Error Floors of LDPC Codes

Tom Richardson

Flarion Technologies
Bedminster, NJ 07921
tjr@flarion.com

Abstract

We introduce a computational technique that accurately predicts performance for a given LDPC code in the error floor region. We present some results obtained by applying the technique and describe certain aspects of it.

1 Introduction

Performance curves of iterative coding schemes such as LDPC codes and turbo codes are well-known as "waterfall" curves. Sometimes, one observes the bottom of the waterfall, often referred to as the error floor. In the case of parallel concatenated turbo codes the error floor can be paritally attributed to low-weight codewords in the code [1]. Often, the error floor is not observed because it is out of reach of the simulation performed. Recent simulation of LDPC codes performed on high speed hardware platforms indicate that LDPC codes, even regular ones like (3,x) and (5,x) Gallager codes, do exhibit error floors. Unlike parallel concatenated turbo codes, however, the error floor does not typically arise from low-weight codewords.

In a recent paper [2], MacKay and Postol discovered in simulation a "weakness" in the Margulis construction of a regular (3,6) Gallager code that gives rise to a relatively high error floor. We have reproduced their simulation results using a slightly different decoder (5 bit approximation to belief propagation) and plotted the results in Fig. 1. The source of the problem is identified in [2] as certain "near codewords" in the graph. These are small sets in the graph, consisting of 12 or 14 variable nodes such that the induced sub-graph has a small number (4) of odd (1) degree neighbors. In fact, using techniques discussed in this paper, we find 1320 distinct sets of each type, all equivalent under the automorphism of the graph. At $E_b/N_0 = 2.4dB, \sigma = 0.725$ and using a 5 bit decoder each $(12, 4)$ subset gives rise to a failure with probability about 6.5×10^{-10}. Consequently, the frame error rate of the code is lower bounded by $1320 \times (6.5 \times 10^{-10}) = 8.6 \times 10^{-7}$. By our estimates, this accounts for about 75% of the error floor performance, and 1320 $(14, 4)$ "near codewords" account for another 23%. An interesting question at this point is how low an FER error floor can one achieve with a code of these parameters. Our investigations indicate that the answer is about 10^{-10}, see Fig. 1.

In the case of the binary erasure channel (BEC), the error floor region of LDPC code performance is reasonably well understood [3]. This understanding grew out of an elementary observation: Decoding failure of LDPC codes over the binary erasure channel has a purely combinatorial characterization. Certain subsets of variable nodes in the Tanner graph of the LDPC code, called *stopping sets* [3], are precisely those subsets that

