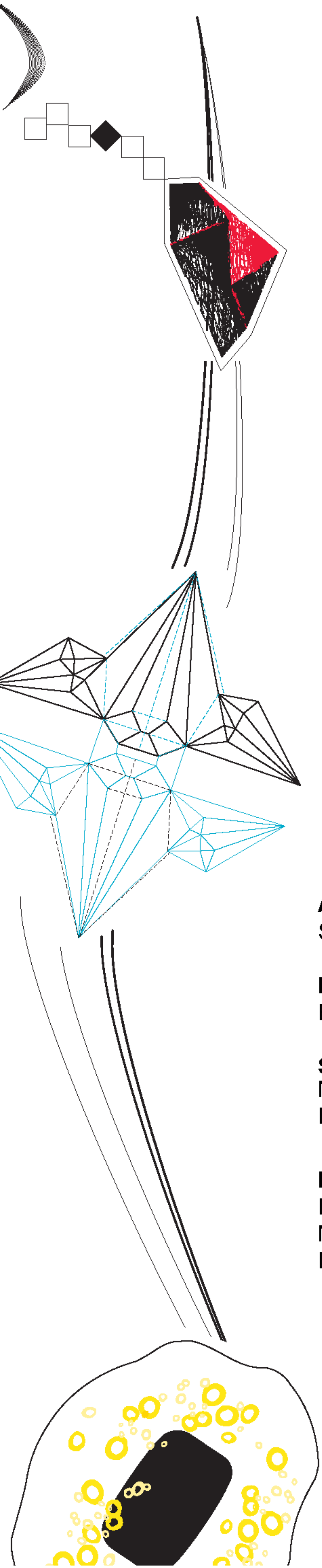

July, 2020 BACHELOR ASSIGNMENT

Comparison of Short Block and Low Rate Channel Codes for UNB communication

Faculty of Electrical Engineering, Mathematics and Computer Science
Radio Systems

Author
Ştefan Miclea

Department
Electrical Engineering

Supervisors
MSc. Siavash Safapourhajari
Dr. ir. Andre B.J Kokkeler

Exam committee
Dr. ir. Andre B.J Kokkeler
MSc. Siavash Safapourhajari
Dr. ir. Harijot S. Bindra

UNIVERSITY OF TWENTE.

Table of contents

Abstract ... 1
1 Introduction ... 1
2 Coding for short blocks ... 2
2.1 BCH code ... 2
2.2 LDPC codes ... 3
2.3 Convolutional codes ... 3
2.4 Turbo codes ... 3
2.5 Polar codes ... 4
2.6 Channel coding used in this work ... 4
3 Implementation of Polar codes ... 5
3.1 Channel polarization: ... 5
3.2 Encoding: ... 6
3.3 Decoding: ... 9
4 Simulations & Discussion ... 13
4.1 Coding schemes with BPSK modulation ... 13
4.2 Coding schemes with hybrid modulation ... 15
5 Conclusion ... 21
6 Appendix ... 25

Abstract

Ultra Narrow Band (UNB) is one of the technologies used for Low Power Wide Area Network (LPWAN). Solutions using UNB for this type of networks encounter problems such as Carrier Frequency Offset (CFO) and time-varying fading channels [1]. An offset tolerant demodulator has been proposed as a solution to overcome the problem of carrier frequency offset. To tackle the time-varying fading channel distortion, time, or frequency diversity together with channel coding can be used. Since this solution would add extra bits to transmit and for this type of networks, saving power is the goal, a higher order modulation can be considered in order to provide the required bit-rate. Hence, a hybrid frequency/phase modulation is used with an offset tolerant demodulator to both tackle the CFO as well as providing the bit-rate which is required for time diversity and encoding. This solution is described and tested using a repetition code as the channel coding, in [1]. To further enhance the performance achieved in [1], better channel coding for this scenario should be considered, implemented, and tested. This paper will focus on implementing the Polar code in MATLAB and test it under the described scenario. Also, MATLAB objects were used to implement a BCH code, test it in the same scenario and compare the results. The results show a better performance of the Polar code in comparison with the BCH code in the final testing environment.

1 Introduction

Ultra Narrowband (UNB) is one of the technologies used for implementing Low Power Wide Area Networks (LPWAN) [2]. Its ultra-narrow bandwidth allows this technology to be less affected by interference since it only accounts for the noise present within its specific bandwidth, which is very narrow. LPWAN networks use this technology to link high numbers of devices operating in the same area due to its increased selectivity. This technology uses short transmissions which limits the power needed for both the transmitter and the receiver.

Time-varying channels and carrier frequency offset are two obstacles which UNB needs to surpass [1]. A time-varying channel changes during the transmission which makes individual symbols experience different channels. Due to this differentiation between the quality of channels, the symbols may experience burst errors.

Carrier frequency offset arises mainly due to two factors. The components imperfection can make the frequencies between the transmitter and the receiver differ slightly. The frequency can also change based on temperature, pressure, and the age of the components [3]. The second reason for carrier frequency offset is the Doppler effect. Movements of either the transmitter, receiver, or other objects around them, that scatter the signal, creates a Doppler shift. This appears due to the fact that a source of emitting waves at a constant frequency, which is moving in a direction, will appear to a static observer as emitting a different frequency depending on the angle and speed of the wave source [4].

In a network composition using UNB, a base station is established and connected to multiple devices in the area [5]. The goal is to prolong the life of these networks by saving as much power as possible at the nodes (devices) which are usually battery powered. In the target application, the simplicity and low power consumption at the devices has higher priority than the power consumption in the base station.

Previous research proposed solutions for these two obstacles. In [6] an offset tolerant demodulator was used to counteract the effect of carrier frequency offset. Then, in [1] the time varying channel obstacle has been treated by using time diversity and repetition coding. The concept of time diversity consists of having the bits composing a codeword interleaved (shuffled in time) with bits from other codewords and, then, transmitted. This makes each bit of a codeword experience different channels and thus, there is an increased probability that enough bits are received correctly to recover the whole codeword.

Other coding techniques can be used instead of the repetition code to enhance the performance of the system, but most coding techniques suffer from decreased performance at short block lengths. Moreover, channel coding adds extra bits to the message which help recover the transmitted message in case of noise corruption. Due to sending extra bits, the transmission time will increase. A hybrid modulation scheme is used to transmit more bits in the same time by increasing the order of modulation so that, after encoding, the total transmission time remains the same.

With this concept in mind, [1] introduces an offset tolerant hybrid modulation technique consisting of a combination of FSK and PSK to increase the modulation order and decrease the total transmission time. The focus of this work is extending the research in [1] by answering the following questions:

1. What are suitable channel coding techniques for the low block length applications?
2. How do this channel coding schemes perform in the scenario investigated in [1]?

This report is organized as follows. In Chapter 2 several coding techniques are introduced and evaluated based on their suitability for the application at hand. Chapter 3 consists of an in-depth analysis of the concept of Polar code as well as its implementation. Then, the implementation of the Polar code is evaluated and compared with the BCH code in Chapter 4. Finally, conclusions are drawn in Chapter 5.

2 Coding for short blocks

Several channel coding techniques are considered for the purpose of this paper based on their wide usage in Ultra Reliable Low Latency Communications (URLLC). According to the literature review, two codes will be chosen based on their suitability for short block length and complexity.

2.1 BCH code

Bose-Chaudhuri-Hocquenghem (BCH) form a class of error correcting codes with high error correcting capabilities. Since they are based on finite fields arithmetic, these codes have large minimum distances which allows for these big error correcting capabilities. An important implication of this large minimum distance is that this type of codes does not suffer from flooring errors at small block lengths, making them known as a very suitable choice for this type of network [7]. Also, due to the construction of the codes, the designer has precise control upon the number of bit errors which can be corrected. Moreover, they can be decoded using a syndrome decoder which is of low complexity and, thus, allows the use of low power electronics to build it. One of the downsides of this coding technique is lack of flexibility in the code length. Arbitrarily choosing the block length and the information bits is not possible. The relation between n and k for this type of code is $n - k \leq mt$ where $n = 2^m - 1$ and t is the error correcting capability [8].

2.2 LDPC codes

Low-density parity-check codes are a class of linear error correcting codes introduced by Gallager in 1960. Although discovered 30 years earlier, only in 1990 they started to be considered again. The construction of these codes is based on Tanner graphs. Using these codes along with an iterative belief propagation decoder has achieved performances within fractions of decibels away from Shannon's limit. Although the complexity for these codes is limited, their performance does not hold for short block lengths. Downsides of this class of codes are related to their lack of flexibility, which can be enhanced using different techniques but at the expense of increased complexity [9].

2.3 Convolutional codes

This class of codes were introduced by Peter Elias in 1955. They are different from block codes by the fact that instead of appending parity bits to message bits, this way of encoding only sends the parity bits. They are obtained in the encoder using a sliding window where the information bits enter one by one and are kept there for *m* instances of time. *m* is referred to as the memory of the encoder and depicts how many information bits can be 'inside' the encoder at a given time. The performance of convolutional codes enhance if their memory increases. By the nature of the code construction, the last *m* bits need to be set to 0 to 'terminate' the code, which results in a rate loss. This rate loss at short block length could be too costly and thus, Tail-Biting Convolutional codes (TB-CC) could be a better option for this type of communication. This technique involves appending the initial *m* info bits instead of the 0's to terminate the code. Although this solves the issue of rate loss, the complexity of the TB-CC decoder is higher than terminated Convolutional codes [9],[10].

2.4 Turbo codes

Turbo codes are a class of powerful error correcting codes introduced in 1993 by Berrou, Glavieux and Thitimajshima. At that time, the results obtained by these codes were revolutionary. They were originally proposed as a parallel combination of two convolutional codes, but lots of different designs arrived with time. They are one of the best codes in terms of performance. Moreover, this good performance is achieved with low encoder and decoder algorithm complexity. However, it needs to be mentioned that these great performances are achieved at medium and large block lengths [9],[11],[12].

2.5 Polar codes

Polar codes were first introduced by Erdal Arikan in 2008 and they are based on the idea of channel polarization. They are also the first codes proved to be capacity achieving. Channel polarization is based on information theory techniques. The main idea is to categorize the individual channels which the bits experience from random channels to either noiseless or very noisy channels. These codes can use Successive Cancellation decoding. Although these codes are capacity achieving for high N (which is the block length), at the block lengths currently used in the industry, its performance is lower than the more common codes such as LDPC and Turbo codes. To enhance the capabilities of this decoder, SCL (Successive Cancellation List) decoder was introduced, but this comes at the expense of higher complexity [13]. The main advantage of this type of code is the very low encoder and decoder complexity.

2.6 Channel coding used in this work

The results found in paper [9] show that the BCH code under Ordered Statistics Decoder (OSD) outperforms all other codes at short block length. Other works such as [7] reinforce the statement of BCH codes being great performers at short block length, thus they seem to be a suitable option to choose for our target.

Moreover, in [9] the suitability of different coding schemes for short block lengths has been investigated. Figure 1 shows a summary of the findings in [9]. In this figure, the vertical axis is represented by the decoding complexity of the coding schemes, while the horizontal axis represents the gap to the normal approximation. The normal approximation is considered to be the benchmark of performance. It is an approximation of the maximum achievable performance of a coding scheme at a certain error rate, considering the block length and the rate of the code [14]. In this figure, the gap to the normal approximation for an error rate of 10^{-4} is considered.

