM Annual International Conference on Mobile Computing and Networking 2014. Lu, Feng; Ling, Patrick; Voelker, Geoffrey M.; Snoeren, Alex C. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Downclocking 802.11b

In this chapter, we describe the design of SloMo, our prototype downclocked radio for 802.11b. We start by discussing how compressive sensing can enable downclocked WiFi frame reception under the direct sequence spread spectrum (DSSS) modulation scheme. We then turn to downclocked transmission and describe our approach for designing shorter chip sequences that are decodable by commercial WiFi chipsets. Next we discuss the practical implementation issues of our downclocked design and network impact due to downclocking. Finally, we evaluate the communication performance of SloMo, such as frame reception rate and throughput, and show that SloMo attains nearly identical performance as 100% clock rate in typical WiFi signal-to-noise ratio (SNR) regimes.

3.1 Reception

Our receiver design is based upon an observation that the process of direct-sequence spread spectrum (DSSS) modulation, as employed by the 802.11b standard, bares a great similarity to a recently proposed compressive sensing (compressive sensing) decoding scheme. When the data rate is 1 or 2 Mbps, only DSSS modulation is employed. The difference between the 1 and 2-Mbps encodings lies in whether the quadrature component of the carrier frequency is used: they employ binary phase shift keying

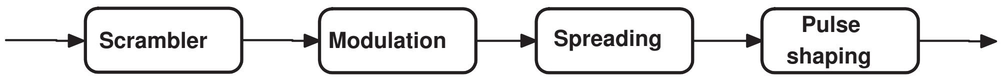


Figure 3.1. Baseband processing Tx chain

(BPSK) and quadrature phase shift keying (QPSK), respectively. To ease our explanation, we focus our discussion on the 1-Mbps BPSK scenario; the methods can be similarly applied to 2-Mbps QPSK encoding as we demonstrate.

In their recent breakthrough, Tropp *et al.* observe that it is possible to employ compressive sensing to decode digital signals while sampling at rates far below the Nyquist rate, provided the signal is sparse in the frequency domain [62]. Their approach mixes the sparse signal they wish to decode with a high-rate chip sequence to spread its signal band. They show that in many cases the information contained in a sub-band of the resulting spread signal turns out to be sufficient for recovering the original signal.

DSSS modulation is analogous to the first stage of this process: the baseband signal is also spread over a wide range of bandwidth. Though the spreading in 802.11b is designed to increase the SNR at the receiver, it also provides the opportunity to apply compressive sensing by only looking at part of the band when SNR is not an issue.

3.1.1 DSSS Modulation

As shown in Figure 3.1, the transmission chain of a standard 802.11b implementation can be summarized as four steps: scrambling, modulation, spreading and pulse shaping. The data is initially "scrambled" by XORing it with a fixed pseudo-random sequence—to avoid long runs of ones or zeros—before being modulated (using BPSK in the 1-Mbps case). The modulated baseband signal is then "spread" by replacing each bit with an eleven-chip Barker sequence to expand the signal. The spreading process serves several purposes. First, it enlarges the spectrum of the original baseband signal by eleven times to make it more robust to channel noise. Secondly, due to the unique

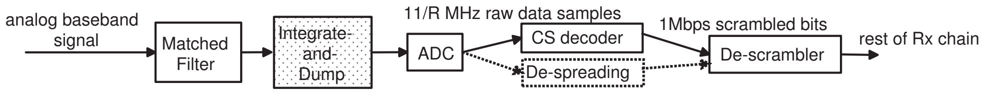


Figure 3.2. Original and modified baseband Rx processing chain. Compared to the original Rx chain, the modified chain adds an additional integrate-and-dump component and replaces the De-spreading part with the compressive sensing decoder.

properties of a Barker sequence, it enables the receiver to more easily synchronize with the transmitted signal. In particular, a Barker sequence has low auto-correlation except when precisely aligned with itself, so receivers can easily determine when they have correctly synchronized with the incoming chip sequence.

Mathematically, one can consider the DSSS spreading process as computing an eleven-chip signal, $\mathbf{C}$, for each bit, $\mathbf{C} = \mathbf{M} \cdot \mathbf{b_i}$, where $\mathbf{b_i}$ is a 2×1 sparse vector ($b_1 = [\mathbf{0}\ \mathbf{1}]^{\mathbf{T}}$ corresponds to a 1 and $b_0 = [\mathbf{1}\ \mathbf{0}]^{\mathbf{T}}$ for a 0), and the Barker sequence $\mathbf{M}$ is given by

$$\mathbf{M} = \begin{bmatrix} +1-1+1+1-1+1+1+1-1-1-1 \\ -1+1-1-1+1-1-1-1+1+1+1 \end{bmatrix}^{\mathbf{T}}.$$

Note that the two rows of $\mathbf{M}$ are simply inverses of each other; hence, both Barker sequences have identical auto-correlation magnitudes—they just result in either positive or negative correlation.

Subsequently, the pulse-shaping stage ensures that the resulting signal spectrum shape conforms to the IEEE 802.11b specification. In particular, the shaped signal has a bandwidth of 22 MHz; therefore, a minimum sampling rate of 44 MHz is required to meet the Nyquist sampling criteria at the receiver side.

Conversely, Figure 3.2 presents a high-level description of an 802.11b receiver baseband processing chain. A matched filter recovers the chip values. In particular, the matched filter correlates the incoming chip samples with the Barker sequence to locate where the bit boundary is, i.e., the first chip in the bit. Once the signal is synchronized, it is sampled every chip time. Therefore, over the course of a single bit duration, eleven sample values will be collected corresponding to the eleven-chip Barker sequence. This chip sequence is "de-spread" by once again correlating it with the Barker sequence to determine whether a 1 or 0 was encoded, resulting in (hopefully) the original 1-Mbps bit stream which is then de-scrambled by XORing with the same scrambler sequence.

3.1.2 Compressive Sensing

We implement compressive sensing using an integrate-and-dump sampler as suggested by Tropp *et al.* [62]. As shown in Figure 3.2, we extend the match filter by introducing an integrate-and-dump stage, which accumulates the output from the matched filter for multiple chip durations, allowing for a lower sampling rate than the standard 11 MHz. The radio can then be downclocked appropriately to achieve a desired compression ratio: sampling is performed on the accumulated output (as opposed to each chip) and the discrete samples—which contain multiple chips—are fed to the rest of the receiver chain.

We can formalize the DSSS sampling process described in the previous subsection as extracting a sample $\mathbf{Y}$ from the received signal, $\tilde{\mathbf{C}}$ (which is the transmitted DSSS signal $\mathbf{C}$ encoded as described above but distorted by the channel), with the diagonal sampling matrix $\mathbf{H}$:

$$\mathbf{Y} = \mathbf{H}\tilde{\mathbf{C}}. \tag{3.1}$$

In a standard receiver operating at full clock rate, $\mathbf{H}$ is an 11×11 identity matrix which simply samples each chip exactly once. $\mathbf{Y}$ is then correlated with the Barker sequence $\mathbf{M}$ to determine the transmitted bit.

With an integrate-and-dump sampler, the measurements can be viewed as a linear combination of the original chip values. For example, suppose only three measurements are desired (i.e., a downclocking ratio of 3/11). The measurements can be viewed as substituting a compressive measurement matrix into Equation 3.1:

$$\hat{\mathbf{H}} = \begin{bmatrix} 1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,1\,1\,1\,1\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1 \end{bmatrix}.$$

Here, our sampling matrix has only three rows because we intend to sample each bit's Barker sequence only three times. Because eleven cannot be evenly divided by three, the integrate-and-dump sampler needs to accommodate varied accumulation length. (We relax this assumption in a later subsection.) In this particular example, to reduce the clock rate by 11/3=3.67×, we choose to take two samples of four chips and one of three. Once the compressive samples are obtained, the baseband logic can be re-engineered to work with the compressed measurements. For example, Davenport *et al.* show the following decision rule[1] can be used [19]:

$$d_i = (\mathbf{Y} - \mathbf{HMb_i})^{\mathbf{T}}(\mathbf{HH^T})^{-\mathbf{1}}(\mathbf{Y} - \mathbf{HMb_i}). \tag{3.2}$$

If $d_0 < d_1$, the bit is decoded as 0, and 1 otherwise. Our proposed receiver baseband processing chain is also presented in Figure 3.2. Since only a single bit is decoded at a time, the decision rule can be simplified as

$$d_i = \mathbf{Y^T}(\mathbf{HH^T})^{-\mathbf{1}}(\mathbf{HMb_i}). \tag{3.3}$$

[1]The middle term (pre-whitened matrix) of Davenport's decision rule is actually $(\mathbf{HMM^TH^T})$ because they assume the basis matrix $\mathbf{M}$ is applied during decoding after the signal has been received. In our DSSS modulation scheme, the matrix is applied during transmission, so we can drop it from our rule.

3.2 Transmission

Recall from the previous section that one of the key roles of the Barker sequence is to allow the receiver's matched filter to identify the beginning of the bit sequence. In particular, given 802.11b's eleven-chip Barker sequence, a bit boundary is within the next ten samples of any chip. Hence, the matched filter simply correlates the chip samples at each of these eleven positions. Because of the auto-correlation properties of the Barker sequence, the start of the bit sequence is clearly indicated by a correlation peak. In theory, the Barker code's correlation maximum is $11\times$ larger than the second maximum. However, when a signal is transmitted over the air, it may get distorted and noise is added. Hence, real receivers never use a peak criteria as high as eleven; on the contrary, commercial WiFi cards use much lower thresholds as our experiments reveal.[2]

Based on this observation regarding the decoding threshold, we design Barker-like sequences whose auto-correlation properties are not as strong as regular Barker sequences, but are still likely to satisfy the matched filter's threshold to allow the receiver to properly identify the bit boundary. Similarly, our sequences have the property that, when correlated with a properly-aligned 802.11b Barker sequence, they can be successfully decoded. (Recall that de-spreading is only performed on properly-aligned chip sequences.) Again, they do not have perfect correlation with the true Barker sequence, but sufficiently high enough to either exceed the threshold for 1s, or low enough to pass for 0s.

The key feature of our Barker-like sequences is that they are shorter than the original Barker sequence, yet transmitted over the same time interval. As a result, each chip in our Barker-like sequence lasts longer than a standard Barker chip. The exact number of chips in the sequence—and, thus, the chip duration—can be chosen to match an intended downclock rate. To search for these sequences, at a high level, we use

[2]For example, Sora [60] decides the maximum value is a peak if the maximum value is at least twice the second maximum.

correlation peak-to-average ratio as a close approximation to decide how good the code sequence is.

Let us represent the original Barker sequence as:

$$C(t) = \sum_{i=0}^{n-1} c_i \cdot g(t - iT_c),$$

where each chip $c_i \in \{-1,+1\}$, T_c is the chip duration, $g(.)$ is a rectangular function with duration $[0,T_c]$, and $n = 11$. The Barker-like sequence we are looking for is denoted as:

$$\bar{C}(t) = \sum_{j=0}^{m-1} \bar{c}_i \cdot \bar{g}(t - i\bar{T}_c),$$

where m is the number of chips in the Barker-like sequence and $\bar{T}_c = \frac{n}{m} \cdot T_c$. This Barker-like sequence is then intended for the clock to operate at $\frac{m}{n}$ of the full rate. Furthermore, the chip values $\bar{c}_j$ remain restricted to -1 and +1 as in the original Barker sequence.

To construct a Barker-like sequence that can be successfully decoded, we consider the correlation result:

$$f_{cor}(iT_c) = \int_{iT_c}^{(i+n-1)T_c} C(t) * \bar{C}(t - iT_c)\, dt. \tag{3.4}$$

Notice that $C(.)$ will repeat itself every nT_c period in Equation 3.4 so we only need to evaluate f_{cor} from 0 to $(n-1)T_c$. For any particular m (we are interested in $1 < m < 11$), there are only a finite number of possible Barker-like sequences, so we search all possible combinations and choose the one with the best auto-correlation property, i.e., maximum correlation result when both sequences align with each other and nearly zero elsewhere.

Figure 3.3 shows some examples of the Barker-like sequences we obtain, and how they compare to the original eleven-chip Barker sequence. To operate at a particular downclocked rate of $m/11$, we select a Barker-like sequence of length m to use for

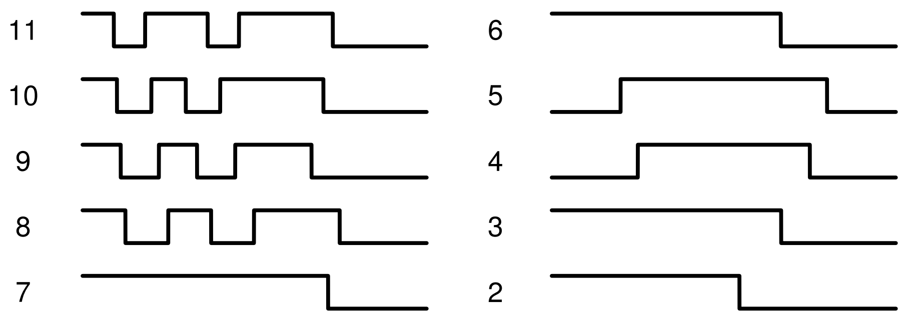


Figure 3.3. Spreading sequences of various lengths, including the 802.11-standard eleven chip Barker sequence and the Barker-like sequences used by SloMo when downclocked accordingly.

spreading. Because the radio is downclocked, each chip will last $(11/m)\times$ as long as a standard chip,[3] and the signal will be more narrow than usual. Specifically, the signaling rate for a Barker-like sequence is smaller than that of the original Barker sequence, given that bandwidth is proportional to signaling rate, we expect the signal spectrum of Barker-like transmission to be narrower than the spectrum of standard WiFi. Figure 3.4 confirms that our Barker-like sequence transmission indeed occupies less spectrum in proportion to downclocking rate for real SloMo (WiFi) packets captured over the air.

Our Barker-like sequences of length four and five deserve special mention. When searching for sequences using Equation 3.4, we exhausted the entire search space without identifying any sequences with only a single clear peak when aligned with the original Barker sequence. Instead, we identified sequences that produce two peaks when correlated with the original Barker sequence: one when perfectly aligned and another when almost completely (roughly 60%) offset. Hence, the receiver will either correctly identify the bit boundary, or be mis-aligned by roughly “half a bit.” In that instance, the receiver will attempt to decode the latter half of one bit sequence and the first half of the

[3]Except when $m = 2$ where the two chips last for 6 and $5\times$, respectively, which we found to be more reliable than two chips of $5.5\times$ A caveat here is that the 6-and-$5\times$ design might increase the complexity of clocking component substantially.

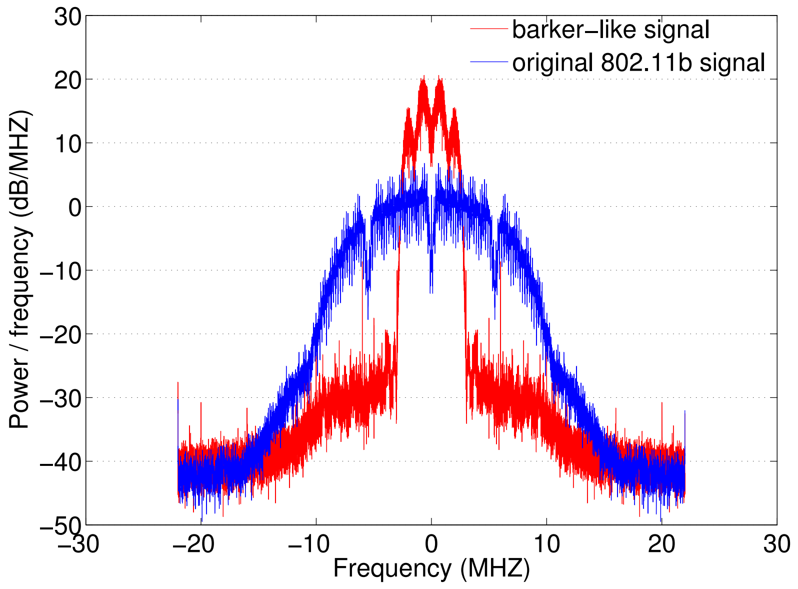


Figure 3.4. Power spectrum density comparison between standard 802.11b and downclocked Barker-like transmission for real WiFi packets captured (downclocking rate: 27.2%).

subsequent bit sequence as one bit sequence. We selected our 4 and 5-chip Barker-like sequences such that the decoding of this mis-aligned chip sequence will decode as the inverse of the correctly aligned one. Because both BPSK and QPSK actually encode data based on the transitions between bits (as opposed to their actual magnitude), either sequence will be decoded correctly.

3.3 Practical Design Considerations

In the foregoing description of our compressive-sensing-based receiver, we assume 1) that the bit boundary is known, 2) compressive measurements are only taken over chips belonging to the same bit, and 3) the number of chips to be integrated varies (as reflected in the measurement matrix given in Section 3.1.2). Here, we first relax the latter two requirements and then return to address the former.

3.3.1 Fixed-Length Integrate-and-Dump

Rather than have variable-length integration periods, an alternative is to have a fixed integration length l and occasionally integrate fractions of a chip value into a measurement. For example, the following measurement matrix (which we employ when the clock is operated at 4/11 of the original rate) serves as a concrete example:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 1 & 1 \end{bmatrix}.$$

Note that the third and fourth samples each integrate a fraction of the ninth chip. Alternatively, if integrating a fraction of the chip turns out to be challenging, we could integrate a fixed number of chips and extend the decoding to multiple bits as a group.

At a raw data rate of 1 Mbps, Nyquist requires a minimum of two measurements per eleven-chip sequence; we cannot downclock the receiver a full $11\times$. Hence, the useful range of integration lengths is between two and five chips. Since eleven is a prime number, for any integration length k ($2 \leq k \leq 5$), the number of compressed samples is not an integer for a single bit. In fact, we need to perform compressive sensing over a minimum of $11k$ chips to produce an integer number of measurements (i.e., eleven measurements). Therefore, rather than decoding one bit at a time, we jointly decode k bits in a group—which exactly corresponds to $11k$ chips. The same decoding rule (i.e., Equation 3.2) can be applied with slight modifications. In particular, $\mathbf{b_i}$ is now a $2^k \times 1$ sparse vector where only one element is 1 and the rest are 0; $\mathbf{M}$ is changed to a Barker sequence matrix of size $11k \times 2^k$ which enumerates all the 2^k possible Barker waveforms corresponding to k bits in its columns; finally, the sampling matrix $\mathbf{H}$ becomes a size

$11 \times 11k$ matrix and each row r_i ($0 \leq i \leq 10$) contains k 1s starting from element index $(i-1)*k$ and 0s elsewhere.

3.3.2 Synchronization

Symbol synchronization is a fairly standard technique and is often implemented in hardware [50]. Unfortunately, locating the bit boundary is slightly more challenging when using compressive sensing. After passing through the integrate-and-dump circuit, the compressed measurements no longer exhibit the excellent auto-correlation property provided by the original Barker sequence. Therefore, the standard correlation and peak searching-based method described in the previous section no longer suffices.

Recall that we are decoding our sample stream in groups of k bits at a time, where each bit consists of eleven chips, but our integrate-and-dump sampler has reduced these chips by a factor of k. Hence, we are always decoding exactly eleven samples at once. If we knew where to start decoding, the first compressed measurement would correspond to the sum of the first k chips of the first bit. Rather than trying to identify the bit boundaries ahead of time, we observe that eleven is a prime number, so one and only one alignment with the sample stream will produce successful decodings—all others will never align regardless of the downclocking factor k.

Because we have no idea which one of the eleven compressed measurements starts a group, we store the decoding results for each possible position simultaneously. For implementation purposes, we keep eleven bit arrays (B_0–B_{10}). Suppose the incoming compressed measurements are labeled S_0, S_1, $\cdots$; we decode S_0–S_{10} and store the result in B_0, S_1–S_{11} to B_1, S_2–S_{12} to B_2, $\cdots$, S_{10*j+i}–$S_{10*j+i+10}$ to B_i ($0 \leq i, j, \leq 10$). Each incoming compressed measurement will complete the decoding of one of the eleven-bit arrays. Meanwhile, we look for the fixed bit pattern of the Start of Frame Delimiter (SFD) among the eleven stored bit-arrays. Once the SFD is identified and found in one of the arrays, we then know the correct bit boundary and only need to keep decoding in one of the eleven arrays.

While the synchronization operation can be conducted in parallel, we implement the process in a single software thread in SloMo as the synchronization stage only lasts for the duration of the preamble (72 μs and 144 μs for short and long preambles, respectively). The SFD is guaranteed to be found within this well-defined time bound (or equivalently, a fixed number of decoded bits) for any valid frame. If no SFD is detected after a reasonable amount of time, the synchronization process is aborted and we start to search for the next packet.

3.4 Interacting with Existing Networks

Because SloMo requires modifications only to the downclocked wireless node and is entirely 802.11b-compliant, it is fully compatible with existing WiFi deployments. No changes need to be made to the access point or other devices on the network to support SloMo. In this section, we discuss how SloMo interacts with a standard 802.11b/g/n/ac basestation, as well the potential interactions with other client nodes due to its use of 802.11b (as opposed to 802.11g/n/ac).

3.4.1 Rate Selection

When operating in downclocked mode, a SloMo node can only decode frames encoded using DSSS—in particular, it is not able to use complementary code keying (CCK) encoding (i.e., 5.5 and 11-Mbps 802.11b frames) or communicate at 802.11g/n/ac rates. Fortunately, the 802.11b standard includes mechanisms for the SloMo node to convey these constraints to the access point (AP). If the SloMo node is currently connected to an AP, before it goes into downclocked mode it can transmit a re-association request frame to inform the AP it only supports 1 and 2 Mbps.

Even if a SloMo node fails to notify the AP of the supported rate change, most APs employ a dynamic transmission rate adjustment algorithm that will throttle the sending rate until it successfully communicates with the SloMo station: when the AP fails to receive an acknowledge (ACK) for frames it transmits at a higher data rate, it will retry at a lower rate and eventually step down to 1 or 2 Mbps.