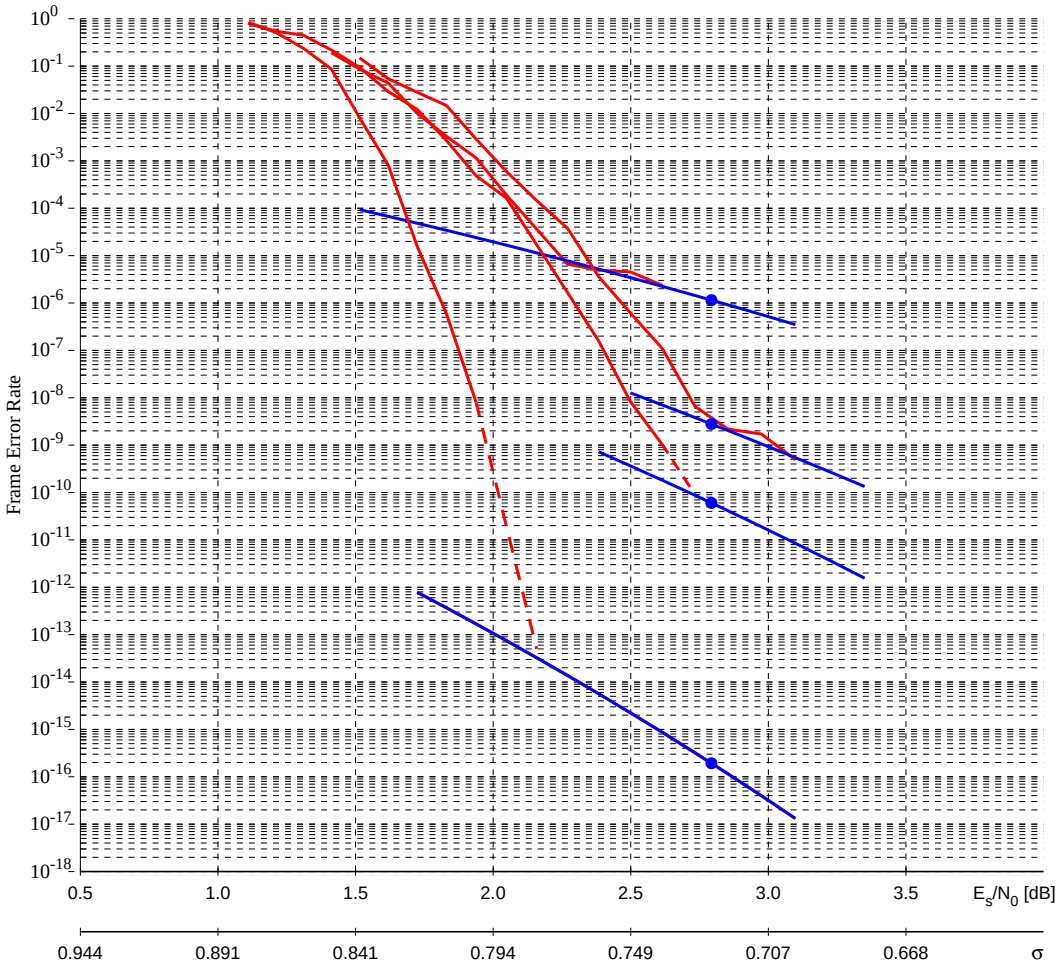

Figure 1: Simulation and error floor predictions for some regular (3,6) LDPC codes using a 5-bit decoder. The codes in order from highest to lowest error floor are the Margulis graph (n=2640), an n=2048 code, an n=2640 code (same as the Margulis graph), and an n=8096 code. The dashed curves are extrapolations. Except for the Margulis code, code simulations were performed on an FPGA platform. Error floor predictions are computed on a PC.

can constitute final undetermined variable nodes under iterative decoding. Since stopping sets have a combinatorial characterization, their distribution within ensembles of graphs can be analyzed, although with some difficulty. Given a BEC with erasure probability ϵ and given the collection of stopping sets in a Tanner graph, the performance of the code over the BEC is completely determined. In the error floor region, performance is dominated by the small stopping sets in the graph.

We extend the insight gained in the analysis of the BEC to other memoryless channels such as the additive white Gaussian noise channel (AWGNC) and the binary symmetric channel (BSC). The approach taken in this work is slightly different from that taken for the BEC in [3]. Rather than attempting to analyze performance of ensembles, we develop a methodology for predicting the error floor behavior of a given graph. From a practical standpoint, this approach is more desireable. Applications requiring very deep error floors can now be seriously addressed. Although we do not, and probably cannot,

obtain a simple closed combinatorial characterization of decoding failure as for the BEC, certain combinatorial aspects still play a key role. (This aspect will not be developed in this short version of the paper.)

The main result in this paper is not an analytical prediction of error floors but rather a computational technique that allows prediction of error floors, with very good accuracy, in regimes far beyond the reach of simulation. The accuracy of the technique is verified in regimes that can be reached by simulation. For this reason we have focussed on a decoding algorithm that is available as a hardware implementation. Permitting simulation of error floors for frame error rates down to about 10^{-9}. Furthermore, all codes other than the Margulis code are quasi-cyclic LDPC codes with cycles of length 64, the class of codes supported by a particular version of the above-mentioned hardware implementation.

Fig. 1 presents some results for (3,6) graphs. The error floor prediction curves were computed on a PC. From these examples one can see both the accuracy of the technique and the feasibility of predicting deep error floors. Many other examples including irregular graphs and other rates have also verified the accuracy of the technique.

2 Some Experimental Results

The starting point of much of the work in this paper was a series of experiments. In this section we review one set of such experiments and indicate the observations made.

The Tanner graphs simulated were all cyclic "liftings" [4] of size 64 of a "projected" graph. The projected graphs each have 64 degree 3 variable nodes, 3 degree 14 check nodes and 10 degree 15 check nodes. Thus, the resulting codes are nearly regular quasi-cyclic codes of length 4096 with code rate $51/64 = 0.7969$.

We have simulated examples drawn from three different ensembles or, more precisely, three different levels of expurgation from the random ensemble. The first ensemble consists of random parameters except graphs with multiple edges between a single variable and check node (multiple edges in the projected graph are allowed) are excluded. The second ensemble has been optimized with respect to girth. An optimization on the edge lifting is run to achieve girth 8 (the largest possible) but no further optimization is attempted. The third ensemble has been further "neighborhood" optimized to reduce certain multiplicities of loops of size 8 through the edges. (Details of the method will be given in the full paper. See [5] and [6] for relevant and related methods.) Frame error rate curves are shown in fig. 2.

