Efficient Error Estimating Coding: Feasibility and Applications*

Binbin Chen Ziling Zhou Yuda Zhao Haifeng Yu
National University of Singapore
Republic of Singapore
{chenbinb, zhouzl, zhaoyuda, haifeng}@comp.nus.edu.sg

ABSTRACT

Motivated by recent emerging systems that can leverage partially correct packets in wireless networks, this paper investigates the novel concept of error estimating codes *(EEC). Without correcting the errors in the packet, EEC enables the receiver of the packet to estimate the packet's bit error rate, which is perhaps the most important meta-information of a partially correct packet. Our EEC algorithm provides provable estimation quality, with rather low redundancy and computational overhead. To demonstrate the utility of EEC, we exploit and implement EEC in two wireless network applications, Wi-Fi rate adaptation and real-time video streaming. Our real-world experiments show that these applications can significantly benefit from EEC.*

Categories and Subject Descriptors

E.4 [**Coding and Information Theory**]: Error control codes; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design – Wireless communication

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Error Estimating Coding, Bit Error Rate, Partially Correct Packet, Partial Packet, Error Correcting Coding

1. INTRODUCTION

Error correcting codes [22] have long been playing a key role in serving the performance and reliability needs in wireless networks. Over the years, researchers have proposed numerous interesting error correcting codes. The traditional philosophy behind error correction is that the application/ network can or should only use/relay completely correct packets.

Recent advances in wireless networking, however, have invalidated this assumption. In particular, many designs [9, 10, 12, 18, 19, 21, 34, 37] now leverage information in a packet that is partially correct (i.e., some bits are correct but others are corrupted). Such a *partial packet* can be useful when:

- The destination may be able to obtain incremental redundancy from the source to recover the partial packet (i.e., incremental redundancy ARQ) [21].
- The destination may collect and combine multiple partial packets to obtain a correct copy [9, 12, 18].
- The packet has *forward error correction* (i.e., pre-encoded with error correcting codes), and thus can potentially fully recover the errors. For example, forward error correction is often used in real-time video streaming to tolerate errors in wireless networks [10, 37].
- The application may be able to directly use partial packets to some extent. For example, for image or video packets, a partially correct packet can still carry useful information (where the information amount depends on the number of errors in the packet) [19, 34].

Given these designs, a natural question is whether there is a benefit looking beyond error correcting codes.

This paper takes the first step in answering this question, by investigating the novel concept of *error estimating codes* (EEC). Without actually correcting the errors in the packet, EEC enables the receiver to estimate the *fraction* of corrupted bits in the packet, which is perhaps the most important meta-information of a partial packet. We call such fraction as the the packet's *bit error rate* or *BER*. Here the receiver of the packet may or may not be the packet's final destination. In particular, it can be a wireless router that is *oblivious* to how the application will eventually use/recover the partial packet.

The utility of EEC depends on two key questions:

Feasibility Is it possible to construct highly efficient EEC? In particular, EEC's redundancy and computational overhead must be substantially smaller than error correcting codes, since otherwise one should just directly use error correcting codes to correct the errors.

Applications Does the BER meta-information provided by EEC significantly benefit upper-layer applications?

Affirmative answers to these two questions would imply that EEC indeed achieves a new and interesting tradeoff point, on the spectrum between overhead and functionality. Such tradeoff point was not previously available with error correcting codes.

Efficient EEC: Feasibility. This paper provides affirmative answers to both questions above, thus confirming the utility of the

*The first two authors of this paper are alphabetically ordered.

SIGCOMM'10, August 30–September 3, 2010, New Delhi, India.

novel concept of EEC. First for feasibility, we propose a novel EEC algorithm that only needs to add $O(\log n)$ extra bits to the packet (n being the number of data bits) to estimate BER. As concrete numerical examples in the two EEC applications that we implement (described later), the relative redundancy added to each packet is only about 2%. In fact in cases where we only need to estimate whether the BER exceeds a certain threshold, the redundancy added by EEC is only 4 bytes (as in one of our two applications). Such a small redundancy makes it possible to even view EEC as generalized CRC. Namely, CRC tells us whether the BER exceeds 0, while EEC can tell whether the BER exceeds any given threshold. We also trivially show that error correcting codes, in order to correct the errors, would require much higher redundancy.

In addition to low redundancy, our EEC algorithm also incurs rather small $O(n)$ computational overhead. Experiments show that on a typical hardware platform (Soekris net5501-70[1]) for wireless mesh networks, our software-implemented EEC algorithm can process packets at maximum 802.11a/g data rate. Again, we trivially show that software-implemented error correcting codes (such as Reed-Solomon codes [22]) would be much slower (10 to 100 times slower), which prevents today's commercial 802.11 devices from using these codes in software at Wi-Fi data rate.

In terms of estimation quality, our EEC algorithm provides formal and provable guarantee on the BER estimation accuracy. The algorithm does *not* make any assumption on the positions of the corrupted bits in a packet, and in particular, does *not* assume independent errors. The corrupted bits may be correlated in an arbitrary and unknown way (e.g., fully clustered or widely spread).

Achieving these salient properties in our EEC algorithm is non-trivial. At a high level, the algorithm defines logarithmic number of randomized groups of data bits (in the packet), with geometrically distributed group sizes. It then infers the total number of errors in the packet based on the *parity* of the number of the errors (i.e., whether the number is odd or even) in each group. It may appear that parity provides rather limited information – it cannot even distinguish 3 errors from 1 error. Interestingly and as the key technical step in the design, we show that under certain conditions that our design meets, even rather limited (i.e., logarithmic) amount of parity information can almost tell us the *exact* number of errors (instead of just the parity) in a group.

Efficient EEC: Applications. To see whether EEC helps upper-layer applications, we first sketch out how the BER information provided by EEC can naturally be beneficial in a number of scenarios [2, 7, 14, 16, 18, 20, 21, 23, 25, 38, 32]. Specifically, EEC can enable new techniques such as BER-aware packet retransmission/scheduling/forwarding, BER-aware routing, and wireless carrier selection based on per-packet BER.

Out of these, we implement two representative applications (Wi-Fi rate adaptation and real-time video streaming) and incorporate our EEC implementation into these two applications. In Wi-Fi rate adaptation, EEC enables the system to adapt rate based on the fine-grained and direct per-packet BER information provided by EEC. This helps to achieve much better rate adaptation than previous schemes [2, 11, 38] that rely on course-grained packet loss statistics or indirect measures such as Signal-to-Noise Ratio. In multi-hop real-time video streaming with forward error correction [10, 37], EEC enables the intermediate forwarding wireless routers to determine whether the packet's BER exceeds the error correction threshold. A retransmission is requested if and only if the BER exceeds the threshold. Such BER-aware retransmission has clear advantage over schemes that simply forward all partial packets (in the hope that the destination can recover the packets with forward error correction) [32] or simply require retransmission to correct all partial packets.

Our real-world evaluation of these two applications clearly demonstrates EEC's utility: When compared to state-of-the-art approaches, our BER-guided Wi-Fi rate adaptation scheme achieves up to 50% higher goodput in walking scenarios and up to 130% higher goodput in outdoor challenging environments. For real-time video streaming, BER-aware retransmission achieves up to 5dB gains on the PSNR [26] of the streamed video. PSNR difference that is above 0.5dB is considered visually noticeable [31].

In the next, Section 2 explains how EEC can benefit various applications. Our EEC design, implementation, and evaluation are presented in Section 3 and 4. Section 5 and Section 6 detail the implementation and evaluation of the two applications. Finally, Section 7 discusses related work, and Section 8 draws the conclusions. Our EEC implementation source code is publicly available at http://www.comp.nus.edu.sg/~yuhf/eec_impl.html.

2. EEC APPLICATIONS

Given that many designs [9, 10, 12, 18, 19, 21, 34, 37] in wireless networks can now use partial packets, this section examines *how the BER meta-information of partial packets can benefit the application*. The simplest usage example is perhaps to use the BER to predict the amount of incremental redundancy needed, in various incremental redundancy ARQ schemes [21]. However as shown next, BER information can benefit the applications in much more interesting ways.

In the next we will explain how the sender and the receiver of a partial packet can leverage the packet's BER information, respectively. The receiver can obtain the BER information from EEC, while the sender can obtain the information via receiver feedback. The sender/receiver does not need to be the source/destination of the message, and can be a forwarding router in a multi-hop wireless network. While the source and destination are able to use/recover partial packets, we allow the sender/receiver to be *oblivious* to how the partial packets will be used/recovered (e.g., oblivious to the specific forward error correction scheme used). Allowing the routers to be oblivious is critical to supporting various applications with their different ways of using partial packets.

2.1 Using BER Information on Sender

The BER of the packets contains valuable information about the current wireless *carrier*, where the carrier can include factors such as the modulation and coding scheme, frequency band, transmission power setting, and routing path. For systems that can use partial packets, this fine-grained and direct BER information enables the sender to better (and adaptively) select a carrier with the best *goodput*. Here goodput is defined as the number of useful (application-level) bits per second that the system can transmit. The following presents some concrete examples.

Wi-Fi rate adaptation. In Wi-Fi networks, a sender has the choice over different *data rates*. Higher rate means larger number of bits transmitted per second, but also with higher probability of error. Rate adaptation thus aims to select the best data rate, dynamically based on the time-varying wireless channel condition. Previous rate adaptation schemes are often based on coarse-grain information (such as packet delivery ratio [2, 38]), or indirect information (such as SNR [13, 4]) that needs in-situ training. In comparison, the fine-grained and direct BER information provided by EEC enables the sender to better find a rate with the best goodput. Section 5 will present our implementation and evaluation of this application.