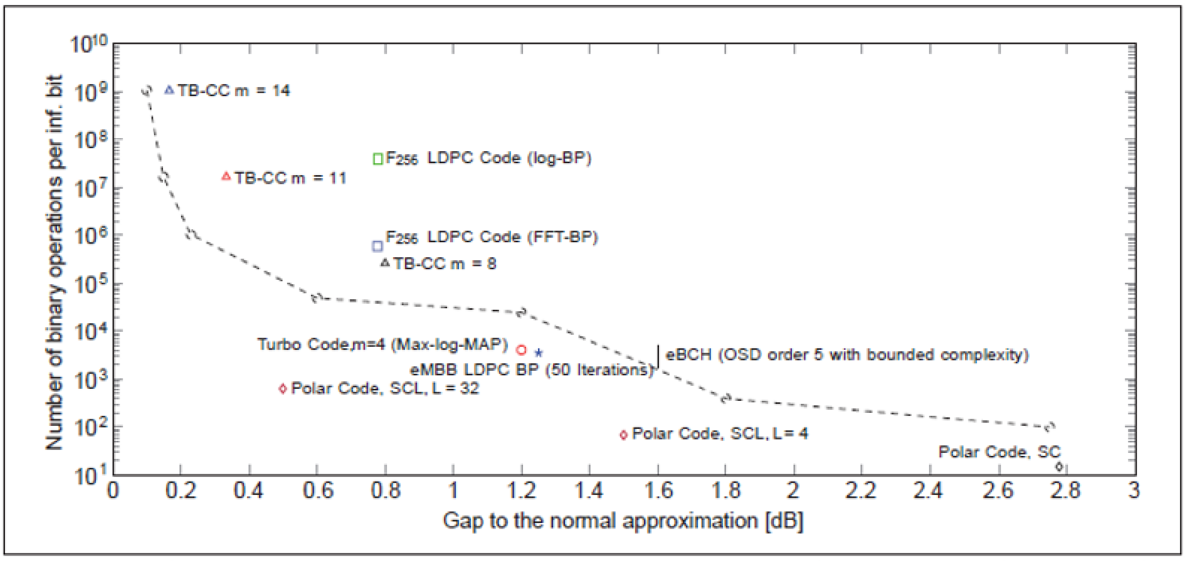

Figure 1: Complexity vs Performance of various channel codes [3].

As can be seen in Fig. 1, the polar code under Successive Cancellation decoder is the least complex of all the considered codes at a block error rate of 10^{-4}. Also, it has the worst performance compared to other coding techniques having a gap of 2.8 dB to the normal approximation. On the opposite side, Tail-Biting Convolutional codes show the highest performance as well as the highest complexity at the same block error rate. The graph shows that the BCH code, contrary to the conclusion of [9], not having the best performance among the other codes. This is due to the fact that in this graph, the decoder used for BCH was a bounded complexity decoder. This was used to trade the performance of the code for a simpler implementation, hence, it achieves a lower performance.

Based on the requirements of the network in question, the Polar code was chosen to further analyze and implement due to its very low complexity, which is as discussed, very desirable in our context for low power communication nodes. Moreover, the BCH code was chosen to be implemented and used for comparison with the Polar code.

3 Implementation of Polar codes

In this chapter the theoretical concepts behind the construction of the polar code as well as its decoder are analyzed.

3.1 Channel polarization:

To be able to better explain the concept of channel polarization, a Binary Erasure Channel should be defined. A $BEC(p)$ is defined as a channel with an erasure probability *p*. Hence, the bits transmitted over this channel have a probability *(1-p)* to arrive correctly at the receiver and a probability *p* to be erased and the receiver would get a warning that the bit was erased.

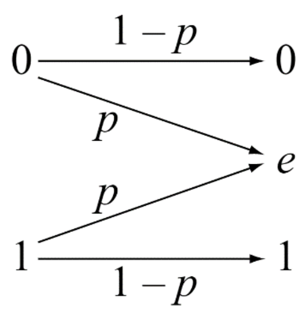

Figure 2:Visualization of Binary Erasure Channel

Figure 2 shows a visual representation of the characteristics of a Binary Erasure channel.

Sending each bit separatly on a $BEC(p)$ channel would actually yeild two identic channels through which both bits are sent. This means they both experience a probability of error of *p*. In a simple version of polar code with only two bits, instead of sending the bits, the XOR of the bits and the second bit are transmitted. This way of encoding the bits will be addresed as the Polar transform further in this paper. A visual representation can be seen in Figure 3 where U represents the message bits, and Y the received bits.

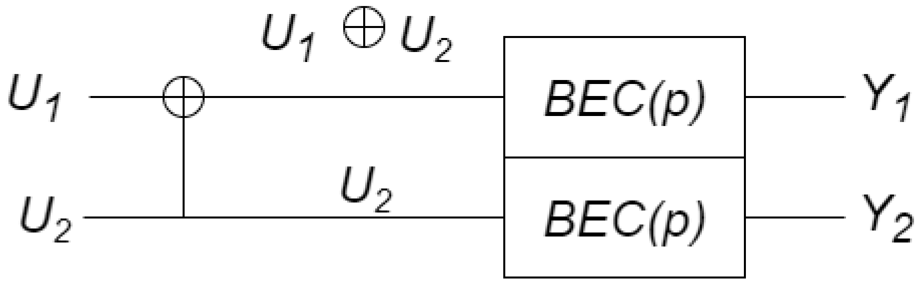

Figure 3:Schematic of Polar encoded bits over a BEC channel.

The Polar transform creates two new channels W- and W+. Y1 and Y2 are used to decode U1 under W- channel and the value for U1 **is assumed** to be correct. Using U_1, Y_1 and Y_2; U_2 is decoded under the W+ channel (see Figure 4).

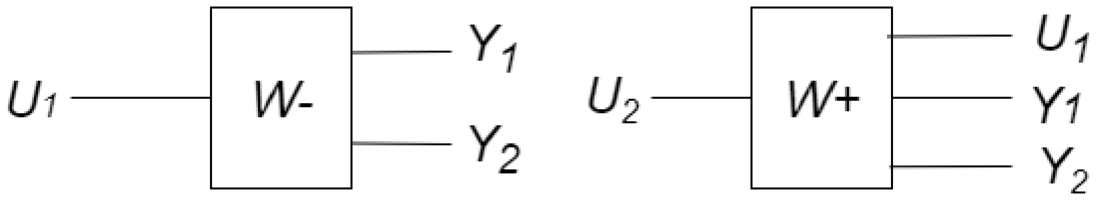

Figure 4: Decoding under the newly created channels.

Analysing W-, it is observed that if Y_1 is correctly received, then $U_1 + U_2$ is known. If Y_2 is correct, U_2 is known, thus, both of them have to be received correctly in order to decode U_1 ($U_1 = Y_1\ xor\ Y_2$), making the probability for this to happen equal to $(1-p)^2$. Thus, the new channel is a BEC with $(1-(1-p)^2)$ as the probability of erasure.

Analysing W+, it is observed that if Y_2 is received correctly, so is U_2. Also, since we assume U_1 to be correct, if Y_1 is received correctly, then U_2 is also received correctly ($U_2 = Y_1 + U_1$). This makes W+ a BEC with probability p^2 since the chance of failure is when both of Y_1 and Y_2 are received erroneous.

Using the polar transform, the channels experienced by the two bits are changed from $BEC(p)$ for both bits to $BEC(1-(1-p)^2)$ for the first bit and $BEC(p^2)$ for the second one. As can be seen, the channel is polarized using this transform. W+ is a better channel than initial channel, as p^2 is a smaller error probability than *p*. In contrast, W- is a worse channel compared to $BEC(p)$ since $BEC(1-(1-p)^2)$ is higher than the initial channel error.

This concept is used by Arikan to find the good and bad channels and send the information bits on the noiseless channels while through the noisy channels, no information is sent and the respective bit positions are set to 0. [15],[16].

3.2 Encoding:

The Polar transform is defined using the principle of channel polarization. The G_2 shown in (1) is the building block of the Polar code. Multiplying the message bits with the kernel G_2 yields the encoded version of the N=2 Polar code.

$$G_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}; [u_1\ u_2]G_2 = [u_1 + u_2\ u_2] \qquad (1)$$

Constructing larger Polar codes is done by calculating the Kronecker product of G_2 with itself as many times as needed to get the wanted *N* value for the code. A Kronecker product is defined by taking each element position from a matrix and replace it with the second matrix, multiplied by the respective element. An example is shown in equation (2) [15].

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}; B = \begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix}; A \otimes B = \begin{bmatrix} 1\begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} & 2\begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} \\ 3\begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} & 4\begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 & 5 & 0 & 10 \\ 6 & 7 & 12 & 14 \\ 0 & 15 & 0 & 20 \\ 18 & 21 & 24 & 28 \end{bmatrix} \tag{2}$$

Thus, using this concept, the G_4 matrix along with the encoded message for *N=4* Polar code are computed as can be seen in Eq. (3) and (4) respectively.

$$G_4 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{3}$$

$$[u_1,\ u_2,\ u_3,\ u_4]G_4 = [u_1 + u_2 + u_3 + u_4, u_2 + u_4, u_3 + u_4, u_4] \tag{4}$$

In (4) u_1, u_2, u_3, u_4 represent the message bits, and the resulting vector after G_4 multiplication is the Polar encoded message. To gain a better understanding, a binary tree representation is used. The representations for a N=2 (using G_2) and N=4 (using G_4) Polar code construction can be seen in Figure 5.

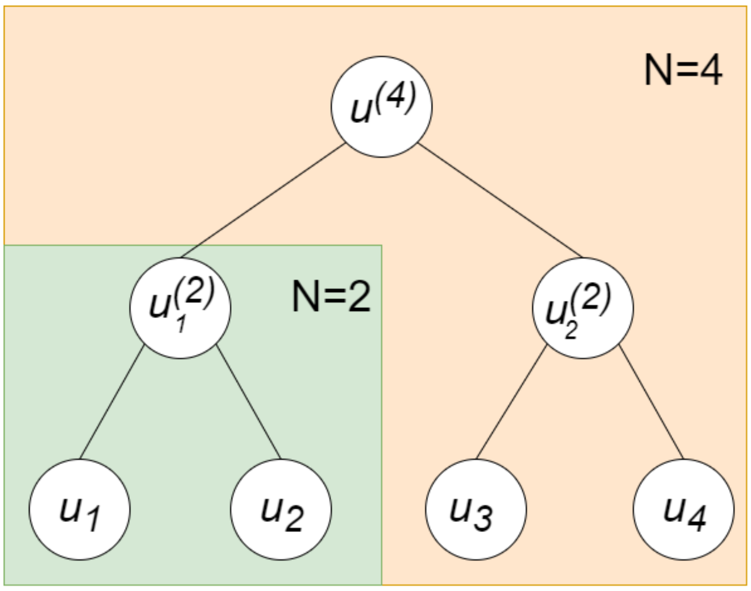

Figure 5: N=2, N=4 binary tree representations

The construction of the nodes $U_1^{(2)}$, $U_2^{(2)}$ and $U^{(4)}$ from Figure 5 can be seen in Eq. (5)

$$U_1^{(2)} = [U_1 + U_2, U_2];\ \ U_2^{(2)} = [U_3 + U_4, U_4]; U^{(4)} = [U_1^{(2)} + U_2^{(2)}, U_2^{(2)}] \tag{5}$$

Moreover, the '+' sign in Eq. (5) signifies the XOR operation, not simple addition. The leaves of the tree represent the message bits (which can either be information bits or 0 set bits, depending on the respective channel quality) and the root of the tree represents the encoded message. Using the concepts of Polar tarnsform, the polar encoder can be described as follows.

Assume that the message bit stream is a vector of *K* bits. Then another vector, 'u' of size $N = 2^n$ is created to represent the leaves of the binary tree. Similar to what was explained earlier about a polarized channel (see Figure 4), when polar transform is used, each entity of vector 'u' experiences different qualities of channels. An essential idea of the polar code is to determine the *N*-*K* positions of the 'u' vector which experience the least reliable *N*-*K* channels and set them to 0. These are called 'frozen' bits.