The hardware platform on which these experiments were performed returned only the final state of the decoding, so oscillations are not identified as such. Nevertheless, the results are quite revealing. In the error floor region virtually all failures are due to "near codewords." Later, we will call them trapping sets. These are sets with a relatively small number of variable nodes such that the induced sub-graph has only a small number of odd degree check nodes. Invariably in these experiments, the sub-graph contains only degree 2 and degree 1 check nodes. This can be understood as a combinatorial bias, especially for optimized graphs. Larger check node degrees are possible but are unlikely in the smallest (dominant) trapping sets. This is especially true for girth optimized graphs. The clustering observed in the performance curves for random graphs is generally attributable to the worst case trapping set. The worst performing graphs have trapping sets of the form (3,1), 3 variable nodes with an induced subgraph having one degree 1 check node. The next cluster are those graphs with (5,1) trapping sets, etc. Note that the group structure of the graphs exagerates the differences between graphs since each

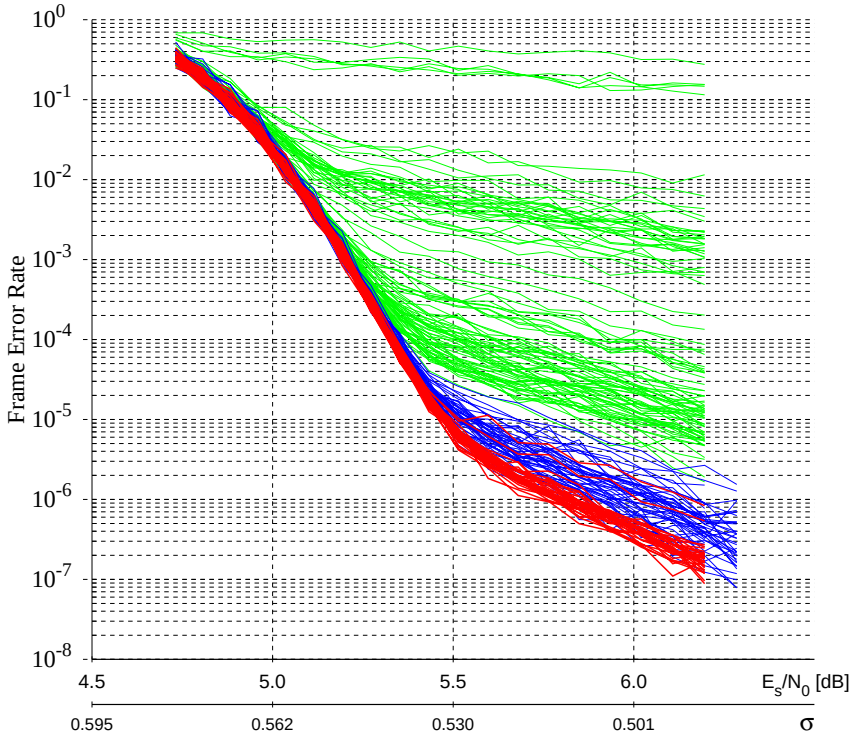

Figure 2: Frame error rates for random graphs (green), girth optimized (blue), and neighborhood optimized (red) graphs all of identical degree structure (nearly regular (3,15)).

sub-structure occurs with multiplicity 64.

It is interesting to note that the girth optimized graphs still show significant variation in their error floor performance and the neighborhood optimized graphs display better performance on average and smaller variation. Even in this set of graphs, however, there are some relatively poor performing outliers. This shows that our optimization still has its limitations. Even if the optimization always gave a best performing graph, it is still desirable to be able to predict that performance, especially when it is not within reach of simulation.

3 Preliminaries

To study failure of iterative decoding of LDPC codes we need to formalize failure to some extent. Therefore, we introduce a definition of the set of variable nodes on which decoding fails.