[1] http://www.soekris.com/net5501.htm

The above discussion can also be generalized to using per-packet BER information for better selection of wireless channel [25], transmission power [16], or directional antenna orientation [23].

BER-aware routing. In a multi-hop wireless network, a source can often choose among different routes to send packets to the destination, which can be viewed as one kind of carrier selection. Existing route selection schemes [7] usually consider correct (full) packet delivery only, and thus optimize for minimizing the expected number of transmissions (including retransmission of partial packets) needed to deliver the packet. For systems that can use partial packets, one would imagine that the route selection process should instead optimize for maximizing the goodput of the end-to-end route. Obviously, EEC can readily provide the BER information for each wireless link to enable such route selection.

2.2 Using BER Information on Receiver

Instead of feeding back the BER information to the sender, the receiver of a partial packet can also directly utilize such information. We focus on scenarios where the receiver itself is an intermediate router in a multi-hop wireless network. Following are some concrete examples showing how the receiver can use the BER information to make informed decisions when processing a partial packet.

BER-aware packet retransmission. Today wireless mesh networks have been widely deployed as a cost-effective way to provide Internet access for both urban [1] and rural areas [5]. To enable services such as remote learning and remote health-care, there are strong demands to support real-time multimedia applications (e.g., video chatting, video conferencing, and VoIP) in these networks [29]. Let us take real-time video streaming as an example. To deal with errors in wireless communication, the source often adds forward error correction on the packets, to avoid the extra delay involved in packet retransmission [10, 37]. With forward error correction, the receiver (router) will now simply forward all packets (correct or partial) to the next hop [32], with the hope that the final destination can recover the partial packets via error correction. But with the time-varying quality of the wireless links, it is impractical to add sufficient error correction redundancy to ensure that *all* partial packets can be recovered.

The BER information provided by EEC conveniently enables the receiver (router) to avoid this problem. Namely, the source (knowing the details of the forward error correction applied to the packet) can easily include a threshold in the packet header, indicating the maximum BER that the forward error correction can tolerate. The router can now request retransmission of those packets whose BER exceeds such threshold (which will likely be a small fraction), instead of naively relaying the packets to the next hop. Section 6 will present our implementation and evaluation of this application.

Of course, an alternative approach would be for the receiver to decode the error correcting code on the partial packet, and request retransmission if decoding fails. But this will require the router to be non-oblivious and to know the exact error correcting mechanism employed by the application. Furthermore, as we will show later, the computational overhead of error correcting codes may prevent the router from decoding (in software) at Wi-Fi data rate.

BER-aware packet scheduling. Consider a wireless image sensor network for emergency response (e.g. forest fire, flood, or earthquake). In such scenarios, the system needs to send back as much information as possible and as fast as possible [14]. For image data, a partially correct packet often still carries useful information, where the information can be a function of the packet's BER. As the data funneling to the base station, the BER information on the packets enables the sensors to prioritize the forwarding of packets with lower BER. Doing so will maximize the amount of information collected by the base station at any given time point.

BER-aware packet forwarding. In a typical setting of *cooperative relay* [18], a dedicated relay node may help one node A to better transmit packets to another node B (within A's radio range). The relay node, within the radio range of both A and B, simply relays the packets that it overhears from A to B. B will eventually combine these (potentially partial) packets. When relaying, the relay node can choose between *amplify-and-forward* (AAF) and *decode-and-forward* (DAF). DAF can remove noise before forwarding, but suffers from error propagation if the decoded packet contains many errors. AAF has the exactly opposite property. Researchers thus suggest [20] that ideally the relay should adaptively choose between the two depending on the error level of the packet. The quantitative BER information provided by EEC naturally fits such needs.

3. EEC DESIGN

This section describes our EEC algorithm and its formal guarantees. While the algorithm is elegant at the implementation level, the algorithmic ideas can be more complex. We will thus focus on the intuition, and leave the rigorous arguments to the proofs in [6].

3.1 Error Estimation Formal Framework

Let n denote the total number of data bits in a packet (see Table 1 for a summary of notations in this section). From the n data bits, the EEC encoding process will generate k *EEC bits* for error estimation later. The sender will send these $n + k$ bits in a *packet* to the receiver. Here the notion of a packet is logical: It can be an 802.11 packet, or a segment in an 802.11 packet, or multiple 802.11 packets (in which case EEC will estimate the average BER over these multiple 802.11 packets).

We model a packet as $n + k$ *slots*, where each slot holds exactly one bit. A slot may be *erroneous* and cause the bit in that slot to be flipped during transmission, and that flipped bit is called an *error*[2]. A slot that is not erroneous is called *correct*. Let p denote the fraction of erroneous slots, or equivalently, the BER of the packet[3]. Notice that p is a fraction instead of a probability. The $(n+k)p$ errors may be in arbitrary positions in the packet. In particular, the errors may be correlated in an arbitrary and unknown way (e.g., fully clustered or widely spread). The randomization used in the EEC algorithm exactly serves to deal with such (arbitrary) correlation, and our algorithm does *not* assume that the errors are independent.

The goal of EEC is to use the EEC bits to output an estimation ($\hat{p}$) for p, with certain estimation *quality*. We use the standard (ϵ, δ) guarantee as the metric for quality. The estimation $\hat{p}$ is said to be an (ϵ, δ)-approximation of p if $Pr[(1-\epsilon)p \leq \hat{p} \leq (1+\epsilon)p] \geq 1-\delta$. Here the probability is taken over the random coin flips in the randomized algorithm. To help understanding in the next, Section 3.2 through 3.5 assume $p \leq 1/4$. Section 3.6 removes this assumption, without compromising any of the guarantees achieved.

[2]Theoretically speaking, whether a bit in a slot is flipped not only depends on the slot (i.e., transmission time), but also may depend on whether the bit is 0 or 1. However, given the scrambling (randomization) and modulation steps [30] in wireless communication systems today, this will not happen on today's wireless hardware.

[3]Here p does not necessarily equal the BER of the data bits in the packet, since the packet contains both data bits and EEC bits. However, since usually the EEC bits comprise a rather small fraction of the packet (i.e., $< 5\%$) and because the EEC bits are inserted into uniformly random slots, the BER of the data bits and the BER of the whole packet make no real difference in practice.

n	# of data bits in a packet
k	# of EEC bits in a packet (i.e., $s \times l$)
s	# of EEC bits in one level
l	# of EEC levels
g	# of data bits in one group
p	fraction of erroneous slots in a packet
p_0	fraction of erroneous data bits in a packet
c_1, c_2	algorithm constants ($c_1 = 0.25$, $c_2 = 0.4$)
$\phi(x, y)$	the sum of all the odd terms in a binomial distribution $\mathbf{B}(x, y)$

Table 1: Key notations in EEC algorithm.

3.2 EEC Algorithm Overview

Our EEC algorithm has three procedures, for encoding at the sender, decoding at the receiver, and estimating BER at the receiver, respectively. These procedures are all randomized. The sender and the receiver should use the same random seed to initialize their pseudo-random number generators, so that they generate the same random sequence.

Algorithm 1 EEC Encoding Procedure.

for $i = 1$ to $\lfloor \log_2 n \rfloor$ **do**
  for $j = 1$ to $j = s$ **do**
    Select $2^i - 1$ data bits where each bit is chosen independently and uniformly randomly (with replacement) out of the n data bits;
    Compute a parity bit (as an EEC bit) for them;
  end for
end for /* Total $k = s \cdot \lfloor \log_2 n \rfloor = s \cdot l$ EEC bits. */
Place the EEC bits (in arbitrary order) into k uniformly random positions in the packet;
Place the data bits (in arbitrary order) into the remaining n positions in the packet;

The encoding procedure (Algorithm 1) adds $l = \lfloor \log_2 n \rfloor$ *levels* of EEC bits to the original data, with s EEC bits per level. Thus the total redundancy introduced is $k = l \times s$ bits. The value of s determines the estimation quality (i.e., ϵ and δ). An EEC bit at level i ($1 \le i \le l$) is simply the parity bit for $2^i - 1$ randomly chosen data bits (Figure 1). Each of these $2^i - 1$ data bits is chosen uniformly randomly and independently (with replacement) from the original n data bits. We repeat such process (independently) to obtain s EEC bits for each level. Since the encoding procedure does not modify the original data bits, decoding is trivial and thus we do not include the pseudo-code here.

Algorithm 2 EEC Estimating Procedure.

for $i = 1$ to $i = \lfloor \log_2 n \rfloor$ **do**
  Compute the fraction (q_i) of parity bits at level i that fail parity check;
  if $q_1 \ge c_2$ **then**
    Output $\hat{p} = 1/4$ and exit;
  end if
  if $c_1 < q_i < c_2$ **then**
    Output $\hat{p} = q_i / 2^i$ and exit;
  end if
end for
Output $\hat{p} = 0$ and exit;

The estimating procedure (Algorithm 2) estimates the BER of the packet. For each level i ($1 \le i \le l$) of the EEC bits, the procedure computes the fraction (denoted as q_i) of the s parity bits that fail the parity check. (Usually these q_i's will be monotonically increasing.) Then if the algorithm finds a q_i that falls within a range (c_1, c_2) where c_1 and c_2 are algorithm constants, it will estimate p to be $q_i/2^i$ and then exits. The algorithm also needs to handle two corner cases. First, at the very first level, if the algorithm finds that $q_1 \ge c_2$, it directly outputs $1/4$ as the estimation (i.e., the largest p possible since we assume $p \le 1/4$ in the section). Second, if $q_1 < c_2$ and the algorithm fails to find a $q_i \in (c_1, c_2)$, the algorithm simply outputs 0.