Assessing which bit positions should be set to 0 is a heavily researched topic. There are different mehtods to estimate this but it is not straightforward and it can also change based on the noise levels [16]. To determine the frozen bits a reliability sequence is used. The reliability sequance represents a vector where the bit channels (entities of vector 'u') are ordered from the worst reliable to the most reliable, under the concept of channel polarization. For the purpose of this paper, the reliability sequance used in the 5G standard is used and adapted to the desired block length.

Since in the reliability sequance the channels are arranged from the worst to the best, the first *N*-*K* channel numbers found in the sequance are selected as frozen positions in the 'u' vector. The remaining positions in the vector are set to the values of the message bits.

To be able to continue with encoding, a way of traversing the tree needs to be defined. This is done using depth and numbering each node at each depth. For example the root node $u^{(4)}$ of Figure 5 is defined as depth 0, node 0, so (0,0). Consequently, node $u_2^{(2)}$ is defined as (1,1)(is at depth one and the second node). u_4 would be defined as (2,3) depth 2 node 3.

In Figure 6 an example of how the binary tree is annotated and how message and frozen bits are arranged at the leaves of the tree can be seen.

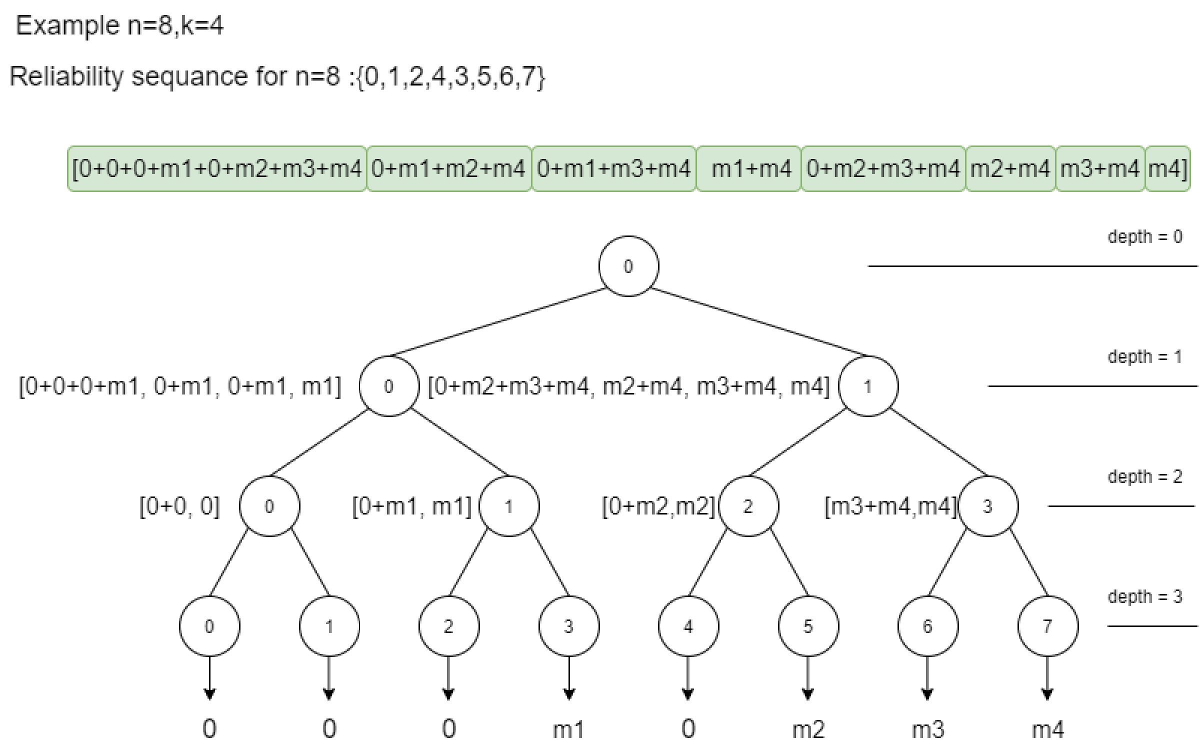
Example n=8,k=4
Reliability sequance for n=8 :{0,1,2,4,3,5,6,7}
[0+0+0+m1+0+m2+m3+m4 | 0+m1+m2+m4 | 0+m1+m3+m4 | m1+m4 | 0+m2+m3+m4 | m2+m4 | m3+m4 | m4]
depth = 0
depth = 1
depth = 2
depth = 3
[0+0+0+m1, 0+m1, 0+m1, m1]
[0+m2+m3+m4, m2+m4, m3+m4, m4]
[0+0, 0]
[0+m1, m1]
[0+m2,m2]
[m3+m4,m4]
0
0
0
m1
0
m2
m3
m4

Figure 6: Encoding example for (8,4) Polar code

Using the node deffinition and the polar transform, the message is encoded gradually from the bottom of the tree to the top, where the final encoded message is found. As can be seen in Figure 6 the transform takes 8 bits and returns 8 bits. Notice that the four message bits (m1,m2,m3,m4) are encoded into an 8-bit codeword. As a specification, the + sign used in the figure does not denote normal addition, but the XOR of the bits, or addition modulo 2[16], [17].

3.3 Decoding:

In this work a Successive Cancellation (SC) decoder is used for the polar code. It is a sequential decoder, meaning that it decodes the bits one by one and it uses the previous decoded bit to decode the next one.

In case of this coding scheme, both the transmitter and the receiver have the reliability sequence and know which positions in the transmission are frozen. Thus, instead of decoding these bit positions, they are simply set to 0 and the decoder moves to the next step. In case the bit is not frozen, an estimate for it is computed using a soft decoding technique. It is called a soft decoder since it uses the value obtained straight from the decision variable at the output of the demodulator, before making a hard decision (0 or a 1) for the received bit. The freezing technique is very advantageous since, initially, frozen bits are located in the positions of the bit sequence which experience very noisy channels. If they were actual information bits, their detection could cause an error at the receiver which would further propagate due to the sequential nature of the decoder. Hence, by knowing which positions must be 0, at the decoder, these bit positions are forced to 0, thus, preventing error propagation.

Starting with a length two polar encoder, the principles of the SC decoder are described below. Figure 7 shows the polar encoding and transmission over an AWGN channel using G_2. r_1 and r_2 are the received decision variables thus they are beliefs for x_1,x_2.

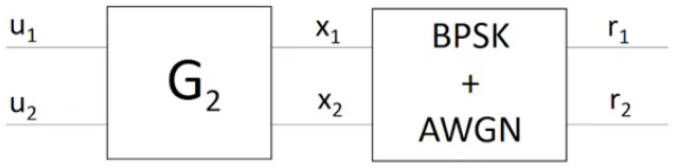

Figure 7: Transmission of N=2 Polar code under BPSK and AWGN channel.

Since G2 is invertible, a relation between x_1, x_2 and u_1,u_2 can be found which is shown in Figure 8.

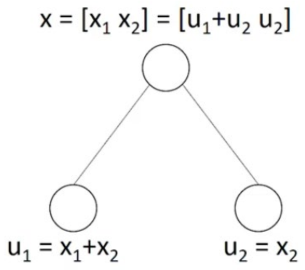

Figure 8: Decoding of N=2 Polar code.

Then, knowing that r_1 and r_2 are beliefs for x_1 and x_2, a function known as min-sum is used to get the soft belief for u_1 which can further be used to get the hard decision by putting a threshold.

The min-sum function is defined as follows.

$$f(r_1,r_2) = sgn(r_1)sgn(r_2)\min(|r_1|,|r_2|) \tag{6}$$

Where *sgn(.)* is the sign function, considering the sign of each belief, and *min(.)* is the minimum function which in this case is used to get the minimum absolute value of the two beliefs.

After u_1 is obtained, it is assumed to be correct and further used to decode u_2. If u'_1 is 0, then what was initially transmitted was $[u_2, u_2]$, meaning that both r_1 and r_2 are actually beliefs for u_2, and adding them would give the final belief for u_2. If u_1' is 1, then what was transmitted was $[1 + u_2, u_2]$ which is the complement of u_2 and u_2. This means that r_1 is the belief for the complement of u_2, making -r_1 a belief for u_2. Also, r_2 is a belief for u_2, hence the final belief for u_2 will be $r_2 + (-r_1)$.

This can be visualized in Figure 9 where u_1' ,u_2' are the decoded message bits and r_1,r_2 are the received beliefs for u_1 and u_2, respectively. The figure shows how to obtain u_2 as a function of u_1 and the received beliefs r_1 and r_2.

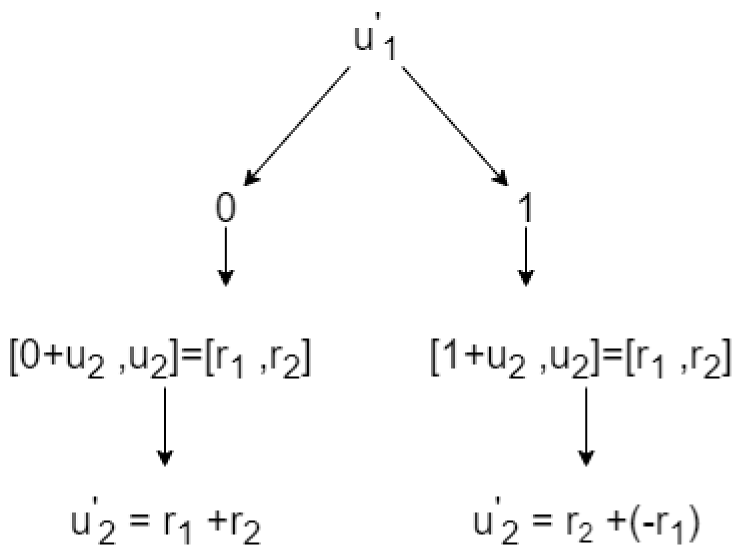

Figure 9:Visualization of the repetition decoding principle.

The function used to model this behavior is:

$$g(r_1, r_2, u_1) = r_2 + (1 - 2u_1)r_1 \tag{7}$$

To be able to implement the decoder, the same definition as for the encoder will be used to traverse the binary tree. After identifying the nodes, operations are made at each of them separately until the whole tree is traversed and the decoding is done.

Starting from (0,0) which is the root node, the binary tree is traversed using 3 steps. Before continuing to the steps, a few terms need to be defined. A 'parent' node is referred to any node which is not at maximum depth. Then, 'children' nodes can be any nodes except the root node which is at the lowest depth. The 'left child' and the 'right child' refer to, respectively, the left node and the right node that are connected to a parent. To identify the respective children to any parent node, it is observed that the left child of parent number *p* which is at depth of *d* is situated at depth *d+1* and is numbered as *2p*. Similarly, the right child of any node is identified again by depth *(d+1)* and numbered as *2p+1*.

An example was devised for easing the understanding of these definitions. Two nodes were randomly selected to verify the definitions made for locating the children. As can be seen in Figure 10 the definitions hold and the location of the respective children is correctly found.

