SloMo: Downclocking WiFi Communication

Feng Lu, Geoffrey M. Voelker, and Alex C. Snoeren

Department of Computer Science and Engineering
University of California, San Diego

Abstract

As manufacturers continue to improve the energy efficiency of battery-powered wireless devices, WiFi has become one of—if not the—most significant power draws. Hence, modern devices fastidiously manage their radios, shifting into low-power listening or sleep states whenever possible. The fundamental limitation with this approach, however, is that the radio is incapable of transmitting or receiving unless it is fully powered. Unfortunately, applications found on today's wireless devices often require frequent access to the channel.

We observe, however, that many of these same applications have relatively low bandwidth requirements. Leveraging the inherent sparsity in Direct Sequence Spread Spectrum (DSSS) modulation, we propose a transceiver design based on compressive sensing that allows WiFi devices to operate their radios at lower clock rates when receiving and transmitting at low bit rates, thus consuming less power. We have implemented our 802.11b-based design in a software radio platform, and show that it seamlessly interacts with existing WiFi deployments. Our prototype remains fully functional when the clock rate is reduced by a factor of five, potentially reducing power consumption by over 30%.

1 Introduction

Smartphones and other battery-powered wireless devices are becoming increasingly popular platforms for all manner of network applications. As a result, the energy usage of the radios on these devices is a source of considerable concern. Unsurprisingly, a large number of techniques have been proposed to help manage the power consumption of both cellular and WiFi devices. Focusing particularly on the WiFi domain, the basic approach has been to implement extremely low-power listening or sleep modes, and transition the devices into operational mode as little as possible [12, 18, 27]. The fundamental limitation with such approaches, however, is that the radio is incapable of transmitting or receiving unless it is fully powered. Unfortunately, recent studies have shown that a wide variety of popular applications make frequent and persistent use of the network [21], 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 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 [18, 39]. 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 [18, 21]. To amortize these costs, the 802.11 PSM specification has nodes wake up at the granularity of the 100-ms AP beacon interval when they do not have packets to transmit. (Indeed, the popular Nexus One wakes up only every 300 ms [18].) Hence, while useful for bulk data transfers [12] or situations where traffic pattens can be predicted precisely [24], PSM-style power saving approaches are often ineffective for applications that need to send or receive data frequently [39].

In this paper, we consider an alternative to the traditional on/off model. Instead, we explore a technique that reduces the power consumption of the WiFi chipset across all of its operating modes: i.e., not just sleep and listen, but send and receive as well. Our approach leverages the excess channel capacity provided by many WiFi networks when compared to the bandwidth demands of most smartphone applications. 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. We observe that one can instead turn excess channel capacity into an opportunity to save power at the *receiver*.

Our power savings comes from operating the WiFi chipset at a lower clock rate. Zhang and Shin demonstrated a wireless receiver that can be downclocked yet still detect packets [39]. We show how to allow transceivers to remain downclocked during reception and transmission as well. We propose a receiver design based on recent advances in compressive sensing [33] that takes

advantage of the inherent sparsity of the Direct Sequence Spread Spectrum (DSSS) modulation used by 802.11b. With our design, clients with low demand can operate their radios at a reduced clock rate while continuing to communicate with commercial WiFi devices.

We have implemented a prototype of our 802.11b-based design, called SloMo, in the Sora software radio platform. We show that SloMo seamlessly communicates with multiple vendors' commercial chipsets using standard 802.11b frames. Our measurements of frame reception rates demonstrate that SloMo remains fully functional even when the clock rate is reduced by more than a factor five. Our trace-based simulations across a range of popular smartphone applications show that SloMo reduces WiFi power consumption by up to 30–34% on the iPhone 4S and Nexus S, respectively. Moreover, SloMo outperforms two other proposed approaches, U-APSD and E-MiLi, in almost all cases.

2 Related work

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 PSM, 2) systems that duty cycle the WiFi device, and 3) attempts to decease transmit power.

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 [27]. Others have proposed dynamically adjusting sleep periods based upon a client's traffic pattern [2, 16]. 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 [17], relying upon 802.11 devices retransmitting unacknowledged frames to limit losses. Catnap [12] 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. 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 [18].

