Accurate WiFi Packet Delivery Rate Estimation and Applications

Muhammad Owais Khan and Lili Qiu
The University of Texas at Austin
{owais,lili}@cs.utexas.edu

***Abstract*—The signal-to-noise ratio (SNR) is the gold standard metric for capturing wireless link quality, but offers limited predictability. Recent work shows that frequency diversity causes limited predictability in SNR, and proposes effective SNR. Owing to its significant improvement over SNR, effective SNR has become a widely adopted metric for measuring wireless channel quality and served as the basis for many recent rate adaptation schemes. In this paper, we first conduct trace driven evaluation, and find that the accuracy of effective SNR is still inadequate due to frequency diversity and bursty errors. While common wisdom says that interleaving should remove the bursty errors, bursty errors still persist under the WiFi interleaver. Therefore, we develop two complementary methods for computing frame delivery rate to capture the bursty errors under the WiFi interleaver. We then design a new interleaver to reduce the burstiness of errors, and improve the frame delivery rate. We further design a rate adaptation scheme based on our delivery rate estimation. It can support both WiFi and our interleaver. Using extensive evaluation, we show our delivery rate estimation is accurate and significantly out-performs effective SNR; our interleaver improves the delivery rate over the WiFi interleaver; and our rate adaptation improves both throughput and energy.**

I. Introduction

Motivation: The signal-to-noise ratio (SNR) is the gold standard metric that captures the quality of a wireless link. However, it is well known that it lacks predictable power – the performance of a wireless link under the same SNR can be dramatically different. This significantly complicates a wide range of wireless network management tasks, such as rate adaptation, scheduling, routing, and diagnosis.

Recently, [10] shows that the fundamental reason that SNR fails to predict wireless performance is frequency diversity. Due to frequency selective fading and narrow-band interference, the signal-to-noise ratio across even a 20-MHz WiFi channel can vary significantly. The frequency diversity is only going to increase, as the channel width further increases in order to satisfy explosive growth of traffic demands. Based on this observation, [10] develops a new metric, called effective SNR, which offers better predictability over SNR. Effective SNR is computed by first getting SNR across each OFDM subcarrier based on Channel state information (CSI), then mapping SNR to BER for each subcarrier, and finally finding the SNR that has the same BER as the average BER across the subcarriers. Owing to its significant performance benefit over SNR, effective SNR has become a widely adopted metric for wireless channel quality and used as the basis for many recent rate adaptation schemes (*e.g.*, [10], [9], [15], [19]).

Our approach: While effective SNR improves over SNR, we find that its accuracy in predicting wireless link performance is still inadequate. This is because effective SNR only captures average BER before FEC decoding, but for wireless decoders, such as the widely used Viterbi decoder, not only does average BER matter, but also the burstiness of corruptions is important. For the same average BER, the frame delivery rate can vary significantly depending on the burstiness of the corrupted bits.

A natural question is why we care about burstiness given that interleavers are widely used in practical systems. However, we find the standard interleaver used in WiFi does not completely remove the burstiness of corrupted bits. Using the traces collected from WiFi networks via Intel WiFi Link 5300 (iwl5300) IEEE a/b/g/n wireless network adapters, we show errors remain bursty even after interleaving. This is because the existing interleaver spreads immediate neighbors to 4 subcarriers apart, which may still experience bursty errors.

Motivated by this observation, we develop two methods to estimate WiFi delivery rates under bursty errors. Our first method estimates the delivery rate using the error patterns in a frame as follows. We generate random error patterns based on BER, slide a window over a frame, and use a lookup table to determine the delivery rate of each sliding window based on its error pattern. Then we compute frame delivery rate by combining delivery rates of all relevant sliding windows. Our second method applies machine learning to learn the function that maps BER to the frame delivery rate. We use neural network due to the non-linear relationship between the two. Our two methods are complementary: the first approach (i) provides insight on what leads to frame corruption, and (ii) quickly generates lots of training data for the machine learning approach without performing frame-level simulation; the second approach is fast as it simply applies the learned function to the input extracted from the received signals.

To further address the bursty errors, next we apply our insights to design a better interleaver. Unlike existing interleavers, our new interleaver takes advantage of CSI information, and explicitly interleaves symbols to reduce bursty errors and improve decoding rates.

In addition, due to inaccurate delivery rate estimation, effective SNR may select an inappropriate rate. This motivates us to develop a rate adaptation scheme based on our delivery rate estimation. Our rate adaptation can work with different interleavers, including both the WiFi and our interleavers.

Our contributions can be summarized as follow:

978-1-4673-9953-1/16/$31.00 ©2016 IEEE

- We use measurement to show that wireless errors remain bursty even after applying the standard WiFi interleaver (Section II). This may lead to inaccurate delivery rate estimation and sub-optimal performance in rate adaptation.
- We develop two complementary methods that compute frame delivery rates in the presence of frequency diversity and bursty errors. The first method gives insight into what leads to frame corruption, and the second method is more efficient (Section III).
- We design a new CSI-aware interleaver to reduce bursty error and improve decoding rate (Section IV).
- We further develop a rate adaptation scheme based on our delivery rate estimation. It is flexible and can support different interleavers (Section V).
- Our evaluation shows that both our delivery rate estimations are accurate and significantly out-perform effective SNR. Moreover, our interleaver and rate adaptation built on top of our estimation out-perform existing interleaver and rate adaptation (Section VI).

II. Motivation

Background: In OFDM, a channel is divided into multiple orthogonal subcarriers, and the data is spread on to these subcarriers for simultaneous transmission. Due to frequency selective fading and narrow-band interference, the SNR of these subcarriers vary across different frequencies.

The IEEE 802.11n standard specifies how to measure and report channel state information (CSI). CSI is essentially a collection of $K_t \times K_r$ matrices H_s, each of which specifies amplitude and phase between pairs of K_t transmitting antennas and K_r receiving antennas on subcarrier s. SNR relates with amplitude A as follows: $SNR = 10\log_{10}(A^2/N)$, where N denotes the average power of white noise. Following the IEEE 802.11n, wireless network adapters (*e.g.*, Intel WiFi Link 5300 (iwl5300)) report the CSI of a preamble in each frame.

