mEEC: A Novel Error Estimation Code with Multi-Dimensional Feature

Zhenghao Zhang and Piyush Kumar
Computer Science Department
Florida State University Tallahassee, FL 32306, USA

***Abstract*—Error estimation code estimates the bit error ratio in the data bits, and is very useful in many applications, such as estimating the number of errors in a packet transmitted over a wireless link. In this paper, we propose a novel error estimation code, mEEC, that outperforms the existing code by more than 10%-20% depending on the packet sizes, as well as being more unbiased. mEEC is mainly based on the idea of grouping multiple blocks of sampled data bits into a super-block, thus creating a multi-dimensional feature, then compressing features into a single number, called the color, as the coded bits. Through an intelligent coloring scheme, the blocks in a super-block share the cost of covering low probability cases, which allows the decoder to recover the actual feature values from the color value even in the presence of error. mEEC also adopts a lightweight redistribution step, which is guided by the solution of an optimization problem and further reduces the estimation errors and bias. We also show that mEEC can be implemented within reasonable table storage size and low time complexity.**

I. Introduction

Error estimation code has been recently studied, which can estimate the Bit Error Ratio (BER) of the data bits by computing certain *features* of sampled data bits, and using the features as the *code bits* to be transmitted along with the data bits. It is very useful in many applications, such as estimating the number of errors in a *partial packet*, i.e., a packet with some corrupted bits, to allow a wireless transmitter to send just enough parity bits to correct the errors [2], [15], [13]. The first error estimation code was proposed in [1], followed by several other codes [14], [9], [3], [13], [4], with current state of art being the seminal work described in [4], gEEC, which has brought the performance of error estimation to a new level, as well as revealing important theoretical insights.

In this paper, we propose a new error estimation code, called the *mEEC*, which significantly outperforms gEEC when both gEEC and mEEC code bits are *immune* to errors, i.e., not corrupted, although the data bits can be corrupted. Error estimation both with and without immunity have been studied in the literature [4]; however, as estimation performance is understandably better with than without immunity, immunity is often preferred in practice when implementing wireless network protocols harnessing partial packets [15], [13]. Typically, such protocols add the error estimation code bits as part of the packet header, guarded by the header CRC or even protected by error correction codes, because to process a partial packet, the packet header, which contains important information such as the sender or receiver addresses, must still remain intact. As length of the error estimation code is small, such as 84 bits for the mEEC code, it does not significantly increase the packet header length. mEEC achieves smaller estimation errors than gEEC on metrics such as the relative mean squared error, on average by more than 10-%20% depending on the packet sizes, sometimes as high as over 40%. At the same time, mEEC is more unbiased than gEEC, e.g., the mean bias of mEEC is within $\pm 3.5\%$ for 12000-bit packets when the BER is in $[0.001, 0.1]$, while the mean bias of gEEC can be much larger, such as over 9% when the BER is 0.001. Also, even with these improvements, mEEC still has reasonably low encoding and decoding complexities.

The main innovation of mEEC is to introduce a novel *multi-dimensional feature* and a color mapping as the code, exploiting the fact that the error ratios in partial packets are typically small [1], [4]. To be more specific, mEEC divides data bits into *blocks*, then groups multiple blocks into a *super-block*, and uses only one number, i.e., the *color*, to represent all features of the blocks in the same super-block. The advantage of this approach is that the color can be represented by much fewer bits than the full features of the blocks, while actually not reducing the ability of the decoder to reproduce all feature values because only few features are expected to be changed significantly by errors. In addition, mEEC adopts a novel redistribution step, which is guided by the solution of a linear programming problem to bounded estimation bias while minimizing estimation error.

The rest of the paper is organized as follows. Section II discusses discusses related work. Section III explains the encoding and decoding process of mEEC. Section IV describes the details of the design of mEEC. Section VI presents the evaluation results. Section VII concludes the paper.

II. Related Work

The oldest form of error estimation code is just the pilot bits embedded in the data bits such as that implemented in [7], which uses the ratio of corrupted pilot bits as the estimate of the BER in the data bits. The pilot bit solution however does not perform well for low error ratios, because the number of pilot bits is small and may not encounter any error in such cases. To overcome this, EEC was proposed in [1], which essentially uses the parity of multiple data bits to replace the pilot bit and adopts a simple decoding algorithm. Since the introduction of EEC, many EEC has been proposed at about the same time [13], [14], [9], [3], [4], outperforming EEC by using the maximum likelihood estimation [13], random

walk [14], exploiting the bursty nature of errors in wireless links [9], and different coding structures [3], [4]. The latest is gEEC [4], which was also used as a basic building block for error estimation in insertion and deletion channels [5]. mEEC is built on the foundation of earlier codes but achieves even better performance by novel techniques such as grouping blocks into super blocks and the redistribution step.

Side information from the physical layer, such as the symbols constellation and bit confidence level, has been used for better error estimation or error correction [10], [6], [8]. In this paper, however, juts as in [4], we focus on the case with no such side information, therefore mEEC is not platform-dependent because the availability and quality of such side information may vary depending on the physical layer modulation. There has been recent interest in using EEC for additional error correction [12], [11], which is out of the scope of this paper.

III. Encoding and Decoding of mEEC

In this section, we explain the encoding and decoding process of mEEC.

A. Encoding

The construction of the mEEC code is very simple:

1) *Whiten* the original packet, i.e., do an exclusive-or with a random bit string of the same length, then randomly sample M bits, *with no repeat*.
2) Divide the M sampled bits into N *blocks*, each has W bits. For block i, calculate x_i, which is the *feature* of this block and is basically the number of 2-bit sequences that are either 00 or 11 among all 2-bit sequences.
3) Group every 3 blocks into a *super-block*, and find a more succinct representation of their features, called the *color*, which is coded in U bits and denoted as C_I for super block I. This is implemented by pre-computing a color assignment pattern in the 3-D space, stored at both the encoder and the decoder, such that every integer point, which represents a possible combination of the features of 3 blocks, is given a color in the assignment. mEEC code bits are basically the colors of all super-blocks.

