On Error Probability of CRC/Polar Codes with List Decoding

Stephen G. Wilson  
Department of Electrical and Computer Engineering  
University of Virginia  
Charlottesville, VA, 22904, USA  
sgw@virginia.edu

Kumud S. Altmayer  
Member IEEE  
ksa2te@virginia.edu

Abstract—Error performance of polar codes on the AWGN channel is discussed in conjunction with list decoding with or without an outer CRC code. The two mechanisms for block error are formulated, called type I (list misses) and type II (list hits but wrong vector selected). Two slightly-different ways of utilizing the CRC code are described, having identical type I error probability, but differing type II error probability dependent primarily on CRC code length. Results are presented for rate 1/2 polar codes and antipodal modulation with blocklengths 256 and 2048. The cases presented illuminate design tradeoffs on the selection of list size, CRC length, and especially the issue of final list sorting.

Index Terms—polar code, list decoding, CRC

I. INTRODUCTION

Polar codes, a class of linear block codes invented by Arikan [1], are the first provably capacity-achieving codes on a binary-input memoryless channel with an explicit construction and with a low decoding complexity. Though capacity-achieving as blocklength grows, even with simple successive cancellation decoding, finite blocklength performance is inferior to turbo codes and LDPC (low density parity-check) codes with the same blocklength.

It was shown in [2] that it’s possible to improve the performance of polar codes by using a successive cancellation list (SCL) decoder. The SCL decoder with list size $L$ does a tracking of a list of $L$ candidate codewords, and then picks the best and most probable path as its decision during the final stage of the decoding. By adding an outer CRC code, and using this to select from the inner decoder’s list, performance can be improved to be competitive with ‘best’ comparable codes, at modest complexity. The outer CRC code can also have the benefit of increasing the minimum distance of the concatenated code, [3, 4].

In this work we study performance on the binary antipodal AWGN channel for rate 1/2 coding approaches, with short and medium blocklengths. The emphasis is on the particular contributions of various events leading to block error and their dependence on signal-to-noise ratio, decoder list size, CRC length, if any, and the role of list sorting. While we focus on classic polar codes, the presentation generalizes to variations, including Reed-Muller/polar codes [5] and others that follow the polarization and successive decoding paradigm.

The paper is organized as follows. In Section II, polar codes codes are reviewed. In section III we study polar codes without CRC, formulating the error types, and in section IV the results are extended to use of CRC outer codes. Conclusions are provided in Section V.

II. BACKGROUND

A. Polar Codes

We assume familiarity with the literature on polar codes, first proposed by Arikan [1], including code construction and successive cancellation decoding. For notation we let $N = 2^n$ denote the blocklength of the code, $R_{in} = K_{in}/N$ the code rate of the polar code, and $L$ the decoder’s list size. In polar coding an underlying message vector $u$ contains $K_{in}$ unfrozen bits, and $N - K_{in}$ frozen bits, commonly set to 0, although message-dependent choice of frozen bit state can offer some gains, [6]. Unfrozen (message-carrying) bit locations are chosen by some sort of selection process in the design of the code, typically based on mutual information of the $N$ bit channels, conditioned on previous correct bit positions.

The transmitted codeword $x$ is generated by

$$x = uF^\otimes n$$  \hspace{1cm} (1)

where $F = [1\ 0; 1\ 1]$ is the kernel of the polarizing transformation, and $F^\otimes n$ denotes the $n$-fold Kronecker product of $F$.

In contrast with the original bit-by-bit successive cancellation decoding of Arikan, list decoding [2] is normally supported by an error-detecting outer code, typically a low-overhead CRC code, that can usually locate the correct message vector, if it is on the list produced by the inner polar list decoder. While the simplest decoder in such situations follows the conventional checking list candidates for zero CRC remainder, superior performance at a given CRC overhead is obtained by adding list sorting to CRC checking, as seen below. Such a decoder is capable of approaching the performance of maximum likelihood decoding for the concatenated code, as list size becomes sufficiently large and the CRC code has sufficiently small undetected error probability.

