Error floor prediction with Markov models for QC-MDPC codes

Sarah Arpin[1], Jun Bo Lau[2], Antoine Mesnard[3], Ray Perlner[4], Angela Robinson[4], Jean-Pierre Tillich[3], and Valentin Vasseur [5]

[1] Virginia Polytechnic Institute and State University, Department of Mathematics
[2] KU Leuven, COSIC, Belgium
[3] INRIA
[4] National Institute of Standards and Technology, Computer Security Division
[5] Thales, Gennevilliers, France

Abstract. Quasi-cyclic moderate-density parity check (QC-MDPC) code-based encryption schemes under iterative decoders offer highly-competitive performance in quantum-resistant cryptography, but their IND-CCA2 security is an open question because the decoding failure rate (DFR) of these algorithms is not well-understood. The DFR decreases extremely rapidly as the blocklength increases, then decreases much more slowly in regimes known as the waterfall and error floor, respectively. The waterfall behavior is rather well predicted by a Markov model introduced by Sendrier and Vasseur [SV19b] but it does not capture the error floor behavior. Assessing precisely for which blocklength this error floor begins is crucial for the low DFRs sought the context of cryptography.
By enriching the Markov model [SV19b] with information about near codewords we are able to capture this error-floor behavior for a step-by-step decoder. This decoder displays worse decoding performance than the parallel decoders used in practice but is more amenable to a Markov chain analysis. We already capture the error floor with a simplified model. A refined model taking into account certain structural features of the secret key is even able to give accurate key dependent predictions both in the waterfall and error floor regimes. We show that error floor behavior is governed by convergence to a near codeword when decoding fails. We ran this model for the BIKE cryptosystem with this simpler step by step decoder to better ascertain whether the DFR is low enough to achieve IND-CCA2 security. Our model gives a DFR below $2^{-131.2}$, using a block length $r = 13477$ instead of the BIKE parameter $r = 12323$. This paper gives some strong evidence that the IND-CCA2 requirement can be met at the cost of a modest increase of less than 10% in the key size.

1 Introduction

1.1 Motivation

The NIST PQC standardization process. The U.S. National Institute of Standards and Technology (NIST) Post-Quantum Cryptography (PQC) standardization process, which began with 82 submissions, has selected four algorithms for standardization while three algorithms remain under consideration in the fourth round. One of the remaining candidates is BIKE, a cryptosystem based on quasi-cyclic moderate-density parity check (QC-MDPC) codes which are decoded by an iterative Black-Grey-Flip (BGF) decoder [ABB+21,DGK20]. The BIKE cryptosystem offers competitive performance but lacks a formal security claim in the static-key setting, unlike its code-based competitors.

Estimating the DFR of BIKE. Cryptosystems with a non-zero probability of decryption failures require careful analysis when proving IND-CCA2 security. In particular, the standard security proof [HHK17] requires the average failure probability to be below $2^{-\lambda}$ where λ is the security parameter. BIKE is one such cryptosystem, so meeting IND-CCA2 security requires the average Decoding Failure Rate (DFR) of the QC-MDPC code used to be below $2^{-\lambda}$. A low DFR

for BIKE is not just a theoretical concern, since a DFR sufficiently higher than $2^{-\lambda}$ enables the GJS key-recovery attack [GJS16], which exploits decoding failures in an IND-CCA security model.

The target DFR of $2^{-\lambda}$ is too small to measure directly, so another approach is needed to prove that the DFR of BIKE is below $2^{-\lambda}$ for the given parameter sets. Previous work [SV19b,SV19a,DGK20] approximates the DFR by (i) directly measuring the average DFR for smaller code sizes by running the decoder (which is doable since the DFR is high enough so that decoding failures can be observed) then (ii) extrapolating the behavior to estimate the DFR for larger parameters that are out of reach of experiments. However, the DFR falls into two regimes, making such extrapolations unreliable. The first regime is called the *waterfall region*, where the DFR decays more than exponentially in the block size n of the scheme (with other parameters fixed). The second regime is called the *error floor* which kicks in when the block size is sufficiently large and where the decay is much slower (Figure 1.1). The issue is that the experiments are done in the waterfall region and extrapolation is made under the hypothesis that for the BIKE parameters we are still in the waterfall region. This could lead to underestimation of the DFR and consequently, an underestimation of the block size needed to achieve a particular security parameter λ.

It remains an open problem to predict for which value of n the error floor begins for BIKE. It is really a problem of being able to predict the iterative decoding performance of a QC-MDPC code. For LDPC codes (which have rows and columns of weight $O(1)$ compared to $O(\sqrt{n})$ for MDPC codes), when using the same kind of iterative decoder, the error floor phenomenon is better understood. For LDPC codes, the error floor is either due to the existence of low-weight codewords which would fool any decoder, or of small *near codewords*. They are also called trapping sets and are errors of small weight with a syndrome also of small weight [Ric03,HB18,VCN14]. Error vectors which have a large enough intersection with those codewords or near codewords are known to cause the error floor behavior in LDPC codes.

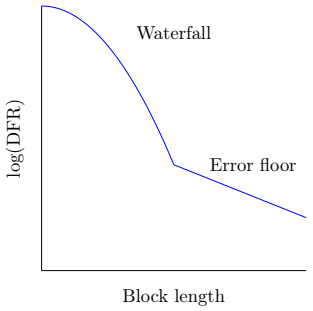


Fig. 1.1: Waterfall and error floor regions for decoding failure rate.

On the other hand, MDPC codes typically have no low-weight codewords or near codewords. However, BIKE is based on QC-MDPC codes, and this gives BIKE more structure than a random MDPC code. BIKE admits a parity-check matrix $\boldsymbol{H}$ formed by two circulant blocks. Such codes have codewords of weight equal to the row weight w of $\boldsymbol{H}$ (which is of order $O(\sqrt{n})$) [SV19a,BBC$^+$21]. There are near codewords of half the row weight which, due to the block-circulant structure, is also equal to the column weight d of $\boldsymbol{H}$. In particular, the set of *near codewords* consists of error vectors where either the first or second half of bits is precisely a row of $\boldsymbol{H}$ and the rest of the bits are 0 [Vas21]. The lower bound[6] on the DFR based on those moderate-weight codewords is too low to be of concern for the BIKE parameters [BBC$^+$21].

However, a clear explanation of the error floor for QC-MDPC has not been found to date. The effect of the near codewords put forward in [Vas21] could be a factor since it has been shown in [Vas21] that if the error vector has a large intersection with one of the near codewords, then the DFR conditioned on this event is non-negligible and can be observed experimentally for the BIKE parameters. However, the probability that the error has a large enough intersection with those near codewords so that the DFR conditioned on this event can be measured experimentally is too small to be of concern for the BIKE parameters, even if it is bigger than the contribution to the DFR coming from the moderate-weight codewords. The experimental study conducted in [ABH$^+$22] shows that for scaled-down BIKE parameters where the error floor can be observed experimentally, the errors that contribute to the DFR do not have large intersection with the near codewords of [Vas21].

[6] It corresponds to the probability that the error covers at least half one of those codewords.

Remarkably, the waterfall region is much better understood. Sendrier and Vasseur showed that an iterative decoder which works step by step can be analyzed by a Markov chain approach [SV19b,Vas21]. Decoding step by step means that bits are flipped one at a time: at each step a random position is considered and flipped according to whether the number of unsatisfied parity-checks involving this position (called the *counter* of the position) is greater than some threshold. The rationale behind this rule is as follows: Denote by π_0 the probability that a parity-check involving a bit not in error is not satisfied. Similarly we denote by π_1 the probability that a parity-check involving a bit in error is not satisfied. The crucial observation is that

$$\pi_0 < \pi_1. \tag{1.1}$$

Recall that due to the regular structure of $\boldsymbol{H}$ all positions are involved in the same number d of parity checks. We therefore expect that the counter of a position which is not in error behaves as a binomial random variable $\mathrm{Bin}(d, \pi_0)$ for parameters d and π_0[7] whereas the counter of a position which is in error behaves as $\mathrm{Bin}(d, \pi_1)$. Therefore a bit in error tends to be involved in more unsatisfied parity checks than a bit which is correct. For the standard iterative decoder, all positions are considered at once and flipped or not according to the same rule. The fact that the decoder is step by step makes it much more amenable to a Markov-chain modeling where the states are the pairs (s, t), t being the number of positions which are in error and s the syndrome weight of the error. This Markov-chain model closely predicts the DFR in the waterfall region but does not capture the error floor region. Roughly speaking, the DFR is predicted here by computing the resulting probability distributions of the pairs (s, t). The predicted DFR is the probability to attain a state where t is not zero at the end.

1.2 Intuition behind our approach

Convergence to near codewords in the error floor regime. Our work started with a fundamental observation, namely that in the toy examples for which we could observe the error floor phenomenon experimentally, the remaining error at the end of the iterative decoding process when decoding failed covered in many cases one of the near codewords put forward in [Vas21]. Interestingly enough, [ABH+22, Fig 3,4 and 5] shows that in the error floor regime, errors which result in a decoding failure tend to have a bigger intersection with at least one of those near codewords than a random error of the same size. It was also experimentally shown in [ABH+24] that the majority of decoding failures in the error floor of a scaled-down version of BIKE were due to these near codewords. Moreover, when we increase the block size resulting in parameters even further from the waterfall region, this phenomenon becomes even more prominent.

There is a good justification for this behavior as explained in §2.4. It was noted in [BBC+21] that there exist *bad counters* which can lead to decoding failures. We provide a full classification of such bad counters. The point is that we actually do not have two kinds of behaviors for the counters depending on if the bit is in error or not as explained before, but rather four different behaviors of the counters. At each iteration of the iterative decoder, consider the near codeword $\boldsymbol{\nu}$ that is the closest to the actual error (incidentally, as explained before, all near codewords are of weight d, the column weight of $\boldsymbol{H}$) and denote by u the size of its intersection with the actual error $\boldsymbol{e}$. As we will explain in §2.4, we expect the following four kinds of behavior:

1. The counters of the u bits belonging to the support of $\boldsymbol{\nu}$ and $\boldsymbol{e}$ behave like the sum of two independent binomial variables $\mathrm{Bin}(u-1, \pi_0) + \mathrm{Bin}(d-u+1, \pi_1)$
2. The counters of the $d-u$ bits not in error but in the support of $\boldsymbol{\nu}$ behave as the sum of two binomial variables $\mathrm{Bin}(u, \pi_1) + \mathrm{Bin}(d-u, \pi_0)$.
3. The counters of the rest of the $t-u$ bits in error behave "as usual" as a binomial variable $\mathrm{Bin}(d, \pi_1)$.

[7] The notation $\mathrm{Bin}(n, p)$ denotes a binomial random variable of parameters n and p; *i.e.* the sum of n i.i.d. Bernoulli variables X_i of parameter p, that is $\mathbf{P}(X_i = 1) = p$ for all i in $\{1, \cdots, n\}$.

4. The counters of the rest of the $n-d-t+u$ bits that are not in error also behave as expected, namely as a binomial variable $\mathrm{Bin}(d, \pi_0)$.

In other words, since $\pi_0 < \pi_1$, we expect an abnormal behavior from the d bits which belong to the closest near codeword. Indeed, those that are in error are less likely to be corrected by the iterative decoding process than the others which are in error: their counter is somewhat in between a counter of a bit which is in error and a counter of a bit which is not in error. Similarly, the $d-u$ bits of the near codeword that are not in error have a greater chance to be wrongly flipped by the iterative decoding process than the other bits in error. This is even worse, because the more bits are in error in the near codeword, the more the bits involved in this near codeword behave as their opposite: those that are still not in error tend to look even more like bits which are in error. We experience in this case a snowball effect. As soon as enough bits are in error in the near codeword, there is a non negligible chance to end up at the end of the iterative decoding process with the whole near codeword being in error. Due to the peculiar structure of near codewords, the iterative decoding process gets stuck there.

This discussion suggests that errors which give rise to a decoding failure in the error floor region should be somewhat closer to one of those near codewords at the beginning, and the intersection size only increases during the decoding process. This is why we adapted the Markov model of [SV19b,Vas21] to also keep track of this parameter u which is the biggest intersection of a near codeword with the residual error vector in the decoding process.

1.3 Our approach

A new Markov model. We have followed two different but related approaches for keeping track of the size of the intersection of the error vector with the near codewords in a Markov model. At each step, it is possible to move from one state to another with prescribed probabilities. It is assumed that transitioning to any state depends only on the current state.

- The first approach is to use the simple model for the counters described before together with a simplified model of the parity-check matrix for computing the transition probabilities. This simple model keeps track of the size u of the intersection of the error with a single, randomly selected near codeword $\boldsymbol{\nu}$. We estimate the contribution to the DFR coming from $\boldsymbol{\nu}$. This contribution is measured and used to account for the contributions coming from all near codewords. See Section 5.
- The second approach is much more involved. It computes the transition probabilities much more accurately. Among several improvements, it takes into account the secret key structure by using for this computation the specific structure of the near codewords corresponding to a given key. The only simplifying assumption which is made is that in the error floor regime the decoding failure is dominated by errors which are close to a single near codeword, and that this does not change during the decoding process. In this case, the Markov model keeps track of the size u of the intersection of the error with the closest near codeword. We capture the DFR in the error floor regime with high accuracy within this model. See Section 6.

A Markov model taking into account the structure of the key. The first approach has been tested with scaled-down BIKE parameters and by taking a random key $\boldsymbol{H}$. To simplify the computation of π_0 and π_1 we made the heuristic assumption that the columns of $\boldsymbol{H}$ are drawn uniformly at random up to fixed column weight d. This new Markov model captures the error floor region and provides a much more conservative estimate than the DFR estimate based on the Markov model of [SV19b,Vas21] which is too optimistic in the error floor regime. It turns out that the new DFR estimate is larger than the true estimate in both the waterfall and error floor regions (and not only in the waterfall regime as was the case for the Markov model of [SV19b,Vas21]). This is quite encouraging since even with this simple model there is a tool for choosing the parameters of BIKE in a conservative manner.

However, one drawback of this simple Markov model is that it does not take into account the structure of the key. It is indeed well known that there are weak keys [DGK20,Vas21,NSP+23,WWW23] for which the decoding process behaves really badly. Moreover, there is a non-negligible difference between the best and worst key of the DFR in the few thousand of instances we took in the toy example we considered. This led us to refine the Markov model. We have tested Approach 1.3 and improved it in two ways : we took into account the structure of the key and we computed the transition probabilities in a more sophisticated way.

One of the difficulties of having a Markov model working for all sorts of keys is that the near codewords split into two classes, those that have their support in the first half of the code positions and those having their support in the second half. These two classes of near codewords might affect the DFR in a very different way. To circumvent this problem, we added a bit to the Markov state, *i.e.* getting a state (s, t, u, b) where $b = 0$ indicates that the closest near codeword has all its support in the first half of the positions and 1 otherwise. At the expense of increasing the state space size and adding more involved transition probability formulas, we get a Markov model which captures now all possible keys. Remarkably enough, in the toy examples we tried, this model is still accurate for predicting the error floor for the step-by-step decoder and other variants: the majority decoder (the threshold for flipping being half the total number of equations involving a bit), the BGF decoder, or a decoder with a new custom threshold rule which improves the DFR by a significant amount.

Moreover, we can also use our model to examine in more detail what causes the DFR. In order to do so, we split the contribution of the DFR into two parts. The first one is the contribution $\mathrm{DFR_e}$ to the DFR where decoding failed because it ended up covering a near codeword: The final state where we got blocked in the Markov chain was a state (s, t, u, b) for which $u = d$. It can be viewed in some sense as the error-floor term in the DFR. The second one is just the total DFR minus $\mathrm{DFR_e}$: $\mathrm{DFR_w} = \mathrm{DFR} - \mathrm{DFR_e}$. The second one can be viewed as the waterfall part of the DFR. It turns out that not only did our Markov model accurately predict the DFR, but did also predict well each term separately. Indeed, even in the error floor region where $\mathrm{DFR_e}$ is really bigger than $\mathrm{DFR_w}$, the $\mathrm{DFR_w}$ term predicted by our model matches the $\mathrm{DFR_w}$ term computed in the experiments. This is a further indication that the model predicts accurately the error floor and the waterfall behaviors. All in all, in this work, we conclude that the dominant contribution to the error floor for QC-MDPC codes under iterative decoders is due to convergence to a near codeword.

1.4 Summary of the main results

We present two Markovian models for the decoding behavior of QC-MDPC codes under iterative, hard-decision decoders: the simplified model and the refined model. The Simplified Model considers the distance of the error vector at each round of decoding from one fixed, randomly-selected near codeword. This model captures the error floor regime and gives a more conservative estimate than experimental data and the [SV19b] model. It is discussed in Section 5. The Refined Model for All Keys considers the distance of the error vector at each round of decoding from the closest near codeword and takes the structure of the key into account. This is done by using the degree distribution of a certain graph associated with the key to derive the Markov model. This model is discussed in Section 6. We use this model to

- give an extremely close approximation of the experimental data in both the waterfall and error floor regimes, further improving upon the first model and the work of [SV19b]
- estimate for the BIKE 1 parameters and with the previous BIKE 1 decoder using a single threshold function a DFR of about 2^{-91}
- show that for this decoding rule, there is an error floor behavior which appears already for slightly larger values of the block length than the one which was chosen and depending on the gap parameter δ of the decoder gives a contribution to the DFR which is between 2^{-100} and 2^{-120}

– improve this decoding rule by simply raising the minimum allowed threshold from 36 to 39 to show that for a slightly larger block length ($r = 13477$ instead of $r = 12323$), we obtain a DFR which is below $2^{-131.2}$ for a typical key.

These models predict and experiments confirm that the dominating cause of failures in the error floor regime in a wide range of parameters is due to convergence to near codewords. All models use the step-by-step decoder which is experimentally shown to produce more decoding failures than parallel decoders. The most impactful model is the Refined Model for All Keys which matches the experimental data, also run using a step-by-step decoder, remarkably well throughout both the error floor and waterfall regimes. This model is applied to the BIKE cryptosystem parameter sets to give a DFR approximation; it is expected that the DFR for the same parameter sets using a parallel bit-flipping decoder as specified in [ABB+21] and decoding rule based on iteration dependent threshold functions would result in a lower DFR than what is approximated by the model.

All in all, this paper solves a major issue for the fourth round candidate BIKE of the NIST PQC competition, namely to give a convincing prediction for the DFR of this cryptosystem. Being able to show that this quantity is below $2^{-\lambda}$ is a major concern for BIKE to prove its IND-CCA2 security. This paper gives some strong evidence that this requirement can be met at the cost of a modest increase of less than 10% in the key size, by raising a little bit the minimum threshold, and filtering out atypical keys during key generation.

2 Background

2.1 Notation

Basic notation. Vectors and matrices are respectively denoted in bold letters and bold capital letters such as $\boldsymbol{a}$ and $\boldsymbol{A}$. Vectors are assumed to be row vectors and $\boldsymbol{x}^\intercal$ denotes the column vector which is the transpose of the row vector $\boldsymbol{x}$. For a matrix $\boldsymbol{A}$, the vector that is the transpose of the j-th column of $\boldsymbol{A}$ is denoted by $\boldsymbol{A}_j$. The entry at index i of the vector $\boldsymbol{x}$ is denoted by x_i or $x(i)$. The Schur (component-wise) product $(x_i y_i)_{1 \leqslant i \leqslant n}$ of two vectors $\boldsymbol{x} = (x_i)_{1 \leqslant i \leqslant n}$ and $\boldsymbol{y} = (y_i)_{1 \leqslant i \leqslant n}$ of the same length is denoted by $\boldsymbol{x} \star \boldsymbol{y}$. The Hamming weight $|\boldsymbol{x}|$ of a vector $\boldsymbol{x}$ is its number of nonzero entries.

