Enfold: Downclocking OFDM in WiFi

Feng Lu, Patrick Ling, Geoffrey M. Voelker and Alex C. Snoeren
University of California, San Diego

Abstract

Dynamic voltage and frequency scaling (DVFS) has long been used as a technique to save power in a variety of computing domains but typically not in communications devices. A fundamental limit that prevents decreasing the clock frequency is the Nyquist(-Shannon) sampling theorem, which states that the sampling rate must be twice the signal bandwidth. Recently, researchers have leveraged compressive sensing to demonstrate the possibility of decoding a sparse signal below Nyquist rate. In this work, we dramatically extend the state of the art by showing how to decode non-sparse signals, in particular, OFDM systems at sub-Nyquist rates. 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. Based on our observations, we present Enfold, which allows existing WiFi chipsets to decode standards-compliant WiFi frames while operating at 50% and 25% of their rated clock rate. Our design is able to attain greater than 96% and 83% raw packet reception rates for moderate SNR while reducing the clock rate by $2\times$ and $4\times$, respectively. Moreover, our approach can be easily applied to other communication systems based on OFDM modulation. When evaluated on popular smartphone app traces, Enfold reduces energy consumption by up to 34%.

Categories and Subject Descriptors

C.2.1 [**Computer Communication Networks**]: Network Architecture and Design—*Wireless Communication*

General Terms

Algorithms, Design, Experimentation, Measurement, Performance

Keywords

Downclocking; OFDM; Energy Efficiency; WiFi; DVFS

1. INTRODUCTION

The link rate of popular wireless technologies, such as 802.11, has increased markedly over the past two decades: 802.11ac promises near-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 [13, 23] that many common applications for both smartphones and laptops keep radios in full-power mode a large fraction of the time.

Given the importance of energy efficiency in power-constrained environments like battery-powered laptops, smartphones, and the like, a number of recent research efforts have focused on bringing a widely used technique for energy savings in CMOS, namely dynamic voltage and frequency scaling (DVFS), to standards-compliant radios. In particular, researchers have demonstrated that it is possible to downclock a radio when channel quality is sufficiently high, yet still detect [23], receive, and even transmit [13] WiFi frames. Modern WiFi chipsets typically have a single clock driving both RF and baseband. The potential power savings are significant—over 40% in some cases—yet the existing systems have limited application. Specifically, they either require modifying the 802.11 standard [23], or limit communication to 1 or 2 Mbps (DSSS) encodings [13].

In this work, we demonstrate that it is feasible to successfully receive and decode OFDM modulation while sampling below the Nyquist frequency. In particular, we show that a standards-compliant 802.11a/g frame transmitted at 6 or 9 Mbps can be detected, received, and decoded by a receiver running at a 50% or even 25% clock rate given sufficient channel quality. While our implementation focuses specifically on WiFi to demonstrate that we are able to overcome the various challenges in addition to decoding, such as synchronization, frequency offset and phase recovery, our basic approach can be applied to any communications system based upon OFDM modulation, significantly broadening the applicability of downclocking when compared to the state of the art.

Technically, our approach is based on exploiting the aliasing that is inherent in under-sampling an OFDM signal. Unlike previous work, which assumed the signal was sparse, OFDM signals are extremely dense and, therefore, downclocking yields significant interference across subcarriers. We show, however, that the particular structure of this interference can be used to convert the modulation into a more complicated, but still decodable, form. In particular, by folding the subcarriers on top of each other, we decode frames transmitted with a form of QAM as if they were encoded with a more dense QAM modulation.

This paper makes three main contributions:

- We explore the fundamental structure of downsampled OFDM encodings, and demonstrate that it can be leveraged to decode QAM-based modulations at both half and quarter clock rates.

- We design and implement a standards-compliant WiFi receiver based on the Microsoft Sora software radio platform

MobiCom'14, September 7-11, 2014, Maui, Hawaii, USA.

ACM 978-1-4503-2783-1/14/09 ...$15.00.
http://dx.doi.org/10.1145/2639108.2639123.

that is able to inter-operate with commercial WiFi chipsets while using downclocked reception.

- Through experimentation with our prototype, we quantify the effectiveness of our design, and explore the limitations of our prototype implementation.

