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

Updated May 2011

Abstract

Motivated by recent emerging systems that can leverage partially correct packets in wireless networks, this paper proposes the novel concept of *error estimating coding* (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 design 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.

1 Introduction

Error correcting coding [29] has 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 [10, 11, 18, 25, 26, 28, 42, 47] can now use 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) [28].
- The destination may collect and combine multiple partial packets to obtain a correct copy [10, 18, 25].
- 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 [11, 47].
- 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 some useful information [26, 42].

These designs raise the natural question of whether there is any benefit looking beyond error correcting codes.

This paper takes the first step in answering this question, by proposing the novel concept of *error estimating coding* (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 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 coding.

Efficient EEC: Feasibility. This paper provides affirmative answers to both questions above, thus confirming the utility of the novel concept of EEC. First for feasibility, our EEC design only needs to add $O(\log n)$ extra bits to the packet (n being the number of data bits) to estimate BER. As concrete

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

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 design 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 implementation of EEC 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 [29]) 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 design provides formal and provable guarantee on the BER estimation accuracy. We do *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).

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, 8, 20, 22, 25, 27, 28, 30, 32, 48, 40]. 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, 16, 48] 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 [11, 47], 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) [40] or simply require retransmission to correct all partial packets.

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

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 [33] of the streamed video. PSNR difference that is above 0.5dB is considered visually noticeable [39].

In the next, Section 2 explains how EEC can benefit various applications. Our EEC design, implementation, and evaluation are presented in Section 3, 4 and 5. Section 6 and Section 7 detail the implementation and evaluation of the two applications. Finally, Section 8 discusses related work, and Section 9 draws the conclusions.

2 EEC Applications

Given that many designs [10, 11, 18, 25, 26, 28, 42, 47] 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 [28]. 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 recoverable (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, 48] or indirect information such as SNR [4, 19]. 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 6 will present our implementation and evaluation of this application.

The above discussion can also be generalized to using BER information for better selection of wireless channel [32], transmission power [22], or directional antenna orientation [30].

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 [8] 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 [37]. 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 [11, 47]. With forward error correction, the receiver (router) will now simply forward all packets (correct or partial) to the next hop [40], 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, instead of naively relaying the packets to the next hop. Section 7 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 [20]. 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* [25], 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 opposite property. Researchers thus suggest [27] 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. We will focus on the intuition here and leave the rigorous arguments to the proofs in the appendix. Our EEC algorithm is closely related to some prior algorithms in other domains — namely, algorithms [17, 24] for nearest neighbor search in high-dimensional spaces (mainly in computational geometry) and sketching algorithms [7, 6, 12, 13] for hamming distance estimation (mainly between massive data streams). We defer a detailed comparison to Section 8.

Table 1: Key notations in EEC algorithm.

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 $\mathbf{B}(x, y)$, with a closed form of $\phi(x, y) = \frac{1}{2}(1 - (1 - 2y)^x)$, see [14] and the appendix

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*, with each slot holding 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. We only aim to estimate p for $p \in [0, 1/4]$ since in practice, a packet with BER larger than $1/4$ rarely has any value. 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.

[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 [38] 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. This is intentional since the BER of the data bits is affected by how the coding algorithm places them into the packet. For the elegance of the model, we want to avoid reasoning about estimating a quantity that is itself affected by the algorithm. On the other hand, 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.

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.

1: **for** $i = 1$ to $\lfloor \log_2 n \rfloor$ **do**
2: **for** $j = 1$ to $j = s$ **do**
3: Select $2^i - 1$ data bits where each bit is chosen independently and uniformly randomly (with replacement) out of the n data bits;
4: Compute a parity bit (as an EEC bit) for them;
5: **end for**
6: **end for** /* Total $k = s \cdot \lfloor \log_2 n \rfloor = s \cdot l$ EEC bits. */
7: Place the EEC bits (in arbitrary order) into k uniformly random positions in the packet;
8: 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 ($1 \leq i \leq l$) 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 data bits, decoding is trivial and thus we omit the decoding pseudo-code.

Algorithm 2 EEC Estimating Procedure.

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

The estimating procedure (Algorithm 2) estimates the BER of the packet. For each level i ($1 \leq i \leq l$) 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. If the algorithm finds a q_i that falls within a range (c_1, c_2) where c_1 and c_2 are algorithm

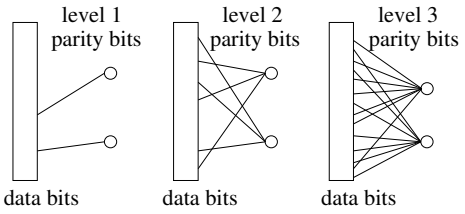


Figure 1: The first three levels of EEC bits ($s = 2$).

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 $q_1 \geq c_2$, it directly outputs $1/4$. 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.

3.3 Using One Bit to Sample a Group of Bits

To better explain the intuition in the algorithm, this section first assumes that all the EEC bits are in correct slots and we 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 then remove this key assumption in the second part of this section.

Naive sampling. A trivial way to estimate 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 inefficiency in this naive sampling approach arises however, when p_0 is small. This is particularly relevant in wireless 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 is 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 [9] 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.

Use a parity bit to sample a group of data bits. The above discussion already hints that it might help if we can sample multiple data bits together. For example, consider *groups* of data bits where each group has 50 randomly chosen data bits. Imagine that we use a single parity bit for each group as an EEC bit. Then even with a small number of EEC bits (i.e., a small number of groups), we will encounter sufficient number of parity check failures on the receiver side even for $p_0 = 0.02$. However, how to translate the observed parity check failures to an estimation on p_0 is not obvious. Since a parity bit can only tell whether the number of errors in the group is odd or even, it cannot even distinguish 3 errors from 1 error.

It turns out that the limited information provided by these parity bits over groups of data bits does suffice to estimate p_0. Such observation has also been previously made and leveraged in algorithms [7, 24] for nearest neighbor search and hamming distance estimation, though those algorithms have never been adopted/adapted to estimate packet BER in the context of coding (see more discussion in Section 8). The next describes how to properly use such parity information.

Properly using parity information. 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 $\mathbf{B}(g, p_0)$.

When p_0 is small enough, 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. 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. Similarly, if the parity check fails, it is very likely that the number of errors is exactly 1.

To leverage the above fact, however, one needs to first know whether p_0 is small enough. Let us define $\phi(g, p_0)$ to be the sum of all the odd terms in the binomial distribution $\mathbf{B}(g, p_0)$. Notice that $\phi(g, p_0)$ is essentially the probability of parity check failure, which can be estimated by the receiver if we use multiple independent groups (each with a parity bit) of the same size g.

If p_0 is not small enough, then $\phi(g, p_0)$ will exceed a certain threshold. This is easy to understand since for example, with $p_0 \geq 1/g$, 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. This indicates that the receiver can easily decide whether p_0 is small enough based on the (estimated) value of $\phi(g, p_0)$.

Allowing erroneous EEC bits. 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 slots within the packet. The n data bits will go into the remaining n slots 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.

Properly reasoning about such correlation is non-trivial. In particular, because the number of EEC bits can be small (compared to the data bits), the fraction of EEC bits that are in erroneous slots does not concentrate very well. In reasoning about the previous correlation, our analysis in the appendix derives a relatively weak concentration property, and shows that this weak property is sufficient to provide the desirable final guarantee.

Single-level EEC. Putting the above together gives what we call the *single-level EEC*. Single-level EEC uses s independent groups, where each group consists of g data bits and 1 EEC bit. The parameter g determines the range of p that the receiver can estimate. The receiver first determines the fraction q of parity failures among the s groups. This fraction q can be viewed as an estimation of $\phi(g+1,p)$. If $q \geq c_2$ where c_2 being an algorithm constant, then p is probably too large and the parity bits do not carry much information about p. Thus the algorithm simply stops without outputting anything (or simply outputs "BER too large").

If $q < c_2$, the algorithm will be able to estimate the total number of errors among all the $s \cdot (g+1)$ bits simply to be the total number of parity failures $s \cdot q$. For exactly the same reason as in naive sampling, the above estimation is good only when we see enough number of errors (i.e., when $s \cdot q$ is large enough). Thus the algorithm only outputs a final estimation of $\hat{p} = (s \cdot q)/(s \cdot (g+1)) = q/(g+1)$ if $q > c_1$ for some algorithm constant c_1 where $c_1 < c_2$. If $q \leq c_1$, the algorithm again stops without outputting anything (or simply outputs "BER too low").

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 $p \in (p_1, p_2)$.

3.4 Multiple Levels of EEC Bits

Multi-level EEC. Extending the single-level EEC algorithm to multi-level will enable us to 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 2 provides some numerical examples.

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 2: $\phi(2^i,p)$ for $1 \leq i \leq \lfloor \log_2 n \rfloor$, $n = 100$, and 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 ($1 \leq i \leq \lfloor \log_2 n \rfloor - 1$) such that $\phi(2^i,p) \leq c_1$. 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:

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

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 estimate the exact number of errors). Thus our EEC algorithm simply picks $c_1 = 0.25$ and $c_2 = 0.4$.

Reasoning about multiple levels. Reasoning about the formal guarantees of multi-level EEC is not as trivial as it appears. We have ensured that for any p, there exists some level i that can properly estimate p. But the algorithm does not know which level that is, and it needs to check level 1 through $i-1$ first. It is possible that under rare event, some level j ($j < i$) may appear suitable for estimating p even though it is actually not. While such probability for a given j is small, there can be many (i.e., $\Theta(\log n)$) such j values. A naive analysis may thus fail to produce our intended result using only $O(\log n)$ EEC bits. In our analysis in the appendix, we show that the sum of all these probabilities is actually well bounded. This is true even though the EEC bits may be corrupted themselves.

Flexible number of levels. Many applications (including the two applications that we implement) need 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 further reduce the redundancy of EEC. In real-time video streaming, the reduction enables us to use only one EEC level with 32 bits, making the EEC redundancy even comparable to typical CRC redundancy.

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 $c_1 < \phi(2^{l_1}, b) < c_2$ for integer l_1 will give us the value of l_1.[4] Similarly, we find l_2 such that $\phi(2^{l_2}, a) \in (c_1, c_2)$.[5] The algorithm only needs to keep all levels from level l_1 to level l_2 (both inclusive).

3.5 Optimizing Estimation Quality in Practice

We explained earlier that when q_i falls within the proper region, if the parity check fails, then 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 $q_i/2^i$ at Step 7 of Algorithm 2, one can reason more carefully about q_i's relation with p. It can be shown (see proofs in the appendix) that:

- The distribution of q_i concentrates near $\phi(2^i, p)$.
- The distribution of $\frac{q_i + 2q_{i-1}(1-q_{i-1})}{2}$ concentrates near $\phi(2^i, p)$.

Thus Step 7 can now directly solve the p from either of the following two equations as the final estimation:

$$\begin{aligned} \phi(2^i, \hat{p}) &= q_i \\ \phi(2^i, \hat{p}) &= \frac{q_i + 2q_{i-1}(1-q_{i-1})}{2}, \quad \text{for } i \geq l_1 + 1 \end{aligned}$$

Our formal asymptotic guarantees in the next section hold for both estimation approaches. In practice however, solving the second equation tends to give better quality, since it leverages the information from both level $i-1$ and level i. Thus our implementation solves the second equation to estimate p (whenever $i \geq l_1 + 1$) at Step 7 of Algorithm 2. The equation of $\phi(2^i, \hat{p}) = y$ has a closed-form solution (see appendix for proof) of $\hat{p} = (1 - (1-2y)^{2^{-i}})/2$.