WiFi interleaver: According to the IEEE 802.11 standard, all data bits are interleaved by a block interleaver with a block size corresponding to the number of bits in one OFDM symbol [1]. For simplicity, we consider interleaving in a single antenna case for illustration, and a similar performance issue exists in MIMO cases. Let N_{CBPS} denote the block size, and N_{BPSC} denote the number of coded bits per subcarrier. The block interleaver is a two-step permutation procedure. The first permutation step maps adjacent coded bits to non-adjacent subcarriers. The second permutation maps adjacent coded bits alternatively to less and more significant bits of the constellation to avoid long runs of low reliability bits. Let k denote the index of the coded bit before the first permutation, i denote the index after the first permutation, and j denote the index after the second permutation. The two permutation steps are defined as follows, where $k = 0, 1, ..., N_{CBPS} - 1$, $i = 0, 1, ..., N_{CBPS} - 1$, and s is determined by N_{BPSC} according to $s = max(N_{BPSC}/2, 1)$.

$$i = (N_{CBPS}/13)(k \bmod 13) + floor(k/13)$$
$$j = s \times floor(i/s) + (i + N_{CBPS} - floor(13 \times i/N_{CBPS})) \bmod s$$

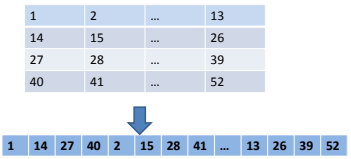


Fig. 1. The WiFi interleaver for BPSK.

Figure 1 shows the standard interleaver for BPSK. IEEE 802.11n has 52 data subcarriers, so there are 52 BPSK symbols in each OFDM symbol. The interleaver stripes 52 bits by placing them in a 4×13 grid row-wise and reading them column-wise so that adjacent bits are spread 4 subcarriers apart. It is quite likely SNR at these subcarriers still experience similar performance, thereby resulting in bursty errors. This effect persists under other modulation schemes: the adjacent bits are still 4 subcarriers away and bursty errors still exist.

Trace collection: We analyze real WiFi traces to understand their burstiness after interleaving. The traces were collected from WiFi networks using Intel WiFi Link 5300 (iwl5300) wireless network adapters. Three channel traces are from static environments, and another three traces are from mobile environments with human walking speed. The three mobile traces are collected in an office environment using 1 moving receiver and 3 static senders. The three static senders are 7m away from each other. Each trace corresponds to one of the three senders transmitting while the receiver is moved at a walking speed. The NICs have three antennas, which are all enabled in our measurement. The modified driver [11] reports the channel matrices for 30 subcarrier groups, which is about one group for every two subcarriers in a 20 MHz channel according to the standard [1] (*i.e.*, 4 groups have one subcarrier each, and the other 26 groups have two subcarriers each). We use 1000-byte frames, MCS-16, and a transmission power of 15 dBm. MCS-16 has 3 streams, so the NICs report CSI in the form of 3×3 matrices for each frame. So altogether we have traces of 27 static links and 27 mobile links.

Bursty errors: The IEEE 802.11n supports four types of FEC: 1/2 (*i.e.*, half redundancy), 2/3 (*i.e.*, one third redundancy), 3/4, and 5/6. So we focus on these FEC throughout this paper. Figure 2 plots CDF of the gap between two consecutive errors for 1/2 FEC. The results are similar for the other FEC. The gaps in the WiFi interleaver have skewed distribution, with a much larger fraction concentrated on the lower end than the ideal interleaver, which equally spaces the error across a frame. The gap in the ideal interleaver is not a constant due to a varying number of errors in each frame. These results show that the errors are still bursty under the WiFi interleaver.

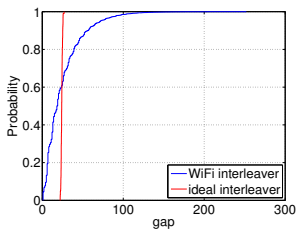


Fig. 2. CDF of gaps between two consecutive errors in bits when using the WiFi interleaver and an ideal interleaver.

III. Estimating Delivery Rate

Motivated by the bursty errors despite the use of the standard WiFi interleaver, in this section we develop two methods to estimate delivery rates and explicitly capture the bursty errors. We focus on convolutional coding, the default used in IEEE 802.11 a/b/g/n/ac. The convolutional code used in WiFi takes data bits as input and generates coded bits that do not include original data bits (*i.e.*, non-systematic coding). Either all bits in a frame are decoded or nothing is decoded.

A. Method 1: Lookup Table Based Estimation

Our first scheme randomly samples the error patterns based on CSI, data rate, and interleaver. To achieve high efficiency, we build a lookup table that maps an error pattern in a sliding window to a delivery rate and then online computes the frame delivery rate. Below we describe the detailed scheme.

1) Random sampling: The success rate of Viterbi decoding is hard to model analytically (*e.g.*, [22], [25]). Instead, we can empirically compute the delivery rate for a given CSI over many frames, and use the average delivery rate as the estimation. Specifically, we can generate frames with random payload, use the current data rate (*i.e.*, modulation scheme and FEC) and interleaver to map bits onto subcarriers, corrupt the frames according to the CSI, and feed each corrupted frame to the Viterbi decoder to derive the average frame delivery rate after FEC decoding.

2) Overview: In order to achieve high accuracy of delivery rate estimation, hundreds of frames are required. It is too expensive to run Viterbi decoding on hundreds of frames online. A natural approach is to run Viterbi decoding offline and build a lookup table in advance. Note that this lookup table takes error patterns as the input, so it is independent of wireless channel or hardware, and we can use the same table across all devices under all channel conditions.

Specifically, the table includes whether decoding is successful for each error pattern (*e.g.*, 0 or 1 sequence where 0 indicates no error in the bit and 1 indicates an error). For example, one entry in the table for a given FEC may have [110000011111000000, fail], which indicates that the frame decoding fails if the bits in the frame (after de-interleaving at the receiver) has the corruption pattern 110000011111000000. Note that we only consider the bit corruption pattern, but not the content of the frame in order to determine whether it can be successfully decoded, because the frame content is not available a priori and the impact of frame content on delivery rate is much less significant than the error patterns.

A major challenge is what should be the index for the lookup table in order to achieve high accuracy and efficiency. Using an error pattern for an entire frame is prohibitively expensive since there are $2^{FrameSize}$ error patterns. We develop a sliding-window-based lookup table approach to enhance efficiency. Specifically, the lookup table is built in advance, and a receiver performs the following steps online:

1. uses the current data rate and interleaver to determine which subcarrier each bit is assigned to;
2. estimates the SNR of each subcarrier using a preamble;
3. determines BER or joint error probability based on the modulation and SNR of the assigned subcarrier;
4. generates random samples of error patterns based on BER;
5. looks up tables to determine the decoding success rate for each error pattern in a sliding window;
6. estimates frame delivery rate based on delivery rates of relevant sliding windows for each sampled frame;
7. estimates the delivery rate as the average frame delivery rate over all randomly sampled frames.