Finally, even otherwise-effective power saving mechanisms implemented by the WiFi chipset can be overridden by applications in many popular frameworks [4, 5]: some apps prevent the WiFi device from entering PSM mode, forcing the WiFi card to stay awake in an effort to improve performance [9, 35]. Because SloMo decreases power consumption across all WiFi states, it can still reduce energy consumption in these cases.

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 [7], 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 [1, 31]. 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 [3] or cell towers [26], or historical device movement patterns [20].

Limited transmit power. 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 [15, 22].

Downclocking. We take a radically different approach by enabling the radio to communicate while in a low-power state. Our efforts are inspired by previous observations that radios can conserve power by operating at lower clock rates. Researchers have argued that devices could dynamically adjust their sampling rate based upon the frequencies contained within the observed signal [11], but their approach is not directly applicable to the encoding schemes employed by WiFi. 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 [6] (although stations operating in different bandwidths cannot decode each other's transmission and the 17-ms switching overhead makes co-existence challenging), and develop mechanisms to detect packet arrivals in a downclocked state [39]. Downclocking a receiver through dynamic frequency scaling has been applied in the wireline context in the past [29], but we are not aware of any similar schemes in the wireless domain.

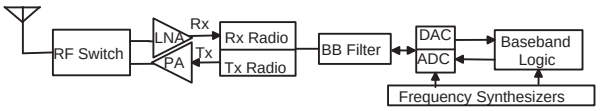


Figure 1: A simplified WiFi card architecture.

3 Motivation

As wireless link speeds continue to increase, mobile devices are increasingly likely to want to use only a small fraction of the channel capacity. With WiFi, however, use of the network is an all-or-nothing affair in terms of power: if a transceiver is not fully powered, no data can be sent or received.

3.1 The potential of downclocking

The power consumption of a CMOS computing device is proportional to its clock rate [25]. Not surprisingly, dynamic frequency scaling (DFS) has long been used as a technique to save power in a variety of computing domains [36]. 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 Nyquist theorem dictates that to successfully receive a signal, the receiver must sample the channel at twice the bandwidth of the signal [30]. In practice, 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 1 shows a typical WiFi transceiver architecture. 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 frame to the host. The entire radio card 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 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. Previous studies have shown that the power consumption of popular WiFi chipsets (e.g., from Atheros and Netgear) does indeed vary with frequency [6, 39], although the precise relationship depends on what the device is doing (sending frames, receiving frames, or idling) and differs across chipsets. As an example, Table 1 shows the reported energy consumption of a popular WiFi chipset while operating at various clock rates [39].

Not surprisingly, the power savings are sub-linear (40% savings while receiving packets at a 25% clock rate), but they are still substantial. However, current devices were not designed to be downclocked. Hence, it is unlikely they are optimized to be power-efficient at frequencies other than their target operating point.

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

Table 1: Power draw of the Atheros 5414 WiFi chipset in the LinkSys WPC55AG NIC at various clock rates [39].

3.2 Downclocked transmission

It is not obvious that downclocking a radio would be beneficial while transmitting data: the lower the data rate, the longer the transmission takes. Hence, in theory 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 the device only receives data, the 802.11 specification requires that it transmit an ACK frame to confirm receipt of the data frame—and the ACK frame must be sent within a strict, 20-μs interframe 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 [39]). In such cases, to realize the benefits of downclocked reception, the transceiver needs to transmit at a slower clock rate an ACK frame that a standard-compliant WiFi transmitter will accept. (The Rx power draws in Table 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 [18, 39]. 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 1, the measured Atheros chipset consumes 47% less power in idle listening mode when downclocked by a factor of 4.

However, once the channel is detected to be idle, a WiFi station needs to attempt to transmit a frame within 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, in order to perform CCA while downclocked, the WiFi device must be prepared to transmit while downclocked as well.

3.3 Network impacts

Clearly, downclocked nodes have the potential to realize significant power savings. An obvious concern, however, 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 [34]—its likelihood depends both on the background usage level in the network and the communication patterns of the downclocked node.

For example, for 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 [34]. 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 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 have studied, the impact on free channel airtime is limited ($\leq 16\%$, see Section 6.3). 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 [8].