3.4.2 Protocol Interactions

While SloMo devices must operate at 802.11b speeds, it is clearly desirable to ensure that other network nodes can continue to transmit at 802.11g/n/ac rates if they are so capable. The concern in such environments is that the SloMo node cannot decode such frames, and might cause collisions. Luckily, collisions are straightforward to avoid. 802.11b specifies three different clear channel assessment methods: energy detection, frame detection, or a combination of the two. An 802.11b-compliant device can implement whichever method it chooses. Relying on energy detection alone as its clear channel assessment (CCA) method, our SloMo node could co-exist with any other 802.11g/n/ac node in the network *without* requiring them to turn on protection mode to minimize the impact of throughput loss due to slower 11b rates. This approach may require the network operator to manually turn off protection mode on the AP if SloMo nodes are the only possible set of 802.11b clients.

Additionally, because 802.11b and 802.11g employ different inter-frame timings (for example, the slot time is 20/9 μs for 802.11b/g with protection mode off), one might be concerned about the potential unfairness in channel access contention. We could modify the inter-frame timings for SloMo nodes to ensure fair channel access, but we observe that the standard settings penalize the SloMo node, not the other nodes, and the SloMo node is unlikely to have high demand for the channel given it has elected to go into downclocked mode. Hence, we have not deployed this change on our prototype.

3.5 Network Impacts

Since SloMo only supports 1 or 2-Mbps data rates, an obvious concern is that the lower bitrate transmissions require more airtime, thereby decreasing overall network performance, or, worse, increasing the energy consumption of other nodes in the WiFi network and negating the gains realized by the downclocked node. While certainly possible in theory—or even in practice for highly congested networks [63]—its likelihood depends both on the background usage level in the network and the communication patterns of the downclocked node.

For example, for voice over Internet Protocol (VoIP) applications the typical packet size is roughly 40 bytes, implying that a VoIP node's air time usage is dominated by inter-frame spacing and channel contention resolution rather than data transmission or reception [63]. Hence, a VoIP flow's impact on network throughput is likely to be negligible regardless of the bitrate (clock rate) the node chooses to employ. In other scenarios, however, where the downclocked node is transmitting or receiving large packets, or the network is already reaching maximum capacity, the impact may be noticeable. We observe that both the station and the AP could detect and address such situations. In particular, an AP can monitor the current traffic load on the network, number of power save mode (PSM) clients, and any other pertinent information. For severely congested networks, the downclocked operation may not be allowed. In practice, for most of the popular smartphone apps we study, the impact on free channel airtime is limited.

Moreover, many networks are lightly loaded. For example, a study of our department's wireless network found that 60% of all frames are transmitted without contention—i.e., the initial back-off counters expire without needing to wait for other channel activity [16].

3.6 Prototype

To assess the feasibility of our approach, we implement a prototype compressive-sensing-based 802.11b transceiver architecture in Microsoft Sora, a fully programmable software-defined radio [60]. We show that compressive sensing achieves similar packet reception rates as standard WiFi under reasonable network conditions, even when clock rates are reduced by a factor of five. We also show that downclocked transmission using short Barker-like sequences is feasible when communicating with standard WiFi devices.

3.6.1 Implementation

To allow maximum generality of their radio platform, the Sora architecture differs from the typical WiFi chipset design discussed previously. Rather than implementing a matched filter in hardware and sampling thereafter, the Sora radio board has a fixed sampling rate of 44 MHz and passes the raw data samples directly to the processing pipeline. The matched filter and decoding stages are all implemented in software.

We modify the Sora code by adding an integrate-and-dump sampler into the receiver chain and re-design the bit-decoding algorithm as described by Equation 3.3. We implement both versions of the integrate-and-dump sampler (variable and fixed length). Since Sora's clock rate is fixed at 44 MHz, we are unable to downclock it while transmitting. Instead, we emulate downclocked transmission by repeating data samples to effectively simulate a slower clock. We then employ a root-raised-cosine filter for pulse shaping. Since Sora does not have an on-board automatic gain control (AGC) circuit, we have to realize the AGC in software. Finally, to compensate for the clock oscillator difference between transmitter and receiver, we also implement a phase tracking component to ensure correct decoding of multiple-bit groups. We describe both implementations below.

3.6.2 Phase Tracking & Automatic Gain Control

Recall from Equation 3.3 that the receiver multiplies the compressed measurements ($\mathbf{Y}$) with a set of known coefficients ($\mathbf{H}, \mathbf{M}$, and $\mathbf{b_i}$). However, since in practice we need to decode multiple bits at a time, we have to *subtract* a set of reference values (constellation points) from the compressed measurements. Subtraction requires the magnitude of the signal to be properly matched to the reference values. This equalization is typically achieved through an automatic gain control (AGC) circuit. Unfortunately, Sora does not have a hardware AGC on board so we have to estimate the measurement magnitude in software.

For BPSK and QPSK without compressive sensing, channel samples are taken either from the Barker sequence (representing bit 1) or a negative copy of itself (representing bit 0). Both channel samples are therefore of equivalent magnitude. However, with compressive sensing, the set of chips underlying each measurement are unknown *a priori*. To address this issue, we search for multiple samples with constant magnitude regardless of the underlying data bits. Fortunately, for the general integrate-and-dump design, we can locate such measurements for all possible integration lengths.

Controlling for the signal's magnitude solves half the problem. Because clock oscillators on the transmitter and receiver never perfectly synchronize with each other, constellation points start to rotate unless phase tracking is performed to compensate. In particular, when decoding a group of bits, a bit error will occur whenever the constellation point drifts across an axis (either imaginary or real). We leverage the fact that when a constellation point is not crossing an axis, the results from both in-phase and quadrature phase are valid. When one component is about to cross an axis, the magnitude of the other component is significantly larger. We then quickly switch the decoding results based on the larger one (whose sign remains unchanged).

3.6.3 Experimental Evaluation

In this section, we empirically study the impact on communication performance due to downclocking. First, we measure the frame reception rate for downclocked reception and transmission under different SNR and clock rate combinations. We then evaluate the network throughput under downclocking, and show that the SloMo design is not tied to a particular commercial WiFi chipset. Finally, we demonstrate that it is also possible to run unmodified real application (e.g., Skype) on SloMo.

We conduct most of our experiments using two nodes, a Sora node running our SloMo implementation and a laptop with a commercial WiFi device. The Sora hardware is a Shuttle XPC SX58J3 machine with eight central processing unit (CPU) cores configured with a Sora radio control board and an Ettus Research XCVR2450 radio transceiver. It runs Windows XP modified to support the baseline Sora software and our SloMo modifications. The laptop is a Lenovo T410 with two CPU cores running Ubuntu 10.04 with an Intel 6200 WiFi card. We operate the Sora node and laptop as an ad-hoc network for flexibility. By default we perform our experiments using the 1-Mbps link rate of 802.11b (experiments using 2-Mbps double application throughput as expected).

To experiment with different network conditions, we vary the distance and path between nodes: we fix the location of the SloMo node in a room, and move the laptop to various locations inside the building.

Downclocked Reception

We start by evaluating downclocked reception in isolation using compressive sensing. We transmit packets using the commercial WiFi device on the laptop to our experimental Sora node, which receives them using compressive sensing with a configurable decoding clock rate. For each clock rate and location, we transmit 1,000 User Datagram Protocol (UDP) packets (each 1,000-bytes long) paced to allow the network to settle between transmissions. We repeat each experiment ten times to account for variations. We perform the experiment across a wide range of clock rates, and in different locations that result in a variety of network conditions. In each case we record the fraction of transmitted packets successfully received and decoded using compressive sensing on the Sora node, and also report the corresponding SNR value for each location.

Figure 3.5 shows that downclocked reception operates nearly as well as standard WiFi across a wide range of decoding clock rates. Each point is the average of ten runs, and the error bars show the standard deviation. A clock rate of 100% corresponds to standard WiFi processing as the baseline, and smaller rates correspond to more aggressive use of compressive sensing with lower power requirements. When the SNR is good ($\geq$48 dB), packet reception using compressive sensing is nearly equivalent to standard WiFi, even for very low clock rates of 18–36%. Recall from Section 2.3 that downclocking at such rates corresponds to $\geq$ 40% savings in power consumption for a WiFi chipset.

Unfortunately, our ability to evaluate SloMo downclocked reception performance for a wider range of SNRs is limited by the Sora platform. We observe that Sora has a rather narrow dynamic range in terms of receiver sensitivity and exhibits a sharp cut-off behavior when the SNR is around 46 dB, likely due to the lack of hardware automatic gain control. While operating in this regime, Sora's standard WiFi implementation only achieves a 53% reception rate, and compressive sensing delivers 73% of that performance at the lowest clock rate.

Downclocked Transmission

Next we evaluate downclocked transmission in isolation using the shorter "Barker-like" sequences. We send packets from our experimental Sora node using downclocked transmission to the commercial WiFi device on the laptop, and record the fraction of transmitted packets successfully received and decoded by the commercial device. We use

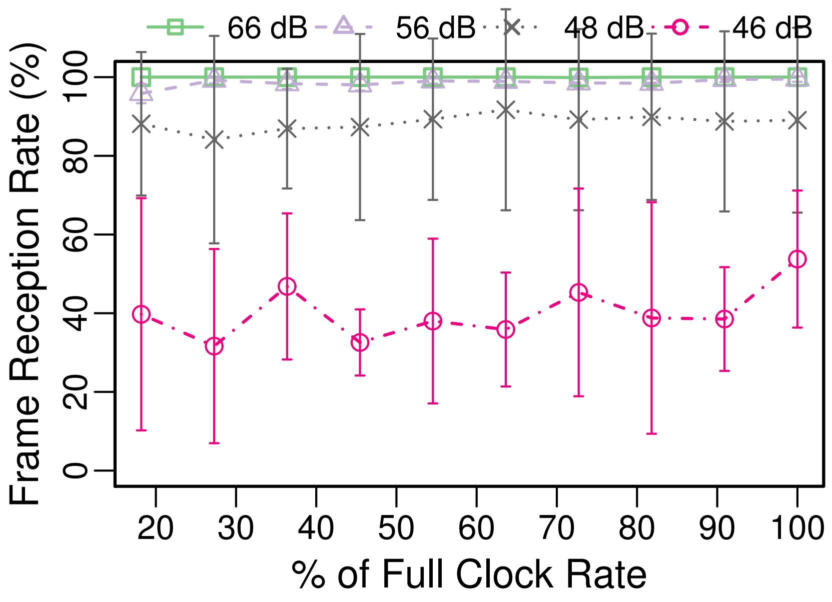
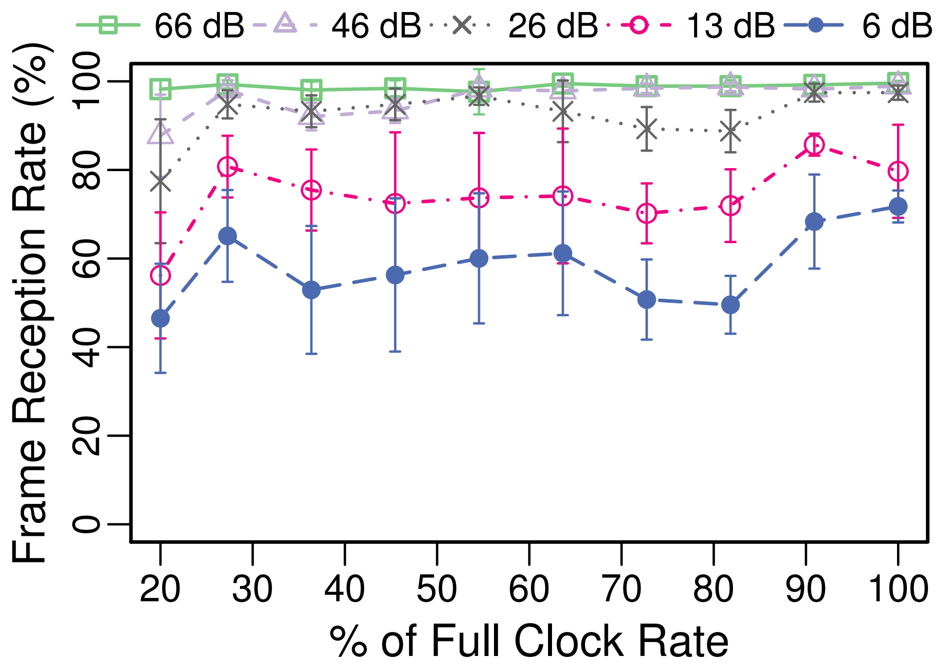


Figure 3.5. Frame reception rates using downclocked compressive sensing at SloMo node for packets sent by commercial WiFi device. As a baseline, the 100% clock rate corresponds to using the default 802.11b implementation.

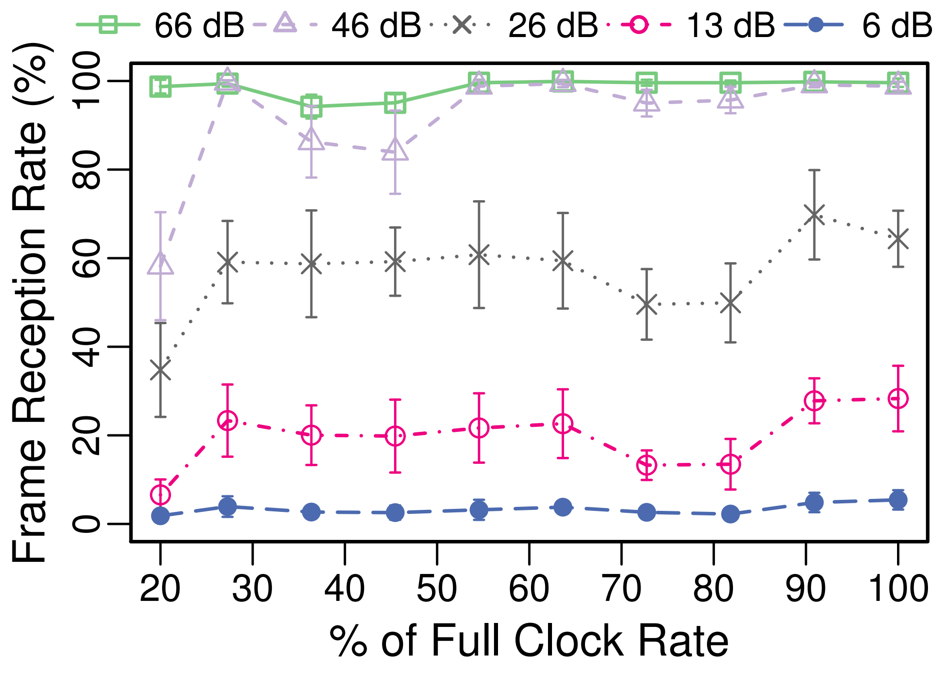
the same methodology as with compressive sensing: ten runs of 1,000 UDP packets at each combination of downclock rate and network location. We also experiment with two packet sizes. The first is a small packet size of 60 bytes, corresponding to apps sending small data packets and sending ACKs in response to a packet received using compressive sensing. The second is a larger packet size of 1,000 bytes.

Figures 3.6a and 3.6b show the results for downclocked transmission for small and large packets, respectively. Compared to downclocked reception with compressive sensing, we note that the operational SNR range is much larger; commercial WiFi cards have much better receiver sensitivity than Sora.

Focusing on results relative to the commercial WiFi baseline, however, shows that downclocked transmission using shorter “Barker-like” sequences more strongly depends on network conditions, clock rate and packet size. A clock rate of 100% transmits using the full Barker sequence in standard WiFi, and smaller rates correspond to transmission



(a) 60-bytes packets



(b) 1,000-bytes packets

Figure 3.6. Frame reception rates at commercial WiFi device for packets sent by SloMo node using downclocked Barker-like transmission. As a baseline, the 100% clock rate corresponds to using the default 802.11b implementation.

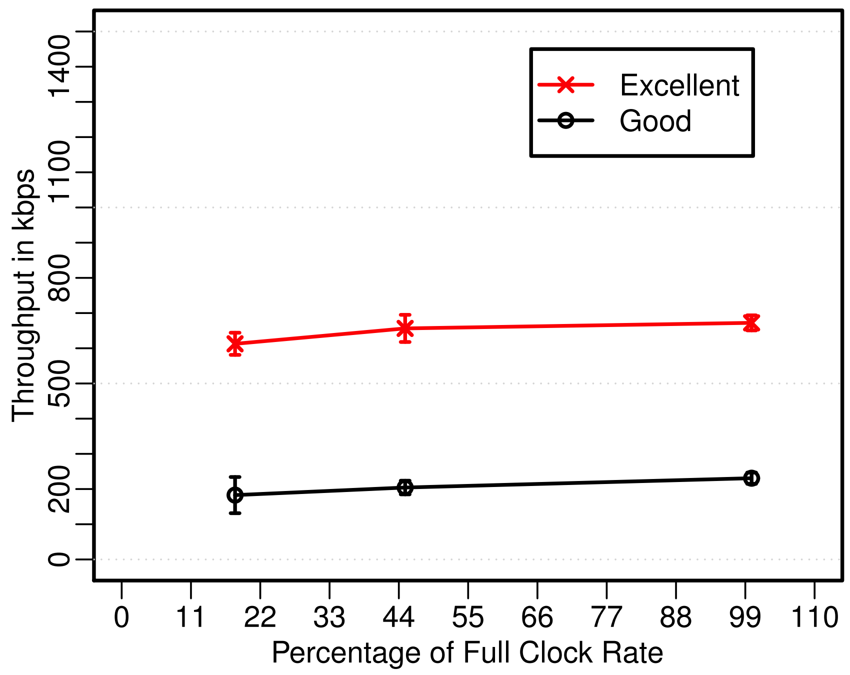
using increasingly shorter Barker-like sequences (Figure 3.3); the lowest transmission clock rate is 20%, which corresponds to transmitting with just two chips (Section 3.2). As shown in Figure 3.6a, with small packet sizes downclocked transmission is nearly as good as standard WiFi for moderate and good network conditions ($\geq$ 26 dB) for nearly all downclock rates (at the lowest 20% clock rate, reception rates are 10–20% below the baseline). With larger packets sizes, as shown in Figure 3.6b, downclocked transmission continues to do well for the majority of clock rates.

Note that downclocked rates of 73% and 82% underperform other clock rates by 7–10% when the SNR is moderate or low ($\leq$26 dB). This variation is due to how well a "Barker-like" sequence approximates the original Barker sequence; a longer sequence (higher clock rate) does not necessarily yield better correlation results. As with small packets, the lowest clock rate of 20% substantially degrades reception relative to the baseline, pushing the limit of downclocking.

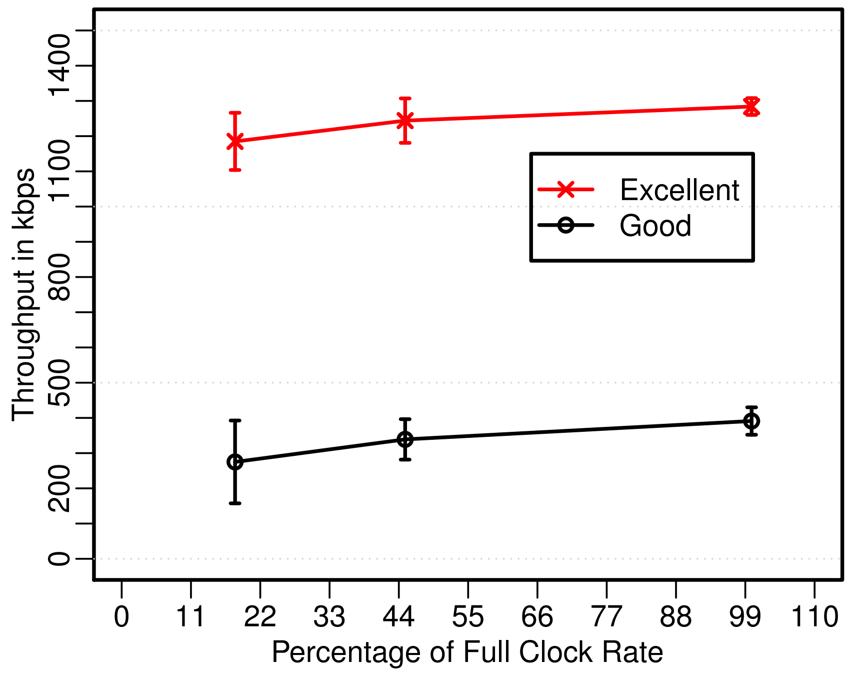
Overall, when SNR is poor ($\leq$ 13 dB), downclocked reception rates are on average 10% less than the standard WiFi implementation; otherwise, the packet reception rates are approximately the same. These results indicate that downclocked transmission is feasible for a wide range of SNR scenarios, especially transmitting ACKs at the same downclocked rate used to receive data frames.

Transport Layer Throughput

To stress SloMo's downclocking implementation, we next evaluate application throughput at both 1-Mbps and 2-Mbps link rates. We use `iperf` to transmit 1,000-byte UDP packets as fast as possible between the laptop and Sora node, repeating the application ten times. We perform the experiment at three clock rates, including 100% as the standard WiFi baseline, and at two locations where the SNR condition is excellent and good, respectively.



(a) 1 Mbps



(b) 2 Mbps

Figure 3.7. UDP throughput using `iperf` sending packets from the commercial WiFi device to the SloMo Sora node using downclocked reception with compressive sensing.

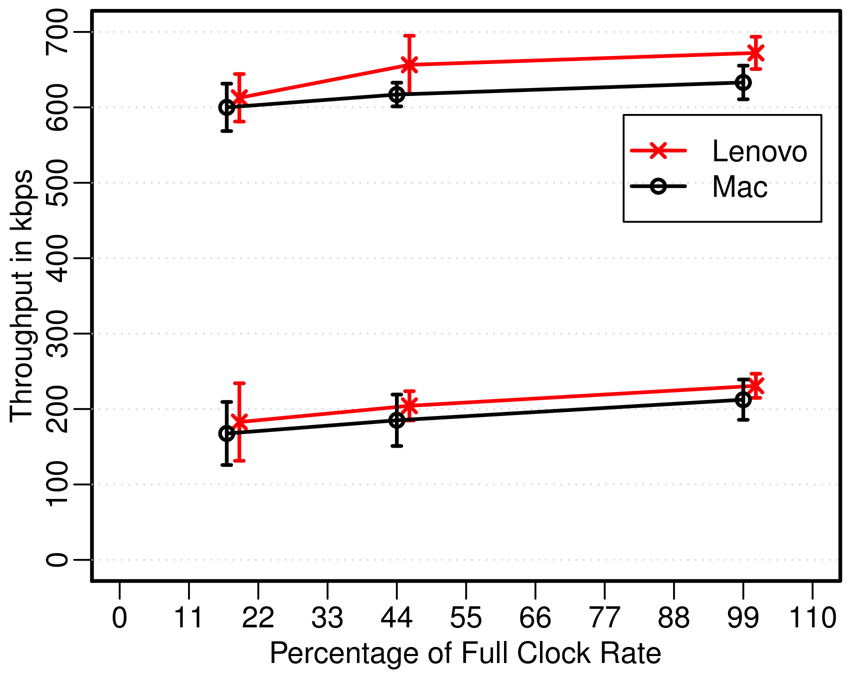


Figure 3.8. Comparing UDP throughput when sending from two different commercial WiFi devices to the SloMo Sora node.