B. Related work on list decoding

Though polar coding was shown to be capacity-achieving for binary symmetric memoryless channels, the approach to
capacity with blocklength is relatively slow, and bit-by-bit decoding is inferior to good LDPC codes, say, with equal rate and blocklength. First, [2] showed that list decoding could improve the situation so that performance is competitive with other well-known error control techniques, while providing an attractive complexity/performance tradeoff.

Murata et al [3] built upon this by studying the effect of CRC code selection on block error probability, for a fixed list size of 32. There the tradeoff on CRC length becomes evident–too short means the undetected error probability is large and false positives degrade the performance, but too long a CRC code implies an energy penalty, shifting the performance curve to the right in SNR. Murata et al showed there is a weaker dependence on the particular code polynomial of a given length as well. Reference [4] has also examined the impact of CRC selection.

A study in [7] is also relevant to this work, focusing on the coding gain versus length of CRC code and the tradeoffs involved in this paper, without specific separation of decoding error mechanisms.

More broadly, Seshadri and Sundberg [8] have studied the benefits of list Viterbi decoding of convolutional codes, when supplemented with a reliable error-detecting outer code. Their analysis shows that in the low $P_e$ region, the asymptotic gain in SNR relative to the case of $L = 1$ on the same code is

$$G = 10 \log_{10}(\frac{2L}{L+1}) \text{ dB} \tag{2}$$

which approaches 3 dB for large $L$. Even small values of $L$ provide remarkable gain, e.g. $L = 4$ projects about 2 dB asymptotic gain.

III. LIST DECODING WITHOUT CRC CODE

To establish the framework for the rest of the paper, we first consider list decoding, but without a CRC code.

Decoding of polar codes is based on efficient sequential computation of the posterior probability of unfrozen bits in the underlying message vector, derived from all the measurements of the noisy bits under the assumption that previously-decided bits are correct. This is referred to as successive cancellation (SC) decoding. The computation operates in a highly-regular graph of $2 \times 2$ butterflies having breadth $N$ and depth $n = \log_2 N$, and can be efficiently implemented in a depth-first traversal of a binary tree of depth $n$, providing $O(N \log_2 N)$ complexity.

In contrast to the original bit-by-bit successive cancellation decoder of Arikan, the list decoder introduced by the authors in [2] sequentially builds a size-$L$ list of candidate message vectors having largest cumulative metrics to the current bit position in the estimate of $u$, with $L$ being a key design variable. Decoding begins with deciding the first unfrozen bit among the $K_{in} = NR_{in}$ unfrozen bits contained in $u$. As the decoding proceeds, skipping over known frozen bit positions, the decoder forms two extensions of each previous list member, forms the cumulative metric to date for these $2L$ candidates, and by a sort on metrics preserves the best $L$ vectors for the next cycle. This continues until the message vector estimates have length $N$. Reference [9] provides a comprehensive description of efficient list decoding in the LLR domain.

The performance advantage of list decoding rests on the possibility to avoid early dismissal of the eventually-correct codeword that a bit-by-bit decoder may produce. Since polar codes are linear codes, on the binary Gaussian channel we can assume the message vector $u$ is the all-0's vector, an $N$-tuple including frozen bit positions set to 0. The list decoder will fail to include the all-0 vector on its final list if at some decision stage $k$ the prefix vector (all 0's as well) does not survive this extend-and-sort cycle, for once the all-0 prefix is flushed from the candidate list, it can never reappear later.

Once the list is built at the end of decoding, we have $L$ $N$-tuples as candidate codewords. All are valid codewords from among the $2^{NR_{in}}$ codewords of the polar code. Final choice of the decoded vector (again, no-CRC help as yet) is based on a final sort of the $K_{in}$ likelihoods, i.e. we output the best in the list as the decided codeword.

We denote by $P_e$ the block error probability for the decoder, implicitly a function of $N$, $L$ and SNR, as well as the finer details of frozen bit selection. There are two ways a block error can occur at the end of decoding, which we denote type I and type II. A type I error occurs when the correct vector (all-0's) is not on the list, and a type II error event occurs when the correct vector is on the list, but one of the other $L - 1$ vectors has a better overall likelihood, or in other words the correct vector is not at the top of the list after final metric sort. When we refer to SNR we mean $E_b/N_0$ as usual.