4 Downclocked 802.11b

In this section we describe the design of SloMo, our prototype downclocked radio for 802.11b. SloMo can fully interoperate with standard-compliant WiFi devices (i.e., 802.11a/b/g/n/ac) at both 1 and 2-Mbps DSSS rates, with no modifications to the access point. While these data rates are admittedly modest, we show later that they suffice for many popular applications. Further, the 802.11b rates remain widely supported in both deployed WiFi networks and the upcoming 802.11ac chipsets (e.g., Broadcom 4335) and routers (e.g., Cisco EA6500). Indeed, due to its robust communication range and low cost, 802.11b is the only supported WiFi mode in some special-purpose devices [13, 14, 37].

4.1 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 (CS) decoding scheme. DSSS and complementary code keying (CCK) are the two modulation techniques specified in the IEEE 802.11b standard. 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 (BPSK) and quadrature phase shift keying (QPSK), respectively. To ease our explanation, we will 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 [33]. 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 signal to noise ratio (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.

4.1.1 DSSS modulation

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 11-chip Barker sequence to expand the signal. The spreading process serves several

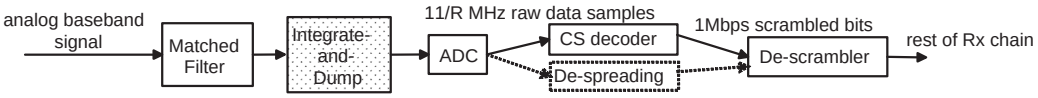


Figure 2: Original and modified baseband Rx processing chain. Compared to the original Rx chain, the modified chain adds an additional integrate-and-dump component and replaces the De-spreading part with the CS decoder.

purposes. First, it enlarges the spectrum of the original baseband signal by $11\times$ to make it more robust to channel noise. Secondly, due to the unique 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 11-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 ($b_1 = [\mathbf{0\ 1}]^\mathbf{T}$ corresponds to a 1 and $b_0 = [\mathbf{1\ 0}]^\mathbf{T}$ 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 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, 11 sample values will be collected corresponding to the 11-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.

4.1.2 Compressive sensing

We implement compressive sensing using an integrate-and-dump sampler as suggested by Tropp *et al.* [33]. 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 $\mathbf{H}$:

$$\mathbf{Y} = \mathbf{H}\tilde{\mathbf{C}}. \tag{1}$$

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-dumper sampler, the measurements can be viewed as a linear combination of the original chip values. For example, suppose only 3 measurements are desired (i.e., a downclocking ratio of 3/11). The measurements can be viewed as substituting a compressive measurement matrix into Equation 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 11 cannot be evenly divided by 3, 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$\times$, we choose to take two samples of 4 chips and one of 3. 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 [10]:

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

[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.

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 2. Since only a single bit is decoded at a time, the decision rule can be simplified as

$$d_i = \mathbf{Y^T}(\mathbf{HH^T})^{-1}(\mathbf{HMb_i}). \quad (2)$$

4.2 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 11-bit Barker sequence, a bit boundary is within the next 10 samples of any chip. Hence, the matched filter simply correlates the chip samples at each of these 11 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 11; 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. We omit the detailed mathematical steps involved to search for these sequences. At a high level, we use correlation peak-to-average ratio as a close approximation to decide how good the code sequence is. Figure 3 shows some examples of the Barker-like sequences we obtain, and how they compare to the original 11-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 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.

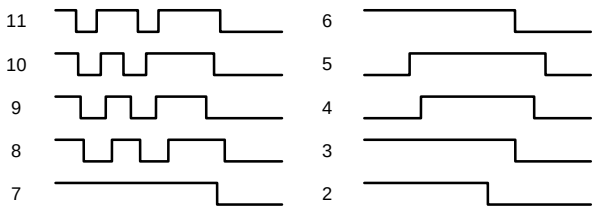


Figure 3: Spreading sequences of various lengths, including the 802.11-standard 11-chip Barker sequence and the Barker-like sequence used by SloMo when downclocked accordingly.

4.3 Practical design considerations

In the previous 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 4.1.2). Here, we first relax the latter two requirements and then return to address the former.

4.3.1 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 9th 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 group.