Finally, in Algorithm 2, when failing to find an appropriate level to estimate BER at Step 10, the algorithm currently simply outputs 0. Instead of doing so, we will estimate the BER based on q_{l_2} (i.e., the information at the last level). This modification will slightly increase the estimation quality when BER is small. Such modification has no effect on our formal proofs.

3.6 Formal Guarantees

We will prove that in order to produce an (ϵ, δ) approximation for p with constant ϵ and δ, EEC only needs 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 $1/(n+k) \leq a < b \leq 1/4$) 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 $\hat{p} \leq (1+\epsilon)a$.*
- *If $p > b$, output $\hat{p}$ where $\hat{p} \geq (1-\epsilon)b$.*

The theorem's proof is available in the appendix. The next theorem is about EEC's computational overhead, whose proof is trivial:

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

4 EEC Implementation

This section describes how we apply standard optimizations to minimize EEC's computational overhead. We will only discuss optimizations for EEC's encoding procedure, since the exactly same optimizations apply to decoding and estimating. EEC's encoding procedure involves first calculating the $k = s \times l$ EEC bits, and then placing the k EEC bits and n data bits into the $n+k$ slots in a randomized way.

To make it concrete, our discussion below will be based on the Wi-Fi rate adaptation application, which uses 9 EEC levels (i.e., level 1 through level 9) with $s = 32$ bits per level. By default, we consider a packet containing 1536 bytes of data and $9 \times 32/8 = 36$ bytes of EEC bits. The EEC implementation in the real-time video streaming application is more straightforward as it only uses a single EEC level.

4.1 Calculating EEC Bits for Level 1 through Level 6

To calculate an EEC bit, the algorithm needs to select a certain number of data bits uniformly randomly and then compute their XOR. Selecting a uniformly random bit involves using a random index between 1 and n. Instead of generating this random index on the fly, we pre-compute all the random indices and store them in an array. Each array element will

[4] Note that it is possible for the equation to have two solutions, in which case the smaller solution should be used as l_1. It is not possible for the equation to have more than two solutions.

[5] Again, if the equation has two solutions, then the larger solution should be used as l_2.

be 2-byte long since $n = 1536 \times 8 = 12288$ in our case. An EEC bit at level i will need $2^i - 1$ indices.

For level 2 through level 6, we do not compute each EEC bit from scratch. Rather, to obtain the jth EEC bit in level i, we XOR the jth EEC bit in level $i-1$ and 2^{i-1} additional random data bits. This will reduce the computational overhead as well as the number of random indices needed roughly by half. Altogether, the total number of random indices needed for all levels from 1 to 6 will be $(1 + 2 + \ldots + 2^5)s = 2016$, which translates to roughly 4KBytes. Note that this optimization clearly introduces correlation across different levels. However, our formal proof does not require that different levels are independent, and thus the proof continues to hold under this optimization.

4.2 Calculating EEC Bits for Level 7 through Level 9

For level 7 through 9, we could also potentially apply the same optimizations as for level 1 through 6. But given the large group sizes on these levels, these optimizations are no longer effective enough simply because the number of data bits to be XORed is too large.

Calculating EEC bits in parallel. To reduce the computational overhead for computing the EEC bits for level 7 through 9, notice that there are total $3s$ EEC bits on level 7, 8, and 9. We can maintain a bit vector of length $3s$ for each data bit. The ith entry in the bit vector denotes whether the current data bit is selected for the ith EEC bit, for $1 \leq i \leq 3s$. Again, one can pre-compute all these bit vectors to reduce the computational overhead. To compute the $3s$ EEC bits, we can simply scan all the data bits sequentially. If a data bit is 1, we will XOR the corresponding bit vector to the result. On our target platform, Soekris net5501-70 with 32-bit Geode LX CPU, XORing $3s$-bit vectors can be done in three machine instructions (for $s = 32$).

Operating on bytes instead of bits. Our actual implementation further extends the above design. Instead of scanning the data bits one by one, we scan entire data bytes. We maintain a table which maps each possible byte value to a bit vector of $3s$ bits. The bit vector can be viewed as the pre-computed result for the 8 bits in the corresponding byte. Directly applying this optimization, however, will result in large tables and poor cache behavior. For example when $s = 32$, each data byte will require a table with $2^8 \times (3 \times 32/8) = 3072$ bytes. For a packet with 1536 data bytes, the 1536 tables will take about 4.7MBytes.

To reduce memory consumption, we observe that the 1536 tables serve to allow perfect randomness in the selection of data bits for each group. In practice, we may not need such perfect randomness (in fact, we cannot obtain perfect randomness anyway with pseudo-random number generators). Thus we divide the packet into multiple segments such that different segments use the same set of tables. Our implementation divides the packet into 12-byte long segments, and a packet with 1536 data bytes will have 128 segments. We only maintain 12 tables for the 12 bytes in a segment. As each table consumes 3072 bytes, the 12 tables only requires 36KBytes memory. These tables are computed when the EEC module is initialized.

Implications. There are two subtle implications worth discussion regarding our implementation. Firstly, as all segments use the same set of tables, each segment will contribute the same number of data bits to each group. This means that the total number of data bits selected in a group can only be a multiple of the number of segments (128 in our default packet size). As a result, we cannot have exactly $2^i - 1$ data bits for a group in level i (for $i \geq 7$), instead, our implementation selects 2^i data bits, increasing the group size by one. As the group size on level 7 through 9 is over 100, the difference is negligible in practice.

Secondly, since we reuse the same tables across multiple segments, the indices of data bits selected to form a group are no longer mutually independent. Our experiments show that such reduced randomness does not result in any noticeable impact. It is possible to improve the randomness by shifting the offset by some constant for each segment – but that does not seem to be needed from our results.

4.3 Placing EEC Bits and Data Bits into the Slots

The EEC algorithm requires that the EEC bits be placed into uniformly random slots (out of all slots). Our implementation achieves this by first placing all the n data bits into the first n slots within the packet (which has total $n + k$ slots). The last k slots are initially unoccupied. We then select k uniformly random slots out of the $n + k$ slots in the packet to place the EEC bits. These k slots are selected one by one and without replacement. Each selection requires a random index between 1 and $(n + k)$. For each slot selected, if it already contains a data bit, the data bit is removed from the slot and kept in a temporary buffer. (Notice that it is not possible for the slot to contain an EEC bit since we did the selection without replacement.) After all EEC bits have been placed into the packet, we place all data bits in the temporary buffer to all the unoccupied slots in the packet.

We again pre-generate k random indices to avoid creating them on the fly. For 288 EEC bits, the total memory consumption will be less than 1KByte.

4.4 Dealing with Variable Packet Sizes

Our optimizations so far all require maintaining certain data structures of non-trivial sizes. So far these data structures all depend on the packet size $n + k$. If the application uses variable packet sizes, it is clearly not possible to maintain one version of these data structures for each possible packet size. We next discuss how variable packet sizes can be supported. Without loss of generality, we assume that the number of data

bits in a packet is no larger than M bits, where M is a power of 2.

Calculating EEC bits for level 1 through 6. Here the only data structure we used is the array containing random indices for selecting data bits for each group. We will still maintain this array only for the maximum data size of M bits, which means that each array element is an integer between 0 and $M-1$.

If the packet contains $M/2^i$ data bits for some integer i, we can still use the random indices in the array, by setting the top i most significant bits of each index to be 0. This effectively gives a random index between 0 and $M/2^i-1$. If the number of data bits in the packet is not in the form of $M/2^i$, we must be able to find an i such that n falls between $M/2^{i+1}$ and $M/2^i$. We will simply use random indices between 0 and $M/2^i-1$, except that if the index exceeds the number of data bits in the packet, we discard that index and use the next one in the array. One can easily show that we will at most discard (i.e., waste) about half of the random indices in the array. Simply tripling the size of the array will be sufficient to compensate.

Placing EEC bits and data bits into the slots. Here the only data structure we used is the array containing random indices for placing the EEC bits into uniformly random slots. We will still maintain this array only for the maximum packet size, and then use the same technique as above to get random indices in different ranges.

Calculating EEC bits for level 7 through 9. The data structure that we use to facilitate the calculation of these EEC bits is a set of tables. Each table maps a byte value to a vector of $3s$ bits. Below we briefly sketch a possible way to deal with variable packet sizes while still using this optimization technique.

To achieve this, one can create different sets of tables for data size of M, $M/2$, $M/4$, and so on. If the data size n is not in the form of $M/2^i$, we must be able to find an i such that $n \in (M/2^i, M/2^{i-1})$. We will do EEC encoding on the last $M/2^i$ data bits only, and then place those data bits into uniformly randomly slots (out of the n slots that belong to the data bits). Notice that strictly speaking, the BER of those $M/2^i$ data bits can be different from the BER of the whole packet, and EEC's formal guarantee may no longer hold. But since the $M/2^i$ data bits constitute a majority out of the n bits, we expect the difference to be negligible.

Naively inserting the $M/2^i$ data bits into uniformly randomly slots will incur significant overhead. One can apply the same trick for placing EEC bits into uniformly random slots here. Specifically, we initially place the first $n-M/2^i$ data bits into the first $n-M/2^i$ slots in the packet. Then we insert the remaining $M/2^i$ data bits into uniformly random slots out of the first n slots, in a similar way of inserting EEC bits.

Finally, to further optimize for performance, instead of placing each of the $M/2^i$ data bits into a random slot, one could place each of the $(M/2^i)/8$ data bytes into a random byte position. While theoretically this means the positions of the data bits are now correlated, we do not expect the difference to be noticeable in practice.

5 EEC's Redundancy and Computational Overhead

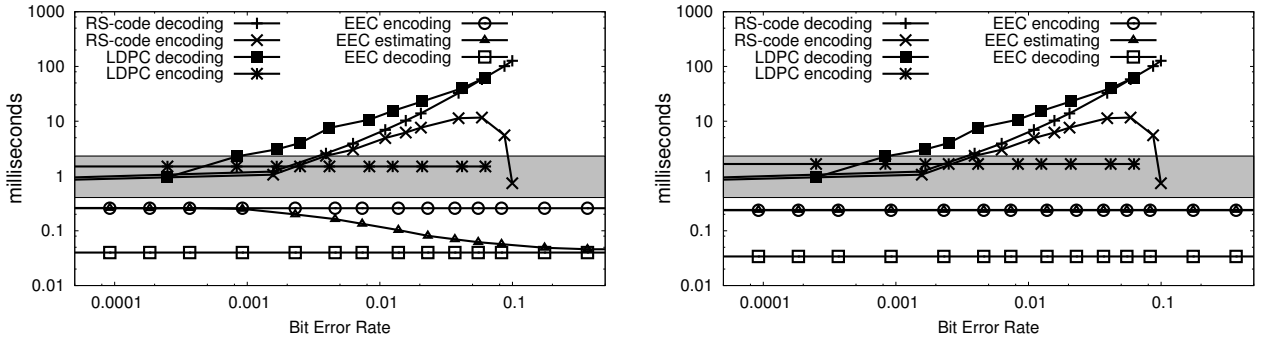
This section quantifies the redundancy and computational overhead of EEC in practical scenarios, and further compares against error correcting codes. Since error correcting codes provide stronger functionality than EEC, any comparison here will be an apple to orange comparison. Rather, our comparison 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 6 and 7 for why the two applications use these particular EEC parameters.

For error correcting codes, we use Reed-Solomon codes (RS codes) [29] and Low Density Parity Check codes (LDPC codes) [29] as examples. We use the RS codes software implementation from *DSP and FEC Library* v3.0.1[6], and use the LDPC codes software implementation from [43][7] with the default column weight of 3. There are other error correcting codes commonly used in wireless communications, such as convolutional codes and turbo codes. Convolutional codes have been shown to be computationally expensive for software implementation on general purpose CPU [44]. Specifically, software-based Viterbi decoding for convolutional codes at the data rate of 24Mbps requires 1.4G cycles/second computational power, even after careful optimization. A classic turbo code decoder, which uses convolutional code decoders as its components, tends to incur even higher computational overhead. As a result, most these error correcting codes need to be implemented in hardware. Since we need to deal with general flipping errors, erasure codes [3, 31, 34, 41] are not applicable.