Steps 1), 2), 4) and 7) are either standard or straight-forward, so below we focus on steps 3), 5), and 6).

3) Mapping SNR to BER: Mapping SNR to BER is an important step in many wireless schemes, including effective SNR. We improve the accuracy of existing mapping by capturing that (i) different bit positions in QAM experience different BER under the same SNR and (ii) there is correlation between BER of different bits in the same symbol. This contribution is useful to many wireless schemes, including other rate adaptation schemes.

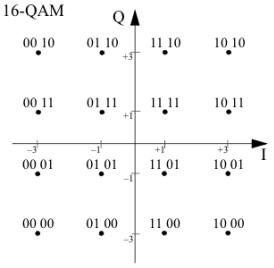
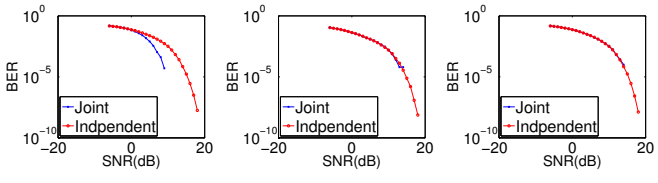
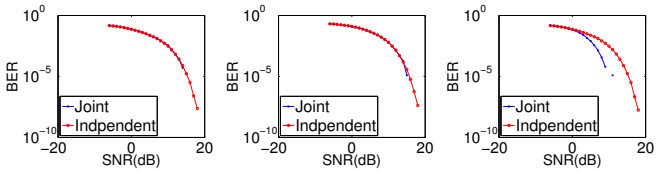


Fig. 3. Constellation points for QAM-16 with bits $\mathbf{b_1 b_2 b_3 b_4}$

We use the standard formulas to compute BER in BPSK and QPSK: $Q(\sqrt{2snr})$ for BPSK and $Q(\sqrt{snr})$ for QPSK. We find that the standard formulas do not work for QAM for two reasons. First, different bits in QAM may experience different BER. As shown in Figure 3, the symbols that differ in bits 1 or 3 have larger minimum distance than those that differ in bits 2 or 4. So BERs of bits 1 and 3 are lower than those of bits 2 and 4. We empirically find that the BERs of bits 1 and 3 match well with $\frac{1}{2}Q(\sqrt{snr/5})$, and BERs of bits 2 and 4 match $Q(\sqrt{snr/5})$. The corresponding functions for QAM-64 are $\frac{1}{4}Q(\sqrt{snr/21})$ for bits 1 and 4, $\frac{1}{2}Q(\sqrt{snr/21})$ for bits 2 and 5, and $Q(\sqrt{snr/21})$ for bits 3 and 6.



(a) Bits 1 and 2 joint error (b) Bits 1 and 3 joint error (c) Bits 1 and 4 joint error



(d) Bits 2 and 3 joint error (e) Bits 2 and 4 joint error (f) Bits 3 and 4 joint error

Fig. 4. Joint error probabilities of 2-bits.

Moreover, we find the bits in QAM have considerable correlation. Figure 4 compares the joint error probability of two bits from real traces versus estimation based on independence assumption. The estimated joint error deviates from the independence assumption for bits 1, 2 and bits 3, 4, whereas the match is good for the other 2-bit combinations. The gap for bits 1,2 and bits 3,4 indicate negative correlation between these bits. A closer look at Figure 3 reveals that if bit 1 is correct, bit 2 is incorrect when the received symbol is over $r/2$ away from the transmitted symbol, where r is the horizontal or vertical separation between two adjacent symbols; if bit 1 is incorrect, bit 2 is incorrect when the received symbol is over r away from the transmitted symbol. The same reasoning applies for bits 3 and 4. Moreover, using similar analysis, we find that the bits 1 and 2 are independent of bits 3 and 4. Based on these observations, we map SNR to joint error probability of bits 1, 2 and bits 3, 4. This is achieved as follow: for each SNR, we generate random symbols, go through modulation and demodulation to determine error patterns, and then compute average probability of error patterns 00, 01, 10, and 11. This yields a lookup table that maps SNR to the probability of having error patterns 00, 01, 10, and 11. We conduct similar analysis for QAM-64, and find bits 1, 2, and 3 are correlated, so are bits 4, 5, and 6, whereas bits 1, 2, 3 are independent of bits 4, 5, and 6. Therefore we generate a similar lookup table that maps SNR to the probability of any 3-bit error pattern.

4) Building a lookup table for a sliding window: Due to finite state space of the convolutional code, the impact of errors does not propagate beyond a certain point. Therefore, we can use a sliding window to capture impact of errors. The sliding window size depends on the type of FEC. We empirically derive the window size for different FEC codes by gradually increasing the window size until a further increase offers little improvement in the delivery rate estimation. We find that the window size is 75 bits for 1/2 FEC, 50 bits for 2/3 and 3/4 FEC, and 40 bits for 5/6 FEC. Therefore, instead of building a lookup table for entire frames, we build a lookup table for a sliding window with different error patterns.

Pruning the table: Building a table for a sliding window is still too expensive since there are $2^{winSize}$ error patterns, where $winSize$ is the sliding window size. Interestingly, after analyzing the delivery rate of different error patterns, we observe that whenever the number of errors is lower than $lowThresh$ in the window, the Viterbi decoding is always successful; whenever there are more than $highThresh$ errors, the decoding fails almost all the time; only when the number of errors is in between, the decoding error varies according to the error pattern. This is consistent with our expectation of FEC decoding. We empirically find $lowThresh$ are 5, 3, 2, and 2 for 1/2, 2/3, 3/4 FEC, and 5/6 FEC, respectively, and the corresponding $highThresh$ are 11, 5, 4 and 4, respectively. When the number of errors is below $lowThresh$, the decoding failure rate of a sliding window is within 1.65%. When the number of errors is above $highThresh$, the decoding failure rate of a sliding window is above 67% for 1/2, 96% for 2/3, and even higher for 3/4 and 5/6. We choose $highThresh$ for 1/2 to have smaller corresponding error rates to reduce storage. In practice, due to multiple sliding windows within a frame, 46% decoding error of a sliding window will likely result in decoding error of a frame. Based on these thresholds, we build lookup tables for the middle cases (*i.e.*, 5-11 errors for 1/2 FEC, 3-5 errors for 2/3 FEC, and 2-4 errors for both 3/4 FEC and 5/6 FEC).