The remainder of this paper is structured as follows. We begin in Section 2 with a brief survey of related work. Section 3 describes the fundamental underpinnings of our approach, while Section 4 surveys the practical challenges to implementing a real downclocked WiFi receiver. We describe how our design addresses each of these challenges in Section 5. We then introduce our implementation and the modifications needed to support Enfold on existing infrastructure in Section 6. Section 7 evaluates our approach, both in simulation and in practice with real WiFi devices. Finally, in Section 8 we evaluate the energy saving benefits of Enfold based on network traces of popular smartphone apps.

2. RELATED WORK

Our work follows on the heels of several recent efforts to bring DVFS to radios. In particular, in order to reduce power consumption, researchers have applied compressive sensing or sparse recovery to bring the sampling rate under 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 ADC sampling rate by orders of magnitude [19]. Recently, Hassanieh *et al.* [10] showed how to reduce the runtime of GPS synchronization by exploiting the sparse nature of synchronization.

Our own previous work on the SloMo system [13] leverages the inherent sparsity in Direct Sequence Spread Spectrum (DSSS) modulation used by 802.11b at 1 and 2 Mbps to allow WiFi transceivers to operate their radios at lower clock rates. The approach taken by SloMo, however, is limited to these rates and cannot be applied to OFDM. Enfold fundamentally differs both from SloMo and from all previous approaches we are aware of because there is no sparsity anywhere in the system. Instead, we exploit the fact that there exists well-defined structure within the signal spectrum when aliasing occurs.

3. HARNESSING ALIASING

Nyquist sampling theory dictates that to successfully receive a signal, the receiver must sample the channel at twice the bandwidth of that signal. When the sampling rate is below Nyquist rate, aliasing occurs, where the signal spectrum folds up and the transmitted signal is no longer recoverable. Clearly, aliasing is an undesirable effect in any communication system. Consequently, modern wireless transceivers are fundamentally gated by the Nyquist sampling theory and must operate with a minimal clock frequency of Nyquist rate. 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.

3.1 Downclocked OFDM modulation

At a high level, an OFDM communication system is defined by the inverse FFT and 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; b) the time domain signal is sampled at the receiver and FFTs are performed over the data samples to recover the encoded values. To focus on 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 802.11g. 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:

$$D_k = \frac{1}{64}\sum_{i=0}^{63} C_iH_ie^{j2\pi ki/64} + W_k, \quad k = 0, \ldots, 63 \qquad (1)$$

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

$$F_l = \sum_{k=0}^{63} D_ke^{-j2\pi kl/64}, \quad l = 0, \ldots, 63 \qquad (2)$$

Now let us substitute Eq. (1) to Eq. (2):

$$\begin{aligned} F_l &= \sum_{k=0}^{63}(\frac{1}{64}\sum_{i=0}^{63} C_iH_ie^{j2\pi ki/64} + W_k)e^{-j2\pi kl/64} \\ &= \frac{1}{64}\sum_{i=0}^{63}\sum_{k=0}^{63} C_iH_ie^{j2\pi k(i-l)/64} + N_l \\ &= C_lH_l + N_l \end{aligned}$$

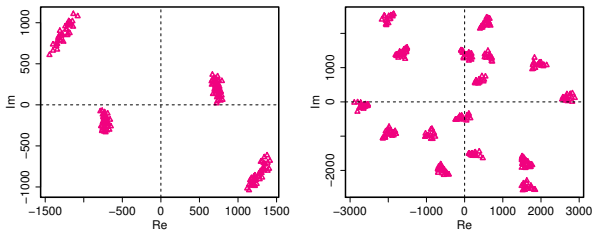
where $N_l = \sum_{k=0}^{63} W_ke^{-j2\pi kl/64}$ is the channel noise expressed in the frequency domain at subcarrier l, which results in a non-zero decoding error. Subsequently, the transmitted data is estimated as F_l/H_l.

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:

$$\begin{aligned} \tilde{F}_l &= \sum_{k=0}^{31} D_{2k}e^{-j2\pi kl/32} \\ &= \sum_{k=0}^{31}(\frac{1}{64}\sum_{i=0}^{63} C_iH_ie^{j2\pi 2ki/64} + W_{2k})e^{-j2\pi kl/32} \\ &= \frac{1}{64}\sum_{i=0}^{63}\sum_{k=0}^{31} C_iH_ie^{j2\pi k(i-l)/32} + \tilde{N}_l \\ &= 0.5 * (C_lH_l + C_{l+32}H_{l+32}) + \tilde{N}_l \end{aligned} \qquad (3)$$

Notice that $\sum_{k=0}^{31} C_iH_ie^{j2\pi k(i-l)/32}$ equals to 0 if $(i-l)/32$ is not an integer. We refer to 0.5 in Eq. (3) as the downclocking scaling coefficient. One way to interpret the aliasing effect given Eq. (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 under 25% downclocking.