3.1 Defining Failure

Let $\mathcal{Y}$ denote the set of all possible inputs to the decoder. We assume a binary code associated to a Tanner graph with N variable nodes. An iterative decoder is defined to be a sequence of maps $\mathsf{D}^{\ell}(\mathsf{y}) : \mathcal{Y} \to \{0,1\}^N, \quad \ell = 0, 1, 2, 3, 4, ...$ We shall assume a symmetric channel and decoder so we can assume that the all 0 codeword was transmitted. (For the BEC let 1 denote an erasure.) Let $\mathsf{D}^{\ell}(\mathsf{y})[i]$ denote the ith output bit. The index ℓ

indicates the iteration number. We say that bit i is *eventually correct* if there exists I such that, for all $i \geq I$, $\mathsf{D}^{\ell}(\mathsf{y})[i] = 0$. A decoding is *successful* if all bits are eventually correct. For a given input y we define the *failure set* $\mathsf{T}(\mathsf{y})$ to be the set of bits that are *not* eventually correct. Thus, decoding is successful on y if and only if $\mathsf{T}(\mathsf{y}) = \emptyset$. If $\mathsf{T}(\mathsf{y}) \neq \emptyset$ then we say that $\mathsf{T}(\mathsf{y})$ is a *trapping set*. We say T is a (a, b) trapping set if it has a variable nodes and b odd degree check nodes neighbors in the sub-graph induced by T.

In practice, under simulation, see Sect. 4, we identify the trapping set in the following way. Some large fixed number of iterations (say 200) is performed unless the decoder converges to a codeword earlier. If it has not converged after the fixed number of iterations then we do some further iterations (20 say) and identify the trapping set as the union of all bits which do not decode correctly during those 20 iteration.

Note that the trapping set definition depends on the decoder input space and the decoding algorithm. We could, for example, consider a maximum likelihood decoder which decodes to the most likely codeword in one iteration. In this case the set of trapping sets is precisely the set of non-zero codewords.

Fact 1. *If the decoder is the one-step maximum likelihood decoder then the trapping sets are precisely the non-zero codewords.*

Of course, the cases of interest are iterative decoders.

Fact 2. *If the channel is the BEC and the decoder is belief propagation then the trapping sets are precisely the stopping sets.*

There are other cases in which trapping sets have entirely combinatorial characterizations. Consider a regular (3,6) (Gallager) LDPC graph (with no multiple edges) used over the BSC with a serial flipping algorithm in which bits are flipped, at most one per iteration, if the number of neighboring unsatisfied constraints for that bit is 2 or 3.

Fact 3. *For serial flipping on a (3,6) graph subset is a trapping set if and only if each node in the set has at most one induced odd degree neighbor check node in the subgraph induced by the trapping set and no other variable node in the graph neighbors more than one of the odd degree check nodes.*

(To prove this first note that the number of unsatisfied check nodes is monotonically decreasing under serial flipping.)

Fact 4. *Serial flipping trapping sets are trapping sets for the Gallager A (or B) algorithm.*

We cannot presently say that these are all the trapping sets because Gallager A and B may produce oscillations on other types of sets.

The decoding algorithm of greatest interest is belief propagation and its approximations. The results in this paper are obtained for a finite (5 bit) approximation of belief propagation. Error floor behavior of coding systems can depend strongly on the form of quantization used, see [7] for some examples. To illustrate the point, consider again a regular (3,6) graph and suppose the decoder is belief propagation (with messages represented as log-likelihoods) except that messages are saturated to lie in the interval $[-K, K]$. If the input not saturated, then a single node can be trapping set. If the input is saturated, to $[-K, K]$ say, then a single node (most likely) will not be a trapping set. In any case, the question of exactly what can and what cannot be a trapping set is not the critical question. The goal of the work is to understand the error floor region by identifying the most *relevant* trapping sets.

3.2 Decomposition of Failure

Let $\xi_{\mathsf{T}} \subset \mathcal{Y}$ denote the set of inputs that give rise to a failure on a trapping set T. Then,

$$\text{FER} = \sum_{\mathsf{T}} \Pr\{\xi_{\mathsf{T}}\}.$$

Of course, $\Pr\{\xi_{\mathsf{T}}\}$ depends on the channel. Typically, we consider a parameterized family of channels ordered by physical degradation and we write $\Pr\{\xi_{\mathsf{T}} : \sigma\}$. Here, σ is the channel parameter and increasing σ degrades the channel. For a maximum likelihood decoder the FER is dominated by low-weight codewords when σ is small. This implies that the number of terms that dominate FER is relatively small in this region. Something similar happens under iterative decoding of LDPC codes in the error floor region. Relatively small trapping sets in the graph give rise to failure events that dominate performance in the error floor region. The size of the trapping set is not the only relevant parameter. Even for fixed b, it sometimes happens that (a, b) trapping sets fail at a higher rate than (a', b) trapping sets with $a > a'$.