Probabilistic notation. $\mathrm{Bin}(n, p)$ denotes a random variable following a binomial distribution with parameters n and p, that is the sum $\sum_{i=1}^{n} X_i$ of n i.i.d. Bernoulli variables with parameter p. Expressions like $\mathrm{Bin}(n_1, p_1) + \mathrm{Bin}(n_2, p_2)$ stand for the sum of the two independent binomial random variables $\mathrm{Bin}(n_1, p_1)$ and $\mathrm{Bin}(n_2, p_2)$. By abuse of notation, we write $X \sim Y$ when two random variables X and Y follow the same distribution. For a finite set $\mathscr{S}$, $x \xleftarrow{\$} \mathscr{S}$ means that x is an element of $\mathscr{S}$ sampled uniformly at random in $\mathscr{S}$.

Binary linear codes. All codes considered here are binary and linear, which means that they are vector spaces over $\mathbb{F}_2$. A code of length n and dimension k is a k-dimensional subspace of $\mathbb{F}_2^n$. We say that such a code is an $[n, k]$ code. They are specified here by a parity-check matrix, meaning a matrix whose kernel is the corresponding code, *i.e.* an $[n, k]$ code $\mathscr{C}$ is specified by a full-rank parity-check matrix $\boldsymbol{H} \in \mathbb{F}_2^{(n-k) \times n}$ as $\mathscr{C} = \{\boldsymbol{c} \in \mathbb{F}_2^n : \boldsymbol{H}\boldsymbol{c}^\intercal = 0\}$.

Quasi-cyclic codes and polynomial notation. On top of being binary and linear, all the codes considered are quasi-cyclic, *i.e.* their parity-check matrix is formed by blocks of circulant matrices. More precisely, they are double circulant codes, meaning that the parity-check matrix has the form $\boldsymbol{H} = \begin{pmatrix} \boldsymbol{C}_0 & \boldsymbol{C}_1 \end{pmatrix}$ where both $\boldsymbol{C}_0$ and $\boldsymbol{C}_1$ are circulant matrices. From now on, r will denote the size of $\boldsymbol{C}_0$ and $\boldsymbol{C}_1$ and the length n of the code will be $n = 2r$. $\boldsymbol{C}_0$ and $\boldsymbol{C}_1$ are also supposed to have the same column weight (and therefore also the same row weight). This weight is

denoted by d. The row weight w of $\boldsymbol{H}$ is constant and satisfies $w = 2d$. Such codes are conveniently represented by polynomials since $r \times r$ binary circulant matrices form a ring $\mathscr{R}$ which is isomorphic to $\mathbb{F}_2[x]/(x^r - 1)$, where the isomorphism Ψ from $\mathscr{R}$ to $\mathbb{F}_2[x]/(x^r - 1)$ is given by

$$\begin{pmatrix} c_0 & c_{r-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{r-1} & & c_2 \\ \vdots & c_1 & c_0 & \ddots & \vdots \\ & \vdots & \ddots & \ddots & c_{r-1} \\ c_{r-2} & & & & \\ c_{r-1} & c_{r-2} & \cdots & c_1 & c_0 \end{pmatrix} \mapsto c_0 + c_1 x + \cdots + c_{r-1} x^{r-1}.$$

We use this isomorphism to view $\boldsymbol{H} = \begin{pmatrix} \boldsymbol{C}_0 & \boldsymbol{C}_1 \end{pmatrix}$ as the pair $(h_0(x), h_1(x))$ where $h_0(x) = \Psi(\boldsymbol{C}_0)$ and $h_1(x) = \Psi(\boldsymbol{C}_1)$. We overload the notation from now on:

Notation 1. *We denote the parity-check matrix* $\boldsymbol{H} \stackrel{def}{=} (h_0, h_1)$. *We also write*

$$h_0(x) = \sum_{j=1}^{d} x^{l_j} \quad \text{and} \quad h_1(x) = \sum_{j=1}^{d} x^{r_j},$$

where l_j *and* r_j *run over the nonzero coefficients of* h_0 *and* h_1, *respectively.*

We likewise represent error vectors $\boldsymbol{e} = (e_0, \cdots, e_{n-1}) \in \mathbb{F}_2^n$ with polynomials

$$\boldsymbol{e} \rightsquigarrow e(x) = e_0(x) \oplus e_1(x) = \sum_{j=0}^{r-1} e_j x^j \oplus \sum_{j=0}^{r-1} e_{r+j} x^j \in (\mathbb{F}_2[x]/(x^r - 1))^2$$

We use here "$\oplus$" to denote a formal sum, and "+" to denote addition in the ring $\mathbb{F}_2[x]/(x^r - 1)$. The syndrome $\boldsymbol{s} = (s_0, \cdots, s_{r-1})$ of an error $\boldsymbol{e} = (e_0, ..., e_{n-1})$, *i.e.* $\boldsymbol{s}^\intercal = \boldsymbol{H}\boldsymbol{e}^\intercal$ is also represented conveniently as a polynomial $s(x) = \sum_{j=0}^{r-1} s_j x^j$ and it is readily seen that

$$s(x) = e_0(x)h_0(x) + e_1(x)h_1(x). \tag{2.1}$$

Here and elsewhere it is important to note that all polynomial operations will be performed in the ring $\mathbb{F}_2[x]/(x^r - 1)$, that is modulo $x^r - 1$.

2.2 Iterative Decoders

The problem we will be interested to solve efficiently during the deciphering process is the decoding problem which in its syndrome form is described as

Problem 2.1 (syndrome decoding problem (SDP)). Given an $r \times n$ parity-check matrix $\boldsymbol{H} \in \mathbb{F}_2^{r \times n}$ and an error vector $\boldsymbol{e}$ sampled uniformly at random of weight t with syndrome $\boldsymbol{s}^\intercal = \boldsymbol{H}\boldsymbol{e}^\intercal$, find an error vector $\boldsymbol{e}'$ that satisfies $\boldsymbol{H}\boldsymbol{e}'^\intercal = \boldsymbol{s}^\intercal$ and $|\boldsymbol{e}'| = t$.

Iterative, bit-flipping syndrome decoders generally solve the SDP by guessing that columns of $\boldsymbol{H}$ with *many* bits in common with $\boldsymbol{s}$ are likely affected by errors. Recall that $\boldsymbol{H}_i$ denotes the transpose of the i-th column of matrix $\boldsymbol{H}$, and define $\sigma_i(\boldsymbol{H}, \boldsymbol{s}) = |\boldsymbol{H}_i \star \boldsymbol{s}|$, the number of unsatisfied parity-check equations involving i. We will denote this quantity simply by σ_i in what follows and say that it is the *counter* corresponding to bit i. The bit-flipping decoders initialize $\boldsymbol{e}' = \boldsymbol{0}$, compute σ_j for all or some $j \in \{0, \ldots, n-1\}$, then flip the j-th bit e'_j whenever $\sigma_j \geq T$, where T is a given threshold function. We say a decoding failure has occurred whenever the output $\boldsymbol{e}'$ satisfies: $\boldsymbol{H}\boldsymbol{e}'^\intercal \neq \boldsymbol{s}^\intercal$ or $\boldsymbol{e}' \neq \boldsymbol{e}$.

The accuracy of bit-flipping decoders greatly depends on the threshold function T. If the threshold is too high, the decoder ceases to flip bits before arriving at the correct vector $\boldsymbol{e}'$. If the threshold is too low, more bits are flipped than should be, again resulting in an incorrect vector

$\boldsymbol{e}'$. T is always chosen such that $T \geqslant \frac{d+1}{2}$. This ensures that the weight of $\Delta\boldsymbol{s}^\intercal \stackrel{\text{def}}{=} \boldsymbol{s}^\intercal + \boldsymbol{H}\boldsymbol{e}'^\intercal$ is strictly decreasing, so that we may expect it to reach 0 in order to have $\boldsymbol{H}\boldsymbol{e}'^\intercal = \boldsymbol{s}^\intercal$ at the end.

It is most common for iterative decoders to consider each column of $\boldsymbol{H}$ in comparison with $\boldsymbol{s}$ during each round of decoding. This can result in multiple bits of the guess $\boldsymbol{e}'$ being flipped in one round of decoding. The BIKE cryptosystem round 4 specification uses the Black-Grey-Flip (BGF) iterative decoder [DGK20]. A closed form analysis of the DFR of the BGF decoder remains an open problem.

In this work we analyze the step by step decoder - an iterative decoder which randomly selects one column of $\boldsymbol{H}$ for comparison with $\boldsymbol{s}$ during each decoding round: At most one bit is flipped per round. This simplifies the computation of the Markov chain transition probabilities by restricting the number of possible states in the next step. It is detailed in Algorithm 2.1.

Algorithm 2.1 Step by step decoder.

Input: $\boldsymbol{H} \in \mathbb{F}_2^{r\times n}$, $\boldsymbol{s} \in \mathbb{F}_2^r$, $t \in \{1,\dots,n\}$
Output: $\boldsymbol{e} \in \mathbb{F}_2^n$ such that $\boldsymbol{H}\boldsymbol{e}^\intercal = \boldsymbol{s}^\intercal$ and $|\boldsymbol{e}| = t$ or $\perp$ (decoding failure)
Require: a threshold function T such that $T \geqslant \frac{d+1}{2}$
Ensure: the weight of $\Delta\boldsymbol{s}^\intercal = \boldsymbol{s}^\intercal + \boldsymbol{H}\boldsymbol{e}'^\intercal$ decreases strictly in the algorithm below

$\boldsymbol{e}' \leftarrow \boldsymbol{0}$, $\Delta\boldsymbol{s} \leftarrow \boldsymbol{s}$
while $\Delta\boldsymbol{s} \neq 0$ **do**
  $\mathscr{E} \leftarrow \{\{0,\cdots,n-1\} : |\boldsymbol{H}_j \star \Delta\boldsymbol{s}| \geqslant T\}$ ▷ $|\boldsymbol{H}_j \star \Delta\boldsymbol{s}|$ is the counter σ_j
  if $\mathscr{E} \neq \emptyset$ **then**
    $j \xleftarrow{\$} \mathscr{E}$
    $e'_j \leftarrow 1 - e'_j$, $\Delta\boldsymbol{s} \leftarrow \Delta\boldsymbol{s} + \boldsymbol{H}_j$ ▷ $|\Delta\boldsymbol{s}|$ has decreased by $d - 2\sigma_j$
  else return $(\perp)$ ▷ The decoder is then in a blocked state
if $|\boldsymbol{e}'| = t$ **then return** $(\boldsymbol{e}')$
else return $(\perp)$

The step by step decoder has been studied in detail and experimental evidence [Vas21, Chap 7, Fig. 7.1] shows that for the classical bit flipping strategy, the step by step decoder performs significantly worse than the parallel version.

2.3 Near codewords

Near codewords were introduced in the literature of iterative decoding as a way to formalize a collection of error vectors with lower-weight, nonzero syndrome than expected, which are therefore difficult to decode.

Definition 2.2 (near codeword of type (s,t)). *An error vector $\boldsymbol{e} \in \mathbb{F}_2^n$ is a near codeword of type (s,t) if $|\boldsymbol{e}| = t$ and $|\boldsymbol{H}\boldsymbol{e}^T| = s$.*

MDPC codes are known to have minimum distance which is typically linear in the code-length n and are unlikely to have small near codewords. This is not the case for QC-MDPC codes: the minimum distance is of order $O(\sqrt{n})$. This is because $h_1(x) \oplus h_0(x)$ belongs to the code of parity-check matrix $\boldsymbol{H} = (h_0, h_1)$. This leads one to suspect that such codes also have small-weight near codewords. This is indeed the case and that this is most likely the main issue for decoding as identified in [Vas21]. To define them, fix a parity-check matrix $\boldsymbol{H} = (h_0, h_1)$.

Definition 2.3 (The set $\mathcal{N}$ of near codewords). *The set $\mathcal{N}$ of near codewords is the union of all n-bit vectors with polynomial representations of the form $x^i h_0(x) \oplus \boldsymbol{0}$ and $\boldsymbol{0} \oplus x^i h_1(x)$, for all $i \in \{0,...,r-1\}$.*

In this work, when we refer to a near codeword, we mean an element of $\mathcal{N}$.

Unusually low syndrome weight. Elements of $\mathcal{N}$ are (d,d)-near codewords as per Def. 2.2: they are of weight d and have an unusually small syndrome weight of d. Moreover, even errors $\boldsymbol{e}$ which have an overlap of size u with a near code codeword $\boldsymbol{\nu}$ of this kind (meaning that $|\boldsymbol{e}\star\boldsymbol{\nu}| = u$) have an unusally low weight syndrome. If $|\boldsymbol{e}| = t$, the syndrome weight can be in general be as large as $d \cdot t$. Here it is always $\leqslant d \cdot t - u(u-1)$.

This can be read off directly from the polynomial representation of the syndrome. To verify the first claim, the near codeword $e(x) = x^i h_0(x) \oplus 0 \in \mathcal{N}$, for example, has syndrome $s(x) = (x^i h_0(x))h_0(x) = x^i h_0^2(x)$, which is of weight d as $h_0(x)$ has precisely d nonzero coefficients, and in $\mathbb{F}_2[x]/(x^r-1)$ squaring and multiplication by x preserve the number of nonzero coefficients. To verify the second claim suppose that the near codeword $\boldsymbol{\nu}$ is say of the form $x^i h_0(x) \oplus 0$, the error $\boldsymbol{e}$ can be written as a polynomial as $e(x) = e_0(x) \oplus e_1(x)$, $x^i h_0(x) = a(x) + b(x)$ and $e_0(x) = a(x) + c(x)$ where $a(x)$ is the polynomial for the common part $\boldsymbol{e}\star\boldsymbol{\nu}$. The syndrome of $\boldsymbol{e}$ is

$$\begin{aligned} s(x) &= e_0(x)h_0(x) + e_1(x)h_1(x) \\ &= x^{r-i}(a(x)+b(x))(a(x)+c(x)) + e_1(x)h_1(x) \\ &= x^{r-i}\left(a(x)^2 + a(x)b(x) + a(x)c(x) + b(x)c(x)\right) + e_1(x)h_1(x). \end{aligned}$$

Recall here that we perform the polynomial computation over $\mathbb{F}_2[x]/(x^r-1)$ and that $x^{r-i}x^i = 1$ in this ring. Let us denote by $|P|$ the weight of a polynomial P, namely the number of its non zero coefficients. The weight satisfies $|P \cdot Q| \leqslant |P| \cdot |Q|$ and $|P+Q| \leqslant |P| + |Q|$ for all polynomials P and Q. Notice that $d \cdot t$ is exactly $|e_0| \cdot |h_0| + |e_1| \cdot |h_1| = |a|^2 + |a| \cdot |b| + |a| \cdot |c| + |b| \cdot |c| + |e_1| \cdot |h_1|$. The point is that $|a^2| = |a|$ because of the *forced cancellations* in the product $a(x)^2$. Indeed, as in the previous point, squaring preserves the number of nonzero coefficients. This implies that we have at least a drop of $|a|^2 - |a| = u(u-1)$ in the syndrome weight as claimed above.

2.4 Tanner graphs

We use the language of *Tanner graphs* to fully explain the role of *bad counters*, first considered in [BBC+21, §4.1].

Definition. A Tanner graph is a very handy tool for analyzing iterative decoding of LDPC or MDPC codes. For LDPC codes, this notion dates back to Gallager who explained and studied his iterative decoding algorithms [Gal63] by using them. In a more general form, they have been defined in [Tan81]. It is a bipartite graph which represents the parity-check matrix $\boldsymbol{H} = (H_{i,j})_{\substack{0 \leqslant i < r \\ 0 \leqslant j < n}}$ of a code. It has two types of vertices, the *variable nodes* which are in bijection with the code positions $\{0, \cdots, n-1\}$, and the *check nodes* which are in bijection with the parity checks, *i.e.* the rows of $\boldsymbol{H}$. There is an edge between a variable node j and the check node i if and only if $H_{i,j} = 1$.

In our case, where $n = 2r$ and the parity-check matrix is formed by two circulant blocks, we can relate this to the polynomial notation established in §2.1 and index the variable nodes and check nodes a little bit differently. More specifically, the variable node associated to the i-th code position is indexed by $x^i \oplus 0$ if $i < r$ and by $0 \oplus x^{i-r}$ if $i \geqslant r$. Polynomial notation is also used to index the check nodes. The check node associated to the i-th row of $\boldsymbol{H}$ is indexed by x^i. When $\boldsymbol{H} = (h_0, h_1)$, the set of check nodes adjacent to the variable node $x^i \oplus 0$ is the set of monomials in $x^i h_0(x)$ and the set of check nodes adjacent to the variable node $0 \oplus x^i$ is the set of monomials appearing in $x^i h_1(x)$. Note that the syndrome weight $|\boldsymbol{H}\boldsymbol{e}^\intercal|$ of an error $\boldsymbol{e}$ can be read off from the Tanner graph:

Fact 1. *Consider the subgraph of the Tanner graph induced by an error $\boldsymbol{e}$: It is formed by the variable nodes belonging to $\boldsymbol{e}$ and all the check nodes adjacent to it. The number of check nodes of odd degree in this graph is $|\boldsymbol{H}\boldsymbol{e}^\intercal|$. The counter of a bit is the number of check nodes in this subgraph that are of odd degree and that are adjacent to it in the complete Tanner graph.*

Subgraph induced by a near codeword. The subgraph $\mathscr{G}$ of the Tanner graph induced by a near codeword $\boldsymbol{\nu}$ in $\mathcal{N}$ has a special structure. It has a unusually small number of check nodes for subgraphs induced by d variable nodes. We detail below the structure when $\boldsymbol{\nu}$ is $h_0(x) \oplus 0$ and recall that $h_0(x) = \sum_{i=1}^{d} x^{l_i}$, but the general case can be deduced from this one, by a shift of the positions and/or replacing l_i by r_i in the case of a near codeword $0 \oplus x^m h_1(x)$.

Fact 2. *Consider the subgraph $\mathscr{G}$ of the Tanner graph induced by $\boldsymbol{\nu} = h_0(x) \oplus 0$. The variables nodes are the d variable nodes labeled $x^{l_i} \oplus 0$ for $i \in \{1, \cdots, d\}$. The check nodes are all the $x^{l_i+l_j}$. There are at most $\binom{d+1}{2}$ of them. Each pair of variable nodes $(x^{l_i} \oplus 0, x^{l_j} \oplus 0)$ in $\mathscr{G}$ has at least one check node in common, namely $x^{l_i+l_j}$.*

This simple fact has a number of consequences concerning the influence that near codewords have on the counters. It will be helpful here to distinguish four kind of bits.

Four types of bits.

Definition 2.4. *Given a fixed near codeword $\boldsymbol{\nu}$ and a fixed error vector $\boldsymbol{e}$ intersecting $\boldsymbol{\nu}$ in u positions, we classify the n error vector bits into the four categories:*

- u bad bits: *the u bits belonging to the supports of both $\boldsymbol{e}$ and $\boldsymbol{\nu}$;*
- $d-u$ suspicious bits: *the other $d-u$ bits of $\boldsymbol{\nu}$ (not in the support of $\boldsymbol{e}$);*
- $t-u$ normal bits in error: *the other $t-u$ bits of $\boldsymbol{e}$ (not in the support of $\boldsymbol{\nu}$);*
- $n-d-t+u$ good bits: *neither in $\boldsymbol{\nu}$ nor $\boldsymbol{e}$.*

Bad bits and suspicious bits really display an abnormal behavior concerning the counters due to Fact 2 which explains why they are a nuisance for iterative decoding. Indeed we expect that the counters σ_j behave like (to simplify the discussion we assume that $\boldsymbol{\nu} = h_0(x) \oplus 0$)

bad bit: $\sigma_j \sim \mathrm{Bin}(u-1, \pi_0) + \mathrm{Bin}(d-u+1, \pi_1)$. The bad bit which is some $x^{l_i} \oplus 0$ is adjacent to $u-1$ check nodes of the form $x^{l_i+l_j}$ where $x^{l_j} \oplus 0$ is another bad bit. The contributions of these two bits for this parity-check cancel out, and this parity-check behaves in first approximation as a parity adjacent to a bit which is not in error. Its probability for not being satisfied is therefore π_0. Similarly the bad bit is adjacent to the check node x^{2l_i} and $d-u$ check nodes of the form $x^{l_i+l_j}$ where $x^{l_j} \oplus 0$ is now a suspicious bit. In both cases, the contribution of those bits to the parity-check is 1 and these $d-u+1$ parity-checks should behave as parity-checks adjacent to a bit which is in error, in which case the probability of not being satisfied is π_1.

sus. bit: $\sigma_j \sim \mathrm{Bin}(u, \pi_1) + \mathrm{Bin}(d-u, \pi_0)$. This comes from the fact that the suspicious bit which is some $x^{l_i} \oplus 0$ is adjacent to u check nodes of the form $x^{l_i+l_j}$ where $x^{l_j} \oplus 0$ is a bad bit (they contribute together to 1 in the parity-check) and to $d-u$ check nodes of the form $x^{l_i+l_j}$ where $x^{l_j} \oplus 0$ is another suspicious bit (they contribute together to 0 in the parity-check).