Figures 3.7a and 3.7b show the `iperf` throughput results for link rates of 1 Mbps and 2 Mbps, respectively. These results evaluate downclocked reception using compressive sensing: `iperf` runs on the laptop with the commercial WiFi card and sends UDP packets to the SloMo Sora node, which receives the packets at downclocked rates. For the "good" and "excellent" network conditions, application throughput at the downclocked rates is nearly equivalent to throughput using standard WiFi. Figure 3.7b shows that downclocked reception indeed can take full advantage of 2-Mbps link rates under reasonable network conditions. For "good" and "excellent", the application throughputs at 2 Mbps are double those at 1 Mbps.

Hardware Heterogeneity

To test the sensitivity of SloMo to specific commercial WiFi implementations, we performed similar throughput experiments between the Sora node and a Macbook

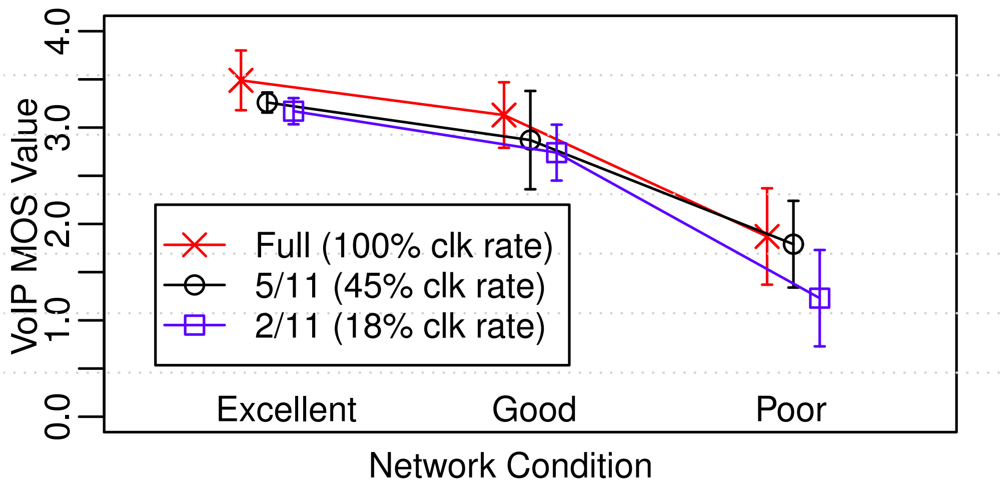


Figure 3.9. The Mean Opinion Score (MOS) for VoIP calls using Skype at different downclocked rates and network conditions. The SloMo node uses both downclocked reception and transmission.

Pro laptop with an Apple Airport Extreme WiFi card using the Broadcom BCM43xx firmware. As above, we used `iperf` to send 1,000-byte UDP packets as fast as possible for combinations of downclock rates and network conditions, repeating each run ten times. We found that both downclocked reception and transmission operated as expected between SloMo and both commercial WiFi devices on the laptops. Figure 3.8 shows the `iperf` results for downclocked reception. Although application throughput with the Intel card on the Lenovo was consistently 5–10% higher than with the Airport Extreme on the Macbook, this difference was independent of whether we were using normal or downclocked communication, clock rate, and network location. We also performed the same experiment in the reverse direction—using `iperf` to send packets from the SloMo Sora node using downclocked transmission to the laptops—and experienced the same behavior.

Skype Application

Finally, we combine downclocked reception and transmission in an application-level experiment that evaluates the quality of Skype VoIP communication using SloMo. The Lenovo laptop communicates directly with the SloMo Sora node, which acts as a bridge to our LAN. One Skype user is on the Lenovo laptop, and the second is on a Macbook laptop connected to the LAN such that the Skype connection flows through the SloMo Sora node. The SloMo Sora node receives Skype data and ACK packets from the Lenovo laptop using downclocked compressive sensing, and transmits ACKs and data packets to the laptop using the downclocked short "Barker-like" sequences.

We make Skype calls between the two users in the three network locations and at three clock rates: standard WiFi (100%), 5/11 (45%), and 2/11 (20%). For each location and clock rate combination, we make ten calls lasting between 25–30 seconds and record packet traces for the connections. With the packet traces, we use the methodology described by Verkaik *et al.* [63] to calculate the Mean Opinion Score (MOS) for each call, a standard metric for scoring call quality that ranges from 5 to 1 (good to bad). Similar as Verkaik *et al.* [63], we calculate a version of MOS based on network-level characteristics (jitter, packet loss, etc.) derived from packet traces [18] that are further calibrated using a model tuned to the particular codec used [21].

Figure 3.9 shows that downclocked communication using SloMo only significantly degrades call quality when network conditions are poor, as expected. The points show the average MOS score across calls made in each configuration, with the standard deviation as error bars. The full-clock-rate curve corresponds to standard WiFi, and the other curves are for downclocked SloMo. When network conditions are "good" and "excellent", VoIP calls even using an 18% downclock rate achieve MOS scores that are at least 90% of the MOS scores using standard WiFi. When network conditions are poor, calls at the 45% downclock rate are still comparable (MOS scores are 96% of standard WiFi), but the quality of calls at the 18% downclock rate suffers: the already low MOS scores are just 66% of standard WiFi. As with previous experiments, when network conditions are very poor only moderate downclocking works well.

3.7 Summary

Leveraging the inherent information sparsity in DSSS modulation, we illustrate how to design and implement a compressive sensing based 802.11b receiver that decodes standard WiFi frames sent at 1 or 2 Mbps. In addition, we notice that there exists large bit-decision headroom for WiFi (DSSS) frame reception on commercial WiFi devices. Based on this observation, we propose “Barker-like” sequences with chip rates that are much smaller than the standard Barker sequence, which essentially enable downclocked WiFi transmission. We have also carefully addressed the intricacies of making a SloMo receiver practical, i.e., fixed-length integrate-and-dump, synchronization, network protocol interactions, etc. We then implement our design on a software-defined radio platform and demonstrate that SloMo can seamlessly communicate with existing commercial WiFi chipsets. Under moderate SNR conditions, the performance of SloMo is almost identical to a standard 802.11b transceiver. However, the SloMo system is limited to 1 and 2 Mbps data rates, which may not be sufficient to support bandwidth hungry applications. In the next chapter, we discuss how our WiFi downclocking design can be extended to higher data rates.

Chapter 3, in part, is a reprint of the material as it appears in Proceedings of the USENIX Symposium on Networked Systems Design and Implementation 2013. Lu, Feng; Voelker, Geoffrey M.; Snoeren, Alex C. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Downclocking 802.11a/g

The previous chapter showed that downclocked 802.11b reception and transmission using compressive sensing and Barker-like sequences are both practical and useful. However, the supported data rates are somewhat limited with these approaches. In this chapter, we show how to extend our downclocking design to higher data rates in WiFi with Enfold, in particular orthogonal frequency division multiplexing (OFDM) modulated WiFi frame reception. First we describe the fundamental underpinnings of the Enfold design which exploits the aliasing effect that is inherent in under-sampling an OFDM signal. We then survey the practical challenges of implementing a real downclocked OFDM WiFi receiver, such as synchronization, frequency offset and phase recovery, and describe techniques that address the various challenges in addition to decoding. We next present an implementation of our Enfold design on the Microsoft Sora platform and the modifications needed to support downclocked OFDM reception on existing infrastructure. Finally, we evaluate the Enfold system both in simulation and in practice with real WiFi devices, and demonstrate that Enfold can attain greater than 96% and 83% packet reception rates for moderate signal-to-noise ratio (SNR) while reducing the clock rate by two times and four times, respectively.

4.1 Harnessing Aliasing

Unlike the direct sequence spread spectrum (DSSS) modulation scheme used in 802.11b, OFDM signals are extremely dense as data bits are encoded on all subcarriers across the entire frequency spectrum. Therefore, compressive sensing techniques are no longer applicable in downclocked OFDM reception. When the sampling rate is below the Nyquist rate, aliasing occurs, where the signal spectrum folds up and the transmitted signal is no longer recoverable. However, we observe that there exists well-defined structure in terms of how the signal spectrum folds up under aliasing. Based on the design of OFDM communication systems, we show how the structure can be exploited to enable downclocked operation.

4.1.1 Downclocked OFDM Modulation

At a high level, an OFDM communication system is defined by the inverse fast Fourier transform (IFFT) and fast Fourier transform (FFT) operations. In the context of 802.11, this implies the following: a) data bits are modulated and coded on a number of subcarriers and inverse FFTs are taken over these coded values to produce the time-domain signal at the transmitter; and b) the time-domain signal is sampled at the receiver, which then performs FFTs over these data samples to recover the encoded values. To convey the essence of our idea, we defer details related to timing, frequency synchronization, channel estimation, etc., to later sections, and focus for the moment entirely on modulation.

Here, we consider a single OFDM symbol and denote the data encoded on each subcarrier as C_i, and channel impulse response for each subcarrier as H_i, where $0 \leq i \leq 63$ in the case of 802.11a/g. Thus, the sampled signal at the receiver, when expressed in the frequency domain, will be C_iH_i for subcarrier i. Correspondingly, the 64 time-domain data samples at the receiver can be written as:

$$D_k = \frac{1}{64}\sum_{i=0}^{63} C_i H_i e^{j2\pi ki/64} + W_k, \quad k = 0,\ldots,63. \tag{4.1}$$

where W_k is the channel noise. To recover the transmitted data, the receiver performs FFT on the 64 data samples, D_k, and the resulting frequency response is expressed as:

$$F_l = \sum_{k=0}^{63} D_k e^{-j2\pi kl/64}, \quad l = 0,\ldots,63. \tag{4.2}$$

Now let us substitute Eq. (4.1) to Eq. (4.2):

$$\begin{aligned} F_l &= \sum_{k=0}^{63}\left(\frac{1}{64}\sum_{i=0}^{63} C_i H_i e^{j2\pi ki/64} + W_k\right) e^{-j2\pi kl/64} \\ &= \frac{1}{64}\sum_{i=0}^{63}\sum_{k=0}^{63} C_i H_i e^{j2\pi k(i-l)/64} + N_l \\ &= C_l H_l + N_l. \end{aligned}$$

where $N_l = \sum_{k=0}^{63} W_k e^{-j2\pi kl/64}$ is the channel noise expressed in the frequency domain at subcarrier l, which results in a non-zero decoding error. Subsequently, the transmitted data is estimated as F_l/H_l.

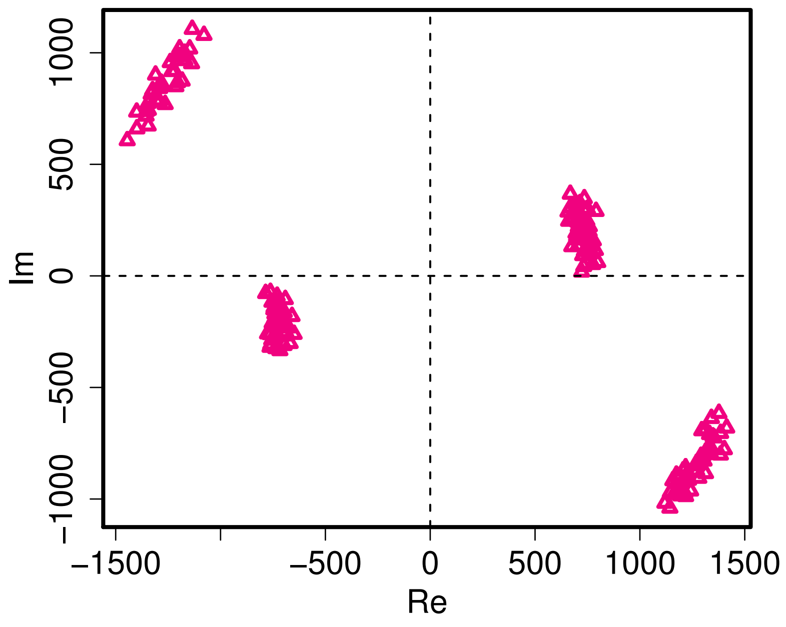
Now let us suppose the incoming data is sampled at a 50% clock rate. 32 samples will be produced, i.e., $D_{2k}(k = 0,\ldots,31)$. Correspondingly we perform a 32-point FFT operation on these data samples to yield the following frequency response:

$$\begin{aligned}
\tilde{F}_l &= \sum_{k=0}^{31} D_{2k} e^{-j2\pi kl/32} \\
&= \sum_{k=0}^{31} (\frac{1}{64} \sum_{i=0}^{63} C_i H_i e^{j2\pi 2ki/64} + W_{2k}) e^{-j2\pi kl/32} \\
&= \frac{1}{64} \sum_{i=0}^{63} \sum_{k=0}^{31} C_i H_i e^{j2\pi k(i-l)/32} + \tilde{N}_l \\
&= 0.5 * (C_l H_l + C_{l+32} H_{l+32}) + \tilde{N}_l.
\end{aligned} \tag{4.3}$$

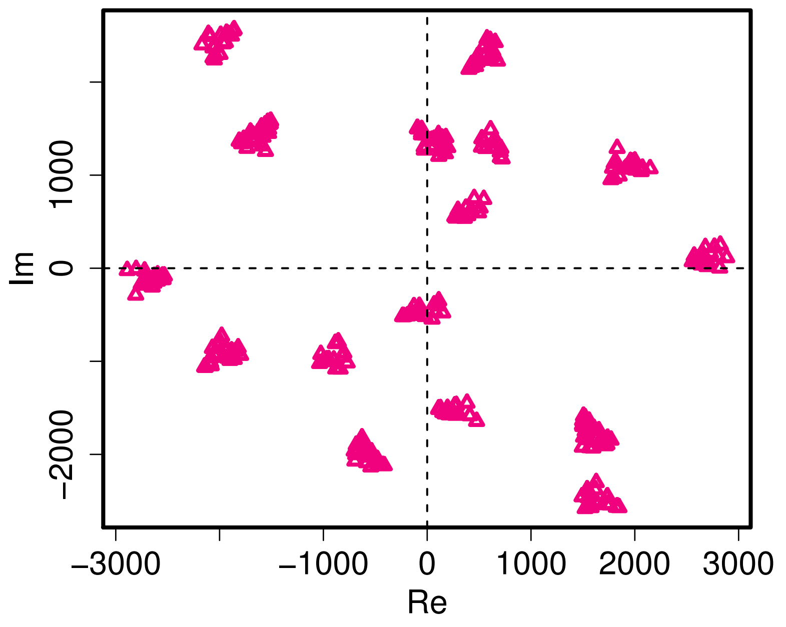
Notice that $\sum_{k=0}^{31} C_i H_i e^{j2\pi k(i-l)/32}$ equals to 0 if $(i-l)/32$ is not an integer. We refer to 0.5 in Eq. (4.3) as the downclocking scaling coefficient. One way to interpret the aliasing effect given in Eq. (4.3) is that the frequency response for subcarrier l is the average of the responses for subcarrier l and $l+32$ (if they were sampled at full rate), for $l = 0, \ldots, 31$. Similarly, the frequency response for subcarrier l at a 25% clock rate would be the sum of the responses for subcarrier l, $l+16$, $l+32$, and $l+48$ (should the full sampling rate have been applied), scaled by the downclocking coefficient, 0.25.

Hence, aliasing effectively transforms the original n-QAM (quadrature amplitude modulation) system to n^2-QAM at the receiver side for a 50% clock rate. For example, if BPSK (binary phase shift keying) is used at the sender, the resulting modulation due to aliasing would be 4-QAM. Similarly when downsampling at 25% clock rate, n-QAM transmissions will be transformed into n^4-QAM. Figure 4.1 shows the constellation diagram for a subcarrier of two real WiFi packets when received at 50% and 25% clock rates, respectively, which serves as a concrete example of how the modulation scheme is transformed.

In sum, when aliasing happens the signal spectrum folds up and superimposes on itself in a well-defined manner, thereby enabling a systematic decoding of the compounded frequency responses.



(a) 50% downclocking.



(b) 25% downclocking.

Figure 4.1. Frequency response (constellation diagrams) on subcarrier 6 for two real WiFi packets under 50 and 25% downclocking. Notice how the constellation diagram changes as clock rate decreases.

4.1.2 Decoding Aliasing Induced Modulation

We now explain how to decode the original data bits. We choose 802.11a/g 6-Mbps transmissions and a 50% reception clock rate as an example to illustrate our idea with an understanding that the approach holds for other rates and downclocking options. At 6 Mbps, a group of 48 bits are mapped into 48 complex numbers during the modulation mapping, where each bit is mapped to either $1+0j$ or $-1+0j$ using BPSK modulation. These 48 complex numbers are then encoded on 48 out of the 64 subcarriers. Recall from Eq. (4.3) that the frequency response is the sum of $C_l H_l$ and $C_{l+32} H_{l+32}$ for subcarrier l at the receiver side. Given the values of C_l the frequency response F_l takes one of the four values, $\frac{1}{2}(\pm H_l \pm H_{l+32})$, depending on the bits mapped to subcarrier l and $l+32$ on the sender side (e.g., 00, 01, 10, and 11). This encoding functions identically to 4-QAM: two bits are mapped to one of four complex numbers.

Therefore, similar to how a QAM system is decoded, we employ minimum distance decoding to recover the original bits. However, since the modulation transformation is induced by aliasing, and the channel response for subcarriers is random,[1] the resulting 4-QAM modulation differs from a standard one. In a standard 4-QAM scheme, two bits are mapped to one of the four complex numbers (also known as constellation points) equispaced around a circle. On the other hand, with aliasing induced modulation, the four code points could be anywhere in the constellation diagram depending on the channel response. Their separation depends on the relationship between the channel responses. In the absolute worse case where $H_l = H_{l+32}$, two constellation points collapse into one, which leads to at least a 25% bit error rate (BER). We observe, however, that this devastating scenario is unlikely in practice (see Section 4.5.2). Similarly, when a 25% clock rate is used, the original BPSK modulation becomes 16-QAM modulation. Aliasing

[1]Subcarrier response is not entirely independent, but we defer a careful study of the impact of this phenomena to future work.

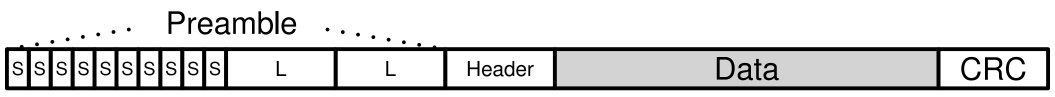


Figure 4.2. WiFi (802.11a/g) packet structure.

induced modulation can be extended to other rates (e.g., 12 Mbps and higher) as well; the resulting modulation scheme would simply yield more densely packed constellation points compared to the original scheme.

4.2 WiFi Reception

Recent WiFi generations (e.g., 802.11a/g/n/ac) are all OFDM-based communication systems. We showed in the previous section that it is entirely possible to decode OFDM signals by exploiting the frequency spectrum structure under aliasing. However, real communication systems are much more complex than what we have discussed so far and there are many other components that need to be in place before a receiver is in a position to decode an actual OFDM symbol.

The designers of WiFi obviously never anticipated it would be used by downclocked radios, so existing approaches to clock synchronization, frequency compensation, pilot tracking, etc., might not function when downclocked. We start in this section with a brief overview of the 802.11 a/g frame structure and the typical decoding pipeline of a standard, full clock rate 802.11a/g-based WiFi receiver, and then address the challenges imposed by downclocked operation in the following section.

4.2.1 Frame Structure

Two key tasks for any OFDM communication system are timing synchronization and frequency offset compensation. To meet these requirements, a WiFi frame is prepended by a preamble, which serves the purpose of preparing the receiver to decode the actual data part. Figure 4.2 shows the detailed structure of an 802.11a/g WiFi frame. The preamble consists of ten identical “short” OFDM symbols and two identical “long” OFDM symbols, with durations $0.8\mu s$ and $4\mu s$, and sequence lengths 16 and 64 (80 including the cyclic prefix), respectively. We explain below how a receiver uses the repetitive structure of the preamble to perform both time and frequency synchronization.

Immediately following the preamble is the physical layer (PHY) header, which contains information such as the data encoding rate, payload length, service field, etc. The first 24 bits of the PHY header are encoded at 6 Mbps while the remaining 16 bits are encoded at the same rate as the actual data; depending on the current channel condition, the data may be transmitted at different modulation rates. In most WiFi devices, the PHY header (first 24 bits) is added by the hardware without the driver’s intervention.

4.2.2 Reception Pipeline

In addition to decoding, a receiver performs five important steps to correctly recover the transmitted signal.

Timing synchronization. A WiFi receiver will continuously sample the channel to look for a preamble. The ten short OFDM symbols in the preamble help the receiver to lock onto the data stream and define the OFDM symbol boundary. Once the symbol boundary is determined, FFT can be performed on the time-domain samples to obtain their corresponding frequency-domain values.

Frequency compensation. Before computing the FFT, however, since the clock oscillator frequency is never exactly the same at the sender and receiver, the frequency offset needs to be estimated and compensated. Otherwise, orthogonality among sub-carriers will be compromised and data transmitted on one subcarrier would interfere with another. Hence, receivers also use both the short and long OFDM symbols in the preamble to perform frequency offset estimation (or frequency synchronization).

Channel estimation. Once both timing and frequency are synchronized, the next decoding stage estimates the wireless channel response, i.e., channel estimation. With the estimated channel response, the results of computing the FFT on the time-domain samples can be mapped to the actual data bits transmitted.