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



(a) 50% downclocking. (b) 25% downclocking.

Figure 1: Frequency response (constellation diagrams) on subcarrier 6 for two real WiFi packets under downclocking.

at 25% clock rate, n-QAM transmissions will be transformed into n^4-QAM. Figure 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.

3.2 Decoding aliasing induced modulation

We now explain how to decode the original data bits. We choose 802.11g 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 step, 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. (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 possible 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 the 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 (Section 7.2.1).

Similarly, when a 25% clock rate is used, the original BPSK modulation becomes 16-QAM modulation. Aliasing 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.

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

4. WIFI RECEPTION

Previous work has demonstrated that common smartphone apps would benefit greatly from dynamic voltage and frequency scaling (DVFS) by reducing the clock frequency. The only existing downclocked receiver design [13], however, is limited to 802.11b. 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 the actual OFDM symbol.

The designers of WiFi obviously never anticipated it would be used by downclocked radios, so existing approaches to any or all of 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 frame structure and the typical decoding pipeline of a standard, full clock rate WiFi receiver, and then address the challenges imposed by downclocked operation in the following section.

4.1 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. The top portion of Figure 3 shows the detailed structure of an 802.11g WiFi frame. The preamble consists of 10 identical "short" OFDM symbols and 2 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 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 10 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 subcarriers 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 re-

sponse, 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, 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 would 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.

5. DOWNCLOCKING WIFI

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

5.1 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 10 repetitive short OFDM symbols. The IEEE 802.11 specification [11] does not mandate any particular timing synchronization algorithm. Although there are many timing synchronization algorithms for OFDM, we, like many others [17, 20, 22], use the auto(cross)-correlation approach.

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:

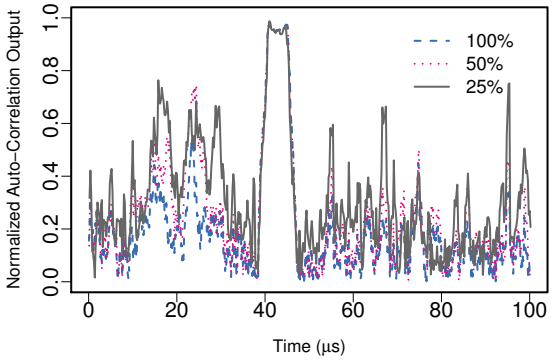
$$cros(i) = \frac{\sum_{j=0}^{15} r(i+j)S^*(j)}{\sum_{j=0}^{15} |S(j)|^2} \quad (4)$$

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 compared to the peak value. Thus, the cross-correlation peak enables accurate OFDM symbol boundary detection (i.e., fine timing synchronization).

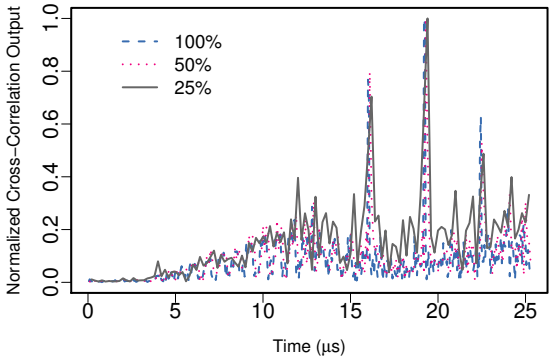
Due to the repetitive structure of the short preamble, a receiver can correlate the received data samples with themselves at one short OFDM symbol delay. Specifically, the receiver tracks of the following auto-correlation output:

$$auto(i) = \sum_{j=0}^{15} r(i+j)r^*(i+j+16). \quad (5)$$

The rational is that, when a valid WiFi frame is present, $r(i) = r(i+16)$ as 16 samples corresponds to one short OFDM symbol duration. When the auto-correlation index i is within the range of the short preamble, the output from Eq. (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) 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.



(a) Auto-correlation on short sequence.



(b) Cross-correlation on long sequence.

Figure 2: Correlation responses at variout clock rates for raw data samples of a real WiFi packet captured by USRP.

Downclocking will reduce 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. (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 2(a)).