We prefer the name trapping sets over "near codewords" because failure is a decoder dependant local dynamic effect in which the state of the set gets trapped on the incorrect values even though unsatisfied constraints remain (except in the case of a low-weight codeword). If the number of odd degree neighbors is sufficiently large then, for reasonable decoders, one can show that the incorrect decoding is unstable provided the rest of the graph converges sufficiently well. Because the trapping set structures involved in the error floor are relatively small, and because their cardinality is not too large, it is feasible to discover, enumerate, and evaluate them.

3.3 A Two-Stage Attack

The technique that this paper introduces has two phases. The first phase is to search the Tanner graph for candidate trapping sets. The candidates are drawn largely from combinatorial considerations. These considerations are guided partly by heuristics and partly by consideration of various decoders, flipping decoders in particular. (We will not develop this stage of the technique in this short version of the paper.) The goal of this stage is to form a nearly complete list of candidates, $\mathsf{T}_1, \mathsf{T}_2, \ldots, \mathsf{T}_k$, covering the error floor region. Graphs that possess an automorphism are advantageous since each candidate trapping set is a member of an equivalence class of trapping sets whose size is the order of the automorphism. We need only keep one representative from each equivalence class on the list. The Margulis graph has an automorphism of size 1320 and all other graphs in this paper have an automorphism of size 64. Once a rich set of candidates is found we begin the second phase - evaluating their contribution to the error floor. Sometimes, during this evaluation, additional trapping sets may be discovered: they are added to the list, see Sect. 4.2. We then obtain, in effect, a lower bound $\text{FER} \geq \sum_{i=1}^{k} \Pr\{\xi_{\mathsf{T}_k}\}$, although $\Pr\{\xi_{\mathsf{T}_k}\}$ is generally not known exactly.

4 Trapping Set Evaluation

In this section we describe a technique for evaluating the error rate associated to a given trapping set $\Pr\{\xi_{\mathsf{T}_k}\}$ and a given channel and terative decoder. The main idea is to determine the error rate conditioned on some dominating aspect of the input to the trapping set. Due to space limitations we consider here only the AWGN channel.

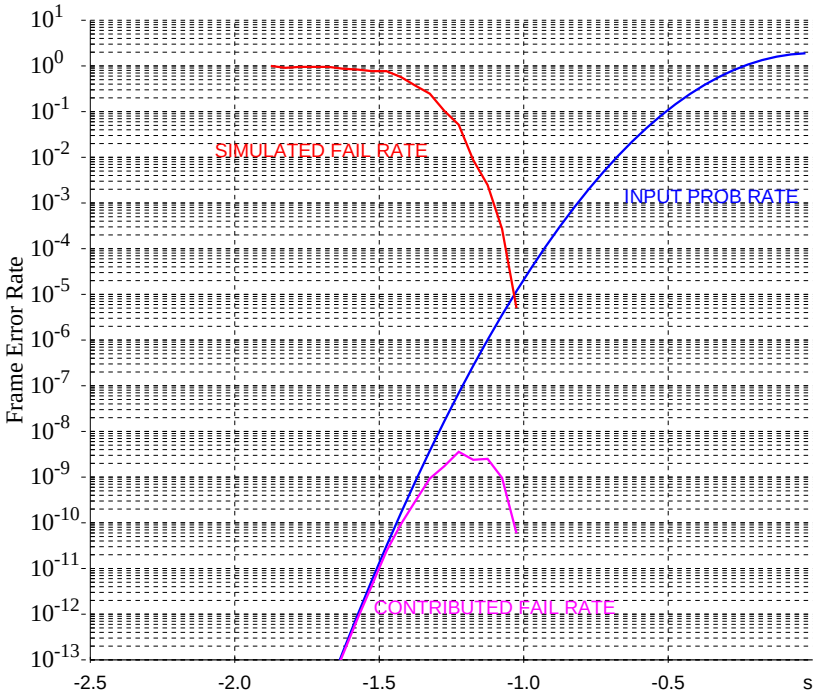

Figure 3: Summary of error floor evaluation of a (12,4) trapping set T in the Margulis graph. The horizontal axis indicates the value of the random variable $s = 12^{\frac{1}{2}}\gamma$. The curve labelled "Simulated Fail Rate" indicates the probability of an otherwise random input failing on the given trapping set as a function of s. The curve labelled "Input Prob Rate" indicates the probability density of s. The curve labelled "Contributed Fail Rate" gives the product (sum in the log-domain) of these two curves. It indicates the failure rate on T as a function of s. The total failure rate on T is given as the integral of this function with respect to s. The result is about 6.5×10^{-10}.

