On the Block Error Rate Performance of Spatially Coupled LDPC Codes for Streaming Applications

Mitchell, David G. M.; Pusane, Ali E.; Lentmaier, Michael; Costello, Jr., Daniel J.

Published in:
Proc. IEEE Information Theory Workshop

DOI:
10.1109/ITW.2016.7606831

2016

On the Block Error Rate Performance of Spatially Coupled LDPC Codes for Streaming Applications

David G. M. Mitchell*, Ali E. Pusane†, Michael Lentmaier‡, and Daniel J. Costello, Jr.§

*Klipsch School of Electrical and Computer Engineering, New Mexico State University, Las Cruces, USA, dgmm@nmsu.edu
†Dept. of Electrical and Electronics Engineering, Bogazici University, Istanbul, Turkey, ali.pusane@boun.edu.tr
‡Dept. of Electrical and Information Technology, Lund University, Lund, Sweden, Michael.Lentmaier@eit.lth.se
§Dept. of Electrical Engineering, University of Notre Dame, Notre Dame, USA, costello.2@nd.edu

***Abstract*—In this paper, we study the *block error rate (BLER)* performance of spatially coupled low-density parity-check (SC-LDPC) codes using a sliding window decoder suited for streaming applications. Previous studies of SC-LDPC have focused on the *bit error rate (BER)* performance or the *frame error rate (FER)* performance over the entire length of the code. Here, we consider protograph-based constructions of SC-LDPC codes in which a window decoder continuously outputs blocks in a streaming fashion, and we examine the BLER associated with these blocks.**

We begin by examining the effect of protograph design on the streaming BLER by varying the block size and the coupling width in such a way that the overall constraint length of the SC-LDPC code remains constant. Next, we investigate the BLER scaling behavior with block size and coupling width. Lastly, we consider the effect of employing an outer code to protect blocks, so that small numbers of residual errors can be corrected by the outer code. Simulation results for the additive white Gaussian noise channel (AWGNC) are included and comparisons are made to LDPC block codes (LDPC-BCs).

I. Introduction

Spatially coupled low-density parity-check (SC-LDPC) codes [1], [2], [3] with sliding *window decoding (WD)* [2], [4] are ideally suited for block-streaming applications in the sense that decoded blocks are produced continuously without the need to specify an overall frame length in advance. Because SC-LDPC codes are convolutional codes, previous work has mainly focused on their *bit error rate (BER)* performance (*e.g.*, [5]). Also, the finite-length scaling behavior of SC-LDPC codes has been studied [6] with respect to the *frame error rate (FER)*, assuming *flooding schedule block decoding* over the entire length of the code. However, when a window decoder is used to decode a protograph-based SC-LDPC code for a streaming application, a long continuous sequence of decoded blocks is generated. In this case, the *block error rate (BLER)* associated with these decoded blocks is also of interest.

In this paper, we examine the BLER of SC-LDPC codes with continuous WD from several points of view. We first consider how the protograph design can affect the BLER performance. In particular, we begin with code designs having small coupling width and large block size and then look at the effect on the BLER of increasing the coupling width and reducing the block size such that the overall constraint length stays constant. Next, we examine how the BLER scales with block size and coupling width. Finally, we consider employing an outer code to clean up any residual errors that may remain after the iterations in a particular window position are finished and a block is ready to be decoded.

This work was partially supported by NSF Grant CCF-1161754 and TUBITAK Grant 111E276.

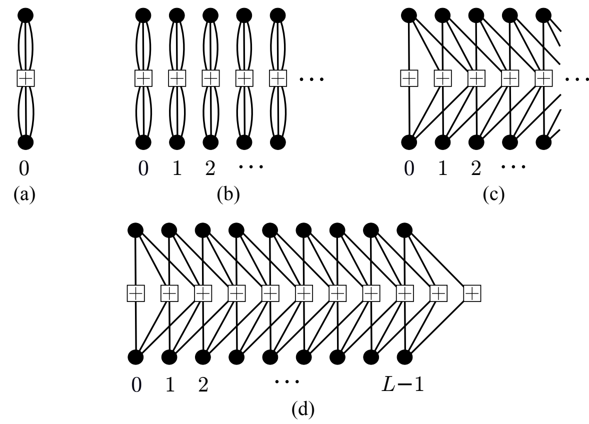

Fig. 1: Tanner graphs associated with (a) a $(3,6)$-regular block protograph, (b) an infinite chain of uncoupled $(3,6)$-regular block protographs, (c) an infinite chain of coupled $(3,6)$-regular block protographs, and (d) a terminated chain of coupled $(3,6)$-regular block protographs with $L = 9$.