An illustration of this principle is given in Appendix D. Notice that the greater u is, the greater the chances are to flip a suspicious bit which further increases u. This is what we call the snowball effect leading to a potential convergence to a near codeword. This error is readily seen to uncorrectable by further steps of iterative decoding.

3 Convergence to a near codeword in $\mathcal{N}$ in the error floor

To verify the convergence to a near codeword in the error floor region we have tested parameters for which we can experimentally observe the error floor behavior. We have chosen a random QC-MDPC code with $r = 1723$, $d = 17$ and tested two decoding algorithms:

1. The majority step by step decoder: $T = \frac{d+1}{2} = 9$. It flips a bit every time it decreases the syndrome weight. For this decoder, we expect to hit the error floor faster than for other decoders because it has the most chance to flip suspicious bits, thus getting the snowball effect described in § 2.4.

2. An improved step by step decoder which uses a better (more conservative) decoding threshold. It starts with a high threshold value T_0 which favors to flip bits which are more likely to be in error. It is only when no bit can be flipped that a threshold $T_1 \leqslant T_0$ is used. The values are given below for a syndrome $s = |\boldsymbol{\Delta s}|$ in the step by step algorithm.

$$\begin{aligned} T_0 &= \min\left\{\max\left(\lfloor \alpha \cdot s + \beta \rfloor, \tfrac{d+1}{2}\right)\right\} \\ \text{where } \alpha &\stackrel{\text{def}}{=} 0.006016213884791455, \qquad \beta \stackrel{\text{def}}{=} 8.797325112097532. \\ T_1 &= \frac{d+1}{2}. \end{aligned} \tag{3.1}$$

Figure 3.1 shows the experimental results obtained for both decoders. It is a striking illustration of the phenomenon of convergence to a near codeword in the error floor regime for both decoders. The curves corresponding to the "better threshold" decoder correspond to a threshold pair (T_0, T_1) given in (3.1). Each point corresponds to at least 200 decoding failures. Together with the experimental curve we have drawn two other curves: "Contrib. of ncw" is the contribution to the DFR coming solely from the convergence to a near codeword. This contribution dominates in the error floor regime since the DFR is almost identical to this curve in this region. This can also be seen by removing from the DFR the contribution from the near codewords. These are the curves referred to as "Contr. of other". These curves show both a waterfall phenomenon and that, as we move further in the error floor region, the DFR contribution coming from convergence to a near codeword dominates the DFR. Both decoders display this phenomenon.

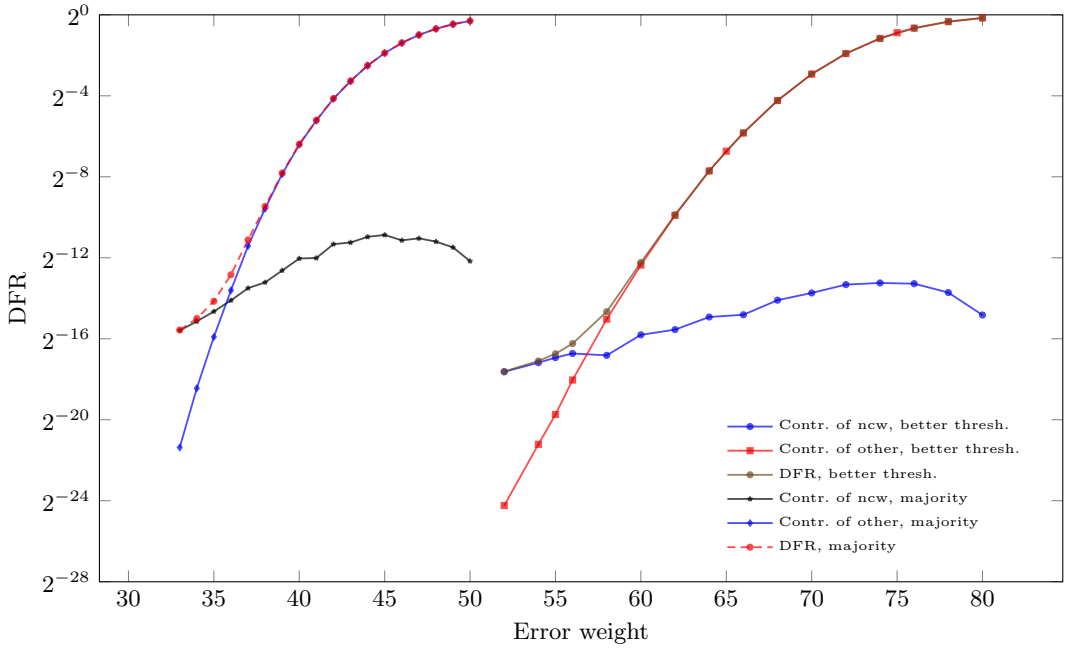


Fig. 3.1: DFR vs. error weight ($r = 1723$, $d = 17$), experimental curves.

4 Modeling Step by Step Decoding with a Markov Chain

4.1 The Markov Model

The Markov model introduced in [SV19b] kept track of the pair (s, t) where s and t are respectively the syndrome weight and the error weight during the step by step decoding process. Here by error

we mean the difference $\boldsymbol{e} - \boldsymbol{e}'$ between the true error $\boldsymbol{e}$ and the estimated error $\boldsymbol{e}'$ in Algorithm 2.1 and by syndrome we mean the syndrome of this difference, namely $\boldsymbol{H}(\boldsymbol{e} - \boldsymbol{e}')^\intercal = \boldsymbol{s}^\intercal - \boldsymbol{H}\boldsymbol{e}'^\intercal$ which is nothing but the vector $\boldsymbol{\Delta s}$ in Algorithm 2.1. This syndrome weight is therefore always decreasing during the decoding process and we hope to get an error weight of 0 at the end. The decoding process picks at random a certain error bit j and checks if $\sigma_j \geqslant T(s)$. If it is the case, it flips it and the resulting syndrome has weight equal to $s + d - 2\sigma$. t is either decreased or increased by 1 if the bit that is flipped was in in error or not. It may also happen that all counters of the n error bits are less than $T(s)$. In this case, we go to the `blocked` state and decoding has failed. If we get to the $(0, 0)$ state, decoding has succeeded.

This model does not take into account the crucial role of near codewords in the decoding process. Indeed, when the error has a large intersection with one of such near codewords, then it has also an influence on the syndrome weight which is smaller. For the two Markov models that we are considering, we are going to take near codewords into account in two ways: a first method consists in choosing a particular near codeword $\boldsymbol{\nu}$ and defines a quantity u which is the size of the intersection of this near codeword with the residual error during iterative decoding. A second method defines u as the size of the intersection of the error with the *closest* near codeword. In all cases, the Markov chain is defined in a similar way by letting the state be the triple (s, t, u) (actually there is an additional bit b in the definition of the state in the second method, but this is unessential for now). There are five transition cases:

- all bits have a counter which is too low to flip them. This we call the blocked state, denoted `blocked`. This corresponds to a decoding failure. This is an absorbing state of the Markov chain: once we are in this state, we stay in it. We denote by $p_{\texttt{blocked}}$ the probability that we are in this state.
- If there exists a bit which can be flipped, we choose one of such bits at random. We flip it and go to the state $(s + d - 2\sigma, t', u')$, where if it was a bad bit $t' = t - 1$ and $u' = u - 1$, a suspicious bit $t' = t + 1$ and $u' = u + 1$, a normal bit in error $t' = t - 1$ and $u' = u$, and a good bit $t' = t + 1$ and $u' = u$. Depending on whether the bit was respectively a bad bit, a normal bit in error, a suspicious bit or a good bit, we denote by $q_{\text{bad}}(\sigma)$, $q_{\text{err}}(\sigma)$, $q_{\text{sus}}(\sigma)$, $q_{\text{good}}(\sigma)$ the respective probabilities to flip the corresponding bit conditioned on the fact that there is at least one bit that we can flip. The corresponding transition probabilities are then $(1 - p_{\texttt{blocked}})q_\cdot(\sigma)$ where $\cdot \in \{\text{bad, err, sus, good}\}$.

The transitions are depicted in Fig. 4.1.

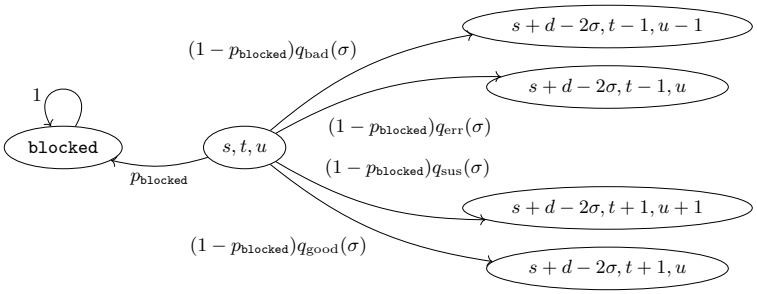


Fig. 4.1: Transition diagram starting with syndrome weight s, error weight t, and intersection u with a (d, d)-near codeword $\boldsymbol{\nu}$.

All these probabilities can be computed easily once we have the counter probabilities $p_{\text{bad}}(\sigma)$, $p_{\text{err}}(\sigma)$, $p_{\text{sus}}(\sigma)$ and $p_{\text{good}}(\sigma)$ that a bit chosen uniformly at random among respectively the bad bits, the normal bits in error, the suspicious bits and the good bits, has a counter equal to σ. If we let $p'_{\text{bad}}(\sigma)$, $p'_{\text{err}}(\sigma)$, $p'_{\text{sus}}(\sigma)$ and $p'_{\text{good}}(\sigma)$ the probabilities that a bit chosen uniformly at random has a counter equal to σ and belongs respectively to the group `bad`, `norm`, `sus` and

good, then clearly $p'_{\rm bad}(\sigma) = \frac{u}{n} p_{\rm bad}(\sigma)$, $p'_{\rm err}(\sigma) = \frac{t-u}{n} p_{\rm err}(\sigma)$, $p'_{\rm sus}(\sigma) = \frac{d-u}{n(1-p)} p_{\rm sus}(\sigma)$, $p'_{\rm good}(\sigma) = \frac{(n-t)-(d-u)}{n(1-p)} p_{\rm good}(\sigma)$. If we let $p \stackrel{\rm def}{=} \sum_{\sigma<T} \big(p'_{\rm bad}(\sigma)+p'_{\rm err}(\sigma)+p'_{\rm sus}(\sigma)+p'_{\rm good}(\sigma)\big)$, then the transition probabilities of Fig. 4.1 are given by

$$p_{\texttt{blocked}} = \Big(\sum_{\sigma<T} p_{\rm bad}(\sigma)\Big)^u \Big(\sum_{\sigma<T} p_{\rm err}(\sigma)\Big)^{t-u} \Big(\sum_{\sigma<T} p'_{\rm sus}(\sigma)\Big)^{d-u} \Big(\sum_{\sigma<T} p_{\rm good}(\sigma)\Big)^{n-t-d+u},$$

$$q_{\rm bad}(\sigma) = \frac{p'_{\rm bad}(\sigma)}{1-p}, \;\; q_{\rm err}(\sigma) = \frac{p'_{\rm err}(\sigma)}{1-p}, \;\; q_{\rm sus}(\sigma) = \frac{p'_{\rm sus}(\sigma)}{1-p}, \;\; q_{\rm good}(\sigma) = \frac{p'_{\rm good}(\sigma)}{1-p}.$$

The probabilities of the counters $p_{\rm bad}(\sigma)$, $p_{\rm err}(\sigma)$, $p_{\rm sus}(\sigma)$ and $p_{\rm good}(\sigma)$ depend on the model and will be dealt with later in the paper.

4.2 Computing the DFR with the help of the Markov chain

Let us show now how to efficiently compute the stationary probability distribution $\mathbf{P}_\infty(\texttt{blocked}|s,t,u)$ which is the probability that, after an infinite number of iterations, we end up in the blocked state (where decoding fails). We let $\mathbf{P}(s',t',u'|s,t,u)$ be the probability of transitioning to state (s',t',u') given that we were previously in state (s,t,u). We make use of the following equality

$$\mathbf{P}_\infty(\texttt{blocked}|s,t,u) = \mathbf{P}(\texttt{blocked}|s,t,u) + \sum_{s',t',u'} \mathbf{P}_\infty(\texttt{blocked}|s',t',u') \cdot \mathbf{P}(s',t',u'|s,t,u) \quad (4.1)$$

and the simple but crucial remark that the only transition probabilities $\mathbf{P}(s',t',u'|s,t,u)$ which are non zero have $s' < s$. This suggests to compute the probabilities $\mathbf{P}_\infty(\texttt{blocked}|s,t,u)$ by starting from $s = 0$ and increasing s by 1 each time and the recursion formula (4.1) to compute the remaining ones. For $s = 0$ we set $\mathbf{P}_\infty(\texttt{blocked}|0,t,u) = 0$ if $t < 2d$ and $\mathbf{P}_\infty(\texttt{blocked}|0,t,u) = 1$ otherwise. The first rule accounts for the fact that there should be no codeword of weight less than $w = 2d$. We also set $\mathbf{P}_\infty(\texttt{blocked}|s,t,u) = 0$ if $s < t(d-t+1)$ and $t \leqslant d$ to account for the fact that we make the assumption that there is no near-codeword of size $t < d$ which has a smaller syndrome than a subset of size t of a near codeword of $\mathcal{N}$. This leads to Algorithm 4.1. Computing the initial state distribution $\mathbf{P}(s,t,u)$ is detailed in Appendix §A. Note that once this

Algorithm 4.1 Algorithm for computing the DFR

for all t,u **do** ▷ initialization
  if $t < 2d$ **then**
    $\mathbf{P}_\infty(\texttt{blocked}|0,t,u) \leftarrow 0$
  else
    $\mathbf{P}_\infty(\texttt{blocked}|0,t,u) \leftarrow 1$
for all s,t,u s.t. $s < t(d-t+1)$ and $t \leqslant d$ **do**
  $\mathbf{P}_\infty(\texttt{blocked}|0,t,u) \leftarrow 0$
for $s = 1$ to $s_{\max}$ **do** ▷ main loop
  for all t,u **do**
    $\mathbf{P}_\infty(\texttt{blocked}|s,t,u) \leftarrow \sum_{s'<s,t',u'} \mathbf{P}_\infty(\texttt{blocked}|s',t',u') \cdot \mathbf{P}(s',t',u'|s,t,u) + \mathbf{P}(\texttt{blocked}|s,t,u)$
$\texttt{DFR} \leftarrow \sum_{s,t,u} \mathbf{P}_\infty(\texttt{blocked}|s,t,u)\, \mathbf{P}(s,t,u)$.

initial probability distribution is computed, the complexity for computing the DFR is just of order $O(E+S)$ where E is the number of possible transitions in the Markov chain and S the number of states.

5 A Simple Markovian Model

Since convergence to a near codeword dominates the error floor, it must be accounted for in a Markov state modeling the decoding process. We define the Markov state as the triple (s, t, u), where s is the syndrome weight, t is the error weight, and u is the size of the intersection of the support of the error with a *fixed, random near codeword* $\boldsymbol{\nu}$. To amplify this effect by the size of the set of near codewords, we post-process the DFR obtained by this model together with a DFR obtained from a variant model which does not take into account the effect of near codewords. The equation used to combine these DFRs is given in Section 5.2. This is in contrast to the models used in Section 6), where only the effect of the closest near codeword is taken into account, and this remains the only near codeword which is assumed to have an effect on the decoding process.

Model 1 (M). *Let $\boldsymbol{e}$ be an error of weight t with syndrome of weight s, and let $\boldsymbol{\nu}$ be a fixed random near codeword. Let $u \stackrel{def}{=} |\boldsymbol{e} \star \boldsymbol{\nu}|$. This model follows the values (s, t, u) through the decoding process using a step by step decoder.*

Model 2 (M0). *Let $\boldsymbol{e}$ be an error of weight t with syndrome of weight s. This model follows the same values as Model 1, but the value u is set to 0 in the initial vector, so this model does not take into account the effect of a near codeword.*

Our model, which combines Models 1 and 2 according to the formula given in Section 5.2, is bound to see an error floor behavior: when the number of overlaps between an error vector and the near codeword becomes as large as possible (i.e., as u approaches d), decoding fails, and the probability that $u = d$ at the beginning of decoding is lower bounded by $\frac{t^d}{n^d}$ where t is the initial error weight. In the waterfall region, the DFR decays more than exponentially in n with other parameters fixed and therefore this $\frac{t^d}{n^d}$ term is going to dominate and provoke an error floor phenomenon.

In summary: we distinguish bad, suspicious, normal, and good bits (Definition 2.4). We estimate the probabilities π_0 and π_1 to take into account s, the syndrome weight, t the error weight, and u the size of the intersection of the error with the near codeword. Recall π_0 is the probability that a parity-check adjacent to a given bit which is a good bit is not satisfied, and π_1 is the probability that a parity-check adjacent to a given bit which is a normal bit in error is not satisfied. The transition probabilities are obtained under the assumptions described in the following section.

5.1 Assumptions

The forced cancellations are a result of the quasi-cyclic structure of $\boldsymbol{H}$. For the remainder of Section 5, we make the heuristic assumption that the columns of $\boldsymbol{H}$ are drawn uniformly at random up to fixed column weight d.

For convenience of counting arguments, the error vector $\boldsymbol{e}$ may be permuted so that the bad, suspicious, normal, and good bits are collected. Let $\mathbf{Q}$ denote this permutation matrix. Likewise, the bits of the syndrome $\boldsymbol{s}$ may be permuted to collect the 1's and 0's. Let $\mathbf{P}$ denote this permutation matrix. Applying these permutations to the parity check matrix $\boldsymbol{H}$ and flipping the canceling pairs yields a matrix $\mathbf{H}'_{\text{eff}} := \mathbf{P}\boldsymbol{H}\mathbf{Q}$. Details may be found in Appendix C. We use random variables to describe the columns of $\mathbf{H}'_{\text{eff}}$. Let $X_{\text{eff}} := \sum_{j \in \text{supp}(\boldsymbol{e})} \sigma_j - |\boldsymbol{s}|$. X_{eff} measures the deviation from the ideal case where exactly one error is involved in each unsatisfied parity-check equation.