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

[7] Note that 802.11n chipsets now implement LDPC in hardware. But the hardware is usually dedicated to coding/decoding needed by 802.11n, to meet the hard processing deadlines. The hardware usually cannot be time-shared by the upper layer to do LDPC encoding/decoding (e.g., as the end-to-end forward error correction used in our real-time video streaming application). In fact, typically the hardware does not even export such interface to the upper layer.



(a) Under parameters in Wi-Fi rate adaptation

(b) Under parameters in video streaming

Figure 3: 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 rates (6Mbps to 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. Similarly, under the LDPC code parameters used in these experiments (optimized for lower computational overhead), the LDPC codes cannot correct BER above 0.08.

5.1 Redundancy Overhead

EEC 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 32 bits per level. The relative redundancy added to a 1500-byte packet is thus $(9\times32)/(1500\times8) = 2.4\%$. 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 32 bits. This adds extra 4 bytes to each block, with a relative redundancy of around 2.0%.

RS codes redundancy. 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 [28]. 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.

LDPC codes redundancy. We use experiments to quantify the redundancy overhead of LDPC codes. Given a target packet BER to recover, using larger LDPC codewords will incur less redundancy overhead (though at the cost of larger computational overhead). To make our results pessimistic, we treat the entire packet as a single LDPC codeword. Our results show that LDPC codes can hardly recover a packet with BER as low as 1% even with 10% relative redundancy.

5.2 Computational Overhead

EEC computational overhead. Figure 3 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 3(b)), since we only use a single level EEC there. In the Wi-Fi rate adaptation setting (Figure 3(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 3 also plots the time available for processing a packet under the 802.11a/g

data rates. The results show that the computational overheads of our pure software implementation of EEC are small enough to support even the highest data rate.

RS codes computational overhead. Figure 3 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 $4\rho p$). 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 [28]. 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 3 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%.

LDPC codes computational overhead. Figure 3 also includes the computational overhead of LDPC codes, using our packet trace collected from an 802.11g link with intermediate link quality. To make our results pessimistic, we carefully choose the parameters of the LDPC code to minimize its overall computational overhead for processing a packet (even at the cost of increasing redundancy overhead). Specifically, we set LDPC codeword size to 648 bits, which is the smallest LDPC codeword size suggested by the 802.11n standard. We further set the relative redundancy to 90% — such a high relative redundancy allows LDPC codes to take fewer number of iterations to decode. This in turn can often reduce the overall decoding time for packets with BER above 1%. For packets with smaller BER (e.g., 0.01%), our experiments show that using 90% relative redundancy only increases the decoding time by less than 15% compared to using lower relative redundancy (such as $1/6$). Note that there are no parameters that can consistently minimize computational overhead of LDPC codes in all settings. Nevertheless, the parameters we choose ensure that our results are close to the optimal for packets with different BER values. As shown in Figure 3, even under such parameters, LDPC codes still tend to incur even larger computational overhead than RS codes.

6 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 [21], 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.[8] Thus we will directly use goodput as the measure of goodness. For a packet with BER p, we assume that the fraction of application-level 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 5). 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.

6.1 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) [48], and Receiver-Based AutoRate (RBAR) [16]. We do not consider SoftRate [45], which uses SoftPHY [18] 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

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

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, 16, 48] for similar reasons why SoftRate outperforms them [45]. 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, 48] or indirect SNR information [16].

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 [18], 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 [48], 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. First in normal cases, the receiver feeds back the BER if 4ms has passed and if it has received a partial packet since the last feedback. The 4ms interval is chosen such that the feedback overhead is below 5%. Second, the receiver also feeds back the BER when i) a partial packet is received at a new rate (in which case prompt positive feedback helps to retain the sender at this new rate), or ii) the feedback will trigger a rate change, or iii) a packet is received at an unexpected rate (implying a loss of synchronization between the sender and the receiver). Notice that since the receiver has the BER information, it can track the expected sending rate of the sender.

Implementation. We have implemented EEC-Rate in MadWifi 0.9.4[9]. We disable the default MAC-layer auto retransmission for failed packets. For comparison, we have also implemented RRAA [48] and an SNR-based scheme following the design of RBAR [16]. 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 [19] and uses the weighted moving average of SNR across multiple packets. Since we already have low overhead asynchronous feedback, we do not need to rely on the channel reciprocity assumption in CHARM. When evaluating the SNR-based scheme, we always carefully train the data-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 [46].

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.

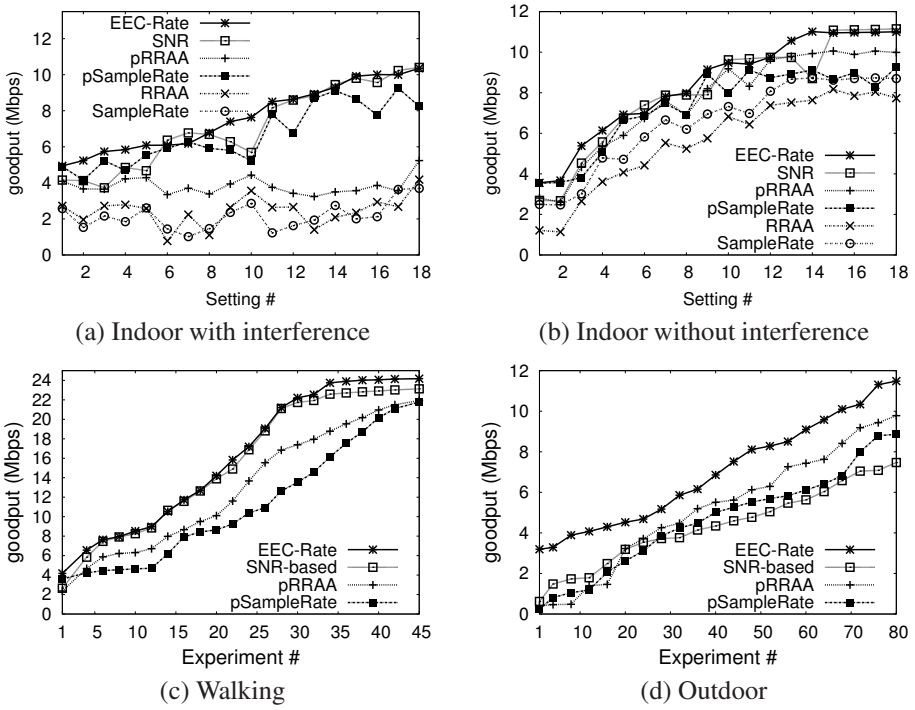
6.2 Evaluation Results

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 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 scenario. 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 4(a) presents the goodput when the links operate on

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



(a) Indoor with interference

(b) Indoor without interference

(c) Walking

(d) Outdoor

Figure 4: Goodput of different rate adaptation schemes under various environments.

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 [48] does not improve RRAA's or pRRAA's goodput in our setting. Similar conclusions have been made elsewhere [45] as well. The indoor interference-free results for 802.11a channel are shown in Figure 4(b). There all the schemes have similar relative performance as in Figure 4(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 4(c) 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 5 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 4(d) 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.

7 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* (denoted as `BER-aware`). We use `no-retran` to denote the existing approach [40] described in Section 2, where a partial packet is always directly forwarded to the next hop[10], 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.

Quantify the potential gain. We want to first gain some insight into whether `BER-aware` will likely lead to significant improvements. Figure 5 presents the complementary CDF for packet BER as measured on an 802.11a link with intermediate quality in our indoor mesh network testbed. Results from other researchers [28] 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.

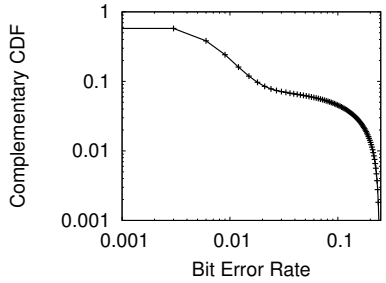


Figure 5: Complementary CDF for packet BER on an intermediate-quality link.

7.1 Implementation and Experimental Setting

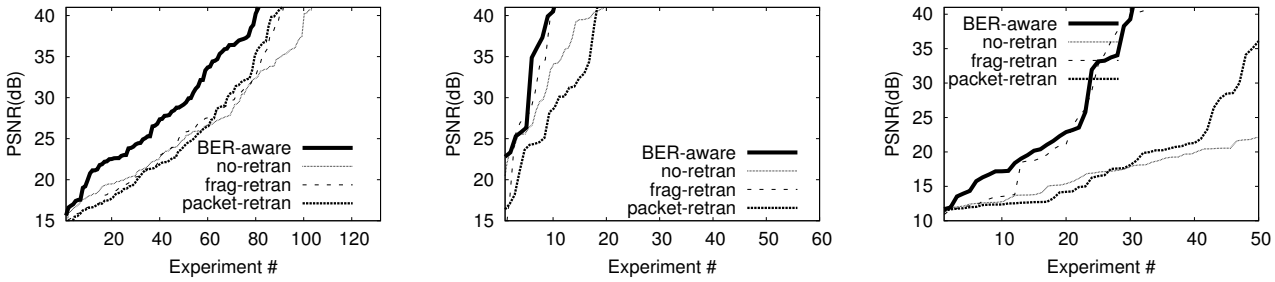
We implement the four schemes on Soekris net5501-70 routers as a Linux kernel module to interface with MadWifi 0.9.4. We configure all wireless interfaces in monitor mode, and implement a glue layer to expose the wireless interface as an Ethernet interface to the native Linux kernel network stack. We 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 [23] 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[11], 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 5. 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

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

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



(a) Link with PRR $\in [\frac{2}{4}, \frac{3}{4}]$ (b) Link with PRR $\in [\frac{3}{4}, 1]$ (c) Setting with a hidden terminal

Figure 6: Performance of different retransmission schemes under various link conditions.

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.

7.2 Evaluation Results

Measure of goodness. We use the standard Peak Signal-to-Noise Ratio (PSNR) [33] 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 [39]. Usually the video quality is considered "Excellent" if PSNR is above 37dB, and "Bad" if PSNR is below 20dB [23].

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)[12] 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 6(a) and (b) 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.[13] 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 6(a) 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 Figure 6(b), though all schemes in Figure 6(b) 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 6(c) presents the PSNR results under such setting. The results show `BER-aware` easily outperforms `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.

8 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 of wireless packets. The need for error estimation has been discussed in some

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

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

prior work in different contexts (e.g., [36], [35] and [45]). The general technique of inserting known bits (called *pilot bits*) into a packet can be viewed as a naive form of EEC, and Section 3.3 already explained why pilot bits are poorly suited for estimating BER. Fragmented CRC can be applied to determine the fraction of corrupted fragments in a packet [36]. This fraction can be a poor estimation for packet BER, especially under bursty errors. Error correcting codes can obviously also determine packet BER as a side benefit [36], though with substantially higher overhead than EEC.

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, EEC's specific way of forming groups and using the parity information is different from other coding algorithms. For example, LDPC codes [29] also compute parity bits over groups of data bits and are perhaps the most relevant codes to EEC. Each data bit in LDPC codes needs to belong to multiple groups to enable error correction. In comparison in EEC, data bits in one group may or may not be in other groups. This helps to keep redundancy overhead low, while still allowing error estimation if the parity information is used properly. The second difference is that the group sizes in LDPC codes are by definition small on average, since the codes are *low density*. In comparison in EEC, the group sizes can reach n and the average is as large as $\Theta(n/\log n)$. Similarly, EEC also differs from many other error correcting codes (such as Tornado code [3], LT codes [31], Raptor codes [41], Online codes [34], and Growth codes [20]) in terms of the group sizes and in terms of how the parity bits are actually used.