II. Protograph-Based SC-LDPC Codes and Window Decoding

A popular way of designing SC-LDPC codes is to use a *protograph* representation of the code ensemble [7]. A block code protograph is a small bipartite graph, with c variable nodes and $c-b$ check nodes, where c and b are typically small integers and $R = b/c$ is the *code design rate*. An example of a block code protograph with $c = 2$ variable nodes of degree 3 and $c-b = 1$ check node of degree 6 is shown in Fig. 1(a). If an M-fold *graph cover* is now formed by applying the copy-and-permute, or *graph lifting*, operation [7], which can be represented by placing a randomly selected permutation of size M on each edge of the graph, a $(3,6)$-regular *LDPC block code (LDPC-BC)* ensemble results. The corresponding $(c-b) \times c = (1,2)$ *base parity-check matrix* in this case is given by $\mathbf{B} = [3,3]$, where the entries in $\mathbf{B}$ denote that the check node in the graph is connected by 3 edges to each variable node. The graph lifting operation is then equivalent to replacing each entry in the base matrix $\mathbf{B}$ with the (modulo-2) sum of 3 randomly chosen (and non-overlapping) $M \times M$ permutation matrices.

In Fig. 1(b), an infinite chain of uncoupled block code protographs is shown. Selecting one of the possible codewords from the $(3,6)$-regular LDPC-BC represented by each of these protographs can then be viewed as transmitting a sequence of codewords from the rate $R = 1/2$ LDPC-BC over the channel.

Now consider *spreading the edges* of each protograph so that they connect to one or more nearby protographs in the chain in such a way that the degrees of all the nodes are preserved [8], as illustrated in Fig. 1(c). This has the effect of *coupling* the different protographs together in an infinite chain, which is equivalent to introducing *memory* into the code design, *i.e.*, transitioning from a rate $R = 1/2$ block code to a rate $R = 1/2$ convolutional code, where the *memory* or *coupling width* w, representing the number of nearby protographs connected to a given protograph, equals 2 in this case. This *edge-spreading* technique is equivalent to splitting the block code base matrix $\mathbf{B}$ into $w+1$ component submatrices $\mathbf{B}_0, \mathbf{B}_1, \ldots, \mathbf{B}_w$, such that $\mathbf{B} = \mathbf{B}_0 + \mathbf{B}_1 + \cdots + \mathbf{B}_w$. For the case shown in Fig. 1(c), the edge spreading is given by $\mathbf{B} = [3,3] = \mathbf{B}_0 + \mathbf{B}_1 + \mathbf{B}_2 = [1,1] + [1,1] + [1,1]$, and the associated infinite convolutional code base matrix is

$$\mathbf{B}_{\mathrm{cc}} = \begin{bmatrix} \mathbf{B}_0 & & \\ \mathbf{B}_1 & \mathbf{B}_0 & \\ \vdots & \mathbf{B}_1 & \ddots \\ \mathbf{B}_w & \vdots & \ddots \\ & \mathbf{B}_w & \\ & & \ddots \end{bmatrix}, \tag{1}$$

where the *constraint length* (the maximum width in symbols of any row of $\mathbf{B}_{\mathrm{cc}}$) is given by $\nu = 2(w+1)$. In what follows, we refer to this as the $[\mathbf{B}_0; \mathbf{B}_1; \cdots; \mathbf{B}_w] = [1\,1; 1\,1; 1\,1]$ edge spreading.

If the graph lifting operation is now applied to the convolutional protograph by placing randomly selected permutations of size M on each edge of the graph, an unterminated $(3,6)$-regular SC-LDPC code ensemble with $\nu = 2M(w+1)$ results, where we note that the protograph has a slight irregularity at the beginning. In particular, the first two check nodes have reduced degrees, a property that leads to the *wave-like decoding effect* characteristic of iterative decoding of SC-LDPC codes and accounts for their improved thresholds compared to the underlying block code ensemble [2], [3].