Taking into account position of the sliding window: For the middle cases where the number of errors is not too high or too low, the decoding rate depends on not only the error pattern but also the position of the window. For example, when using 1/2 FEC, the decoding rate of an error pattern varies according to whether the first error bit in the sliding window resides at an odd or even offset in the original frame (not the offset in the sliding window). This is illustrated in Figure 5, which shows the delivery rates of three error patterns as we move the error patterns to start at varying locations from 30 to 40. As shown in Figure 5(a), the decoding rate of an error pattern with 5 erroneous bits represented by the black line in the figure is 0 if it starts from an odd offset and 0.85 if it starts from an even offset. When using 2/3 FEC, the decoding rate of an error pattern varies depending on the offset modular 3, as shown in Figure 5(b). For instance, the decoding rate of an error pattern represented by the red curve, which has 4 bits in error, is 0.75, 0, and 0.55, respectively, as the offset modular 3 varies from 0 to 2. Similarly, the decoding rate varies with the offset modular 4 for 3/4 FEC, and varies with the offset modular 6 for 5/6 FEC. The cycles in the delivery rate patterns correspond exactly to the number of bits in the puncturing pattern for each convolutional code (*e.g.*, 1/2 FEC has a puncturing pattern of [1 1], 2/3 FEC has a pattern of [1 1 1 0], 3/4 FEC has a pattern of [1 1 1 0 0 1], and 5/6 FEC has a pattern of [1 1 1 0 0 1 1 0 0 1]) [1]. Therefore, we maintain 2 lookup tables for 1/2 FEC, 3 lookup tables for 2/3 FEC, 4 lookup tables for 3/4 FEC, and 6 lookup tables for 5/6 FEC. For each sliding window, we find the first error in the window, and identify its offset in the original frame to select the appropriate table to look up.

Supporting a large sliding window: Using the above techniques, we can reduce the lookup tables for 2/3, 3/4, and 5/6 FEC to reasonable sizes. The sliding window for 1/2 FEC has 75 bits, and the number of errors in the middle cases ranges from 5 to 11. The corresponding lookup table sizes are still too large. To further reduce the table build-up time and storage cost, instead of using a sliding window of 75, we split the window into two parts: 40 bits and 35 bits. We enumerate error patterns in the first 40 bits, but only consider the number of errors in the next 35 bits (as opposed to the exact error patterns) to reduce the cost. This yields a 2-dimension lookup table, where the first dimension is the error pattern in the first 40 bits and the second dimension is the number of errors in the next 35 bits. In our implementation, the first window has error patterns involving 4-7 errors, whereas the second window has up to 6 errors. To further reduce the overhead, we only

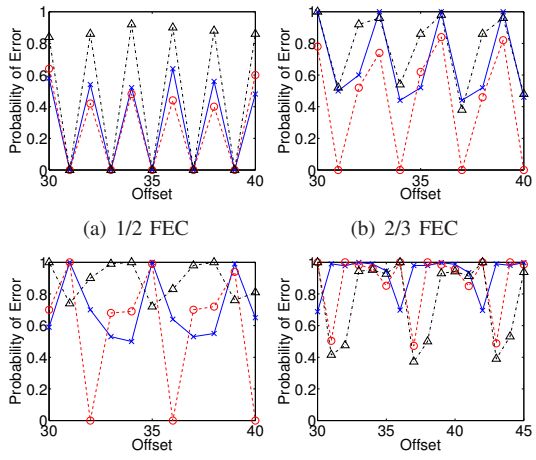


Fig. 5. The decoding success depends on not only the error pattern in a sliding window, but also where the error pattern starts in a frame. Different curves correspond to different error patterns in a sliding window. They are shifted to different offsets in a frame.

consider the second window when the first window has 4 or 5 errors. When the first window has 6 or 7 errors, the delivery rate is already low and the second window has little impact. So the total lookup tables for all FEC schemes is around 58 MB before compression and 8 MB after compression, which is affordable given 1 GB RAM in iPhone 5 and 6.

5) Estimating frame delivery rate: How to compute the delivery rate for a frame, which spans multiple sliding windows? One approach is to approximate it using a product of delivery rates over all sliding windows because all the sliding windows should succeed in order for a frame to succeed. However, since the delivery rates over adjacent sliding windows are not independent, this approximation is inaccurate.

To improve the accuracy, we avoid counting the same error pattern multiple times as the window slides. More specifically, we move the starting position of the sliding window to the first error and look up the delivery rate based on the error sequence within this window. Next, we move the sliding window to start from the next error. If any new error appears in the window, we look up the probability of error for the new pattern. If no new errors appear, we ignore the current sliding window and move the next sliding window to start from the next error and repeat the process until we have gone over all the errors. The product of delivery rates across all such windows gives us the final frame delivery rate estimate.

B. Method 2: Machine Learning Based Estimation

Our second method uses machine learning to estimate delivery rate based on the CSI. The advantage of this method is that the online computation is fast once we learn the function offline. Our main tasks involve selecting appropriate features and a machine learning algorithm. A natural choice of features are SNRs across all subcarriers. However, this requires us to learn a new delivery rate function for each modulation and FEC. Moreover, mapping from SNR to BER is quite involved, as shown in Section III-A3, and not easy to learn. Since BER has more direct relationship with the frame delivery rate, we leverage our domain-knowledge to map SNR to BER, as described in Section III-A3, and use BER for each bit as features to map to the frame delivery rate. Due to the use of OFDM and the nature of frequency diversity, BER per bit repeats every OFDM symbol. So we use BER for each bit in an OFDM symbol as the feature set. We choose neural network as the machine learning algorithm. The advantage of using neural networks is that they can approximate any function with arbitrary accuracy [6], [14], which is appropriate as the frame delivery rate functions are non-linear.

We use a feed-forward artificial neural network model called multi-layer perceptron (MLP) [4] that maps BER per bit to frame delivery rate. An MLP consists of multiple layers of nodes, where each layer is fully connected to the next layer. The first layer of nodes is called input layer, and the last layer is called the output layer. The intermediate layers are called hidden layers. Each node corresponds to a neuron, with a non-linear transfer (activation) function. Following the common practice, we use 'mapminmax' to normalize the inputs, use 'sigmoid' transfer function for hidden layers [12], and use 'purelin' as the output transfer function. The neural network is trained using Levenberg-Marquardt algorithm [8].