Unfortunately for cross-correlation, the downsampled subsequences 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 2(b) 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, confirming that the approach works for all clock rates. In Section 7.1, we show in practice that timing synchronization works quite well even for a 25% clock rate at low SNR values.

5.2 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 $1 \leq i \leq n$:

$$r(n+i) = e^{j2\pi\Delta f_c T} r(i), \tag{6}$$

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. (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 10 repetitive short OFDM symbols and 2 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 7.1 we show that our frequency compensation technique also works well in practice.

5.3 Channel estimation

Once the timing and frequency synchronization is 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

$$H_i = F_i / C_i, \;\; 0 \leq i \leq 63 \tag{7}$$

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 [15, 21] 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 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. (7). Now when sampled at 50(25)% clock rate, we obtain $32(16)$ equations after performing FFT, where each equation consists of $2(4)$ unknowns (see Eq. (3)). We leverage the fact that the WiFi channel response remains 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.

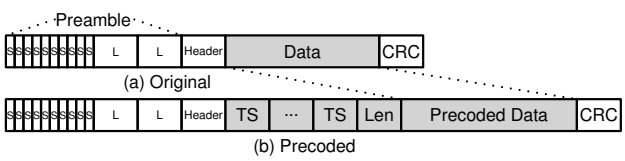


Figure 3: WiFi packet structure for both original and precoded versions. Note that our modifications are restricted to the data payload only (gray area).

5.4 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 re-define 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.

5.5 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.

5.5.1 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 raw data bits are XOR-ed first with scrambler bits, which then pass through 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, every 24 scrambled input bits generate a block of 48 bits, i.e., input bit k will produce output bits $2k$ and $2k+1$, where $0 \leq k \leq 23$. These 48 bits, after interleaving, modulation mapping and subcarrier assignments, generate a single OFDM symbol. In addition, 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.

5.5.2 Pilot restoration

For 802.11g, 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, ..., b_n and the scrambler sequence output as $x_0, x_1, x_2, \ldots$. At 6 Mbps, each group of 48 convolutional encoder output bits is mapped to a single OFDM symbol (48 out of 64 subcarriers). We run the convolutional encoder as normal while the input bits are coming through. For each incoming data bit b_i, it first gets scrambled with some scrambler output x_j. The scrambled bit is then fed into the convolutional encoder and two output bits are generated. 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 (which will be 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.

5.5.3 Training symbols

We describe how we generate the additional training symbols with reference to 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 1 and (C_j, C_{j+32}) from symbol 2 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 2, 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.

5.6 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 [13]) 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 proposed by Lu *et. al.* [13]. We experimentally verified that a WiFi device will accept 802.11b (DSSS) frames, e.g., layer-2 ACKs, when sending 802.11g (OFDM) frames. We used a commercial WiFi NIC (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 NICs often use an internal 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 802.11g, our approach is generic to other OFDM communication systems. The newer WiFi specs such as .11n and .11ac are based on OFDM as well. A direct way of applying our solution to .11n/ac is to configure these systems to operate in Single-Input-Single-Output (SISO) mode [8], which is similar to the setup of .11g 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, both leading to simplified downclocked decoding at the receiver and better performance. We leave combining 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 and 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 [8].

6. IMPLEMENTATION

We implement Enfold on the Microsoft Sora software defined radio platform [20]. The modifications are standards compatible and do not require any sender hardware changes.

6.1 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 2 or 8 channel training symbols as part of the encoded data payload.[2] The pilot restoration logic described in Section 5.5.2 consists of about 300 LoC.

Since our modifications are restricted to the data portion only, the NIC generates a standard physical layer packet header and CRC. As a result, the OFDM symbols corresponding to the header and CRC cannot be phase compensated 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 original data so that the receiver knows when to stop decoding.

Figure 3 (bottom half) 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 23, 39, 55, 11 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 a Enfold-precoded WiFi packet; the data content would simply not be recognizable, similar to encrypted data.

6.2 Receiver

Due to hardware limitations with our Sora receiver, we cannot change the clock rate of the hardware. Instead, we emulate downclocking 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 descrambled. 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 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 8 reserved bits and use the same approach for decoding the service field as the header symbol.

6.3 Network interactions

In this section, we discuss how Enfold nodes interact with 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 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 a 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 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 protocol for such a purpose. Alternatively, realizing that most deployed WiFi networks have good SNR [6], lower data rates such as 6/9 Mbps may never be picked by the rate selection algorithm of non-Enfold nodes.. Thus, and 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

[2]We tried other numbers of channel training symbols as well; 2 and 8 yield a good balance between overhead and estimation accuracy.