Viterbi decoding. To compensate and correct for bit errors, the raw decoded bits are fed into a Viterbi decoder [65], whose output is de-scrambled and subsequently passed on. (Obviously, symmetric operations are performed at the sender before transmission.)

Phase compensation. There is one additional step that is performed throughout the entire data decoding process: phase compensation. Since the sampling clock could drift slowly as time goes by, FFT results might vary even when the same OFDM symbol is transmitted repeatedly. Therefore, phase compensation is used to correct the variation of FFT results across OFDM symbols.

Viterbi decoding, scrambling, and subsequent stages work on the decoded data bits and are therefore entirely agnostic to the underlying clock and sampling rate. The remaining stages of the pipeline—timing synchronization, frequency compensation, channel estimation, and phase compensation—operate on the time-domain data samples, and are impacted by operating at lower clock rates.

4.3 Downclocking WiFi

In this section, we systematically examine the impact of downclocking in detail and propose a series of techniques to address these challenges.

4.3.1 Timing Synchronization

The purpose of timing synchronization is to enable the receiver to first detect the presence of a valid WiFi packet and then correctly locate the boundary between adjacent OFDM symbols. Timing synchronization uses the short preamble only, i.e., the ten repetitive short OFDM symbols. The IEEE 802.11 specification [33] does not mandate any particular timing synchronization algorithm. Although there are many timing synchronization algorithms for OFDM, we, like many others, use the auto(cross)-correlation approach [41, 60, 69].

The short OFDM symbol has a sequence length of 16 and exhibits very good cross-correlation properties. Let us denote the incoming time-domain data samples as $r(.)$. Then the cross-correlation is computed as:

$$cros(i) = \frac{\sum_{j=0}^{15} r(i+j)S^*(j)}{\sum_{j=0}^{15} |S(j)|^2}, \tag{4.4}$$

where $S^*(.)$ is the complex conjugate of the short OFDM sequence expressed in the time domain. If sampling at full clock rate, there will be 16 time-domain samples corresponding to one short OFDM symbol, and $S(.)$ has the same length. The correlation output, $cros(i)$, produces a peak only when the incoming signal $r(.)$ fully aligns with $S(.)$, i.e., $r(i+j) \approx S(j)$ (the approximation is due to the presence of noise). Otherwise, the magnitude of the correlation output is much smaller than the peak value. Thus, the cross-correlation peak enables accurate OFDM symbol boundary detection (i.e., fine timing synchronization).

Due to the repetitive structure of the short preamble, a receiver can correlate the received data samples with themselves at one short OFDM symbol delay. Specifically, the receiver tracks the following auto-correlation output:

$$auto(i) = \sum_{j=0}^{15} r(i+j)r^*(i+j+16). \tag{4.5}$$

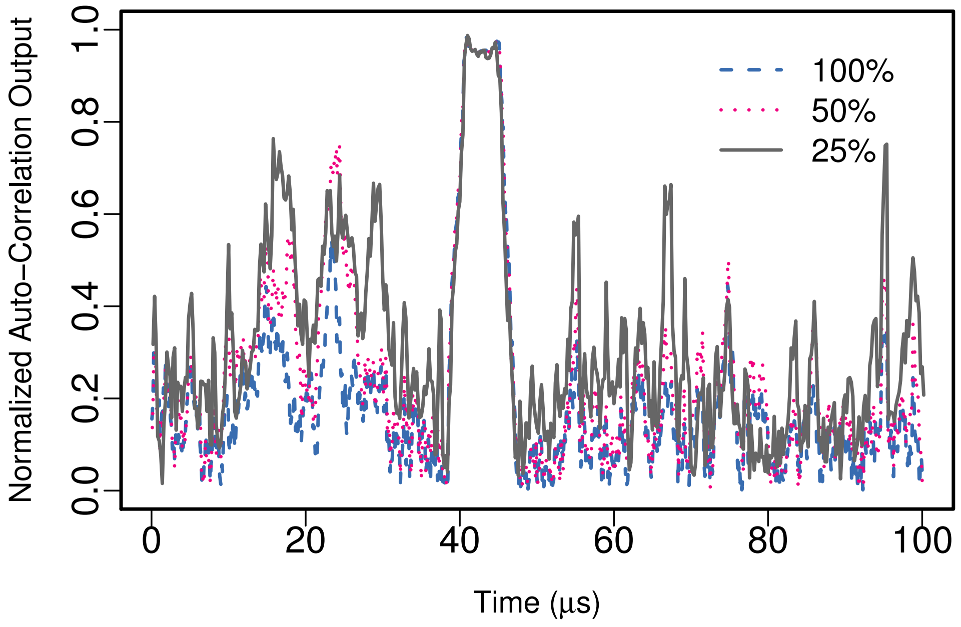
The rational is that, when a valid WiFi frame is present, $r(i) = r(i+16)$ as 16 samples correspond to one short OFDM symbol duration. When the auto-correlation index i

is within the range of the short preamble, the output from Eq. (4.5) spikes and stays high. This auto-correlation result produces a plateau over time, which is known as coarse timing synchronization. Clearly, the cross-correlation peak defined in Eq. (4.4) must happen within the plateau to be meaningful. Thus, by combining the outputs of both results, we can safely detect the presence of a WiFi frame and correctly identify symbol boundaries.

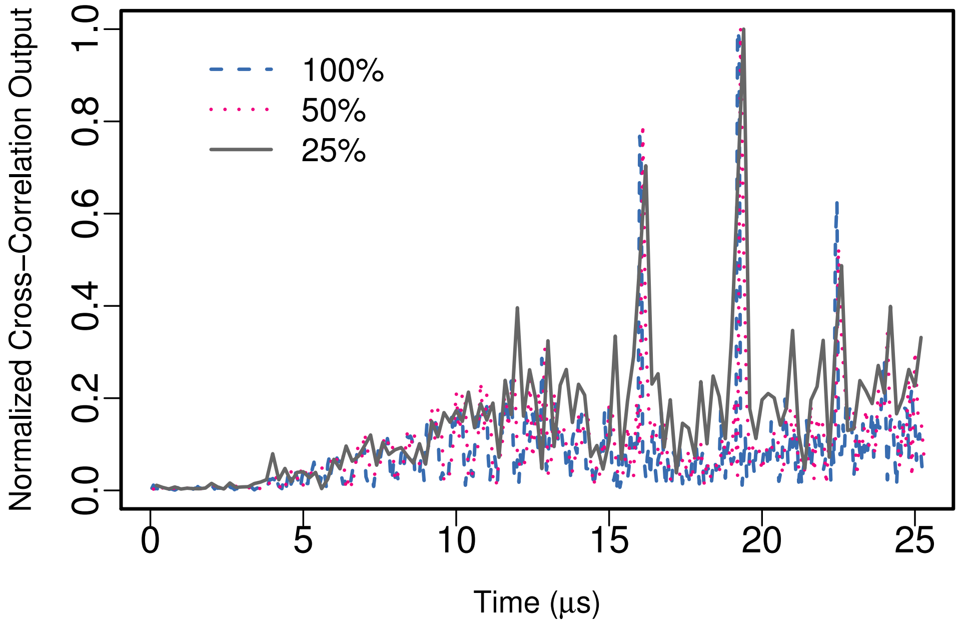
Downclocking reduces the number of time-domain samples by half (or a quarter) depending on the clock rate. First we note that, regardless of the sampling rate, Eq. (4.5) will hold when $r(i) = (i+8)$ or $r(i) = (i+4)$ if sampling at 50% or 25% clock rates, although the noise floor (when the auto-correlation is failing within the short preamble) approaches the plateau. Nevertheless, the plateau could still have separation and, perhaps more importantly, its width roughly equals the duration of the short preamble, which is invariant among different clock rates (see Figure 4.3a).

Unfortunately for cross-correlation, the downsampled sub-sequences no longer exhibit convenient correlation properties: there could be multiple comparable peaks within one short OFDM symbol. So we turn to the long OFDM symbols and evaluate its cross-correlation properties. It turns out that the long preamble sequence (consisting of 128 samples across two symbols) also exhibits excellent cross-correlation properties, in part because its time-domain samples are generated similarly to the short preamble.

Figure 4.3b illustrates the effect of using the 128-sample long OFDM symbols for timing synchronization on an actual WiFi packet. The cross-correlation output produces three peaks. The middle peak corresponds to full alignment, while the first corresponds to a partial alignment with the first OFDM symbol and the third corresponds to a partial alignment with the second OFDM symbol. These correlation peaks make it straightforward to identify the starting sample index for the remaining OFDM symbols. We also note that the peaks for different clock rates are superimposed on each other,



(a) Auto-correlation on short sequence.



(b) Cross-correlation on long sequence.

Figure 4.3. Correlation responses at 100%, 50% and 25% clock rates for raw data samples of a real WiFi packet captured by Universal Software Radio Peripheral (USRP).

confirming that the approach works for all clock rates. In Section 4.5.1, we show in practice that timing synchronization works quite well even for a 25% clock rate at low signal-to-noise ratio (SNR) values.

4.3.2 Frequency Compensation

Standard frequency compensation algorithms explore the following relationship between two transmitted back-to-back identical OFDM symbols. Let $r(1),\ldots,r(n)$ and $r(n+1),\ldots,r(2n)$ correspond to the time-domain samples at the receiver for two identical OFDM symbols, respectively. Then the following property always holds for any $1 \leq i \leq n$:

$$r(n+i) = e^{j2\pi\Delta f_c T} r(i), \tag{4.6}$$

where Δf_c is the frequency offset and T is the OFDM symbol duration. Hence the frequency offset is estimated as the ratio between $r(n+i)$ and $r(i)$, taking the average for all possible values of i.

With downclocking, the relationship described by Eq. (4.6) still holds for data samples spaced by one symbol duration, although the average is now taken over fewer samples. Since downclocking will only use half or a quarter of the samples compared to the full clock rate, the frequency offset estimation accuracy is reduced by 3 dB and 6 dB, respectively. As discussed in the previous section, the preamble consists of ten repetitive short OFDM symbols and two repetitive long OFDM symbols. When downclocking, we use both for frequency offset estimation (i.e., coarse and fine frequency synchronization), taking the average over fewer data samples.

As with timing synchronization, in Section 4.5.1 we show that our frequency compensation technique also works well in practice.

4.3.3 Channel Estimation

Once timing and frequency synchronization are achieved, the next step is to understand the channel response. In a WiFi frame, the long preamble supports channel estimation because the data bits encoded on each subcarrier are known *a priori*. Let us assume that the FFT results for the long OFDM symbol are $F_0, F_1, \ldots, F_{63}$; the wireless channel response is estimated as

$$H_i = F_i / C_i, \;\; 0 \leq i \leq 63. \tag{4.7}$$

where H_i is the channel response for subcarrier i and C_i is the data encoded on subcarrier i (known to both sender and receiver). However, when downclocking, the channel response will fold up and estimating individual subcarrier channel responses is no longer feasible. While there are approaches in the literature that determine channel responses under such conditions [39, 61] using compressive sensing, they make certain assumptions regarding the channel. Unfortunately, these assumptions are not necessarily true in general for the WiFi environment. Hence, we develop an alternative method to find the per-subcarrier channel response (coefficients).

The aliasing effect, when translated into the frequency domain, is equivalent to summing the channel coefficients at indices spaced by 32 or 16, depending on the downclocking rate. More specifically, when sampling at full clock rate, FFT is taken over 64 time-domain samples and produces 64 subcarrier responses, which are used to estimate channel response as defined by Eq. (4.7). Now when sampled at 50(25)% clock rate, we obtain $32(16)$ equations after performing FFT, where each equation consists of two(four) unknowns (see Eq. (4.3)). We leverage the fact that the WiFi channel response remains relatively invariant over the course of a single packet. Therefore, if we transmit multiple known OFDM symbols (e.g., training symbols), we could potentially recover all the unknowns by collecting equations from multiple symbols. One caveat is that all the training OFDM symbols must be phase compensated.

4.3.4 Phase Compensation

To compensate for any possible phase shift across OFDM symbols, the WiFi standard inserts known data in certain subcarriers (pilots) for every OFDM symbol. By comparing the pilot subcarrier responses between symbols, the phase shift is estimated and the data subcarrier values are compensated accordingly. Unfortunately, due to the particular set of subcarriers selected to be pilots, when downclocking the pilot subcarriers fold on top of data subcarriers (which contain unknown data). Hence, downclocking cannot use the pilots to compensate the phase difference. We could redefine the pilot subcarriers such that their indices are spaced by 16 so that one pilot subcarrier will always add up with another pilot under downsampling rate of 50% or 25%. However, doing so would violate our goal of being entirely 802.11 standard compliant.

4.3.5 Data Precoding

Rather than change the WiFi frame structure or how the subcarriers are mapped, we elect to transmit additional symbols in the payload of the frame. At a high level, we re-encode the original data bits such that: 1) additional channel training symbols are introduced; 2) pilot structure is restored. It is non-trivial to realize both (1) and (2) for a number of reasons, however.

From Bits to Subcarriers

Up to now, our discussion has primarily centered around coded values on subcarriers. However, there are multiple stages such as scrambling, convolutional encoding, interleaving, and subcarrier mapping sitting between raw data bits and the values encoded on the subcarriers. In particular, the transmitter will first XOR the raw bits with scrambler bits, and then pass the XOR-ed bits to a convolutional encoder. The output of the convolutional encoder not only depends on the current bit but also the past bit history, where the history length depends on the size of the convolutional encoder register. In the context of 6-Mbps WiFi, the transmitter produces a block of 48 bits for every 24 scrambled input bits after the convolutional encoder, i.e., input bit k leads to output bits $2k$ and $2k+1$, where $0 \leq k \leq 23$. Finally, the transmitter generates a single OFDM symbol by performing interleaving, modulation mapping and subcarrier assignments on these 48 bits. For the generated OFDM symbol, there is a fixed one-to-one correspondence between the 48 bits and the 48 (out of 64) subcarriers. For example, bit 16 maps to subcarrier 39: the coded values on subcarrier 39 will be $1+0j(-1+0j)$ if bit 16 is 1(0).

Conversely, if we want to set the coded value on subcarrier 39 to be $1+0j$ (i.e., for use as a pilot), we need to ensure that bit 16 is 1. Since the convolutional output bit is produced by taking XOR operations among the current input bit and some past input bits, there always exists some input bit value (i.e., could be 0 or 1) such that the output bit has the desired value. Therefore, we try both values for input bit 8 and check which one produces a value 1 on output bit 16.

Pilot Restoration

For 802.11a/g, the four pilots are inserted at the following subcarriers: 7, 21, 43 and 57. Under downclocking, these pilots will fold up with various data subcarriers, which we coin *pilot-images*. For example, the pilot-image for subcarrier 7 is subcarrier 39 under 50% sampling rate. To restore these pilots with downclocking, the pilot-images have to be encoded with known values. To simplify our design, we set the data encoded on a pilot-image subcarrier to be the same as its corresponding pilot. To enforce the same data encoded on both pilot and pilot-image, we modify the input bit stream to the convolutional encoder in the following way.

Let us denote the original raw input data bits (excluding the additional training symbols) as $b_0, b_1, \ldots, b_n$ and the scrambler sequence output as $x_0, x_1, \ldots$. At 6 Mbps, the transmitter will map a group of 48 convolutional encoder output bits to one OFDM symbol (48 out of 64 subcarriers). We run the convolutional encoder as normal while the input bits are coming through. The transmitter XORs each incoming data bit b_i with some scrambler output x_j, and feeds the scrambled bit into the convolutional encoder and generates two output bits as a result. Whenever we are about to produce an output bit that is going to be mapped onto a pilot-image subcarrier, we "pause" the input bit stream to the convolutional encoder and instead insert a new bit c such that the value encoded on the pilot-image subcarrier is the same as the corresponding pilot value.

For example, if output bit 16 is about to be produced (mapped to subcarrier 39), we check the current value of pilot subcarrier 7. If subcarrier 7 has value $1+0j$, we need to ensure that output bit 16 is 1, and 0 otherwise. The newly inserted bit c is selected such that bit 16 yields the desired value. The two convolutional encoder output bits (generated by the same input bit) are never mapped onto pilot-image subcarriers at the same time, thus there always exists a bit c such that the pilot-image subcarrier possesses the same value as its respective pilot. We update the convolutional encoder with bit c.

Next, given the desired input bit c, we then XOR it with the current scrambler bit, say x_k, to produce the raw data bit $\tilde{c}$. Afterwards, x_k is discarded and the next raw data bit b_i is scrambled with x_{k+1} to produce the new input bit for the convolutional encoder, which resumes normal operation. Correspondingly, the precoded data will be: $\ldots, b_{i-1}, \tilde{c}, b_i, \ldots$. This process is repeated until all input bits are exhausted and $\tilde{c}$ is inserted for every pilot-image subcarrier encountered. Since the positions of the convolutional output bits that are mapped to pilot-images are fixed, the bit indices for inserted bits $\tilde{c}$ are known as well. Hence, the receiver can simply discard the inserted pilot bits $\tilde{c}$ to recover the original data.

Training Symbols

We describe how we generate the additional training symbols with reference to a 50% clock rate; the same strategy applies for a 25% clock rate. With 50% downclocking, the FFT is performed over 32 data samples for one OFDM symbol. Let us denote the data encoded on subcarrier i as C_i at the sender side and the channel response for subcarrier i as H_i ($0 \leq i \leq 63$); the frequency responses after FFT can be written as:

$$\tilde{F}_j = \frac{H_j}{2}C_j + \frac{H_{j+32}}{2}C_{j+32}, 0 \leq j \leq 31.$$

To recover the channel coefficients H_js, we need at least two such equations and thus two OFDM training symbols. Furthermore, the two equations must be independent from each other, i.e., (C_j, C_{j+32}) from symbol one and (C_j, C_{j+32}) from symbol two must form a full rank 2×2 matrix.

We first generate a random binary string of 48 bits (sufficient for two OFDM symbols) and precode these bits so that the pilots are restored. We then compute the rank of the resulting matrix (e.g., $[C_j, C_{j+32}]^{S_1}$, $[C_j, C_{j+32}]^{S_2}$) for each subcarrier, where S_i is the i^{th} OFDM symbol. If all the matrices have a rank two, we use S_1 and S_2 as the training symbols. If not, we repeat until they do. This offline process only needs to be executed once and the resulting training symbols are used for every precoded packet.

4.3.6 Additional Considerations

So far we have covered the main components in enabling downclocked OFDM frame reception. Here, we mention a few additional details that are important for a standards-compliant design.

Transmission. While our approach allows downclocked reception of OFDM symbols, we cannot transmit OFDM symbols while downclocked. There are two options.

First, some WiFi chipsets (e.g., MAXIM 2831 [70]) are able to switch their clocks within 9 μs (i.e., less than SIFS), which would allow the WiFi chipset to spend most of the time in downclocked mode and only switch back to full clock rate for transmission. Alternatively, we could transmit using 802.11b encodings while remaining downclocked as shown in Chapter 3. We experimentally verify that a WiFi device will accept 802.11b (DSSS) frames, e.g., layer-2 ACKs, when sending 802.11a/g (OFDM) frames. We use a commercial WiFi network interface controller (Intel 6200) to send OFDM frames to our prototype while the latter replies with DSSS frames. Both nodes are able to communicate with each other without any problem.

Scrambler Seed. Our design assumes that the scrambling sequence is known, which can be derived deterministically from the scrambler seed. Commercial network interface controllers (NICs) often use an internal linear feedback shift register (LFSR) to generate the scrambler seeds. If the LFSR design and its initial seed are exposed, we could potentially determine the current scrambler seed at any moment. Unfortunately, we are unable to find a specific NIC that currently does so. The closest that we have found is the Atheros 93xx series chipset, where the scrambler seed can be temporarily disabled by configuring the *MAC_PCU_DIAG_SW* register. When scrambling is disabled, we could implement a soft scrambler in the driver to avoid long runs of 1s and 0s. We believe there is no reason the NIC could not expose either the scrambler seed or the LFSR directly to the driver.

Extension to 802.11n/ac. Although our discussion focuses on 802.11a/g, our approach is generic to other OFDM communication systems. The newer WiFi specs such as 802.11n and 802.11ac are based on OFDM as well. A direct way of applying our solution to 802.11n/ac is to configure these systems to operate in single-input-single-output (SISO) mode [28], which is similar to the setup of 802.11a/g we described here. In fact, 802.11n/ac include a channel-sounding process to estimate channel response and could apply channel precoding for transmitted frames, leading to both simplified downclocked decoding at the receiver and better performance. We leave combining multiple-input-multiple-output (MIMO) and downclocking for future study.

Added Complexity. Both Enfold transmitters and receivers have additional tasks to perform when compared to regular nodes. However, most of these tasks employ standard communication algorithms such as minimum-distance decoding, correlation, etc. Therefore, the additional computation power incurred is negligible compared to the communication cost. Although downclocking does convert the modulation scheme into a denser one, the power required to decode does not depend significantly on constellation density in existing chipsets [28].

4.4 Implementation

We implement Enfold on the Microsoft Sora software-defined radio platform [60]. The modifications are standards compatible and do not require any sender hardware changes.

4.4.1 Sender

Since Sora uses a fixed scrambler sequence for all packets, i.e., a constant scrambler seed, we take advantage of this fixed sequence to precode data before it is transmitted by the NIC. Depending on the downclocked clock rate at the receiver (50% or 25%), we add two or eight channel training symbols as part of the encoded data payload.[2] The pilot restoration logic described in Section 4.3.5 consists of about 300 lines of code (LoC).

Since our modifications are restricted to the data portion only, the NIC generates a standard physical layer packet header and cyclic redundancy check (CRC). As a result, the OFDM symbols corresponding to the header and CRC cannot be phase compensated

[2]We tried other numbers of channel training symbols as well; two and eight yield a good balance between overhead and estimation accuracy.