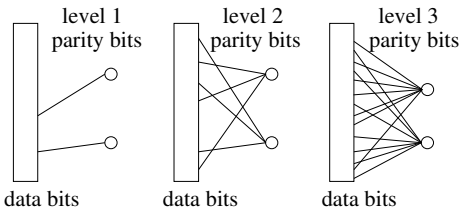


Figure 1: The first three levels of EEC bits ($s = 2$).

3.3 Using One Bit to Sample a Group of Bits

To better explain the intuition, this section first assumes that all the k EEC bits are in correct slots (i.e., they will not be flipped). We will aim to estimate the fraction p_0 of errors among the data bits (instead of the fraction p of errors among all bits). We will remove the assumption and explain how to estimate p toward the end of this section.

Naive sampling and the challenge. The first natural idea for estimating p_0 is to sample some small number of data bits, uniformly randomly out of the n data bits. If x fraction of the sampled bits are flipped, we simply output x as an estimation for p_0. To determine whether a sampled data bit is flipped during transmission, we can simply use an EEC bit (which is assumed to be in a correct slot) to replicate that data bit. We can tell whether the data bit has been flipped by comparing the EEC bit with the data bit. Equivalently, one can also insert known bits (typically called *pilot bits*) into the packet as samples.

The challenge in this naive sampling approach arises however, when p_0 is small. This is particularly relevant in error estimation context, since packet BER tends to be a small value in most cases. For example when $p_0 = 0.02$, on expectation we only see 1 error out of every 50 data bits sampled. Before seeing enough errors, the estimation quality on p_0 will be poor. To make it more concrete, in the two EEC applications that we implement, the EEC redundancy added is 36 bytes and 24 bytes per 1500-byte packet respectively. The ratio of EEC bits to data bits will only need to be about 2%. We have also performed simple experiments showing that if one were to use naive sampling to achieve similar estimation quality, the redundancy needed will be roughly 600 bytes and 450 bytes per 1500-byte packet, respectively. This translates to a ratio (of EEC bits to data bits) of above 40%.

Such drawback of naive sampling is *fundamental*. A well known lower bound [8] shows that to obtain an (ϵ, δ) estimation quality, the number of samples taken needs to reach $\Omega(\frac{1}{p_0}\frac{1}{\epsilon^2}\log\frac{1}{\delta})$. The $\frac{1}{p_0}$ term exactly shows that naive sampling will incur prohibitive overhead when BER is small.

Sample groups of data bits. The above discussion already hints that it might help if we can sample multiple data bits together. For example, imagine that we define *groups* of bits where each group has 50 data bits. Suppose that the redundancy needed to sample a group is constant and is independent of group size. Then even with

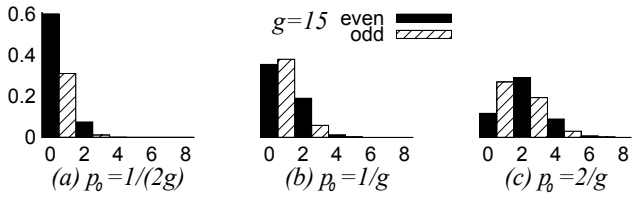


Figure 2: Probability of having x ($0 \leq x \leq 15$) number of errors in a group with 15 data bits.

low levels of redundancy, we will encounter sufficient number of errors even for $p_0 = 0.02$.

We now need to clarify exactly what information we want to obtain when sampling a group. For example, we can require that sampling a group should tell us the number of errors in a group. While such information is quite useful, it is rather hard to provide with low redundancy. Another example is that sampling a group will tell us whether the group contains any error. This information is less useful, but is also easier to provide (e.g., by adding a CRC to each group).

In our algorithm instead, we only use a single parity bit for each group. Obviously a single parity bit provides only rather limited information – it only tells us whether the number of errors in the group is odd or even. In fact, it cannot even distinguish 3 errors from 1 error. Quite interestingly, as the key step in our EEC design, we are able to show that logarithmic number of parity bits is already sufficient for error estimation.

Key step 1: Parity information is sufficient when p_0 is small enough. Consider a group with g data bits. Regardless of the positions of the erroneous slots, if we choose each of the g bits independently and uniformly randomly (with replacement) out of the n data bits, then each of them is flipped with probability p_0 independently. The number of errors in the group thus follows a binomial distribution. This is important since it means that we do not need to consider adversarial distributions.

Our first intuition is that when p_0 is small enough, then the probability mass will mostly concentrate on having 0 or 1 error, since the chance of having more than 1 error in the group is much smaller. For example for $p_0 \leq \frac{1}{2g}$ and $g = 15$ (see Figure 2 (a)), the probability of having more than 1 error is always below 0.09. This in turn means that if the parity check succeeds (i.e., the number of errors is even), then it is very likely that the number of errors is actually 0 (instead of 2, 4, or other even numbers). Similarly, if the parity check fails, it is very likely that the number of errors is exactly 1. In other words, here the parity bit almost tells us the exact number of errors in a group.

Key step 2: Parity information can be used to test whether p_0 is small enough. The above idea may appear circular since our goal is exactly to estimate p_0 and thus we do not know whether p_0 is small enough beforehand. For convenience, we define the probability of having odd number of errors in a group with g data bits as $\phi(g, p_0)$. In other words, $\phi(g, p_0)$ is the sum of all the odd terms in a binomial distribution $\mathbf{B}(g, p_0)$.

Our second intuition is that when p_0 is not small enough, then $\phi(g, p_0)$ will exceed a certain threshold. Continuing with our numerical example, if $p_0 > \frac{1}{2g}$ and $g = 15$ (see Figure 2 (b) and (c)), then $\phi(g, p_0)$ will exceed 0.32. Fundamentally, this is because when $p_0 > \frac{1}{2g}$, the mean of the binomial distribution is away from 0. This makes the sum of the odd terms and the sum of the even terms in the binomial distribution comparable. In fact, when the mean is above 1, the sum of the odd terms and the sum of the even terms should both be roughly 0.5.

Now remember that parity check on a group fails exactly when the number of errors is odd. Thus if we use multiple (independent) groups (each with a parity bit) of the same size, we will be able to estimate whether $\phi(g, p_0)$ is too large. This in turn, tells us whether p_0 is small enough.

Removing the two assumptions. So far our discussion has been assuming that the EEC bits are always in correct slots and we only estimate p_0. To remove these restrictions, our encoding algorithm (Step 7 and 8 in Algorithm 1) inserts the k EEC bits into k uniformly random positions within the packet. The n data bits will go into the remaining n positions in the packet. Each EEC and data bit now has the same probability (p) of being in an erroneous slot, though all these probabilities are correlated. Now the probability of parity failure on a group with g data bits is roughly $\phi(g + 1, p)$. It is only "roughly" because of correlation. Our formal analysis in Theorem 1 will take into account such correlation.

3.4 Geometrically Distributed Group Sizes

Single-level EEC. Putting the above ideas together gives us what we call *single-level EEC*. Here we use s independent groups, where each group consists of g data bits and 1 EEC bit. The parameter g determines the range of p that we can estimate. The receiver uses the information in the s EEC bits *twice* in the following way.

At the first step, we check the parity bits to determine the fraction q of parity failures. This fraction q can be viewed as a random variable, whose expectation is roughly $\phi(g + 1, p)$. If $q > c_2$ where c_2 being an algorithm constant, we know that p is probably too large and the parity bits do not carry much information about p. Thus the algorithm simply stops without outputting anything.

We move on to the second step when $q \leq c_2$, which implies a small p. A small p in turn implies that the parity bits almost tell us the *exact* number of errors in a group. Now we simply treat the total number of parity failures $s \cdot q$ as the total number of errors among all the $s \cdot (g + 1)$ bits, and output $\hat{p} = q/(g + 1)$.

Finally, for exactly the same reason as in naive sampling, the above estimation for p only succeeds when we see enough number of errors (i.e., when $s \cdot q$ is large enough). Thus the algorithm only outputs an estimation if $q \geq c_1$ for some algorithm constant c_1 where $c_1 < c_2$. If the condition is not met, then the algorithm again stops without outputting anything.

Roughly speaking, the single-level EEC algorithm will successfully estimate p when $q \in (c_1, c_2)$. $E[q]$ is roughly $\phi(g + 1, p)$, and $\phi(g + 1, p)$ can be shown to be monotonically increasing with p. Thus if we define p_1 and p_2 such that $\phi(g + 1, p_1) = c_1$ and $\phi(g+1, p_2) = c_2$, then the single-level EEC algorithm will be able to produce an estimation for p when $p \in (p_1, p_2)$.

Multi-level EEC. One can easily extend the single-level EEC algorithm to multi-level, so that we can properly estimate all $p \in [1/n, 1/4]$. All we need is to use $l = \lfloor \log_2 n \rfloor$ *levels* of groups, where a group at the ith level has $g = 2^i - 1$ data bits and 1 parity bit. Our goal is to ensure $\forall p \in [1/n, 1/4]$, there always exists some level i such that $\phi(2^i, p)$ falls within (c_1, c_2). Figure 3 provides some numerical examples for $\phi(2^i, p)$ at different level i, and highlights which $\phi(2^i, p)$ falls within (c_1, c_2).