At a raw data bit rate of 1 Mbps, Nyquist requires a minimum of two measurements per 11-bit chip sequence; we cannot downclock the receiver a full $11\times$. Hence, the useful range of integration lengths is between 2 and 5 chips. Since 11 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

[2]For example, Sora [32] decides the maximum value is a peak if the maximum value is at least twice the second maximum.

[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 $5.5\times$.

(i.e., 11 measurements). Therefore, rather than decoding one bit at a time, we jointly decode k bits in a group—which exactly corresponds to $11k$ chips.

4.3.2 Synchronization

Symbol synchronization is a fairly standard technique and is often implemented in hardware [23]. 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 11 chips, but our integrate-and-dump sampler has reduced these chips by a factor of k. Hence, we are always decoding exactly 11 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 11 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 11 compressed measurements starts a group, we store the decoding results for each possible position simultaneously. For implementation purposes, we keep 11 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 11 bit arrays. Meanwhile, we look for the fixed bit pattern of the Start of Frame Delimiter (SFD) among the 11 stored bit arrays. Once the SFD is identified in one of the arrays, we know the correct bit boundary and we only need to keep decoding in one of the 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.

4.4 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 basestation, as well the potential interactions with other client nodes due to its use of 802.11b (as opposed to 11g or 11n).

4.4.1 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 CCK encoding (i.e., 5.5 and 11-Mbps 802.11b frames) or communicate at 802.11g/n rates. Fortunately, the 802.11b standard includes mechanisms for the SloMo node to convey these constraints to the 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 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.

4.4.2 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 11g or 11n 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 CCA method, our SloMo node could co-exist with any other 11g/n 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 11b and 11g employ different inter-frame timings (for example, the slot time is 20 μs and 9 μs for 11b and 11g with protection mode off, respectively), 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.

5 Prototype

To assess the feasibility of our approach, we implement a prototype CS-based 802.11b transceiver architecture in Microsoft Sora, a fully programmable software defined radio [32]. 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.

5.1 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 modified the Sora code by adding the integrate-and-dump sampler in the receiver chain and re-design the bit decoding algorithm as described by Equation 2. We have implemented both versions of the integrate-and-dump sampler. 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 the phase tracking component to ensure correct decoding of multiple-bit groups.

5.2 Experimental configuration

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 8 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 2 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 link rates double application throughput as expected).

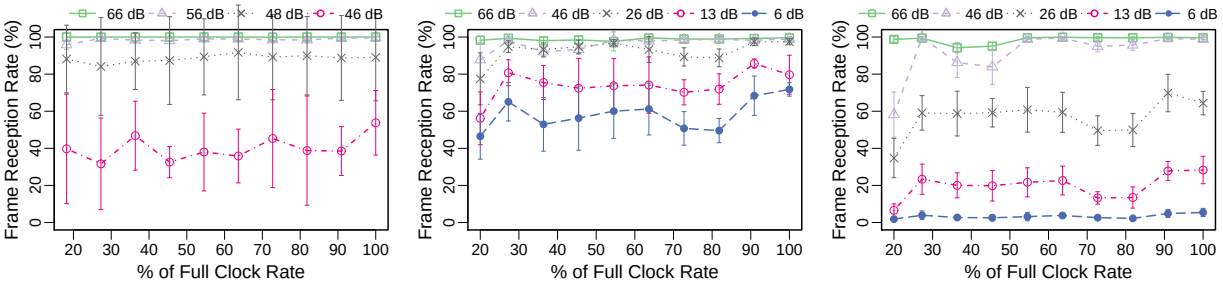
To experiment with different network conditions, we varied the distance and path between the nodes. We fixed the location of the SloMo node in a room, and moved the laptop to various locations inside the building.

5.3 Downclocked reception

We start by evaluating downclocked reception in isolation using compressive sensing (CS). We transmit packets using the commercial WiFi device on the laptop to our experimental Sora node, which receives them using CS with a configurable decoding clock rate. For each clock rate and location, we transmit 1,000 UDP packets (each 1,000-bytes long) paced to allow the network to settle between transmissions. We repeat each experiment 10 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 CS on the Sora node, and also report the corresponding SNR value for each location.