We vary the number of hidden layers from 1 to 10, and the number of neurons in each hidden layer from 5 to 25. The accuracy of using 1 hidden layer is noticeably worse than multiple hidden layers, and the accuracy of 5 neurons per hidden layer is noticeably worse than 10 or more neurons. As long as multiple layers and 10 or more neurons are used, the performance is comparable. We use 5 hidden layers and 20 neurons in each hidden layer since it slightly out-performs the other configurations involving multiple layers and 10 or more neurons.

The remaining issue is how to generate the data for training. We get traces from real WiFi networks and synthetic traces generated using IEEE 802.11n TGn channel model [7]. IEEE 802.11n channel models have six profiles, including flat channel, indoor residential, residential/office, typical office, large indoor and outdoor open spaces. We use all profiles to generate channels to capture a wide range of wireless link conditions. These traces generate SNRs across subcarriers, which we then use for frame-level simulation to determine the frame delivery rate. To further speed up training data generation, we also use lookup table based approach in Section III-A to generate part of the training data, and find it is much faster than frame-level simulation with only a slight increase in the error. We then partition the 15000 CSI values (half from the real traces and half from the channel model) into two thirds for training and one third for testing.

C. MIMO Extension

To support MIMO diversity, we use the CSI values to derive the post-processed SNR (pp-SNR) values for each subcarrier under each supported transmission configuration to capture the effect of MIMO processing. In MIMO, since a transmitted symbol is received on multiple antennas, the final SNR experienced by the symbol is the combination of the multiple receptions and the combined SNR dictates whether it will be

decoded correctly. We apply standard formulas to compute pp-SNR in MIMO based on the original CSI [16]. Then we derive BER according to pp-SNR as in Section III-A3. The remaining processing, including mapping pp-SNR to BER and mapping BER to frame delivery rate, is the same as SISO.

In MIMO multiplexing, a sender stripes a frame across multiple antennas and transmits the multiple streams simultaneously. Different streams experience different channel loss depending on their transmitter and receiver pairs. Therefore, to support MIMO multiplexing, we determine to which antenna and subcarrier each symbol is assigned for a given interleaver and calculate pp-SNR according to MIMO multiplexing. The remaining processing is again the same as SISO.

IV. Our New Interleaver

Next we propose a CSI-aware interleaver. The receiver feeds back the CSI according to the IEEE 802.11n standard. Instead of blindly interleaving the bits regardless of the channel condition, the sender uses our new interleaver to spread erroneous bits as far apart as possible to reduce bursty errors and improve decoding rate. The interleaving pattern does not need to be transmitted, since the sender can indicate the latest sequence number of the data frame that it receives the CSI feedback and the receiver stores the CSI of the last few frames and uses the corresponding CSI for interleaving. Since the interleaving computation is very fast (*e.g.*, 27μs in C++ implementation), the interleaving can be re-computed upon every CSI update.

Our interleaver sorts the bits in a decreasing order of BER. The bits are spread onto the OFDM subcarriers to maximize the distance between the high error bits as follows:

- Following the standard interleaver, we create a table of size $r \times c$, where r is the number of bits per symbol based on the modulation and c is the number of subcarriers.
- But different from the standard interleaver, we sort the bits and re-arrange the sorted bits in column-wise from top to bottom in the table. For example, the highest BER is at (1,1), the second highest BER is at (2,1), and so on.
- Next we re-arrange the columns to maximize the distance between the columns with high BER. We keep the first column at its initial location, move the second column to the center column so that the distance between the two highest BER columns are separated by $c/2$. Note that moving the second highest error column to the last column is not good since it will be close to the highest error bit in the next OFDM symbol. Then we use the first, center, and last columns as anchors, and calculate the middle columns between them and place the next two worst columns at these column indices (*e.g.*, the third worst column is moved to $round(c/4)$-th column, and the fourth worst column is moved to $round(3c/4)$-th column). This process continues recursively until all locations have been filled. In this way, the columns with larger BERs have larger separation between them. One caveat in this step is that since we observe the delivery ratio depends on the exact offset as shown in Figure 5, we shift the columns around to avoid occupying bad offsets. For example, in 1/2 FEC, an error starting at an even offset is worse. So whenever the calculated offset is even, we shift it up or down by 1 column as long as that column is not occupied. If the nearby columns are both occupied, we just place it at the originally computed column. Since we place columns in a decreasing order of BER, we minimize the chance of placing the worst few bits at bad offsets.
- Finally, we read the table row-wise and map the bits to the subcarriers sequentially, where each subcarrier is assigned the number of bits the modulation can support. This step is similar to the standard interleaver to reduce bursty errors.

V. Rate Adaptation

In this section, we develop rate adaptation that takes advantage of our enhanced delivery estimation and interleaver. The receiver first extracts Channel State Information (CSI) from the preamble of the previous frame, computes delivery ratio for each rate as described in Section III, and selects the rate that maximizes the total throughput (*i.e.*, maximizing the product of maximum capacity and the delivery ratio) and feeds it back to the sender to use for the next transmission. To reduce computation cost, we may search the data rates close to the current rate instead of all possible rates.

Our rate adaptation works with both the standard WiFi interleaver and our enhanced interleaver. When our interleaver is used, we use CSI to derive an appropriate interleaving as described in Section IV, based on which we derive BER after de-interleaving, estimate frame delivery rate, and select the rate that maximizes throughput.

Data rates not only affect the throughput, but also impact energy consumption by changing the transmission or reception time. When energy is used as an objective, we can compute energy based on our delivery model and an energy model. Specifically, we first use the approaches in Section III to compute the delivery rate and estimate expected transmission time (ETT), which denotes how long it takes to successfully deliver a frame on average. Then we plug ETT into the energy model in [16] to obtain the transmission or receiving energy under a given data rate. [16] reports that transmission energy and receiving energy are both linear functions of ETT; the coefficients depend on the type of wireless cards and can be determined in advance using measurements. Then we select the data rate that results in the minimum energy.