The fundamental reason why our goal can be achieved is that $\phi(2^i, p)$ monotonically increases with i, and the increase rate is well bounded. Specifically, to achieve our goal, we first set c_2 such that $\phi(2, p) < c_2$ for all $p \leq 1/4$. This guarantees that regardless of how large p is, $\phi(2^i, p) < c_2$ at least at the first level. Second, we set c_1 such that $\phi(2^{\lfloor \log_2 n \rfloor}, p) > c_1$ for all $p \geq 1/n$. This guarantees that regardless of how small p is, $\phi(2^i, p) > c_1$ at least at the last level.

$\phi(2^i, p)$	$i=1$	2	3	4	5	6
$p=0.25$	0.38	0.47	0.50	0.50	0.50	0.50
$p=0.05$	0.095	0.17	0.28	0.40	0.48	0.50
$p=0.01$	0.020	0.039	0.075	0.14	0.24	0.36

Figure 3: $\phi(2^i, p)$ (which is also roughly the expectation on q_i) at level i ($1 \le i \le \lfloor \log_2 n \rfloor$), for $n = 100$ and under different p values. We also highlight the $\phi(2^i, p)$ that falls within ($c_1 = 0.25, c_2 = 0.4$).

Finally, we ensure sufficient gap between c_1 and c_2, so that some $\phi(2^i, p)$ will fall between c_1 and c_2. Let j be the largest i ($1 \le i \le \lfloor \log_2 n \rfloor - 1$) such that $\phi(2^i, p) \le c_1$. If such j does not exist, it already means that $\phi(2, p) \in (c_1, c_2)$. If such j exists, we require c_2 to be such that $\phi(2^{j+1}, p) < c_2$, which means level $j+1$ will be the level where $\phi(2^{j+1}, p) \in (c_1, c_2)$. The constraints so far on c_1 and c_2 are summarized below:

$$\begin{aligned} \phi(2^{\lfloor \log_2 n \rfloor}, p) &> c_1 \\ \phi(2, p) &< c_2 \\ \phi(2^{j+1}, p) &< c_2, \text{ where } j \text{ is the largest } i \\ & \qquad \text{such that } \phi(2^i, p) \le c_1 \end{aligned}$$

One can show that the above constraints can be satisfied by all c_1 and c_2 where $c_1 < 0.3$, $c_2 > 0.375$, and $c_2 > 2c_1(1-c_1)$. For better estimation quality, c_1 should be as large as possible (so that the number of errors we see when estimating p is large), while c_2 should be as small as possible (so that the parity information can better predict the exact number of errors). Thus in our EEC algorithm, we simply pick $c_1 = 0.25$ and $c_2 = 0.4$.

Flexible number of levels. Many applications (including the two applications that we implement) needs BER estimation only when p is within some range $[a, b]$. For example, in our Wi-Fi rate adaptation application, the application only needs to estimate p when $p \in [1/1000, 0.15]$. For $p > 0.15$ (or $p < 1/1000$), all the application needs to know is that p is close to or above 0.15 (or close to or below $1/1000$). Our second real-time video streaming application does not even need to estimate p — all it needs to know is that whether p exceeds 0.01. One can consider that this application only needs BER estimation when $p \in [(1-\epsilon)\cdot 0.01, (1+\epsilon)\cdot 0.01]$.

For these applications, it is possible to avoid using all $\lfloor \log_2 n \rfloor$ levels, and to further reduce the redundancy of EEC. In some cases (e.g., in real-time video streaming), the reduction enables us to use only one EEC level with 32 bits. Such low redundancy is even comparable to CRC overhead.

To see which levels we should keep, we only need to check which levels will be used at Step 7 of Algorithm 2 for $p \in [a, b]$. Let l_1 be the level used when $p = b$, which means that roughly $\phi(2^{l_1}, b) \in (c_1, c_2)$. Solving the equation will give us l_1. Similarly, we find l_2 such that $\phi(2^{l_2}, a) \in (c_1, c_2)$. The algorithm only needs to keep all levels from level l_1 to level l_2 (both inclusive).

An optimization — Adjustment at Step 7. We explained earlier that when q_i falls within the proper region, if the parity check fails, the number of errors in the corresponding group is most likely to be 1. However, the probability of having other odd number of errors is not 0. Thus to be more accurate, instead of simply estimating p as $q_i/2^i$, Step 7 now directly solves the p from the equation $\phi(2^i, p) = q_i$ as the final estimation. The solution [6] for this equation is $p = (1 - (1-2q_i)^{2^{-i}})/2$.

3.5 Formal Guarantees

We will prove that in order to produce an (ϵ, δ) approximation for p with constant ϵ and δ, EEC only need to add $O(\log n)$ EEC bits (or more specifically, $O(\log n)$ levels with $O(1)$ bits per level) to the original n data bits.

THEOREM 1. *Consider any given positive constants ϵ and δ. For sufficiently large n, there exists constant $s = O(1)$ such that using s in our EEC algorithm (together with input a and b where $1/n \le a < b \le 1/4$) will provide the following guarantee: With probability at least $1-\delta$,*

- *If $p \in [a, b]$, output $\hat{p}$ where $\hat{p} \in [(1-\epsilon)p, (1+\epsilon)p]$.*
- *If $p < a$, output $\hat{p}$ where $\hat{p} \le (1+\epsilon)a$.*
- *If $p > b$, output $\hat{p}$ where $\hat{p} \ge (1-\epsilon)b$.*

The theorem's proof is available in [6]. The conditions of n being sufficiently large, $1/n \le a$, and $b \le 1/4$ are not actually necessary for the theorem to hold, if we apply *virtual padding* (described in the next section) to the packet before it is encoded. Our next theorem is about EEC's computational overhead, whose proof is trivial:

THEOREM 2. *The EEC encoding, decoding, and estimating time complexity are all $O(n)$.*

3.6 Further Discussion

Virtual padding is mainly for theoretical interests and serves to remove the artificial conditions in Theorem 1. Conceptually, virtual padding pads additional dummy bits to the data bits for encoding and for estimating BER. The sender does *not* send these dummy bits to the receiver, thus the padding does not increase the redundancy added. See [6] for details on virtual padding.

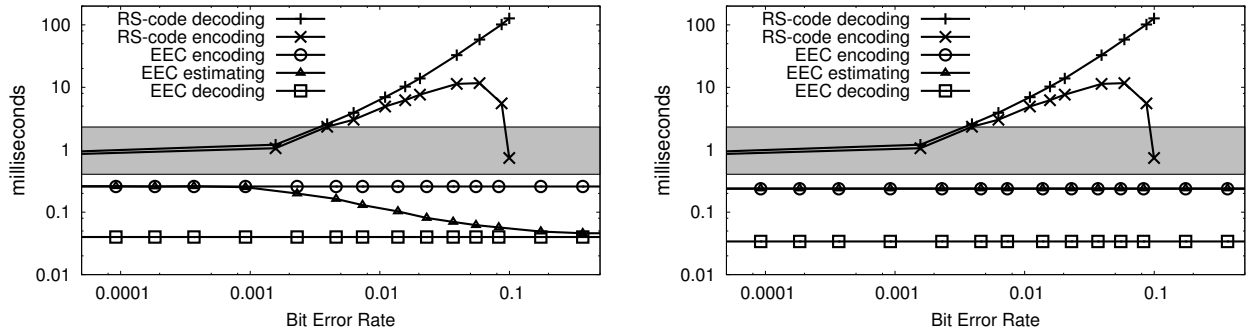
Second, when estimating p, currently our algorithm only uses the information (i.e., q_i) from one particular level i. One would imagine that while the information from other levels is not of as high quality, it can still be useful. Our technical report [6] explains how to combine information from more than one level for better estimation, together with formal proofs.

Finally, when implementing EEC, we apply standard optimizations (such as pre-computation and lookup tables) that are commonly used for implementing other codes (e.g., Reed-Solomon codes [22]). We leave the details to [6].

4. EEC'S REDUNDANCY AND COMPUTATIONAL OVERHEAD

This section quantifies the redundancy and computational overhead of EEC in practical scenarios, and further compares against error correcting codes. Since error correcting codes provide stronger functionality than EEC, any comparison here will be an apple to orange comparison. Rather, our comparison intends to show that EEC provides a new interesting point on the tradeoff spectrum between overhead and functionality.

Obviously, the overhead of the codes depends on the relevant parameters (e.g., EEC overhead will be close to zero when $s = 1$). To be meaningful, we quantify EEC's redundancy and computational overhead under the EEC parameters that we use later in our two applications. Conveniently, the EEC parameters in those two applications happen to differ substantially (due to different application needs), allowing a more comprehensive comparison. See Section 5 and 6 for why the two applications use these particular EEC parameters. There are many kinds of error correcting codes, and we use



(a) Under parameters in Wi-Fi rate adaptation

(b) Under parameters in video streaming

Figure 4: EEC's computational overhead for BER ranging from $1/20000$ to $1/2$, with both axes in log-scale. Notice that with 1500-byte packet size, the smallest possible non-zero BER is $1/12000$. The grey area corresponds to the time available for processing a packet under 802.11a/g data rate (6-54Mbps). Since RS codes need $10p$ relative redundancy to recover a packet with p BER (see text), here RS codes cannot correct BER above 0.1.