Figure 4(a) 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 10 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 3.1 that downclocking at such rates corresponds to more than 40% savings in power consumption for a popular 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.



(a) WiFi → SloMo (b) SloMo → WiFi (small packets) (c) SloMo → WiFi (large packets)

Figure 4: Frame reception rates at SloMo Sora node (commercial WiFi device) for packets sent by commercial WiFi device (SloMo Sora node) using downclocked compressive sensing reception (downclocked "Barker-like" transmission). As a baseline, the 100% clock rate corresponds to using the default 802.11b implementation.

5.4 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 the same methodology as with compressive sensing: 10 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 4(b) and 4(c) 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 using increasingly shorter Barker-like sequences (Figure 3); the lowest transmission clock rate is 20%, which corresponds to transmitting with just two chips (Section 4.2). As shown in Figure 4(b), 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 4(c), 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.

5.5 Further prototype experiments

We performed additional experiments with the SloMo prototype, which we summarize for space considerations. First, we combined downclocked reception and transmission to evaluate the quality of Skype VoIP communication using SloMo. We found that downclocked VoIP using SloMo only significantly degrades call quality when network conditions are poor, as expected, but otherwise delivers equivalent Mean Opinion Scores (MOS) for calls. To stress SloMo's downclocking implementation, we also evaluated application throughput at both 1 Mbps and 2 Mbps link rates using `iperf` with 1,000-byte UDP packets. The 1-Mbps results track the packet reception results in Figure 4(a) very closely. SloMo can also take full advantage of 2-Mbps link rates under stable network conditions: application throughputs at 2 Mbps are double those at 1 Mbps. Finally, in addition to evaluating SloMo with the Intel WiFi card, we also performed similar throughput experiments between the Sora node running SloMo and a Macbook Pro laptop with an Apple Airport Extreme WiFi card using the Broadcom BCM43xx firmware. Both downclocked re-

ception and transmission performed as expected between SloMo and the Macbook.

6 Trace-based energy evaluation

Our experiments with the SloMo implementation demonstrate the feasibility and performance of downclocked 802.11 communication. Next we evaluate the potential energy savings when using downclocking in the context of contemporary smartphones and popular apps.

6.1 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 MAC-layer packet traces of a variety of real apps running on two different smartphones. We use these packet 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.

6.1.1 WiFi power model

Similar to Mittal *et al.* [19], we parameterize our smartphone WiFi power model on the measurements of a Nexus One reported by Manweiler and Choudhury [18]. 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. Table 2 summarizes the model parameters we used. Most are reproduced from [18, 19]; 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 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. 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.

Parameter	Full Clock / Downclocked (1/4)
Beacon Interval (ms)	100 / 100
Beacon Wakup Period (ms)[4]	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*