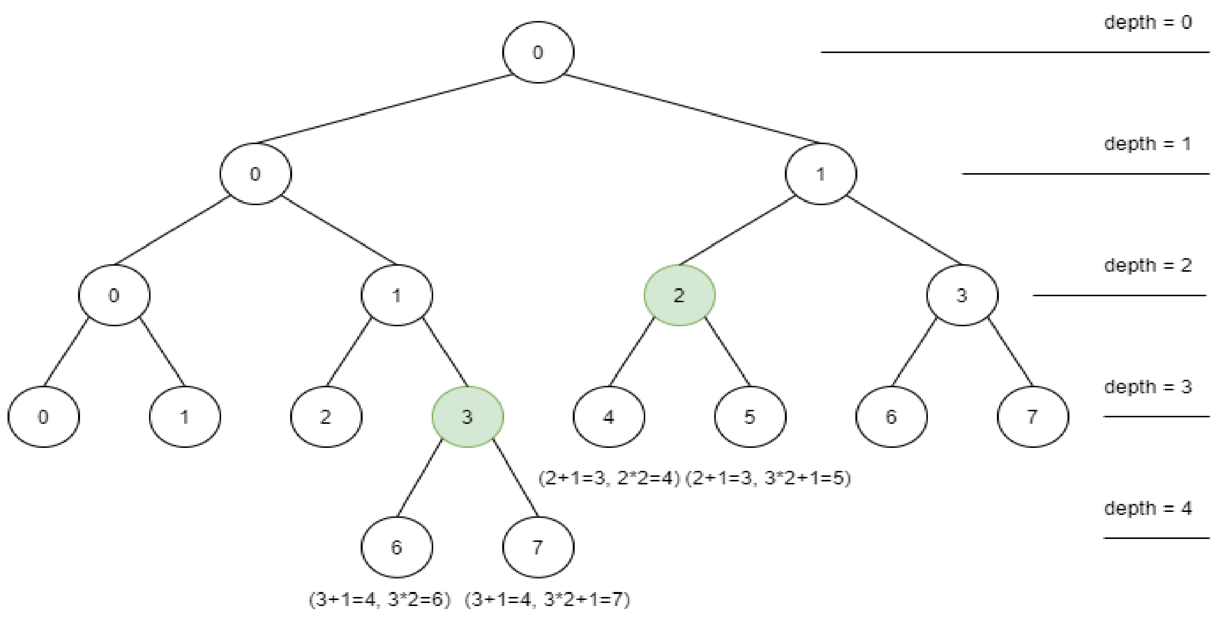

Figure 10:Example locating the children of a node.

After correctly locating all the nodes, the decoding is done by traversing the tree using the following steps, starting from the root node.

- Step 1 will go to the left child of the node, doing the min-sum function described in (6) until a leaf node is encountered (depth is maximum). When the leaf node is reached, the belief for the first bit is obtained using a threshold and is sent back up to the parent node.
- Step 2 begins when a node receives a belief from its left child. In this step, the node goes to its right child using the function defined in (7) and the belief for its left child to find a belief for the right child.
- Step 3 is reached when a node receives a belief from its right child, then, it will send the beliefs for its left and right child to its parent node.

In Figure 11 a visualization of these steps is represented for simplicity. The implementation code for both the encoder and decoder can be seen in the Appendix.

[16], [17]

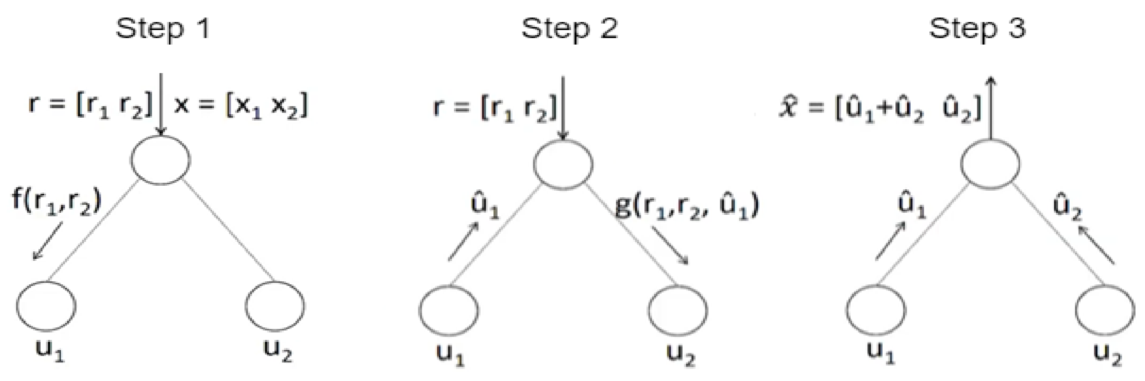

Figure 11: Decoding steps

4 Simulations & Discussion

As discussed in Chapter 1, the BCH code will be also used and is compared with the polar code. The implementation of this coding technique was done using MATLAB communication toolbox system objects designed for encoding and decoding BCH codes. The Polar code is implemented in MATLAB codes according to the discussions during the previous chapter.

The block length for both coding schemes was chosen to be 256 since this number is in the range of UNB communications block length [1]. Moreover, the hybrid modulation used in this work is a combination of 4FSK and 4PSK which transmits 4 bits per symbol and can achieve a bit-rate four times higher than the one of BPSK. Thus, the rate of the coding scheme will be set to ¼ so that, in combination with the hybrid modulation, the total transmission time is not affected. This would yield a (256,64) code for Polar code. As a consequence of the low flexibility of BCH codes in terms of block length and message length, a (255,63) BCH code is used as the closest version to the used block length and code rate for Polar code (256 and 1/4, respectively) for comparison.

The coding techniques are evaluated in two stages and the results are presented in the following sections. The first section includes results for a Binary Phase Shift Keying (BPSK) modulation and Additive White Gaussian Noise (AWGN) channel while the second section includes the results for hybrid modulation and different channel models.

4.1 Coding schemes with BPSK modulation

During the first stage, BPSK modulation was used along with added white gaussian noise. The block diagram for this stage can be seen in Figure 12. Furthermore, the BER curve and the BLER (Block Error Rate) can be seen in Figure 13 and Figure 14, respectively.

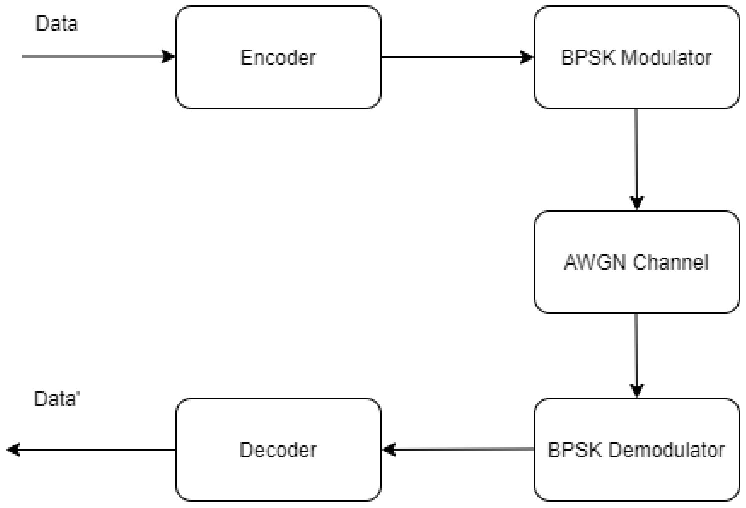

Figure 12:Block diagram of Stage 1 testing environment.

As can be seen in Figure 12, the system used for the first test includes encoding the information data, modulating it using BPSK, adding white gaussian noise to the signal, demodulating it, decoding the received data and finally comparing it with the initial information stream.

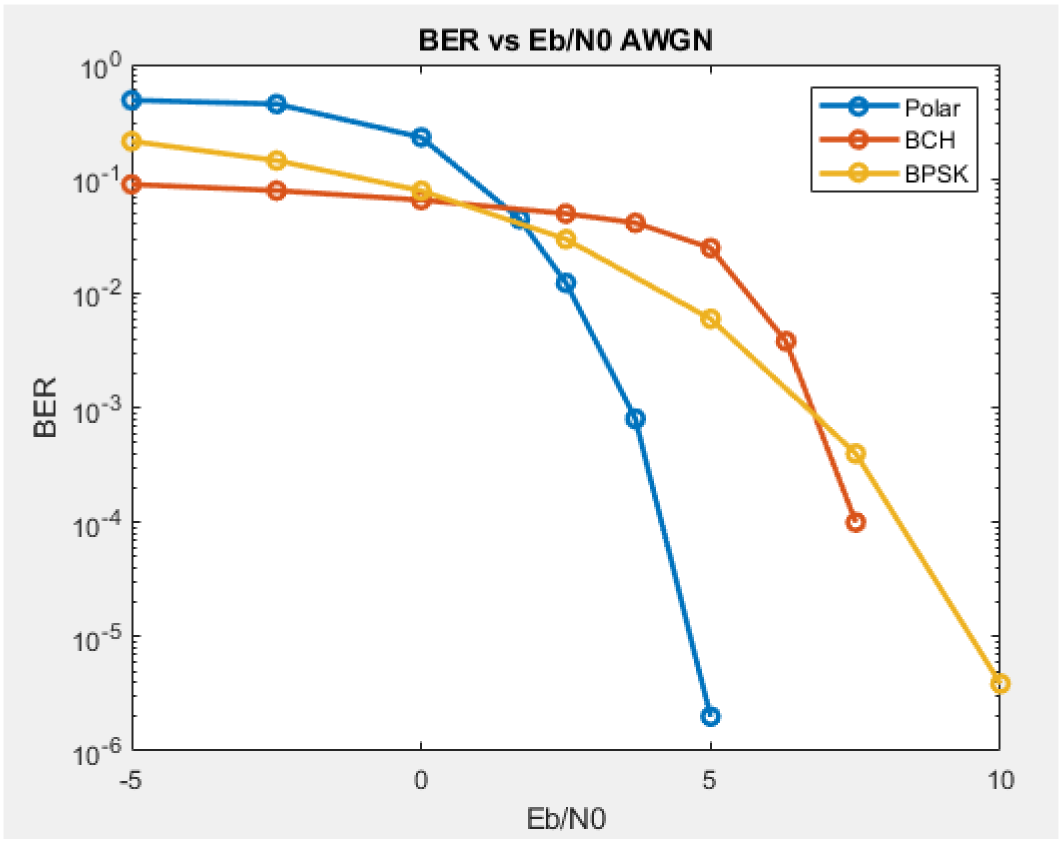

Figure 13:Bit Error rate for BPSK modulation and AWGN channel.

In Figure 13 the simulation results for the Polar and BCH codes can be seen. The horizontal axis represents Eb/N0 values, while the vertical axis represents the Bit Error Rate. Also, for a better comparison, the BER of uncoded BPSK under AWGN channel is computed and shown in the graph as well. As can be seen, for negative values of Eb/N0, the BCH code has better performance than Polar code and the BPSK alone. This is due to its fixed error correcting capability which gives it an edge over the other codes in noisy environments. Then, at 2dB Eb/N0 the polar code starts outperforming the other two and keeps a better performance throughout the whole range of Eb/N0. Moreover, it is noted that from 0 to 6 dB Eb/N0, the BPSK alone performs better than BCH, but from 6 dB Eb/N0, BCH performance increases rapidly, making it better for higher values of Eb/N0.

Figure 14 shows the block error rate for BCH and Polar code. The behavior shows that the Polar code has a better performance. It’s performance keeps improving from -2.5 dB Eb/N0 while, in case of the BCH code, only at about 4 dB Eb/N0 the performance starts improving. This is due to the error correction capability. Since the (255,65) BCH code can correct 30 bit errors per block, until the noise is small enough to corrupt less than 30 bits in a block, the BCH will not be able to correctly decode any codeword. This explains the behavior seen in Figure 14.