The coupled convolutional protograph can also be terminated after L sections, resulting in reduced check node degrees at both ends, as shown in Fig. 1(d), in which case the terminated convolutional base matrix is of size $(L+w)(c-b) \times Lc$. Now applying the graph lifting operation results in a terminated $(3,6)$-regular SC-LDPC code ensemble, which can also be viewed as an LDPC-BC. In what follows, we refer to a codeword in this terminated SC-LDPC code, which has total length $2ML$, as a *frame*. The FER behavior of SC-LDPC codes with flooding schedule decoding has been studied in [6] from the perspective of how it scales as a function of both the *lifting factor* M and the *coupling length* L. We note that the code rate associated with this terminated SC-LDPC code is slightly less than the rate $R = 1/2$ of the underlying LDPC-BC. However, as L becomes large, the code rate $R \to 1/2$.

For BP decoding of a terminated SC-LDPC code, the flooding schedule decoder architecture for an LDPC-BC includes all the variables nodes in a block, which equals cML, *i.e.*, the total frame length. However, in a single iteration, the localized structure of the graph implies that messages cannot be passed between variable nodes separated by more than one constraint length. This observation has spurred interest in a modified WD architecture for SC-LPDC codes with much reduced latency and memory requirements [2], [4], where the required window

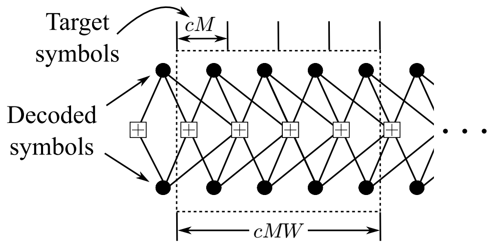

Fig. 2: Example of a window decoder with $W = 4$ operating on the protograph of a $(3,6)$-regular SC-LDPC code. The window moves from left to right and is shown in the second position.

size is typically just a few constraint lengths, resulting in greatly reduced memory and latency requirements compared to the flooding schedule block decoder. The concept of a window decoder is illustrated in Fig. 2. Assuming a window size of cMW symbols, where the *window span* W is the number of protograph sections within the window, a parallel message-passing schedule is used until some stopping criterion is met or a fixed number of iterations has been performed, after which the window shifts and the cM *target symbols* shifted out of the window are decoded. This results in both a *decoding latency* and a required *decoder memory* of cMW symbols, equal to the window size. The key feature of a window decoder is that, since typically $W \ll L$, the latency and memory requirements of a window decoder are much less than for the flooding schedule block decoder.

Because the initial few (depending on the coupling width w) positions of the window include the part of the graph with reduced check node degrees, the information passed to variable nodes during the iterations associated with these initial window positions is highly reliable. The design of the window decoder insures that this highly reliable information then propagates down the chain as the window is shifted.

Since SC-LDPC codes are convolutional codes, research to date (*e.g.*, [5]) has focused on their BER performance, similar to the way convolutional codes are viewed with, say, Viterbi decoding. However, when using the protograph constructions described above combined with WD, a *block* of cM target symbols is decoded each time the window shifts, making it natural to also consider their BLER performance, a view we adopt in this paper. In the remaining sections, we investigate several aspects of the BLER performance of SC-LDPC codes with WD, where decoded blocks are output sequentially from the decoder in a block streaming fashion. Throughout the paper, we consider $(3,6)$-regular SC-LDPC codes as an example, but the techniques discussed are applicable to all SC-LDPC codes, including irregular designs.

III. The Effect of Protograph Design

In Fig. 1(c), we considered a $(3,6)$-regular SC-LDPC code with the $w = 2$ $[1\,1; 1\,1; 1\,1]$ edge spreading. However, this is just one of many possible edge spreadings that can be obtained by varying the coupling width w and the particular connections between protograph sections. For example, for $w = 1$, three distinct edge spreadings are possible without zero entries, viz., $[2\,2; 1\,1]$, $[2\,1; 1\,2]$, and $[1\,1; 2\,2]$. The $[2\,2; 1\,1]$ edge spreading was originally proposed in [4] as a way of reducing the window size, which determines the decoding latency, of SC-LDPC codes, and the iterative WD threshold behavior of each of these edge spreadings was examined in [9].