[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.

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

Table 1: Probability of detection P_d with different SNRs at 50% and 25% downclock rates.

using higher data rates while in downclocked mode. However, as demonstrated in Section 8, the majority of the energy saving benefits can be attained with 6 and 9 Mbps for most smartphone apps.

7. 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) [7], which allows us to control SNR over a wider range. We then evaluate the Enfold prototype system implementation on Sora [20], a fully programmable software defined radio platform.

7.1 Microbenchmark results

For our microbenchmarks we use the GNU Radio USRP platform to capture raw channel samples of 802.11g 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 SRN range from 9 dB to 35 dB (a gain above 50 leads to saturation in our experiments).

For each gain setting we capture 10 seconds of raw channel samples, containing approximately 3000 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. In an actual implementation, unlike reduced bandwidth operations where the number of digital samples remains unchanged [5], the baseband logic needs to be similarly adjusted to take a correspondingly lower number of samples.

7.1.1 *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 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 3000 packets captured in 10 seconds) when downclocked.

[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.

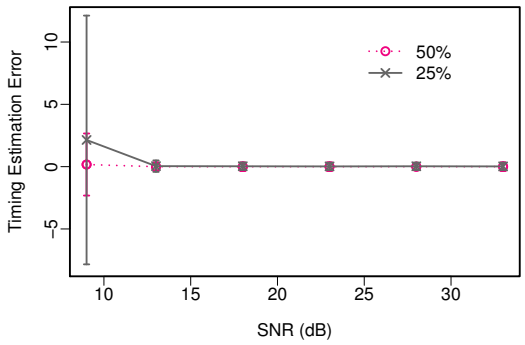


Figure 4: Timing synchronization offset error with downclocking at 50% and 25%.

7.1.2 *Timing synchronization*

Next, we confirm that we can make timing synchronization also perform well for downclocked OFDM decoding. Recall from Section 5.1 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 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 likely 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 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.

7.1.3 *Frequency offset estimation*

In Section 5.2, 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 [12] has characterized that a frequency estimation error of 10% leads to $\leq 2dB$ reduction in SNR, which is almost negligible for the SNR regime we consider.

7.2 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

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

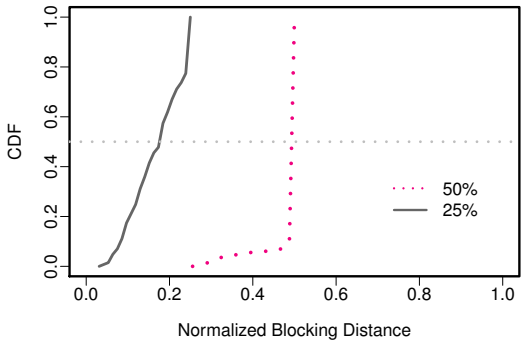


Figure 5: CDF of normalized blocking distance (relative to 100%) for 2000 WiFi packets.

explore the decoding performance of individual subcarriers, the decoding performance of entire packets across a range of SNR values, and the ability to trade off 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 [4] to calibrate the SNR of two Enfold nodes at the start of each experiment.

7.2.1 Aliasing induced modulation

As discussed in Section 3.2, 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 constellations to capture the effect of aliasing. If aliasing multiple subcarriers introduces many bit errors, the constellation points would have poor separation as represented by a too small minimum distance.

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 5 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 2000 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% on average, respectively. In our next experiment we show how this degradation impacts overall packet reception, which of course requires all subcarriers to be successfully decoded.

7.2.2 Packet reception rate vs SNR

With a sense of decoding performance per-subcarrier 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 (1000 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. 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.

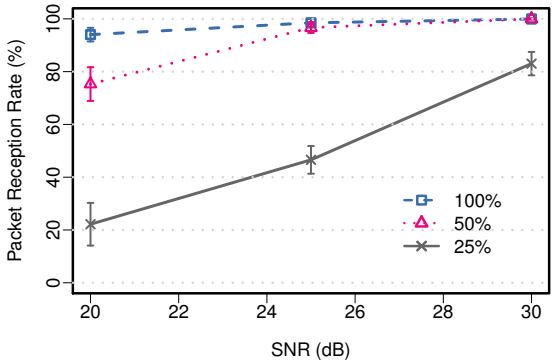
We also vary the distance between the two Enfold nodes to induce different SNRs. Unfortunately, as noted above, due to the lack of hardware AGC on the Sora platform our SNR dynamic range is rather limited (20–30 dB as reported by Sora). As discussed above, this SNR range is at the very low end for recommended 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 6 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.