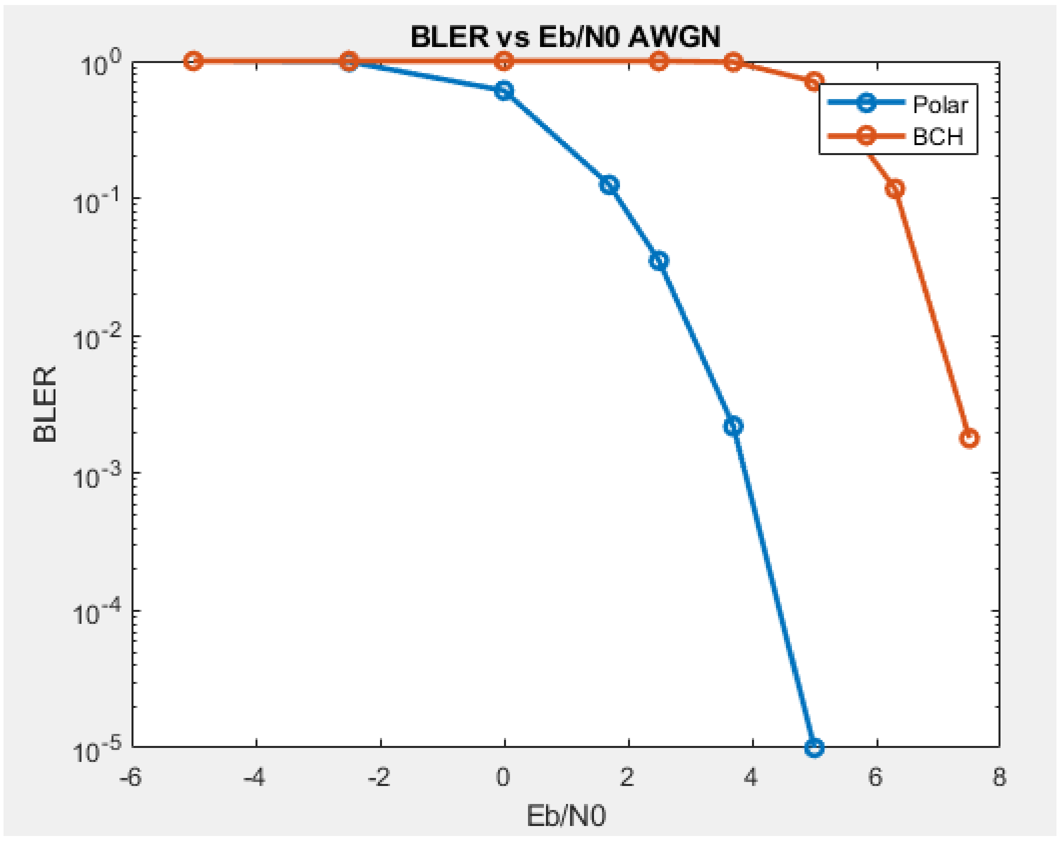

Figure 14: Block error rate for BPSK modulation and AWGN channel.

4.2 Coding schemes with hybrid modulation

In this section, the coding schemes are evaluated in the final system framework with the hybrid modulation and different channel models. The system setup for using both coding and time diversity is as follows. The information vector of 256 bits was split into 4 blocks of data. These blocks, now containing 64 bits each, were encoded using Polar and BCH encoding. After coding each block of 64 bits is encoded into a 256-bit codeword, 4 codewords were ready to be sent. The bits of different codeword are interleaved as can be observed in Figure 15 to make sure that each codeword is spread through the whole packet time so that maximum time diversity is achieved. Then the bits were transmitted using the hybrid modulation technique, each symbol containing one bit from each codeword. This would result in four 256-bit encoded messages which are then decoded, and the results are put together to form the initial 256-bit message.

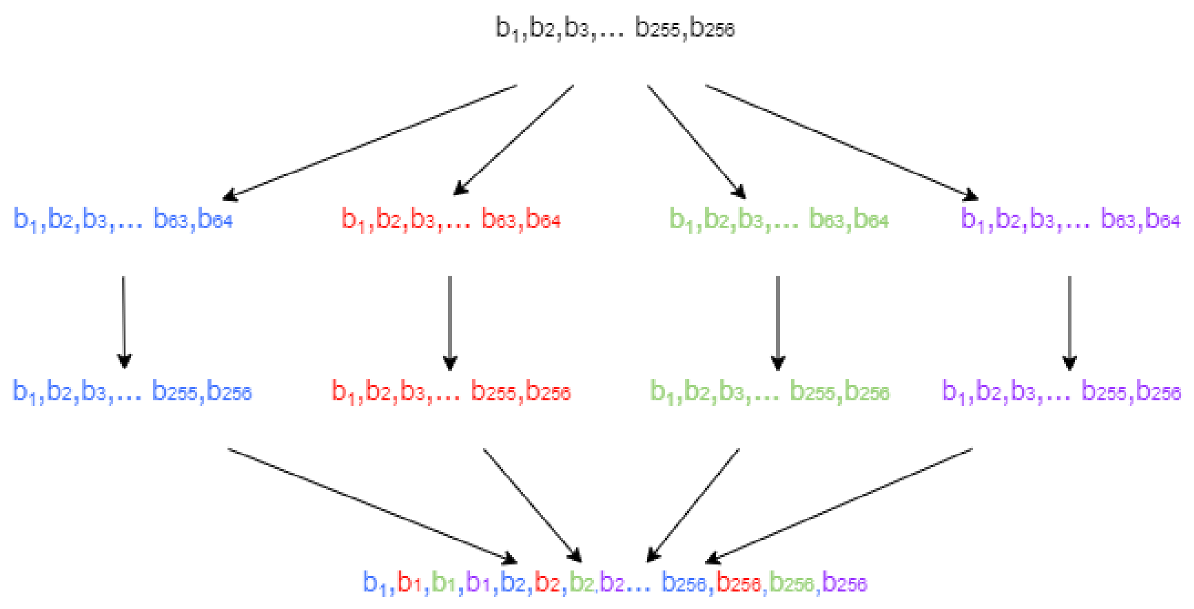

Figure 15: Dividing, Encoding and Interleaving of the message bits.

In our simulations a Rayleigh fading channel with a Jakes Doppler spectrum and a maximum Doppler shift of 2Hz, a Rician (K=1) fading channel with a Rounded Doppler spectrum and a maximum Doppler shift of 2Hz are considered with the coding and interleaving. The simulation results for an AWGN channel in presence of coding is also presented for completeness. The block diagram of the system can be observed in Figure 16.

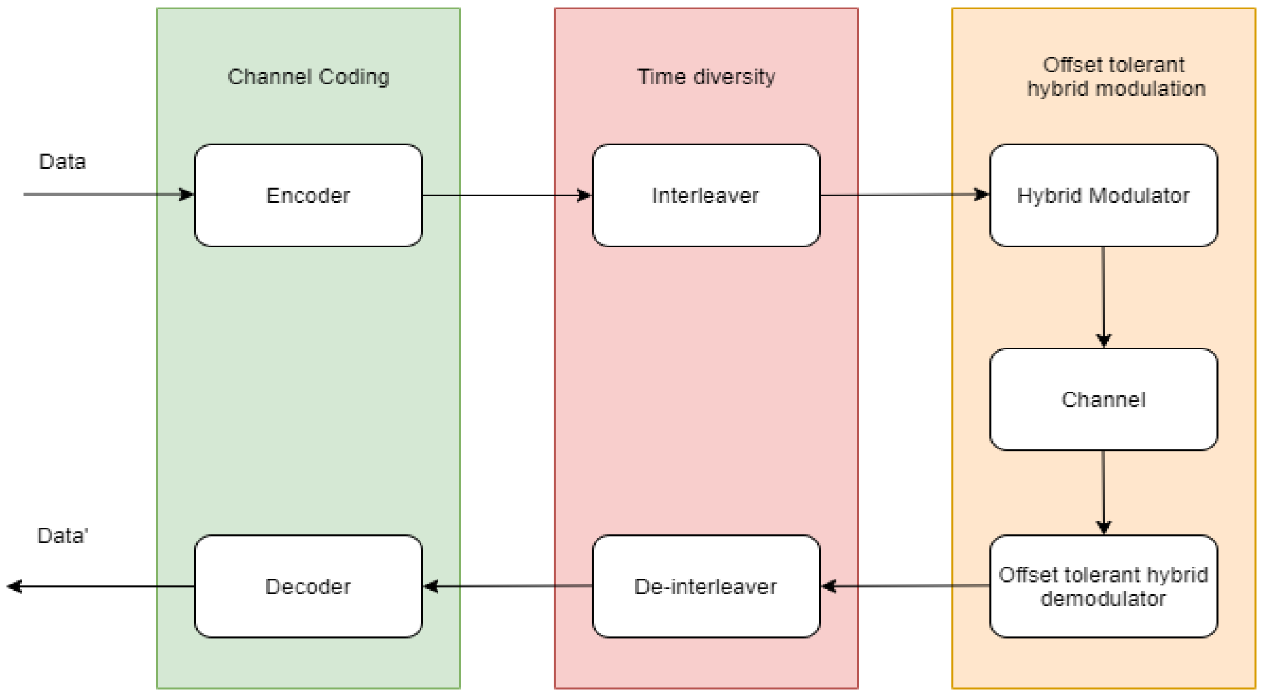

Figure 16: Block diagram of Stage 2 testing environment.

Using this setup, the performance of the coding techniques is investigated under all channels described above. The results can be seen in Figures 17- 22.

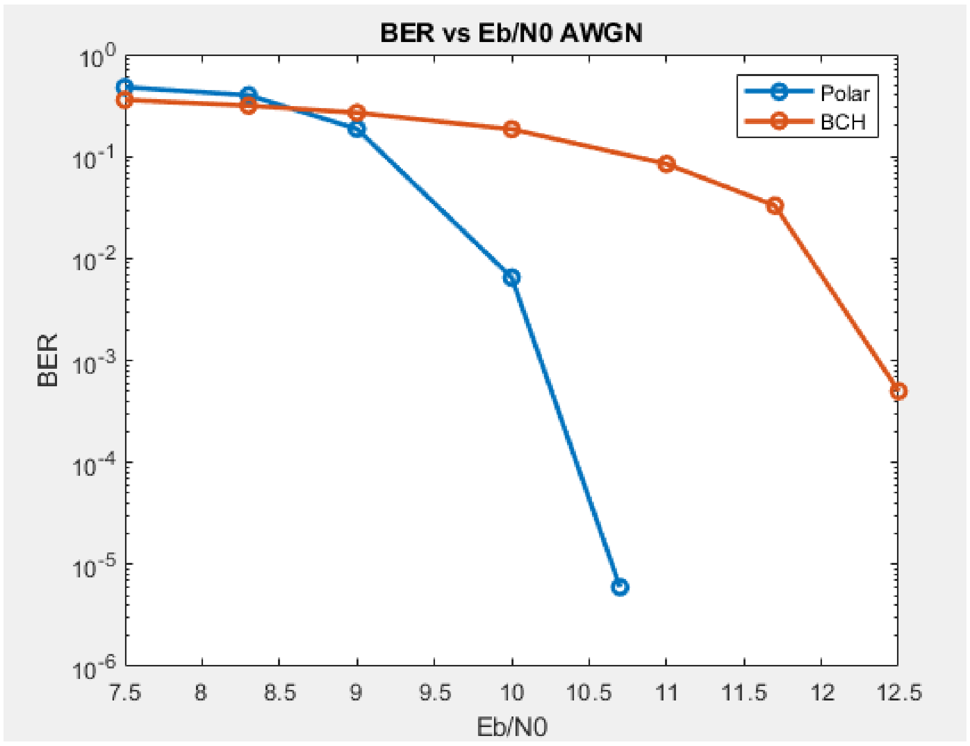

Figure 17: Bit error rate AWGN channel and hybrid modulation.