TABLE I: Parameters for example edge spreadings.

Edge spreading	w	W	M	N	ν	S
$[2\,1; 1\,2]$	1	12	1500	3000	6000	$36,000$
$[1\,1; 1\,1; 1\,1]$	2	18	1000	2000	6000	$36,000$
$[1\,1; 0\,0; 1\,0; 1\,1]$	3	24	750	1500	6000	$36,000$
$[1\,1; 0\,1; 0\,0; 1\,0; 1\,1]$	4	30	600	1200	6000	$36,000$

In this section, we take a somewhat different view of varying the coupling width w. Note that, if the lifting factor M remains constant in the above example, reducing w from 2 to 1 has the effect of also reducing the constraint length $\nu = 2M(w+1)$. Since the *minimum free distance* of regular SC-LDPC codes is known to grow linearly with ν [10], reducing w gives us a weaker code, with an expected negative impact on error floor performance. In other words, reducing w without increasing M results in a smaller decoding latency at a cost of worse performance in the error floor. Since we wish to examine the effect of protograph design on the BLER of SC-LDPC codes, with an emphasis on the error floor, we begin by varying both w and M while holding ν constant.

With this in mind, we choose to examine and compare the following protograph edge spreadings for $(3,6)$-regular codes: $[2\,1; 1\,2]$, $[1\,1; 1\,1; 1\,1]$, $[1\,1; 0\,1; 1\,0; 1\,1]$, and $[1\,1; 0\,1; 0\,0; 1\,0; 1\,1]$, corresponding to $w = 1$, 2, 3, and 4, respectively. The $[2\,1; 1\,2]$ edge spreading was shown in [9] to have the best WD threshold among the possible $w = 1$ cases, and $[1\,1; 1\,1; 1\,1]$ is the classic $w = 2$ edge spreading most studied in the literature. To the best of our knowledge, the proposed $w = 3$ and $w = 4$ edge spreadings have not been previously considered. The likely reason for this is that larger coupling widths w necessitate the use of larger window spans W, thus increasing latency and memory requirements if M is held constant. However, if we reduce M as w is increased, in order to hold ν constant, there is no reason not to consider edge spreadings with larger w.

Experience with WD (see, *e.g.*, [4], [11], [12]) has shown that it is necessary to have a sufficiently large window size, typically a small integer multiple of the constraint length ν, in order not to suffer a performance loss relative to standard block (high latency) decoding. With this in mind, along with the requirement that w and M be varied in such a way that ν stays constant, we considered the sets of parameters shown in Table I for the four edge spreadings given above, where $N = 2M$ is the number of target symbols in a block, or the *block size*, and $S = 2MW = NW$ is the *window size* in symbols. In our example, we set $2MW = 12M(w+1)$, or $W = 6(w+1)$, to ensure good WD performance, but this could be decreased if a lower latency is desired.

Binary-input additive white Gaussian noise channel (AWGNC) simulation results showing the BLER performance of these different edge spreadings in a block-streaming environment, along with results for $(3,6)$-regular LDPC-BCs, denoted by $B(3,6)$, with the same values of N, are presented in Fig. 3. In comparing the SC-LDPC codes to the LDPC-BCs, we note that the SC-LDPC codes enjoy a considerable coding gain and that the gain increases as the block size N decreases. This is due to the fact that the decoding latency S of the SC-LDPC codes is larger than the block size (latency) N of the LDPC-BCs and that this latency advantage increases as N decreases (see Table I). However, after the fixed initial delay of

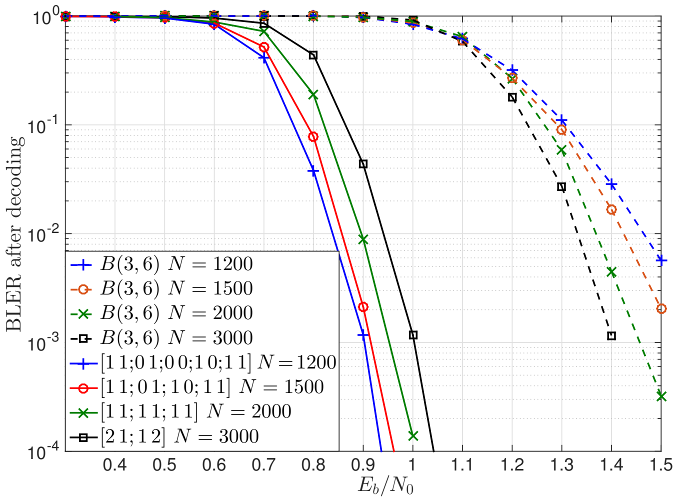

Fig. 3: BLERs for SC-LDPC codes with fixed ν and LDPC-BCs.