Letting $\text{miss}$ denote the event that the correct word is not on the list, we can write

$$P_e = P(\text{miss})P(\text{error}|\text{miss})$$

$$+ (1 - P(\text{miss}))P(\text{error}|\text{no miss})$$

$$= P(\text{miss}) + (1 - P(\text{miss}))P(\text{error}|\text{no miss})$$

$$= P_I + P_{II} \tag{3}$$

Another expression for block error probability is obtained by alternate expansion of the above:

$$P_e = P(\text{miss})(1 - P(\text{error}|\text{no miss})) + P(\text{error}|\text{no miss}) \tag{4}$$

These two expressions give an immediate lower bound to error probability:

$$P_e \geq \min\{P(\text{miss}), P(\text{error}|\text{no miss})\} \tag{5}$$

An upper-bound also readily follows:

$$P_e \leq P(\text{miss}) + P(\text{error}|\text{no miss}) \tag{6}$$

The two key probabilities influencing the error probability are thus $P(\text{miss})$ and $P(\text{error}|\text{no miss})$. Both of these are functions of $N$, $L$ and SNR, though they appear difficult to obtain analytically, given the sequential nature of decoding.
and the need for order statistics. We resort to simulation to obtain these probabilities in what follows.

\( P_1 = P(\text{miss}) \) is clearly a monotonically-decreasing function of SNR, for any given \( L \), and likewise in \( L \) for any fixed SNR. In the other direction \( P(\text{miss}) \to 1 \) as SNR or \( L \) decreases. It is worth note that \( P(\text{miss}) \to 0 \) for sufficiently large \( L \) at any non-zero SNR, though very large \( L \) is required as SNR drops near or below the Shannon limit.

The second term of interest, \( P(\text{error|no miss}) \), is the conditional probability that the transmitted code vector resides on the final list, but that some other list member has better likelihood. We compute this empirically as well by Monte Carlo methods, checking for all-0 vector being on the list, but not the best of the list, and normalizing by number of list hits.

While \( P_1 \) steadily decreases for any SNR as \( L \) grows, \( P_{II} \) saturates in \( L \) and eventually dominates the overall block error probability; choice of \( L \) involves subtle balancing of these under the constraint of computational complexity.

Though unrealizable for typical blocklength, an ML decoder’s block error probability is also of interest, as a standard of comparison. ML decoding can be viewed as letting list size become arbitrarily large, so that all codewords have likelihoods evaluated. Then \( P(\text{miss}) \) vanishes. For any finite \( L \) we have a lower bound

\[
P_e,ML \geq P(\text{error|no miss})
\]

and a good approximation to the ML decoder performance is simply \( P(\text{error|no miss}) \) for \( L = 64 \) say.

### A. Numerical results: no CRC case

Simulation results for \( N = 256, R = R_{in} = 1/2 \) are now shown, versus SNR in the range of -1 dB (slightly beyond the Shannon limit for \( R = 1/2 \)) to 5 dB, and for \( L \in \{1, 4, 16, 64\} \). \( E_b/N_0 \) is the energy-per-information bit to noise power density ratio, as usual.

First we show \( P(\text{miss}) = P(\text{type I error}) \) (Fig 1), then \( P(\text{error|no miss}) \) (Fig 2). As discussed above \( P(\text{miss}) \) steadily diminishes with \( L \), though slowly after \( L = 16 \), and all curves are approaching 1 as SNR drops. \( P(\text{error|no miss}) \) on the other hand is basically insensitive to \( L \). The simulated probability of block error is shown in Figure 3, and is a weighted sum of the first two by (3) above. Note that with no CRC assistance, \( P_e \) saturates versus \( L \) quickly around \( L = 4 \), as \( P_{II} \) dominates thereafter. This behavior is tied to the relatively poor minimum distance of the polar code, 8 for this blocklength. (Codes with larger blocklength will not saturate until a larger \( L \).) Moreover for this case, list decoding provides little or no benefit over SC decoding, except at low SNR (and commensurate high error probability).