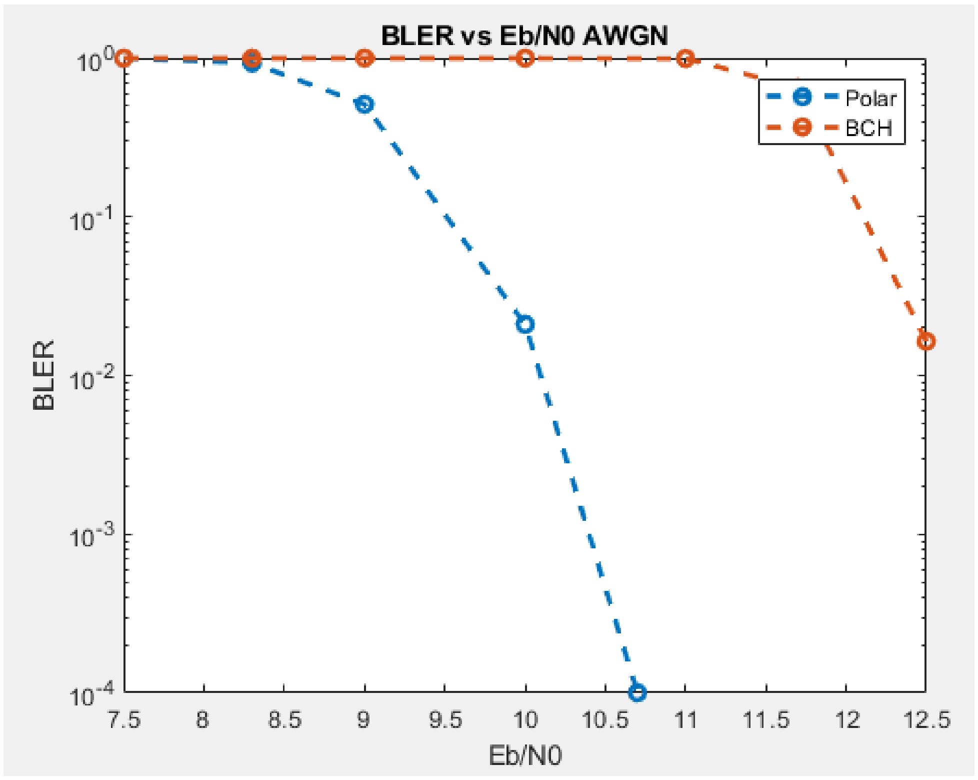

Figure 18: Block error rate AWGN channel and hybrid modulation.

Figure 17 and Figure 18 show the results of BCH and Polar code in the final testing environment under an AWGN channel. As it can be seen in Figure 17, BCH performs slightly better in terms of bit error rate until 8.5dB Eb/N0, where the Polar starts outperforming BCH. In terms of block error rate, as can be seen in Figure 18, the Polar code is showing consistent better performance.

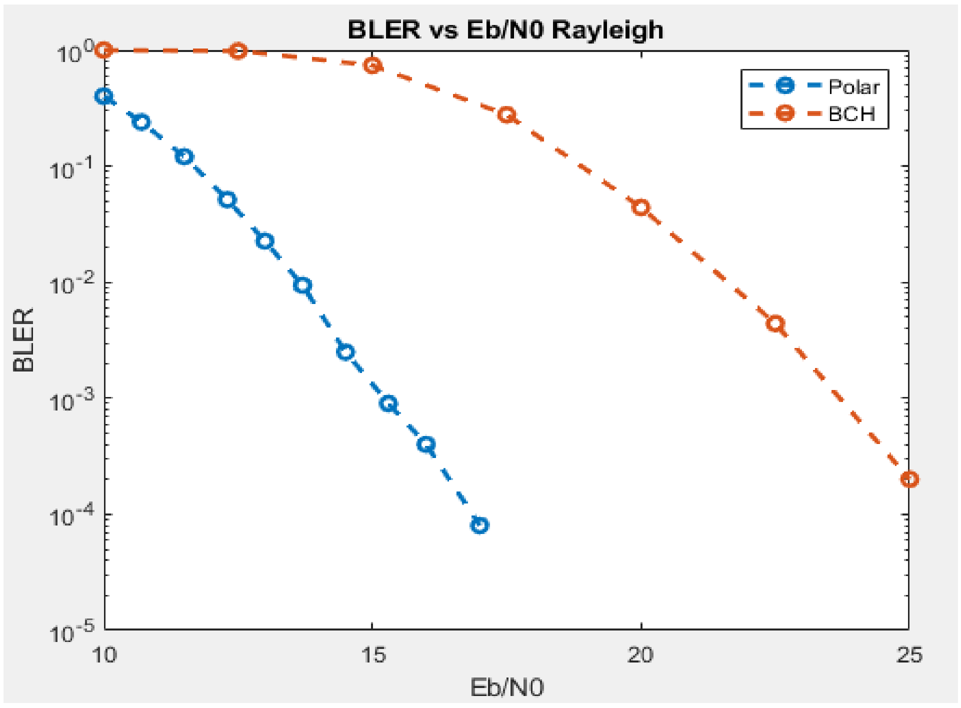

Figure 19: Block error rate vs Eb/N0 under Rayleigh channel

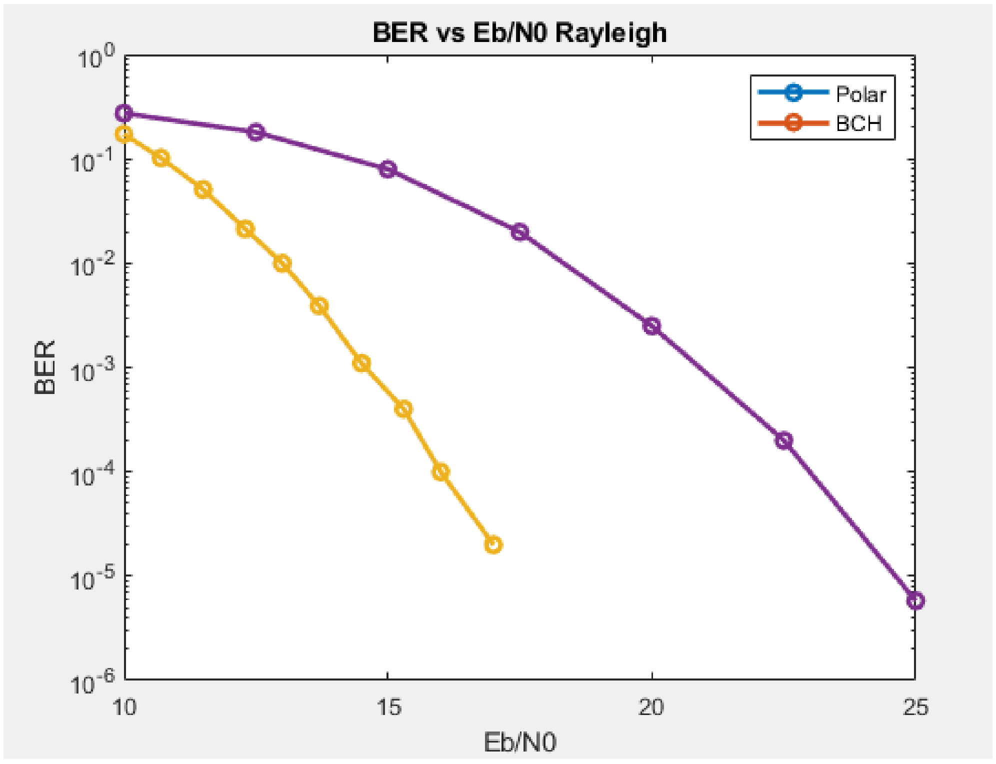

Figure 20:Bit error rate vs Eb/N0 under Rayleigh channel

Figure 19 and 20 show the simulations under a Rayleigh channel. Again, the Polar code has a better performance for all the Eb/N0 simulated values.

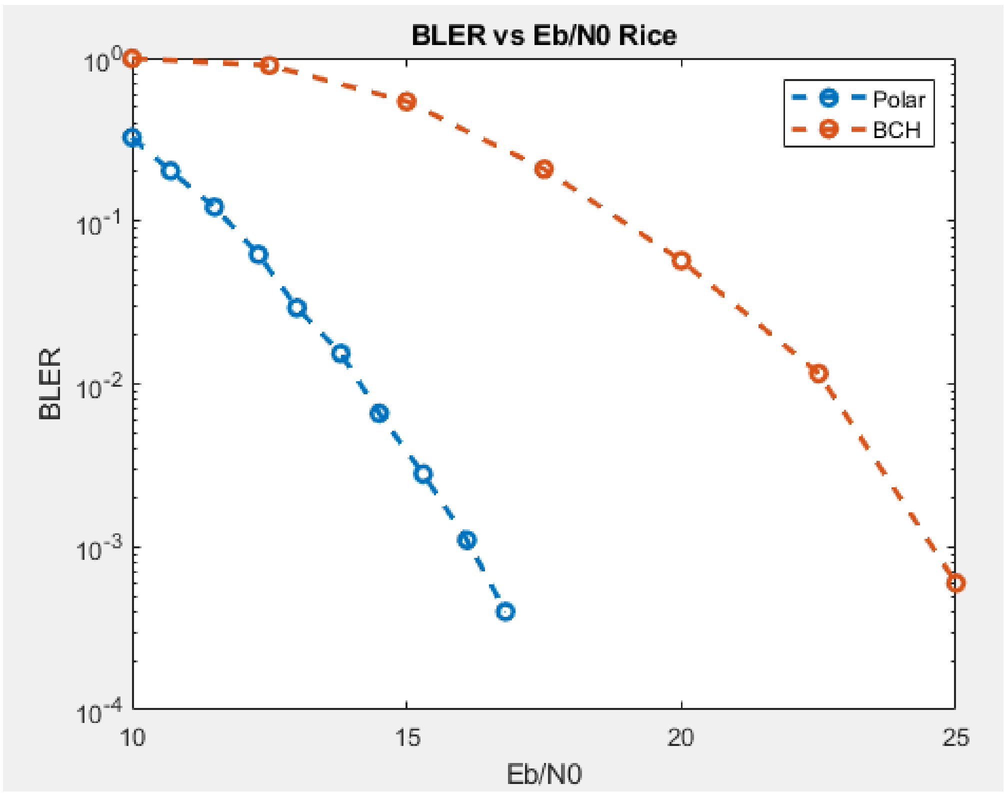

Figure 21: Block error rate vs Eb/N0 under Rician channel

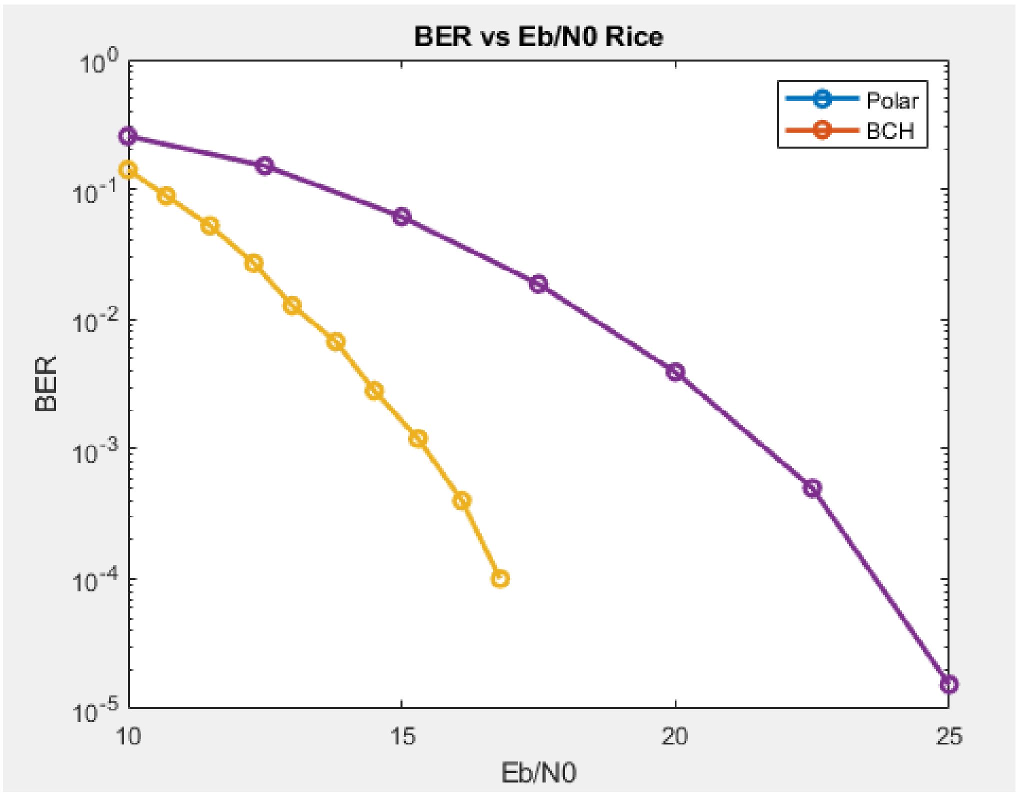

Figure 22:Bit error rate vs Eb/N0 under Rician channel

Figure 21 and 22 show the simulation results for the Rician channel. The Polar code performs better than BCH also in the Rician channel for all the Eb/N0 values simulated.