S, the SC-LDPC codes produce decoded blocks of size N in a continuous fashion, a nice feature for streaming applications. In addition, on an equal latency basis, SC-LDPC codes are known to outperform LDPC-BCs in terms of the BER, as reported in [11], [12]. Finally, we also note that, no matter how large N becomes, $(3,6)$-regular LDPC-BCs can never perform better than 1.11dB, their iterative decoding threshold, whereas $(3,6)$-regular SC-LDPC codes can approach their threshold of 0.46dB for large constraint length ν.

It is worth pointing out that the "protograph stretching" considered above can be continued in the same fashion by increasing w further and reducing M such that ν remains constant. This results in protographs with larger coupling widths and smaller lifting factors, the limiting case of which is a graph with large w, lifting factor $M = 1$, and $N = 2$ target symbols in each block. This corresponds to the classic view of a rate $R = 1/2$ LDPC convolutional code obtained directly from an already-designed LDPC block code by applying the *cut-and-paste*, or *unwrapping*, procedure first presented in [1]. It also corresponds to the case where the BER and BLER performance measures are essentially equivalent. Indeed, we begin to see this trend in Fig. 3, where the BLERs of the SC-LDPC codes are improving as N gets smaller, *i.e.*, they are starting to tend towards the BER results. An interesting question that we plan to pursue in future research is to examine how WD of SC-LDPC codes behaves as one varies w and M over such a wide range of possible values.

IV. BLER Finite-Length Scaling Behavior

As we noted earlier, the strength of an SC-LDPC code depends on its free distance, which, for regular codes, is known to grow linearly with the constraint length $\nu = cM(w+1)$. Another interesting question about SC-LDPC codes with WD is how the BLER performance scales with ν, or more specifically, with the lifting factor M and coupling width w. AWGNC simulation results exploring the scaling behavior of WD are shown in Fig. 4. In one instance, the $w = 2$ $[11; 11; 11]$ edge spreading is assumed and the values of M shown in Table II are simulated. In this case, the results illustrate the BLER scaling as a function of M, for fixed w. In the other case, the lifting factor is fixed at $M = 750$, and the four edge

TABLE II: Simulation parameters for the $w = 2$ $[1\,1; 1\,1; 1\,1]$ edge spreading.

M	N	ν	S
500	1000	3000	$12,000$
750	1500	4500	$18,000$
1000	2000	6000	$24,000$
1250	2500	7500	$30,000$

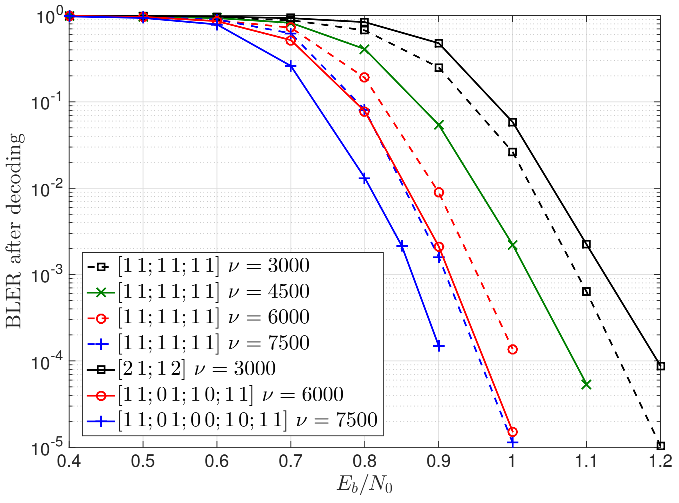

Fig. 4: Scaling behavior of BLERs for SC-LDPC codes with M and w.

spreadings in Table I with coupling widths $w = 1$, 2, 3, and 4, are considered. Here the block size $N = 1500$ stays constant, while the constraint lengths ν and the window sizes S, which increase with w, are the same as those in Table II. In this case, the results show the BLER scaling as a function of w, for fixed M.