Reed-Solomon codes (RS codes) [22] as an example. We choose RS codes because it is particularly suitable for low computational overhead software implementation, while many other error correcting codes often need to be implemented in hardware. Since we need to deal with general flipping errors, erasure codes [3, 24, 27, 33] are not applicable.

EEC's redundancy. Our first application, Wi-Fi rate adaptation, is concerned with estimating packet BER within the range of $[1/1000, 0.15]$. Thus we use 9 EEC levels, with $s = 32$ bits per level. The relative redundancy added to a 1500-byte packet is thus $(9 \times 32)/(1500 \times 8) = 2.4\%$. Our second application, real-time video streaming, needs to determine whether the BER of individual 240-byte block within a packet is above a certain threshold. Thus we use a single EEC level with $s = 32$ bits. This adds extra 4-byte to each block, with a relative redundancy of around 2.0%.

It is easy to imagine that the redundancy needed by error correcting codes is much larger. For RS codes, the redundancy needed to recover a packet depends on the packet BER and the size of the RS *symbols*. Each RS symbol is simply a certain number of bits. Let *symbol error rate* (SER) be the fraction of corrupted symbols. Given a packet BER p, the SER of the packet is almost never below $2p$ (since even if we randomly set each bit to 0 or 1, half of the bits in a symbol remains correct).

Depending on the RS symbol size, a 1500-byte packet may need to hold multiple RS codewords. Different codewords may have different SER (some above $2p$ and some below $2p$), and we usually need to add sufficient redundancy to recover the codeword with the most error. We capture such factor using the results from [21]. Their results characterize the ratio ρ between the largest SER of a codeword in a packet to the SER of that packet, in real-world scenarios. Under symbol size of 8, 6, and 4, this ratio ρ is 2.5, 7.4, and 35.1, respectively. We have also independently confirmed these results with our own wireless traces. For symbol size ≥ 11, a 1500-byte packet only holds one codeword, thus the ratio is 1. However, such large symbol size also incurs substantially higher computational overhead (see below).

RS codes can recover one corrupted symbol with two redundancy symbols. Given that the largest SER of a codeword in a packet is $2\rho p$, the relative redundancy needed for RS codes to recover a packet with BER p is $4\rho p$. Even for symbol size of 8, this will be $10p$. If one were to use the 2.4% EEC redundancy for error correction via RS codes, it would only recover packets with BER below 0.24% in such a case.

EEC's computational overhead. Figure 4 presents EEC's computational overhead under the two settings in the two applications, when processing 1500-byte packets. For the second setting, it is for processing all EECs on all blocks in a packet. These results are obtained on Soekris Net5501-70 platform with a 500MHz Geode LX single chip processor, which is a typical platform for wireless mesh networks.

EEC encoding and decoding overheads are independent of the packet BER. Estimating BER involves verifying the EEC bits. This overhead is independent of BER in the video streaming setting (Figure 4(b)), since we only use a single level EEC there. In the Wi-Fi rate adaptation setting (Figure 4(a)), the algorithm verifies the 9 EEC levels sequentially from the first to the last. For large BER, the algorithm may be able to exit before verifying all levels and thus the computational overhead decreases with BER.

To put these overheads into context, Figure 4 also plots the time available for processing a packet under the 802.11a/g data rates (i.e., from 6Mbps to 54Mbps). The results show that the computational overheads of our pure software implementation of EEC are small enough to support even the highest data rate.

Figure 4 further includes the computational overhead of RS codes (with symbol size 8) for correcting a packet with a given BER p (i.e., using redundancy $4\rho p$). We use the RS codes implementation from *DSP and FEC Library* v3.0.1[4]. The decrease of RS encoding time when BER exceeds 5% (or relative redundancy exceeds 50%) is due to RS codes' inherent properties. The results show that on our platform, RS codes overheads are too large to support the 54Mbps data rate. Even for 6Mbps, the overhead can keep up with the data rate only when BER is rather small. These observations are consistent with previous work that needs to use a PC with 3.0GHz CPU (instead of typical wireless routers) to perform RS encoding/decoding at 802.11 data rates [21]. Thus we believe that EEC, with its 10 to 100 times lower computational overhead, is an interesting approach that can provide useful meta-information about the packets at Wi-Fi data rate.

Finally, we have also experimented with other RS symbol sizes (results not included in Figure 4 for clarity). Our results show that smaller RS symbol sizes result in similar computational overhead as symbol size of 8, because the increased ρ factor roughly offsets the computational overhead reduction from smaller symbol sizes. Larger symbol size (e.g., 11) will incur much higher overhead (up to 4 times), except in corner cases where BER is below 0.04%.

[4]http://www.ka9q.net/code/fec/

5. EEC APPLICATION: WI-FI RATE ADAPTATION

This section presents the implementation and evaluation results of *EEC-Rate*. EEC-Rate is a rate adaptation scheme that uses EEC to guide Wi-Fi rate change decisions. It is designed for systems that can use partial packets. A concrete example of such a system would be large unicast transfers over wireless network [15], where partial packets are recovered via end-to-end error correction. For these systems, packet-level throughput (which only counts correct packets) fails to capture the system goodput.[5] Thus we will directly use goodput as the measure of goodness. For a packet with BER p, we assume that the fraction of application-levels bits that can be recovered is $(1-\gamma p)$, where the constant γ depends on how the application utilizes partial packets. For example, if the application uses forward error correction with RS codes with symbol size of 8, then to correct a BER of p, the relative redundancy needs to be about $10p$ (as explained in Section 4). In such a case, $\gamma = 10$, since only $(1-10p)$ fraction of the packet is application-level bits. We assume that the source of each packet includes γ in the packet header, to expose this information to the wireless routers for better rate adaptation. Except that, we allow the routers to be oblivious to how the application uses partial packets.

5.1 EEC-Rate Design and Implementation

Existing approaches. Over the years, researchers have proposed many different rate adaptation schemes, and we do not intend to provide a complete survey here. For comparison purpose, we consider three representative prior rate adaptation schemes: SampleRate [2], Robust Rate Adaptation Algorithm (RRAA) [38], and Receiver-Based AutoRate (RBAR) [11]. We do not consider SoftRate [35], which uses SoftPHY [12] and thus requires special hardware not commercially available today.

SampleRate and RRAA both adjust rates based on packet loss statistics. SampleRate sends packets at different data rates periodically to obtain packet loss statistics on those rates, while RRAA adjusts rate purely based on the statistics from the current rate. In RBAR, the receiver measures the SNR of the RTS packet received. This information is then piggybacked on the CTS packet to the sender for it to adjust rate.

EEC-Rate overview. We design EEC-Rate by combining key ideas from RRAA and SoftRate. As in SoftRate, we modify RRAA to use packet BER information instead of packet loss statistics. Different from SoftRate, EEC-Rate obtains the BER information from EEC instead of from SoftPHY. Also, EEC-Rate does not use the BER under one rate to predict the BER under other rates (since we are not clear about the prediction accuracy).

Fundamentally, EEC-Rate is able to achieve better goodput than the other schemes [2, 11, 38] for similar reasons why SoftRate outperforms them [35]. Namely, SoftRate and EEC-Rate can leverage the fine-grained and direct BER information, while other schemes only use coarse-grained packet loss statistics [2, 38] or indirect SNR information [11].

EEC-Rate design details. In EEC-Rate, each packet comes with EEC which allows the receiver to estimate the packet BER. EEC-Rate maintains a moving BER average $\bar{p}$ of the recent (both correct and partial) packets, with a weighting factor of 0.2 for the most recent packet. Given two consecutive packets sent at the same rate, if the second packet's BER minus the first packet's BER is larger than 0.1, then the second packet is considered as being interfered and will not be included in the moving average. This is similar to SoftRate [12], which also excludes interfered packets.

Similar to RRAA and SoftRate, EEC-Rate maintains two constants (α_i and β_i) and one variable (w_i) for each rate R_i. If $\bar{p} > \beta_i$, then the sender will decrease rate from the current rate R_i to the next lower rate R_{i-1}. The value of β_i is obtained by solving $R_i \times (1-\gamma\beta_i) = R_{i-1}$. This means that when $\bar{p} > \beta_i$, the goodput at R_i is likely to be smaller than the goodput at rate R_{i-1}, if the BER will be zero at rate R_{i-1}. Thus it may be better to decrease rate. The current rate R_i is increased if $\bar{p} < \alpha_i$ and if the number of packets received at rate R_i exceeds w_i. Here w_i (initialized to 2) is a dynamic window size to limit excessive rate-increase attempts. A rate-increase attempt *fails* if after rate increase, the protocol immediately decreases rate due to the high BER at the higher rate. Each failed rate-increase attempt from R_i doubles w_i (with a cap of 32). Any successful rate-increase attempt brings w_i back to 2. Notice that here the fine-grained BER information enables EEC-Rate to use a smaller window than RRAA. Following [38], we set the rate-increase threshold $\alpha_{i-1} = \beta_i/3$ for all i.

For 802.11a/g data rates and $\gamma = 10$ (i.e., a packet with BER of p contains $(1-10p)$ fraction of recoverable application-level bits), the previous two formulas yield α_i and β_i values ranging from 0.2% to nearly 5% (for different i). Together with the need to detect interference, EEC-Rate thus needs to estimate BER ranging from 0.1% to 15%. EEC-Rate uses 9 levels of EEC bits to do so, where each level has 32 bits. This is sufficient to provide an average relative estimation error (i.e., average over $|\hat{p}-p|/p$) of roughly 30%. Finally, the receiver in EEC-Rate feeds back the BER to the sender in a way that balances timeliness and overhead, which is detailed in our technical report [6].