Proposition 5.1. *Assuming that the random variables that indicate the number of errors $|\boldsymbol{H}_i \star \boldsymbol{e}|$ for a given equation i are independent and if $|\boldsymbol{e}| = t$, the expectation of X_{eff} knowing s, t, and $u = |\boldsymbol{e} \star \boldsymbol{\nu}|$ is*

$$E[X_{\text{eff}}|s, t, u] = s \cdot \frac{\sum_l 2l\rho_{2l+1}}{\sum_l \rho_{2l+1}}, \text{ where} \tag{5.1}$$

$$\rho_l = \sum_{j=0}^{l} \binom{t-u}{j}\binom{u}{l-j}\left(\frac{d}{r}\right)^j \left(\frac{r-d}{r}\right)^{t-u-j} \left(\frac{d+1-u}{r}\right)^{l-j} \left(\frac{r-d-1+u}{r}\right)^{u-l+j} \tag{5.2}$$

is the probability that any row in $\mathbf{H}'_{eff}$ restricted to the support of $\boldsymbol{e}$ has weight l.

Proof. For any row $i \in \{0, \ldots, r-1\}$ in $\mathbf{H}'_{\text{eff}}$, consider the t columns corresponding to the support of $\boldsymbol{e}$. There are $t-u$ columns corresponding to bits in $\text{supp}(\boldsymbol{e}) \setminus \text{supp}(\boldsymbol{\nu})$ and u columns corresponding to bits in $\text{supp}(\boldsymbol{e}) \cap \text{supp}(\boldsymbol{\nu})$. All columns in $\text{supp}(\boldsymbol{e}) \setminus \text{supp}(\boldsymbol{\nu})$ have weight d and all columns in $\text{supp}(\boldsymbol{e}) \cap \text{supp}(\boldsymbol{\nu})$ have weight $d+1-u$ due to forced cancellations. Any entry in row i on $\text{supp}(e) \setminus \text{supp}(\boldsymbol{\nu})$ has probability $\frac{d}{r}$ of being nonzero and probability $\frac{r-d}{r}$ of being zero. Any entry in row i on $\text{supp}(\boldsymbol{e}) \cap \text{supp}(\boldsymbol{\nu})$ has probability $\frac{d+1-u}{r}$ of being nonzero and probability $\frac{r-d-1+u}{r}$ of being zero.

Assumption 1. *The counters σ_j follow binomial distributions:*

$$\sigma_j \sim \begin{cases} \text{Bin}(d, \pi_0) & \text{if } j \text{ is a good bit} \\ \text{Bin}(d, \pi_1) & \text{if } j \text{ is a normal bit} \\ \text{Bin}(d-u, \pi_0) + \text{Bin}(u, \pi_1) & \text{if } j \text{ is a suspicious bit} \\ \text{Bin}(d-u+1, \pi_1) + \text{Bin}(u-1, \pi_0) & \text{if } j \text{ is a bad bit} \end{cases} \tag{5.3}$$

where

$$\pi_0 = \frac{s}{r}, \quad \pi_1 = \frac{s + \xi E[X_{eff}|s,t,u]}{s_{max}}, \quad s_{max} = d \cdot t - 2\binom{u}{2}$$

for some constant ξ.

Remark 5.2. The probability estimate π_0 in Assumption 1 is particularly pessimistic. Indeed, it is the probability that a *random* parity check is not satisfied, which is certainly an upper bound for the probability that a parity check corresponding to a bit which is not in error is unsatisfied. The model in Section 6 uses a more refined estimate of the probability π_0.

Remark 5.3. The constant ξ in Assumption 1 is inherited from a refinement given in [Vas21] of the previous Markov model [SV19b]. Reducing the value of ξ from 1 was intended to compensate for Non-Markovian effects relative to the state information (s,t) used in that model, which would otherwise produce an overly optimistic DFR. The experimental data in Figure 5.1 assumes the same value $\xi = 0.955$ used by [Vas21], which further contributes to the pessimism of Assumption 1 as reflected in our experimental data. There is no analogous factor to ξ used in Section 6.

Assumption 2. *The step by step bit flipping decoder is a time homogeneous Markov chain, i.e., for all $i \geqslant 1$, we have:*

$$\begin{aligned} &\Pr[(s_{i+1}, t_{i+1}, u_{i+1}) = (a_{i+1}, b_{i+1}, c_{i+1}) \big| \neg L_i, (s_i, t_i, u_i) = (a_i, b_i, c_i), \ldots, \neg L_0, (s_0, t_0, u_0))] \\ &= \Pr[(s_{i+1}, t_{i+1}, u_{i+1}) = (a_{i+1}, b_{i+1}, c_{i+1}) \big| \neg L_i, (s_i, t_i, u_i) = (a_i, b_i, c_i)], \end{aligned}$$

where $L_i := \{\sigma_j^{(i)} < T \ \ \forall j\}$ is the blocked state at the i-th iteration.

Markov end states. There are two possible Markov end states that dominate the experiment findings: $(0,0,0)$ when we have successful decoding or (d,d,d) when the state converges to a (d,d)-near codeword.

5.2 Markov Model Estimation of the DFR

Our simple Markov model experiments yield the decoding failure rate under the assumptions of models 2 and 1. Model 2 assumes no effect of the near codewords $\mathcal{N}$ on the decoding failure rate and Model 1 records the effect of a single near codeword on the decoding failure rate. Model 2 uses the same code but initializes the value of u to 0 in the initial vector.

Let DFR(M) and DFR(M0) denote the respective decoding failure rates of Model 1 and 2. From these two values, we wish to estimate the true decoding failure rate, which we denote DFR, taking into account the existence of $|\mathcal{N}| = r$ near codewords. The difference $\mathrm{DFR}(M) - \mathrm{DFR}(M0)$ accounts for the number of decoding failures due to the existence of one near codeword, so $|\mathcal{N}|(\mathrm{DFR}(M) - \mathrm{DFR}(M0))$ gives the increase in DFR due to the existence of $|\mathcal{N}|$ near codewords. This philosophy yields the formula:

$$\mathrm{DFR} = \mathrm{DFR}(M0) + |\mathcal{N}|(\mathrm{DFR}(M) - \mathrm{DFR}(M0)). \tag{5.4}$$

This assumes an additive effect of the elements of $\mathcal{N}$ on DFR, and we refer to Equation (5.4) as the *linear model*. See Figure 5.1. We compare our data with experimental data obtained using [Vas24], as well as a previous Markov model [SV19b] which does not take into account the effect of the error having a large number of overlaps with a near codeword. The Simple Markov model gives a more conservative estimate for the DFR, in instances where models not taking into account the effect of near codewords underestimate the DFR.

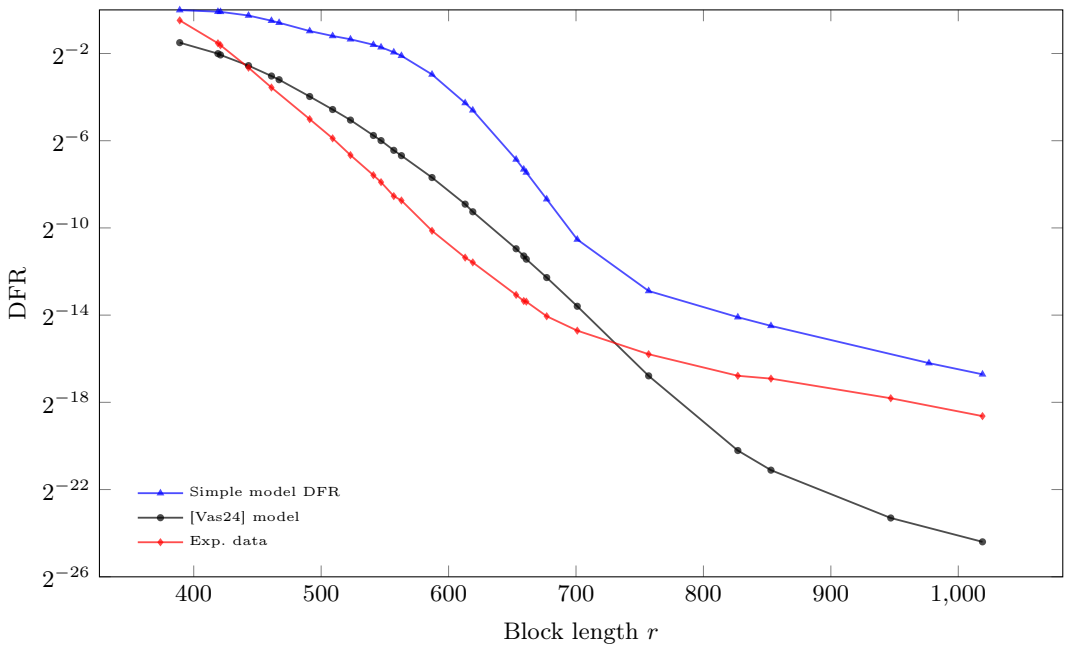


Fig. 5.1: Plot of experimental DFR data, Simple Markov model DFR, and the DFR of the Markov model in described in [SV19b].

6 A Key Dependent Markov Model

The simplified model does not depend on the secret key. However, experimental evidence shows that the DFR is definitely key dependent: Even if it does not vary much for typical keys, there are rare keys for which this DFR behaves better and weak keys which behave significantly worse. In

this section, we give a way to predict the iterative decoding performance which is key dependent. It makes use of the degree distribution of the subgraph $\mathscr{G}$ of the Tanner graph associated to the near codeword closest to $\boldsymbol{e}' + \boldsymbol{e}$ denoted in the whole section by $\boldsymbol{\nu}$.

6.1 Derivation of the Markov chain model

The state space. The state space is the set of (s, t, u, b) where $s = |\Delta \boldsymbol{s}|$ and $t = |\boldsymbol{e}+\boldsymbol{e}'|$ as before, $u = |(\boldsymbol{e} + \boldsymbol{e}') \star \boldsymbol{\nu}|$ and $b \in \{0, 1\}$ encodes whether or not $\boldsymbol{\nu}$ is $x^a h_0(x) \oplus 0$ (a *left near codeword*) or $0 \oplus x^b h_1(x)$ (a *right near codeword*). The structure of $\mathscr{G}$ depends on b. Its degree structure is given by the following proposition which follows immediately from Fact 2

Proposition 6.1. *The parity checks of even degree $2m$ in the subgraph $\mathscr{G}$ corresponding to a left near codeword are associated to sets of m different pairs $\{l_{a_1}, l_{b_1}\}, \cdots, \{l_{a_m}, l_{b_m}\}$ such that $l_{a_1} + l_{b_1} = \cdots = l_{a_m} + l_{b_m}$, where the a_i's and the b_i's all belong to $\{0, \cdots, d-1\}$ and $a_i \neq b_i$ for all $i \in \{1, \cdots, m\}$.*
The parity checks of odd degree $2m + 1$ in $\mathscr{G}$ are associated to an $a \in \{0, \cdots, d-1\}$ and sets of m different pairs $\{l_{a_1}, l_{b_1}\}, \cdots, \{l_{a_m}, l_{b_m}\}$ such that $2l_a = l_{a_1} + l_{b_1} = \cdots = l_{a_m} + l_{b_m}$, where $a_i, b_i \in \{0, \cdots, d-1\}$ and $a_i \neq b_i$ for all $i \in \{1, \cdots, m\}$.

A similar proposition holds for right near codewords where the r_i's replace the l_i's. We can partition the set of code positions in four sets, the bad bits, the suspicious bits, the normal bits in error and the good bits. We will simplify notation in what follows and label a parity check x^a simply by a. We assume from now on that the closest near-codeword $\boldsymbol{\nu}$ to the error intersects it in exactly u positions and that the error has weight t. We also assume that $\boldsymbol{\nu} = h_0(x) \oplus 0$. The case $\boldsymbol{\nu} = 0 \oplus h_1(x)$ is similar. The label $x^a \oplus 0$ of a variable node is also simplified here to a. In this section, we explain how we model the counters for each kind of bits. This will lead to Model 3 which is then used as explained in Section 4.

The case of bad bits. We have three kinds of parity checks adjacent to a bad bit. If the label of the bad bit is a_1, the labels of the other bad bits are $a_2, \cdots, a_u$, and the labels of the suspicious bits are $a_{u+1}, \cdots, a_d$, then we can group the parity check nodes adjacent to a_1. The first group consists of a single parity check node of odd degree in $\mathscr{G}$ labeled $2a_1$. The second group consists of $(u-1)$ parity check nodes labeled $a_1 + a_2, \cdots, a_1 + a_u$. The third group consists of $(d-u)$ parity check nodes labeled $a_1 + a_{u+1}, \cdots, a_1 + a_d$.

We do not know the label $a_1, \cdots, a_u$, and we model the probability that the corresponding parity checks are unsatisfied by using the following lemma.

Lemma 6.2. *Let a be a bad bit and let*

- *$\rho^{b,1}_{\Delta,\ell}$ be the probability that the parity check labeled $2a$ is adjacent to ℓ bits in error given that it is of degree Δ in $\mathscr{G}$;*
- *$\rho^{b,2}_{\Delta,\ell}$ be the probability that a parity check labeled $a + b$ of degree Δ in $\mathscr{G}$ is adjacent to ℓ bits in error given that it b is another bad bit;*
- *$\rho^{b,3}_{\Delta,\ell}$ be the probability that a parity check labeled $a + b$ is adjacent to ℓ bits in error given that it is of degree Δ in $\mathscr{G}$, that b is a suspicious bit;*

Then

$$\rho^{b,1}_{\Delta,\ell} = \frac{\sum_{j=1}^{\min(\Delta,\ell)} \binom{\Delta-1}{j-1}\binom{d-\Delta}{u-j}\binom{w-\Delta}{\ell-j}\binom{n-d-w+\Delta}{t-u-\ell+j}}{\binom{d-1}{u-1}\binom{n-d}{t-u}}$$

$$\rho^{b,2}_{\Delta,\ell} = \frac{\sum_{j=2}^{\min(\Delta,\ell)} \binom{\Delta-2}{j-2}\binom{d-\Delta}{u-j}\binom{w-\Delta}{\ell-j}\binom{n-d-w+\Delta}{t-u-\ell+j}}{\binom{d-2}{u-2}\binom{n-d}{t-u}}$$

$$\rho^{b,3}_{\Delta,\ell} = \frac{\sum_{j=1}^{\min(\Delta,\ell)} \binom{\Delta-2}{j-1}\binom{d-\Delta}{u-j}\binom{w-\Delta}{\ell-j}\binom{n-d-w+\Delta}{t-u-\ell+j}}{\binom{d-2}{u-1}\binom{n-d}{t-u}}.$$

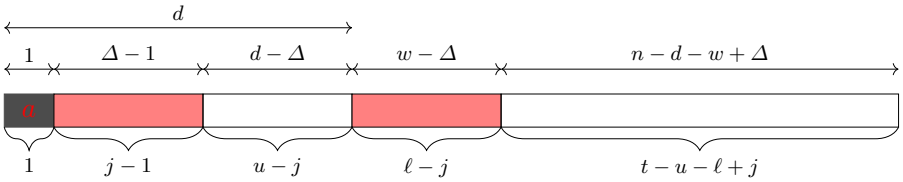


Fig. 6.1: The configuration of the error and the parity check. The black and pink are the positions of the parity check. The numbers on top are the block sizes. The numbers below are the number of errors in each block. The first block of size 1 is the bad bit a. The second block consists of the other positions involved in the parity check which are in the support of the near codeword $\boldsymbol{\nu}$. The third block consists of the positions in the support of $\boldsymbol{\nu}$ not involved in the parity check equation. The fourth block consists of the positions of the parity check which are not in the support of $\boldsymbol{\nu}$. The last block contains the remaining positions.

Proof. The reasoning is similar in all three cases.
The formula for $\rho^{b,1}_{\Delta,\ell}$. We compute the probability that an error of weight t which has weight u on the d bits of the near-codeword $\boldsymbol{\nu}$ and $t-u$ on the complement of the support of $\boldsymbol{\nu}$ intersects a given parity check of weight w in exactly ℓ positions. The parity check $2a$ involves the bad bit a we are interested in, plus $\Delta-1$ other bits of $\boldsymbol{\nu}$. If we denote by $\rho^{b,1}_{\Delta,\ell,j}$ the probability that this error is also of weight j on the Δ bits which belong at the same time to the support of the parity check labeled $2a$ and the support of $\boldsymbol{\nu}$, then we have

$$\rho^{b,1}_{\Delta,\ell,j} = \frac{\binom{\Delta-1}{j-1}\binom{d-\Delta}{u-j}\binom{w-\Delta}{\ell-j}\binom{n-d-w+\Delta}{t-u-\ell+j}}{\binom{d-1}{u-1}\binom{n-d}{t-u}}$$

This can be verified by counting the number of ways the $\binom{d-1}{u-1}\binom{n-d}{t-u}$ configurations of $t-1$ errors where there are $u-1$ errors among the $d-1$ bits of $\boldsymbol{\nu}$ other than the bad bit a we are interested in satisfy all the needed constraints. Figure 6.1 gives a picture of the configuration we are interested in. This yields the formula for $\rho^{b,1}_{\Delta,\ell}$ by summing over all possible values of j.
The formula for $\rho^{b,2}_{\Delta,\ell}$. We introduce $\rho^{b,2}_{\Delta,\ell,j}$, the probability that the error is also of weight j on the Δ bits which belong at the same time to the support of the parity check labeled $a+b$ and the support of $\boldsymbol{\nu}$. Then,

$$\rho^{b,2}_{\Delta,\ell,j} = \frac{\binom{\Delta-2}{j-2}\binom{d-\Delta}{u-j}\binom{w-\Delta}{\ell-j}\binom{n-d-w+\Delta}{t-u-\ell+j}}{\binom{d-2}{u-2}\binom{n-d}{t-u}}$$

Figure 6.2 displays the corresponding configuration. The difference with the previous case is that now the parity check labeled $a+b$ contains two bits which are in error by definition, namely a and b.

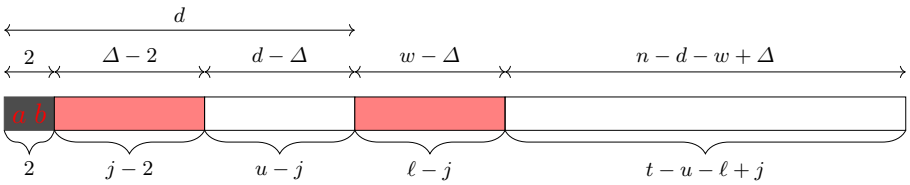


Fig. 6.2: The configuration of the error and the parity check, as in Figure 6.1. The first block of size 2 corresponds to bad bits a and b. The second block consists of the other positions involved in the parity check which are in the support of the near codeword $\boldsymbol{\nu}$. The last three blocks are as in Figure 6.1.

The formula for $\rho^{b,3}_{\Delta,\ell}$. This uses a similar reasoning, with the only difference that the parity check labeled $a+b$ contains by definition a bit which is in error, namely a and a bit which is not in error, namely b. □

Proposition 6.3. *For* $g \in \{1,2,3\}$*, let* $\overline{\pi^{b,g}}$ *be the average probability that a parity check of group* g *associated to a bad bit is unsatisfied. Then*

$$\overline{\pi^{b,1}} = \frac{1}{d}\sum_{a\in\boldsymbol{\nu}}\sum_{\ell}\rho^{b,1}_{\deg(2a),2\ell+1},$$
$$\overline{\pi^{b,g}} = \frac{1}{d(d-1)}\sum_{\substack{a\in\boldsymbol{\nu}\\ b\in\boldsymbol{\nu},\, b\neq a}}\sum_{\ell}\rho^{b,g}_{\deg(a+b),2\ell+1},\ \textit{for } g\geqslant 2,$$

where $\deg(c)$ *is the degree of the parity check node* c *in* $\mathscr{G}$*.*

Proof. The probability that the parity check labeled $2a$ associated to a bad bit a is unsatisfied is $\sum_\ell \rho^{b,1}_{\deg(2a),2\ell+1}$. The average is taken over all possible bits of the near codeword $\boldsymbol{\nu}$. The reasoning is the same for $\overline{\pi^{b,g}}$, the only difference is that we take the average over all possible edges leaving a variable node labeled a to a check node labeled $a+b$ for all possible b's different from a. □

The case of suspicious bits. The case of a suspicious bit is similar to the case of a bad bit. We partition the parity checks associated to it in three groups, as we did for a bad bit. There is a single parity check in the first group associated to a suspicious bit a, namely the parity check labeled $2a$. The second group contains $d-u-1$ other parity checks labeled $a+b$ where b is another suspicious bit. The last group contains u parity check bits labeled $a+b$ where b is a bad bit.