B. Decoding

After the packet is received, the BER is estimated as follows:

1) With the same procedure as the encoder except not whitening the packet, the decoder calculates the features of the received blocks, denoted for block i as y_i, which will likely be different from x_i if some bits have been corrupted.
2) For each super-block, the decoder selects a list of *candidate points*, denoted as Π_I, which is explained in the following using super-block 1 as an example. The decoder finds a point in the 3-D space with coordinates $[y_1, y_2, y_3]$, called the *received point*. Π_1 basically contains points in the 3-D space of color C_1 that are close to the received point. In our current implementation, for simplicity, two points are considered close if the total difference in their coordinates is less than 30.
3) As the prior distribution of the error ratio is not always known, the decoder searches for an error ratio with the maximum likelihood as the *initial estimate*, by assuming that each bit is corrupted independently. To be more specific, let the likelihood of error ratio θ be $L(\theta)$. Under the independence assumption, $L(\theta) = \prod_{I=1}^{\frac{N}{3}} L_I(\theta)$, where $L_I(\theta)$ is the likelihood value based only on super-block I. The calculation of $L_1(\theta)$ is explained in the following which can be applied to other super-blocks. Basically, the decoder calculates the *point likelihood* of every point in Π_1, and selects point $[\hat{x}_1, \hat{x}_2, \hat{x}_3]$ as the *representative*, if it has the maximum point likelihood, which is used as $L_1(\theta)$. Again under the independence assumption, the point likelihood of a point in Π_1, such as $[\hat{x}_1, \hat{x}_2, \hat{x}_3]$, is $\prod_{i=1}^{3} W(\hat{x}_i) T_\theta(\hat{x}_i \to y_i)$, where:
   - $W(x)$ denotes the probability that the feature of a block at the encoder side is x. As the bits are randomly selected, x follows Binomial distribution denoted as $B(x, \frac{W}{2}, 0.5)$, where
   $$B(x, \frac{W}{2}, 0.5) = \binom{\frac{W}{2}}{x} x^{0.5} (\frac{W}{2} - x)^{0.5}.$$
   - $T_\theta(x \to y)$ denotes the probability that given the error ratio is θ, the errors changed the feature from x to y. It can be calculated as
   $$\sum_{u=\max\{0, x-y\}}^{\min\{x, \frac{W}{2}-y\}} B(u, x, 2\theta) B(y+u-x, \frac{W}{2} - x, 2\theta).$$

   As the the likelihood value as a function of the error ratio is usually smooth with only one global maximum, to speed up the search process, the Newton method should be employed, as explained in more details in Section **??**.
4) Redistribution: Given the initial estimate θ, mEEC randomly selects η as the final output according to a fixed Probability Density Function (PDF) for θ, denoted as $\Phi_\theta()$; to be more specific, the probability that η is selected at this step is $\Phi_\theta(\eta)$. The redistribution step, from a high level, is like ironing out the wrinkles of the maximum likelihood estimation, which sometimes leads to non-optimal results due to certain approximations used. The calculation of $\Phi_\theta()$ is explained in Section **??**.

C. An Example

Fig. 1 shows a simple example, which helps explaining some key concepts in mEEC. In the example, the code has only two super-blocks, each consists of 3 blocks, where each block has only 2 bits. The feature value, in this case, is clearly either 0 or 1, and there are a total of 8 combinations of feature values of a super-block. 2 bits are used as the color, therefore there are 4 colors, with the color assignments shown in the right of the figure. Suppose the bits sampled by the encoder are 011100101000, resulting in features $[0, 1, 1]$ and $[0, 0, 1]$ for the two super-blocks, respectively. Therefore,

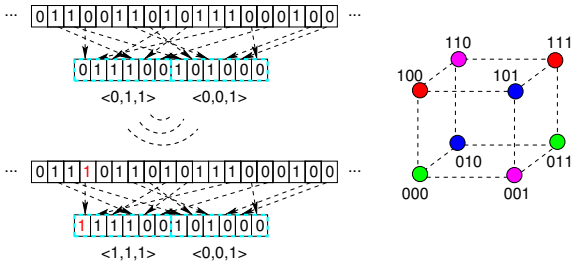

Fig. 1. A simple example of mEEC.

the transmitted colors are green and purple, respectively. The channel corrupted the first sampled bit and turned the bits into $\underline{1}11100101000$, therefore the received points are $[1, 1, 1]$ and $[0, 0, 1]$, respectively. For the first super-block, the candidate list has two points of color green, which are $[0, 0, 0]$ and $[0, 1, 1]$. At low error ratios, the decoder will pick $[0, 1, 1]$ as the representative, as it is closer to $[1, 1, 1]$. Similarly, $[0, 0, 1]$ will be picked for the second super-block.

D. Values of the Parameters

In our current implementation, the parameters have been empirically selected as $M = 3360$, $N = 21$, $W = 160$, which can be applied to packets of 420 bytes or above. Different parameters will need to be picked for smaller packets, which is not discussed in this paper due to the limit of space. $U = 12$, and therefore the mEEC code is 84 bits, i.e., less than 11 bytes.

IV. The Design Rationale of mEEC

In this section, we explain the details of various aspects of mEEC, mostly following the order of their appearances in the encoding and decoding process.