Implementation. We have implemented EEC-Rate in MadWifi 0.9.4[6]. We disable the default MAC-layer auto retransmission for failed packets. For comparison, we have also implemented RRAA [38] and an SNR-based scheme following the design of RBAR [11]. Since we cannot modify firmware to feedback SNR at MAC layer as in RBAR (which is a limitation of RBAR itself), we feedback the SNR asynchronously as in EEC-Rate. Our SNR-based protocol also incorporates a key salient feature from CHARM [13] and uses the weighted moving average of SNR across multiple packets. Since we already have low overhead asynchronous feedback [6], we do not need to rely on the channel reciprocity assumption in CHARM. When evaluating the SNR-based scheme, we always carefully train the rate-to-SNR-threshold mapping before our experiments. Finally for SampleRate, we directly use the implementation from the MadWifi driver, except that we use one second as the interval over which transmission time averages are computed, since it gives SampleRate a better performance [36].

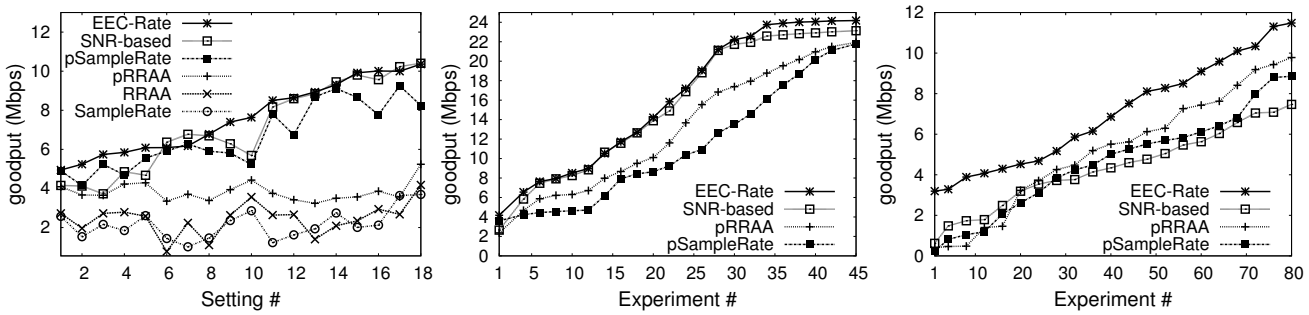
The original versions of SampleRate and RRAA equate partial packets to lost packets. In our evaluation, to make our results pessimistic, we also consider simple optimizations to the original versions so that partial packets are not treated as lost packets. We call these optimized versions as pSampleRate and pRRAA, respectively. Specifically, the sender in pSampleRate and pRRAA is informed (through asynchronous feedback as in EEC-Rate) of the delivery of partial packets. The BER of the partial packets, however, is not fed back since they do not use EEC.

5.2 Evaluation Results

We compare the goodput achieved by EEC-Rate, RRAA, SampleRate, pRRAA, pSampleRate, and the SNR-based scheme, under $\gamma = 10$ (see earlier discussion for the rationale behind this γ

[5] As defined in Section 2, goodput refers to the number of useful application-level bits that the system can transmit per second.

[6] http://madwifi-project.org



(a) Indoor with Interference (b) Walking (c) Outdoor

Figure 5: Goodput of different rate adaptation schemes under various settings.

value). All experiments are performed using Soekris net5501-70 routers with 802.11a/b/g Mini-PCI (Wistron CM9) cards. The bit errors that we observe in our experiments are often bursty.

Indoor environment with interference. Our first set of results are obtained on 6 different wireless links in our indoor mesh network testbed. We experiment with each link under three different transmission power levels (5dBm, 10dBm, and 15dBm). In experiment, the sender continuously sends 1500-byte packets for 30 seconds. For a given link and transmission power, we evaluate each rate adaptation scheme in 10 independent experiments (in round-robin fashion), and then compute the average goodput.

Figure 5(a) presents the goodput when the links operate on 802.11g channel. These experiments are subject to strong interference from an overlapping campus Wi-Fi network and a dozen of other access points nearby. For clarity, the 18 different settings (link $\times$ transmission power) in the figure are sorted in increasing order of their goodput under EEC-Rate. Among the 6 schemes, EEC-Rate achieves the highest goodput, while the SNR-based scheme and pSampleRate achieve similarly high goodput as well. RRAA and SampleRate have the lowest goodput since they are unaware of the delivery of partial packets, and thus often operate on inappropriate rates. In all our later experiments, their performance is still always worse than pRRAA and pSampleRate, respectively. Thus for clarity, we will not report their results further. pRRAA's goodput is significantly below EEC-Rate's goodput, because the interference triggers pRRAA to decrease rate frequently. Our experiments also show that enabling RRAA's adaptive RTS/CTS mechanism [38] does not improve RRAA's or pRRAA's goodput in our setting. Similar conclusions have been made elsewhere [35] as well. For space limitations, we leave the indoor interference-free results for 802.11a channel to our technical report [6]. There all the schemes have similar performance as in Figure 5(a), except that pRRAA's performance becomes better (though still not as good as EEC-Rate).

Overall, in indoor environments, the channel coherence time tends to be large (i.e., multiple seconds) and the best rate is often quite stable, making it relatively easy to choose the best rate. Because of this, simple schemes (e.g., pSampleRate) can already obtain close to optimal performance [2], and more advanced techniques will not bring significant extra improvement.

Walking scenario. In our second set of experiments, the sender (operating on 802.11a channel) is moved at walking speed along a straight corridor. Each *walk* lasts for around 90 seconds, with the sender moving away from and then moving back to the receiver. The total walking distance is 90 meters. We perform 5 walks for each scheme (in round-robin fashion). Each 90-second walk for a given scheme is considered as nine 10-second experiments for that scheme, and we measure the goodput in each experiment.

Figure 5(b) plots the goodput measured. For clarity, the 45 goodput values (from the 45 experiments) for each scheme are plotted in increasing order. The figure shows that EEC-Rate and the SNR-based scheme achieve similarly high goodput, and outperform pRRAA and pSampleRate. The average relative goodput (across all the experiments) of EEC-Rate, as compared to pRRAA and pSampleRate, reaches 130% and 150% respectively. pSampleRate performs the worst since it allows only one rate change per second, for the purpose of stability. This prevents it from adapting promptly enough. EEC-Rate performs better than pRRAA, because per-packet BER information allows it to i) use a smaller rate adaptation window and ii) be more robust to random packet losses.

Outdoor challenging scenario. In our last set of experiments, the sender and receiver (operating on 802.11a channel) are placed on the opposite sides of a busy road and are 30 meters apart. The pedestrian speed is around 3kmph, and the vehicular speed is less than 20kmph. Our setting is similar to the residential urban outdoor environment in [4], where the average coherence time between a pair of static nodes is usually 100 ms, but passing cars can drive the coherence time down to 15 ms. Notice that Section 4 showed that evaluating a packet's BER using EEC takes less than 0.3 ms. Such processing delay is over an order of magnitude smaller than the coherence time even in such challenging environments.

For each scheme, we do 80 experiments of 10 seconds each (in round-robin fashion), and measure the goodput in each experiment. Figure 5(c) plots the goodput achieved. Again, we plot the goodput values in increasing order for clarity. Here EEC-Rate significantly outperforms the other three schemes. Its average relative goodput is about 230% when compared to pRRAA and pSampleRate, and is about 180% when compared to the SNR-based scheme. pRRAA suffers from frequent and unnecessary rate reduction caused by the large number of random packet losses in this environment. pSampleRate, on the other hand, is not able to adapt promptly enough to the fast-varying channel quality. Finally, even though we trained the rate-to-SNR-threshold mapping for the SNR-based scheme, during our experiments, the changing traffic pattern quickly renders the mapping inaccurate. This causes the goodput of the SNR-based scheme to decline, as compared to EEC-Rate.

6. EEC APPLICATION: REAL-TIME VIDEO STREAMING

This section presents the implementation and evaluation results of applying EEC in real-time video streaming over wireless mesh networks. Section 2 explained how EEC enables the routers in this application to perform *BER-aware packet retransmission* (de-

noted as BER-aware). We use no-retran to denote the existing approach [32] described in Section 2, where a partial packet is always directly forwarded to the next hop[7], in the hope that forward error correction will take care of the errors. For comparison, we further consider two other schemes, packet-retran and frag-retran, where the routers always request the retransmission of a partial packet (and forward error correction is thus not needed). packet-retran uses whole packet retransmission, while frag-retran splits a packet into multiple fragments, uses CRC on each fragment, and only retransmits corrupted fragments. We set the fragment size to 240 bytes, which gives the best results for frag-retrans in our experiments.

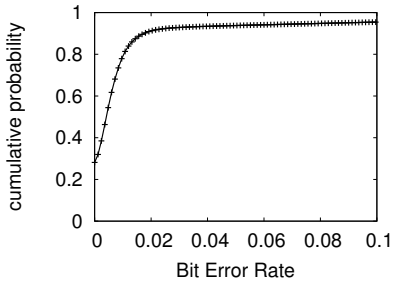


Figure 6: Packet BER distribution.