In our prior work [50] 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), despite adversarial interference. In comparison, our EEC algorithm relies on knowing whether the number of errors in a group is even or odd. A second key difference is that the algorithm in [50] is interactive and requires multiple rounds.

Finally, given a channel model such as binary symmetric channel or Gaussian noise channel, researchers have also studied how to estimate the average output BER of LDPC decoders [49]. This is largely not related to EEC, since the BER here is simply the average BER (after LDPC decoding) under the given channel model, instead of the BER of a given packet under unknown channel conditions.

SoftPHY. Coding is certainly not the only way of estimating BER. Recently, some researchers propose the SoftPHY [18] physical layer design, which exposes 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, 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 (especially for application scenarios where cost is not a major concern), EEC will continue to be useful on lower-end wireless devices such as wireless sensors. For these lower-end devices, 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. Also, even when the packet is entirely correct, SoftPHY can still provide quantitative confidence information while EEC can only output BER being zero. 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, 15, 19, 51] is to use Signal-Noise Ratio (SNR) in place of packet BER, since the theoretical relationship between SNR and BER is well-understood. SNR (either overall channel SNR [4, 19, 51] or SNR values of individual sub-carriers [15]) is fundamentally an indirect measure and needs to be (explicitly or implicitly) mapped to packet BER. The mapping is affected by various real-world factors such as hardware calibration, interference, and mobility. As a result, using SNR to estimate BER is not able to provide hard guarantees. EEC, on the other hand, provides mathematically provable estimation quality that is independent of the hardware or deployment environment. For example, EEC can provide the same provable guarantee even in non-WiFi settings. Furthermore, SNR-based approaches [15, 19, 51] often only measure SNR at the beginning of the packet, and the SNR may become different later in the packet. In comparison, EEC continues to provide its provable guarantee regardless of how the error are distributed within the packet.

Nearest neighbor and hamming distance estimation. After the initial conference version of this paper had been published, some existing nearest neighbor search algorithms and hamming distance sketching algorithms (which we had not been aware of) were brought to our attention. Nearest neighbor search in high dimensional spaces [17] is a key topic in computational geometry. The search asks for the nearest point (out of all candidate points in the database) to a given query point, according to some metric such as hamming distance. Many nearest neighbor search algorithms [17] (conceptually) first compute a *sketch* for each of the candidate point and the query point. A pair of sketches can provide use-

ful information on (sometimes allow a direct estimation of) the distance between the corresponding pair of points. Similar sketching techniques/algorithms [7, 6, 12, 13] have been further developed for hamming distance estimation, usually in the context of streaming databases. Here the algorithms compute a short sketch for each of the data streams, and then use the sketches to estimate the hamming distance between data streams.

In hindsight for the EEC problem, it is clear that since we do not make assumptions on error independence/correlation, the packet received can be arbitrarily different from the packet sent, and thus we might just as well view them as two *unrelated* packets. Further if we restrict to considering only "systematic codes" (i.e., the data bits are not changed during encoding), then packet BER directly corresponds to the hamming distance between these two unrelated packets. In particular, the EEC bits in our algorithm can be viewed as a sketch, and the BER is estimated based on the (potentially corrupted) sketch of the packet sent and the sketch of the packet received.

Despite the above connection, to the best of our knowledge, those sketching algorithms [7, 6, 12, 13, 17, 24] have never been adopted/adapted to estimate packet BER in the context of coding. Furthermore, the EEC problem imposes an additional requirement that these previous algorithms do not need to address in their corresponding contexts. Namely, the EEC bits may be corrupted during transmission as well. It turns out that reasoning about potentially corrupted EEC bits is a key technical difficulty in our analysis and proof in this work. None of those sketching algorithms (need to) consider potentially corrupted sketches, and there has been no existing analysis on their robustness in our context. One could potentially use error correcting coding to protect the sketches, which unfortunately would add both complexity and extra overhead.

Relationship to specific sketching algorithms. The above has discussed the overall difference between our work and existing sketching algorithms. Since our EEC algorithm does use some rather similar techniques to some of the sketching algorithms, the following provides a comparison with several specific sketching algorithms.

Among the various existing nearest neighbor search algorithms, Kushilevitz et al.'s algorithm [24] for the hamming distance metric is perhaps the most related. Here the points considered are n-bit binary vectors. Their algorithm conceptually computes n levels of parity bits as the sketch, for all the points in the database collectively. The parity bits in the ith level each correspond to a group of data bits with the expected group size being $n/(4i)$, and serve (conceptually) to test whether the hamming distance exceeds i. Notice that our single-level EEC algorithm is very similar to their algorithm, except that we further reason about and prove the effects of corrupted EEC bits. On the other hand, given their target application, their algorithm does not optimize for the total size of the sketch (e.g., the total number of levels). Thus directly using their algorithm for multi-level EEC purpose is not feasible.

The large sketch size in Kushilevitz et al.'s algorithm is not fundamental. Later, Cormode et al. [7] adapt that algorithm explicitly for hamming distance estimation between two documents. Cormode et al.'s algorithm uses total $\log_\beta n$ parity bits as the sketch for each document. The $\log_\beta n$ parity bits are conceptually from $\log_\beta n$ levels, with 1 bit per level. The ith level parity bit corresponds to a group of β^i data bits (i.e., geometrically distributed group sizes as in our algorithm). To increase estimation quality, the algorithm will use more levels (i.e., with a β closer to 1). In comparison, our EEC algorithm fixes the number of levels (i.e., $\log_2 n$) and uses more bits on each level for better estimation quality. Cormode et al. [7] only show that their algorithm works for ϵ above some threshold, while we prove that our algorithm works for all $\epsilon > 0$. But as hinted in [12], we suspect that a better analysis (e.g., as ours) should be able to remove this limitation of Cormode et al.'s algorithm.

There are also many other sketching algorithms (mainly in the context of streaming databases) [6, 13] for hamming distance estimation. These algorithms are less similar to our EEC algorithm, and use more complex quantities (instead of parity bits) as sketches. It is worth noting that [13] requires a sketch size of only $O(\log n)$ bits to achieve an (ϵ, δ) approximation for any constant ϵ and δ. Such sketch size is asymptotically the same as the redundancy added by our EEC algorithm.

9 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 coding (EEC). Without correcting the errors, EEC enables the receiver of a partial packet to estimate the packet's BER. Our EEC design 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 [35]. Generally speaking, EEC may find potential application wherever partially correct data can be utilized.

Acknowledgments

We thank Piotr Indyk for drawing our attention to prior research efforts on nearest neighbor search in high-dimensional

spaces and sketching algorithms for hamming distance estimation between massive data streams. We thank Mun Choon Chan, Piotr Indyk, Ben Leong, Ramachandran Ramjee, IEEE/ACM Transactions on Networking anonymous reviewers, and SIGCOMM'10 anonymous reviewers for many helpful comments on this paper. We thank Kyle Jamieson and Brad Karp for helpful discussions. We thank Yajun Ha and Samarjit Chakraborty for answering our hardware-related questions.

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] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). *IEEE Transactions on Knowledge and Data Engineering*, March 2003.

[7] G. Cormode, M. Paterson, S. Sahinalp, and U. Vishkin. Communication complexity of document exchange. In *SODA*, 2000.

[8] 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.

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

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

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

[12] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. Wright. Secure multiparty computation of approximations. In *ICALP*, 2001.

[13] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L1-difference algorithm for massive data streams. *SIAM Journal on Computing*, 32(1), 2002.

[14] R. Gallager. *Low Density Parity Check Codes*. MIT Press, 1963.

[15] D. Halperin, W. Hu, A. Shethy, and D. Wetherall. Predictable 802.11 packet delivery from wireless channel measurements. In *SIGCOMM*, New Delhi, India, August 2010.

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

[17] P. Indyk. Nearest neighbors in high-dimensional spaces. *Handbook of Discrete and Computational Geometry*, 2003.

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

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

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

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

[22] 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.

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

[24] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30(2), 2000.

[25] 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.

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

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

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

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

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

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

[32] 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.

[33] 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.

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

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

[36] E. Newcombe and S. Pasupathy. Error rate monitoring for digital communications. *Proceedings of the IEEE*, 70(8):805–828, August 1982.

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

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

[39] D. Salomon. *Data Compression: The Complete Reference*. Springer, 2007.

[40] 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.

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

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

[43] S. Takamura. LDPC code implementation. `http://ivms.stanford.edu/~varodayan/multilevel/index.html`.

[44] K. Tan, J. Zhang, J. Fang, H. Liu, Y. Ye, S. Wang, Y. Zhang, H. Wu, W. Wang, and G. M. Voelker. Sora: High performance software radio using general purpose multi-core processors. In *NSDI*, 2009.

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

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

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

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

[49] H. Xiao and A. Banihashemi. Error rate estimation of low-density parity-check codes on binary symmetric channels using cycle enumeration. *IEEE Trans. Comm.*, 57(6):1550–1555, June 2009.

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

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

Table 2: Key notations used in our proof. Extended from Table 1.

n	number of data bits in a packet
k	number of EEC bits in a packet (i.e., $s \times l$)
s	number of EEC bits in one level
l	number of EEC levels (i.e., $l_2 - l_1 + 1$)
g	number of data bits in one group
p	fraction of erroneous slots in the given packet (between 0 and $1/4$)
$\hat{p}$	estimated value of p as returned by the algorithm
a	algorithm parameter
b	algorithm parameter (no larger than $1/4$)
q_i	the fraction of parity check failures at level i (which is a random variable)
$\phi(x, y)$	sum of odd terms in the binomial distribution $\mathbf{B}(x, y)$
$\psi(x)$	$\psi(x) = 2x(1 - x)$
c_1	constant of 0.25
c_2	constant of 0.4
d_0	constant of 0.26 (i.e., slightly larger than c_1)
d_1	constant of $\psi(d_0)$ (roughly 0.385)
d_2	constant of $\psi^{(2)}(d_0)$ (roughly 0.474)
d_{-1}	constant of $\psi^{(-1)}(d_0)$ (roughly 0.154)
d_{-2}	constant of $\psi^{(-2)}(d_0)$ (roughly 0.084)
d_{-3}	constant of $\psi^{(-3)}(d_0)$ (roughly 0.044)
d_{-4}	constant of $\psi^{(-4)}(d_0)$ (roughly 0.022)
log	base 2, i.e $\log = \log_2$
l_1, l_2	selected levels are from level l_1 to l_2 (both inclusive)
P_1	fraction of erroneous data bits among all data bits (which is a random variable since placement is randomized)
P_2	fraction of erroneous EEC bits among all EEC bits (which is a random variable since placement is randomized)

A Formal Proof for Theorem 1

This appendix provides a formal proof for Theorem 1. Note that we have not optimized for the constants used in our proof. It is possible that the same results can be proved with better constants, though that will not affect our final asymptotic claims. Table 2 summarizes the notations used in our proof.