A. Block Feature

To form the blocks, mEEC takes random samples from the data bits, which randomizes the errors patterns in the sampled bits, a common practice in most error estimation codes. mEEC does not sample the same data bits more than once, which avoids possible biases when some bits are sampled multiple times in case the size of the packet is small. The feature of a block in mEEC is the number of two-bit sequences that are either 00 or 11, and is designed to better match the expected error ratio, which is usually low in wireless transmissions. The feature of a block should be sensitive to the number of errors, i.e., corrupted bits should likely lead to a change in the feature value, while using as few bits as possible to reduce the overhead. Commonly used features, such as that in gEEC, usually start by counting the number of 1s in the block then using various ways to compress the count. For a block of with W bits, the size of the uncompressed feature is $\log_2(W)$ bits. To reduce the size of the feature, we note that for low error ratios, it rarely occurs that two neighboring bits are both corrupted. Therefore, if the 00 and 11 sequences are in one category and 01 and 10 in another category, the errors will almost always change a sequence from one category to the other; as a result, counting the number of sequences in one category is almost as sensitive as counting the number of 1s. However, this reduces the size of the uncompressed feature by one bit. Actually, counting the number of 00 and 11 sequences can be considered as a generalization of counting the number of 1s, where the former is to consider sequences of length 2 and the latter 1. It is, however, difficult to further generalize this to longer sequences because the assumption that only one bit is corrupted in a sequence will cease to hold.

B. Grouping Blocks into Super-Blocks

Grouping blocks into super-blocks is the most important aspect of mEEC, which helps mEEC using fewer bits to represent the features without losing accuracy in most cases. It is guided by a simple idea, which can be explained in the following at a high level. If the features of the blocks are to be transmitted individually, enough bits must be allocated to represent each individual feature without introducing too much ambiguity in the decoding process; in other words, the binary representation of a feature must cover many cases that occur with relatively low probability. However, as such cases occur with low probability, among a group of blocks, usually no or just few blocks should be in such cases. mEEC groups the blocks into a super-block, and therefore amortizes the cost of covering the low probability cases among all blocks. In theory, the group size should be as large as possible; the size is 3 in our current implementation because of the limitations in the implementation regarding to the sizes of certain tables.

The main advantage of grouping can be further illustrated by a simplified example in Fig. 2, where each super-block has only 2 blocks. Note that, with U bits to represent the features of the blocks in a super-block, the simplest approach would be to divide U bits evenly among the blocks, and transmit the lower $\frac{U}{2}$ bits of each feature as the color. This simple approach basically partition the points in the 2-D plane into squares, and repeat the same coloring pattern for all squares. Consider point A at the center of the green square in the left of Fig. 2, some points with the same color in neighboring squares are also shown. Clearly, errors can be corrected if the received point is still within the green square because it is closer to A than to any point of the same color as A. However, if, as in the example, errors changed the feature of one block but left the other intact, resulting received point A', the decoder will mistakenly choose B. A better color assignment would be the one on the right of the figure, where the boundary are hexagons instead of squares. In effect, it allows the feature of one block to deviate further, as long as the feature of the other block does not change too much.

Mathematically, the problem is to find a color assignment to maximize the minimum distances between all points of the same color. One simple yet effective strategy is to partition the 3-D space using regular space-filling shapes, where each shape should contain the same number of integer points and repeat the same coloring pattern. As in the earlier example, in a 2-D plane, one good option is to partition the plane with hexagons. In the 3-D space, our choice is the hexagon prism, where each hexagon prism has 14 vertical layers and each layer is basically a hexagon with 291 points, leading

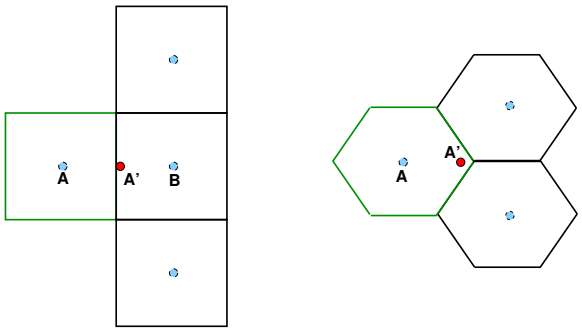

Fig. 2. Advantage of intelligent coloring.

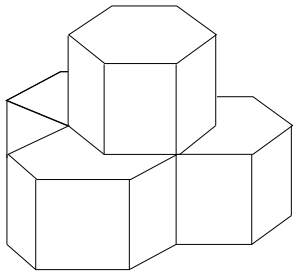

Fig. 3. The hexagon prisms in mEEC.

to a total of 4074 points in each hexagon prism. Hexagon prisms are also arranged in vertical layers, where each layer consists of hexagon prisms with the same vertical coordinates. Different vertical layers are stacked on each other, however shifted by 10 and 5 in two directions on the plane to increase the minimum distance which will otherwise be just 14, as shown in Fig. 3. The minimum distance with this approach between any points of the same color is 16.88 for 1274 colors and 17.20 for the other 2800 colors, which is fairly close to the theoretical upper bound of 19.85 shown in the Appendix. It should be mentioned that as the geometrical shapes have to partition points with integer coordinates, certain quantization errors may exist, which we have reduced by manually shifting the locations of hexagon prisms be a small amount; still, there are a very small number of points on the boundaries belong to both hexagon prisms and we basically arbitrarily assign such points to a hexagon prism.

C. The Newton Search

mEEC basically uses the Newton search to find the root of the first derivative of the likelihood function. Our current implementation makes further simplifications, which has been proven to work well in practice. To be more specific, the search is on a discrete set of error ratios from 0 to 0.175 at a step of $\frac{1}{M}$. In each iteration, the decoder calculates $L(\frac{e-1}{M})$, $L(\frac{e}{M})$, and $L(\frac{e+1}{M})$ for some integer e, where initially $e = 2$. If the first derivative at error ratio $\frac{e}{M}$, approximated as $L(\frac{e}{M}) - L(\frac{e-1}{M})$, did not change more than 1% since the last iteration, the peak usually has been found and the search is completed; otherwise, e is updated according to

$$e \leftarrow e - \lfloor \frac{L(\frac{e}{M}) - L(\frac{e-1}{M})}{L(\frac{e}{M+1}) - 2L(\frac{e}{M}) + L(\frac{e-1}{M})} \rfloor.$$

The maximum number of iterations is set at 20. Once the search is completed, the best in $[\frac{e-3}{M}, \frac{e+3}{M}]$ is chosen as the output. In rare occasions, e may become invalid values after an iteration, i.e., $e < 2$ or $\frac{e+1}{M} > 0.175$, the search also exists. Usually in such cases, based on the evaluated error ratios, the likelihood function is clearly a monotonic function and the maximum should be on one of the boundaries; otherwise a linear scan on all error ratios can be used.