Quantify the potential gain. We want to first gain some insight into whether BER-aware will likely lead to significant improvements. Figure 6 presents the CDF for packet BER as measured on an 802.11a link with intermediate quality in our indoor mesh network testbed. Results from other researchers [21] are similar. The figure shows that most of the partial packets have BER below 2%, confirming the utility of using forward error correction. However, the distribution also has a long tail, and 5% of the packets even have BER over 10%. This implies that no-retran may greatly suffer from unrecoverable packets. The figure also implies that packet-retran and frag-retran will request retransmission for over 70% of the packets. Doing so can potentially cause more packets to miss deadline, especially when there is no idle bandwidth. Leveraging EEC, BER-aware will only need to retransmit much fewer packets, while ensuring that the packets can be recovered.

6.1 Implementation and Experimental Setting

We implement the four schemes on Soekris net5501-70 routers as a Linux kernel module to interface with MadWifi 0.9.4. We configure all wireless interfaces in monitor mode, and implement a glue layer to expose the wireless interface as an Ethernet interface to the native Linux kernel network stack. We disable the default MAC-layer auto retransmission for failed packets. Packets are transmitted in increasing order of their deadlines.

For our experiments, we set up real-time video streaming over two wireless links and three dual-radio routers ($A \rightarrow B \rightarrow C$), in an office environment. Following common practice, we let the two wireless links operate on 802.11g and 802.11a channel respectively to avoid self-interference. Both wireless links use the fixed (lowest) data rate of 802.11a/g (i.e., 6Mbps). Same as before, the bit errors in these real-world experiments are often bursty. The source (connected to A) uses EvalVid [17] to stream video to the destination (connected to C). We generate the test video at 30 frames per second from the 300-frame Foreman sequence[8], which is commonly used for such purpose. The test video is encoded using the x264 encoder under MPEG4 baseline profile setting, with one I-frame every 30 video frames.

We use RS codes (with symbol size of 8 and codeword/block size of 240 bytes) to add forward error correction to the video packets, so that the destination can recover a codeword with 2% BER. This 2% value is roughly at the knee of the BER distribution in Figure 6. The figure also shows that given the long tail, further increasing the redundancy will not likely provide significant benefit. For BER-aware, the router applies a single level EEC with 32 EEC bits to each 240-byte block in the packet. We intentionally use 32 EEC bits to demonstrate that EEC's overhead can be as low as a typical CRC while still be able to significantly benefit the end application.

6.2 Evaluation Results

Measure of goodness. We use the standard Peak Signal-to-Noise Ratio (PSNR) [26] to measure the quality of the streamed video. PSNR is defined on log-scale in terms of dB, and thus a small difference in PSNR values can be significant visually. PSNR difference above 0.5dB is usually considered visually noticeable, and the MPEG committee uses an informal threshold of 0.5dB to decide whether to incorporate a coding optimization [31]. Usually the video quality is considered "Excellent" if PSNR is above 37dB, and "Bad" if PSNR is below 20dB [17].

Results without interference. In each of our first set of experiments, we stream the test video under one specific scheme. We evaluate the different schemes in a round-robin fashion. We adjust the transmission power of the router B so that the link between B and C is often of intermediate quality due to weak signal. Because of human movements in our office environment, the link quality here can still vary non-trivially over time. On the other hand, video streaming quality as measured by PSNR can be highly sensitive to wireless channel conditions. To compare the schemes under similar channel conditions, we group the experiments into four quartiles based on the packet reception ratio (PRR)[9] of the link in each experiment.

When the channel condition is bad (i.e., PRR $\in [0, \frac{1}{4}]$ or $[\frac{1}{4}, \frac{2}{4}]$), all the schemes achieve similarly low PSNR (most experiments below 20dB), though BER-aware slightly outperforms all other schemes by around 0.5dB. Here PSNR is largely determined by whether the link can provide enough raw bandwidth to stream the video, and retransmission schemes have little effect. The poor PSNR in most of the experiments are due to the fact the link is fundamentally not able to deliver certain I-frames on time.

Figure 7 and Figure 8 present the PSNR of the different schemes under the remaining two PRR quartiles. We only plot PSNR below 40dB because video quality is considered "Excellent" for PSNR above 37dB. The difference between two PSNR values that are above 40dB bears little practical relevance.[10] For similar reasons, we focus on PSNR values above 15dB. For clarity, the PSNR values in different experiments of each scheme are plotted in increasing order.

Figure 7 shows that for channel condition with PRR $\in [\frac{2}{4}, \frac{3}{4}]$, BER-aware achieves on average about 5dB higher PSNR over other schemes. As expected, frag-retran suffers from congestion due to its higher bandwidth consumption, while no-retran suffers from many unrecoverable packets. Such a trend continues in

[7]Note that if a packet is entirely lost or if its header is corrupted, the packet will still be retransmitted.

[8]http://www.cipr.rpi.edu/resource/sequences/sif.html

[9]We count both correct and partial packets as received packets when calculating PRR.

[10]Including those PSNR values above 40dB actually would make our results better.

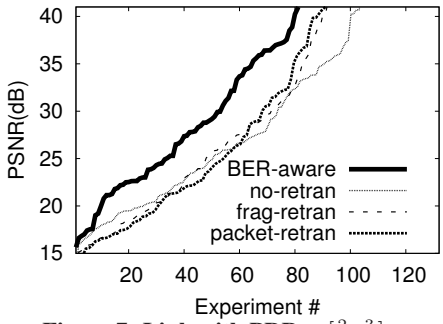


Figure 7: Link with PRR $\in [\frac{2}{4}, \frac{3}{4}]$.

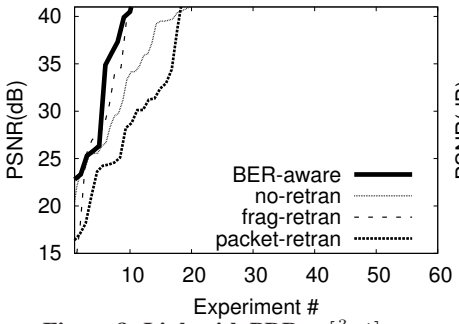


Figure 8: Link with PRR $\in [\frac{3}{4}, 1]$.

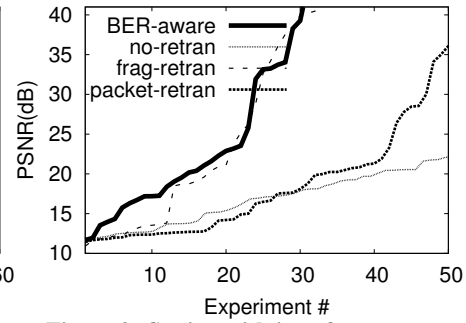


Figure 9: Setting with interference.

Figure 8, though all schemes in Figure 8 perform fairly well. This is simply because with the good link quality (PRR $\in [\frac{3}{4}, 1]$), there are few errors and even naive schemes can achieve near-optimal results.

Effects of interference. For these experiments with interference, we set up a hidden terminal to router B. To isolate the effect of interference from that of weak signal, we set B to transmit at its maximum power. Also because of this, we no longer need to categorize the experiments based on PRR. Figure 9 presents the PSNR results under such setting. The results show `BER-aware` easily out-performs `no-retran` and `packet-retran` by over 5-10dB. This is simply because `no-retran` relays many unrecoverable packets, while `packet-retran`'s aggressive retransmission causes many packets to miss their deadlines. `frag-retran` performs similarly well as `BER-aware`. Compared to the earlier results under weak signal, here the errors in a packet are much more clustered. This enables `frag-retran` to effectively correct the errors by retransmitting only a rather small number of fragments.

7. RELATED WORK

Error estimating codes. While error correcting codes have been extensively studied, we are not aware of any prior rigorous study on codes for estimating BER. The need for error estimation has been occasionally mentioned in some prior work in different contexts (e.g., [35] and [28]). The general technique of inserting known bits (called *pilot bits*) into a packet can be viewed as a naive form of EEC. Section 3.3 explained that pilot bits are poorly suited for estimating BER, which tends to be a small value such as a few percent. This limitation stems from fundamental lower bound [8] on sampling. Given certain target estimation quality (e.g., as in our two applications), pilot bits can incur over 20 times more redundancy than our EEC algorithm.

Computing parity bits over groups of data bits is a common technique in coding. However, because our goal is to estimate instead of to correct errors, our specific way of forming groups and using the parity information is rather different from other coding algorithms. For example, in terms of algorithmic technique, the most relevant coding algorithm to our EEC algorithm is perhaps low-density parity-check codes (or LDPC codes) [22], which is a kind of error correcting codes. Both LDPC codes and our EEC algorithm compute parity bits over groups of data bits. In LDPC codes, each data bit belongs to multiple groups to enable error correction. While in our EEC algorithm, bits in one group may or may not be in other groups. Such design makes low redundancy possible, but also requires a completely different and new way of properly using the parity information. The second key difference is the *degree* distribution of the parity bits, where the *degree* of a parity bit is the size of the corresponding group of data bits. LDPC codes are by definition *low density* with small average degree for the parity bits. In comparison, the degrees of our parity bits are geometrically distributed, with the average being as large as $\Theta(n/\log n)$. These high-degree parity bits are crucial for estimating small BER effectively. We are able to still achieve $O(n)$ computational overhead despite these high-degree parity bits. Similarly, our EEC algorithm is also related to many other error correcting codes, such as Tornado code [3], LT codes [24], Raptor codes [33], Online codes [27], and Growth codes [14]. Again, our EEC algorithm differs in terms of the geometrically distributed degrees and in terms of how the parity bits are actually used.