Lemma 6.4. *Let* a *be a suspicious bit and let*

- $\rho^{s,1}_{\Delta,\ell}$ *be the probability that the parity check labeled* $2a$ *in* $\mathscr{G}$ *is adjacent to* ℓ *bits in error given that it is of degree* Δ *in* $\mathscr{G}$*;*
- $\rho^{s,2}_{\Delta,\ell}$ *be the probability that a parity check labeled* $a+b$ *is adjacent to* ℓ *bits in error given that it is of degree* Δ *in* $\mathscr{G}$ *and* b *is another suspicious bit;*
- $\rho^{s,3}_{\Delta,\ell}$ *be the probability that a parity check labeled* $a+b$ *is adjacent to* ℓ *bits in error given that it is of degree* Δ *in* $\mathscr{G}$*, that* b *is a bad bit.*

Then
$$\rho^{s,1}_{\Delta,\ell} = \frac{\sum_{j=0}^{\min(\Delta,\ell)}\binom{\Delta-1}{j}\binom{d-\Delta}{u-j}\binom{w-\Delta}{\ell-j}\binom{n-d-w+\Delta}{t-u-\ell+j}}{\binom{d-1}{u}\binom{n-d}{t-u}}$$
$$\rho^{s,2}_{\Delta,\ell} = \frac{\sum_{j=0}^{\min(\Delta,\ell)}\binom{\Delta-2}{j}\binom{d-\Delta}{u-j}\binom{w-\Delta}{\ell-j}\binom{n-d-w+\Delta}{t-u-\ell+j}}{\binom{d-2}{u}\binom{n-d}{t-u}}$$
$$\rho^{s,3}_{\Delta,\ell} = \frac{\sum_{j=1}^{\min(\Delta,\ell)}\binom{\Delta-2}{j-1}\binom{d-\Delta}{u-j}\binom{w-\Delta}{\ell-j}\binom{n-d-w+\Delta}{t-u-\ell+j}}{\binom{d-2}{u-1}\binom{n-d}{t-u}}.$$

Proposition 6.5. *For* $g \in \{1,2,3\}$*, let* $\overline{\pi^{s,g}}$ *be the average probability that a parity check of group* g *associated to a suspicious bit is unsatisfied. Then*

$$\overline{\pi^{s,1}} = \frac{1}{d}\sum_{a\in\boldsymbol{\nu}}\sum_{\ell}\rho^{s,1}_{\deg(2a),2\ell+1}$$
$$\overline{\pi^{s,g}} = \frac{1}{d(d-1)}\sum_{\substack{a\in\boldsymbol{\nu}\\ b\in\boldsymbol{\nu},\, b\neq a}}\sum_{\ell}\rho^{s,g}_{\deg(a+b),2\ell+1},\ \textit{for } g\geqslant 2.$$

Here $\deg(c)$ *refers to the degree of the parity check node* c *in* $\mathscr{G}$*.*

The case of normal bits in error. We do not need to distinguish several groups of parity checks in this case.

Proposition 6.6. *Let $\rho^e_{\Delta,\ell}$ be the probability that a parity check node of degree Δ in the subgraph $\mathscr{G}$ of the Tanner graph induced by $\boldsymbol{\nu}$ and adjacent to a normal bit in error contains exactly ℓ erroneous bits. A parity check not in $\mathscr{G}$ is said to be degree of 0. We also let $\overline{\pi^e}$ be the average of these probabilities over all edges of the Tanner graph leaving the bits which are not in $\boldsymbol{\nu}$.*

$$\rho^e_{\Delta,\ell} = \frac{\sum_{j=0}^{\min(\Delta,\ell-1)} \binom{\Delta}{j}\binom{d-\Delta}{u-j}\binom{w-\Delta-1}{\ell-j-1}\binom{n-d-w+\Delta}{t-u-\ell+j}}{\binom{d}{u}\binom{n-d-1}{t-u-1}} \tag{6.1}$$

$$\overline{\pi^e} = \frac{1}{d(n-d)}\sum_{\Delta}(w-\Delta)n_\Delta \sum_{\ell}\rho^e_{\Delta,2\ell+1}, \tag{6.2}$$

where n_Δ is the number of check nodes in the subgraph $\mathscr{G}$ of the Tanner graph induced by $\boldsymbol{\nu}$ which are of degree Δ.

Proof. The reasoning for (6.1) is similar to what is done for Lemma 6.2 and is omitted. For computing the average of (6.1), observe that there are $d(n-d)$ edges in the Tanner graph leaving the $n-d$ positions which are not in $\boldsymbol{\nu}$. $\overline{\rho^e_{\Delta,\ell}}$ is nothing but the average probability over all edges that such an edge is adjacent to a check node which is not satisfied. If such a check node is of degree Δ in $\mathscr{G}$, then the probability that it is in error is exactly $\sum_\ell \rho^e_{\Delta,2\ell+1}$. There are exactly $(w-\Delta)n_\Delta$ edges of this kind that are adjacent to a check node of degree Δ in $\mathscr{G}$. This explains (6.2).

The case of good bits. This case is similar to the case of normal bits in error.

Proposition 6.7. *Let $\rho^g_{\Delta,\ell}$ be the probability that a parity check node of $\mathscr{G}$ of degree Δ adjacent to a good bit contains exactly ℓ erroneous bits. We also let $\overline{\pi^g}$ be the average of these probabilities over all edges of the Tanner graph leaving the bits which are not in $\boldsymbol{\nu}$. We have*

$$\rho^g_{\Delta,\ell} = \frac{\sum_{j=0}^{\min(\Delta,\ell)} \binom{\Delta}{j}\binom{d-\Delta}{u-j}\binom{w-\Delta-1}{\ell-j}\binom{n-d-w+\Delta}{t-u-\ell+j}}{\binom{d}{u}\binom{n-d-1}{t-u}} \tag{6.3}$$

$$\overline{\pi^g} = \frac{1}{d(n-d)}\sum_{\Delta}(w-\Delta)n_\Delta \sum_{\ell}\rho^g_{\Delta,2\ell+1}, \tag{6.4}$$

This analysis of bad, suspicious, normal, and good bits yields the following model:

Model 3. *We model the counters σ_i according to the type of bit:*

- *bad bit:* $\sigma_i \sim \mathrm{Bin}(1, \frac{s\overline{\pi^{b,1}}}{\mathbb{E}(s|t,u)}) + \mathrm{Bin}(u-1, \frac{s\overline{\pi^{b,2}}}{\mathbb{E}(s|t,u)}) + \mathrm{Bin}(d-u, \frac{s\overline{\pi^{b,3}}}{\mathbb{E}(s|t,u)})$
- *suspicious bit:* $\sigma_i \sim \mathrm{Bin}(1, \frac{s\overline{\pi^{s,1}}}{\mathbb{E}(s|t,u)}) + \mathrm{Bin}(d-u-1, \frac{s\overline{\pi^{s,2}}}{\mathbb{E}(s|t,u)}) + \mathrm{Bin}(u, \frac{s\overline{\pi^{s,3}}}{\mathbb{E}(s|t,u)})$
- *normal bit in error:* $\sigma_i \sim \mathrm{Bin}(d, \frac{s\overline{\pi^e}}{\mathbb{E}(s|t,u)})$
- *good bit:* $\sigma_i \sim \mathrm{Bin}(d, \frac{s\overline{\pi^g}}{\mathbb{E}(s|t,u)})$

where $\mathbb{E}(s|t,u)$ is given by:

$$\begin{aligned}\mathbb{E}(s|t,u) = \frac{1}{w}\Big(&u\overline{\pi^{b,1}} + u(u-1)\overline{\pi^{b,2}} + u(d-u)\overline{\pi^{b,3}} + (d-u)\overline{\pi^{s,1}} \\ &+ (d-u)(d-u-1)\overline{\pi^{s,2}} + (d-u)u\overline{\pi^{s,3}} + (t-u)d\overline{\pi^e} + (n+u-t-d)d\overline{\pi^g}\Big)\end{aligned}$$

6.2 Results

First, we checked this more general model on the same toy example of Section 5 with a random key (see Figure 6.3). The new model is not really a generalization of the one given in Section 5. The π_i's are estimated in this model applying to any key in a direct fashion by scaling the expected syndrome on each set of parity checks according to the whole syndrome weight. It gives a better behavior in the waterfall region than the previous model, based on Chaulet's approach for estimating the π_i's. We partition decoding failures both in the experiments and in the model in two parts: the decodings that failed because they converged to a near codeword ("Contr. of ncw") and the rest of the decoding failures (called "Contr. of other"). The models and the experiments agree remarkably well.

We also tested two different step-by-step decoders on the same code: one which uses the majority rule for the threshold, and another one which uses the decoder with two thresholds T_0 and T_1, see Figure 3.1. There is a first conservative threshold to flip the bits initially, and a second threshold (actually the majority rule) to flip the bits when there are no more bit flips to perform with the first rule. This decoder is much better than the majority decoder.

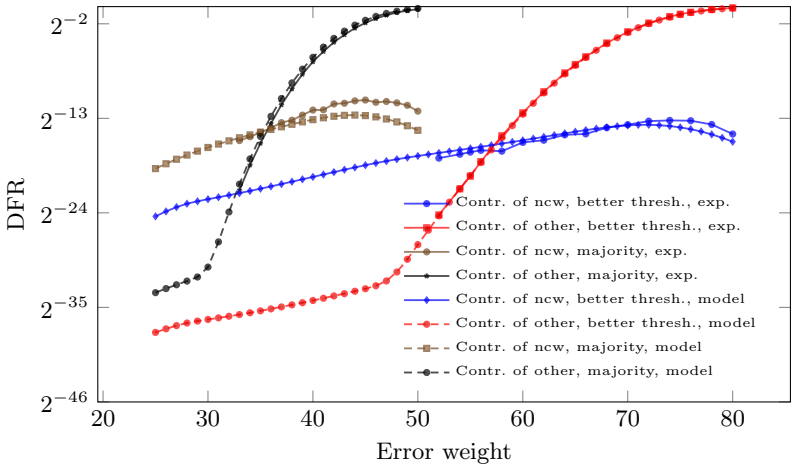


Fig. 6.3: DFR vs. error weight ($r = 1723$, $d = 17$), experiments vs. model.

In [Vas21] which compared parallel decoders to similar step-by-step decoders, the performance of the parallel decoders was much better than those of the step by step decoders. However, the step-by-step decoder with two thresholds used to draw Figure 6.3 is rather competitive with the parallel BGF decoder of BIKE, see Figure 6.4. This seems to confirm that there is a significant gain in choosing conservative thresholds in the early iterations, as observed in [Sen24]. The BGF decoder is slightly superior to the step-by-step decoder, with a somewhat steeper waterfall and a better error floor.

After all this experimental evidence, we ran the model of this section on a key chosen at random for the step-by-step decoder by choosing several different thresholds for various block sizes for $d = 71$ and $t = 134$. See Figure 6.5.

We checked the influence of the gap parameter δ used for decoding. Here the threshold T depends only on the syndrome weight s, the block length r, the error weight t and the gap parameter δ as follows

$$T = \min\left(d, \max\left(\lfloor A_{r,t}(s) + \delta \rfloor, \frac{d+1}{2}\right)\right), \tag{6.5}$$

where $A_{r,t}(s)$ is an affine function $\alpha \cdot s + \beta$ of s depending on r and t which is computed with the same method as for BIKE. For $t = 134$ and various values of r, the coefficients are given in Appendix B.2.

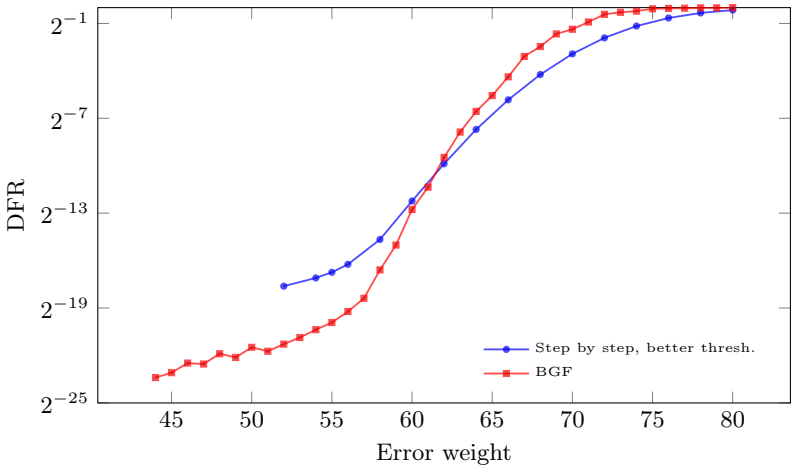


Fig. 6.4: DFR vs. error weight ($r = 1723$, $d = 17$), BGF vs. step by step.

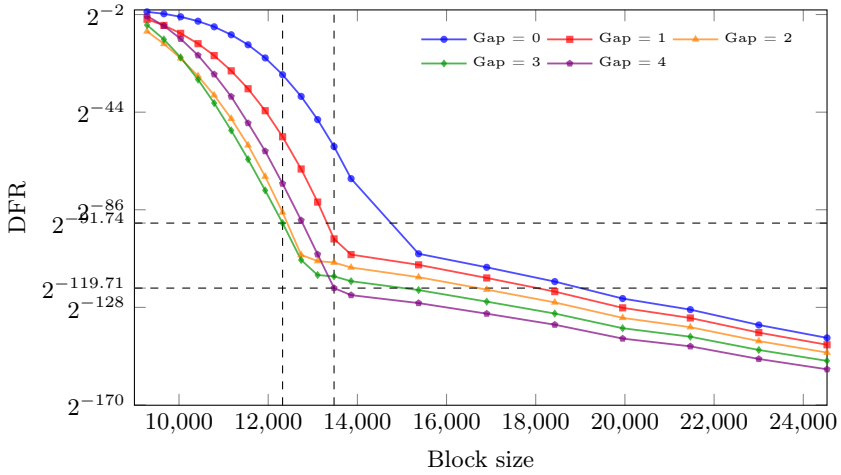
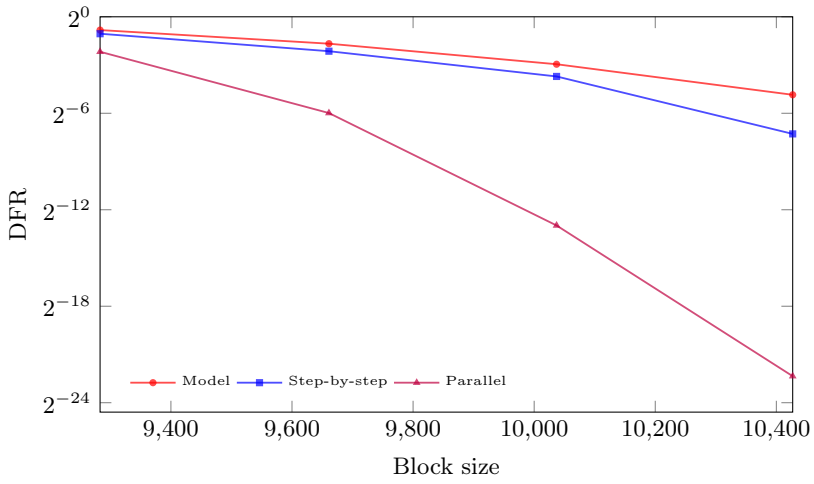


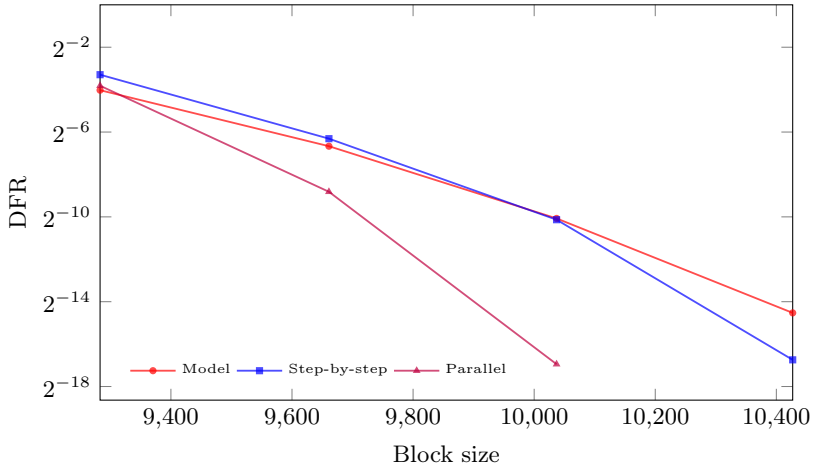
Fig. 6.5: DFR vs. block size ($d = 71$, $t = 134$).

Using the same gap parameter $\delta = 3$ as the one used in the latest BIKE decoder seems in the model to be optimal for the actual BIKE 1 parameter: $r = 12323$. It results in a DFR of about $2^{-91.7}$, unfortunately above $2^{-\lambda}$. Moreover, even if we increase the block length r we do not improve the DFR by much, because the error floor kicks in right after this value of r. We attain a DFR of $2^{-114.1}$ with this gap for $r = 13109$, but need to go to $r = 18427$ to go below $2^{-\lambda}$. It is $2^{-130.6}$ in this case. Using a gap δ equal to 4 allows to an error floor which appears a little bit later and allows to obtain a DFR which is about $2^{-119.6}$ for $r = 13477$. This r-value is less than 10 percent more than the actual BIKE parameter.

Experimental evidence for cryptographic parameters in the waterfall region. However there is a caveat here. It appears that taking a larger threshold function affects the accuracy of the waterfall prediction. Indeed, we have performed two kinds of experiments for values of r close to the BIKE 1 parameters and the same d and t as BIKE 1 so as to be able to observe experimentally the DFR. We are clearly in the waterfall region in this case. We have run the step by step decoder (with an infinite number of iterations so as to comply with the Markov chain modeling) and the parallel version of the decoder, where at each step after computing the syndrome we flip *all* bits above the threshold. The maximum number of iterations has been fixed to 100 in order to be comparable with the step by step decoder. All these experiments clearly show that the parallel version of the decoder outperforms significantly the step by step by step decoder. In particular it shows a much stronger decay than the step by step decoder. The experiments also show that the Markov chain modeling predicts rather well the step by step decoder (even if it becomes more pessimistic as the block size r increases) when the gap is equal 0 and 1. This is shown in Figures 6.6 and 6.7.



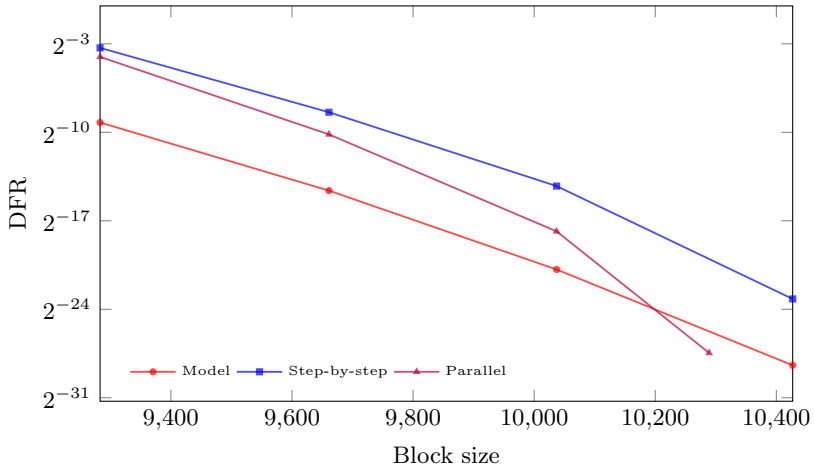
(a) Waterfall behavior for $d = 71$, $t = 134$, gap $\delta = 0$.