4.1 The AWGN Case

Suppose the trapping set has K variable nodes (none of which are punctured). The input $y_1, ..., y_n$ to the decoder can be written $y_i = \frac{2}{\sigma^2}(1+n_i)$, $i = 1, \ldots, n$, where $\{n_i\}_i$ are i.i.d. $N(0, \sigma^2)$ random variables. Consider the subvector $(n_1, ..., n_K)$ of noise contributed to the trapping set. This is a Gaussian vector of with mean 0 and covariance matrix $\sigma^2 I_{K\times K}$. Now, consider an orthonormal basis for $\mathbb{R}^K$ with the first vector being $\frac{1}{K^{\frac{1}{2}}}(1, ..., 1)$. We write

$$(n_1, ..., n_K) = \gamma \frac{1}{K^{\frac{1}{2}}}(1, ..., 1) \; + \; \ldots \; = s(1, ..., 1) \; + \; \ldots$$

where γ is $N(0, \sigma^2)$ and s is therefore $N(0, K^{-1}\sigma^2)$. To evaluate the failure rate of the trapping set we consider conditioning on s. The hope is that the failure rate is strongly dependent on the value of s. This is reasonable since s determines the relative distance of the input to $(1, ..., 1)$ and $(-1, ..., -1)$. Thus, we express the failure rate on the given trapping set as

$$\Pr\{\xi_{\mathsf{T}}\} = \mathbb{E}_s \Pr\{\xi_{\mathsf{T}} |\, s\} = \frac{\sqrt{K}}{\sqrt{2\pi}\sigma} \int_{-\infty}^{\infty} \Pr\{\xi_{\mathsf{T}} |\, s = x\}\, e^{-\frac{x^2 K}{2\sigma^2}}\, dx$$

In practice, we attempt to determine the function $\Pr\{\xi_{\mathsf{T}}|\, s = x\}\left(\frac{\sqrt{K}}{\sqrt{2\pi}\sigma}\, e^{-\frac{x^2 K}{2\sigma^2}}\right)$ in a neighborhood of its maximum. The function is expected to decay quickly as s moves away from the maximum so that the integral is largely determined by the behavior near the maximum. The second factor is available analytically, the first factor $\Pr\{\xi_{\mathsf{T}}|\, s = x\}$ is estimated by *in situ* simulation on a PC. This is where the definition of failure on T becomes important: we count only failures that occur precisely on T. Typically, it is sufficient to simulate down to about $\Pr\{\xi_{\mathsf{T}}|\, s = x\} \simeq 10^{-3}$. This is what makes the technique efficient. Figure 3 displays the results for the (12,4) trapping set in the Margulis graph in the log domain.

4.2 Clustering of Trapping Sets

Very often, especially in highly optimized graphs, trapping sets occur in overlapping clusters. The Margulis graph, despite its weakness, is a case in point. The (14,4) trapping sets are each 2-node extensions of the (12,4) trapping sets. As a consequence of this overlap, one often observes failures on one trapping set while evaluating the other. This lends a certain robustness to the trapping set search: once a single trapping set from a tight cluster of trapping sets is identified, the other trapping sets are typically discovered during the evaluation stage. This reduces the burden of completeness of the initial search phase; we do not worry too much that every trapping set is found, we do hope, however, that every cluster is found.

5 A Test Case

Since the error floor of the Margulis graph is relatively easy to determine, we chose instead one graph from the neighborhood optimized ensemble discussed in Sect. 2 for detailed study. A total of 5000 error events were collected from the hardware simulation at $\sigma = 0.51$ and at $\sigma = 0.48$ respectively. Unfortunately, the hardware platform records only the final error state, so (relatively rare) oscillations are not properly detected. For the same two points, predictions of the error floor were computed on a PC. Partial results (the most dominant trapping sets) are shown in table 1 and the performance curve along with the error floor predictions are plotted in Fig. 5. The accuracy of the predictions is striking. In table 1 one can notice some failures with (3,9) and (4,10) trapping sets. These failures are likely due to 2 iteration oscillations on (7,3) trapping sets.

5.1 Extrapolating a curve from one data point

In all of the presented error floor predictions, the predicted point is indicated by a small circle that lies on a curve. We now explain how the curve is derived from the data used to generate the point marked by the circle.