A.1 Technical Lemmas

This section first proves some useful technical lemmas regarding the two functions $\phi(x, y)$ and $\psi(x)$. For positive integer x and real number $y \in [0, 1]$, $\phi(x, y)$ is defined to be the sum of all the odd terms in a binomial distribution $\mathbf{B}(x, y)$, or formally:

$$\phi(x, y) = \sum_{1 \le j \le x \text{ and } j \text{ is odd}} \binom{x}{j} y^j (1 - y)^{x-j}$$

Lemma 3 *Consider any positive integer x and any real number $y \in [0, \frac{1}{2}]$. We have:*

1. $\phi(x, y) = \frac{1}{2}(1 - (1 - 2y)^x)$. *(Note that this equation is actually well-known from the study on LDPC codes [14], but we still include a proof here for completeness.)*
2. $\phi(x, y) \in [y, \frac{1}{2}]$.
3. $\phi(x, y) \le xy$.
4. *Given y, $\phi(x, y)$ is monotonically increasing with x.*
5. *Given x, $\phi(x, y)$ is monotonically increasing with y.*
6. *For any $c \ge 1$ and $cy \le \frac{1}{2}$, $\phi(x, cy) \le c\phi(x, y)$.*

7. *For any $c \in [0,1]$, $\phi(x, cy) \geq c\phi(x, y)$.*

8. *Let $\phi^{(-1)}(x, z)$ denote the inverse function of $\phi(x, y)$ with respect to y. Then $y = \phi^{(-1)}(x, z) = \frac{1}{2}(1 - (1-2z)^{\frac{1}{x}})$.*

9. *Given x, $\phi^{(-1)}(x, z)$ is monotonically increasing with z (within its domain of $[0, \frac{1}{2}]$).*

Proof:

1. Use induction on x. Obviously $\phi(1, y) = y$ and the equation holds for $x = 1$. For $x \geq 2$, we have:

$$\begin{aligned}\phi(x,y) &= (1-y)\phi(x-1,y) + y(1-\phi(x-1,y)) \\ &= \frac{1}{2}(1-y)(1-(1-2y)^{x-1}) + \\ &\quad y(1-\frac{1}{2}(1-(1-2y)^{x-1})) \\ &= \frac{1}{2}(1-(1-2y)^{x})\end{aligned}$$

2. Use induction on x. Obviously, $\phi(1, y) = y \in [y, \frac{1}{2}]$. For $x \geq 2$, we have:

$$\begin{aligned}\phi(x,y) &= (1-y)\phi(x-1,y) + y(1-\phi(x-1,y)) \\ &= y + (1-2y)\phi(x-1,y) \geq y\end{aligned}$$

   Furthermore:

$$\begin{aligned}& y + (1-2y)\phi(x-1,y) \\ =\ & \phi(x-1,y) + y(1-2\phi(x-1,y)) \\ \leq\ & \phi(x-1,y) + \phi(x-1,y)(1-2\phi(x-1,y)) \\ =\ & 2\phi(x-1,y)(1-\phi(x-1,y)) \leq \frac{1}{2}\end{aligned}$$

3. Notice that $1 - \phi(x, y)$ is the sum of all the even terms in the binomial distribution. This sum is clearly larger than or equal to the term of $\binom{x}{0}(1-y)^x \geq 1 - xy$. Thus we have $1 - \phi(x, y) \geq 1 - xy$ and $\phi(x, y) \leq xy$.

4. For all $x \geq 2$, we have $\phi(x, y) = \phi(x-1, y) + y(1 - 2\phi(x-1, y)) \geq \phi(x-1, y)$.

5. It suffices to show that $\frac{\partial \phi}{\partial y}(x, y) \geq 0$. Consider a given x, we have

$$\begin{aligned}\frac{\partial \phi}{\partial y}(x,y) &= -\frac{1}{2}x(1-2y)^{x-1}(-2) \\ &= x(1-2y)^{x-1} \geq 0\end{aligned}$$

6. Use induction on x. For $x = 1$, we have $\phi(1, cy) = cy = c\phi(1, y)$. For $x \geq 2$, we have:

$$\begin{aligned}\frac{\phi(x,cy)}{\phi(x,y)} &= \frac{cy + (1-2cy)\phi(x-1,cy)}{y + (1-2y)\phi(x-1,y)} \\ &\leq \frac{cy + (1-2y)\phi(x-1,cy)}{y + (1-2y)\phi(x-1,y)} \\ &\leq \frac{cy + c(1-2y)\phi(x-1,y)}{y + (1-2y)\phi(x-1,y)} = c\end{aligned}$$

7. Use induction on x. For $x = 1$, we have $\phi(1, cy) = cy = c\phi(1, y)$. For $x \geq 2$, we have:

$$\begin{aligned}\frac{\phi(x,cy)}{\phi(x,y)} &= \frac{cy + (1-2cy)\phi(x-1,cy)}{y + (1-2y)\phi(x-1,y)} \\ &\geq \frac{cy + (1-2y)\phi(x-1,cy)}{y + (1-2y)\phi(x-1,y)} \\ &\geq \frac{cy + c(1-2y)\phi(x-1,y)}{y + (1-2y)\phi(x-1,y)} = c\end{aligned}$$

8. Since $z = \phi(x, y)$, we have $1 - 2z = (1 - 2y)^x$. For a given $z \in (0, \frac{1}{2}]$, there is only root y satisfying the equation in the domain of $(0, \frac{1}{2}]$. This root is exactly $y = \frac{1}{2}(1 - (1 - 2z)^{\frac{1}{x}})$.

9. It suffices to show that $\frac{\partial \phi^{(-1)}}{\partial z}(x, z) \geq 0$ for all $z \in (0, \frac{1}{2}]$. Consider a given x, we have

$$\begin{aligned} \frac{\partial \phi^{(-1)}}{\partial z}(x, z) &= -\frac{1}{2}\frac{1}{x}(1 - 2z)^{\frac{1}{x}-1}(-2) \\ &= \frac{1}{x}(1 - 2z)^{\frac{1}{x}-1} \geq 0 \end{aligned}$$

□

For convenience later, we define function $\psi : [0, 1] \to [0, \frac{1}{2}]$, where $\psi(x) = 2x(1-x)$. It is easy to verify that ψ is bijective and monotonically increasing in $[0, \frac{1}{2}]$. Thus, we define function $\psi^{(-1)} : [0, \frac{1}{2}] \to [0, \frac{1}{2}]$, where $\psi^{(-1)}(x) = \frac{1}{2}(1-\sqrt{1-2x})$. It is easy to verify that $\psi^{(-1)}$ is the inverse function of ψ in $[0, \frac{1}{2}]$. Define universal constant $d_0 = 0.26 > c_1$. Define $d_1 = \psi(d_0)$ (roughly 0.385) and $d_2 = \psi(d_1)$ (roughly 0.474). Define d_{-1} (roughly 0.154), d_{-2} (roughly 0.084), d_{-3} (roughly 0.044), and d_{-4} (roughly 0.022) to be constants such that $\psi(d_{-1}) = d_0$, $\psi(d_{-2}) = d_{-1}$, $\psi(d_{-3}) = d_{-2}$, and $\psi(d_{-4}) = d_{-3}$.

Lemma 4 *For any positive integer i and real number $y \in [0, 1]$, we have $\phi(2^i, y) = \psi^{(i)}(y)$ where "$\psi^{(i)}$" means applying ψ for i times.*

Proof: Use induction on i. For $i = 1$, we have $\phi(2, y) = 2y(1 - y) = \psi(y)$. For $i \geq 2$, we have:

$$\begin{aligned} \phi(2^i, y) &= 2\phi(2^{i-1}, y)(1 - \phi(2^{i-1}, y)) \\ &= 2\psi^{(i-1)}(y)(1 - \psi^{(i-1)}(y)) \\ &= \psi(\psi^{(i-1)}(y)) = \psi^{(i)}(y) \end{aligned}$$

□

Lemma 5 *For any $y \in (0, d_0]$:*

- *There is exactly one positive integer h such that $\phi(2^h, y) \in (d_0, d_1]$. Furthermore, $h \leq \lfloor \log \frac{1}{y} \rfloor$.*
- *For the above h, any y' where $0.6y < y' < 1.4y$, and any i where $h - 2 \leq i \leq h$, we have $\phi(2^i, y') \in (d_{-4}, d_2)$.*
- *For the above h, any y' where $0.6y < y' < 1.4y$, and any i where $1 \leq i \leq h - 2$, we have $\phi(2^i, y') < 1.4d_{-1}$.*

Proof:

- We first prove that $\phi(2, y) \leq d_1$ and $\phi(2^{\lfloor \log \frac{1}{y} \rfloor}, y) > d_0$. Since $y \leq d_0$, we have $\phi(2, y) \leq \phi(2, d_0) = d_1$. To prove the second inequality, let $w = 2^{\lfloor \log \frac{1}{y} \rfloor - 1} \geq \max(\frac{1}{4y}, 1)$. Furthermore, $w \leq 2^{\log \frac{1}{y} - 1} = \frac{1}{2y}$. We have:

$$\begin{aligned} \phi(w, y) &= \sum_{1 \leq j \leq w, j \text{ is odd}} \binom{w}{j} y^j (1 - y)^{w-j} \\ &\geq wy(1 - y)^{w-1} \geq wy(1 - wy + y) \\ &> wy(1 - wy) \geq \frac{1}{4}(1 - \frac{1}{4}) = \frac{3}{16} \\ \phi(2^{\lfloor \log \frac{1}{y} \rfloor}, y) &= \psi(\phi(w, y)) > \psi(\frac{3}{16}) > d_0 \end{aligned}$$

  Let $h \leq \lfloor \log \frac{1}{y} \rfloor$ be the smallest integer such that $\phi(2^h, y) > d_0$. Such h is guaranteed to exist. We want to show that $\phi(2^h, y) \leq d_1$. If $h = 1$, we already proved that $\phi(2, y) \leq d_1$. If $h > 1$, then by definition of h, we have $\phi(2^{h-1}, y) \leq d_0$ and:

$$\phi(2^h, y) = \psi(\phi(2^{h-1}, y)) \leq \psi(d_0) = d_1$$

  We have shown the existence of such h, and we still need to prove that $\phi(2^{h'}, y) \in (d_0, d_1]$ only for $h' = h$. Since h is the smallest integer such that $\phi(2^h, y) > d_0$, we only need to consider $h' > h$. For any $h' \geq h + 1$, we have:

$$\phi(2^{h'}, y) \geq \phi(2^{h+1}, y) = \psi(\phi(2^h, y)) > \psi(d_0) = d_1$$

- It is easy to show that $\psi^{(-2)}(y) < 0.6y < y' < 1.4y < \psi(y)$. We have:

$$\begin{aligned}
\phi(2^{h-2}, y') &< \phi(2^{h-1}, y') < \phi(2^h, y') < \phi(2^h, \psi(y)) \\
&= \psi^{(h+1)}(y) = \psi(\phi(2^h, y)) \\
&\leq \psi(d_1) = d_2 \\
\phi(2^h, y') &> \phi(2^{h-1}, y') > \phi(2^{h-2}, y') \\
&> \phi(2^{h-2}, \psi^{(-2)}(y)) = \psi^{(h-4)}(y) \\
&= \psi^{(-4)}(\phi(2^h, y)) > \psi^{(-4)}(d_0) = d_{-4}
\end{aligned}$$

- Lemma 3 and 4 tell us that:

$$\begin{aligned}
\phi(2^i, y') &\leq \phi(2^{h-2}, y') < \phi(2^{h-2}, 1.4y) \\
&\leq 1.4\phi(2^{h-2}, y) = 1.4\psi^{(-2)}(\phi(2^h, y)) \\
&\leq 1.4\psi^{(-2)}(d_1) = 1.4d_{-1}
\end{aligned}$$

□

Lemma 6 *For any integer $i \geq 2$ and real number $x, y, z, \epsilon \in [0, 0.5]$ where*

$$\begin{aligned}
|\phi^{(-1)}(2^i, x) - z| &\leq \epsilon z \\
|\phi^{(-1)}(2^{i-1}, y) - z| &\leq \epsilon z,
\end{aligned}$$

we have

$$\left|\phi^{(-1)}\left(2^i, \frac{x + \psi(y)}{2}\right) - z\right| \leq \epsilon z$$

Proof: Since $\phi^{(-1)}(2^{i-1}, y) = \phi^{(-1)}(2^i, \psi(y))$, we have:

$$|\phi^{(-1)}(2^i, \psi(y)) - z| \leq \epsilon z$$

Without loss of generality, assume $x \leq \psi(y)$. We have:

$$\begin{aligned}
& x \leq \frac{x + \psi(y)}{2} \leq \psi(y) \\
\Rightarrow\ & \phi^{(-1)}(2^i, x) \leq \phi^{(-1)}\left(2^i, \frac{x + \psi(y)}{2}\right) \leq \phi^{(-1)}(2^i, \psi(y)) \\
\Rightarrow\ & \left|\phi^{(-1)}\left(2^i, \frac{x + \psi(y)}{2}\right) - z\right| \leq \epsilon z
\end{aligned}$$

□

A.2 Concentration Properties of P_1 and P_2

The EEC algorithm has two randomization components. The first component is the randomized placement of the EEC bits into the slots of a packet. The second component is choosing random data bits to form the groups. We will first focus on the effects of the first randomization component. Let random variable P_1 be the fraction of erroneous data bits among all data bits, after the randomized placement. Similarly, let P_2 be the fraction of erroneous EEC bits among all EEC bits. This section shows that P_1 and P_2 concentrate near p under proper n and s. Notice that for sufficiently large n and $s = O(1)$, P_1 will concentrate better than P_2.