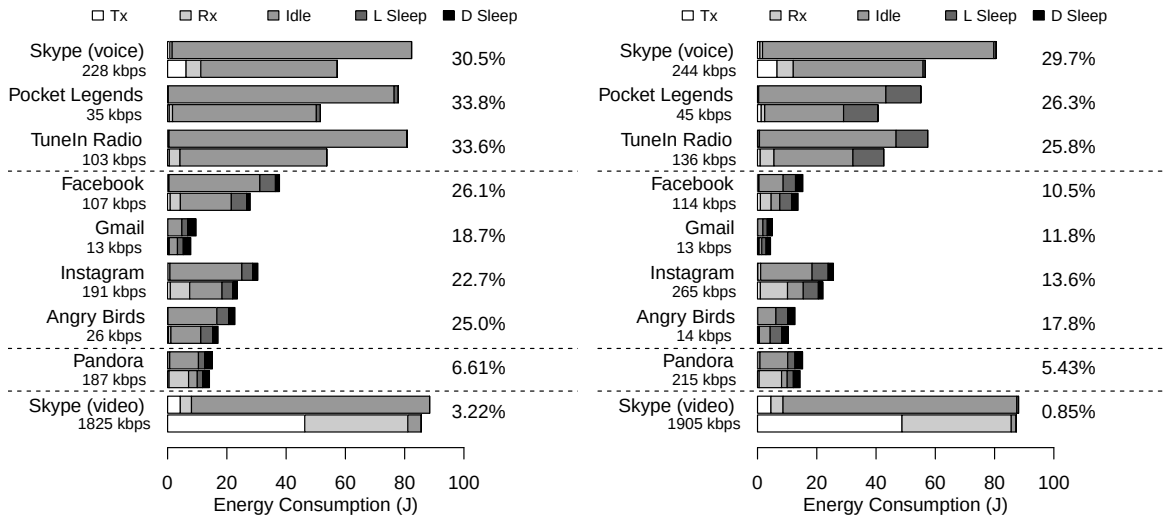
Table 2: WiFi Power Characteristics

6.1.2 Smartphone app traces

To comprehensively evaluate the benefits of SloMo, we sampled a wide range of popular smartphone apps (each has at least 5 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 [28] by configuring two MacBook Pro laptops as sniffer nodes in the vicinity of the smartphone and the 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 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.

[4] 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 [18]; we use 2.5 ms in our model to be conservative.



(a) Google Nexus S

(b) iPhone 4S

Figure 5: Energy cost of various apps under 802.11 PSM and SloMo. For each app, the upper bar corresponds to the breakdown of energy consumption under 802.11 PSM while the lower bar corresponds to SloMo. 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.

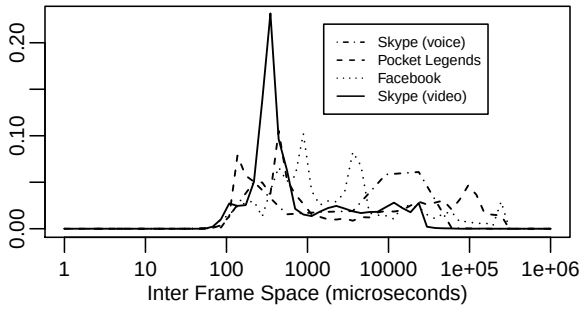


Figure 6: PDFs of the IFT for a selected set of apps on Neuxs S. We remove the inter frame time (SIFS) between DATA and ACK frame for better presentation. IFTs larger than a sleep period are also removed.

6.2 SloMo energy consumption

Figures 5 and 6 combined show the energy and timing behavior of the apps. Figure 5 compares the network energy costs of the apps by power state when using standard 802.11 PSM (top bar) and when using downclocked communication with SloMo (bottom bar). 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.

Figure 5 shows 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 6 shows the 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 2), downclocking significantly reduces idle state energy consumption [39]. And since these apps have low data rates, the energy saved during idle listening far exceeds the additional energy consumed for slower data 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 6 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 [21]. (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 10 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 tradeoff 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.

6.3 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. As discussed in Section 3.3, an app might save itself energy by downclocking but unduly impact other devices on the network by consuming more airtime using lower data rates.

Figure 7 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. The channel airtime results for the iPhone 4S are very similar (the largest contraction ratio is 1.16 for apps other than Skype Video).

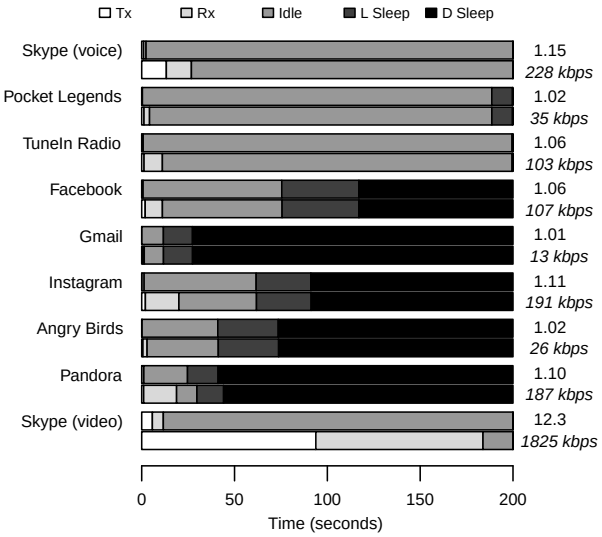


Figure 7: Comparing the timing breakdown for various apps under 802.11 PSM (upper bar) and SloMo on Google Nexus S (lower bar). The number at the end of the bar group shows the free channel airtime contraction ratio, the lower the better with 1.0 as the minimum.