The Newton search is used mainly to reduce the computation complexity compared to the linear scan on all error ratios. Somewhat surprisingly, it also improves the estimation accuracy for relative large error ratios such as 0.075. This is because the Newton search starts at the lowest error ratio and will stop at the first local maxima. In many cases, if the Newton search produces a different result than the linear scan, the likelihood function has two maxima and the second one is the maximum but was not picked, which, turns out to usually *reduce* the estimation error. In such cases, there usually is one super-block with the actual transmitted point far from the received point. When the Newton search is still on low error ratios, it may find a point closer to the received point than the actual transmitted point as the representative, which produces a local maxima, because the searched error ratio is still not large enough to swing points away to larger distances. Not picking the actual transmitted point in such cases improves the estimation because the large distances between the actual transmitted point and the received point is a very rare event, i.e., happens once out of tens or hundreds of tries depending on the error ratio. Had there been enough number of super-blocks for each packet, the effect of such rare events can be averaged out; however, as mEEC has only a very small number of super-blocks, once such event is encountered in a packet, the estimation for this packet will be way off, leading to a very large error. The Newton method, by replacing the actual transmitted point with a closer point, automatically filters out such events.

D. Redistribution

The redistribution step randomly maps the initial estimate to another value, which may appear to be non-intuitive, but actually helps further reducing the estimation error and limiting the bias. The reason is that mEEC depends on certain approximations, such as the independence of the errors, snapping the the received point to a close point in the candidate list, etc., which may lead to non-optimal behaviors, some of which can be easily corrected in this step. For example, the initial estimate never takes certain values in our implementation, such as $\frac{5}{M}$ and $\frac{10}{M}$, due to the approximation error in the transition probability under the independence assumption. During the redistribution step, some other values may be redirected to these values to help reducing the estimation error. However, the mapping has a cascading effect and must be taken with caution. Φ_θ for all θ is therefore calculated to minimize the estimation error under strict constraints, as explained in the following.

Consider a finite set of error ratios from 0 up to some $\frac{E}{M}$ at

a step of $\frac{1}{M}$. Any other error ratios are basically approximated by nearby error ratios in this set. Denote the maximum value of the initial estimate as $\frac{T}{M}$. The process of obtaining the initial estimate basically defines a $T+1$ by $E+1$ matrix Δ, where $0 \le m \le T$, $0 \le n \le E$, and $\Delta(m,n)$ is the probability that the initial estimation is $\frac{m}{M}$ given the actual error ratio is $\frac{n}{M}$. The redistribution step is basically to introduce a $T+1$ by $T+1$ matrix, denoted by Ω, and replace Δ by $\Delta' = \Omega\Delta$, where column n in Ω is exactly Φ_n, i.e., element $\Delta(m,n)$ in Ω is the probability that final output is $\frac{m}{M}$ given the initial estimate is $\frac{n}{M}$.

To achieve good performance, Ω should be found by solving an optimization problem which treats each entry as a variable. Denote $\Omega(m,n)$ as $z_{m(T+1)+n}$. The problem can be formalized as the following:

$$\min \sum_{e=0}^{E}\sum_{m=0}^{T}\sum_{n=0}^{T} z_{m(T+1)+n}\Delta(n,e)(\frac{m-e}{e})^2$$

subject to

$$z_{m(T+1)+n} \ge 0, \quad \text{all } m \text{ and } n \tag{1}$$

$$\sum_{m=0}^{T} z_{m(T+1)+n} = 1, \quad \text{all } n \tag{2}$$

$$|[\sum_{m=0}^{T}\sum_{n=0}^{T} z_{m(T+1)+n}\Delta(n,e)m] - e| \le \gamma e, \quad \text{all } e \tag{3}$$

$$\sum_{m=0}^{T}\sum_{n=0}^{T} z_{m(T+1)+n}\Delta(n,e)(\frac{m-e}{e})^2 \le \sum_{n=0}^{T}\Delta(n,e)(\frac{n-e}{e})^2, \quad \text{all } e \tag{4}$$

Note that the optimization problem requires Δ to be known, which can be achieved by running the estimator for all considered error ratios and using the frequencies of events in the trace to approximate the actual probabilities. The objective function is basically the total relative squared error. Constraints 1 and 2 simply state that the redistribution for any initial estimate value must be a legitimate PDF. Constraint 3 is basically to limit the estimation bias to a small value by controlling γ. Constraint 4 is basically to limit relative squared error to be no less than that before the redistribution. It should also be mentioned that this formalization optimizes the total relative squared error, however, it is equally possible to optimize for other metrics as those defined in Section **??**.

In practice, the number of variables and constraint can be very large which may pose challenge to the LP solver. In such cases, the formulation can be modified to reduce the problem size. To reduce the number of variables, instead of allowing the initial estimate to be redirected to any value, i.e., introducing a variable for each element in a column of Ω, it can be mandated that the initial estimate can be redirected to a smaller number of values. In our current implementation, in addition to itself, an initial estimate of $\frac{n}{M}$ can be redirected to 100 values, half of which are smaller than itself and the other half larger, where the values of the smaller half, multiplied by M for notational simplicity, are evenly distributed in $[\max\{0, n-\max\{\frac{n}{2}, 50\}\}, n-1]$ and the larger half in $[n+1, \min\{T, n+\max\{\frac{n}{2}, 50\}\}]$. To reduce the number of constraints, Constraints 3 and 4 can be applied only to a smaller number of error ratios, which in our current implementation includes error ratios from 0 to $\frac{36}{M}$ at a step of $\frac{1}{M}$, $\frac{37}{M}$ to $\frac{100}{M}$ at a step of $\frac{3}{M}$, and $\frac{101}{M}$ to $\frac{381}{M}$ at a step of $\frac{7}{M}$.