List decoding (with sufficient list size) is a means of achieving ML decoding for the overall code as SNR increases, and this remains true whether an outer CRC code is present or not. To illustrate we calculate the first term of a union bound for the case of \( N = 256 \) and no CRC. Since the minimum Hamming distance for this rate 1/2 code is 8, and there are 96 nearest-neighbor codewords (found by use of a list decoder at high SNR operating on the all-0’s codeword), the dominant term in the union bound is

\[
P_{ML} \approx 96Q(\sqrt{(2E_b/N_0)(8/2)})
\]

which agrees well with the simulated performance in Figure 3.

### IV. List decoding with CRC code

Now we add an outer code based on a simple CRC code imposing \( C \) check bits. The purpose is not to do concatenated, or joint, decoding, but to aid in selecting the correct code vector from the list. A side benefit of such a concatenation is that the overall minimum distance of the code may be increased, see [3, 4]; however the inner decoder does not utilize the constraints imposed by the outer code.

A perfect outer code would point to the correct code vector, assuming it is on the list, and only to this vector. In this case, \( P_e = P(\text{miss}) \), and as seen earlier, increasing list size can make this probability small, at the expense of decoding complexity. However, a finite-length CRC code can produce false positives, i.e. can give a CRC remainder of all-0’s for
an incorrect vector on the list. This undetected error event probability for a random binary $N$-tuple is

$$P(\text{undetected error}) = 2^{-C}$$

(9)

where $C$ is the number of check bits appended by the CRC code, [10]. There emerges a design tradeoff here—larger $C$ gives a more reliable CRC check, but at the cost of an energy penalty and rate loss.

In the following, we assume that the overall code rate $R$ is fixed here, and so the inner polar code rate $R_{in}$ becomes

$$R_{in} = \frac{K + C}{N} > R$$

(10)

If $K >> C$ then the rate change is negligible, but for finite parameters this energy and bandwidth penalty needs to be acknowledged. In this regard, when we plot versus $E_b/N_0$, the small energy penalty on polar code bits is properly incorporated.

Two ways of applying the CRC check are the following:

1) **Case 1**: Simply sequentially test the list candidates for CRC consistency, with the first list member passing produced as the codeword estimate. (Note list vectors are not ranked at the end of list formation according to likelihood, and false positives can occur, particularly for larger lists.

2) **Case 2**: Follow the procedure of [2] (performing CRC checks and then likelihood sorting among those accepted, if necessary). Equivalently, we could first order the list according to likelihood, then sequentially test CRC consistency. False positives can occur here as well, but we exploit the added benefit that the correct vector will typically have greatest likelihood.

Obviously case 2 imposes additional final sorting computation, increasing with $L$, and the procedure of [2] seems better in this regard.

As before, the block error probability in either case is

$$P_e = P(\text{miss})P(\text{error}|\text{miss})$$

$$+ (1 - P(\text{miss}))P(\text{error}|\text{no miss})$$

$$= P(\text{miss}) + (1 - P(\text{miss}))P(\text{error}|\text{no miss})$$

$$= P_I + P_{II}$$

(11)

which again establishes $P(\text{miss})$ as a lower bound on error probability. If the CRC code is a perfect pointer to the correct vector when it’s on the list, then we’d have $P(\text{error}|\text{no miss}) = 0$, and $P_e = P(\text{miss})$. But as earlier argued $P(\text{miss})$ goes to zero for sufficiently large $L$, for any SNR, including those SNR’s beyond the Shannon limit. The necessary reconciliation is to recognize that as $L$ becomes large, the undetected error probability increases, and type-II events dominate. A general design strategy is to make $L$ reasonably large to nearly ensure the correct vector is on the list, and use a sufficiently strong CRC to reach or approach the ML bound.

Before showing results, we approximate $P(\text{error}|\text{no miss})$ for the two cases.