6.4 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 [38] and E-MiLi [39], 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 [38]) could allow devices to enter the sleep state immediately after network activity and avoid the stan-

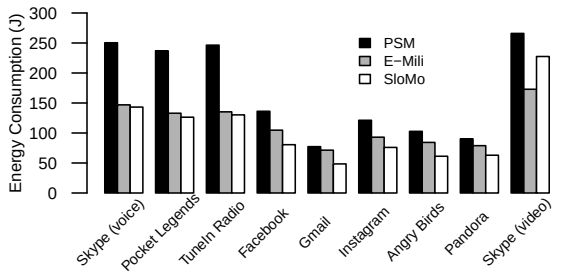


Figure 8: Comparing energy consumption among PSM, E-MiLi and SloMo on the Nexus trace set.

dard tail time in the idle state. Based upon the U-APSD specification, we emulated its use[5] for the Skype Voice and 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 [24] 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 [27].

E-MiLi. E-MiLi redesigns the addressing mechanism of WiFi devices, enabling receivers to determine whether traffic is addressed to them without leaving a low-power listening state [39]. 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.[6] To facilitate the comparison, we apply the E-MiLi power model to SloMo in contrast to our previous experiments.[7] Figure 8 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.

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

[6]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.

[7]As a result, the SloMo energy savings are 10–20% larger relative to PSM compared to the results presented in Figure 5(a).

7 Conclusion

Downclocked 802.11 reception and transmission using compressive sensing is both beneficial and practical. Analysis of the network traffic of a wide range of popular smartphone apps shows that downclocking has the potential to reduce WiFi power consumption on contemporary smartphones by 30%. And our SloMo prototype shows the practicality of implementing downclocking on WiFi clients that communicate seamlessly with unmodified commercial WiFi hardware. While SloMo demonstrates that compressive-sensing techniques are effective for the DSSS encoding used by 802.11b, consumer devices are increasingly employing the higher-rate encodings of 802.11a/g/n. Since the OFDM-based modulation scheme used by these protocols does not share the same inherent spectral sparsity, a tantalizing challenge going forward is to what extent alternative techniques can achieve the same goals for these modulations.

Acknowledgments

We would like to thank Justin Manweiler and Romit Roy Choudhury for providing additional details about their power measurements. We would also like to thank Heather Zheng, our shepherd, and the anonymous reviewers for their detailed feedback. Finally, we are grateful to Lijuan Geng for her assistance with some of the experiments, and Kun Tan for his support and assistance with Sora. This work was supported by generous research, operational and in-kind support from the UCSD Center for Networked Systems (CNS).

References

[1] Y. Agarwal, R. Chandra, A. Wolman, P. Bahl, K. Chin, and R. Gupta. Wireless Wakeups Revisited: Energy Management for VoIP over Wi-Fi Smartphones. In *Proceedings of ACM MobiSys*, June 2007.

[2] M. Anand, E. Nightingale, and J. Flinn. Self-Tuning Wireless Network Power Management. In *Proceedings of ACM MobiCom*, Sept. 2003.

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

[4] Android Code Project: Add WIFI-MODE-FULL-HIGH-PERF (or similar) to official SDK. `http://code.google.com/p/android/issues/detail?id=15549`, Mar. 2011.

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

[6] R. Chandra, R. Mahajan, T. Moscibroda, R. Raghavendra, and P. Bahl. A Case for Adapting Channel Width in Wireless Networks. In *Proceedings of ACM SIGCOMM*, Aug. 2008.

[7] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks. *Wireless Networks*, 8(5), Sept. 2002.

[8] Y.-C. Cheng, M. Afanasyev, P. Verkaik, P. Benkö, J. Chiang, A. C. Snoeren, S. Savage, and G. M. Voelker. Automating Cross-Layer Diagnosis of Enterprise Wireless Networks. In *Proceedings of ACM SIGCOMM*, Aug. 2007.

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

[10] M. Davenport, P. Boufounos, M. Wakin, and R. Baraniuk. Signal Processing With Compressive Measurements. *IEEE Journal of Selected Topics in Signal Processing*, 4(2):445–460, Apr. 2010.