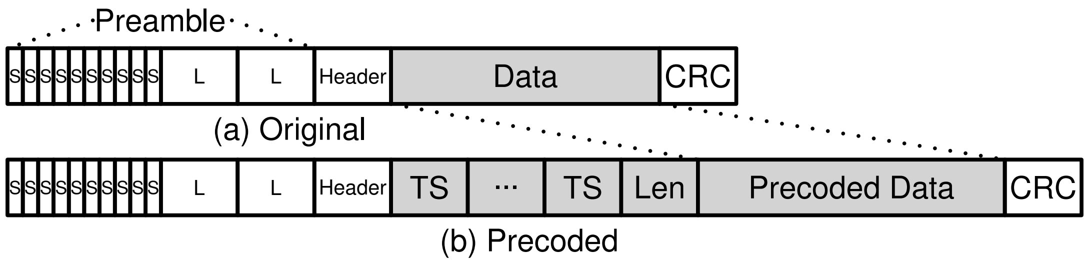


Figure 4.4. WiFi packet structure for both original and precoded versions. Note that our modifications are restricted to the data payload only (gray area).

by a downclocked device. Therefore, on the sender side we also include the original CRC (computed based on the uncoded data) as part of the precoded data. To ensure that the precoded data is an integral number of OFDM symbols—so that the entire coded data portion can be phase compensated—we add trailing zero bits at the end. As a result, we also include an additional OFDM symbol to encode the length of the original data so that the receiver knows when to stop decoding.

Figure 4.4 shows the structure of a precoded WiFi packet. At a 50% downclock rate, the new pilots are subcarriers 7 and 21, and the pilot-image subcarriers 39 and 53 are used to code pilot values. Similarly for 25% downclocking, the new pilots are 7 and 11, and the pilot-image subcarriers are 11, 23, 39, 55 and 59. As a result, the effective throughput for 50% and 25% downclocking are 91.6% and 79.1% of the original data rate (6 Mbps). In our experiments, two pilot subcarriers are sufficient to bound the phase estimation error within 0.12 radians (6.8 degrees).

Any non-Enfold node can receive and decode an Enfold-precoded WiFi packet; the data content would simply not be recognizable, similar to encrypted data.

4.4.2 Receiver

As before, we emulate downclocking in Sora by decimating every-other raw channel sample at for a 50% rate, and three-of-every-four samples for a 25% rate.

We modify and reimplement the timing and frequency synchronization modules to accommodate downclocked rates. We postpone decoding the physical layer header (one OFDM symbol), which contains the packet length and payload modulation scheme, until after the channel has been estimated.[3]

As the header symbol is not phase compensated, we exploit the fact that there are limited possibilities (4 and 16 possibilities for 50% and 25% downclocking, respectively) that a data subcarrier could be. Therefore, we use a data subcarrier as the pilot by enumerating all 4(16) possibilities (obtained from the training symbols) when decoding the header symbol. The header symbol contains sufficient information (e.g., parity and modulation) to enable Enfold to determine the correct output among all possibilities.

Once the header is decoded, the number of data OFDM symbols is known. For each incoming OFDM symbol, we first apply phase compensation and then use a minimum distance decoder to determine the data bits. These data bits go to the Viterbi decoding chain and are subsequently de-scrambled. Once all symbols are decoded, we start to decode the precoded data payload by essentially discarding the inserted pilot bits. Finally, we compare the computed CRC with the CRC included as part of the precoded data payload; we ignore the default media access control (MAC) layer CRC.

One subtlety not discussed earlier is the service field, which consists of 16 bits with the first 8 bits set to zero so that the receiver can determine the scrambler seed. In our implementation, we include the service field as part of the training symbol since the receiver knows the scrambler seed. In the general case where the scrambler seed is not known at the receiver, we can place a known bit pattern on the eight reserved bits and use the same approach for decoding the service field as the header symbol.

[3]Since our implementation currently assumes that the data payload is fixed at 6 Mbps, we do not rely on the packet header to determine the payload modulation scheme.

4.4.3 Network interactions

In this section, we discuss how Enfold nodes interact with the Access Point (AP) and share the network with regular nodes, in particular the modifications needed to support Enfold in existing deployed infrastructure.

Precoding at AP. Given the current pilot subcarrier placement, an Enfold-capable AP has to precode the data to restore pilot structure at downclocked receivers. To remain 802.11-standard compliant, we implement the precoding step in Sora's link-adaptation layer, leaving the lower MAC/PHY layer untouched. To verify that our implementation is truly standard compliant, we transmit a stream of User Datagram Protocol (UDP) packets with our prototype Enfold AP and use Wireshark on a Macbook Pro to capture (in promiscuous mode) the precoded WiFi packets and apply a corresponding decoding process on the dumped packet trace. All packets sent by the Enfold transmitter are correctly decoded, thus confirming the standard—or at least Apple—compatibility of Enfold.

Beacon and Scanning. Although Enfold-precoded packets can be correctly received by non-Enfold nodes, they would need to apply the corresponding decoding process to recover the original data. Therefore, broadcast packets, such as beacon frames, are unlikely to be understood by both Enfold and non-Enfold nodes simultaneously. Hence, to support Enfold-enabled nodes, an AP would need to broadcast two types of beacon frames, original and precoded. To reduce the network overhead due to two beacon frames, the AP could broadcast Enfold beacons at larger time interval (e.g., every few hundred milliseconds). The reduced beacon time is unlikely to impact the scanning and association process, given these happen on the order of several seconds. For traffic indication purposes during power save mode (PSM), since Enfold nodes choose to operate in downclocked mode, it is very unlikely that the incoming network traffic is frequent. Waking up on a larger interval would not impact app-level performance such as user-perceived response time.

Managing Downclocked Mode. Depending on the current network traffic and SNR conditions, an Enfold node may choose to switch from downclocked to normal mode and vice versa. Since the AP must precode data for downclocked operation, it needs to be notified of such mode transitions. We envision two ways of realizing this process. Clearly, one could design customized packets and a protocol for such a purpose. Alternatively, realizing that most deployed WiFi networks have good SNR [15], lower data rates such as 6/9 Mbps may never be picked by the rate selection algorithm of non-Enfold nodes. Thus, an AP could dedicate 6 and 9 Mbps for downclocked operation, and the AP would by default send precoded frame for 6 and 9 Mbps. As a consequence, Enfold nodes could leverage the existing re-association request frame to inform the AP that it only supports 6/9 Mbps and enter downclocked mode. An obvious concern is the potential loss of energy savings by not using higher data rates while in downclocked mode. However, as demonstrated in Chapter 5, the majority of the energy saving benefits can be attained with 6 and 9 Mbps for most smartphone apps.

4.5 Evaluation

In this section we evaluate downclocked OFDM reception in practice. We first evaluate specific steps of frame decoding on raw channel samples captured on the GNU Radio Universal Software Radio Peripheral (USRP) [23], which allows us to control SNR over a wider range. We then evaluate the Enfold prototype system implementation on Sora [60] to understand packet reception rates under downclocking under SNR and environment variations.

Table 4.1. Probability of detection P_d with different SNRs at 50% and 25% downclock rates.

Clock Rate	P_d(9 dB)	P_d(14 dB)	P_d(24 dB)	P_d(34 dB)
50%	0.993	1.0	1.0	1.0
25%	0.958	1.0	1.0	1.0

4.5.1 Microbenchmark Results

For our microbenchmarks we use the GNU Radio USRP platform to capture raw channel samples of 802.11a/g packets sent by a Sora node. We use the USRP board for packet capture because, unlike Sora, it has a hardware automatic gain control (AGC) on-board whose gain is adjustable. By varying the hardware AGC gain from 0 to 50, we can experimentally sweep the SNR range from 9 dB to 35 dB (a gain above 50 leads to saturation in our experiments).

For each gain setting we capture ten seconds of raw channel samples, containing approximately 3,000 WiFi packets. With the raw channel samples, we can emulate the effect of downclocking on various aspects of frame reception on the same data at all clock rates. At the full 100% clock rate, we use all the samples. At 50%, we use every other sample, and at 25% we use every fourth sample.

Packet Detection

First, we show that packet detection is not a constraint for OFDM decoding at downclocked rates. We employ a simple energy-based detector which tracks the average energy for a fixed duration and compares the output with a pre-defined threshold. Table 4.1 shows the detection probability for a range of SNR values. We treat the detection results at the normal clock rate of 100% as ground truth. If using the detection algorithm at full clock rate signals a packet, but using it at a lower clock rate does not, then we count the detection as having failed at the lower clock rate.

When SNR is extremely poor (9 dB), Enfold operating at 50% and 25% clock rates can still detect over 99% and 95% of the packets, respectively. For SNRs of 14 dB (which is still very poor) and above, both clock rates yield a 100% detection rate. These results indicate that packet detection is invariant with respect to clock rate for much of the SNR range, and certainly for SNR regimes expected of 802.11 communication.[4] Conversely, we find only 10–20 falsely detected packets (out of the roughly 3,000 packets captured in ten seconds) when downclocked.

Timing Synchronization

Next, we confirm that we can make timing synchronization also perform well for downclocked OFDM decoding. Recall from Section 4.3.1 that, since the sub-sampled short preamble sequence no longer exhibits excellent cross-correlation properties at downclocked rates, we instead perform cross-correlation on the long preamble sequence. Figure 4.5 shows the timing-estimation error (measured in number of samples) for both 50% and 25% downclock rates relative to the timing-estimation error at the full 100% clock rate. In effect, it measures the additional error of downclocking relative to the baseline error at 100%. Error bars show the standard deviation among the packets captured at each SNR.

When the SNR is extremely poor (9 dB), the synchronization error is substantial for downclocking at 25%. However, the synchronization error quickly converges to zero at 13 dB and above, and certainly for the expected SNR operating regime for 802.11. These results confirm our initial hypothesis that auto-correlation, which looks for repetitively transmitted symbols, will perform poorly when the noise level is very high. At such high noise levels, interference adds together rather than canceling out as in the cross-correlation case. Since we rely on the output of auto-correlation to begin the

[4]To place these values in context, many vendors (e.g., AT&T [1], Blackberry [2] and Cisco [3]) recommend coverage with a minimum of 25 dB when planning WiFi deployments.

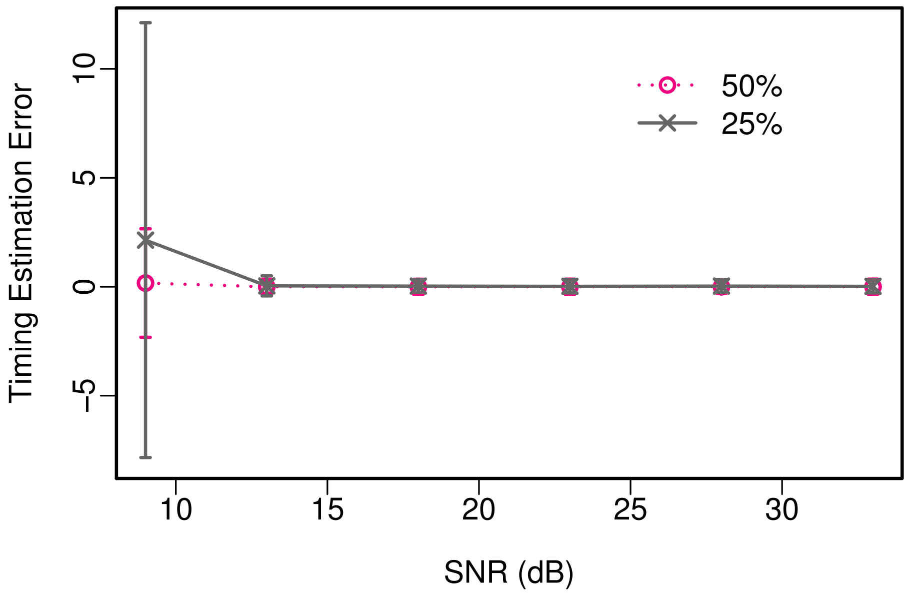


Figure 4.5. Timing synchronization offset error with downclocking at 50% and 25%.

search for cross-correlation peak, at 9 dB SNR, the 25% auto-correlation output already differs significantly from the 100% output. However, even with just slightly lower noise levels (at 13 dB in our case), our method can accurately determine symbol boundaries.

Frequency Offset Estimation

In Section 4.3.2, we showed theoretically that frequency offset estimation is oblivious to clock rate. As a final microbenchmark, we confirm that it is also not a constraint for downclocked OFDM decoding in practice. As above, we consider the frequency offset estimation obtained from decoding at the 100% clock rate as the ground truth for comparing estimations performed at the 50% and 25% clock rates. For the SNR range we could measure across (9–35dB), the estimation error at 50% and 25% clock rates are within 2.2% and 6.8% of the 100% rates on average. Previous work [35] has characterized that a frequency estimation error of 10% leads to $\leq$ 2 dB reduction in SNR, which is almost negligible for the SNR regime we consider.

4.5.2 Prototype System Evaluation

Our second set of experiments evaluates the Enfold implementation on the Sora platform at 6-Mbps data rates.[5] In particular, we explore the decoding performance of individual subcarriers, the decoding performance of entire packets across a range of SNR values, and the ability to trade decoding performance with raw data rate at very fine subcarrier granularities. Note that, since the Sora radio board does not have hardware AGC, in these experiments we use their provided HWVeri tool [6] to calibrate the SNR of two Enfold nodes at the start of each experiment.

Aliasing Induced Modulation

As discussed in Section 4.1.2, aliasing-induced modulation introduces additional constellation points that could lead to decoding errors. Our first experiment with the prototype explores the effect of aliasing on the constellations in practice. Lacking a precise metric to quantify this impact, we calculate what we term the *blocking distance* in constellations.

Each subcarrier at the original 100% clock rate will have a BPSK constellation diagram. At a 50% clock rate, two subcarriers will be aliased into a single 4-QAM constellation (twice the number of points). And at 25%, four subcarriers will be aliased into a single 16-QAM constellation (eight times the number of points). Since decoding performance fundamentally depends upon a distance threshold in the constellation, we calculate a minimum distance among constellation points to capture the effect of aliasing-induced modulation. To be more specific, if aliasing multiple subcarriers introduces many bit errors, the constellation points would have poor separation as represented by too small of a minimum distance.

[5]9 Mbps shares the same BPSK modulation scheme per-subcarrier as 6 Mbps while the coding rate changes to 3/4.

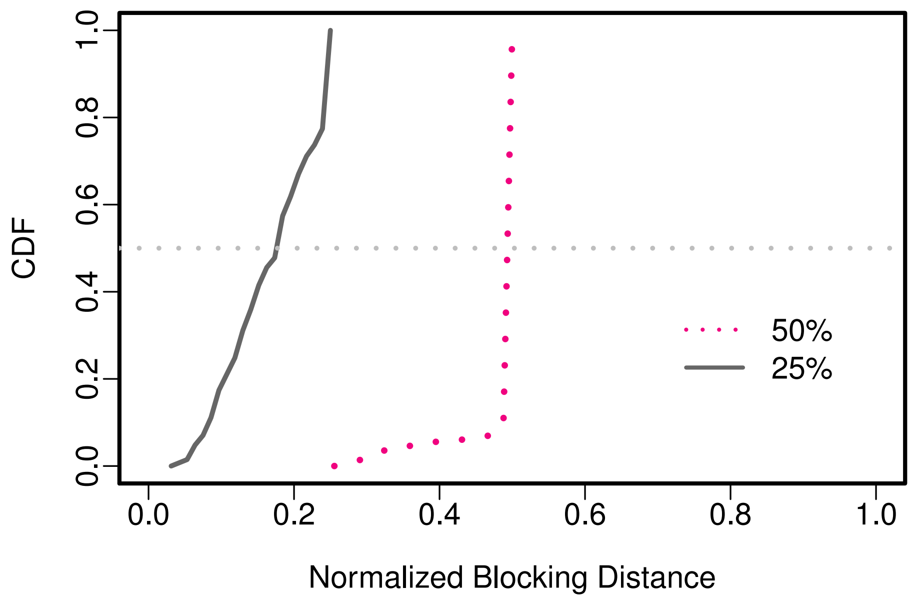


Figure 4.6. Cumulative distribution function (CDF) of normalized blocking distance (relative to 100%) for 2,000 WiFi packets.

As an example, consider the set of four subcarriers $\{10, 26, 42, 58\}$ which would be aliased together when downclocking at 25%. At a 100% clock rate, these subcarriers would each have their own BPSK constellation diagram. At 50%, though, there will be just two 4-QAM constellations for the subcarrier pairs $\{10, 42\}$ and $\{26, 58\}$, and at 25% there will be just one 16-QAM constellation for all four.

For each packet, we calculate the minimal of minimum distances for all constellation diagrams for a particular clock setting. At 100%, we calculate the minimum distance among constellation points for each subcarrier and record the minimal among the four subcarriers; at 50% it is the minimal among two constellations, and at 25% it is the minimum of the one resulting constellation. For each clock rate, we term this minimal distance the blocking distance. In an ideal decoding situation for downclocking, the blocking distance at 50% would be exactly half the blocking distance at 100%, and similarly the blocking distance at 25% would be one quarter of 100%.

Figure 4.6 shows the CDF of the blocking distance for constellation diagrams for the 50% and 25% clock rates normalized to the blocking distance at 100%. Each curve is a distribution over 2,000 packets received by our Sora implementation. We focus on just the four subcarriers $\{10, 26, 42, 58\}$, which are representative of the other subcarrier sets.[6] The distribution of blocking distances at the 50% clock rate is close to ideal: nearly 90% of the packets have a blocking distance that is half of the 100% distance. Performance at 25%, though, is notably worse: the blocking distances range from 0.03 to 0.25, with a median of 0.16; only 20% of the packets have a blocking distance close to a quarter of the 100% distance. This large variation at a 25% clock rate is not surprising. With four subcarrier responses added together, there will be more randomness in the aliasing induced constellation diagram.

Translating these blocking distances into SNR for per-subcarrier signal quality degradation, the penalty due to downclocking is 3 dB and 7.9 dB for 50% and 25% at medium, respectively. In our next experiment we show how this degradation impacts overall packet reception, which of course requires all subcarriers to be successfully decoded.

Packet Reception Rate vs SNR

With a sense of per-subcarrier decoding performance at downclocked rates, we now measure the overall packet reception ratio (PRR) of our prototype implementation under a range of SNR conditions. We measure packet decoding in two rounds of different packet sizes, one with small packets (100 bytes) and another with large packets (1,000 bytes). Each round of packet sizes consists of 50 runs, where each run has 100 back-to-back packet transmissions and attempted decodings. We disable retransmissions.

[6]Standard 802.11 uses only 52 subcarriers (including four pilot subcarriers) out of 64. The unused 12 subcarriers are set to zero, which makes downclocked decoding involving these subcarriers easier. In this experiment, we avoid these 12 subcarriers in the set we chose.

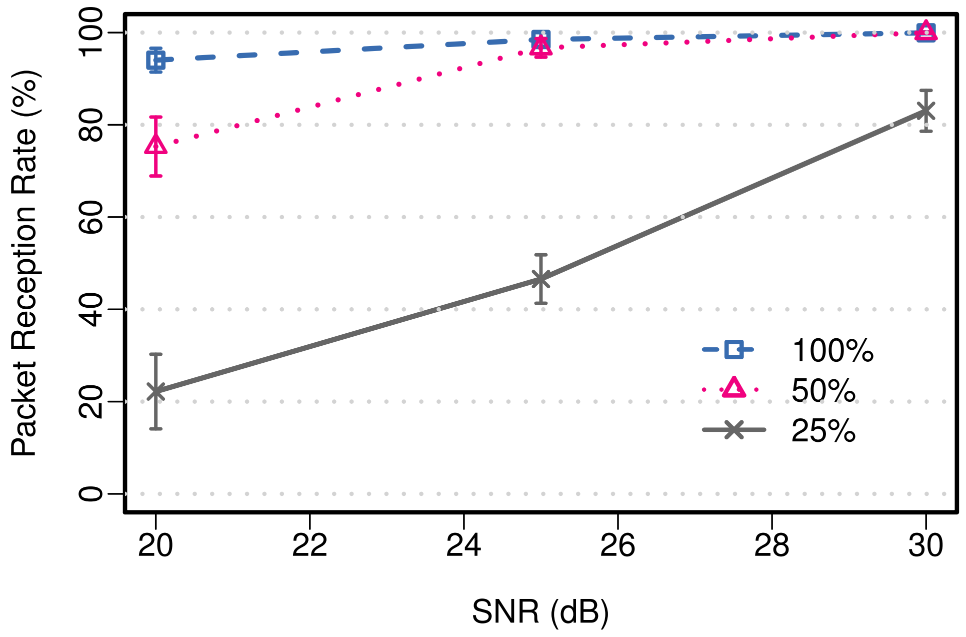
We transmit the two rounds back-to-back at a 100% clock rate, and then repeat the transmissions at 50% downclocked decoding rates and again at 25%. Altogether, the experiment transmits 30,000 packets.

We also vary the distance between the two Enfold nodes to induce different SNRs. Unfortunately, as noted above, due to the lack of hardware AGC on the Sora platform our SNR dynamic range is rather limited (20–30 dB as reported by Sora). As discussed above, this SNR range is at the very low end for recommended 802.11 operation, with 25 dB the recommended minimum when planning coverage. We could not receive any packet below 20 dB, i.e., it represents a sharp cut off in terms of PRR.

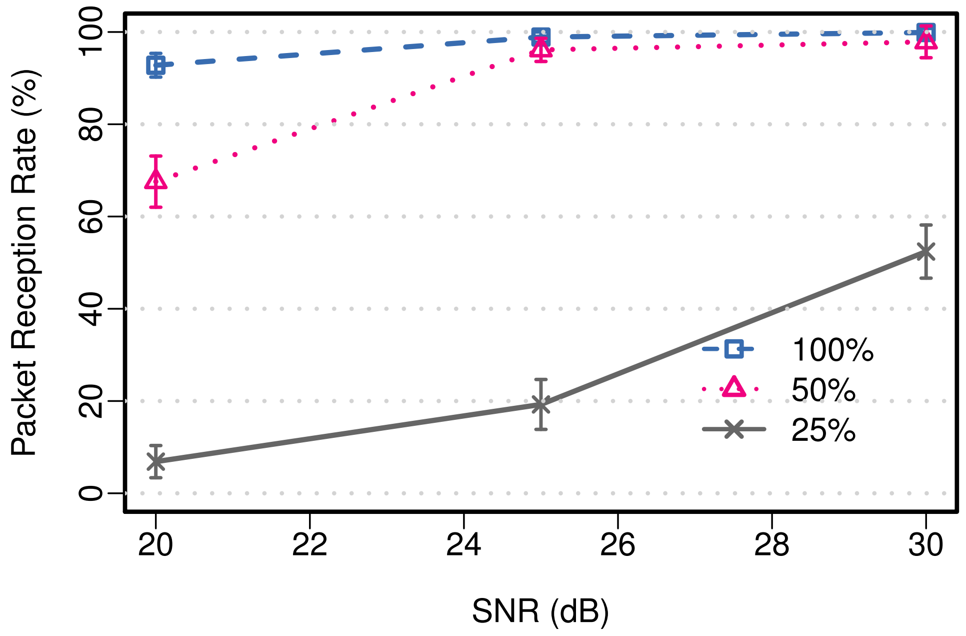
Figure 4.7 shows the results of these experiments. When SNR is at least 25 dB (the recommended minimum SNR for enterprise WiFi networks [1, 2, 3]), decoding at both 100% and 50% clock rates achieve $\geq$ 95% PRR for both packet sizes. For these operating regimes, the performance difference between the two clock rates is negligible. Below the 25 dB threshold, downclocking at 50% has a 67–75% PRR. Although lower than decoding at the full 100% rate, it is quite reasonable at such a low SNR.