The redistribution step has only limited power in improving the performance in practice, because only very few, i.e., about 10% of the initial estimate values may be mapped to other values and are also usually mapped to only a few nearby values. Imposing more strict constraints on bias will not always improve the performance because the resulting LP problem may no longer be feasible.

V. Implementation and Complexity

mEEC can be implemented in with low complexity, because the most time-consuming steps can be pre-computed and stored in tables of reasonable sizes less than 20 MB in total.

A. Encoder

At the encoder, the random selection of bits with no repeat can be achieved by calculating a pseudo-random permutation based on a common random number generator as the decoder. If needed, the complexity can be further reduced by approximating the real-time pseudo-random permutation, i.e., pre-computing a set of pseudo-random permutations for typical packet sizes and applying the permutation with the closest size smaller than the size of the packet. Calculating the features clearly just needs a linear scan of the sampled data bits and counting. Finding the color of a super-block can be completed in constant time by storing a table which directly maps the feature values of the super-block to its color, which is affordable because the number of points in our implementation, i.e., the number of entries of the table, is $81^3 = 531,441$, and the total table size is less than 1 MB as each color is 12 bits.

B. Decoder

At the decoder, the permutation and feature calculation are exactly the same as the encoder. For each super-block, finding the list of points with the same color can also be completed in constant time with the help of a table, which has an entry for each color storing the list of points of this color. Note that the color assignment table at the decoder is different from that at the encoder. In our current implementation, which uses 7 bits to represent a feature value, the size of this table is less than 2 MB as each point is stored exactly once and there are 81^3 points. Determining whether or not a point should be added to the candidate list can be made very simple, e.g., involving only subtractions and additions of the differences in each dimension as in our current implementation.

The most time consuming task in decoding is to search for the initial estimate, which is dominated by the evaluation of the likelihood values of multiple error ratios. An obvious step to

take to reduce the complexity is to replace the multiplications in the likelihood calculations with additions of the log of the multiplicands. The number of evaluations has also been significantly reduced by the Newton search, which usually exits after a small number of rounds, as Fig. **??** in Section **??** shows. Therefore, the complexity of finding the initial estimate is in most cases a reasonably small number, e.g., 50, times the complexity of evaluating the likelihood of a single error ratio. The complexity of the latter, measured by the number of additions, is $O(NL)$, where L represents the size of the candidate list and is around 10 in our current implementation, which is basically calculating $\log[W(\hat{x}_i)] + \log[T_\theta(\hat{x}_i \to y_i)]$ for every block in the candidate list of every super-block. The log of both $W()$ and $T_\theta()$ can be pre-computed and stored in tables. The table for $W()$ has only $\frac{W}{2}+1$ entries. For $T_\theta()$, there are actually T tables, one for each error ratio value, and each table is a square matrix with $\frac{blkb}{2}+1$ rows. In our current implementation, $T = 588$ and $\frac{W}{2}+1 = 81$, making the total number of entries 3,864,429; however, as the number of entries in all tables greater than 10^{-9} is less than 43% and the non-zero entries in each row of the matrix are consecutive, it will require less than 16 MB if each entry is stored as double precision in 8 bytes.

Finally, the redistribution step is very simple in real time, because the redistribution matrix is recomputed, and therefore it only requires generating a random number according to one column in the redistribution matrix. Storing the redistribution matrix is also not a challenge because the size of the matrix is not large and the matrix is very sparse. In our current implementation, the matrix is 589×589 with over 98% of entries less than 10^{-9}.

VI. Evaluation

We evaluate mEEC with extensive simulations and compare it with gEEC [4].

A. Simulation Setup

In the simulations, when evaluating packet size U and error ratio θ, the number of corrupted bits is the closet integer to $U\theta$. As both gEEC and mEEC randomly sample data bits, the error pattern is irrelevant to the estimators. For similar reasons, the content of the data packet is also irrelevant, as both gEEC and mEEC randomize the data with random bit strings. As mentioned earlier, as we focus on the case where the EEC bits are not corrupted, no errors are introduced to the gEEC and mEEC code bits. For gEEC, to obtain the estimate, a linear scan is conducted on the error ratios, capped at 0.175, same as for mEEC. Due to the limit of space, the results for packet with 12000 and 4000 bits are shown, which represent typical data packet sizes in the Internet. The evaluated error ratios are 0.001 to 0.01 at a step of 0.001, followed by 0.02 to 0.15 at a step of 0.01. Each data point is obtained by 10,000 rounds of simulations.

Denote the estimated error ratio as $\hat{\theta}$, the performance of an estimator is measured by 4 metrics as in [4], namely:

1) The relative Mean Squared Error (rMSE): $E[(\frac{\hat{\theta}-\theta}{\theta})^2]$,
2) The relative Mean Squared log Error (rMSlogE): $E\{[\frac{\log\hat{\theta}-\log\theta}{\log\theta}]^2\}$,
3) Large deviation probability (LEProb): the probability of $\hat{\theta} > 2\theta$ or $\hat{\theta} < \frac{\theta}{2}$,
4) Mean bias: $E[\frac{\hat{\theta}-\theta}{\theta}]$.

Often times, an estimator may achieve better performance for some metrics, such as the rMSE, by sacrificing the mean bias metric. In practice, however, an unbiased estimator is usually preferred, i.e., an estimator with a small rMSE but a large bias may not be better than another with comparable rMSE but smaller bias. Therefore, for a fair comparison, a limit on the mean bias must be introduced, which is chosen to be within $\pm 3\%$; mean bias out of this range is considered large and the values of other metrics must be taken with a grain of salt. In figures of all metrics other than the mean bias metric, the curves for an estimator are shown in wider lines if the mean bias is not large. The $\pm 3\%$ lines are also explicitly shown in the mean bias figures.

gEEC has two key parameters, namely the sample size S and sampled bits L. gEEC also uses the Jeffrey's prior distribution by default, which usually improves the performance but also may introduce large bias in certain cases. For a fair comparison, a total of 16 configurations for gEEC are tested. To be more specific, in configurations 1 to 8, S and L are $[4, 128]$, $[4, 256]$, $[5, 128]$, $[5, 256]$, $[5, 512]$, $[6, 128]$, $[6, 256]$, $[6, 512]$ and the Jeffrey's prior is not used; configurations 9 to 16 are identical to configurations 1 to 8 except using the Jeffrey's prior. gEEC is allowed the same or even more code length than mEEC, i.e., gEEC uses 84 bits when $S = 4$ or 6 and 85 bits when $S = 5$.