The Polar code is outperforming the BCH code in the final system setup by a significant margin. This behavior was not expected since after literature review, the BCH code was expected to perform better than Polar code. The reasons for the Polar code outperforming BCH is found by looking at the nature of the decoders used. In [9], where BCH was declared to outperform all other codes which were tested there, including Polar code, an Ordered Statistics Decoder (OSD) was used. This is a soft input decoder. Soft decoders are known to perform better than hard decoders [17]. The decoder used for implementing BCH in this paper was the MATLAB BCH decoder object which is a hard decoder. Thus, the performance shown by BCH is worse than the Polar code that uses a soft decoder.

Moreover, as can be seen in the figures there is another unexpected behavior. The Bit Error Rate curves of the Polar code show a slightly better performance under Rayleigh channel than the Rician channel. Since Rayleigh channel is considered to be worse than Rician, understanding this behavior needs further investigations.

Finally, the results obtained in [1] with the repetition code were compared with the results of the same system while using Polar and BCH coding. The results can be seen in Figures 23 and 24.

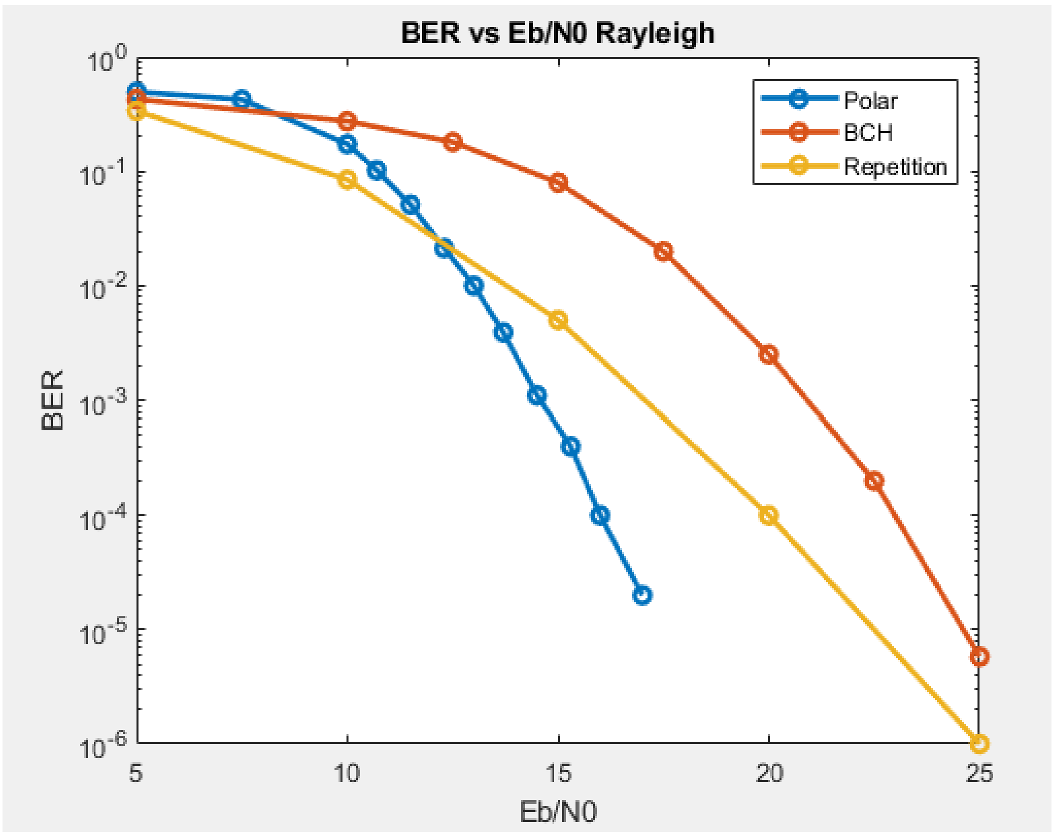

Figure 23:Bit error rate comparison of Polar, BCH and Repetition codes under Rayleigh channel.

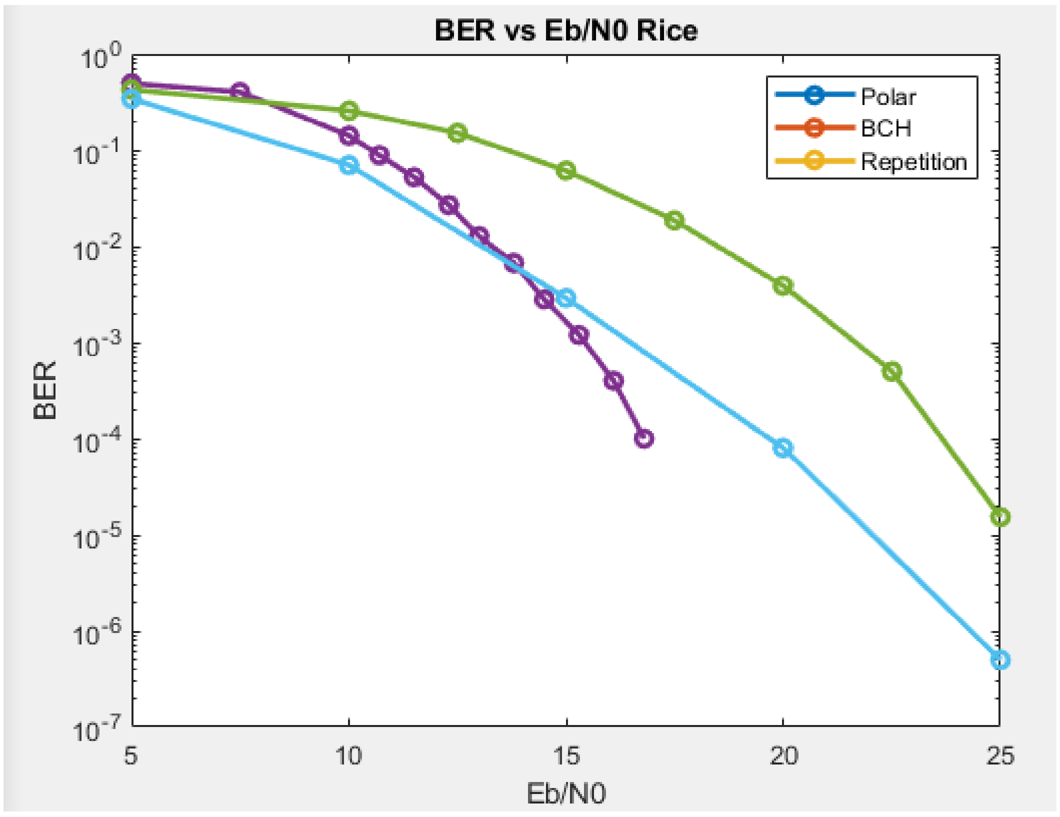

Figure 24: BER comparison of Polar, BCH and Repetition codes under Rician channel.

As can be seen from Figures 23 and 24, the Polar code becomes clearly better than the Repetition code for En/N0 values more than 14 dB under Rician channel and more than 13 dB under Rayleigh channel. Moreover, the repetition coding performs consistently better than BCH for all Eb/N0 values simulated. For large values of Eb/N0, a better performance of BCH in comparison to the repetition is expected since as can be seen in Figure 23 and 24, the error of BCH decreases faster than the repetition BER. This enhances the assumption that, for higher Eb/N0 values, BCH will outperform the repetition coding.

5 Conclusion

The scope of this work was to find suitable candidates for short block Ultra-Narrowband communication and investigate the effect of those on the performance with hybrid modulation and time diversity. Considering the literature, the Polar code was chosen as the main focus due to its low complexity, and BCH was implemented for comparison. The Polar encoder using SC decoding was studied and implemented in MATLAB. Then, MATLAB objects were used to implement the BCH encoder and decoder. The two techniques were compared firstly under BPSK modulation and AWGN channel, and then, in the final system configuration with fading channel and time diversity. The simulation results show that the Polar code performs better in both environments. This behavior can be justified considering the soft decision decoder of Polar code compared to the hard decision of BCH code. Finally, the results of this paper are compared with previous results found in [1] where a repetition code was used as channel coding. It was shown that the Polar code performs better in both Rayleigh and Rician channels compared to the repetition code (as expected), for Eb/N0 values more than 14 db under Rician channel and more than 13 dB under Rayleigh channel. The BCH code had the worst performance among these three codes, performing consistently worse at all simulated Eb/N0 values.

As a conclusion, it can be said that Polar codes are a viable option for the system under investigation due to their decoder simplicity, which is a crucial aspect in saving power, along with good overall performances. For further performance enhancements, instead of a SC decoder, a Successive Cancellation List (SCL) decoder can be considered.

References

[1] S. Safapourhajari and A. B. J. Kokkeler, “Offset Tolerant Demodulator for Frequency/Phase Modulation in Time-Varying Channel,” *IEEE Wireless Communications and Networking Conference, WCNC*, vol. 2019-April, 2019, doi: 10.1109/WCNC.2019.8886086.

[2] R. Sousa, “Evaluation of Narrowband Technologies in Urban Environments,” no. October, 2017.

[3] “What is Carrier Frequency Offset (CFO) and How It Distorts the Rx Symbols | Wireless Pi.” https://wirelesspi.com/what-is-carrier-frequency-offset-cfo-and-how-it-distorts-the-rx-symbols/ (accessed Jun. 30, 2020).

[4] “Physics Tutorial: The Doppler Effect.” https://www.physicsclassroom.com/class/waves/Lesson-3/The-Doppler-Effect (accessed Jun. 30, 2020).

[5] “Sigfox - The Global Communications Service Provider for the Internet of Things (IoT).” https://www.sigfox.com/en (accessed Jul. 02, 2020).

[6] S. Safapourhajari and A. B. J. Kokkeler, “Demodulation of Double Differential PSK in Presence of Large Frequency Offset and Wide Filter,” in *IEEE Vehicular Technology Conference*, Jul. 2018, vol. 2018-June, pp. 1–5, doi: 10.1109/VTCSpring.2018.8417526.

[7] J. van Wonterghem, A. Alloum, J. J. Boutros, and M. Moeneclaey, “Performance comparison of short-length error-correcting codes,” in *2016 Symposium on Communications and Vehicular Technologies (SCVT)*, Nov. 2016, pp. 1–6, doi: 10.1109/SCVT.2016.7797660.

[8] Y. S. Han, “BCH Codes.” Accessed: Jun. 30, 2020. [Online].

[9] M. Shirvanimoghaddam *et al.*, “Short Block-Length Codes for Ultra-Reliable Low Latency Communications,” *IEEE Communications Magazine*, vol. 57, no. 2, pp. 130–137, 2019, doi: 10.1109/MCOM.2018.1800181.

[10] A. Glavieux and S. Vaton, “Convolutional Codes,” *Channel Coding in Communication Networks: From Theory to Turbocodes*, vol. 2010, pp. 129–196, 2010, doi: 10.1002/9780470612422.ch3.