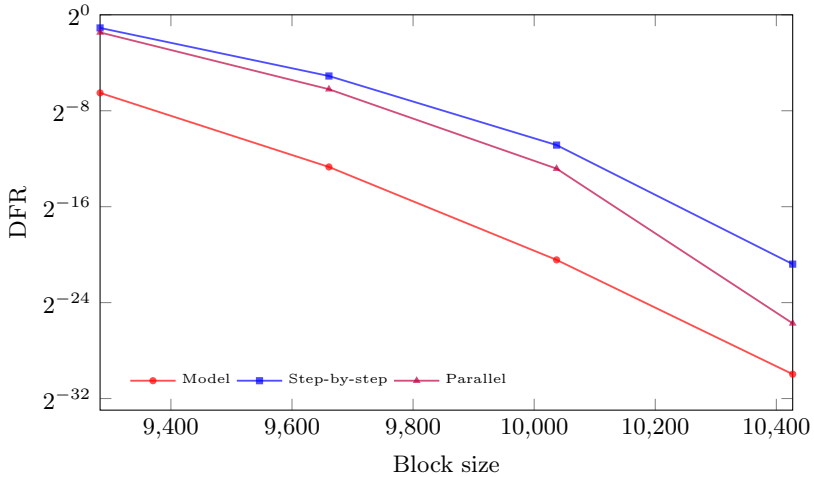
(b) Waterfall behavior for $d = 71$, $t = 134$, gap $\delta = 1$.

Fig. 6.6

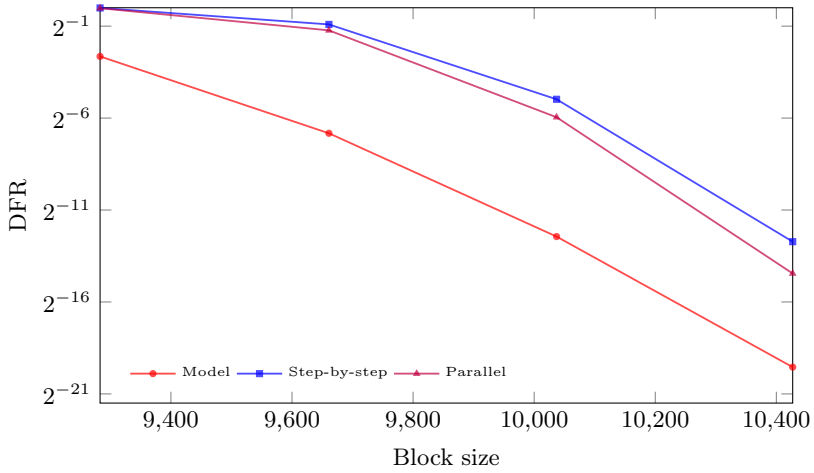
However, the Markov model becomes clearly optimistic in the waterfall region when the gap increases. The DFR is underestimated by about 6 bits for a gap $\delta = 2$ (Figure 6.8a), by 7 bits for a gap $\delta = 3$ (Figure 6.8b) and by about 6.5 bits for a gap $\delta = 4$ (Figure 6.8c). Notice that the experimental curve for the parallel decoder shows a steeper decline of the DFR which seems to catch up (and even go below) the Markov chain predictions for larger values of r. The reason for this too optimistic estimation of the DFR is very likely to be the fact that for larger thresholds we do not take into account in the model that the distribution of the counters of the normal bits in error (which are precisely the first one to get flipped because they have the highest expected counter value) gets truncated during the decoding process. This is because we decimate during it the highest counters. In any case, the waterfall behavior of the Markov model is too optimistic in the beginning of the waterfall region.



(a) Waterfall behavior for $d = 71$, $t = 134$, gap $\delta = 2$.



(b) Waterfall behavior for $d = 71$, $t = 134$, gap $\delta = 3$.



(c) Waterfall behavior for $d = 71$, $t = 134$, gap $\delta = 4$.

Fig. 6.7

Evidence that the error floor behavior is always captured accurately. To have a better understanding of the influence of the gap parameter on the Markov chain parameters we have performed a systematic study on the toy parameters, where we have taken $(r, d) = (1723, 17)$. First we have chosen $t = 65$ to be in the waterfall region. The results are displayed in Figure 6.9. There are several interesting features. We have separated the contribution of the DFR in two parts

$$\mathrm{DFR} = \mathrm{DFR}_{\mathrm{e}} + \mathrm{DFR}_{\mathrm{w}},$$

where $\mathrm{DFR}_{\mathrm{e}}$ gives the contribution to the DFR, where decoding failed because it converged to a near codeword. This contribution can be computed experimentally when $\mathrm{DFR}_{\mathrm{e}}$ is large enough. The Markov model can also give the corresponding contribution. The term $\mathrm{DFR}_{\mathrm{w}}$ (as "waterfall" contribution and "other" in legend of the figure) can be viewed as the waterfall contribution and is defined simply as $\mathrm{DFR} - \mathrm{DFR}_{\mathrm{e}}$. In Figure 6.9 we are in the waterfall region since $\mathrm{DFR} \approx \mathrm{DFR}_{\mathrm{w}}$ or what amounts to the same $\mathrm{DFR}_{\mathrm{e}} \ll \mathrm{DFR}_{\mathrm{w}}$. The Markov chain predictions are slightly pessimistic when δ is in the interval $[0, 1]$ but become optimistic in the case $\delta \in [1, 3]$. However, what is remarkable is that the predictions for the error floor term $\mathrm{DFR}_{\mathrm{e}}$ are in general really accurate for a gap bigger than 0.5. This strongly suggests that the Markov model predicts accurately the error floor behavior irrespective of the gap parameter.

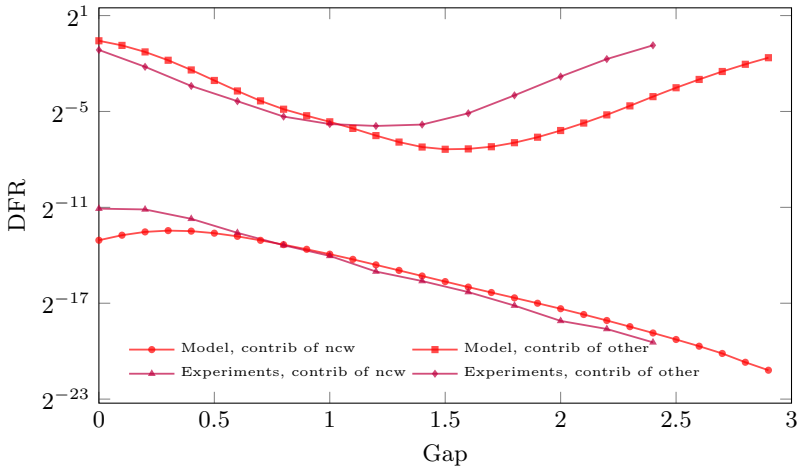


Fig. 6.8: Waterfall region $(r, d, t) = (1723, 17, 65)$

This has been further tested in Figure 6.10 with $(r, d, t) = (1723, 17, 35)$. This time we are clearly in the error floor since $\mathrm{DFR}_{\mathrm{w}} \ll \mathrm{DFR}_{\mathrm{e}}$. The predictions for the DFR are this time rather accurate (even if a little bit pessimistic). The experimental DFR and predicted DFR agree this time for all values of the gap parameter between 0 and 3. This shows the accuracy of the error floor prediction for our model.

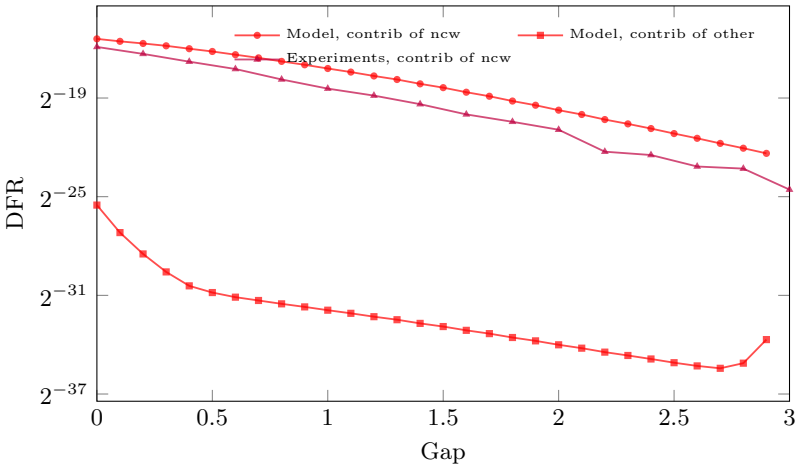


Fig. 6.9: Error floor region $(r, d, t) = (1723, 17, 35)$

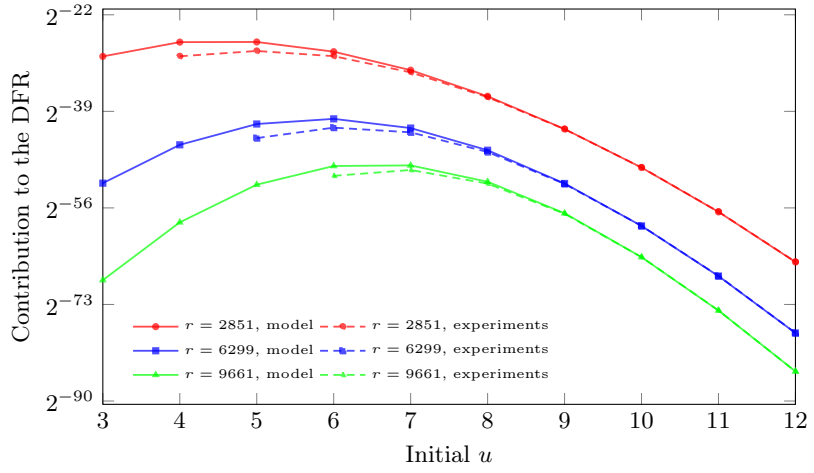
6.3 Further evidence for the error floor region

The rationale for extending experimental evidence beyond toy parameters. Notice that we can write the DFR as

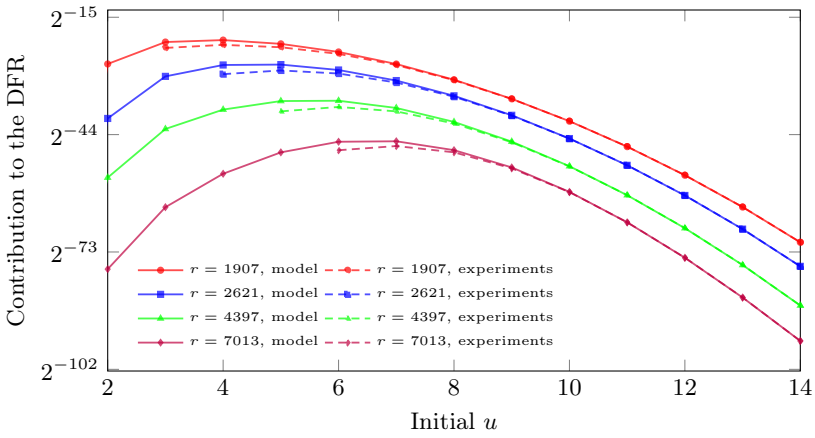
$$\mathrm{DFR} = \sum_{u=1}^{d} \mathrm{DFR}(u)p(u), \tag{6.6}$$

where $\mathrm{DFR}(u)$ is the DFR conditioned on the event that the closest near-codeword intersects the error in exactly u positions and $p(u)$ is the probability that this event occurs. We can tweak the initial distribution of the Markov chain so as to compute exactly $\mathrm{DFR}(u)$. In the case where the dominating terms in this sum correspond to u's for which $p(u)$ is small and the corresponding $\mathrm{DFR}(u)$ are large enough, we can run experiments conditioned that the error intersects the closest near-codeword in exactly u positions and check if $\mathrm{DFR}(u)$ is indeed predicted correctly. This allows to make such verification way beyond DFR's that we can measure experimentally, since we just have to verify $\mathrm{DFR}(u)$ experimentally and we may have $\mathrm{DFR} \ll \mathrm{DFR}(u)$. Of course, these experiments really make sense only in the error floor regime, since it is precisely in this case that the DFR is dominated by terms $\mathrm{DFR}(u)p(u)$ where $p(u)$ is small (and for unusually large values of u). Note that it is really the error-floor behavior that we want to verify.

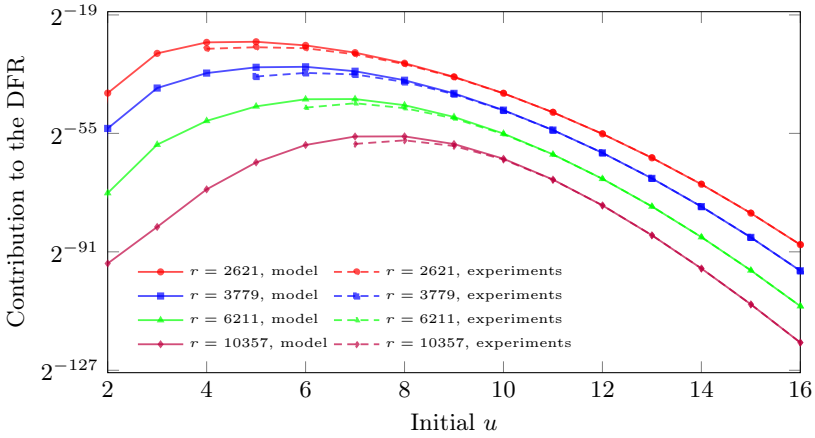
Experimental Evidence. This approach has permitted to expand significantly the regime of parameters that we can verify experimentally. We were able to increase the value of d from 17 to 27, t from 35 to 46 and r from 1723 to 17053. All our parameters that have been found are such that $\mathrm{DFR}(u_{\max})$, $\mathrm{DFR}(u_{\max} - 1)$ and $\mathrm{DFR}(u_{\max} + 1)$ are all verifiable experimentally. Here $u_{\max}$ is the value of u which maximizes the term $\mathrm{DFR}(u)p(u)$ in the decomposition (6.6). This provides compelling evidence that we indeed predict well the DFR with this method, since the experimental measured $f(u)$ is also maximal at $u_{\max}$. We have plotted the function $f(u) = \mathrm{DFR}(u)p(u)$ for all values of u for which we could verify $\mathrm{DFR}(u)$ experimentally. Finding parameters for small values of the gap is easier. This is why most of our parameters are for the gap parameter $\delta = 0$. We also give parameters for larger values of the gap, here $\delta = 2$. We have observed no noticeable difference when the value of the gap changes: in all cases we get a remarkable prediction of $f(u) = \mathrm{DFR}(u)p(u)$ for large values of u and a good prediction at $u_{\max}$ and $u_{\max} - 1$ which is just a little bit pessimistic in this case. All in all, these experiments permit to verify values of DFR that are as low as 2^{-80}.



(a) $(d, t, \delta) = (19, 33, 0)$

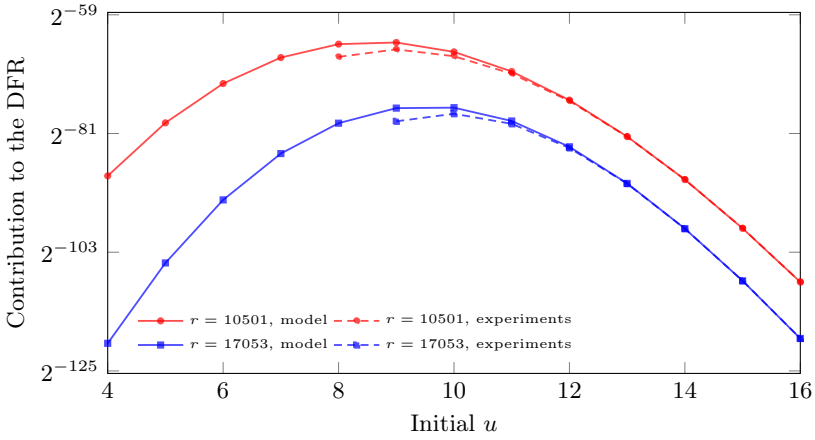


(b) $(d, t, \delta) = (21, 37, 0)$

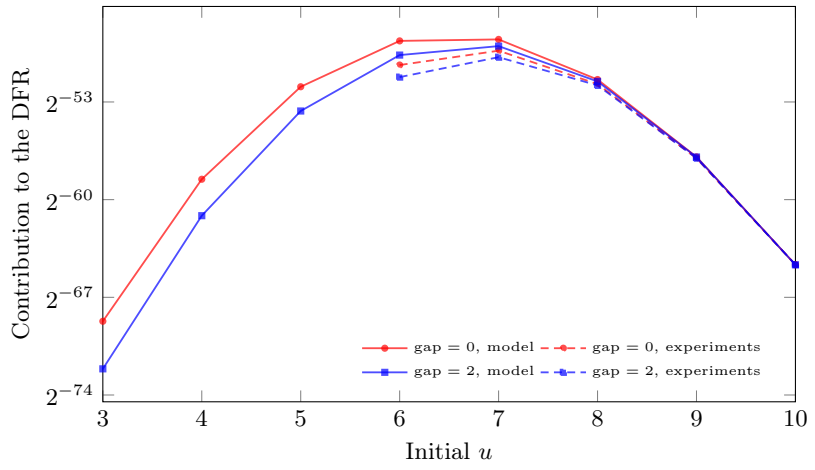


(c) $(d, t, \delta) = (23, 41, 0)$

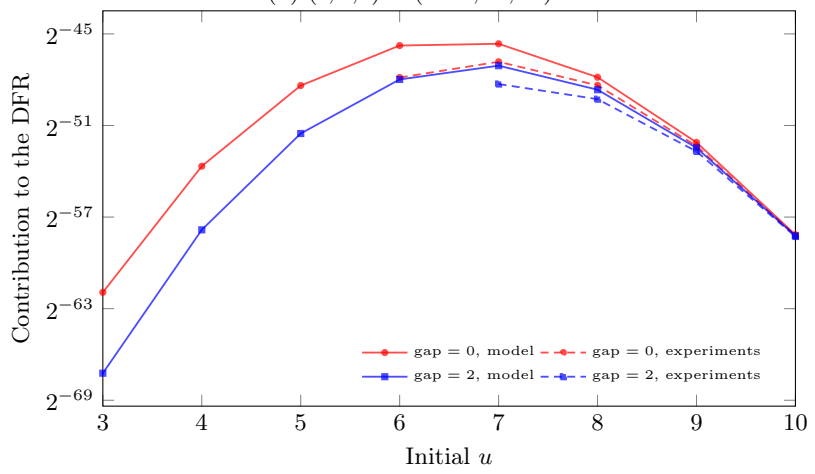
Fig. 6.10



(a) $(d, t, \delta) = (27, 46, 0)$



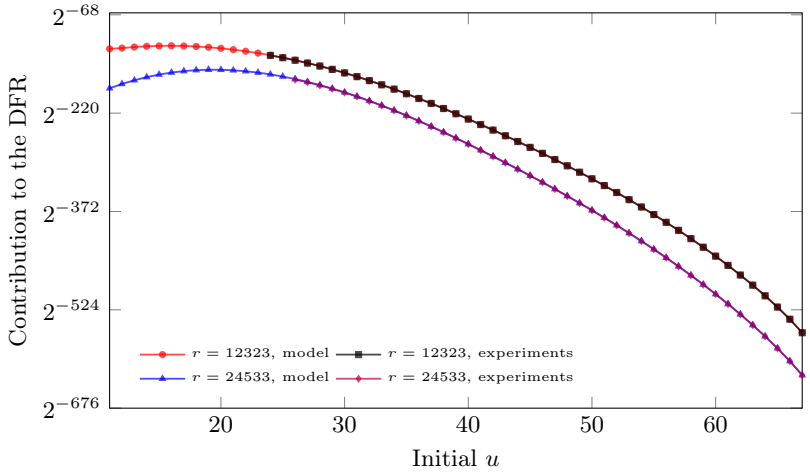
(b) $(r, d, t) = (9661, 19, 33)$



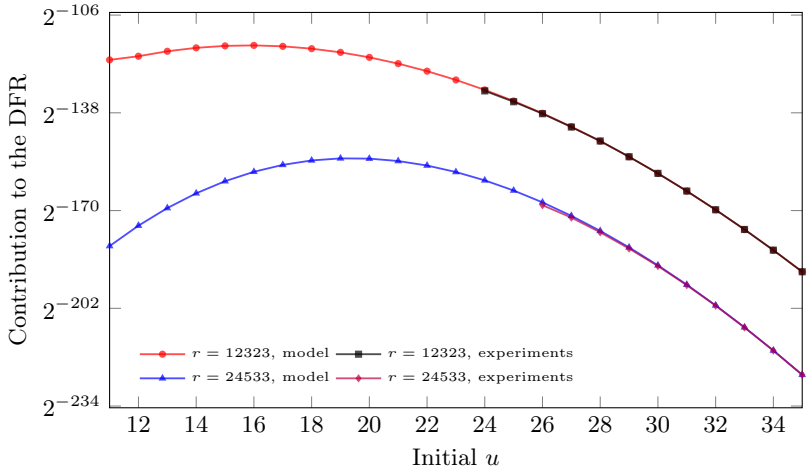
(c) $(r, d, t) = (7013, 21, 37)$

Fig. 6.11

We have also performed experiments for the cryptographic parameters. The Markov model predictions agree well for the DFR(u) that we were able to check.