VI. Performance Evaluation

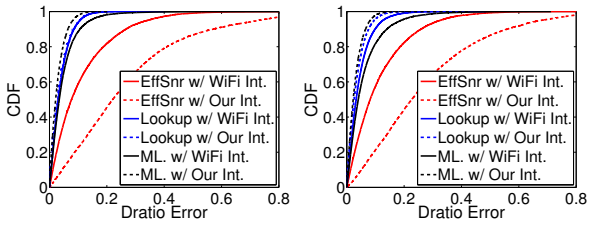
A. Evaluation Methodology

In this section, we use trace driven simulation for our evaluation. We use both real traces collected from Intel WiFi Link 5300 wireless card (Section II), and synthetic traces generated using IEEE 802.11n TGn Channel models (Section III). We first compare the accuracy of our delivery rate estimation with effective SNR. Then we compare our interleaver with the WiFi interleaver. Finally, we compare the throughput and energy of our rate adaptation with effective SNR when applied to either the WiFi interleaver or our interleaver.

B. Performance Results

Accuracy of delivery rate estimation: We first evaluate the accuracy of our delivery ratio estimation by comparing with the ground truth. The delivery ratio is estimated for a range of CSI from the collected traces. As in the previous works (*e.g.*, [2]), we uniformly scale CSI across all subcarriers to maintain the same frequency diversity. We choose different factors for each MCS (Modulation and FEC) in order to cover the complete range of delivery ratio values from 0 to 100%. The ground truth delivery ratio is calculated by transmitting and decoding 100 frames for each scaled CSI. The ground truth is calculated for both the WiFi interleaver and our interleaver.

Figure 6(a) plots the CDF of delivery ratio estimation error using the CSI values for 20 MHz channels from real and synthetic traces not used for training the neural network. Figure 6(b) summarizes the results for 40 MHz channels. Both figures compare the delivery ratio estimation errors of (i) effective SNR using the WiFi and our interleavers, (ii) lookup table based estimation using both interleavers, and (iii) machine-learning (ML) based estimation using both interleavers.



(a) 20 MHz (b) 40 MHz

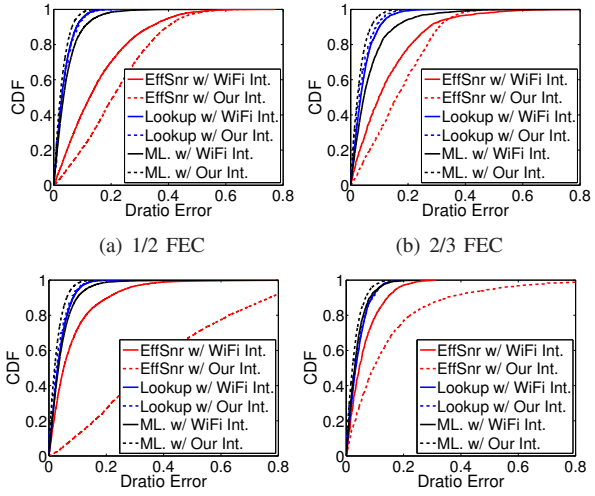
Fig. 6. Comparison of delivery rate estimation across all FEC.

We make several observations. First, our two delivery ratio estimation schemes are both accurate for the WiFi and our interleavers. In 20 MHz channels, the average error of both our schemes is around $3-5\%$. The lookup scheme exceeds 10% error for 5% and 6% under WiFi and our interleavers, respectively. The corresponding numbers for the ML scheme are 11% and 3%, respectively.

For 40 MHz channels, the lookup scheme has average errors of 4.5% and 3% for WiFi and our interleavers, and the ML scheme has average errors of 6% and 3%, respectively. The lookup scheme exceeds 10% error for 9% and 4% under WiFi and our interleaver, respectively. The corresponding numbers for the ML scheme are 18% and 2.5%, respectively.

The ML scheme shows good accuracy and has only a slight performance degradation compared to the lookup scheme. Due to its fast speed and similar estimation error, the ML is the preferred scheme for practical use.

In comparison, the estimation error of effective SNR is much higher: its average errors are 11% and 27% for the WiFi and our interleavers, respectively. For 20 MHz channels, its estimation error exceeds 10% for 41% under the WiFi interleaver and 77% under our interleaver. For 40 MHz channels, its error exceeds 10% for 42% under the WiFi interleaver, and 78% under our interleaver. There is much higher variability in effective SNR. The error varies significantly across different channel widths, across different interleavers, and even across different runs under the same settings. The effective SNR has larger estimation error with our interleaver because it is tailored to the WiFi interleaver and cannot take advantage of an enhanced interleaver, whereas our estimation schemes can incorporate any interleaver as an input to achieve high accuracy.



(a) 1/2 FEC (b) 2/3 FEC

(c) 3/4 FEC (d) 5/6 FEC

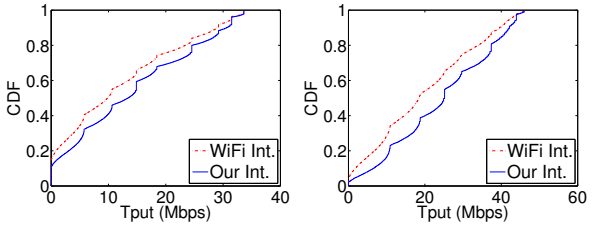
Fig. 7. Comparison of delivery rate estimation under different FEC.

Figure 7 compares the accuracy for each FEC separately in 20MHz. For each FEC, we consider only the modulations supported by IEEE 802.11n for that FEC. For 1/2 FEC, the lookup scheme has average errors of 4% and 3.5% for WiFi and our interleavers, respectively. The average errors of the ML scheme are 4.5% and 2% for the two interleavers, respectively. For 2/3 FEC, the lookup scheme has an average error of 4.5% and 4% for WiFi and our interleaver, respectively. The corresponding numbers for ML are 7% and 3%, respectively. For 3/4 FEC, the lookup scheme has average errors of 4% and 3.5% for WiFi and our interleavers, respectively. The average error of the ML are 4.5% and 2.5% for the two interleavers, respectively. For 5/6 FEC, the lookup scheme has an average error of 4% for both interleavers. The average errors of the ML are 4% and 3% for the two interleavers, respectively.

In contrast, effective SNR has significant variation across FEC and interleavers. For example, in 1/2 FEC, effective SNR yields an average error of 15% for WiFi interleaver and 58% of the time has error exceed 10% . For our interleaver, the average error increases to 21%, and 78% of time has error exceed 10%. For 3/4 FEC, the average errors of effective SNR become 8% and 39% for the WiFi and our interleaver, respectively; and the percentage of cases over 10% error are 27% and 87% for the WiFi and our interleavers, respectively. Its average error for 5/6 FEC is 6% and 14% for the WiFi and our interleaver, respectively, and 21% and 48% of cases have errors exceed 10% for the WiFi and our interleavers. Effective SNR sees lowest error in 5/6 FEC because it can only tolerate very few errors and error patterns are less important in this case.