[11] “Turbo Codes Coding and Communication Laboratory.” Accessed: Jun. 30, 2020. [Online].

[12] “Turbo code - Wikipedia.” https://en.wikipedia.org/wiki/Turbo_code (accessed Jun. 30, 2020).

[13] E. Arikan, N. Ul Hassan, M. Lentmaier, G. Montorsi, and J. Sayir, “Challenges and some new directions in channel coding,” *Journal of Communications and Networks*, vol. 17, no. 4, pp. 328–338, 2015, doi: 10.1109/JCN.2015.000063.

[14] “Design and Optimization for 5G Wireless Communications - Haesik Kim - Google Cărți.” https://books.google.ro/books/about/Design_and_Optimization_for_5G_Wireless.html?id=MtvWDwAAQBAJ&redir_esc=y (accessed Jun. 30, 2020).

[15] E. Arikan, “Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,” *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009, doi: 10.1109/TIT.2009.2021379.

[16] A. Dr. Thangaraj, “NPTEL :: Electronics & Communication Engineering - NOC:LDPC and Polar Codes in 5G Standard.” https://nptel.ac.in/courses/108/106/108106137/ (accessed Jun. 30, 2020).

[17] A. Dr. Thangaraj, “NPTEL :: Electronics & Communication Engineering - Coding Theory.” https://nptel.ac.in/courses/117/106/117106031/# (accessed Jun. 30, 2020).

6 Appendix

Implementation of Polar code used with BPSK and AWGN channel.

```
clear all;
%Reliability sequence
Q=[0 1 2 4 8 16 32 3 5 64 9 6 17 10 18 128 12 33 65 20 256 34
24 36 7 129 66 512 11 40 68 130 ...
    19 13 48 14 72 257 21 132 35 258 26 513 80 37 25 22 136
260 264 38 514 96 67 41 144 28 69 42 ...
    516 49 74 272 160 520 288 528 192 544 70 44 131 81 50 73
15 320 133 52 23 134 384 76 137 82 56 27 ...
    97 39 259 84 138 145 261 29 43 98 515 88 140 30 146 71 262
265 161 576 45 100 640 51 148 46 75 266 273 517 104 162 ...
    53 193 152 77 164 768 268 274 518 54 83 57 521 112 135 78
289 194 85 276 522 58 168 139 99 86 60 280 89 290 529 524 ...
    196 141 101 147 176 142 530 321 31 200 90 545 292 322 532
263 149 102 105 304 296 163 92 47 267 385 546 324 208 386 150
153 ...
    165 106 55 328 536 577 548 113 154 79 269 108 578 224 166
519 552 195 270 641 523 275 580 291 59 169 560 114 277 156 87
197 ...
    116 170 61 531 525 642 281 278 526 177 293 388 91 584 769
198 172 120 201 336 62 282 143 103 178 294 93 644 202 592 323
392 ...
    297 770 107 180 151 209 284 648 94 204 298 400 608 352 325
533 155 210 305 547 300 109 184 534 537 115 167 225 326 306
772 157 ...
    656 329 110 117 212 171 776 330 226 549 538 387 308 216
416 271 279 158 337 550 672 118 332 579 540 389 173 121 553
199 784 179 ...
    228 338 312 704 390 174 554 581 393 283 122 448 353 561
203 63 340 394 527 582 556 181 295 285 232 124 205 182 643 562
286 585 ...
    299 354 211 401 185 396 344 586 645 593 535 240 206 95 327
564 800 402 356 307 301 417 213 568 832 588 186 646 404 227
896 594 ...
    418 302 649 771 360 539 111 331 214 309 188 449 217 408
609 596 551 650 229 159 420 310 541 773 610 657 333 119 600
339 218 368 ...
    652 230 391 313 450 542 334 233 555 774 175 123 658 612
341 777 220 314 424 395 673 583 355 287 183 234 125 557 660
616 342 316 ...
    241 778 563 345 452 397 403 207 674 558 785 432 357 187
236 664 624 587 780 705 126 242 565 398 346 456 358 405 303
569 244 595 ...
    189 566 676 361 706 589 215 786 647 348 419 406 464 680
801 362 590 409 570 788 597 572 219 311 708 598 601 651 421
792 802 611 ...
```

```
    602 410 231 688 653 248 369 190 364 654 659 335 480 315
221 370 613 422 425 451 614 543 235 412 343 372 775 317 222
426 453 237 ...
    559 833 804 712 834 661 808 779 617 604 433 720 816 836
347 897 243 662 454 318 675 618 898 781 376 428 665 736 567
840 625 238 ...
    359 457 399 787 591 678 434 677 349 245 458 666 620 363
127 191 782 407 436 626 571 465 681 246 707 350 599 668 790
460 249 682 ...
    573 411 803 789 709 365 440 628 689 374 423 466 793 250
371 481 574 413 603 366 468 655 900 805 615 684 710 429 794
252 373 605 ...
    848 690 713 632 482 806 427 904 414 223 663 692 835 619
472 455 796 809 714 721 837 716 864 810 606 912 722 696 377
435 817 319 ...
    621 812 484 430 838 667 488 239 378 459 622 627 437 380
818 461 496 669 679 724 841 629 351 467 438 737 251 462 442
441 469 247 ...
    683 842 738 899 670 783 849 820 728 928 791 367 901 630
685 844 633 711 253 691 824 902 686 740 850 375 444 470 483
415 485 905 ...
    795 473 634 744 852 960 865 693 797 906 715 807 474 636
694 254 717 575 913 798 811 379 697 431 607 489 866 723 486
908 718 813 ...
    476 856 839 725 698 914 752 868 819 814 439 929 490 623
671 739 916 463 843 381 497 930 821 726 961 872 492 631 729
700 443 741 ...
    845 920 382 822 851 730 498 880 742 445 471 635 932 687
903 825 500 846 745 826 732 446 962 936 475 853 867 637 907
487 695 746 ...
    828 753 854 857 504 799 255 964 909 719 477 915 638 748
944 869 491 699 754 858 478 968 383 910 815 976 870 917 727
493 873 701 ...
    931 756 860 499 731 823 922 874 918 502 933 743 760 881
494 702 921 501 876 847 992 447 733 827 934 882 937 963 747
505 855 924 ...
    734 829 965 938 884 506 749 945 966 755 859 940 830 911
871 639 888 479 946 750 969 508 861 757 970 919 875 862 758
948 977 923 ...
    972 761 877 952 495 703 935 978 883 762 503 925 878 735
993 885 939 994 980 926 764 941 967 886 831 947 507 889 984
751 942 996 ...
    971 890 509 949 973 1000 892 950 863 759 1008 510 979 953
763 974 954 879 981 982 927 995 765 956 887 985 997 986 943
891 998 766 ...
    511 988 1001 951 1002 893 975 894 1009 955 1004 1010 957
983 958 987 1012 999 1016 767 989 1003 990 1005 959 1011 1013
895 1006 1014 1017 1018 ...
    991 1020 1007 1015 1019 1021 1022 1023]+1;
tic
N = 256;
```

```
K = 64;
n = log2(N);
EbNodB = 5;
Rate = K/N;
EbNo = 10^(EbNodB/10);
sigma = sqrt(1/(2*Rate*EbNo));

Q1 = Q(Q<=N); %reliability sequence for N

F = Q1(1:N-K); %Frozen positions: Q1(1:N-K)
%Message positions: Q1(N-K+1:end)

%Simulate
Nbiterrs = 0; Nblkerrs = 0; Nblocks = 10000;
for blk = 1:Nblocks
    msg = randi([0 1],1,K); %generate random K-bit message

    u = zeros(1,N);

    u(Q1(N-K+1:end)) = msg; %assign message bits
    %Encoder used to test with BPSK and AWGN channel

    m = 1; %number of bits combined
    for d = n-1:-1:0
        for i = 1:2*m:N
            a = u(i:i+m-1); %first part
            b = u(i+m:i+2*m-1); %second part
            u(i:i+2*m-1) = [mod(a+b,2) b]; %combining
        end
        m = m * 2;
    end
    cword = u;
    %End of encoder

    %Using BPSK and adding white gaussian noise
    s = 1 - 2 * cword; %BPSK bit to symbol conversion
    r = s + sigma * randn(1,N); %AWGN channel I

    %SC decoder for BPSK and AWGN
    L = zeros(n+1,N); %beliefs
    ucap = zeros(n+1,N); %decisions
    ns = zeros(1,2*N-1); %node state vector

    f = @(a,b) (1-2*(a<0)).*(1-2*(b<0)).*min(abs(a),abs(b));
%minsum
    g = @(a,b,c) b+(1-2*c).*a; %g function

    L(1,:) = r; %belief of root

    node = 0; depth = 0; %start at root
    done = 0; %decoder has finished or not
    while (done == 0) %traverse till all bits are decoded
        %leaf or not
        if depth == n
            if any(F==(node+1)) %is node frozen
                ucap(n+1,node+1) = 0;
            else
                if L(n+1,node+1) >= 0
                    ucap(n+1,node+1) = 0;
                else
                    ucap(n+1,node+1) = 1;
                end
            end
            if node == (N-1)
                done = 1;
            else
                node = floor(node/2); depth = depth - 1;
            end
        else
            %nonleaf
            npos = (2^depth-1) + node + 1; %position of node
in node state vector
            if ns(npos) == 0 %step 1 and go to left child

                temp = 2^(n-depth);
                Ln = L(depth+1,temp*node+1:temp*(node+1));
%incoming beliefs
                a = Ln(1:temp/2); b = Ln(temp/2+1:end); %split
beliefs into 2
                node = node *2; depth = depth + 1; %next node:
left child
                temp = temp / 2; %incoming belief length for
left child
                L(depth+1,temp*node+1:temp*(node+1)) = f(a,b);
%minsum and storage
                ns(npos) = 1;
            else
                if ns(npos) == 1 %step 2 and go to right child

                    temp = 2^(n-depth);
                    Ln = L(depth+1,temp*node+1:temp*(node+1));
%incoming beliefs
                    a = Ln(1:temp/2); b = Ln(temp/2+1:end);
%split beliefs into 2
                    lnode = 2*node; ldepth = depth + 1; %left
child
                    ltemp = temp/2;
                    ucapn =
ucap(ldepth+1,ltemp*lnode+1:ltemp*(lnode+1)); %incoming
decisions from left child
                    node = node *2 + 1; depth = depth + 1;
%next node: right child
```

```
                    temp = temp / 2; %incoming belief length 
for right child
                    L(depth+1,temp*node+1:temp*(node+1)) = 
g(a,b,ucapn); %g and storage
                    ns(npos) = 2;
                else %step 3 and go to parent
                    temp = 2^(n-depth);
                    lnode = 2*node; rnode = 2*node + 1; cdepth 
= depth + 1; %left and right child
                    ctemp = temp/2;
                    ucapl = 
ucap(cdepth+1,ctemp*lnode+1:ctemp*(lnode+1)); %incoming 
decisions from left child
                    ucapr = 
ucap(cdepth+1,ctemp*rnode+1:ctemp*(rnode+1)); %incoming 
decisions from right child
                    ucap(depth+1,temp*node+1:temp*(node+1)) = 
[mod(ucapl+ucapr,2) ucapr]; %combine
                    node = floor(node/2); depth = depth - 1;
                end
            end
        end
    end

    msg_cap = ucap(n+1,Q1(N-K+1:end));
    %End of SC decoder

    %Counting errors
    Nerrs = sum(msg ~= msg_cap);
    if Nerrs > 0
        Nbiterrs = Nbiterrs + Nerrs;
        Nblkerrs = Nblkerrs + 1;
    end
end
toc
BERP = Nbiterrs/K/Nblocks;
FERP = Nblkerrs/Nblocks;

disp([EbNodB FERP BERP])
```