(a) $(d, t, \delta) = (71, 134, 3)$



(b) Zoom on the previous curve, $(d, t, \delta) = (71, 134, 3)$

Fig. 6.12

6.4 Changing the threshold function

From now on, we use affine functions that are closer to the latest BIKE specification, as specified in Appendix B.2. A closer inspection of the affine function $A_{r,t}$ shows that for a large set of values of s (roughly between 0 and more than 70 per cent of the typical initial syndrome value) the threshold takes the value $\frac{d+1}{2}$. This seems to be a bad choice: the majority threshold rule maximizes the probability of flipping a suspicious bit. This leads to the convergence to a near-codeword and thus to the error floor phenomenon. Instead of (6.5), it is tempting to conjecture that the modified threshold function $T = \min(d, \max(A_{r,t}(s) + \delta, T_{\min}))$ might perform better, where $T_{\min} > \frac{d+1}{2}$. For the BIKE 1 parameters, $d = 71$ and $\frac{d+1}{2} = 36$. Choosing larger values for $T_{\min}$ has a significant effect on the error floor and does not deteriorate the waterfall behavior too much. We have studied this point in detail for a gap $\delta = 3$ and for $r = 13109$, as illustrated in the following table. To see the effect on the DFR we have decomposed the DFR as $\mathrm{DFR} = \mathrm{DFR_e} + \mathrm{DFR_w}$. $\mathrm{DFR_e}$ is the contribution of the DFR coming solely from the near-convergence to a near codeword. Experimental evidence shows that this term is very good for predicting the error floor. $\mathrm{DFR_w}$ is defined as $\mathrm{DFR} - \mathrm{DFR_e}$ and is good for predicting the waterfall behavior.

$T_{\min}$	$\mathrm{DFR_e}$	$\mathrm{DFR_w}$
36	-113.046457928823	-123.378559395570
37	-118.243581226345	-123.378559395558
38	-123.405489448735	-123.378559389356
39	-128.809000162163	-123.378537439281
40	-134.809143919309	-123.310820300205

Increasing $T_{\min}$ by 1, we gain roughly 5 bits on $\mathrm{DFR_e}$.

In order to lower the DFR as much as possible, we first optimized both $T_{\min}$ and δ, and found that $T_{\min} = 39$ and $\delta = 2.6$ reduce the DFR by about 2.5 bits, to $\mathrm{DFR_e} \approx 2^{-127.006}$, $\mathrm{DFR_w} \approx 2^{-126.777}$, $\mathrm{DFR} \approx 2^{-125.887}$. Now let us note that the decoder behavior only depends on the floor of the threshold function, and thus on the intervals $[a_i, b_i[$ on which $\lfloor T \rfloor = i$ for $T_{\min} \leqslant i \leqslant d$. Therefore, we can further optimize the DFR by adjusting the bounds of these intervals. In order to simplify the situation, we upper bound the thresholds by 50 instead of $d = 71$, which does not affect the DFR. With the modified intervals shown on Figure 6.14 and specified in Appendix B.2, we obtain $\mathrm{DFR_e} \approx 2^{-127.358}$, $\mathrm{DFR_w} \approx 2^{-128.248}$, $\mathrm{DFR} \approx 2^{-126.736}$, an improvement of about 0.9 bit. We have obtained this DFR reduction by performing some kind of gradient descent optimization of the endpoints of the intervals. It has ended up with irregular intervals.

For $r = 13477$, with $T_{\min} = 39$ and $\delta = 3.4$ we obtain $\mathrm{DFR_e} \approx 2^{-131.719}$, $\mathrm{DFR_w} \approx 2^{-133.131}$ and $\mathrm{DFR} \approx 2^{-131.259}$. For $r = 12323$, we did not optimize the DFR to the same extent as for $r = 13109$ but we fine-tuned $T_{\min}$ and δ. For $T_{\min} = 36$ and $\delta = 2.47$, we obtain $\mathrm{DFR_e} \approx 2^{-109.754}$, $\mathrm{DFR_w} \approx \mathrm{DFR} \approx 2^{-96.663}$, which is an improvement of about 4.3 bits.

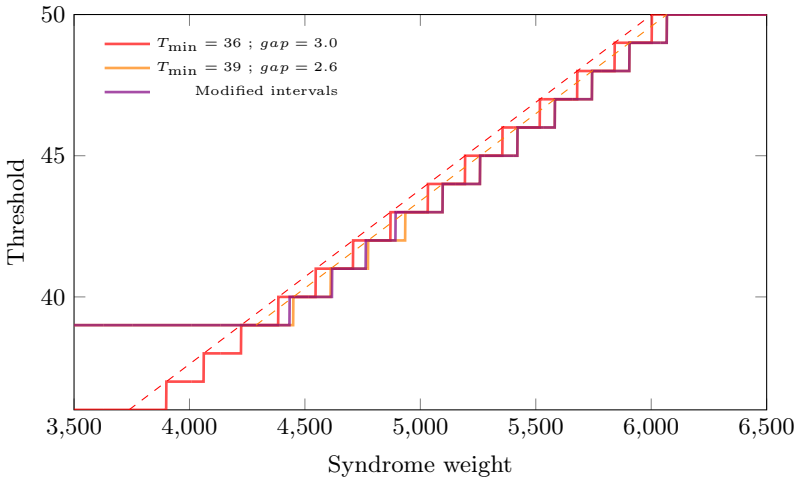


Fig. 6.13: Syndrome weight intervals for the choice of threshold, with $r = 13109$

7 Concluding Remarks

A new accurate Markov model. This work highlights the central role that the near codewords put forward in [Vas21] play in the error floor regime. We enrich the Markov model of [SV19b,Vas21] with the size of the intersection of the error with a near codeword. Using a model which is key-dependent, we obtain an accurate prediction of the DFR in the error-floor regime, matching experimental data for various decoders and keys.

Attaining a DFR below $2^{-\lambda}$. If we run this Markov model on the BIKE level 1 parameters with the former threshold function, we fall short of reaching a DFR of about 2^{-91}, which is below $2^{-\lambda}$. This problem cannot simply be addressed by taking slightly larger parameters, because the error floor kicks in just after the block size $r = 12323$ chosen for BIKE 1. However, this error floor

seems to be due to the fact that the threshold function is equal to its minimum possible value $\frac{d+1}{2}$ for a very large range of values of the syndrome weight s. This can be addressed adequately by increasing this minimum threshold value from 36 to 39 in the case of BIKE 1. In this case, a threshold function with an adjusted gap leads to a DFR for the step-by-step decoder which is below $2^{-131.2}$ for a block size $r = 13477$. This is slightly less than a 10% increase of the current BIKE 1 parameter.

Analyzing a parallel decoder. There are reasons to believe [Vas21] that the parallel decoders used in the actual BIKE implementation produce an even lower DFR. Whereas up to now, only the beginning of the waterfall region could be explored for cryptographic parameters, this work opens a new road for exploring the ability of various decoders to attain even more efficiently the needed $2^{-\lambda}$ DFR and for understanding the effect of various weak keys on the DFR. Concerning the first point, it could be interesting to consider the approach followed in [ABP24a] (for the full version see [ABP24b]) to analyze parallel decoders.

References

ABB+21. Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Santosh Ghosh, Shay Gueron, Tim Güneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Jan Richter-Brockmann, Nicolas Sendrier, Jean-Pierre Tillich, Valentin Vasseur, and Gilles Zémor. BIKE: Bit flipping key encapsulation - spec v4.2. `https://bikesuite.org/files/v4.2/BIKE_Spec.2021.07.26.1.pdf`, 2021.

ABH+22. Sarah Arpin, Tyler Raven Billingsley, Daniel Rayor Hast, Jun Bo Lau, Ray A. Perlner, and Angela Robinson. A study of error floor behavior in QC-MDPC codes. In Jung Hee Cheon and Thomas Johansson, editors, *Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022, Virtual Event, September 28-30, 2022, Proceedings*, volume 13512 of *Lecture Notes in Computer Science*, pages 89–103. Springer, 2022.

ABH+24. Sarah Arpin, Tyler Raven Billingsley, Daniel Rayor Hast, Jun Bo Lau, Ray Perlner, and Angela Robinson. A graph-theoretic approach to analyzing decoding failures of BIKE. Cryptology ePrint Archive, Paper 2024/1736, 2024.

ABP24a. Alessandro Annechini, Alessandro Barenghi, and Gerardo Pelosi. Bit-flipping decoder failure rate estimation for $(v, w)-$regular codes. In *Proc. IEEE Int. Symposium Inf. Theory - ISIT*, pages 3374–3379. IEEE, 2024.

ABP24b. Alessandro Annechini, Alessandro Barenghi, and Gerardo Pelosi. Bit-flipping decoder failure rate estimation for $(v, w)-$regular codes. *CoRR*, abs/2401.16919, 2024.

BBC+21. Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. Performance bounds for QC-MDPC codes decoders. In Antonia Wachter-Zeh, Hannes Bartz, and Gianluigi Liva, editors, *Code-Based Cryptography - 9th International Workshop, CBCrypto 2021, Munich, Germany, June 21-22, 2021 Revised Selected Papers*, volume 13150 of *Lecture Notes in Computer Science*, pages 95–122. Springer, 2021.

DGK20. Nir Drucker, Shay Gueron, and Dusan Kostic. QC-MDPC decoders with several shades of gray. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography*, pages 35–50. Springer, Cham, 2020.

Gal63. Robert G. Gallager. *Low Density Parity Check Codes*. M.I.T. Press, Cambridge, Massachusetts, 1963.

GJS16. Qian Guo, Thomas Johansson, and Paul Stankovski. A key recovery attack on MDPC with CCA security using decoding errors. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016*, volume 10031 of *LNCS*, pages 789–815, 2016.

HB18. Yoones Hashemi and Amir H. Banihashemi. Characterization of elementary trapping sets in irregular LDPC codes and the corresponding efficient exhaustive search algorithms. *IEEE Trans. Inf. Theory*, 64(5):3411–3430, 2018.

HHK17. Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In *Theory of Cryptography Conference*, pages 341–371. Springer, 2017.

NSP+23. Mohammad Reza Nosouhi, Syed W. Shah, Lei Pan, Yevhen Zolotavkin, Ashish Nanda, Praveen Gauravaram, and Robin Doss. Weak-key analysis for BIKE post-quantum key encapsulation mechanism. *IEEE Trans. Inf. Forensics Secur.*, 18:2160–2174, 2023.

Ric03. Tom Richardson. Error floors of LDPC codes. In *Proc. of the 41th Annual Allerton Conf. on Communication, Control, and Computing*, volume 41, pages 1426–1435, 2003.

Sen24. Nicolas Sendrier. BIKE decoding failure – weak keys & error floors. Slides of the panel discussion of the fifth PQC standardization conference, April 2024.

SV19a. Nicolas Sendrier and Valentin Vasseur. About low DFR for QC-MDPC decoding. Cryptology ePrint Archive, Paper 2019/1434, 2019. `https://eprint.iacr.org/2019/1434`.

SV19b. Nicolas Sendrier and Valentin Vasseur. On the decoding failure rate of QC-MDPC bit-flipping decoders. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography 2019*, volume 11505 of *LNCS*, pages 404–416, Chongquing, China, May 2019. Springer.

Tan81. Robert Michael Tanner. A recursive approach to low complexity codes. *IEEE Trans. Inform. Theory*, 27(5):533–547, 1981.

Vas21. Valentin Vasseur. *Post-quantum cryptography: a study of the decoding of QC-MDPC codes*. Phd thesis, Université de Paris, March 2021.

Vas24. Valentin Vasseur. qcmdpc_markov Github codebase, 2024. Accessed: 2024-08-26.

VCN14. Bane Vasić, Shashi Kiran Chilappagari, and Dung Viet Nguyen. Chapter 6 - failures and error floors of iterative decoders. In David Declerq, Marc Fossorier, and Ezio Biglieri, editors, *Academic Press Library in Mobile and Wireless Communications*, pages 299–341. Academic Press, Oxford, 2014.

WWW23. Tianrui Wang, Anyu Wang, and Xiaoyun Wang. Exploring decryption failures of BIKE: New class of weak keys and key recovery attacks. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, volume 14083 of *LNCS*, pages 70–100. Springer, 2023.

A Computing the initial state distribution

Let $\boldsymbol{H}$ be the parity check matrix for a QC-MDPC code of row weight d where the degree distributions n_Δ are known for both blocks. [8] Consider the following variables: the syndrome weight s, error weights in blocks 0 and 1 denoted by t_0 and t_1 (with total error weight $t = t_0 + t_1$), maximal numbers of intersections between the error vector and any near codeword in blocks 0 and 1 denoted by u_0 and u_1 (with $u = \max(u_0, u_1)$) and $b \in \{0, 1\}$ indicating which block contains the closest near codeword (i.e., $b = 0$ if $u_0 > u_1$ and $b = 1$ if $u_1 > u_0$).

With BIKE, the initial decoding conditions involve an error vector of fixed weight t chosen uniformly at random. While t is thus known, determining the joint distribution of (s, u, b) is non-trivial as it depends on the structure of the code, particularly its regularity but also the degree distribution of the Tanner graph given by n_Δ.

Our goal here is therefore to compute the joint distribution $\mathbf{P}(s, u, b|t)$. We decompose this computation where first we compute $\mathbf{P}(u_0, u_1|t_0, t_1)$, which we marginalize to obtain $\mathbf{P}(u, b|t)$. We then compute $\mathbf{P}(s|t, u, b)$. The desired distribution is obtained as

$$\mathbf{P}(s, u, b|t) = \mathbf{P}(u, b|t)\,\mathbf{P}(s|t, u, b)\,.$$

A.1 Distribution $\mathbf{P}(u, b|t)$

For two same-sized blocks of length r, the probability of partitioning t errors between them, with t_0 errors in block 0 and t_1 errors in block 1, follows a hypergeometric distribution: $\mathbf{P}(t_0, t_1) = \frac{\binom{r}{t_0}\binom{r}{t_1}}{\binom{n}{t}}$.

Within each block i, the probability of intersection size k with any particular near codeword follows a hypergeometric distribution where the probability mass function is: $f_i(k) = \frac{\binom{d}{k}\binom{r-d}{t_i-k}}{\binom{r}{t_i}}$. Let $F_i(k)$ denote its cumulative distribution function: $F_i(k) = \sum_{j=0}^{k} f_i(j)$.

According to order statistics theory, since we consider the maximum over r independent such variables, the CDF of the maximum is $[F_i(k)]^r$. Therefore, the probability mass function of the maximum intersection in block i is:

$$\mathbf{P}(u_i|t_i) = [F_i(u_i)]^r - [F_i(u_i - 1)]^r$$

The joint distribution combines three mutually exclusive cases: when both blocks have maximum intersection u, when the maximum intersection u occurs uniquely in block 0, when the maximum intersection u occurs uniquely in block 1.

$$\mathbf{P}(u_0, 0|t) = \sum_{t_0+t_1=t} \mathbf{P}(t_0, t_1) \left(\tfrac{1}{2}\,\mathbf{P}(u_0|t_0)\,\mathbf{P}(u_0|t_1) + \sum_{u_1<u_0} \mathbf{P}(u_0|t_0)\,\mathbf{P}(u_1|t_1) \right)$$

$$\mathbf{P}(u_1, 1|t) = \sum_{t_0+t_1=t} \mathbf{P}(t_0, t_1) \left(\tfrac{1}{2}\,\mathbf{P}(u_1|t_0)\,\mathbf{P}(u_1|t_1) + \sum_{u_0<u_1} \mathbf{P}(u_1|t_1)\,\mathbf{P}(u_0|t_0) \right)$$

A.2 Distribution $\mathbf{P}(s|t, u, b)$

To compute the conditional probability $\mathbf{P}(s|t, u, b)$, we must account for both the number of check nodes connected to an odd number of errors (which determines the syndrome weight s) and the

[8] Remember that n_Δ is the number of check nodes in the subgraph $\mathscr{G}$ of the Tanner graph induced by a near codeword $\boldsymbol{\nu}$ which are of degree Δ.

total number of edges connected to errors in the Tanner graph (which must equal $d \cdot t$). The structure of the code, particularly the degree distributions $n_{\Delta}^{(b)}$ of check nodes in the subgraph $\mathscr{G}$ for block b, plays an important role in this computation.

We need to track the parity of errors connected to each check node while we ensure that the total number of edges connected to errors sums to $d \cdot t$. For this, we use a generating function that encodes both these quantities.

For each check node of degree Δ in block b, let

$$\rho_{\Delta,\ell}^{(b)} := \frac{\sum_{j=0}^{\min(\Delta,\ell)} \binom{\Delta}{j}\binom{d-\Delta}{u-j}\binom{w-\Delta}{\ell-j}\binom{n-d-w+\Delta}{t-u-\ell+j}}{\binom{d}{u}\binom{n-d}{t-u}}$$

denote the probability that a check node of degree Δ in $\mathscr{G}$ involves exactly ℓ errors, where these errors are distributed between: the u bits common to both the error vector and nearest near codeword and the remaining $t-u$ error bits not in the near codeword. The generating function for a single check node of degree Δ is then defined as:

$$G_{\Delta}^{(b)}(x,y) = \sum_{\ell} \rho_{\Delta}^{(b)}(\ell) \cdot x^{\ell \bmod 2} \cdot y^{\ell},$$

where y^{ℓ} tracks the number of errors connected to the check node, and $x^{\ell \bmod 2}$ encodes the parity of ℓ (thus x^1 indicates a syndrome bit equal to 1 and x^0 indicates a syndrome bit equal to 0).

For the $n_{\Delta}^{(b)}$ check nodes of degree Δ in block b, their collective contribution is given by $(G_{\Delta}^{(b)}(x,y))^{n_{\Delta}^{(b)}}$. The full generating function for block b is thus:

$$G^{(b)}(x,y) = \prod_{\Delta} \left(G_{\Delta}^{(b)}(x,y)\right)^{n_{\Delta}^{(b)}}.$$

This generating function encodes all possible configurations of errors and their parities across check nodes in block b, while respecting the degree structure of the subgraph $\mathscr{G}$ under the assumption that parity check equations are independent. However, we know that the regularity of the code induces a constraint on the total number of edges connected to errors in the Tanner graph.