In general, large variable errors in effective SNR introduce significant uncertainty for wireless network optimization. In

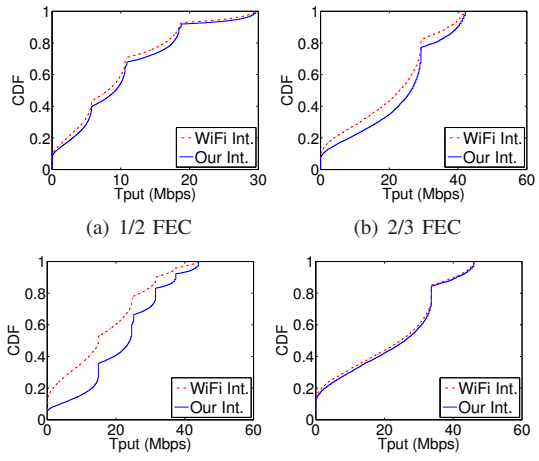
comparison, our estimators are more stable across all scenarios. It can significantly ease network optimization and management, and improve network predictability.



(a) 20 MHz (b) 40 MHz

Fig. 8. Comparison of interleavers under all rates.

Comparison of interleavers: Figure 8 compares the performance of our interleaver with the WiFi interleaver. The throughput is obtained for each interleaver by transmitting and decoding 100 frames for each CSI. The throughput is calculated for a wide range of CSI values as before. Overall, our interleaver increases the throughput over the WiFi interleaver by 18% in 20 MHz channels, and by 23% in 40 MHz channels. We expect the improvement will increase further as wider channels are used (*e.g.*, in IEEE 802.11 ac).



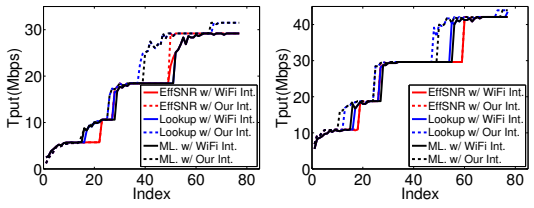
(a) 1/2 FEC (b) 2/3 FEC

(c) 3/4 FEC (d) 5/6 FEC

Fig. 9. Comparison of interleavers under different FEC.

Figure 9 shows the interleaver performance for each FEC separately. Our interleaver out-performs the WiFi interleaver by 8.5%, 13%, 37% and 3.5% for 1/2, 2/3, 3/4 and 5/6 FEC, respectively. We see the highest improvement in 3/4 because our interleaver does a good job in spreading errors apart. Since 1/2 FEC is able to tolerate more errors, it is more robust to bursty errors. At the other end, 5/6 FEC incurs decoding failures even when there are only two errors in a sliding window, which gives less opportunities for our interleaving to optimize. In comparison, the error patterns matter more for 3/4 FEC, so it benefits most from spreading the most erroneous bits apart by our interleaver.

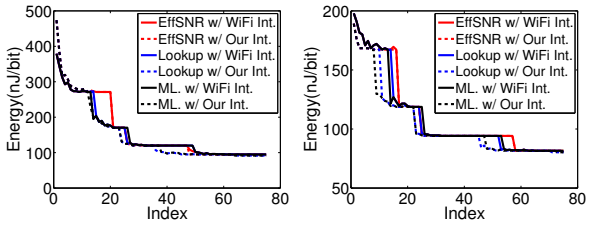
Comparison of rate adaptation schemes: Finally, we evaluate the benefit of our rate adaptation that leverages our delivery rate estimation and interleaver. For ease of illustration, we select two links: a 20 MHz link and a 40 MHz link. The other links exhibit similar performance, and their results are omitted



(a) 20 MHz (b) 40 MHz

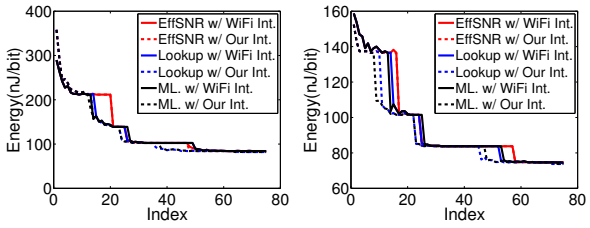
Fig. 10. Throughput comparison of rate adaptation schemes.

in the interest of brevity. Figure 10 shows the rate curves of each scheme for these links. We obtain the rate curves as follow. We impose different scaling factors for each trace and obtain throughput from different schemes. We then plot the sorted throughput for each scheme in the figure. In all cases, the rate for the next transmission is selected using the CSI of the previously received frame. Our scheme shows significant throughput improvement around the rate transition regions (*i.e.*, the boundary between two rates): the improvement ranges between 5–75%. During the non transition regions (*i.e.*, the regions that are far from the boundaries), the improvement is much smaller since the available rates are coarse-grained: two adjacent rates differ by at least 3 dB and it is hard to get close to 3 dB gain just by improving delivery rate estimation and interleaving. The benefit of our scheme will increase with more available rates. In addition, comparing the results from 20 MHz with those from 40 MHz, we observe larger benefit of our rate adaptation in 40 MHz channels due to stronger frequency diversity in a wider channel and larger benefit of our interleaver by spreading high error bits farther apart.



(a) 20MHz (b) 40MHz

Fig. 11. Transmission energy comparison of rate adaptation schemes.



(a) 20MHz (b) 40MHz

Fig. 12. Receiving energy comparison of rate adaptation schemes.

Finally, we compare the energy consumption of rate adaptation. The data rate affects energy by changing the time it takes to transmit or receive a frame. The longer it takes, the more energy a sender/receiver will consume. We assume the access point (AP) will run the rate adaptation scheme, regardless if the AP is transmitting or receiving. If the AP is transmitting, the AP selects the rate that saves the client's receiving energy. If the AP is receiving, the AP selects the rate that saves the client's transmission energy, and feeds the selected rate back

to the client to use for the next transmission. We use the energy model in [16] to derive the client's energy consumption based on expected transmission time (ETT) under different rate adaptation schemes.

Figure 11 and 12 compare transmission and receiving energy of different schemes, respectively. As we would expect, similar to the throughput results, the energy improvement takes place around the rate transition regions, where our rate adaptation reduces transmission energy by 10%–35% and reduces receiving energy by 9%–32% over the effective SNR with the WiFi interleaver. As more rates become available, the gain of our schemes will further increase.