Lemma 7 *For any given $\epsilon, \delta > 0$, there exists constant s' such that $\forall s \geq s'$ and $\forall n \geq s'$:*

$$\Pr[|P_1 - p| > \epsilon] < \delta \quad \textit{and} \quad \Pr[|P_1 - p| > \frac{1}{\log \frac{1}{p}}] < \delta$$

$$\Pr[|P_2 - p| > \epsilon] < \delta \quad \textit{and} \quad \Pr[|P_2 - p| > \frac{1}{\log \frac{1}{p}}] < \delta$$

Proof: Recall the concept of *hypergeometric distribution*, which describes the number of hits in a finite sequence of draws from a finite set *without* replacement. (Binomial distribution is the version with replacement.) The distribution of kP_2 is exactly a hypergeometric distribution with parameters $(k+n,(k+n)p,k)$. Thus we have:

$$\mathrm{VAR}[kP_2] = \frac{p(1-p)nk}{(k+n-1)} \text{ and } \mathrm{VAR}[P_2] = \frac{p(1-p)n}{k(k+n-1)}$$

Apply Chebyshev's inequality and we have:

$$\begin{aligned}
\Pr[|P_2 - p| > \epsilon] &< \frac{p(1-p)n}{k(k+n-1)}\frac{1}{\epsilon^2} < \frac{1}{s\epsilon^2} \\
\Pr[|P_2 - p| > \frac{1}{\log\frac{1}{p}}] &< \frac{p(1-p)n}{k(k+n-1)}\log^2\frac{1}{p} \\
&< \frac{p\log^2\frac{1}{p}}{s} < \frac{2}{s}
\end{aligned}$$

The last step holds because $p\log^2\frac{1}{p} < 2$ for all $p \in [0,1]$. Let $s' = max(\frac{1}{\delta\epsilon^2}, \frac{2}{\delta})$ completes the proof for P_2. Finally, similar arguments on P_1 hold for $\forall n \geq s'$. □

Lemma 8 *For any given $\epsilon, \delta, s > 0$, there exists constant n' such that $\forall n \geq n'$:*

$$\Pr[|P_1 - p| > \epsilon p] < \delta$$

Proof: Here nP_1 follows a hypergeometric distribution with parameters $(k+n,(k+n)p,n)$. Notice that $p > 0$ implies $p \geq \frac{1}{k+n}$. We have:

$$\mathrm{VAR}[P_1] = \frac{p(1-p)k}{n(k+n-1)} \leq \frac{kp}{n(k+n)}$$

Apply Chebyshev's inequality and we have:

$$\Pr[|P_1 - p| > \epsilon p] < \frac{k}{n(k+n)p\epsilon^2} \leq \frac{k}{n\epsilon^2} \leq \frac{s\log n}{n\epsilon^2}$$

When n is sufficiently large, the above quantity is clearly smaller than any given δ. □

Overarching conditions. Our proof so far has shown that for any $\epsilon_1 \in (0, 0.5], \epsilon_2 \in (0, 0.4]$, under proper s and n, with probability at least $1-\delta$, we have $P_1 = p_1$ and $P_2 = p_2$ where p_1 and p_2 satisfy the following 5 *overarching conditions*:

$$\begin{aligned}
|p_1 - p| \leq \frac{d_{-4}}{200}\epsilon_1 \quad &\text{and} \quad |p_2 - p| \leq \frac{d_{-4}}{200}\epsilon_1 \\
|p_1 - p| \leq \frac{1}{\log\frac{1}{p}} \quad &\text{and} \quad |p_2 - p| \leq \frac{1}{\log\frac{1}{p}} \\
|p_1 - p| \leq \epsilon_2 p &
\end{aligned}$$

The above fact enables us to analyze the two randomization components in the EEC algorithm in a clean way. Specifically, Section C through E below will analyze the effects of second randomization component (i.e., choosing random data bits for the groups), conditioned upon the above 5 overarching conditions being met. In other words, all the probabilities in those sections will be only with respect to the random coin flips for choosing random data bits for the groups.

A.3 Concentration Properties of q_i

Define random variable q_i to be the fraction of parity check failures at the ith level. The value of q_i can be observed by the receiver of the packet. This section reasons about q_i's concentration property. We first show that q_i's variance can be made arbitrarily small by increasing s, implying that q_i concentrates near its expectation $\mathbf{E}[q_i]$. Next we show that $\mathbf{E}[q_i]$ is close to $\phi(2^i, p_1)$. Combining these results eventually leads to a proof showing that q_i concentrates near $\phi(2^i, p_1)$. Such concentration property of q_i will enable later reasoning regarding on which level the EEC algorithm will stop, as well as reasoning regarding the estimation quality from that level (if we do not use the optimization from Section 3.5 of using two adjacent levels to estimate p).

Lemma 9 *For any positive integer i, we have:*

$$\begin{aligned}\mathbf{E}[q_i] &= p_2(1-\phi(2^i-1,p_1))+(1-p_2)\phi(2^i-1,p_1)\\ \mathrm{VAR}[q_i] &\le \mathbf{E}[q_i]/s\end{aligned}$$

Proof: Let X_j ($1 \le j \le s$) denote the event of parity check failure for the jth parity bit at the given level i. We have $q_i = \frac{1}{s}\sum_{j=1}^{s} X_j$, and:

$$\begin{aligned}\mathbf{E}[q_i] &= \frac{1}{s}\sum_{j=1}^{s}\mathbf{E}[X_j] = \mathbf{E}[X_1]\\ &= p_2(1-\phi(2^i-1,p_1))+(1-p_2)\phi(2^i-1,p_1)\end{aligned}$$

Next to reason about the variance of q_i, we first show that $\mathrm{Cov}(X_j, X_{j'}) \le 0$ for any $j \ne j'$. For clarity, in the following, $\phi(2^i-1,p_1)$ is abbreviated as ϕ. We have:

$$\begin{aligned}\mathbf{E}[X_jX_{j'}] &= p_2(\frac{kp_2-1}{k-1})(1-\phi)^2 +\\ &\quad 2p_2(1-\frac{kp_2-1}{k-1})\phi(1-\phi)+\\ &\quad (1-p_2)(1-\frac{kp_2}{k-1})\phi^2\\ \mathbf{E}^2[X_j]=\mathbf{E}^2[X_{j'}] &= (p_2(1-\phi)+(1-p_2)\phi)^2\\ &= p_2^2(1-\phi)^2+2p_2(1-p_2)\phi(1-\phi)+\\ &\quad (1-p_2)^2\phi^2\\ \mathrm{Cov}(X_j,X_{j'}) &= \mathbf{E}[X_jX_{j'}]-\mathbf{E}^2[X_j]\\ &= -\frac{p_2(1-p_2)}{k-1}(1-\phi)^2+\\ &\quad \frac{2p_2(1-p_2)}{k-1}\phi(1-\phi)-\frac{p_2(1-p_2)}{k-1}\phi^2\\ &= -\frac{p_2(1-p_2)}{k-1}(1-2\phi)^2 \le 0\end{aligned}$$

We can now calculate the variance of q_i:

$$\begin{aligned}\mathrm{VAR}[q_i] &= \mathrm{VAR}[\frac{1}{s}\sum_{j=1}^{s}X_j]\\ &= \frac{1}{s^2}(\sum_{j=1}^{s}\mathrm{VAR}[X_j]+\sum_{j=1}^{s}\sum_{j'=1,j'\ne j}^{s}\mathrm{Cov}(X_j,X_{j'}))\\ &\le \frac{1}{s}\mathrm{VAR}[X_1] = \frac{\mathbf{E}[X_1](1-\mathbf{E}[X_1])}{s}\\ &= \frac{\mathbf{E}[q_i](1-\mathbf{E}[q_i])}{s} \le \frac{\mathbf{E}[q_i]}{s}\end{aligned}$$

□

Lemma 10 *For any positive integer i, we have:*

$$|\mathbf{E}[q_i]-\phi(2^i,p_1)| \le |p_1-p_2| \le \frac{d_{-4}}{100}\epsilon_1$$

Proof:

$$\begin{aligned}&|\mathbf{E}[q_i]-\phi(2^i,p_1)|\\ =\ &|p_2(1-\phi(2^i-1,p_1))+(1-p_2)\phi(2^i-1,p_1)-\\ &p_1(1-\phi(2^i-1,p_1))-(1-p_1)\phi(2^i-1,p_1)|\\ =\ &|(p_1-p_2)(1-2\phi(2^i-1,p_1))|\\ \le\ &|p_1-p_2| \le |p_1-p|+|p_2-p| \le \frac{d_{-4}}{100}\epsilon_1\end{aligned}$$

□

Lemma 11 *For any given $\delta > 0$, there exists constant s' such that for any $s \geq s'$ and any positive integer i where $\phi(2^i, p_1) > d_{-4}$, we have:*

$$\Pr[|q_i - \phi(2^i, p_1)| \leq \frac{1}{50}\epsilon_1\phi(2^i, p_1)] > 1 - \delta$$

Proof: Take $s' = \frac{10000}{d_{-4}^2\epsilon_1^2\delta} + 1$, then for any $s \geq s'$, Lemma 9 tells us that:

$$\text{VAR}[q_i] \leq \frac{1}{s} < \frac{d_{-4}^2\epsilon_1^2\delta}{10000}$$

By Chebyshev's inequality:

$$\Pr[|q_i - \mathbf{E}[q_i]| \geq \frac{d_{-4}}{100}\epsilon_1] \quad \leq \quad \text{VAR}[q_i] \cdot \frac{10000}{d_{-4}^2\epsilon_1^2} < \delta$$

Lemma 10 tells us that:

$$|\mathbf{E}[q_i] - \phi(2^i, p_1)| \leq \frac{d_{-4}}{100}\epsilon_1$$

Combining the above two equations yields:

$$\Pr[|q_i - \phi(2^i, p_1)| \geq \frac{1}{50}d_{-4}\epsilon_1] < \delta$$

Finally, given $\phi(2^i, p_1) > d_{-4}$, we can conclude:

$$\Pr[|q_i - \phi(2^i, p_1)| \leq \frac{1}{50}\epsilon_1\phi(2^i, p_1)] > 1 - \delta$$

□

A.4 Translating q_i to $\hat{p}$

The EEC algorithm will eventually output $\hat{p}$ by solving $\phi(2^i, \hat{p}) = x$ for some x, where x is close to $\phi(2^i, p_1)$ for some i. This section intends to establish that such $\hat{p}$ will be close to p_1. Since p_1 is close to p, this will help to eventually show that $\hat{p}$ is close to p itself. The key step in the following proof is to show that the small error in x will not be amplified excessively when we translate x to $\hat{p}$.