B. Performance Comparison with gEEC

Fig. 4 and Fig. 5 show the performances of gEEC and mEEC for 4000-bit and 12000-bit packets, respectively. The gEEC configuration is $S = 5$, $L = 512$, and using the Jeffrey's prior, which is the best among all gEEC configurations and is the same as the one suggested in [4] for code length around 80 bits. The following main observations can be made:

- *mEEC achieves significantly lower rMSE, rMSlogE, and LEProb than gEEC for both packet sizes and almost all error ratios when both estimators are not biased*, i.e., when the error ratios are in $[0.004, 0.09]$ and $[0.005, 0.09]$ for 4000-bit packets and 12000-bit packets, respectively. For example, for 4000-bit packets, mEEC has much lower rMSE than gEEC, such as 0.0787 v.s. 0.1374, i.e., a 43% gain when the error ratio is 0.005. For 12000-bit packets, when the error ratio is in $[0.001, 0.003]$, mEEC has larger rMSE, rMSlogE, and LEProb than gEEC, however this is because mEEC is usually not biased while gEEC can be very heavily biased in this range.
- *mEEC is more unbiased than gEEC*. For 4000-bit packets, the mean bias of mEEC is within $\pm 2\%$ for error ratios in $[0.002, 0.09]$; for 12000-bit packets, the mean bias is within $\pm 3\%$ for error ratios in $[0.001, 0.09]$. Even for error ratio 0.1, the bias of mEEC is around -3.5% both

packet sizes and is very close to the threshold we have chosen. gEEC, on the other hand, has bias outside $\pm 3\%$ when the error ratios are in $[0.001, 0.03]$ and $[0.11, 0.15]$ for both 4000-bit and 12000-bit packets.

Additional observations include:

- The gain of mEEC over gEEC is large for smaller packets because mEEC estimates the error ratios in its sampled bits, and the smaller the packet size, the closer the error ratio is to the actual error ratio in the entire packet.
- For the same packet size, mEEC appears to have better performance for lower error ratios than in larger error ratios. For smaller packet such as 4000-bit packets, it is mainly because smaller error ratios result in smaller differences between the error ratios seen in the sampled bit and the overall error ratio and the entire packet. In addition, for both packet sizes, the redistribution step mainly improves the performance for lower error ratios.
- For 12000-bit packets, when the error ratios are in $[0.001, 0.003]$, mEEC is still unbiased but has much lager estimation errors than error ratios in other ranges, mainly because the error ratios seen in the sampled data can differ much more significantly from the actual error ratio in the packet when the error ratio is very small but the packet is large.
- The mean bias becomes much worse when the error ratio increases from around 0.11 to larger values for both estimators, because the error ratios are assumed to be no more than a certain value, 0.175 in our simulations for both estimators, which is a prior knowledge that can help reducing the estimation errors but also prevents the estimators from taking large estimates to balance out the small estimates. For this reason, the performance for error ratios in $[0.11, 0.15]$ is not a good indicator of the actual performance of the estimator.

To more qualitatively measure the gain of mEEC over gEEC, we define the *gain* of a metric at an error ratio as $\frac{A-B}{A}$, where A and B are the metric values of mEEC and gEEC, respectively, which applies to all metrics except the Mean Bias metric. The *average gain* is defined as the average of the gain on all evaluated error ratios when both mEEC and gEEC are not biased. As the number of evaluated error ratios are the same in $[0.001, 0.01]$ and $[0.01, 0.1]$, the average gain gives equal weight to the error ratios in the two ranges of different order or magnitudes. It should also be mentioned that considering only the error ratios for which both mEEC and gEEC are unbiased often underestimates the gain, because this may remove additional problematic data points for gEEC. That is, in almost all such removed data points, if mEEC is biased, so does gEEC; for removed data points that only gEEC is biased, such as error ratio 0.004 for the rMSE metric in Fig. 5, the gain is usually larger than the average gain. Fig. 6 shows the average gains for both packet sizes and for all gEEC configurations that are not excessively biased, where an estimator is considered excessively biased if the bias of 6 or more evaluated errors within $[0.001, 0.1]$ are

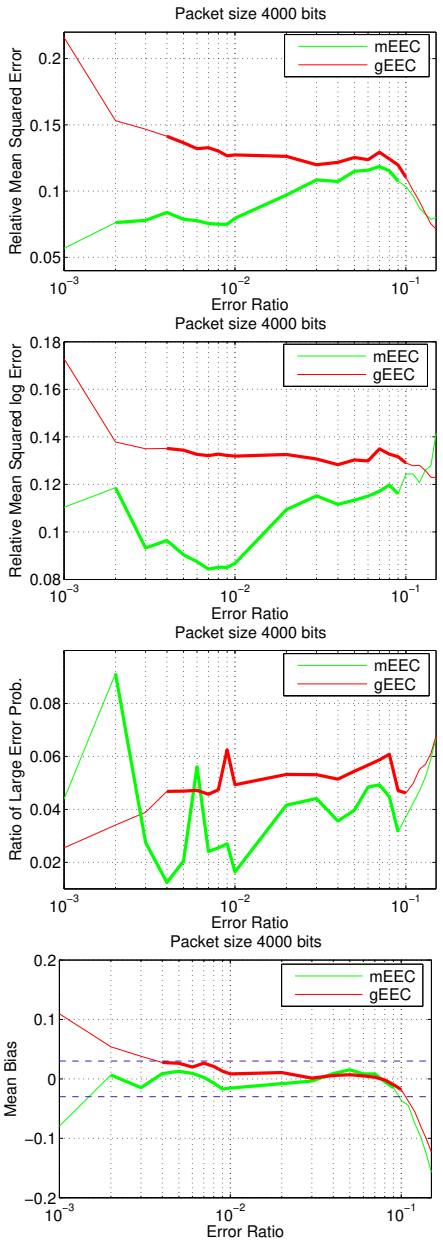

Fig. 4. Performance comparison for 4000-bit packets.