For the challenging 25% clock rate, when SNR is 30 dB (5 dB above the recommended minimum), Enfold achieved 83% and 52% PRR for small and large packet sizes, respectively. For small packets, such a PRR translates to reasonable application performance with retransmissions, but the PRR for large packets will have a noticeable impact. At the edge SNR (25 dB), downclocking at 25% had poor performance, achieving 46% (small) and 20% (large) PRRs, respectively. Below 25 dB, downclocking at 25% is unusable for both packet sizes.

In sum, we find these results very promising. Note that the highest SNR we could achieve with the Sora platform was 30 dB, which is below typical operating conditions. A recent measurement study [15] of deployed university WiFi networks indicates that only 3% of WiFi frames experience retransmission (PRR is roughly 97%). According to



(a) Small packet (100 bytes)



(b) Large packet (1,000 bytes)

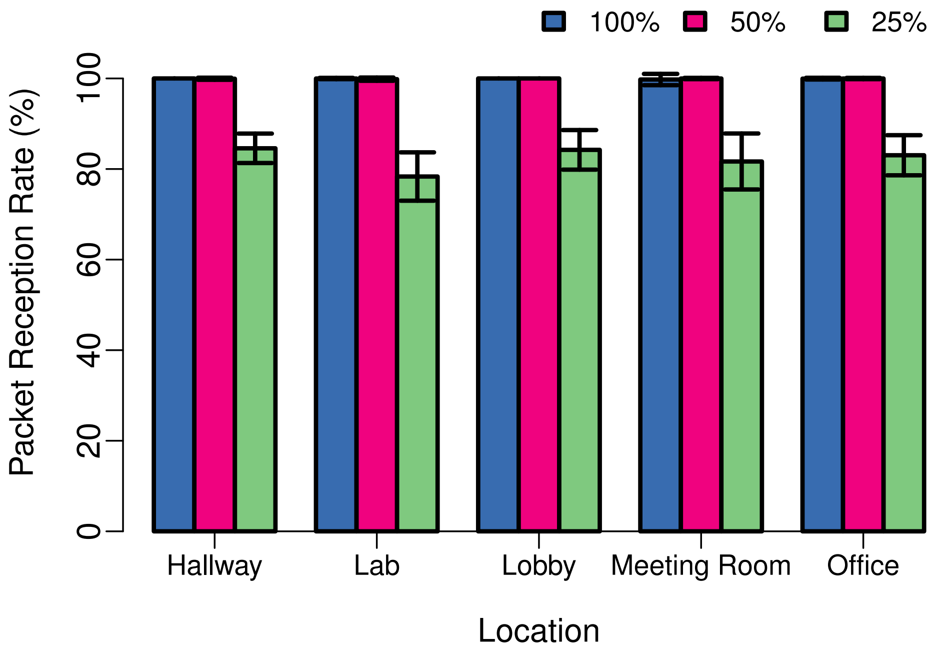
Figure 4.7. Packet reception rate as a function of SNR.

a WiFi hotspot study conducted by Pang *et al.* [43], a near 100% PRR corresponds to an SNR regime around 40 dB or more. On the other hand, when SNR goes below 20 dB, nodes struggle to even obtain Internet Protocol (IP) addresses after association (9 out of 181 successful Dynamic Host Configuration Protocol (DHCP) attempts) [29]. Although we could not measure at higher SNRs, our results show that downclocking at 50% already achieves excellent results even at low SNRs, nearly matching the performance of decoding at 100% at 25 dB. When downclocking at 25%, the situation is more complex. Our results up to 30 dB show that PRR is increasingly close to maximum performance for small packets. However, there is still substantial headroom for large packets and we cannot yet estimate at what SNR both curves will approach 100%. Thus, for poor or moderate SNRs with 25% downclocking, it remains an open question as to whether and when it is worthwhile trading off PRR for energy savings.

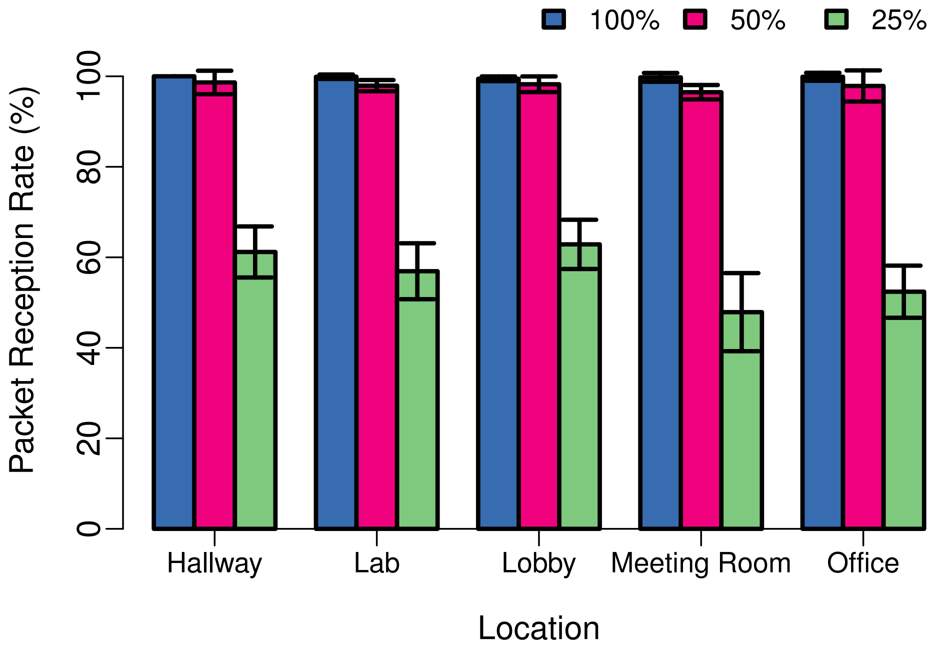
Impact of Wireless Environment

As measured in Section 4.5.2, the performance of aliasing induced modulation entirely depends on the surrounding wireless environment, reflected in the blocking distance. In the extreme case where the channel response is flat for the entire spectrum, given how bits are mapped under BPSK, multiple constellation points would collapse onto each other under aliasing. In our previous experiments, all nodes were in the same room (a seven-person office). In this experiment, we vary locations to evaluate the performance of downclocking under a much wider range of wireless conditions.

We repeat the experiments in Section 4.5.2 at four additional locations in the same building but with significantly different propagation environments. These locations are a meeting room (smaller than the office), a long hallway, a floor lobby of mixed open space and furniture, and a lab (much larger than the office). In all locations we use a fixed SNR of 30 dB.



(a) Small packet (100 bytes)



(b) Large packet (1,000 bytes)]

Figure 4.8. Raw packet reception ratio for different clock rates at five different locations.

Figure 4.8 shows the packet reception ratios for all locations and both packet sizes. Error bars show the standard deviation across 50 runs. For both the 100% and 50% clock rates, the PRRs are indistinguishable for all locations regardless of the packet size. For the 25% downclock rate, though, the PRR does vary across locations. The PRR varies moderately for small packets (78% to 85%), but has a larger variation for larger packets (48% to 63%). These results confirm that the downclocking performance Enfold achieves is stable across different wireless environments, and is not tied the opportune conditions of just one environment.

Trading Throughput for PRR

In the last set of experiments, we explore one of the design freedoms offered by Enfold. Recall from Section 4.3.3 that we restore the pilot subcarriers by placing known values on selected subcarriers. We explore this design parameter further by putting known values on other data subcarriers. The benefits are twofold: these known values (bits) improve the performance of the Viterbi decoder and they help to reduce the size of constellation diagram. For instance, for the subcarrier set from the experiment in Section 4.5.2, if we put a known data value on subcarrier 10, the resulting constellation diagram under 25% clock rate will degenerate to 8-QAM instead of 16-QAM. However, using more subcarriers will reduce the effective data throughput. In situations where both power and interference are challenging, such as sensors, applications might want to trade off reliability for throughput at lower power. To explore this trade-off, we repeat the experiment in Section 4.5.2 at a 25% clock rate while adding known bits to subcarriers.

Figure 4.9 shows the PRR and effective data rate with respect to the number of known bits sent (zero additional bit corresponds to the results for 25% at 30 dB in Section 4.5.2). As discussed above, 802.11 only uses 52 out of 64 subcarriers for data. As a result, only 3 out of the 16 subcarriers are 16-QAM modulated at 25% clock rate.

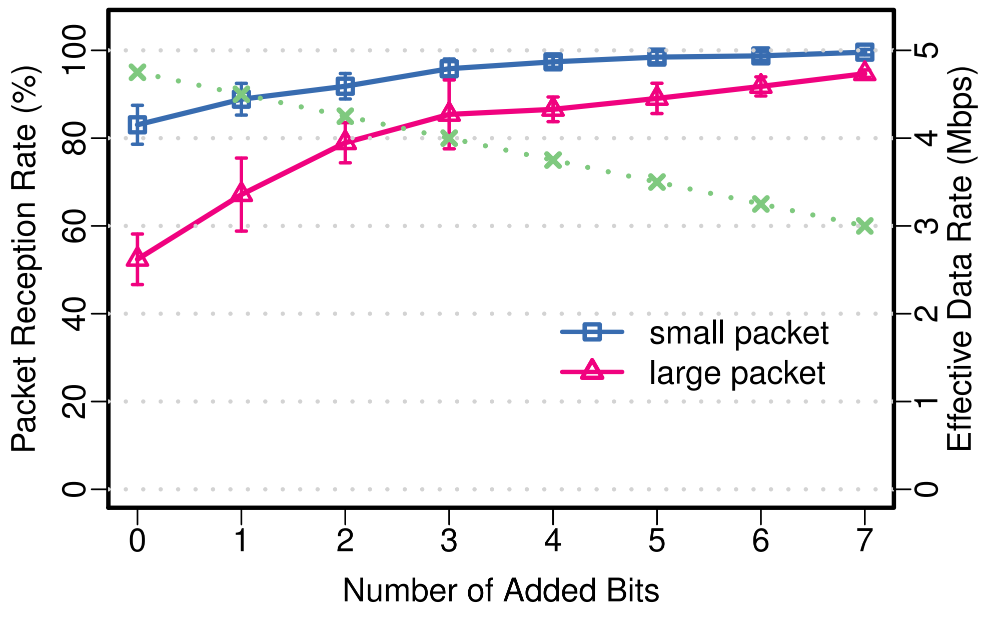


Figure 4.9. PRR with additional known bits on subcarriers for a 25% downclock rate (SNR = 30dB). The right *y*-axis shows the corresponding effective throughput.

Therefore, we put the known bits on these three subcarriers first, which helps PRR by reducing the original 16-QAM modulation to 8-QAM. Consequently, PRR improves substantially for both small and large packet sizes. For example, with three known bits added, PRR jumps to 96% and 85% for small and large packets, respectively. After adding three bits, all constellation diagrams on the 16 subcarriers are 8-QAM modulation (except subcarrier 0 which is 4-QAM). Each additional bit then transforms one of the 8-QAM constellation diagrams to 4-QAM. As expected PRR continues to improve, but with a smaller delta than when transforming 16-QAM to 8-QAM.

The trade-off in improved PRR is a reduction in data throughput in proportion to the number of additional known bits used for pilots. The PRR eventually reaches excellent PRRs of 99.6% (small) and 94.7% (large) when adding seven bits, but at that point the effective data rate drops to 3 Mbps (half of the raw PHY layer data rate). In summary, Enfold provides the flexibility to trade off effective data rate for PRR performance, improving Enfold's applicability in poor SNR regimes where applications favor such a trade-off.

4.6 Summary

In this chapter, we presented an approach for successfully receiving and decoding OFDM modulated WiFi signals while sampling below the Nyquist frequency. We exploit the aliasing that results from under-sampling and observe that there exists well-defined structure in terms of how OFDM signals are "folded up" under aliasing. In particular, we show that a standards-compliant 802.11a/g frame can be detected, received, and decoded by a receiver running at a 50% or even 25% clock rate given sufficient channel quality. We design and implement a standards-compliant 802.11 receiver on the Sora platform and experimentally show that our aliasing induced modulation can attain greater than 96% and 83% raw packet reception rates while reducing the clock rate by $2\times$ and $4\times$, respectively. Given that both SloMo and Enfold trade SNR (throughput) for energy savings, we evaluate whether such a trade-off is worthwhile for typical smartphone applications in the next chapter.

Chapter 4, in part, is a reprint of the material as it appears in Proceedings of the ACM Annual International Conference on Mobile Computing and Networking 2014. Lu, Feng; Ling, Patrick; Voelker, Geoffrey M.; Snoeren, Alex C. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Energy Consumption Evaluation

Our experiments with both SloMo and Enfold demonstrate the feasibility and performance of downclocked 802.11 communication. In this chapter, we evaluate the potential energy savings when using downclocking in the context of contemporary smartphones and popular apps. We start by focusing on SloMo given that SloMo seamlessly communicates with existing deployed WiFi networks, and describe our evaluation methodology first. We then move to understand how downclocking with SloMo reduces energy consumption, impacts free channel airtime, and compares with alternative WiFi energy-saving approaches. Finally we extend our evaluation to Enfold, and demonstrate that Enfold continues to save substantial energy in places where SloMo does not, extending the applicability of downclocking to an even wider range of smartphone apps.

5.1 Methodology

Since we could not directly measure the power consumption of a downclocked WiFi chipset in an actual smartphone, we construct a power model based on measurements of a real device. We also collect media-access-control (MAC) layer packet traces of a variety of real apps running on two different smartphones. We use these traces to infer the instantaneous power state of the smartphones' WiFi chipsets and compute the total energy cost for each phone based on the power model.

Table 5.1. WiFi Power Characteristics

Parameter	Full Clock / Downclocked (1/4)
Beacon Interval (ms)	100 / 100
Beacon Wakup Period (ms)	2.5 / 2.5
Light Sleep Tail time (ms)	500 / 500
Deep Sleep Power (mW)	10 / 10
Light Sleep Power (mW)	120 / 120
Beacon Wakeup Power (mW)	250 / 185*
Idle Power (mW)	400 / 260*
Rx Power (mW)	600 / 360*
Tx Power (mW)	700 / 460*

5.1.1 WiFi Power Model

Similar to Mittal *et al.* [40], we parameterize our smartphone WiFi power model on the measurements of a Nexus One reported by Manweiler and Choudhury [38]. When actively transmitting and receiving frames, a WiFi chipset must be in a high power state. Once a network transfer completes, the card moves to the idle state. If there is no network activity for a while, the card transitions to the light sleep mode. The light sleep state still consumes a significant amount of power in anticipation of efficiently waking up for incoming traffic. On the Nexus One, the light sleep tail time is roughly 500 ms; if no further network activity occurs, the card returns to the deep sleep state.[1] Table 5.1 summarizes the model parameters we used. Most are reproduced from [38, 40]; the downclocked values marked with an asterix are estimated as follows.

WiFi power consumption falls into two parts, the analog front-end P_a and the digital processing logic P_d. In the sleep state, the digital logic part is turned off. Given the description of the two sleep modes, we infer that the power difference between them is due to the analog front end remaining functional in light sleep mode but turned off in

[1]We observe the Nexus One employing a variety of beacon wakeup periods (2.5,5,10 ms) on the power measurement trace obtained from the authors of [38]; we use 2.5 ms in our model to be conservative.

deep sleep mode. Therefore, we use the light sleep state power as an estimate for the analog power consumption P_a. We then estimate the downclocked power consumption as proportional to the full digital power consumption P_d/α, where α is the clock scaling ratio. When downclocking by a factor of 4, for instance, α at best would be 4 as well. Since it is likely that a practical implementation would experience suboptimal scaling, we conservatively choose $\alpha = 2$ to obtain a lower bound estimate.[2] Note also that the analog part P_a for Tx is greater than Rx since transmission includes an additional power amplifier component. We use the difference between Rx and Tx power (100 mW) from the measurements in previous work to approximate the power consumption of the amplifier.

5.1.2 Smartphone App Traces

To comprehensively evaluate the benefits of SloMo, we sampled a wide range of popular smartphone apps (each has at least five-million downloads). These nine apps include familiar Internet services like Facebook and Gmail, as well as smartphone-specific services like Pocket Legends (a real-time massively multiplayer game) and TuneIn Radio (a streaming audio service). They differ significantly in the way they interact with the network, spanning interactive real-time traffic to content prefetching to intensive data rates.

We collect high-fidelity WiFi packet traces [55] by configuring two MacBook Pro laptops as sniffer nodes in the vicinity of the smartphone and the access point (AP), respectively, and merge the two traces to minimize frame losses. To eliminate bias due to starting and closing the app, we only record a trace when an app is in steady state. Each such capture session lasts for 200 seconds. Finally, to avoid tying our conclusions to a particular smartphone platform, we conduct our experiments on the Google Samsung

[2]The maximum possible energy savings in this model is 35%.

Nexus S (Nexus) and the Apple iPhone 4S (iPhone). We collected the traces with 4–5 other WiFi devices concurrently using the network, and we emulated a typical SNR scenario where the AP and the wireless station are in the same building but different rooms (i.e., no line of sight between the two). Since WiFi devices signal the AP of their intention to sleep and wake up, we are able to faithfully recreate the power state transitions of the WiFi cards on the smartphones using the captured network traces.

5.2 SloMo Energy Consumption

Figure 5.1 and 5.2 compare the network energy costs of the apps by power state when using standard 802.11 PSM and when using downclocked communication with SloMo on the Google Nexus S and iPhone 4S traces, respectively.[3] To emphasize energy consumption, we assume SloMo operates at 2 Mbps and the data rates for PSM are the ones reported by the packet capture software. The graphs show results for running the apps on both the Nexus and iPhone. The trends for both phones are similar, but since the iPhone has shorter idle tail times (30–90 ms in our traces versus 220 ms for the Nexus) the benefits of SloMo are smaller for the iPhone than the Nexus. Both figures show that a wide range of popular apps benefit from SloMo, but they do so for different reasons. To provide insight into the different app behaviors, Figure 5.3 shows the probability density functions (PDFs) of the inter-frame times (IFTs) for four distinctive apps.

Energy consumption in the first group of apps (Skype Voice, Pocket Legends, TuneIn Radio) is dominated by time spent in the idle listening state. Since WiFi cards still consume substantial energy while idle (Table 5.1), downclocking significantly reduces idle state energy consumption [70]. And since these apps have low data rates, the energy saved during idle listening far exceeds the additional energy consumed for slower data

[3]The number at the end of the bar group shows the relative energy saving of SloMo over PSM, the higher the better. We also report the bi-directional MAC layer data rate.

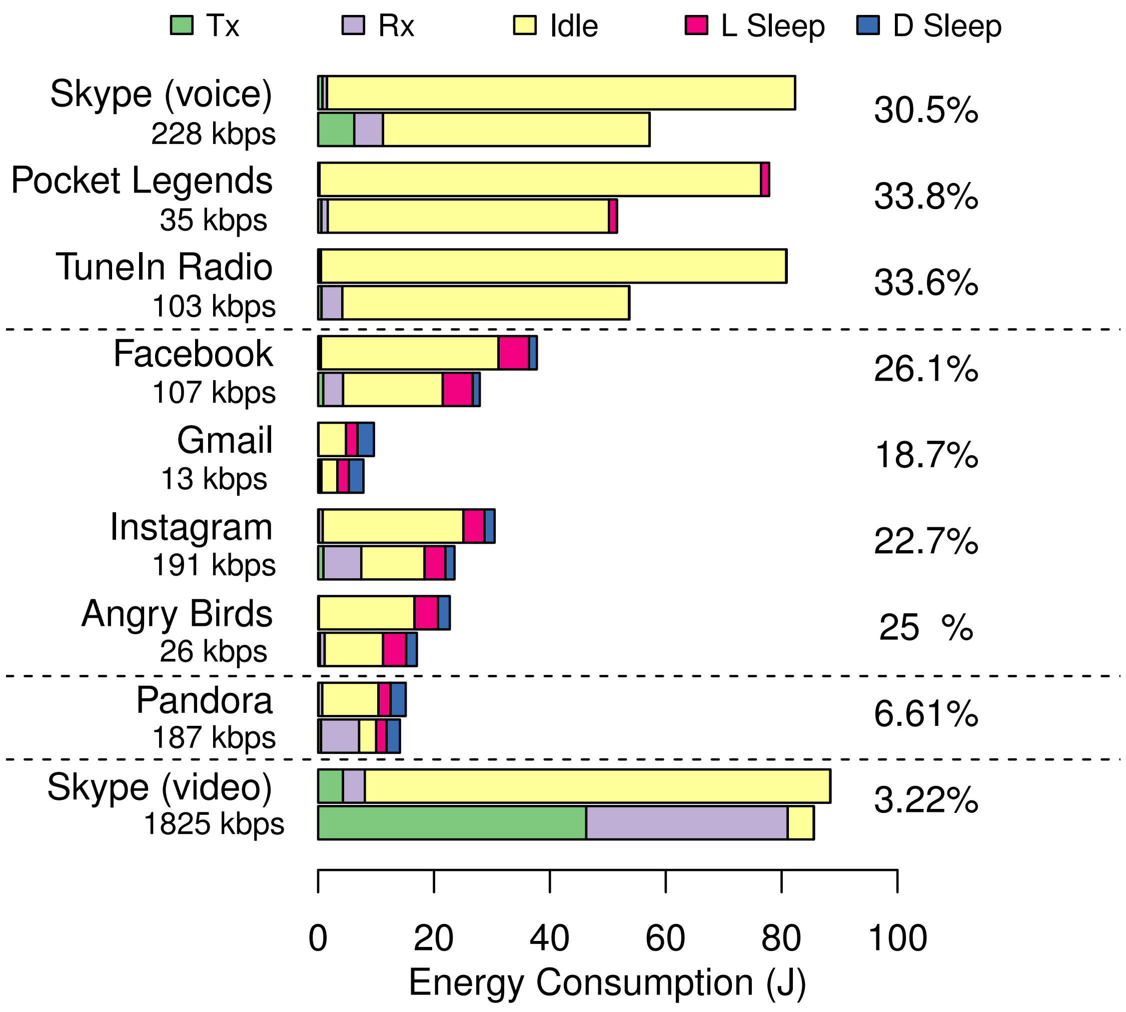


Figure 5.1. Energy cost of various apps under 802.11 power save mode (PSM) and SloMo (Google Nexus S). For each app, the upper bar corresponds to the breakdown of energy consumption under 802.11 PSM while the lower bar corresponds to SloMo.