Lemma 12 *For any positive integer i and any real number x where $\phi(2^i, p_1) < d_2$ and $|x - \phi(2^i, p_1)| \leq \frac{1}{50}\epsilon_1\phi(2^i, p_1)$, we have (deterministically):*

$$|\hat{p} - p_1| \leq \epsilon_1 p_1 \quad \textit{where } \hat{p} = \phi^{(-1)}(2^i, x)$$

Proof: This lemma is trivial when $p_1 = 0$, since it implies $\phi(2^i, p_1) = 0$. Thus we only need to consider $p_1 > 0$. Define:

$$\begin{aligned} x_1 &= \phi(2^i, p_1) < d_2 < 0.5 \\ x_2 &= x \leq (1 + \frac{1}{50}\epsilon_1)\phi(2^i, p_1) \leq 1.01 d_2 < 0.49 \end{aligned}$$

We have:

$$\frac{|\hat{p} - p_1|}{p_1} \quad = \quad \frac{|\phi^{-1}(2^i, x_2) - \phi^{-1}(2^i, x_1)|}{\phi^{-1}(2^i, x_1)}$$

Recall the mean value theorem: If $f : [x_1, x_2] \to \mathbf{R}$ is continuous and differentiable over $[x_1, x_2]$, then there exists $x_3 \in [x_1, x_2]$ such that $f(x_2) - f(x_1) = f'(x_3)(x_2 - x_1)$. Let $f(x) = \phi^{-1}(2^i, x) = (1 - 2x)^{2^{-i}}$, and we have $f'(x) = -2 \times 2^{-i}(1 - 2x)^{2^{-i}-1}$. Mean value theorem tells us that there exists $x_3 \in [x_1, x_2]$ and $x_4 \in [0, x_1]$ such that:

$$\begin{aligned} (1-2x_2)^{2^{-i}} - (1-2x_1)^{2^{-i}} &= -2 \times 2^{-i}(1-2x_3)^{2^{-i}-1}(x_2 - x_1) \\ (1-0)^{2^{-i}} - (1-2x_1)^{2^{-i}} &= -2 \times 2^{-i}(1-2x_4)^{2^{-i}-1}(0 - x_1) \end{aligned}$$

Therefore:

$$\begin{aligned}
\frac{|\hat{p}-p_1|}{p_1} &= \left|\frac{-2\times 2^{-i}(1-2x_3)^{2^{-i}-1}(x_2-x_1)}{-2\times 2^{-i}(1-2x_4)^{2^{-i}-1}(0-x_1)}\right| \\
&= \frac{(1-2x_3)^{2^{-i}-1}}{(1-2x_4)^{2^{-i}-1}}\frac{|x_2-x_1|}{x_1} \\
&\leq \frac{1}{50}\epsilon_1(\frac{1-2x_4}{1-2x_3})^{1-2^{-i}} \leq \frac{1}{50}\epsilon_1(\frac{1}{1-2x_3})^{1-2^{-i}} \\
&\leq \frac{1}{50}\epsilon_1(\frac{1}{1-2x_3}) \leq \epsilon_1
\end{aligned}$$

The last step holds because $x_3 \leq x_2 < 0.49$. □

A.5 Multiple Levels

The EEC algorithm may use up to $\log n$ EEC levels, and the algorithm should ideally stop at some "appropriate" level h. However, it is possible that the algorithm stops and outputs $\hat{p}$ before level h, or does not stop at level h and outputs $\hat{p}$ from later levels. This section thus aims to analyze such stopping behavior. Recall that the algorithm stops and outputs $\hat{p}$ at the smallest i where $q_i \in (c_1, c_2)$. For any given $p \in (0, d_0]$, let h be the unique integer (as shown in Lemma 5) such that $\phi(2^h, p) \in (d_0, d_1] \subset (c_1, c_2)$. We will show that it is rather unlikely for the algorithm to stop at or before level $h-2$, despite that h may reach $\log n = \omega(1)$. We will also show that $q_h \in (c_1, c_2)$ is satisfied with $1-\delta$ probability. Combining these results shows that the algorithm will stop at either level $h-1$ or level h with probability close to 1.

Lemma 13 *For any given $\delta > 0$, there exists constant s' such that for any given $s \geq s'$, any given $p \in (0, d_0]$, and any given h where $\phi(2^h, p) \in (d_0, d_1]$, we have:*

$$\sum_{i=1}^{h-2} \Pr[q_i > c_1] < \delta$$

Proof: First, Lemma 5 assures the existence and uniqueness of h. For any given $i \in [1, h-2]$, Lemma 5 tells us that $\phi(2^i, p_1) < 1.4d_{-1}$. From Lemma 10, we have:

$$\begin{aligned}
\mathbf{E}[q_i] &\leq \phi(2^i, p_1) + \frac{d_{-4}}{100}\epsilon_1 < 1.4d_{-1} + \frac{d_{-4}}{200} < 0.22 \\
\mathbf{E}[q_i] &\leq \phi(2^i, p_1) + |p_2 - p_1| < 1.4\phi(2^i, p) + |p_2 - p_1| \\
&\leq 1.4\times 2^i p + |p_2 - p_1|
\end{aligned}$$

Applying Chebyshev's inequality and Lemma 9 yields:

$$\begin{aligned}
\Pr[q_i > c_1] &\leq \Pr[|q_i - \mathbf{E}[q_i]| > |c_1 - \mathbf{E}[q_i]|] \\
&\leq \Pr[|q_i - \mathbf{E}[q_i]| > c_1 - 0.22] \\
&\leq \frac{\mathrm{VAR}[q_i]}{(0.03)^2} \leq 10000\mathbf{E}[q_i]/(9s) \\
&< 10000(1.4\times 2^i p + |p_2 - p_1|)/(9s) \\
&< 10000(1.4\times 2^i p + \frac{2}{\log\frac{1}{p}})/(9s)
\end{aligned}$$

Since $h \leq \lfloor \log\frac{1}{p} \rfloor \leq \log\frac{1}{p}$, we have

$$\begin{aligned}
\sum_{i=1}^{h-2} \Pr[q_i > c_1] &< \frac{10000}{9s}\left(\frac{2(h-2)}{\log\frac{1}{p}} + \sum_{i=1}^{h-2} 1.4\times 2^i p\right) \\
&< 10000\times(2+0.7)/(9s) = 3000/s
\end{aligned}$$

Thus for $s \geq s' = 3000/\delta$, we have $\sum_{i=1}^{h-2} \Pr[q_i > c_1] < \delta$. □

Lemma 14 *For any given $\epsilon_2 < 0.01$ and any given $\delta > 0$, there exists constant s' such that for $\forall s \geq s'$, any given $p \in (0, d_0]$, and any given h where $\phi(2^h, p) \in (d_0, d_1]$, we have:*

$$\Pr[q_h \in (c_1, c_2)] > 1 - \delta$$

Proof: First, Lemma 5 assures the existence and uniqueness of h. By Lemma 3 and since $p_1 \in [(1-\epsilon_2)p, (1+\epsilon_2)p]$, we have:

$$\begin{aligned} \phi(2^h, p_1) &\in [\phi(2^h, 0.99p), \phi(2^h, 1.01p)] \\ &\subset [0.99\phi(2^h, p), 1.01\phi(2^h, p)] \subset (0.99d_0, 1.01d_1) \end{aligned}$$

Since $\phi(2^h, p_1) > 0.99d_0 > d_{-4}$, we can apply Lemma 11 to find constant s' such that for all $s \geq s'$:

$$\begin{aligned} & \Pr[q_h \in (c_1, c_2)] \\ > \; & \Pr[q_h \in [(1-0.01)0.99d_0, (1+0.01)1.01d_1]] \\ > \; & \Pr[q_h \in [(1-0.01)\phi(2^h, p_1), (1+0.01)\phi(2^h, p_1)]] \\ \geq \; & \Pr[|q_h - \phi(2^h, p_1)| \leq \frac{1}{50}\epsilon_1\phi(2^h, p_1)]] \\ > \; & 1 - \delta \end{aligned}$$

□

The following lemma helps us to later reason about the algorithm's behavior when $p > d_0$.

Lemma 15 *For any given $\epsilon_2 < 0.01$ and any given $\delta > 0$, there exists constant s' such that for all $s \geq s'$, any $p \in (0, \frac{1}{2}]$, and any positive integer i where $\phi(2^i, p) > d_1$, we have:*

$$\Pr[q_i > \psi(c_1)] > 1 - \delta$$

Proof: We first show that $\phi(2^i, p_1) > 0.99d_1$.

- If $p_1 \leq \frac{1}{2}$, we have:

$$\begin{aligned} \phi(2^i, p_1) &> \phi(2^i, (1-\epsilon_2)p) > \phi(2^i, 0.99p) \\ &\geq 0.99\phi(2^i, p) > 0.99d_1 \end{aligned}$$

- If $p_1 > \frac{1}{2}$, notice that we have $p_1 < (1+\epsilon_2)p < 0.505$. We will prove $0.99d_1 < \phi(2^i, p_1) < 0.5$ via an induction on i. For $i = 1$, we have $\phi(2^i, p_1) = 2p_1(1-p_1)$ and $0.99d_1 < 2 \times 0.505 \times 0.495 < 2p_1(1-p_1) < 0.5$. For $i \geq 2$, since $\phi(2^{i-1}, p_1) \in (0, \frac{1}{2})$, we have
$$0.99d_1 < \phi(2^{i-1}, p_1) < \psi(\phi(2^{i-1}, p_1)) < 0.5$$
Since $\phi(2^i, p_1) = \psi(\phi(2^{i-1}, p_1))$, we have $0.99d_1 < \phi(2^i, p_1) < 0.5$.

Since $\phi(2^i, p_1) > 0.99d_1 > d_{-4}$, Lemma 11 shows that there exists constant s' such that for all $s \geq s'$, with probability at least $1 - \delta$:

$$q_i \geq (1 - \frac{1}{50}\epsilon_1)\phi(2^i, p_1) > 0.99 \times 0.99d_1 > \psi(c_1)$$

□

A.6 Proving Theorem 1

As explained in Section 3.5, our EEC algorithm can use either of the following two approaches to output a final $\hat{p}$:

$$\phi(2^i, \hat{p}) = q_i \tag{1}$$

$$\phi(2^i, \hat{p}) = \frac{q_i + \psi(q_{i-1})}{2} \tag{2}$$

Our proof for Theorem 1 holds for both cases.

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 $1/(n+k) \leq a < b \leq 1/4$) 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 $\hat{p} \leq (1+\epsilon)a$.*
- *If $p > b$, output $\hat{p}$ where $\hat{p} \geq (1-\epsilon)b$.*

Proof: It is obvious that the algorithm will return 0 when $p = 0$. Thus we only need to consider $p > 0$, or equivalently $p \geq \frac{1}{n+k}$. Without loss of generality, assume $\epsilon \leq 1$. For any given $\epsilon \in (0, 1]$ and $\delta > 0$, we can trivially find $\epsilon_1 \in (0, 0.5]$, $\epsilon_2 \in (0, 0.01)$, and $\delta_1 > 0$, such that $\epsilon = 2\epsilon_1 + \epsilon_2$ and $\delta = 6\delta_1$.