outside $\pm 3\%$. It can be seen that mEEC achieves about more than 20% gain for 4000-bit packets and more than 10% gain for 12000-bit packets for the selected gEEC configuration. It can also be seen that the mEEC usually achieves larger gains over other configurations of gEEC, except for 12000-bit packets with gEEC configuration 1, which is not eventually chosen because its performance is more unbalanced than the selected configuration, i.e., can be very poor for error ratios in $[0.001, 0.01]$, while many data points in this range are not chosen when calculating the average gain due to high bias.

C. Redistribution of mEEC

Fig. 7 shows the performance of mEEC with or without the Redistribution step, the latter denoted as *mEECwoRed*. Due

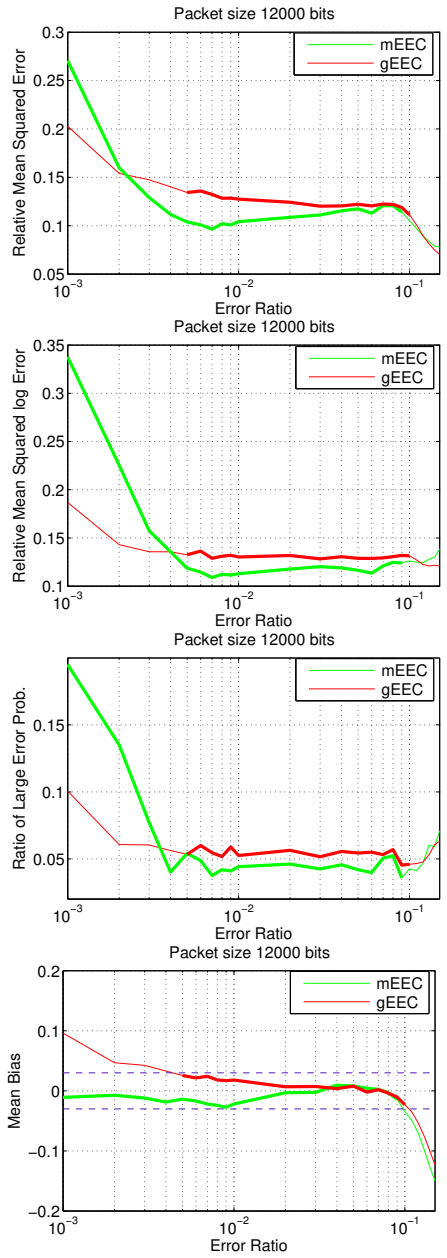

Fig. 5. Performance comparison for 12000-bit packets

to the limit of space, only the rMSE and MnBias metrics are shown, as the optimization problem currently optimizes the rMSE metric under constraints of the bias. It can be seen that the Redistribution step indeed improves the performance as well as reining in the bias to a smaller range.

D. The Newton Search of mEEC

The Newton search determines the complexity of mEEC. Fig. 8(a) shows the average number of rounds of Newton search as a function of the error ratio, where it can be seen that the number of rounds is usually very small, capped at around 13 for the tested error ratios. It is monotonically increasing because the search starts at the lowest error ratio and need more steps to reach the maximum point if the actual error

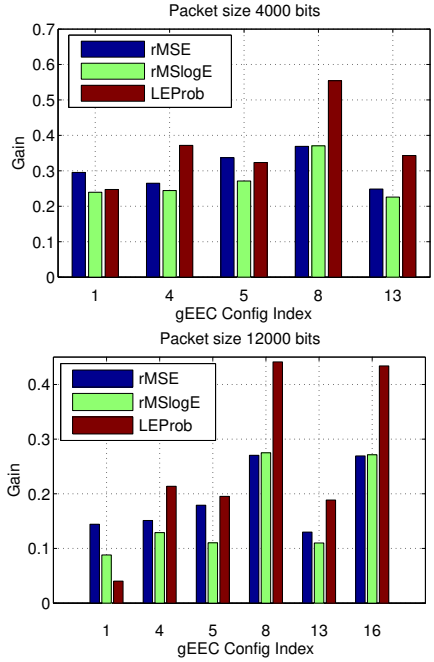

Fig. 6. Gain of mEEC over gEEC.

ratio is larger. Fig. 8(b) shows the fraction of cases mEEC has to do a linear scan of the error ratio, where it can be seen that the fraction is very small across the board.

VII. Conclusions

In this paper, we introduce a new error estimation code, mEEC. Compared to the state of the art code, gEEC, mEEC achieves smaller bias as well as over 10% to 20% gain in various metrics of estimation errors for packets of various sizes, when the error estimation code bits are not corrupted. mEEC is mainly based on the idea of group multiple blocks into a super-block, and using a single number, called the color, as a succinct representation of the features of all blocks. The representation is obtained by regarding the block features as coordinates of points in the multi-dimensional space, and finding a color assignment that results in larger distances between points of the same color. mEEC also adopts a lightweight additional step, called redistribution, which redirects the initial estimate based on the maximum likelihood estimation to other usually close values, guided by the solution to a linear programming problem which minimizes estimation error under constraints on estimation bias. mEEC is also easy to implement with low run time complexity, as most of its time-consuming tasks can be pre-computed and stored in tables of reasonable sizes.

References

[1] B.Chen, Z. Zhou, Y. Zhao, and H. Yu. Efficient error estimating coding: feasibility and applications. In *ACM Sigcomm*, 2010.