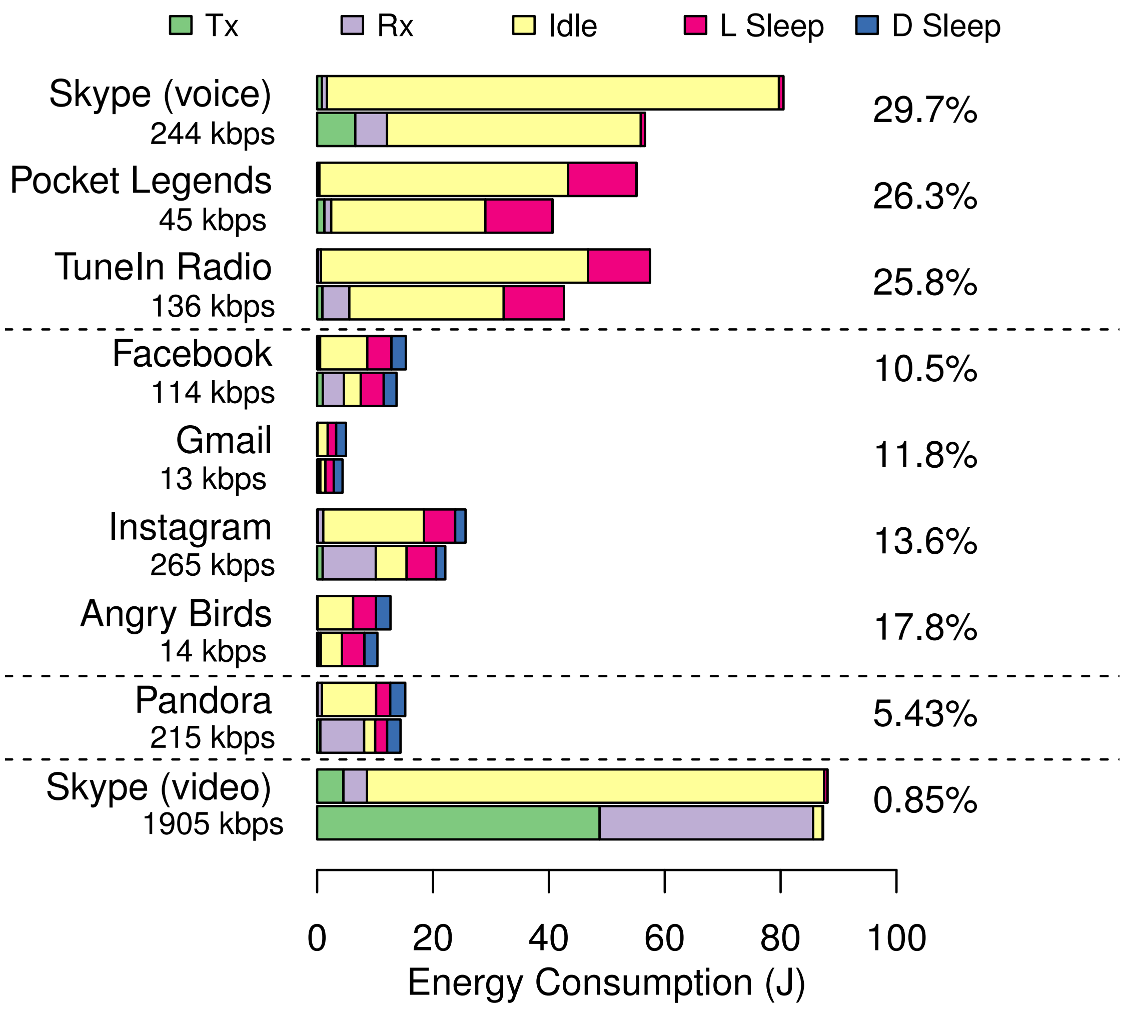


Figure 5.2. Energy cost of various apps under 802.11 power save mode (PSM) and SloMo (iPhone 4S). For each app, the upper bar corresponds to the breakdown of energy consumption under 802.11 PSM while the lower bar corresponds to SloMo.

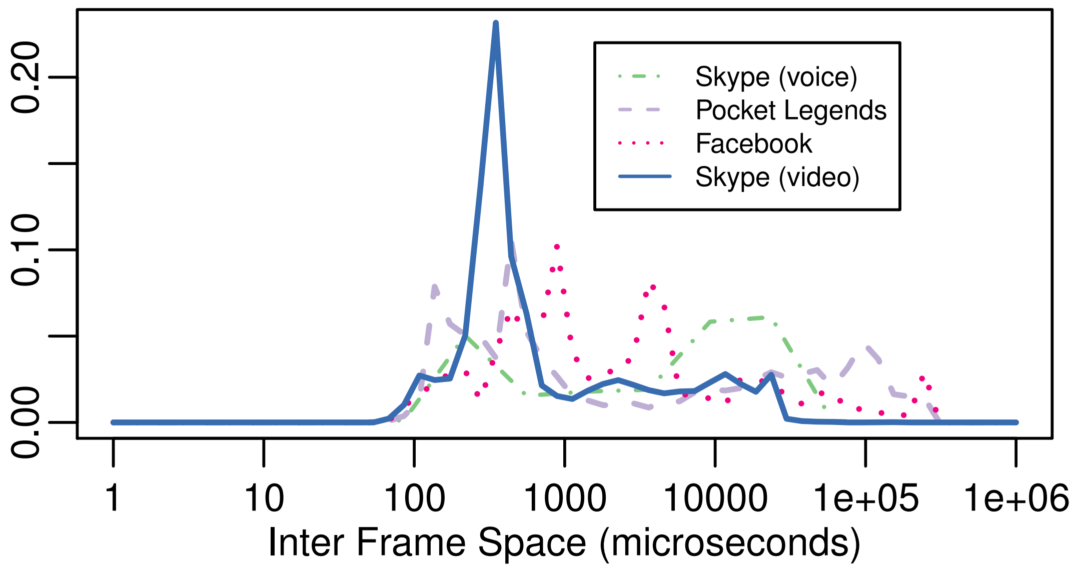


Figure 5.3. PDFs of the IFT for a selected set of apps on Neuxs S. We remove the inter frame time (SIFS) between DATA and acknowledge (ACK) frame for better presentation. IFTs larger than a sleep period are also removed.

transmission and reception, resulting in energy savings of 30–34% overall on the Nexus. Although these apps have low data rates, their network behavior prevents them from entering sleep mode while idle and makes them relatively power-hungry: As real-time apps, they send and receive packets at frequencies that keep the WiFi card awake in constant active mode (CAM). Figure 5.3 shows that Skype Voice exchanges packets roughly every 10 ms, and that the Pocket Legend client exchanges game updates with its server as a burst of packets every 100 ms (the peak near 100 μs is the IFT between packets in a burst). TuneIn Radio similarly keeps the WiFi card awake for frequent incoming packets (curve not shown for clarity).

The next group of apps (Facebook, Gmail, Instagram) interact with the network much more intermittently at human time scales. Users navigate through the app and download bursts of content, with pauses in between (e.g., Instagram had an average pause time of 1.5 seconds). For such apps, the WiFi card wakes up intermittently when downloading content, and transitions first to idle and then to sleep mode during the longer pause times. Even so, SloMo can still reduce energy consumption during the idle tail time after intermittent network activity and, to a minor effect, during the sleep states. Again, the benefits of downclocking and saving energy during these states outweigh (by 19–26%) the additional energy spent transmitting and receiving at low data rates.

Angry Birds is a good example of many "offline" free apps. Although the game itself does not require network interaction, the embedded ads in the free version cause the app to have similar network characteristics as Facebook and Instagram. The app has intermittent network activity uploading user information and downloading tailored ads, but after each interaction the WiFi card enters the idle state before transitioning to sleep. As a result, Angry Birds spends over 95% of its network energy in the idle tail time, which can account for 65–75% of the entire app energy consumption [45]. (Although the data rate of Angry Birds is just 14% of Instagram, it consumes comparable network energy.) Once again downclocking can substantially reduce energy consumption in the idle state for a 25% savings overall.

Although a music streaming service, Pandora differs from the previous apps in that it prefetches entire songs at a time. In our trace, it downloads a song in the first ten seconds and has very little network activity for the next 60 seconds. With this behavior, Pandora already uses the network efficiently. Although SloMo does reduce energy consumption by downclocking during the idle and sleep states, it correspondingly increases it for reception and on balance only marginally improves total consumption.

Finally, Skype Video exhibits a similar trade-off as Pandora. The energy saved by SloMo in downclocking during idle time is matched by the energy expended in using the network at low data rates. In terms of network energy, Skype Video is a wash. As we discuss below, however, SloMo is a poor choice for this kind of app because of the channel airtime it consumes.

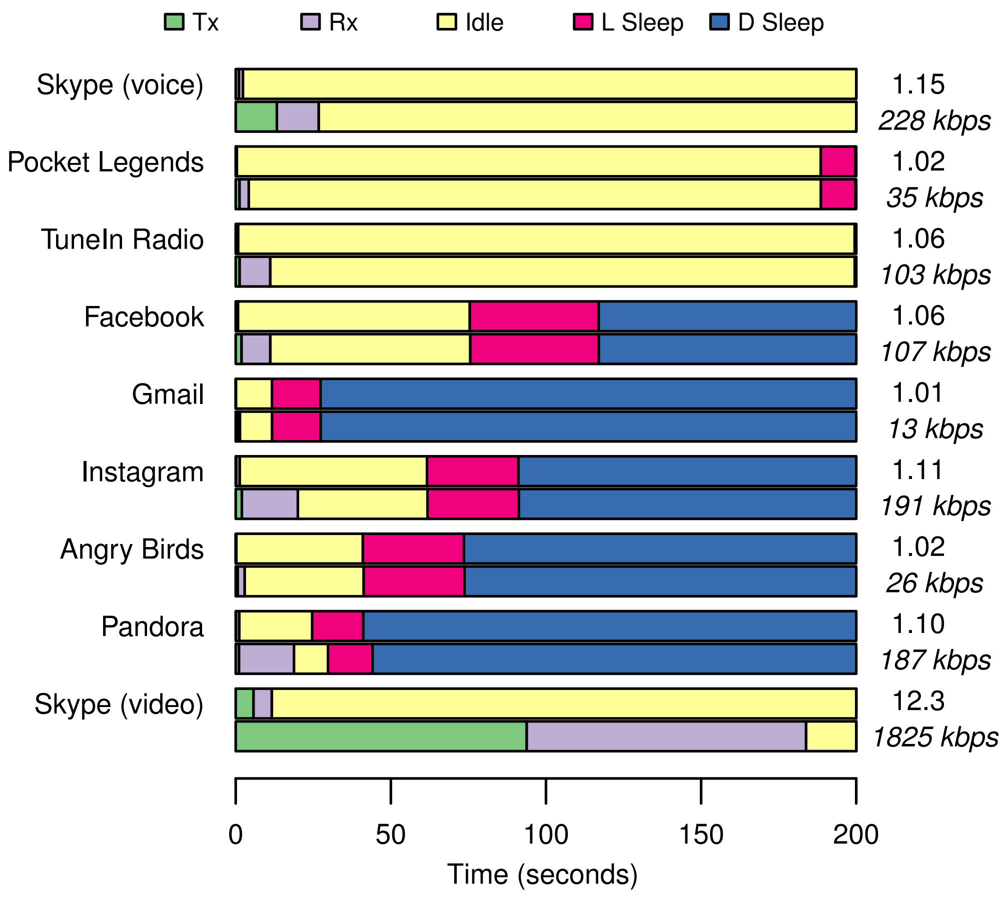


Figure 5.4. Comparing the timing breakdown for various apps under 802.11 PSM (upper bar) and SloMo on Google Nexus S (lower bar). The number at the end of the bar group shows the free channel airtime contraction ratio, the lower the better with 1.0 as the minimum.

5.3 Network Impact

Given that SloMo trades off data rates for energy consumption, it is also important to consider the overall network impact due to the use of slower data rates by SloMo in terms of channel airtime. Since our design explicitly trades throughput for energy savings, an app might save itself energy by downclocking but unduly impact other devices on the network by consuming more channel airtime using lower data rates, thus increasing the overall network contention level.

Figure 5.4 shows the channel airtime breakdown of the apps on the Nexus S. It compares the time spent in the states when the apps use standard 802.11 PSM (top bar) and SloMo (bottom bar). The number to the right of each paired bar denotes the contraction in free channel airtime for using SloMo with the app. For instance, the free channel airtime for Skype Voice using PSM divided by the free channel airtime using SloMo is 1.15. The graph shows that downclocking with SloMo does cause the apps to spend more time in actively transmitting and/or receiving. For all apps except Skype Video, the impact on free channel airtime is modest, with contractions ranging between 1.02–1.15. With the much higher data rates of Skype Video, though, using SloMo causes the app to spend most of its time receiving and transmitting data, greatly reducing the free channel airtime compared to PSM.

5.4 Alternative Approaches

So far we have compared SloMo with current WiFi implementations using PSM. As the traces revealed, though, a critical source of network energy consumption is the tail time of the idle state. Of course, other solutions have been proposed to address this issue as well. As a final evaluation, we compare SloMo with two other approaches, U-APSD [7] and E-MiLi [70], from industry standards and the research community, respectively.

U-APSD. When traffic patterns are periodic, predictable, and symmetric, such as real-time VoIP traffic, the Unscheduled Automatic Power Save Delivery (U-APSD) optimization (defined by the 802.11e standard [7]) could allow devices to enter the sleep state immediately after network activity and avoid the standard tail time in the idle state. Based upon the U-APSD specification, we emulated its use[4] for the Skype Voice and

[4]We attempted to purchase U-APSD compliant APs and WiFi cards to experiment with a real implementation, but could not find a hardware, operating system (OS), and driver combination that enabled its use in practice.

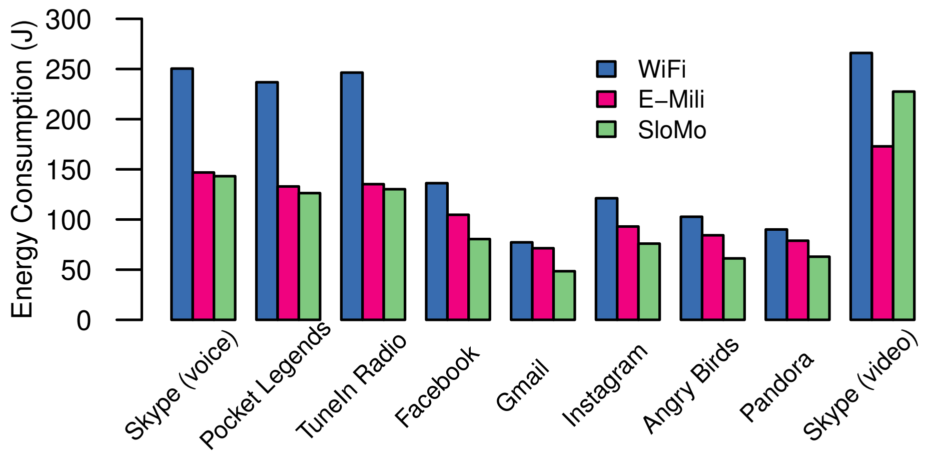


Figure 5.5. Comparing energy consumption among PSM, E-MiLi and SloMo on the Nexus trace set.

Video apps using the Nexus traces and estimate impressive energy savings of 56% and 44%, respectively, compared with savings of 30.5% and 3.2% using SloMo. Although clearly better in the ideal case of Skype, as noted by others [51] U-APSD is not a general optimization because its effectiveness depends greatly on the degree of symmetry in the traffic. For real-time apps where the traffic pattern is asymmetric, such as Pocket Legends and TuneIn Radio in our examples, U-APSD would not apply. Further, U-APSD is not suitable for intermittent traffic, such as with the Facebook and Gmail apps, which could lead to unnecessary energy waste due to frequent polling of the AP [54].

E-MiLi. E-MiLi re-designs the addressing mechanism of WiFi devices, enabling receivers to determine whether traffic is addressed to them without leaving a low-power listening state [70]. We emulate the use of E-MiLi on our Nexus app traces based upon the WiFi power model and measurements reported by the E-MiLi authors.[5] To facilitate the comparison, we apply the E-MiLi power model to SloMo in contrast to our previous

[5]We measure a WiFi card (Atheros AR9380) from the same manufacturer as the published E-MiLi results to obtain details regarding the power consumption of the sleep state not reported in the E-MiLi paper. The card wakes up at every beacon interval and stays awake for 20 ms before going back to sleep.

experiments.[6] Figure 5.5 compares the network energy consumption of PSM, SloMo and E-MiLi on the Nexus apps traces (results were similar for the iPhone traces). Across all apps, downclocking with SloMo saves on average 37.5% energy relative to the default PSM, about 10% more than the 27.7% savings achieved with E-MiLi. For the initial three real-time apps, both SloMo and E-Mili obtain comparable savings. For the others, SloMo performs significantly better than E-Mili, while E-MiLi performs significantly better on Skype Video.[7]

5.5 Extending Energy Saving with Enfold

As shown in Section 5.2, SloMo could reduce energy consumption by up to 34% for many popular smartphone apps. However, for bandwidth hungry apps such as Skype Video, it fails to deliver similar energy savings under downclocking due to its limited 1/2-Mbps direct sequence spread spectrum (DSSS) data rates. By exploiting the aliasing effect, we introduced Enfold in Chapter 4, which extends downclocking to orthogonal frequency-division multiplexing (OFDM) signals and demonstrates that the supported data rate could be much higher (6 Mbps in our current prototype and potentially even higher in future). Therefore, to evaluate the energy benefits of Enfold, in addition to the nine apps studied in Section 5.2, we add three traces of higher bandwidth apps: Wyslink (live TV streaming at 479 kbps on average), FaceTime (1499 kbps), and YouTube (588 kbps).

We compare energy consumption among OFDM downclocking with Enfold, DSSS downclocking with SloMo, and the default power save mode (PSM) in 802.11g as a baseline. While our Enfold prototype employs a 6-Mbps rate, we also introduce a hypothetical case where the downclocked reception data rate is set to 54 Mbps (the

[6]As a result, the SloMo energy savings are 10–20% larger relative to PSM compared to the results presented in Figure 5.1.

[7]The energy savings under Enfold (169 J) slightly outperforms E-MiLi (172 J) for Skype Video.

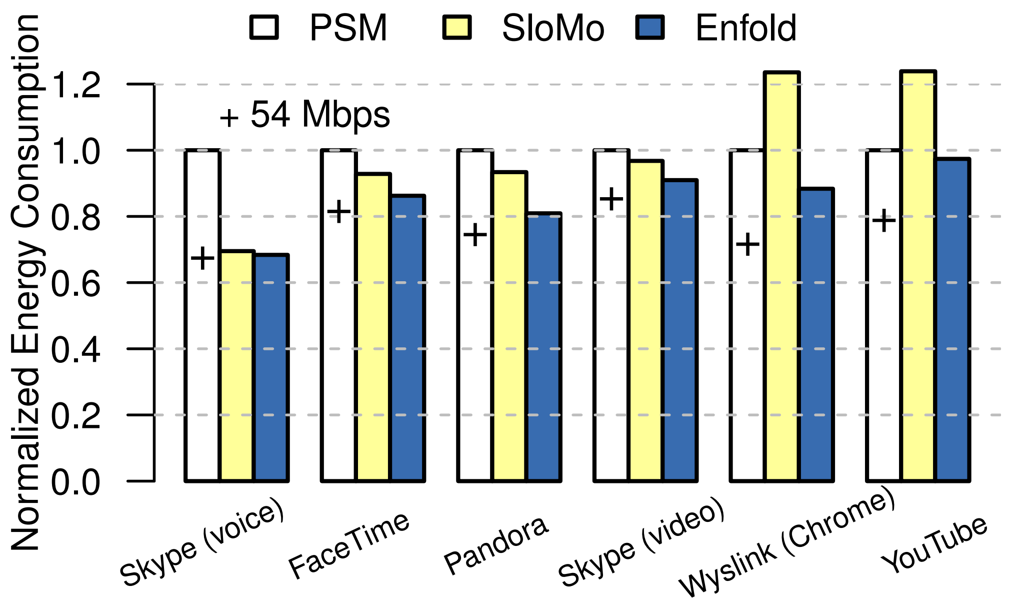


Figure 5.6. Energy comparison among PSM, SloMo, and Enfold. We also include a hypothetical case where the downclocked reception data rate is 54 Mbps and PRR is 100%. Results are normalized with respect to PSM.

highest rate in 11g) with 100% packet reception rate (PRR) to estimate how much potential energy savings remain unrealized because Enfold does not operate at higher rates. We also conservatively assume the chipset does not have a fast-switching clock. Therefore, although an Enfold node can receive at a 6 Mbps OFDM data rate, it would have to transition to a 2-Mbps DSSS data rate for transmission (including ACKs). On the other hand, the data rates used for PSM are the actual ones reported by the packet capture software in the trace.

Enfold consistently outperforms PSM for all the app traces, and matches or exceeds SloMo depending on the app workload. Figure 5.6 compares the normalized energy consumption of PSM, SloMo (2-Mbps DSSS rate with 100% PRR) and Enfold (4.75 Mbps with 63% PRR in Chapter 4). We focus on six representative apps out of the twelve; the remaining apps have low data rates with performance similar to Skype voice (i.e., both SloMo and Enfold improve upon PSM, but behave similarly to each other).