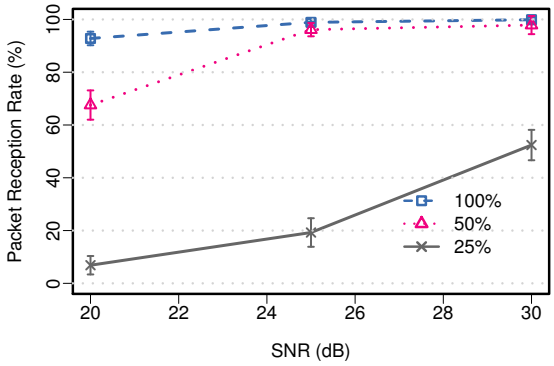
For the challenging 25% clock rate, when SNR is 30 dB (5 dB above the recommended minimum), Enfold achieved 83% and 52% PRR for small and large packet sizes, respectively. For small packets, such a PRR translates to reasonable application performance with retransmissions, but the PRR for large packets 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 [6] of deployed university WiFi networks indicates that

[6]Standard 802.11 uses only 52 subcarriers (including 4 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.

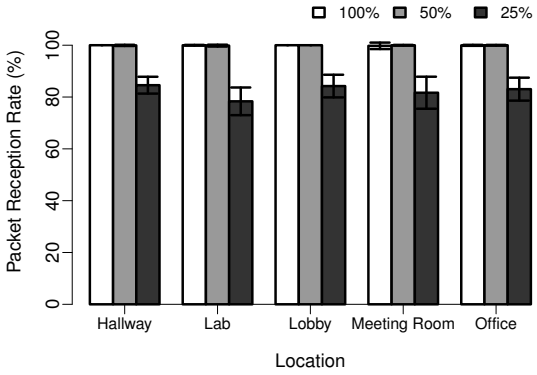


(a) Small packet (100 bytes)

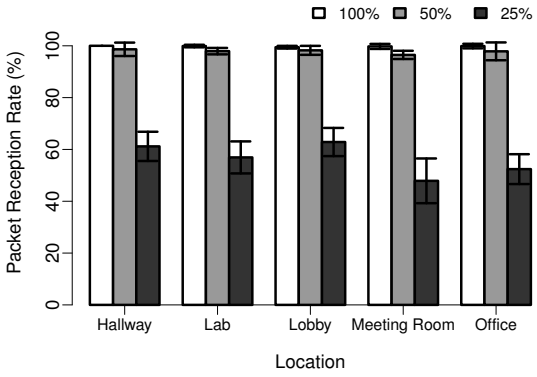


(b) Large packet (1000 bytes)

Figure 6: Packet reception rate as a function of SNR.



(a) Small packet (100 bytes)



(b) Large packet (1000 bytes)

Figure 7: Raw packet reception ratio for different clock rates at five different locations.

only 3% of WiFi frames experience retransmission (PRR is roughly 97%). According to a WiFi hotspot study conducted by Pang *et al.* [18], 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 IP addresses after association (9 out of 181 successful DHCP attempts) [9]. 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.

7.2.3 *Impact of wireless environment*

As measured in Section 7.2.1, 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 7.2.2 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.

Figure 7 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 not tied the opportune conditions of just one environment.

7.2.4 *Trading throughput for PRR*

In the last set of experiments, we explore one of the design freedoms offered by Enfold. Recall from Section 5.3 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 7.2.1, 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 7.2.2 at a 25% clock rate while adding known bits to subcarriers.

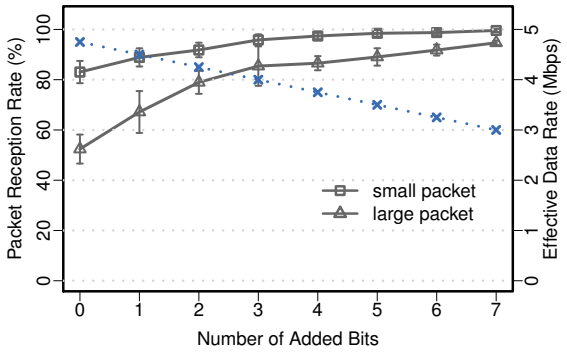


Figure 8: PRR with additional known bits on subcarriers for a 25% downclock rate (SNR = 30dB). The right y-axis shows the corresponding effective throughput.