Lemma 7 shows that for any δ_1, there exists constant s' such that if $s \geq s'_1$ and $n \geq s'_1$, then with probability at least $1 - \delta_1$:

$$|P_1 - p| \leq \frac{d_{-4}}{200}\epsilon_1 \quad \text{and} \quad |P_2 - p| \leq \frac{d_{-4}}{200}\epsilon_1$$
$$|P_1 - p| \leq \frac{1}{\log \frac{1}{p}} \quad \text{and} \quad |P_2 - p| \leq \frac{1}{\log \frac{1}{p}}$$

Lemma 8 shows that for any δ_1 and any s, there exists constant n' such that if $n \geq n'$, then $Pr[|P_1 - p| \leq \epsilon_2 p] \geq 1 - \delta_1$. Thus for $s \geq s'_1$ and sufficiently large n, with probability at least $1 - 2\delta_1$, we have $P_1 = p_1$ and $P_2 = p_2$ where p_1 and p_2 satisfy:

$$|p_1 - p| \leq \frac{d_{-4}}{200}\epsilon_1 \quad \text{and} \quad |p_2 - p| \leq \frac{d_{-4}}{200}\epsilon_1$$
$$|p_1 - p| \leq \frac{1}{\log \frac{1}{p}} \quad \text{and} \quad |p_2 - p| \leq \frac{1}{\log \frac{1}{p}}$$
$$|p_1 - p| \leq \epsilon_2 p$$

All discussions below are conditioned upon the above 5 overarching conditions being met. For $p \leq d_0$, let h be the unique integer (as shown by Lemma 5) such that $\phi(2^h, p) \in (d_0, d_1]$.

Case 1: $p \in [a, b]$**.** Given how we determined the values for l_1 and l_2, and given $(d_0, d_1] \subset (c_1, c_2)$, we know that $h \in [l_1, l_2]$. By Lemma 13 and Lemma 14, for any given δ_1, there exists constant s'_2 such that $\forall s \geq s'_2$, we have

$$\sum_{i=1}^{h-2} Pr[q_i > c_1] \quad < \quad \delta_1$$
$$Pr[q_h \in (c_1, c_2)] \quad > \quad 1 - \delta_1$$

This means that for $s \geq s'_2$, with probability at least $1 - 2\delta_1$, the algorithm will stop at level h or $h-1$ and output $\hat{p}$. (For the boundary case of $h = l_1$, the algorithm will only stop at level h.)

Now consider level i for $h - 2 \leq i \leq h$. Lemma 5 tells us that $\phi(2^i, p_1) \in (d_{-4}, d_2)$. Apply Lemma 11, Lemma 12, and a trivial union bound for $h - 2 \leq i \leq h$. We have for any δ_1, there exists constant s'_3 such that for any $s \geq s'_3$, with probability at least $1 - \delta_1$ all the following 3 equations hold:

$$|\hat{p} - p_1| \leq \epsilon_1 p_1 \qquad \text{where } \hat{p} = \phi^{(-1)}(2^h, q_h)$$
$$|\hat{p} - p_1| \leq \epsilon_1 p_1 \qquad \text{where } \hat{p} = \phi^{(-1)}(2^{h-1}, q_{h-1})$$
$$|\hat{p} - p_1| \leq \epsilon_1 p_1 \qquad \text{where } \hat{p} = \phi^{(-1)}(2^{h-2}, q_{h-2})$$

Applying Lemma 6 further yields:

$$|\hat{p} - p_1| \leq \epsilon_1 p_1 \qquad \text{where } \hat{p} = \phi^{(-1)}\left(2^h, \tfrac{q_h + \psi(q_{h-1})}{2}\right)$$
$$|\hat{p} - p_1| \leq \epsilon_1 p_1 \qquad \text{where } \hat{p} = \phi^{(-1)}\left(2^{h-1}, \tfrac{q_{h-1} + \psi(q_{h-2})}{2}\right)$$

This means that regardless of whether the algorithm stops at level h or $h-1$, and regardless of whether the algorithm uses Equation 1 or 2 for generating $\hat{p}$, we always have $|\hat{p} - p_1| \leq \epsilon_1 p_1$.

Since $|p_1 - p| \leq \epsilon_2 p$, we can now conclude that for $s \geq \max(s_2', s_3')$, with probability at least $1 - 4\delta_1$:

$$\begin{aligned}\left|\frac{\hat{p} - p}{p}\right| &\leq \left|\frac{\hat{p} - p_1}{p_1}\right| \cdot \left|\frac{p_1}{p}\right| + \left|\frac{p_1 - p}{p}\right| \\ &\leq \epsilon_1(1 + \epsilon_2) + \epsilon_2 < 2\epsilon_1 + \epsilon_2 = \epsilon\end{aligned}$$

Finally, the probability that the 5 overarching conditions are not met is at most $2\delta_1$. Thus with probability at least $1 - 6\delta_1$, we have $\hat{p} \in [(1 - \epsilon)p, (1 + \epsilon)p]$.

Case 2: $\frac{1}{n+k} \leq p < a$**.** Imagine an alternative algorithm $\mathcal{A}$ using infinite number of levels, from level l_1 through level ∞. Case 1 already shows that with probability at least $1 - \delta$, algorithm $\mathcal{A}$ would output $\hat{p}$ where $\hat{p} \leq (1 + \epsilon)p < (1 + \epsilon)a$.

The only difference between algorithm $\mathcal{A}$ and the actual EEC algorithm is that the EEC algorithm only has levels l_1 through l_2. It is thus possible for the EEC algorithm to fail to estimate p from any level between l_1 and l_2, and to directly output 0. However, even in such a case, we still have $\hat{p} = 0 < (1 + \epsilon)a$.

Case 3: $b < p \leq d_0$**.** First, since $\phi(2^{l_1}, b) > c_1$, we have

$$\phi(2^{l_1+1}, p) > \phi(2^{l_1+1}, b) = \psi(\phi(2^{l_1}, b)) > \psi(c_1) > d_0$$

This implies that $h \leq l_1 + 1$. For $h = l_1 + 1$, we have $c_1 < \phi(2^{l_1}, b) < \phi(2^{l_1+1}, b) < \phi(2^h, p) \leq d_1 < c_2$, and thus $\phi(2^{l_1+1}, b) \in (c_1, c_2)$ and $l_2 \geq l_1 + 1$. This means that $h \in [l_1, l_2]$, and Case 1 already shows that with probability at least $1 - \delta$, the algorithms will stop at level $h - 1$ or h and output an estimate $\hat{p}$ where $\hat{p} \geq (1 - \epsilon)p > (1 - \epsilon)b$. For $h = l_1$, the same arguments apply. For $h \leq l_1 - 1$, we have $\phi(2^{l_1}, p) > d_1$. Thus for any given δ_1, Lemma 15 tells us that there exists s_4' such that for $s \geq s_4'$:

$$Pr[q_{l_1} > \psi(c_1)] > 1 - \delta_1$$

If $q_{l_1} \geq c_2$, the algorithm will output $\hat{p} = 1/4 > (1 - \epsilon)b$. If $\psi(c_1) < q_{l_1} < c_2$, the algorithm will output (notice that here the algorithm will only use Equation 1 to produce $\hat{p}$):

$$\begin{aligned}&\hat{p} = \phi^{(-1)}(2^{l_1}, q_{l_1}) > \phi^{(-1)}(2^{l_1}, \psi(c_1)) \\ \Rightarrow\quad &\phi(2^{l_1}, \hat{p}) > \psi(c_1)\end{aligned}$$

We claim that $\psi(c_1) \geq \phi(2^{l_1}, b)$, which in turn will imply that $\hat{p} > b > (1 - \epsilon)b$. For $l_1 = 1$, we trivially have $\psi(c_1) = \phi(2, 0.25) \geq \phi(2, b)$. For $l_1 \geq 2$, if $\psi(c_1) < \phi(2^{l_1}, b)$, then we would have $c_1 < \phi(2^{l_1-1}, b)$. This means that even level $l_1 - 1$ should have been used in the algorithm, leading to a contradiction.

Case 4: $d_0 < p \leq 0.5$**.** Since $\phi(2^{l_1}, p) \geq \phi(2, p) > \phi(2, d_0) = d_1$, Lemma 15 shows that for any given δ_1, there exists s_5' such that for $s \geq s_5'$:

$$Pr[q_{l_1} > \psi(c_1)] > 1 - \delta_1$$

Same as in Case 3, it can be shown that (deterministically):

$$q_{l_1} > \psi(c_1) \quad \Rightarrow \quad \hat{p} > (1 - \epsilon)b$$

Finally, combing the 4 cases and taking $s = \max(s_1', s_2', s_3', s_4', s_5')$ complete the proof. □

A.7 Removing the Assumption on n Being Sufficiently Large

This section is mainly for theoretical interests and aims to show that the requirement on n being sufficient large as in Theorem 1 is not necessary. Our discussion here has limited practical relevance since our experiments have already show that the algorithm provides good guarantee in practice.

To remove this assumption on n, notice that the theorem already holds for $n \geq n'$ where n' is some constant. All we need to worry about is where $n < n'$. On the other hand, if we already know that n is upper bounded by some constant, the following extremely trivial algorithm will suffice for proper error estimation: We simply insert r known bits into the packet, where r is a function of ϵ, δ, and n', and thus is a constant $O(1)$ with respect to n. Same as before, these r bits are placed into uniformly random slots, while the data bits go into the remaining slots. The receiver simply examines what fraction among these r bits are flipped and then estimate p to be this fraction. It is trivial to show that doing so provides good estimation quality:

Lemma 16 *For any given $\epsilon > 0$, $\delta > 0$, $n' > 0$, there exists constant r such that for any $p \in [0,1]$ and for any $n \leq n'$, using r known bits in the above trivial algorithm will output $\hat{p}$ with the following guarantee: With probability at least $1-\delta$:*

$$\hat{p} \in [(1-\epsilon)p, (1+\epsilon)p]$$

Proof: The lemma trivially holds for $p = 0$. For $p > 0$, it necessarily means that $p \geq \frac{1}{n+r}$. Obviously, $r\hat{p}$ is a random variable follows hypergeometric distribution with parameter $(n+r, (n+r)p, r)$. We have

$$\mathrm{VAR}[\hat{p}] \quad = \quad \frac{p(1-p)n}{r(r+n-1)} \leq \frac{np}{r(r+n)} \leq \frac{n'p^2}{r(r+n)p} \leq \frac{n'p^2}{r}$$

Invoke Chebyshev's inequality and we have:

$$\Pr[|\hat{p}-p| < \epsilon p] \quad > \quad 1 - \frac{\mathrm{VAR}[\hat{p}]}{\epsilon^2 p^2} \geq 1 - \frac{n'}{r\epsilon^2}$$

Thus for $r = \frac{n'}{\delta\epsilon^2}$, with probability at least $1-\delta$, we have $\hat{p} \in [(1-\epsilon)p, (1+\epsilon)p]$ □

Combining Theorem 1 and the above lemma directly gives us the following:

Theorem 17 *Consider any given positive constants ϵ and δ. There exists constant n' such that:*

- *If $n \geq n'$, then there exists constant $s = O(1)$ such that using s in our EEC algorithm (together with input a and b where $1/(n+k) \leq a < b \leq 1/4$) 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 $\hat{p} \leq (1+\epsilon)a$.*
  - *If $p > b$, output $\hat{p}$ where $\hat{p} \geq (1-\epsilon)b$.*
- *If $n < n'$, then there exists constant $r = O(1)$ such that using r known bits in the earlier trivial algorithm will provide the following guarantee: With probability at least $1-\delta$, output $\hat{p}$ where $\hat{p} \in [(1-\epsilon)p, (1+\epsilon)p]$.*

Finally, the same trivial algorithm (i.e., inserting known bits) can also be used to remove the assumption on $b \leq 1/4$ under any n (see discussion in Section 3 on pilot bits).