VII. Related Work

Rate adaptation has received significant research attention (*e.g.*, [3], [5], [10], [18], [21], [13], [20], [23], [24]). Several rate adaptation schemes use loss rates for rate selection. For example, SampleRate [3] uses probes to select the rate that minimizes the expected transmission time. ONOE [18] in MadWiFi estimates long-term loss rate and uses thresholding for rate selection. RRAA [24] uses a short-term loss rate estimation to compute the throughput and select the rate. Loss rates require a significant number of transmissions in order to measure accurately, so it cannot keep up with the changing wireless channel.

SNR-based scheme is attractive because one can select rate based on SNR of a single frame. The traditional SNR based scheme uses average SNR to select the data rate, and is well known to yield inaccurate selection. More recently, [10] proposes effective SNR based rate adaptation, and shows it improves delivery rate estimation and resulting throughput.

There are a number of approaches that leverage physical layer information. For example, SoftRate [23] develops a rate adaptation that uses PHY-layer hint. [21] leverages the dispersion between the transmitted and received symbols to derive the rate at which the frame could have been transmitted. It focuses on the impact of modulation on delivery rate and does not consider the FEC, which is our focus.

The work closest to ours is [17], which computes packet delivery rate using a metric, called error event probability (EVP). Our independently-developed lookup based approach is close to EVP based approach in spirit, though differs in details. We also advance [17] by using machine learning to significantly speed up the calculation and developing a CSI-aware interleaver to explicitly minimize bursty errors.

VIII. Conclusion

Achieving predictability in wireless performance has been an elusive goal even with complete channel information. This paper reveals an important factor that causes significant variability in wireless performance – significant bursty errors despite the use of an interleaver. We develop two delivery rate estimation techniques that explicitly take into account the bursty errors and improve estimation accuracy. In 20 MHz channels, their average errors are around $3-5\%$ and exceed 10% in only 3%–11% of the cases. The ML is a practical and accurate scheme due to its efficiency. In comparison, effective SNR has an average error of $11-27\%$ and has error exceed 10% for 41%–77% of the cases. The improved accuracy of our estimator allows us to more effectively optimize and manage wireless networks. We further develop a new interleaver to reduce the bursty error, and a rate adaptation scheme that incorporates both enhanced delivery rate estimation and interleaver. Our evaluation shows these approaches are effective in improving the predictability, throughput, and energy of wireless networks.

Acknowledgements: This work is supported in part by NSF Grants CNS-1343383 and CNS-1547239.

References

[1] LAN/MAN Standards Commmittee of the IEEE Computer Society. Part 11: Wireless LAN Medium Access Control and Physical Layer (PHY) Specifications. *IEEE Standard 802.11*, 2009. http://standards.ieee.org/getieee802/download/802.11n-2009.pdf.

[2] A. Bhartia, Y.-C. Chen, S. Rallapalli, and L. Qiu. Harnessing frequency diversity in WiFi networks. In *In Proc. of ACM MobiCom*, Sept. 2011.

[3] J. Bicket. Bit-rate selection in wireless networks. In *MIT Master's Thesis*, 2005.

[4] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995.

[5] J. Camp and E. Knightly. Modulation rate adaptation in urban and vehicular environments: cross-layer implementation and experimental evaluation. In *Proc. of ACM MobiCom*, 2008.

[6] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, Dec. 1989.

[7] V. E. etc. Ieee 802.11 wireless lans: Tgn channel models. 2004.

[8] H. P. Gavin. The levenberg-marquardt method for nonlinear least squares curve-fitting problems. http://people.duke.edu/~hpgavin/ce281/lm.pdf.

[9] A. Gudipati and S. Katti. Strider: automatic rate adaptation and collision handling. In *Proc. of ACM SIGCOMM*, 2011.

[10] D. Halperin, W. Hu, A. Sheth, and D. Wetherall. Predictable 802.11 packet delivery from wireless channel measurements. In *Proc. of ACM SIGCOMM*, 2010.

[11] D. Halperin, W. Hu, A. Sheth, and D. Wetherall. Tool release: Gathering 802.11n traces with channel state information. *ACM SIGCOMM Computer Communication Review (CCR)*, Jan. 2011.

[12] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[13] G. Holland, N. Vaidya, and P. Bahl. A rate-adaptive MAC protocol for multihop wireless networks. In *Proc. of ACM MOBICOM*, Jul. 2001.

[14] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Netw.*, 4(2):251–257, Mar. 1991.

[15] S. G. Kate Lin and D. Katabi. Random access heterogeneous MIMO networks. In *Proc. of ACM SIGCOMM*, 2011.

[16] M. O. Khan, V. Dave, Y.-C. Chen, O. Jensen, L. Qiu, A. Bhartia, and S. Rallapalli. Model-driven energy-aware rate adaptation. In *Proc. of ACM MobiHoc*, July 2013.

[17] Z. Li, Y. Xie, M. Li, and K. Jamieson. Recitation: Rehearsing wireless packet reception in software. In *Proc. of ACM MobiCom*, 2015.

[18] ONOE rate control. http://madwifi.org/browser/trunk/ath_rate/onoe.

[19] H. S. Rahul, S. Kumar, and D. Katabi. Jmb: Scaling wireless capacity with user demands. In *Proc. of ACM SIGCOMM*, 2012.

[20] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. Knightly. Opportunistic media access for multirate ad hoc networks. In *Proc. of ACM MOBICOM*, Sept. 2002.

[21] S. Sen, N. Santhapuri, R. R. Choudhury, and S. Nelakuditi. Accurate: Constellation based rate estimation in wireless networks. In *Proc. of USENIX NSDI*, 2010.

[22] A. J. Viterbi. Principles of digital communication and coding. 1979.

[23] M. Vutukuru, H. Balakrishnan, and K. Jamieson. Cross-Layer Wireless Bit Rate Adaptation. In *ACM SIGCOMM*, 2009.

[24] S. H. Wong, H. Yang, S. Lu, and V. Bharghavan. Robust rate adaptation in 802.11 wireless networks. In *Proc. of ACM MOBICOM*, Sept. 2006.

[25] J. Zhang, K. Tan, J. Zhao, H. Wu, and Y. Zhang. A practical SNR-guided rate adaptation. In *Proc. of IEEE INFOCOM*, 2008.