Finally, in our prior work [39] on secure aggregation queries in sensor networks, we address a similar theoretical problem of efficiently estimating a small value (i.e., the fraction of "errors", in EEC terms). There the algorithm defines groups of different sizes and then relies on knowing whether a group has *any error* (in EEC terms). In comparison, our EEC algorithm relies on knowing whether the number of errors in a group is *even or odd*. Such parity information can be efficiently obtained via a single parity bit. A second key difference is that the algorithm in [39] is interactive and requires multiple rounds.

SoftPHY. Coding is certainly not the only way of estimating BER. Recently, some researchers propose the SoftPHY [12] physical layer design, exposing a confidence level for each bit received. Such information can of course, be used to estimate BER. Today's commercial Wi-Fi hardware does not yet provide SoftPHY functionality. Furthermore, it does not seem possible to provide SoftPHY by only modifying the Hardware Abstraction Layer or firmware (and without changing the hardware). Fundamentally, this is because today's Wi-Fi chips implement physical layer functionality in non-programmable application-specific integrated circuit, and the confidence level information is internal to the physical layer.

Compared to SoftPHY, our EEC design provides a pure software based alternative. A software based approach is usually easier for adoption or for upgrading existing systems. In particular, the computational overhead of our software based approach is small enough to run at maximum 802.11a/g data rate. Even if SoftPHY becomes available on future Wi-Fi hardware, EEC will continue to be useful on lower-end wireless devices (such as sensors), where the additional hardware cost in SoftPHY (such as more output pins or wider system bus) may not justify the benefit. Finally, we also note that SoftPHY provides strictly more information than just packet BER. For applications that need the confidence level information for individual bits, EEC will not be suitable. In summary, with all these differences between EEC and SoftPHY, we believe that EEC and SoftPHY will always have their respective suitable application domains.

Using SNR in place of BER. Another alternative approach [4, 13, 40] is to use Signal-Noise Ratio (SNR) in place of packet BER,

since the theoretical relationship between SNR and BER is well-understood. Such an approach has two known drawbacks. First, to accurately measure SNR, today's commodity wireless network hardwares (e.g., Wi-Fi devices) often require hardware-specific calibration [40]. Second, the mapping from SNR to BER depends on the specific deployment. To obtain a good mapping, in-situ training of the system is often needed [4].

8. CONCLUSION

This paper is motivated by recent emerging systems that can leverage partial packets in wireless networks. We observe that such systems would significantly benefit from the BER information of the partial packets. This paper thus proposes the novel concept of error estimating codes (EEC). Without correcting the errors, EEC enables the receiver of a partial packet to estimate the packet's BER. Our EEC algorithm provides provable estimation quality, with rather low redundancy and computational overhead. We have exploited and implemented EEC in two wireless network applications, Wi-Fi rate adaptation and real-time video streaming. Our real-world experiments have demonstrated that these applications can significantly benefit from EEC.

While we have only focused on applying EEC to wireless networking in this paper, the utility of EEC can be much broader. For example, EEC's functionality can also help data storage recovery from multiple partially correct copies [28]. Generally speaking, EEC may find potential application wherever partially correct data can be utilized.

Acknowledgments

We thank Ramachandran Ramjee for shepherding this paper. We thank the anonymous reviewers, Mun Choon Chan, and Ben Leong for many helpful comments on this paper. We thank Kyle Jamieson and Brad Karp for helpful discussions. We thank Ha Yajun and Samarjit Chakraborty for answering our hardware-related questions. This work is partly supported by National University of Singapore Young Investigator Award R-252-000-334-101.

9. REFERENCES

[1] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level measurements from an 802.11b mesh network. In *SIGCOMM*, 2004.

[2] J. C. Bicket. Bit-rate selection in wireless networks. Master's thesis, MIT, 2005.

[3] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *SIGCOMM*, 1998.

[4] J. Camp and E. Knightly. Modulation rate adaptation in urban and vehicular environments: Cross-layer implementation and experimental evaluation. In *MobiCom*, 2008.

[5] K. Chebrolu, B. Raman, and S. Sen. Long-distance 802.11b links: Performance measurements and experience. In *MobiCom*, 2006.

[6] B. Chen, Z. Zhou, Y. Zhao, and H. Yu. Efficient error estimating coding: Feasibility and applications. Technical report, National University of Singapore, June 2010. Available at `http://www.comp.nus.edu.sg/~yuhf/eec-tr.pdf`.

[7] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *MobiCom*, 2003.

[8] P. Dagum, R. Karp, M. Luby, and S. Ross. An Optimal Algorithm for Monte Carlo Estimation. *SIAM Journal on Computing*, 29(5), 2000.

[9] H. Dubois-Ferriere, D. Estrin, and M. Vetterli. Packet combining in sensor networks. In *SenSys*, 2005.

[10] M. Elaoud and P. Ramanathan. Adaptive use of error-correcting codes for real-time communication in wireless networks. In *INFOCOM*, 1998.

[11] G. Holland, N. Vaidya, and P. Bahl. A rate-adaptive MAC protocol for multi-hop wireless networks. In *MobiCom*, 2001.

[12] K. Jamieson and H. Balakrishnan. PPR: Partial packet recovery for wireless networks. In *SIGCOMM*, 2007.

[13] G. Judd, X. Wang, and P. Steenkiste. Efficient channel-aware rate adaptation in dynamic environments. In *MobiSys*, 2008.

[14] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein. Growth codes: Maximizing sensor network data persistence. In *SIGCOMM*, 2006.

[15] S. Katti, D. Katabi, H. Balakrishnan, and M. Medard. Symbol-level network coding for wireless mesh networks. In *SIGCOMM*, 2008.

[16] V. Kawadia and P. R. Kumar. Principle and protocols for power control in wireless ad hoc networks. *IEEE Journal on Selected Area in Communications*, 23(5):76–88, 2005.

[17] J. Klaue, B. Rathke, and A. Wolisz. EvalVid - a framework for video transmission and quality evaluation. *Performance TOOLS*, 2003.

[18] J. Laneman, D. Tse, and G. Wornell. Cooperative diversity in wireless networks: Efficient protocols and outage behavior. *IEEE Transactions on Information Theory*, 50(12):3062–3080, 2004.

[19] L. Larzon, M. Degermark, and S. Pink. UDP Lite for real time multimedia applications. In *ICC*, 1999.

[20] Y. Li and B. Vucetic. On the performance of a simple adaptive relaying protocol for wireless relay networks. In *VTC-Spring*, 2008.

[21] K. Lin, N. Kushman, and D. Katabi. ZipTx: Harnessing partial packets in 802.11 networks. In *MobiCom*, 2008.

[22] S. Lin and D. J. Costello. *Error Control Coding*. Prentice-Hall, Inc., 2004.

[23] X. Liu, A. Sheth, M. Kaminsky, K. Papagiannaki, S. Seshan, and P. Steenkiste. DIRC: Increasing indoor wireless capacity using directional antennas. In *SIGCOMM*, 2009.

[24] M. Luby. LT codes. In *FOCS*, 2002.

[25] M. Ma and D. H. K. Tsang. Joint design of spectrum sharing and routing with channel heterogeneity in cognitive radio networks. *Physical Communication*, 2:127–137, 2009.

[26] M. O. Martínez-Rach, O. López, P. Pinol, M. P. Malumbres, J. Oliver, and C. T. Calafate. Quality assessment metrics vs. PSNR under packet loss scenarios in manet wireless networks. In *International Workshop on Mobile Video*, 2007.

[27] P. Maymounkov. Online codes. Technical report, New York University, 2002.

[28] M. Mitzenmacher. On the theory and practice of data recovery with multiple versions. In *International Symposium on Information Theory*, 2006.

[29] B. Raman and K. Chebrolu. Experiences in using WiFi for rural Internet in India. *IEEE Communications Magazine*, 45(1):104–110, January 2007.

[30] P. Roshan and J. Leary. *802.11 Wireless LAN Fundamentals*. Cisco Press, 2003.

[31] D. Salomon. *Data Compression: the Complete Reference*. Springer, 2007.

[32] Y. Shan, S. Yi, S. Kalyanaraman, and J. W. Woods. Two-stage FEC scheme for scalable video transmission over wireless networks. In *SPIE Multimedia Systems and Applications VIII*, 2005.

[33] A. Shokrollahi. Raptor codes. *IEEE Trans. on Information Theory*, 52(6):2551–2567, June 2006.

[34] A. Singh, A. Konrad, and A. D. Joseph. Performance evaluation of UDP Lite for cellular video. In *NOSSDAV*, 2001.

[35] M. Vutukuru, H. Balakrishnan, and K. Jamieson. Cross-layer wireless bit rate adaptation. In *SIGCOMM*, 2009.

[36] M. Vutukuru, K. Jamieson, and H. Balakrishnan. Harnessing exposed terminals in wireless networks. In *NSDI*, 2008.

[37] Y. Wang and Q. Zhu. Error control and concealment for video communication: A review. *Proceedings of the IEEE*, 86(5):974–997, May 1998.

[38] S. Wong, H. Yang, S. Lu, and V. Bharghavan. Robust rate adaptation for 802.11 wireless networks. In *MobiCom*, 2006.

[39] H. Yu. Secure and highly-available aggregation queries in large-scale sensor networks via set sampling. In *IPSN*, 2009.

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