**Case 1**: Assume the final list of candidate codewords contains the correct codeword. (Note the complete list is normally not sorted according to likelihood metric.) Let the correct vector be in location $i$, $i = 1, 2, \ldots, L$ in the decoder’s list. These candidates are subjected to the CRC check in natural order, with the first sequence passing the check selected as the transmitted codeword. Again, given a random binary vector, the probability of false positive is $2^{-C}$, [10], and given that the correct vector is in location $i$, any of $i - 1$ false positives can occur. We can upper-bound this conditional false positive probability using a union-bound as

$$P(\text{error}|\text{no miss}, \text{location } i) \leq (i - 1)2^{-C}$$

(12)

Thus the unconditional probability of false positive is

$$P(\text{error|no m}) \leq 2^{-C} \sum_{i=1}^{L} (i - 1)P(\text{correct vector at } i)$$

(13)

This can be further simplified to

$$P(\text{error|no miss}) \leq 2^{-C} \{E[i|\text{no miss}] - 1\}$$

(14)

We have studied via simulation the conditional p.m.f. for correct vector location, and found that it’s a uniform distribution on $1, 2, \ldots, L$, as expected for random message selection.\footnote{Here, the all-0’s message is no longer sufficient for study, as the list decoder tends to place the 0-vector near the head of the list, when it’s on the list, and $P(\text{error|no miss})$ is underestimated.}

Thus, the conditional expected value for $i$ is approximately $(L + 1)/2$ and the approximate value for $P(\text{error|no miss})$ is

$$P(\text{error|no miss}) \approx \frac{L - 1}{2(2^C)}$$

(15)

independent of SNR.

Simulation results show this overbounds somewhat the correct probability, and the conditional probability does decrease...
slowly with SNR. This is due to the fact that vectors on the list tend to be codewords close in Hamming distance to the correct vector, and the random selection model above is less valid. Similar effects are seen in undetected error probability for CRC codes on binary channels as the channel error probability decreases, [10].

A reasonable design approach is to make type II error probability well below the targeted block error probability. If this target is \( P_e \), then the CRC length \( C \) should obey

\[
C > \log_2\left( \frac{L}{2P_e} \right), \quad \text{(case 1)} \tag{16}
\]

for whatever \( L \) is selected in the design to meet \( P_e \). Thus a \( P_e = 10^{-4} \) target dictates \( C \geq 18 \) with \( L = 64 \), for example. By the argument above, this is a conservative estimate.

Figure 4 illustrates for \( N=256 \) and \( C = 8 \) the steady drop in \( P_I \) with list size and SNR, (identical for cases 1 and 2), while Figure 5 shows \( P(\text{error}|\text{no miss}) \) (case 1) for \( L=4, 16, \) and 64.

![Fig. 4: Type I probability, \( N = 256 \), CRC-8, case 1 and 2](image)

![Fig. 5: \( P(\text{error}|\text{no miss}), N = 256 \), CRC-8, case 1](image)

Note the increase with \( L \) in Figure 5, and it is this is the source of the high error ‘floors’ in \( P_e \) for this case (Figure 6). Clearly \( C = 8 \) is a poor choice for this decoding option.

**Case 2:** Again the false positive probability bound holds, but in addition a wrong vector passing the CRC check must have higher likelihood than the correct vector to produce decoding error. Considering the various numbers of false positives and their probabilities of error, applying a union bound, and retaining the dominant term yields

\[
P(e|\text{no miss}) \approx \frac{L-1}{2e} Q\left( \frac{(2E_b/N_0)d_{\text{min}}R_{in}}{\sqrt{2}} \right) \quad \text{(17)}
\]

where \( d_{\text{min}} \) is the minimum Hamming distance attached to the inner code. The extra Q-function leverage compared to case 1 reduces the type II error probabilities typically by several orders of magnitude relative to that for case 1, and explains the improved performance in \( P_e \) seen below. We also note that (17) collapses to (15) as SNR drops.

**A. Numerical results:** \( N = 256 \), (Figures 4, 7)

To illustrate we begin with \( N = 256, R = 1/2, \) and \( C = 8 \), so the rate loss is small. The false-positive probability for CRC checks is now proportional to \( 2^{-8} \) under the random-selection model, so for \( L = 64 \) the approximation for case 1 decoding would be \( P(e|\text{no miss}) = 0.00048 \). Figure 7 (case 2) on the other hand has \( P_e \) performance that mimics \( P_I \), tantamount to saying that \( P_{II} \) events have negligible probability, even with \( C = 8 \).