[11] W. R. Dieter, S. Datta, and W. K. Kai. Power Reduction by Varying Sampling Rate. In *Proceedings of the IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pages 227–232, Aug. 2005.

[12] F. R. Dogar, P. Steenkiste, and K. Papagiannaki. Catnap: Exploiting High Bandwidth Wireless Interfaces to Save Energy for Mobile Devices. In *Proceedings of the ACM MobiSys*, June 2010.

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

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

[15] D. Halperin, B. Greenstein, A. Seth, and D. Wetherall. Demystifying 802.11n Power Consumption. In *Proceedings of USENIX HotPower*, Oct. 2010.

[16] R. Krashinsky and H. Balakrishnan. Minimizing Energy for Wireless Web Access with Bounded Slow Down. In *Proceedings of ACM MobiCom*, Sept. 2002.

[17] J. Liu and L. Zhong. Micro Power Management of Active 802.11 Interfances. In *Proceedings of ACM MobiSys*, June 2008.

[18] J. Manweiler and R. R. Choudhury. Avoiding the Rush Hours: WiFi Energy Management via Traffic Isolation. In *Proceedings of ACM MobiSys*, June 2011.

[19] R. Mittal, A. Kansal, and R. Chandra. Empowering Developers to Estimate App Energy Consumption. In *Proceedings of ACM MobiCom*, Aug. 2012.

[20] A. Nicholson and B. Noble. Breadcrumbs: Forecasting Mobile Connectivity. In *Proceedings of ACM MobiCom*, Sept. 2008.

[21] A. Pathak, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app? Fine Grained Energy Accounting on Smartphones with Eprof. In *Proceedings of ACM EuroSys*, Apr. 2012.

[22] F. I. D. Piazza, S. Mangione, and I. Tinnirello. On the Effects of Transmit Power Control on the Energy Consumption of WiFi Network Cards. In *Proceedings of QSHINE*, Nov. 2009.

[23] J. G. Proakis and M. Salehi. *Digital Communications*. McGraw-Hill, Nov. 2007.

[24] A. J. Pyles, Z. Ren, G. Zhou, and X. Liu. SiFi: Exploiting VoIP Silence for WiFi Energy Savings in Smart Phones. In *Proceedings of ACM UbiComp*, Sept. 2011.

[25] J. N. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits*. Prentice Hall, 2nd edition, Jan. 2003.

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

[27] E. Rozner, V. Navda, R. Ramjee, and S. Rayanchu. NAPman: Network-Assisted Power Management for WiFi Devices. In *Proceedings of ACM MobiSys*, June 2010.

[28] A. Schulman, D. Levin, and N. Spring. On the Fidelity of 802.11 Packet Traces. In *Proceedings of the 9th PAM Conference*, Apr. 2008.

[29] L. Shang, L.-S. Peh, and N. K. Jha. Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks. In *Proceedings of IEEE HPCA*, Jan. 2003.

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

[31] E. Shih, P. Bahl, and M. Sinclair. Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices. In *Proceedings of ACM MobiCom*, Sept. 2002.

[32] K. Tan, J. Zhang, H. Wu, F. Ji, H. Liu, Y. Ye, S. Wang, Y. Zhang, W. Wang, and G. M. Voelker. Sora: High Performance Software Radio Using General Purpose Multicore Processors. In *Proceedings of NSDI*, Apr. 2009.

[33] J. Tropp, J. Laska, M. Duarte, J. Romberg, and R. Baraniuk. Beyond Nyquist: Efficient Sampling of Sparse Bandlimited Signals. *IEEE Transactions on Information Theory*, 56(1):520–544, Jan. 2010.

[34] P. Verkaik, Y. Agarwal, R. Gupta, and A. C. Snoeren. Softspeak: Making VoIP Play Well in Existing 802.11 Deployments. In *Proceedings of NSDI*, Apr. 2009.

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

[36] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. In *Proceedings of OSDI*, Nov. 1994.

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

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

[39] X. Zhang and K. G. Shin. E-MiLi: Energy-Minimizing Idle Listening in Wireless Networks. In *Proceedings of ACM MobiCom*, Sept. 2011.