Figure 8 shows the PRR performance and effective data rate with respect to the number of known bits sent (0 additional bit corresponds to the results for 25% at 30 dB in Section 7.2.2). As discussed above, standard 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 when downclocked to 25%.

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 3 known bits added, PRR jumps to 96% and 85% for small and large packet, respectively. After adding 3 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 7 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.

8. ENERGY CONSUMPTION

Finally, we evaluate the benefits of downclocking OFDM by estimating the energy consumption of Enfold compared to related techniques on traces of smartphone app usage. We start with the SloMo data set used by Lu *et al.* [13], which contains high fidelity WiFi packet traces from nine popular smartphone apps (each with >5 million downloads) including familiar Internet services like Facebook and Gmail as well as smartphone-specific services such as Pocket Legends (an online massively multiplayer game) and TuneIn Radio (a streaming audio service). Considering that the majority of the SloMo traces focus on low data rate apps, we add three traces of higher bandwidth apps: Wyslink (live TV streaming, 479 kbps), FaceTime (1499 kbps), and YouTube (588 kbps).

Given the lack of a real downclocked WiFi chipset with which to measure power consumption, we use a device power model to estimate the network energy consumption of each app given its network behavior in the traces. We use the WiFi power model from [13, 16], which is based on measurements of a Nexus One reported by Manweiler *et al.* [14]. When a WiFi chipset is actively transmitting or receiving, it stays in the high power state. Once network transfers complete, it transitions to the idle state (note that the power consumption in idle is still comparable to the active state). When there is no network activity for a while (the tail time), it transitions to the light sleep state. Note that the light sleep state also consumes a significant amount of power in anticipation of immediate wake-up. When the chipset is not woken up for about 500 ms, it finally moves to the deep sleep state where the power consumption is nearly negligible. The detailed parameters associated with each state and clock rate are given in [13]. Furthermore, the power model used is deliberately conservative.[7] For example, the idle power consumption for the 25% clock rate is only 65% of the idle power at full rate.

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 highest rate in 11g) with 100% 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 9 compares the normalized energy consumption of PSM, SloMo (2-Mbps DSSS rate with 100% PRR) and Enfold (4.75-Mbps with 63% PRR in Section 7.2.3). 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).

Apps like Skype voice frequently require network access but the amount of data transferred is low (data rates range from 10s to 100s 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 for 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 are 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 (~2.4x SloMo rate for the current implementation), Enfold is able to save appreciable energy for these apps: from Skype Video (9%) to Pandora (19%). 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 and an improved PRR of 100%, energy savings are correspondingly better: from Skype Video (10%) to Pandora (21%) (with YouTube at 8%, showing the effects of PRR). In sum, Enfold significantly extends

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

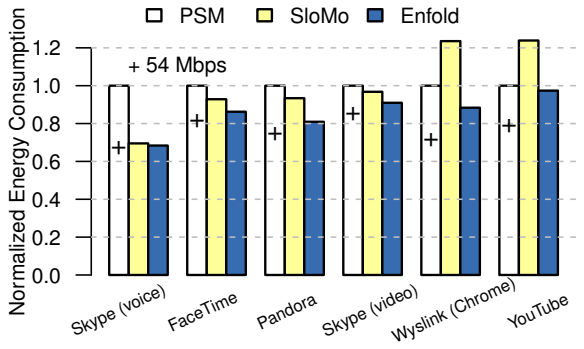


Figure 9: Energy comparison among PSM, SloMo, and Enfold. We also include a hypothetical case where the downclocked reception data rate is 54 Mbps and PRR is 100%. Results are normalized with respect to PSM.

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 evaluated energy consumption with Enfold for other data rates and PRRs (the throughput-PRR trade-off enabled in Section 7.2.4). 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.

9. CONCLUSION

In this paper we present an approach for successfully receiving and decoding OFDM modulation while sampling below the Nyquist frequency. We exploit the aliasing that results from undersampling 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× and 4×, respectively. Enfold explicitly trades throughput (SNR) for energy savings by moving to downclocked states with potentially reduced data rates in proportion to downclocking ratio. Based on published WiFi traces, we show such trade-off is worthwhile; Enfold could reduce network energy consumption by up to 34% for many popular smartphone apps. Even when nodes are in sleep mode, the energy saving benefit still persists for downclocked beacon listening and AP scanning.