In Fig. 4, we see that the BLER scales better with w than with M. This behavior is notable, since it is well known that increasing M results in stronger LDPC-BCs. However, since SC-LDPC codes are *convolutional*, increasing the coupling width (memory) w also increases the strength of the code, and the results of Fig. 4 suggest that this is a somewhat stronger factor in this case. It is also worth pointing out there is no evidence of an error floor down to a BLER of 10^{-5}, which is consistent with the fact that regular SC-LDPC codes are *asymptotically good* [10].

It is important to note here that these results indicate the BLER scaling behavior of WD of SC-LDPC codes, in contrast to the FER scaling behavior reported in [6]. In particular, we look at BLER scaling as a function of M and the coupling width w, which determine the window size, whereas in [6] FER scaling is examined as a function of M and L, which determine the frame length. Our results motivate further analysis of the BLER scaling of SC-LDPC codes, which could be performed using similar techniques to those reported in [6].

V. The Effect of an Outer Code

One way to improve the BLER performance of an SC-LDPC code in the error floor is to protect each block with an outer code. We assume an SC-LDPC code with *systematic encoding* (see, *e.g.*, [13]) and an outer block code with parameters $[n, k, t]$ that protects each set of $N/2 = M$ target information symbols using a *hard-decision bounded distance decoder*. We then consider three cases:

1) M is an integer multiple of n, *i.e.*, $M = dn$, so that each block of M target information symbols is protected by d outer codewords of length n,
2) $M = n$, so that each block of target information symbols is protected by exactly one outer codeword, and
3) n is an integer multiple of M, *i.e.*, $n = pM$, so that each outer codeword protects p blocks of target information symbols.

We also assume that, in a given window position, the window decoder performs a fixed number of iterations in the window and then makes hard decisions based on the N target symbol *log-likelihood ratios (LLRs)*. The reliability of the information symbol hard decisions is then checked using the outer code. In case (1), if t or fewer errors are detected in any of the d codewords comprising a block of target information symbols, they are corrected by the outer code, and the LLRs of the corrected information bits, as well as any affected parity bits, are adjusted to the correct sign. If no errors are detected, the LLRs are not adjusted. Case (2) is exactly like case (1) except that there is only one outer codeword to check. In case (3), the window decoder processes p blocks before the outer code of length $n = pM$ is applied.

Several remarks are in order concerning these possibilities:

- The overall concatenated code rate is reduced by a factor of k/n, the rate of the outer code.
- The error-correcting-capability t of the outer code must be kept small in order to reduce the rate loss.
- A small value of t is also desirable in order to minimize decoding complexity.
- When an *undetected error* occurs, no corrections are made, LLRs are not adjusted, and decoding proceeds as if there was no outer code.
- When a *miscorrection* occurs, additional errors are introduced and LLRs are wrongly adjusted, which may cause *error propagation* into the next block of target symbols.
- The outer code can be designed to do a combination of error detection and error correction, so that it could also be used as a *stopping rule* for the iterative decoder.
- The outer code can also be designed to work with *soft decisions*, but this would increase decoding complexity.

Cases (2) and (3) allow the use of longer outer codes, thus resulting in less rate loss for a fixed level of error correction. Case (3) isn't ideally suited for WD, however, since each block of target symbols must wait for some (up to $p - 1$) succeeding blocks to be processed before the outer code can be applied, thus increasing the decoding latency. Case (2) seems the most natural. But the use of an outer code with a small error-correcting-capability t (to reduce rate loss) is not likely to have much impact on waterfall performance, where we expect incorrect blocks to contain significant numbers of errors. Rather, it is more useful in the error floor, where we expect incorrect blocks to contain only a few errors. Even here, though, since $(4, 2)$ absorbing sets are known to be the primary cause of errors for iterative decoding of $(3, 6)$-regular codes that have not been optimized for girth greater than 6 [14], incorrect blocks could contain as many as 4 information bit errors, and using a $t = 4$ outer code may entail an unacceptable amount of rate loss and decoding complexity.