As described in Section 5.2, apps like Skype voice frequently require network access but the amount of data transferred is low (data rates range from 10s to 100s of kbps). As a consequence, the WiFi card either stays in constant awake mode (CAM) or is woken up intermittently every few seconds, and therefore the network energy consumption is largely dominated by idle listening in the network tail time. Given their small app data rates, the extra energy spent on transmission and reception due to slower data rate or retransmissions is more than compensated for by the energy saved during the idle state, and to a certain extent, sleep state. Even the 2-Mbps DSSS rate in SloMo is sufficient for these apps. As a result, both SloMo and Enfold save significant energy (up to 34%) for these apps compared to PSM.

However, as the app data rate increases (e.g., Skype Video, Wyslink) or interactivity reduces (e.g., Pandora, which prefetches the entire song before playing out) or both (YouTube, which downloads chunks of data every 10–20 seconds), SloMo has small or negative benefits. The energy saved in the idle listening and sleep states is matched or exceeded by the energy expended for transmission and reception at the slower data rate; for some apps, SloMo consumes over 20% more energy than PSM. In contrast, given the increased data rates supported (2.4$\times$ SloMo rate for the current implementation), Enfold is able to save appreciable energy for these apps: from 9% for Skype Video to 19% for Pandora. YouTube sees little benefit (3%) because it already uses the network efficiently, particularly in the face of retransmissions with Enfold's 63% PRR.

In the best case of a network with good SNR conditions where the PRR is 100%, energy savings are correspondingly better: from 10% for Skype Video to 20% for Pandora (with YouTube at 8%, showing the effects of PRR). In sum, Enfold significantly extends the range of apps that benefit from downclocking. Compared to the hypothetical 54-Mbps case at 100% PRR, the additional energy gain over 6 Mbps in Enfold is small compared to the substantial difference in data rate: Pandora (4.5%), FaceTime (3.6%), Skype Video

(3.9%), Wyslink (8.4%) and YouTube (12%). As a result, for these popular apps Enfold at 6 Mbps already captures much of the energy savings.

As a final note, we also evaluate energy consumption with Enfold for other data rates and PRRs (the throughput-PRR trade-off enabled in Section 4.5.2). The results among these Enfold variants are roughly the same with a 2–3% variation. Although the difference in energy consumption is small, higher PRR leads to fewer retransmissions and reduced network contention.

5.6 Summary

As smartphones become ever more pervasive, energy consumption is deemed to be one of the biggest challenges for wireless and mobile computing. Both SloMo and Enfold trade throughput (SNR) for energy savings by moving to downclocked states with potentially reduced data rates in proportion to downclocking ratio. Based on network traces from popular smpartphone applications, we show such a trade-off is worthwhile; SloMo reduces WiFi power consumption on contemporary smartphones by over 30% for a wide range of applications. Realizing the rate limitation of SloMo, Enfold further extends the energy saving benefits to bandwidth hungry apps, with substantial benefits ranging from 10% to 20%. Since our approaches reduce energy consumption by fundamentally re-designing the WiFi transceiver, there is *zero* modification required at the MAC, transportation and application layers, which makes our solutions easily deployable and widely applicable.

Chapter 5, in part, is a reprint of the material as it appears in Proceedings of the USENIX Symposium on Networked Systems Design and Implementation 2013. Lu, Feng; Voelker, Geoffrey M.; Snoeren, Alex C. The dissertation author was the primary investigator and author of this paper.

Chapter 5, in part, is a reprint of the material as it appears in Proceedings of the ACM Annual International Conference on Mobile Computing and Networking 2014. Lu, Feng; Ling, Patrick; Voelker, Geoffrey M.; Snoeren, Alex C. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Conclusion and Future Work

The link rate of popular wireless technologies, such as 802.11, has increased markedly over the past two decades: 802.11ac promises gigabit speeds in handset form factors. Yet only a small fraction of the devices outfitted with such radios actually make use of the full channel capacity, and, even if they do, only do so sporadically. Hence many existing wireless devices frequently under-utilize the channel, yet reap no particular benefits for doing so. In particular, neither transmission nor reception is power proportional in typical devices, even when the channel quality is sufficiently high to support far more energy-efficient modulations and encodings. While devices aggressively try to move their radios into low-power states whenever possible, studies show [45, 68, 70] that many common applications for both smartphones and laptops keep radios in full-power mode a large fraction of the time. For example, an email application could constitute a significant source of energy consumption for syncing with mail server in stand-by mode [68].

Unlike previous studies [22, 38, 54], where the efforts primarily focus on traffic scheduling at the access point (AP) or nodes for reduced energy consumption, we take a dramatically different approach in this dissertation by bringing in a widely used power saving technique from other domains to WiFi radios. Based on our unique observations on the structure of WiFi signals, we manage to bypass the clock-rate constraint imposed by Nyquist sampling theory. More specifically, we design and implement a compressive sensing based WiFi 802.11b transceiver, SloMo, that is able to successfully communicate with existing WiFi networks while reducing the clock rate by a factor of five. Furthermore, our SloMo system is entirely backwards compatible and requires *zero* modification from existing WiFi infrastructure to be deployed. Realizing the rate limitation of SloMo, we further extend the benefit of downclocking to orthogonal frequency division multiplexing (OFDM) systems (e.g., 802.11a/g/n/ac) and explore the aliasing effect resulting from undersampling. We observe that aliasing effectively transforms the original quadrature amplitude modulation (QAM) into a denser but still decodable QAM scheme. Leveraging this observation, our Enfold design significantly advances the state of the art by demonstrating how to decode non-sparse signals below the Nyquist rate.

We implement both SloMo and Enfold in a software-defined radio and show that SloMo can seamlessly communicate with commercial WiFi chipsets. In addition, we verify that the design of Enfold is completely standards-compliant. To make our approaches readily deployable, we spend substantial effort addressing the intricacies of practical communication system design, such as timing synchronization, frequency and phase compensation, channel estimation, etc. Given that our approaches explicitly trade signal-to-noise ratio (SNR) for energy savings, we thoroughly evaluate the impact on communication performance due to downclocking. We empirically show that the packet reception rate (PRR) performance of SloMo and Enfold are nearly identical to their 100% clock rate counterparts for typical SNR conditions in operational WiFi networks. Finally, to understand the energy saving benefits enabled by downclocking, we capture WiFi network traces for a wide range of popular smartphone apps (each has more than five million downloads), ranging from low data rate app, like email to bandwidth hungry apps such as FaceTime and Skype video. When evaluated over the captured traces, our proposed design reduces energy consumption substantially by up to 34% even under a deliberately conservative power model.

In conclusion, this dissertation demonstrates that a downclocked WiFi radio is both practical and beneficial. In his forty data communication research questions, Partridge [44] listed designing wireless radios with multiple power levels as one of the fundamental challenges. Our downclocking techniques are a first step towards realizing this vision.

6.1 Future Directions

There are multiple directions how this dissertation could be extended. First, our approaches introduce a new state (reduced power consumption with potentially limited data rates) into the existing WiFi power management framework. It is up to the system designers how this new downclocked state can be best utilized to reduce energy consumption. Second, it is interesting to study the potential communication-performance improvements of Enfold by combining multiple-input-multiple-output (MIMO) techniques. Finally, we are excited to see how our approaches can be applied in other battery-constrained communication systems as the techniques developed in this dissertation are not tied to WiFi specifics.

Power Management. Essentially, both SloMo and Enfold introduce a new state into the existing WiFi power model: reduced power consumption with potentially limited data rate. The multiple downclocking rates provide system designers with a new knob where they can adjust the power consumption and network throughput at a fine granularity, as opposed to the current all-or-nothing power model in WiFi. We foresee a number of possible ways in how the existing power management policy could be re-designed. For example, downclocked state could be treated as a “doorbell” mode where the AP can send a special packet to wake the WiFi radio up to the full-rate state. More specifically, a node stays in “doorbell” mode for beacon listening, AP scanning, (re)association, etc. Alternatively, we could consider the downclocked state as an integral part of both power management and rate selection. Based on our energy study, downclocking is more power efficient when the app data rate is only a fraction of the current physical layer capacity. Therefore, we envision a power management policy that would dynamically migrate WiFi power state based on the current network demand by moving to higher clock rate as demand increases.

Combination with MIMO. MIMO is the primary physical layer advancement in the newer WiFi generations, i.e., 802.11n/ac. Enfold could leverage the channel sounding process, which is part of the standard MIMO operations, to improve its PRR performance. As discussed in this dissertation, the communication performance of Enfold largely depends on the per-subcarrier channel response. With channel sounding and precoding, the AP could purposely code each subcarrier with different power and constellation points such that the resulting QAM diagram under aliasing is maximally separated. Given that the AP is not power-constrained in general, it could make use of multiple antenna beams to further enhance the PRR performance.

Extension to other domains. Our downclocking approaches work directly with physical layer signals, and are not tied to WiFi specifics. Both direct sequence spread spectrum (DSSS) and OFDM are popular modulation techniques used in other domains as well. For example, smart meters and sensor networks (e.g., ZigBee) are DSSS-based systems while OFDM is the dominant modulation scheme for cellular and 60-Ghz communication. Therefore, it remains an interesting open problem to apply downclocking in these systems since they are also typically battery-constrained.

Bibliography

[1] AT&T Enhanced Push-To-Talk Deployment Guide. http://www.business.att.com/content/other/wi-fi-for-website.pdf.

[2] Best practice: Planning Your Organization's Wi-Fi Infrastructure. http://docs.blackberry.com/en/admin/deliverables/20034/BBMVS_BP_WiFi_infrastructure_1177026_11.jsp.

[3] Cisco 7920 Wireless IP Phone Design and Deployment Guide. http://www.mwtn.net/ASSETS/MANUALS/cisco_7920.pdf.

[4] Csipsimple Bug Fix. http://code.google.com/p/csipsimple/source/detail?r=153.

[5] New IEEE 802.11ac Specification Driven By Evolving Market Need for Higher, Multi-User Throughput In Wireless LANs. http://standards.ieee.org/news/2014/ieee_802_11ac_ballot.html.

[6] Sora SDK Manual. http://research.microsoft.com/apps/pubs/default.aspx?id=160799.

[7] Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications amendment 8: Medium Access Control (MAC) Quality of Service Enhancements, IEEE 802.11e, 2005.

[8] Yuvraj Agarwal, Ranveer Chandra, Alec Wolman, Paramvir Bahl, Kevin Chin, and Rajesh Gupta. Wireless Wakeups Revisited: Energy Management for VoIP over Wi-Fi Smartphones. In *Proceedings of ACM MobiSys*, June 2007.

[9] Manish Anand, Edmund Nightingale, and Jason Flinn. Self-Tuning Wireless Network Power Management. In *Proceedings of ACM MobiCom*, September 2003.

[10] Ganesh Ananthanarayanan and Ion Stoica. Blue-Fi: Enhancing Wi-Fi Performance using Bluetooth Signals. In *Proceedings of ACM MobiSys*, June 2009.

[11] Android Code Project: Add WIFI-MODE-FULL-HIGH-PERF (or Similar) to Official SDK. http://code.google.com/p/android/issues/detail?id=15549, March 2011.

[12] Android Developer Reference: WifiManager. http://developer.android.com/reference/android/net/wifi/WifiManager.html.

[13] Ranveer Chandra, Ratul Mahajan, Thomas Moscibroda, Ramya Raghavendra, and Paramvir Bahl. A Case for Adapting Channel Width in Wireless Networks. In *Proceedings of ACM SIGCOMM*, August 2008.

[14] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks. *Wireless Networks*, 8(5):481–494, September 2002.

[15] Xian Chen, Ruofan Jin, Kyoungwon Suh, Bing Wang, and Wei Wei. Network Performance of Smart Mobile Handhelds in a University Campus WiFi Network. In *Proceedings of ACM IMC*, November 2012.

[16] Yu-Chung Cheng, Mikhail Afanasyev, Patrick Verkaik, Péter Benkö, Jennifer Chiang, Alex C. Snoeren, Stefan Savage, and Geoffrey M. Voelker. Automating Cross-Layer Diagnosis of Enterprise Wireless Networks. In *Proceedings of ACM SIGCOMM*, August 2007.

[17] CNET. Raziko for Android: Full Specs. http://download.cnet.com/Raziko-for-Android/3000-2168_4-12094384.html.

[18] Robert G. Cole and JH Rosenbluth. Voice over IP Performance Monitoring. *SIGCOMM Computer Communications Review*, 31(2):9–24, April 2001.

[19] Mark A. Davenport, Petros T. Boufounos, Michael B. Wakin, and Richard G. Baraniuk. Signal Processing With Compressive Measurements. *IEEE Journal of Selected Topics in Signal Processing*, 4(2):445–460, April 2010.

[20] William R. Dieter, Srabosti Datta, and Wong Key Kai. Power Reduction by Varying Sampling Rate. In *Proceedings of IEEE ISLPED*, August 2005.

[21] Lijing Ding and Rafik A. Goubran. Speech Quality Prediction in VoIP Using the Extended E-Model. In *Proceedings of IEEE GLOBECOM*, December 2003.

[22] Fahad R. Dogar, Peter Steenkiste, and Konstantina Papagiannaki. Catnap: Exploiting High Bandwidth Wireless Interfaces to Save Energy for Mobile Devices. In *Proceedings of ACM MobiSys*, June 2010.

[23] Ettus Research LLC. Universal Software Radio Peripheral (USRP). http://www.ettus.com/.

[24] FitBit Aria: Technical Specs. http://www.fitbit.com/product/aria/specs.

[25] GE Healthcare Dash 5000 Patient Monitors. http://www3.gehealthcare.com/en/Products/Categories/Patient_Monitoring/Patient_Monitors/Dash_5000, November 2011.

[26] Kinshuk Govil, Edwin Chan, and Hal Wasserman. Comparing Algorithm for Dynamic Speed-setting of a Low-power CPU. In *Proceedings of ACM MobiCom*, November 1995.

[27] Gupta, Arpit and Min, Jeongki and Rhee, Injong. WiFox: Scaling WiFi Performance for Large Audience Environments. In *Proceeding of ACM CoNEXT*, December 2012.

[28] Daniel Halperin, Ben Greenstein, Anmol Seth, and David Wetherall. Demystifying 802.11n Power Consumption. In *Proceedings of USENIX HotPower*, October 2010.

[29] Dongsu Han, Aditiya Agarwala, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, and Srinivasan Seshan. Mark-and-Sweep: Getting the ”Inside” Scoop on Neighborhood Networks. In *Proceedings of ACM IMC*, October 2008.

[30] Haitham Hassanieh, Fadel Adib, Dina Katabi, and Piotr Indyk. Faster GPS via the Sparse Fourier Transform. In *Proceedings of ACM MobiCom*, August 2012.

[31] IEEE. IEEE 802.11a-1999: Higher Speed Physical Layer in the 5 GHz Band, 1999.

[32] IEEE. IEEE 802.11b-1999: Higher Speed Physical Layer Extension in the 2.4 GHz Band, 1999.

[33] IEEE. IEEE 802.11g-2003: Further Higher Data Rate Extension in the 2.4 GHz Band, 2003.

[34] Ronnie Krashinsky and Hari Balakrishnan. Minimizing Energy for Wireless Web Access with Bounded Slow Down. In *Proceedings of ACM MobiCom*, September 2002.

[35] Jungwon Lee, Hui-Ling Lou, Dimitris Toumpakaris, and John M. Cioffi. Effect of Carrier Frequency Offset on OFDM Systems for Multipath Fading Channels. In *Proceedings of IEEE GLOBECOM*, December 2004.

[36] Kyunghan Lee, Joohyun Lee, Yung Yi, Injong Rhee, and Song Chong. Mobile Data Offloading: How Much Can WiFi Deliver? In *Proceedings of ACM CoNEXT*, December 2010.

[37] Jiayang Liu and Lin Zhong. Micro Power Management of Active 802.11 Interfances. In *Proceedings of ACM MobiSys*, June 2008.

[38] Justin Manweiler and Romit Roy Choudhury. Avoiding the Rush Hours: WiFi Energy Management via Traffic Isolation. In *Proceedings of ACM MobiSys*, June 2011.

[39] Jia Meng, Wotao Yin, Yingying Li, N.T. Nguyen, and Zhu Han. Compressive Sensing Based High-Resolution Channel Estimation for OFDM System. *IEEE Journal of Selected Topics in Signal Processing*, 6(1):15–25, February 2012.

[40] Radhika Mittal, Aman Kansal, and Ranveer Chandra. Empowering Developers to Estimate App Energy Consumption. In *Proceedings of ACM MobiCom*, August 2012.

[41] Apurva N. Mody and Gordon L. Stuber. Synchronization for MIMO OFDM systems. In *Proceedings of IEEE GLOBECOM*, November 2001.

[42] Anthony Nicholson and Brian Noble. Breadcrumbs: Forecasting Mobile Connectivity. In *Proceedings of ACM MobiCom*, September 2008.

[43] Jeffrey Pang, Ben Greenstein, Michael Kaminsky, Damon McCoy, and Srinivasan Seshan. Wifi-Reports: Improving Wireless Network Selection with Collaboration. In *Proceedings of ACM MobiSys*, June 2009.

[44] Craig Partridge. Forty Data Communications Research Questions. *SIGCOMM Computer Communications Review*, 41(5):24–35, September 2011.

[45] Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. Where is the energy spent inside my app? Fine Grained Energy Accounting on Smartphones with Eprof. In *Proceedings of ACM EuroSys*, April 2012.

[46] Abhinav Pathak, Abhilash Jindal, Y. Charlie Hu, and Samuel P. Midkiff. What is Keeping My Phone Awake?: Characterizing and Detecting No-sleep Energy Bugs in Smartphone Apps. In *Proceedings of ACM Mobisys*, June 2012.

[47] Arogyaswami J. Paulraj, Dhananjay A. Gore, Rohit U. Nabar, and Helmut Bolcskei. An Overview of MIMO Communications - A Key to Gigabit Wireless. *Proceedings of the IEEE*, 92(2):198–218, February 2004.

[48] Francesco Ivan Di Piazza, Stefano Mangione, and Ilenia Tinnirello. On the Effects of Transmit Power Control on the Energy Consumption of WiFi Network Cards. In *Proceedings of QSHINE*, November 2009.

[49] Yvan Lamelas Polo, Ying Wang, Ashish Pandharipande, and Geert Leus. Compressive Wide-Band Spectrum Sensing. In *Proceedings of IEEE ICASSP*, April 2009.

[50] John G. Proakis and Masoud Salehi. *Digital Communications*. McGraw-Hill, November 2007.

[51] Andrew J. Pyles, Zhen Ren, Gang Zhou, and Xue Liu. SiFi: Exploiting VoIP Silence for WiFi Energy Savings in Smart Phones. In *Proceedings of ACM UbiComp*, September 2011.

[52] Jan N. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic. *Digital Integrated Circuits*. Prentice Hall, 2nd edition, January 2003.

[53] Ahmad Rahmati and Lin Zhong. Context-for-Wireless: Context-Sensitive Energy-Efficient Wireless Data Transfer. In *Proceedings of ACM MobiSys*, June 2007.

[54] Eric Rozner, Vishnu Navda, Ramachandran Ramjee, and Shravan Rayanchu. NAP-man: Network-Assisted Power Management for WiFi Devices. In *Proceedings of ACM MobiSys*, June 2010.

[55] Aaron Schulman, Dave Levin, and Neil Spring. On the Fidelity of 802.11 Packet Traces. In *Proceedings of PAM*, April 2008.

[56] Li Shang, Li-Shiuan Peh, and Niraj K. Jha. Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks. In *Proceedings of IEEE HPCA*, January 2003.

[57] Claude E. Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27:379423, July 1948.

[58] Claude E. Shannon. Communication in the Presence of Noise. *Proceedings of the Institute of Radio Engineers*, 37(1):10–21, January 1949.

[59] Eugine Shih, Paramvir Bahl, and Michael Sinclair. Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices. In *Proceedings of ACM MobiCom*, September 2002.

[60] Kun Tan, Jiansong Zhang, Haitao Wu, Fang Ji, He Liu, Yusheng Ye, Shen Wang, Yongguang Zhang, Wei Wang, and Geoffrey M. Voelker. Sora: High Performance Software Radio Using General Purpose Multi-core Processors. In *Proceedings of USENIX NSDI*, April 2009.

[61] Georg Taubock and Franz Hlawatsch. A Compressed Sensing Technique for OFDM Channel Estimation in Mobile Environments: Exploiting Channel Sparsity for Reducing Pilots. In *Proceedings of IEEE ICASSP*, April 2008.

[62] Joel A. Tropp, Jason N. Laska, Marco F. Duarte, Justin K. Romberg, and Richard G. Baraniuk. Beyond Nyquist: Efficient Sampling of Sparse Bandlimited Signals. *IEEE Transactions on Information Theory*, 56(1):520–544, January 2010.

[63] Patrick Verkaik, Yuvraj Agarwal, Rajesh Gupta, and Alex C. Snoeren. Softspeak: Making VoIP Play Well in Existing 802.11 Deployments. In *Proceedings of USENIX NSDI*, April 2009.

[64] Google Play Store: VirtualRadio — What's New. http://bit.ly/SVgzMG.

[65] Viterbi, Andrew J. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, April 1967.

[66] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for Reduced CPU Energy. In *Proceedings of OSDI*, November 1994.

[67] Wi-Fi HVAC Monitor. http://bit.ly/V6JNpn.

[68] Fengyuan Xu, Yunxin Liu, Thomas Moscibroda, Ranveer Chandra, Long Jin, Yongguang Zhang, and Qun Li. Optimizing Background Email Sync on Smartphones. In *Proceeding of ACM MobiSys*, June 2013.

[69] Kun-Wah Yip, Yik-Chung Wu, and Tung-Sang Ng. Timing-Synchronization Analysis for IEEE 802.11a Wireless LANs in Frequency-Nonselective Rician Fading Environments. *IEEE Transactions on Wireless Communications*, 3(2):387–394, March 2004.

[70] Xinyu Zhang and Kang G. Shin. E-MiLi: Energy-Minimizing Idle Listening in Wireless Networks. In *Proceedings of ACM MobiCom*, September 2011.