Acknowledgments

We would like to thank the anonymous reviewers and our shepherd for their detailed feedback. This work was supported by generous research, operational and in-kind support from the UCSD Center for Networked Systems (CNS).

10. REFERENCES

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

[5] CHANDRA, R., MAHAJAN, R., MOSCIBRODA, T., RAGHAVENDRA, R., AND BAHL, P. A Case for Adapting Channel Width in Wireless Networks. In *Proceedings of ACM SIGCOMM* (Aug. 2008).

[6] CHEN, X., JIN, R., SUH, K., WANG, B., AND WEI, W. Network Performance of Smart Mobile Handhelds in a University Campus WiFi Network. In *Proceedings of ACM IMC* (Nov. 2012).

[7] ETTUS RESEARCH LLC. Universal Software Radio Peripheral (USRP). http://www.ettus.com/.

[8] HALPERIN, D., GREENSTEIN, B., SETH, A., AND WETHERALL, D. Demystifying 802.11n Power Consumption. In *Proceedings of USENIX HotPower* (Oct. 2010).

[9] HAN, D., AGARWALA, A., ANDERSEN, D. G., KAMINSKY, M., PAPAGIANNAKI, K., AND SESHAN, S. Mark-and-Sweep: Getting the "Inside" Scoop on Neighborhood Networks. In *Proceedings of ACM IMC* (Oct. 2008).

[10] HASSANIEH, H., ADIB, F., KATABI, D., AND INDYK, P. Faster GPS via the Sparse Fourier Transform. In *Proceedings of ACM MobiCom* (Aug. 2012).

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

[12] LEE, J., LOU, H.-L., TOUMPAKARIS, D., AND CIOFFI, J. M. Effect of Carrier Frequency Offset on OFDM Systems for Multipath Fading Channels. In *Proceedings of IEEE GLOBECOM* (Dec. 2004).

[13] LU, F., VOELKER, G. M., AND SNOEREN, A. C. SloMo: Downclocking WiFi Communication. In *Proceedings of USENIX NSDI* (Apr. 2013).

[14] MANWEILER, J., AND CHOUDHURY, R. R. Avoiding the Rush Hours: WiFi Energy Management via Traffic Isolation. In *Proceedings of ACM MobiSys* (June 2011).

[15] MENG, J., YIN, W., LI, Y., NGUYEN, N., AND HAN, Z. Compressive Sensing Based High-Resolution Channel Estimation for OFDM System. *IEEE Journal of Selected Topics in Signal Processing 6*, 1 (2012), 15–25.

[16] MITTAL, R., KANSAL, A., AND CHANDRA, R. Empowering Developers to Estimate App Energy Consumption. In *Proceedings of ACM MobiCom* (Aug. 2012).

[17] MODY, A. N., AND STUBER, G. L. Synchronization for MIMO OFDM systems. In *Proceedings of IEEE GLOBECOM* (Nov. 2001).

[18] PANG, J., GREENSTEIN, B., KAMINSKY, M., MCCOY, D., AND SESHAN, S. Wifi-Reports: Improving Wireless Network Selection with Collaboration. In *Proceedings of ACM MobiSys* (June 2009).

[19] POLO, Y. L., WANG, Y., PANDHARIPANDE, A., AND LEUS, G. Compressive Wide-Band Spectrum Sensing. In *Proceedings of IEEE ICASSP* (Apr. 2009).

[20] TAN, K., ZHANG, J., WU, H., JI, F., LIU, H., YE, Y., WANG, S., ZHANG, Y., WANG, W., AND VOELKER, G. M. Sora: High Performance Software Radio Using General Purpose Multi-core Processors. In *Proceedings of USENIX NSDI* (Apr. 2009).

[21] TAUBOCK, G., AND HLAWATSCH, F. A Compressed Sensing Technique for OFDM Channel Estimation in Mobile Environments: Exploiting Channel Sparsity for Reducing Pilots. In *Proceedings of IEEE ICASSP* (Apr. 2008).

[22] YIP, K.-W., WU, Y.-C., AND NG, T.-S. Timing-Synchronization Analysis for IEEE 802.11a Wireless LANs in Frequency-Nonselective Rician Fading Environments. *IEEE Transactions on Wireless Communications 3*, 2 (2004), 387–394.

[23] ZHANG, X., AND SHIN, K. G. E-MiLi: Energy-Minimizing Idle Listening in Wireless Networks. In *Proceedings of ACM MobiCom* (Sept. 2011).