For a fixed t, case (1) has the disadvantage of a larger rate loss compared to cases (2) and (3), due to the shorter block length. However, since each codeword is capable of correcting t errors, it will be able to correct at least some patterns of

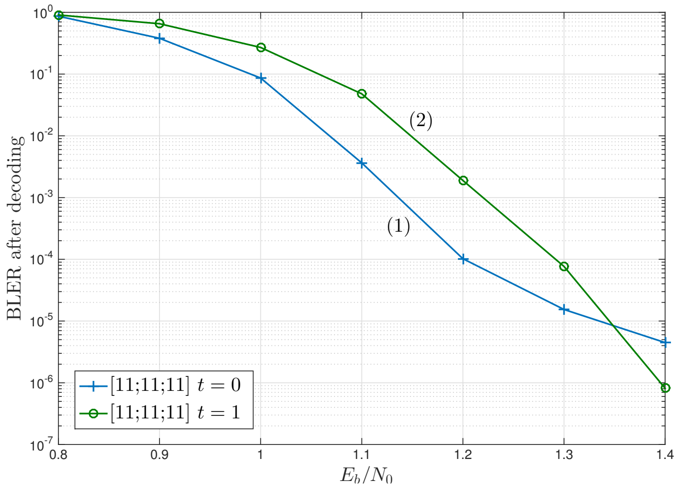

Fig. 5: BLER performance of an SC-LDPC code with (1) no outer code ($t = 0$) and (2) a $[500, 491, 1]$ shortened BCH outer code.

weight greater than t, assuming the errors are distributed so that no more than t occur in any one codeword. This allows us to use low complexity and low rate loss $t = 1$ or $t = 2$ outer codes, in contrast to the larger rate loss and greater decoding complexity that would be needed to correct $t = 4$ errors with an outer code of length $n = M$. Also, since we expect the errors within a $(4, 2)$ absorbing set to be somewhat localized, the chances of having the errors distributed so that at most one occurs in a codeword can be improved by interleaving the d codewords covering a block of M target information symbols.

Preliminary simulation results (see Fig. 5) for the BLER performance of SC-LDPC codes with WD and a low complexity outer code used to correct $t = 1$ errors in a block indicate that improved error floor performance is achieved at the expense of a slight degradation in waterfall performance due to the rate loss of the outer code. We focused on case (2), where we chose a shortened BCH code with $t = 1$ as an outer code. Specifically, we considered the $w = 2$ $[1\,1; 1\,1; 1\,1]$ edge spreading with $M = 500$ and $N = 1000$, along with a shortened BCH outer code of length $n = M = 500$, ($d = 1$, no interleaving). In this case, the rate loss corresponds to a loss in *signal-to-noise ratio (SNR)* of 0.079dB, which explains the slight degradation in waterfall performance. On the other hand, the improvement in error floor performance confirms our expectation that, in this regime, the incorrect blocks contain only a few errors.

It is also possible to use an outer code as a stopping rule for the iterative decoder, with the intention of reducing the number of computations (node updates) needed to achieve a certain level of performance. Although this can be accomplished by designing the outer code to do a combination of error correction and error detection, as noted above, we can also consider the simpler case of employing an outer CRC for error detection only. In this scenario, the outer CRC is chosen to cover exactly one block of M information symbols. Then, after each iteration of the window decoder, the CRC on the set of M information symbols is checked. If no errors are detected, the block is assumed decoded and the window shifts. If errors are detected, iterations continue up to some maximum number of iterations, after which the block is decoded (even if errors remain) and the window shifts.

VI. Concluding Remarks

Protograph-based SC-LDPC codes with WD are ideally suited to a block-streaming environment, since the decoder produces output blocks in a continuous fashion. In this paper, we examined the block error rate (BLER) performance of SC-LDPC codes with WD, where a block is the set of target symbols output by the decoder each time the window shifts. We considered three different aspects of the BLER: (1) how it depends on the tradeoff between the graph lifting factor M and the coupling width w when the constraint length ν is held constant, (2) how it scales when either M or w is increased while holding the other parameter constant, and (3) how it behaves when a high-rate outer code is used to clean up residual errors in a set of target symbols.