[2] W. Dong, J. Yu, and P. Zhang. Exploiting error estimating codes for packet length adaptation in low-power wireless networks. *IEEE Trans. Mob. Comput.*, 14(8):1601–1614, 2015.

[3] N. Hua, A. Lall, B. Li, and J. Xu. A simpler and better design of error estimating coding. In *IEEE Infocom*, pages 235–243, 2012.

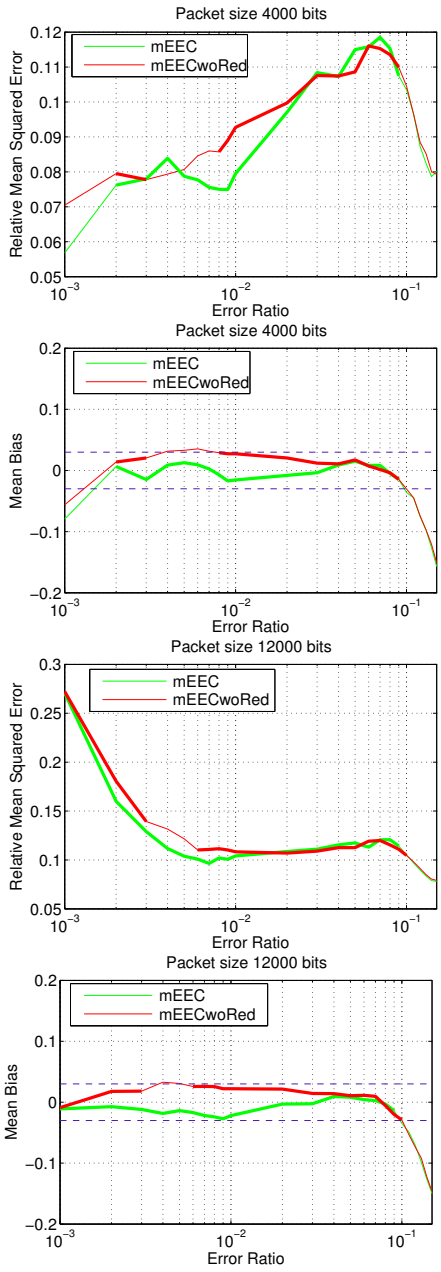

Fig. 7. Gain due to Redistribution.

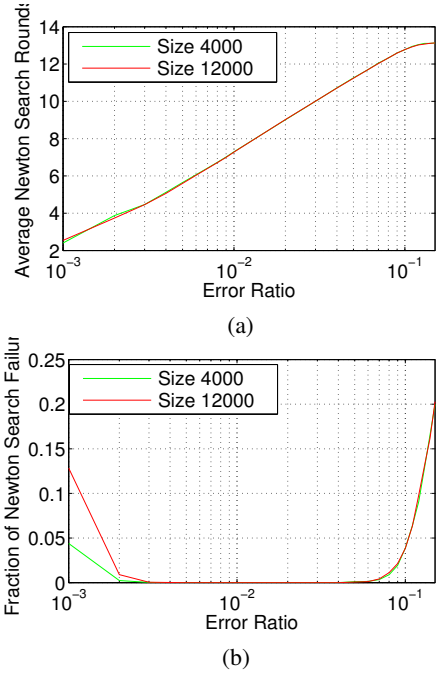

Fig. 8. (a). Average number of rounds of Newton search. (b). Fraction of cases need the linear scan.

[4] N. Hua, A. Lall, B. Li, and J. Xu. Towards optimal error-estimating codes through the lens of Fisher information analysis. In *ACM SIGMETRICS*, pages 125–126, 2012.

[5] J. Huang, S. Yang, A. Lall, J. K. Romberg, J. Xu, and C. Lin. Error estimating codes for insertion and deletion channels. In *ACM SIGMETRICS*, pages 381–393, 2014.

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

[7] K. C.-J. Lin, N. Kushman, and D. Katabi. Ziptx: Harnessing partial packets in 802.11 networks. In *ACM MobiCom*, pages 351–362, 2008.

[8] J. Lu, B. Han, W. Yang, Y. Gao, and W. Dou. Smart error estimating coding: Using symbol error structure in wireless networks. In *IEEE ICCCN*, 2014.

[9] J. Lu, W. Yang, B. Li, J. Wang, and W. Dou. CBEEC: A content-based error estimation coding framework. In *IEEE PDCAT*, pages 621–626, 2012.

[10] J. Ou, Y. Zheng, and M. Li. MISC: Merging incorrect symbols using constellation diversity for 802.11 retransmission. In *IEEE Infocom*, pages 2472–2480, 2014.

[11] Y. Wang, W. Li, and S. Lu. The capability of error correction for burst-noise channels using error estimating code. In *IEEE SECON*, 2016.

[12] X. Wei, W. Li, X. Wang, S. Lu, and X. Fu. Recovering erroneous data bits using error estimating code. In *ISCC*, pages 705–710, 2013.

[13] J. Xie, W. Hu, and Z. Zhang. Efficient software partial packet recovery in 802.11 wireless LANs. *IEEE Transactions on Computers*, 63(10):2402–2415, 2014.

[14] H. Yao and T. Ho. Error estimating codes with constant overhead: A random walk approach. In *IEEE ICC*, 2011.

[15] Z. Zhang, W. Hu, and J. Xie. Employing coded relay in multi-hop wireless networks. In *IEEE Globecom*, Dec. 2012.

APPENDIX

The coloring problem can be formally defined as: given C colors and integer points in the 3-D space, i.e., points with integer coordinates, assign a color to each point, such that the minimum Euclidian distance between any points of the same color, denoted as D, is maximized. A quick argument can be given to establish an upper bound of D at $2(\frac{3C}{4\pi})^{\frac{1}{3}}$. Consider a sphere in the space with radius $(\frac{3C}{4\pi})^{\frac{1}{3}}$. The number of points in this sphere should be more than C, and therefore at least one color has been repeated. The distance between any 2 points of this color in this sphere cannot be more than twice the radius, which is the bound.