The key point is that as σ varies locally, the factor $\Pr\{\xi_{\mathsf{T}}|\, s\}$ changes relatively little, see Fig 4. Most of the variation in the error floor point is due to the analytically determinable factor $\frac{\sqrt{K}}{\sqrt{2\pi}\sigma}\, e^{-\frac{s^2 K}{2\sigma^2}}$. Thus, from the data giving $\Pr\{\xi_{\mathsf{T}}|\, s\}$ for some particular σ, a curve can be easily extrapolated. The error floor curves predicted from $\sigma = 0.48$ and $\sigma = 0.51$ seen in Fig. 5 are almost identical.

In Fig. 6 we present error floor evaluation results for $(10, x)$ and $(12, x)$ trapping sets for $x = 0, 2$, and 4. What is interesting to note here is the consistency of the curves.

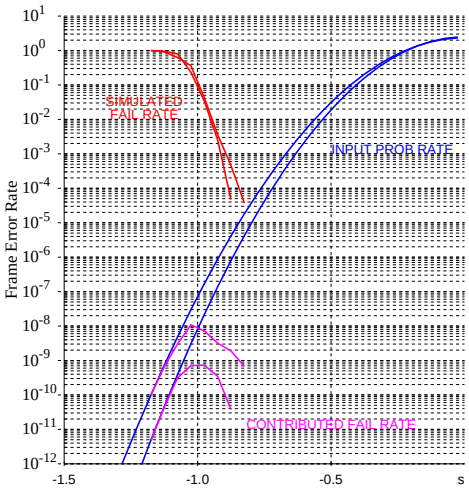

Figure 4: [Left plot.] The figure is similar Fig. 3. The data is for the test rate 0.8 graph for the (9,1) trapping set evaluated at 2 data points, $\sigma = 0.48$ and $\sigma = 0.51$. Notice that the simulation data changes relatively little.

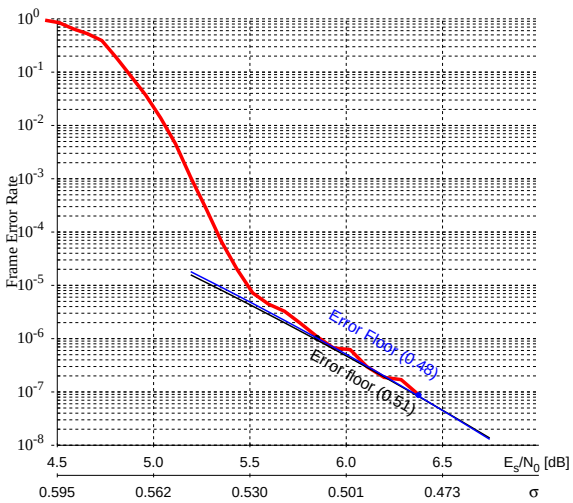

Figure 5: [Right plot.] Hardware code simulation and two PC error floor predictions for test case graph. Predictions performed at $\sigma = 0.48$ and $\sigma = 0.51$ and curves extrapolated from the data. We note that the two error floor curves are nearly identical.

Simulated 8.89e-08					Predicted 8.73e-08		
(a,b)	# failures	# classes	rel rate	fail rate	fail rate	rel rate	# classes
(8,2)	1767	9	0.38955	3.56828e-08	3.5008e-08	0.400916	10
(7,3)	816	14	0.17989	1.64783e-08	2.0399e-08	0.233614	171
(9,3)	505	397	0.11133	1.01979e-08	8.6179e-09	0.0986908	1609
(9,1)	431	1	0.09501	8.70361e-09	7.3155e-09	0.0837768	1
(10,0)	338	1	0.07451	6.82557e-09	5.0943e-09	0.0583401	1
(10,2)	262	72	0.05776	5.29082e-09	5.1576e-09	0.0590649	80
(11,3)	141	139	0.03108	2.84735e-09	2.5950e-09	0.0297177	12500⋆
(12,2)	49	48	0.01080	9.89506e-10	8.066e-10	0.00923716	611
(10,4)	36	36	0.00793	7.26984e-10	7.7422e-10	0.00886628	204000⋆
(11,1)	34	4	0.00749	6.86596e-10	7.1408e-10	0.00817754	4
(12,4)	28	28	0.00617	5.65432e-10	6.9905e-11	0.00080054	221500⋆
(3,9)	27	5	0.00595	5.45238e-10	*	*	*
(13,3)	20	20	0.00441	4.03880e-10	1.6945e-10	0.00194057	30800⋆
(8,4)	17	17	0.00374	3.43298e-10	3.3476e-10	0.00383372	18400⋆
(4,10)	13	3	0.00286	2.62522e-10	*	*	*
(14,4)	9	9	0.00198	1.81746e-10	*	*	*
(14,2)	7	7	0.00154	1.41358e-10	9.6280e-11	0.00110259	3200⋆
(15,3)	7	7	0.00154	1.41358e-10	1.1642e-12	1.333e-05	24600⋆
(16,4)	5	5	0.00110	1.00970e-10	*	*	*
(16,2)	3	3	0.00066	6.05820e-11	1.7349e-12	1.9868e-05	3300⋆
(12,0)	2	1	0.00044	4.03880e-11	1.0112e-10	0.00115802	1

⋆: the number of classes saved was limited by the allocated database size.
The reported number of classes is an estimate.