Computer simulations were used to examine each of these cases. The results indicated that substantial coding gains are available with block streaming of SC-LDPC codes compared to LDPC-BCs of the same block size, although the decoding of SC-LDPC codes entails an initial delay of several blocks. They also indicated that the BLER scales better with the coupling width w than with the lifting factor M. Finally, the results indicated that using an outer BCH code to clean up residual errors improves the error floor performance at the cost of a slight degradation in the waterfall performance.

References

[1] A. Jiménez Felström and K. Sh. Zigangirov, “Time-varying periodic convolutional codes with low-density parity-check matrices,” *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 2181–2191, Sept. 1999.

[2] M. Lentmaier, A. Sridharan, D. J. Costello, Jr., and K. Sh. Zigangirov, “Iterative decoding threshold analysis for LDPC convolutional codes,” *IEEE Trans. Inf. Theory*, vol. 56, no. 10, pp. 5274–5289, Oct. 2010.

[3] S. Kudekar, T. J. Richardson, and R. L. Urbanke, “Threshold saturation via spatial coupling: why convolutional LDPC ensembles perform so well over the BEC,” *IEEE Trans. Inf. Theory*, 57:2, pp. 803–834, 2011.

[4] A. R. Iyengar, M. Papaleo, P. H. Siegel, J. K. Wolf, A. Vanelli-Coralli, and G. E. Corazza, “Windowed decoding of protograph-based LDPC convolutional codes over erasure channels,” *IEEE Trans. Inf. Theory*, vol. 58, no. 4, pp. 2303–2320, Apr. 2012.

[5] A. E. Pusane, R. Smarandache, P. O. Vontobel, and D. J. Costello, Jr., “Deriving good LDPC convolutional codes from LDPC block codes,” *IEEE Trans. Inf. Theory*, vol. 57, no. 2, pp. 835–857, Feb. 2011.

[6] P. Olmos and R. Urbanke, “A scaling law to predict the finite-length performance of spatially-coupled LDPC codes,” *IEEE Trans. Inf. Theory*, vol. 61, no. 6, pp. 3164–3184, June 2015.

[7] J. Thorpe, “Low-density parity-check (LDPC) codes constructed from protographs,” Jet Propulsion Laboratory, Pasadena, CA, INP Progress Report 42-154, Aug. 2003.

[8] D. G. M. Mitchell, M. Lentmaier, and D. J. Costello, Jr., “Spatially coupled LDPC codes constructed from protographs,” *IEEE Trans. Inf. Theory*, vol. 61, no. 9, pp. 4866–4889, Sep. 2015.

[9] L. Wei, D. G. M. Mitchell, T. Fuja, and D. J. Costello, Jr., “Design of spatially coupled LDPC codes over GF(q) for windowed decoding,” *submitted to the IEEE Trans. Inf. Theory*, 2014. [Online]. Available: http://arxiv.org/abs/1411.4373

[10] D. G. M. Mitchell, A. E. Pusane, and D. J. Costello, Jr., “Minimum distance and trapping set analysis of protograph-based LDPC convolutional codes,” *IEEE Trans. Inf. Theory*, vol. 59, no. 1, pp. 254–281, 2013.

[11] M. Lentmaier, M. M. Prenda, and G. Fettweis, “Efficient message passing scheduling for terminated LDPC convolutional codes,” in *Proc. IEEE International Symposium on Inf. Theory*, St. Petersburg, Russia, Aug. 2011, pp. 1826–1830.

[12] K. Huang, D. G. M. Mitchell, L. Wei, X. Ma, and D. J. Costello, Jr., “Performance comparison of LDPC block and spatially coupled codes over GF(q),” *IEEE Trans. Comm.*, vol. 63, no. 3, pp. 592–604, 2015.

[13] A. E. Pusane, A. Jiménez Felström, A. Sridharan, M. Lentmaier, K. Sh. Zigangirov, and D. J. Costello, Jr., “Implementation aspects of LDPC convolutional codes,” *IEEE Trans. Comm.*, 56:7, pp. 1060–1069, 2008.

[14] L. Dolecek, Z. Zhang, V. Anantharam, M. Wainwright, and B. Nikolić, “Analysis of absorbing sets and fully absorbing sets of array-based LDPC codes,” *IEEE Trans. Inf. Theory*, 56:1, pp. 181–201, 2010.