The joint probability of observing s check nodes with odd parity and a total of $d \cdot t$ edges connected to errors is given by the coefficient of $x^s y^{d \cdot t}$ in $G^{(b)}(x,y)$, denoted $[x^s y^{d \cdot t}]\, G^{(b)}(x,y)$. The conditional probability $\mathbf{P}(s|t,u,b)$ is then obtained by normalizing over all possible syndrome weights:

$$\mathbf{P}(s|t,u,b) = \frac{[x^s y^{d \cdot t}]\, G^{(b)}(x,y)}{\sum_{s'} [x^{s'} y^{d \cdot t}]\, G^{(b)}(x,y)}.$$

Efficient computation of the distribution. While the generating function provides a theoretical framework, direct computation using this approach would be inefficient, as it would require calculating probabilities for all possible configurations, only to select a small subset where the edge count equals $d \cdot t$. We instead use a more practical computational approach.

Let us decompose the syndrome weight s into components by degree, where s_{Δ} denotes the number of degree-Δ parity check equations in $\mathscr{G}$ that have an odd number of errors, for $\Delta = 0, 1, \ldots, \Delta_{\max}$. Thus, $s = \sum_{\Delta=0}^{\Delta_{\max}} s_{\Delta}$. The computation begins by constructing vectors for each degree Δ that map the number of errors ℓ to the unnormalized probability $\rho_{\Delta,\ell}^{(b)}$, separated into odd and even components. For each possible value $s_{\Delta} \in \{0, \ldots, n_{\Delta}^{(b)}\}$, we compute s_{Δ} convolutions of the odd-error vector and $(n_{\Delta}^{(b)} - s_{\Delta})$ convolutions of the even-error vector. These results are then divided by $s_{\Delta}!$ and $(n_{\Delta}^{(b)} - s_{\Delta})!$ respectively[9] The next step involves performing convolutions first within each Δ group to obtain the distribution of total edge count for each possible value of

[9] These factorial divisions account for the multinomial coefficient that counts the number of ways to partition s into $\Delta_{\max} + 1$ components.

s_Δ. We then convolve across different Δ groups to compute the joint distribution of the sum of s_Δ values and their corresponding edge count distributions. Finally, we filter to retain only the configurations where the total edge count equals $d \cdot t$, and normalize to obtain the final probability distribution.

B Details on threshold rules

B.1 Computation of the affine part coefficients

Recall that we use thresholds of the form

$$T = \min\left(d, \max\left(\lfloor A_{r,t}(s) + \delta \rfloor, \frac{d+1}{2}\right)\right)$$

where $A_{r,t}(s) = \alpha_{r,t} \cdot s + \beta_{r,t}$.

In this subsection, we explain the procedure to compute these coefficients $\alpha_{r,t}$ and $\beta_{r,t}$. This is based on the approach from [**?**] which is used in [ABB+21] and the following binomial counter model. The counter values of bits in error (resp. not in error) are namely assumed to follow a binomial distribution $\mathrm{Bin}(d, \pi_1)$ (resp. $\mathrm{Bin}(d, \pi_0)$) where :

$$\pi_1 = \frac{s+X}{dt} \text{ and } \pi_0 = \frac{(w-1)s - X}{d(n-t)}.$$

Here X is a certain function of s and t which is related to the random variable $Y \stackrel{\text{def}}{=} \sum_{j \in e} \sigma_j - s$. Depending on the version of BIKE [ABB+21], X is either given by

$$\begin{aligned} \text{version V1 (before Oct. 2022):} \quad & X \stackrel{\text{def}}{=} \mathbb{E}(Y) = r \sum_l 2l \cdot \rho_{2l+1} \\ \text{version V2 (Oct. 2024):} \quad & X \stackrel{\text{def}}{=} \frac{\mathbb{E}[Y]}{\mathbb{E}[s]} s = s \frac{\sum_l 2l \cdot \rho_{2l+1}}{\sum_l \rho_{2l+1}} \\ \text{where} \quad & \rho_l \stackrel{\text{def}}{=} \frac{\binom{w}{l}\binom{n-w}{t-l}}{\binom{n}{t}}. \end{aligned}$$

Therefore, for $0 \leqslant \sigma \leqslant d$:

$$\mathbb{E}[|\{j \in \boldsymbol{e} \mid \sigma_j = \sigma\}|] = t\binom{d}{\sigma}\pi_1^\sigma(1-\pi_1)^{d-\sigma} \tag{B.1}$$

$$\mathbb{E}[|\{j \notin \boldsymbol{e} \mid \sigma_j = \sigma\}|] = (n-t)\binom{d}{\sigma}\pi_0^\sigma(1-\pi_0)^{d-\sigma} \tag{B.2}$$

We define the function T_0 by :

$$T_0(s) = \frac{\log\left(\frac{n-t}{t}\right) + d\log\left(\frac{1-\pi_0(s)}{1-\pi_1(s)}\right)}{\log\left(\frac{\pi_1(s)}{\pi_0(s)}\right) + \log\left(\frac{1-\pi_0(s)}{1-\pi_1(s)}\right)}$$

$T_0(s)$ is chosen such that, for $\sigma \geqslant T_0(s)$, among the bits whose counter is σ, we expect to flip more bits in error that not in error, i.e. $\mathbb{E}[|\{j \in \boldsymbol{e} \mid \sigma_j = \sigma\}|] \geqslant \mathbb{E}[|\{j \notin \boldsymbol{e} \mid \sigma_j = \sigma\}|]$. T_0 is defined here for positive real values of s and $T_0(s)$ corresponds to the real value of σ for which the right-hand terms in (B.1) and (B.2) coincide.

Then we define $A_{r,t}$ by:

$$A_{r,t}(s) = T_0(\mathbb{E}[s]) + T_0'(\mathbb{E}[s]) \cdot (s - \mathbb{E}[s])$$

i.e.

$$\alpha_{r,t} = T_0'(\mathbb{E}[s]) \text{ and } \beta_{r,t} = T_0(\mathbb{E}[s]) - \alpha_{r,t}\mathbb{E}[s]$$

Note that the graph of $A_{r,t}$ is the tangent to the curve of T_0 at point $(\mathbb{E}[s], T_0(\mathbb{E}[s]))$.

B.2 Details on the model computations of Subsection 6.2

Table B.1: Affine part $A_{r,t}$ (V1) of the threshold function corresponding to Figure 6.5 6.6 6.7 6.8a 6.8b 6.8c 6.13a 6.13b. The threshold $T_r(s)$ is given for a block length r and a syndrome weight s by the formula $T_r(s) = \min\left(d, \max\left(\lfloor A_{r,t}(s) + \delta \rfloor, \frac{d+1}{2}\right)\right)$.

r	α	β
9283	0.007835394510038963	13.297180113728302
9661	0.00739310530092027 4	13.567391558872282
10037	0.007061848501098616	13.594659657463222
10289	0.006883993637624069	13.520590051799362
10427	0.006798500986468654	13.456553521076303
10789	0.006606617361836905	13.228860262727327
11171	0.006444309408601018	12.921143566520255
11549	0.0063141209528076135	12.572979649160397
11933	0.006205362314938619	12.192085979992001
12323	0.0061135837435673315	11.789323238365313
12739	0.00603182965107241	11.351398154970589
13109	0.005970219758260547	10.96007467501192
13477	0.00591732887186274 2	10.572456366568613
13859	0.0058697311301313835	10.173985156386603
15373	0.005733664525418103	8.655332501782194
16901	0.00565184187736275	7.233670336326141
18427	0.005604464463216208	5.915303733756538
19949	0.005580554453148446	4.683511294611625
21467	0.0055736761566337325	3.521321141011274
23003	0.005579882422098863	2.399420709618189
24533	0.005596416334069782	1.32559949672806

Table B.2: Coefficients of the affine part $A_{r,t}$ (V2) of the threshold function for figures 6.9 6.10

t	α	β
35	0.018258736281959287	2.6333251606224053
65	0.008490421212064447	6.592815448465522

Table B.3: Coefficients of the affine part $A_{r,t}$ (V2) of the threshold function for figures 6.11a 6.11b 6.11c 6.12a 6.12b 6.12c

r	d	t	α	β
2851	19	33	0.022496109582380977	-0.31520011389499025
6299	19	33	0.03208537235574351	-7.516590006034072
9661	19	33	0.04113979857756371	-13.578711997396
1907	21	37	0.017594633411850264	3.477612429106303
2621	21	37	0.018705456845101318	1.4280333460625592
4397	21	37	0.02205136991866539	-2.4420447265246334
7013	21	37	0.027305118972899983	-7.246375579783971
2621	23	41	0.016505660253078373	2.577358096338255
3779	23	41	0.017907602055643648	-0.0325178801759094
6211	23	41	0.021411794519924247	-4.476148584279839
10357	23	41	0.027577185118666955	-11.035680668050869
10501	27	46	0.02174043010171235	-9.178324290764118
17053	27	46	0.028256124734199098	-17.95017858186116

Table B.4: Affine part $A_{r,t}$ (V2) of the threshold function used as a starting point for optimization, closer to the latest BIKE specification

r	α	β
12323	0.0062549227639326485	11.101167408537833
13109	0.006180462568160698	9.8998430883248
13477	0.006147235585082764	9.39527217136597

Table B.5: Upper bounds b_i of the threshold intervals shown on fig 6.14

i	$T_{\min} = 36$; $\delta = 3.0$	$T_{\min} = 39$; $\delta = 2.6$	Modified intervals
36	3900	0	0
37	4062	0	0
38	4224	0	0
39	4385	4450	4434
40	4547	4612	4618
41	4709	4774	4764
42	4871	4935	4893
43	5033	5097	5097
44	5194	5259	5259
45	5356	5421	5421
46	5518	5583	5583
47	5680	5744	5744
48	5842	5906	5906
49	6003	6068	6068

C Simple Model: Permutations of the parity check matrix

Fix a parity check matrix $\boldsymbol{H} \in \mathbb{F}_2^{r \times n}$ Apply permutations $\mathbf{P} \in \mathbb{F}_2^{r \times r}, \mathbf{Q} \in \mathbb{F}_2^{n \times n}$ to $\boldsymbol{H}, \boldsymbol{e}$ and $\boldsymbol{s}$ so that $\boldsymbol{H}' = \mathbf{P}\boldsymbol{H}\mathbf{Q}^{-1}$, $\boldsymbol{e}' = \boldsymbol{e}\mathbf{Q}^{\intercal}$, and $\boldsymbol{s}'^{\intercal} = \mathbf{P}\boldsymbol{s}^{\intercal}$ corresponding to Figure C.1. Let $S := |\boldsymbol{s}|$. When $|\mathbf{B}| = S$, there is exactly one error per unsatisfied parity-check equation. This is the ideal case for error correction as flipping a bit in $\boldsymbol{e}$ from $1 \to 0$ will not affect the other counters.

Let $X = \sum_{j \in e} |\boldsymbol{H}_j \star \boldsymbol{s}| - S$. The ideal case above corresponds to $X = 0$.

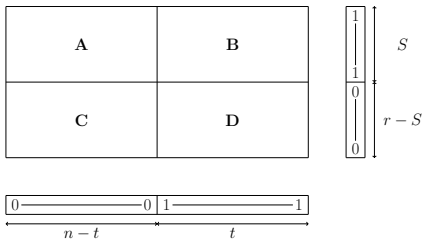


Fig. C.1: $\mathbf{H}'$ matrix, obtained by permuting rows and columns of $\mathbf{H}$. Below, we have $\boldsymbol{e}'$ from rearranging the entries of $\boldsymbol{e}$, and likewise $\boldsymbol{s}'$ to the right, computed from $\mathbf{H}'\boldsymbol{e}'^{\intercal}$.

Construct the matrix $\mathbf{H}'_{\text{eff}}$ by permuting columns of $\boldsymbol{H}'$ according to the support of $\boldsymbol{\nu}$ as pictured in Figure C.2. Since $\boldsymbol{H}'$ has been permuted, we likewise have permuted $\boldsymbol{\nu}$ and $\boldsymbol{e}'$, which have u nonzero overlaps in the $\mathbf{B}$ and $\mathbf{D}$ sections of $\mathbf{H}'$. In particular, there will be u columns in the $\mathbf{B}, \mathbf{D}$ sections of $\mathbf{H}'$. In the rows of these columns, we will find the pairs of 1's which are forced to cancel by the algebraic structure of the circulant blocks. We flip these pairs from 1's to 0's. A total of $\binom{u}{2}$ pairs of 1's will be flipped: Together, blocks $\mathbf{B}$ and $\mathbf{D}$ of $\mathbf{H}'$ have weight $d \cdot t$. After the cancellations, $\mathbf{H}'_{\text{eff}}$ will have blocks $\mathbf{B}_{\text{eff}}$ and $\mathbf{D}_{\text{eff}}$ which will be weight $S_{\max} := d \cdot t - 2\binom{u}{2}$.

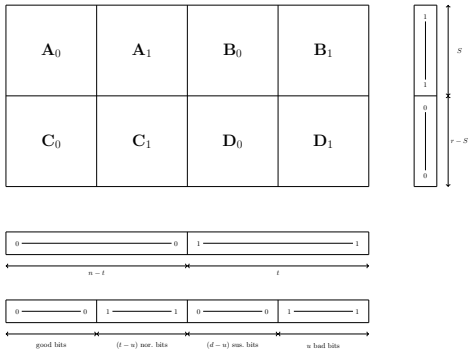


Fig. C.2: $\mathbf{H}_{\text{eff}}$ matrix, obtained by permuting rows and columns of $\mathbf{H}$ according to the intersection of $\boldsymbol{e}$ and $\boldsymbol{\nu}$.

D A useful partition of the parity checks

In order to understand the probabilistic model for the counters it is helpful to partition the parity checks in 6 groups. To simplify the discussion we assume that we are in the ideal case when all the $l_i + l_j$'s are different when $i \leqslant j$.

P_1 : The set of parity checks which are of degree 1 in the subgraph $\mathscr{G}$ induced by the closest near codeword and which are adjacent to a bad bit.

P_2 : The set of parity checks in $\mathscr{G}$ of degree 2 and are adjacent to two bad bits.
P_3 : The set of parity checks in $\mathscr{G}$ of degree 2 and are adjacent to one bad bit and one suspicious bit.
P_4 : The set of parity checks in $\mathscr{G}$ of degree 2 and are adjacent to two suspicious bits.
P_5 : The set of parity checks which are of degree 1 in the subgraph $\mathscr{G}$ and which are induced by the dominant near codeword and which are adjacent to a suspicious bit.
P_6 : The rest of the parity checks (namely those that do not belong to $\mathscr{G}$).

In the ideal case we have

$$|P_1| = u,\ |P_2| = \binom{u}{2},\ |P_3| = u(d-u),\ |P_4| = \binom{d-u}{2},\ |P_5| = d-u.$$

We define a red edge in the Tanner graph as an edge connecting a bit to a syndrome bit which is equal to 1. A syndrome bit equal to 1 in the Tanner is represented by a red parity check equation. Figure D.1 depicts this partition of bits and parity checks.

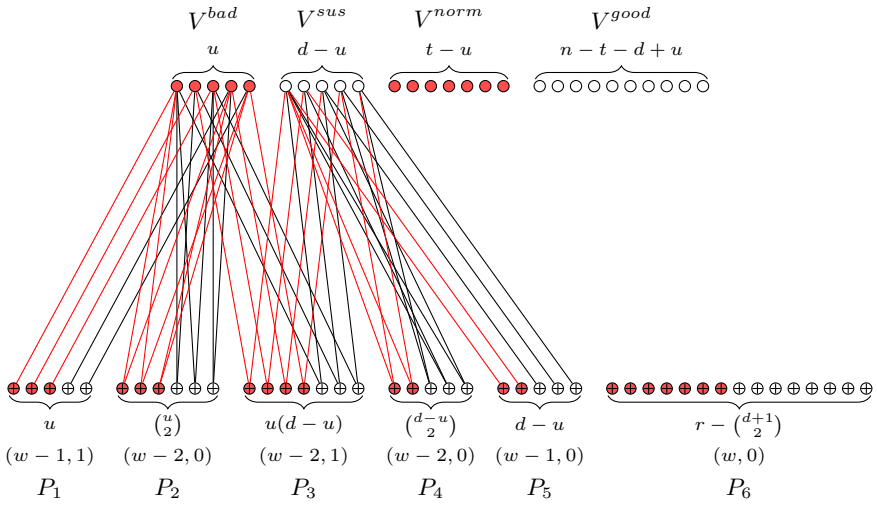


Fig. D.1: Illustration of the different groups. The size of the group is given just below the group and just below the size of the group, there is a pair (a,b) which gives the type of the parity-bit. a gives the number of additional edges arriving at this group coming from the set of bits which do not belong to the Tanner graph $\mathscr{G}$ induced by the closest near codeword, namely $V^{\text{norm}} \cup V^{\text{good}}$. b indicates the contribution to the parity check belonging to the group coming solely from the bits in $\mathscr{G}$. For instance for the first group P_1, the size is u and the corresponding pair is $(w-1,1)$.

The reason why we decompose the parity checks in several groups comes from the fact that they behave differently during decoding depending on their type. What we call *type* of a parity check is the pair (a,b) where a is the number of edges arriving at this parity check from bits not belonging to $\mathscr{G}$. b is a bit which indicates the contribution to the syndrome of parity check coming from bits in the subgraph $\mathscr{G}$ of the Tanner graph induced by the closest near codeword. All groups of parity checks have different types, with the exception of P_2 and P_4 which both have type $(w-2,0)$. P_2 has type $(w-2,0)$ because a parity check belonging to it is adjacent to exactly 2 bits in $\mathscr{G}$ and the contribution to the syndrome is 0 because it is adjacent to 2 bits in error in $\mathscr{G}$. P_4 has the same type because it is adjacent to 2 bits which are correct in $\mathscr{G}$ and their contribution to the syndrome is also 0.

Let us see how the counters are impacted by the near codeword $\boldsymbol{\nu}$. Say that u positions among $\boldsymbol{\nu}$ are in error. Let π_1 be the probability that a parity check of a position in error is not satisfied. Let π_0 denote the probability that a parity check of a position not in error is not satisfied. In

essence, (hard decision) iterative decoding of LDPC/MDPC codes relies on the fact that $\pi_1 > \pi_0$. The counters of positions in error are generally higher than the counters of positions which are not in error. We expect that the counter of a position which is in error to be about $d\pi_1$ whereas the counter of a position which is not in error is expected to be about $d\pi_0$. Now let us look at the u positions which are in error in the near codeword (V^{bad}): Their counter is expected to be about $\pi_1 + (u-1)\pi_0 + (d-u)\pi_1 = (u-1)\pi_0 + (d-u+1)\pi_1$ which is typically much less than $d\pi_1$. This is due to the fact that there are $(u-1)$ parity checks adjacent to such a bit which are adjacent to another position which is in error (P_2), canceling the contribution of the error to this position (and these parity checks behave roughly as a parity check adjacent to a position which is not in error). These positions are therefore less likely to be corrected by the iterative decoding process than the other positions which are in error. But the situation is even worse for the $d-u$ positions which belong to the support of the near codeword, but which are not in error (V^{sus}). For those positions a similar reasoning now conduces to model such a counter by the sum of $d-u$ Bernoulli random variables of parameter π_0 (from $P_4 \cup P_5$) and u Bernoulli variables of parameter π_1 (from P_3). We therefore expect that the counter of such positions to be about $(d-u)\pi_0 + u\pi_1$. Those positions are therefore more likely to be wrongly flipped than the other positions which are not in error.

In the general case, check nodes of the Tanner graph $\mathscr{G}$ induced by the closest near codeword can have degrees greater than 2. This is what the values of n_Δ account for in Section 6.