Table 1: Some simulated and predicted results for test graph at $\sigma = 0.48$.

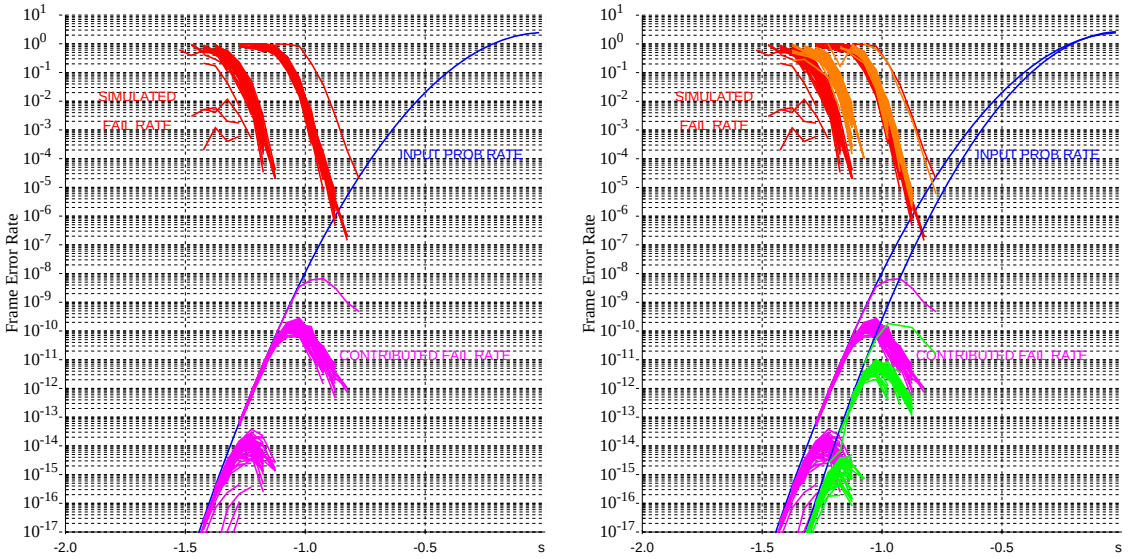

Figure 6: Evaluation results for trapping sets (10,0), (10,2), and (10,4), (on the left) and superimposed with (12,0), (12,2), and (12,4) (on the right).

Clearly, some simplified models are possible. The odd looking curves among the $(10, 4)$ sets are due to trapping sets that extend to a $(11, 3)$ trapping set. Such sets fail on the $(11, 3)$ set more than on the $(10, 4)$ set.

References

[1] S. Benedetto and G. Montorsi, "Unveiling turbo codes: some results on parallel concatenated coding schemes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 409–428, Mar. 1996.

[2] D. MacKay and M. Postol, "Weaknesses of margulis and ramanujan-margulis low-density parity-check codes," *Electronic Notes in Theoretical Computer Science*, vol. 74, 2003.

[3] C. Di, D. Proietti, E. Telatar, T. Richardson, and R. Urbanke, "Finite length analysis of low-density parity-check codes," *IEEE Trans. on Information Theory*, pp. 1570–1579, June 2002.

[4] T. Richardson, "Matched liftings of ldpc codes." in preparation.

[5] Y. Mao and A. H. Banihashemi, "A heuristic search for good low-density parity-check codes at short block lengths," in *Proc. IEEE Int. Conf. Commun.*, (Helsinki, Finland), June 2001.

[6] T. Tian, C. Jones, J. Villasenor, and R. D. Wesel, "Construction of irregular ldpc codes with low error floors," in *Proceedings IEEE International Conference on Communications*, 2003.

[7] J. Thorpe, "Low-complexity approximations to belief propagation for ldpc codes," in *Proceedings of IEEE ISIT*, (Yokohama, Japan), July 2003. Extended version available at http://www.ee.caltech.edu/∼jeremy/research/papers/research.html.