The net gain offered by the outer code with \( N = 256 \) can be found by comparing Figures 3 and 7. At \( P_e = 10^{-3} \) for example, \( L = 64 \) with CRC-8 provides a net 1.4 dB gain over \( L = 1 \) without CRC. (With no CRC, a smaller \( L \) is sufficient as shown earlier.) This gain fairly incorporates the small overhead attached to the CRC code. The energy gain grows as one moves to even smaller \( P_e \). We assert that for this concatenated CRC/polar code performance with \( L = 64 \) and case II processing is close to ML performance.

**B. Results for \( N = 2048 \), \( R = 1/2 \), Figures 8-10**

Simulations were similarly made for blocklength 2048, here with CRC-16 outer code\(^2\) shown in Figures 8-10.

\(^2\)We used the CCITT code, with list size \( L \in \{1, 4, 16, 64 \} \).
Figure 8 shows $P_I$ which again pertains to both cases 1 and 2, while Figures 9 and 10 present the block error probability for the two cases, differing due to $P_{II}$.

First, the curves are steeper and shifted to the left as expected. Notice case 1, (Figure 9) exhibits flaring of the performance curve at lower $P_e$ whereas this disappears in case 2, (Figure 10) due to much smaller type II error probability.

Case 1 performance can be easily made to match that of case 2 by increasing the CRC length to $C = 24$, which for $N = 2048$ represents a negligible overhead, and simplifies decoder processing. Finally, we note case 2 results are consistent with those of [2] for this blocklength; there the CRC length was actually 8.

V. CONCLUSION

We have made explicit the two types of error leading to decoding error with list decoding of polar codes, for both simple CRC checking and list sorting followed by CRC checking. The latter can work well for shorter CRC code, but the former works well for only minor increase in CRC length, which often is a negligible overhead penalty. Type II error probability falls as $2^{-C}$ in both cases, so a small increase in $C$ can repair any ‘floors’ in performance due to type II error dominance. In fact the floor with $L = 64$ is roughly equivalent for both blocklengths, as suggested by the approximate analysis earlier in the paper. Again, a longer CRC code can repair this degradation, and for $N = 2048$ say, the overhead penalty is negligible.

REFERENCES

[1] E. Arikan, “Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channel,” IEEE Transactions on Information Theory, vol. 55, no. 7, pp. 3051–3073, 2009.
[2] I. Tal and A. Vardy, “List Decoding of Polar Codes,” IEEE Transactions on Information Theory, vol. 61, no. 5, pp. 2213–2226, 2015.
[3] T. Murata and H. Ochiai, “Performance Analysis of CRC Codes for Systematic and Non-systematic Polar Codes with List Decoding,” Hindawi, Wireless Communications and Mobile Computing, vol. 2018.
[4] J. Piao, K. Niu, J. Dai, and C. Dong, “Sphere Constraint Based Enumeration Methods to Analyze the Minimum Weight Distribution of Polar Codes,” IEEE Transactions
[5] B. Li, H. Shen, and D. Tse, “RM-Polar Codes,” *CoRR*, vol. abs/1407.5483, 2014.

[6] P. Trifonov and V. Miloslavskaya, “Polar Subcodes,” *IEEE Journals on Selected Areas in Communications*, vol. 34, no. 2, pp. 254–266, 2016.

[7] D. Wasserman, “Polar Coding, with CRC-aided List Decoding,” SSC Pacific, San Diego, CA, Technical Report, 2014.

[8] N. Seshadri and C.-E. W. Sundberg, “List Viterbi Decoding with Applications,” *IEEE Transactions on Communications*, vol. 42, no. 234, pp. 313–323, 1994.

[9] A. Balatsoukas-Stimming, M. B. Parizi, and A. Berg, “LLR-Based Successive Cancellation List Decoding of Polar Codes,” *IEEE Transactions on Signal Processing*, vol. 63, no. 10, pp. 6562–6582, 2015.

[10] T. Fujiwara, T. Kasami, A. Kitai, and S. Lin, “On Undetected Error Probability for Shortened Hamming Codes,” *IEEE Transactions on Communications*, vol. 33, no. 4, pp. 570–574, 1985.