Channel Coding: Theory and Applications in LTE Networks

Fe-eze Anyafulu
Electrical and Electronics Engineering
City, University of London
EC1V 0HB, London
Website: *fe-eze.com*

***Abstract*—This paper examines the Concept of Channel Coding in Digital communications system, it starts with a brief summary of Information theory and Source coding then moves on to channel coding concepts like Rectangular codes, Hamming Distance and Odd parity. Next, linear block codes were introduced and finally Convolutional codes were explored culminating in their main application in LTE systems (i.e Turbo Codes). SIMULINK models were then used to simulate the Channel coding schemes in a wider communications system over a wide range of parameters, the results clarified the reasons why certain standards are used in modern communication systems (6144 bit codeblocks in LTE communications and 1/3-coderates for tailbiting codes).**

I. Introduction

Channel Coding is done in a Digital communications system to ensure that bits sent from a source and transmitted through a noisy analog channel has a very high chance of reaching the intended reciever intact (i.e without disruption). To show how to achieve this, we shall slowly build upon concepts such as Hamming distance, Linear block coding, Syndrome decoding, Convolutional Coding, Cyclic redundancy Check, Viterbi Decoding and finally Turbo Coding which is the method used in modern communications systems (most notably LTE). We shall cover concepts such as the channel (modelled with Additive White Gaussian Noise) and the methods (mentioned above) used to ensure a high probablility of intact reception later in the paper, but first we need some background in information theory and source coding. This will lay the groundwork on which we shall build for the rest of this paper.

II. Literature review - theoretical background of Channel Coding concepts

A. Information Theory

In his 1948 paper[1], C.E. Shannon, building on earlier work by R.V. Harley[2], put forth a general definition of information. He surmised that information, regardless of the semantics involved, can be defined.

Defining Information -The higher the probablity of an event happening, the smaller the actual information associated with knowing that the event has occured. For example, if at the start of the 2015/2016 footbal season we know that Manchester United has a 0.4 probability of winning the Premier league and Leicester City has a 0.0002 probability of winning then a news item at the end of the season saying Leicester won the league has a lot more information than one that announces Manchester Utd as the winner.

This notion of more infomation being gleaned from a small probability event than a high one can be put into math as

$$Information\,learned = log(\frac{1}{P})$$

where P is the associated probability of the event happening Information is measured in bits

The bit can be thought of as a measure of surprise, an unlikely (more surprising) event has more bits of information than a more probable (less surprising) event. Using our above example, the information conveyed in a Man Utd victory is $log(\frac{1}{0.4}) = 0.39bits$ while the information in a Leicester Victory is $log(\frac{1}{0.0002}) = 3.69bits$

1) Entropy: From the above introduction we have a way to measure the amount of information contained in a given event, next we shall quantify the expected/average information in a set of mutually exclusive events. If an event i occurs with probability $p_i(1 \leq N)$ out of a set of N mutually exclusive events, then the average information conveyed is:

$$H(p_1, p_2, ..., p_N) = \sum_{i=1}^{N} p_i log(\frac{1}{P_i})$$

This is called the Entropy; the information in a given event $log(\frac{1}{P_i}$ weighted by its probability of occuring (P_i). Entropy is also measured in bits.

Next, we consider a binary memoryless source (Memoryless means that the output is statistically independent of all other outputs) for which 0 occurs with a probability p_o and 1 occurs with a probability of $p_1 = 1 - p_0$. So, the probability of one of two mutually exclusive events occuring;

$$H(p_0, 1-p_0) = p_0 log(\frac{1}{p_0}) + (1-p_0) log(\frac{1}{1-P_0})$$

$$H(p_0, 1-p_0) = -p_0 log(p_0) - (1-p_0) log(1-P_0) bits$$

The case H is usually just refered to as H(p)

B. Source Coding

In source coding, the goal is to represent discrete data in an efficient form. The device that does this is called a source encoder. In order for our encoder to be efficient, we need to have knowledge about the statistics of the data being sent, with

such knowledge we can determine which parts of the data conveys less information and represent them with less bits. An example would be the English alphabet, more commonly occuring letters like 'a' and 'e' occur with a much higher probability, hence we can use less bits to represent such letters.

There are two broad types of source coding: Fixed coding and variable length coding. An example of fixed coding is ASCII (American Standard Code for Information Interchange) where each of the letters of the alphabet are represented by an 8-bit codeword. In Fixed coding, the number of bits needed to send a particular information is always constant, so an ASCII text file of 1000 words will take 8000 bits to send. This is where variable length coding has an edge over fixed coding as we can reduce the amount of bits needed to send the same message without affecting the actual data being transfered (this is called lossless compression). Something to note about variable length coding is that it gets closer to the theoretical entropy the longer the message is

Entropy gives us the theoretical maximum compression we can achieve. A good source code is one that gets us as close to this theoretical limit as possible. We shall briefly explore two types of source codes

1) Huffman: Huffman coding [6] offers a method for constructing minimum reundancy codes hence compressing it. Huffman developed his coding scheme based on the probability of the letters in the English alphabet. An example of a huffman code tree is shown below

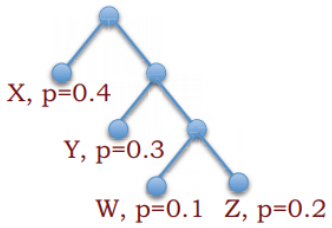


Fig. 1. Huffmann Code tree

2) Lempel-Ziv: The Lempel-Ziv Algorithm[9] is a variable rate lossless data compression algorithm. Contemporary uses include the GIF image format. It is further explained below:

We shall consider sending the following string with Lempel-Ziv. "abcabcabcabcabcabcabcabcabcabcabc"

Although it is highly unlikely that we get a message that repeats itself twelve times like the one above, it is useful to demonstrate the compression that can be achieved with Lempel Ziv when compared to a fixed code like ASCII. The codebook for Lempel-Ziv is based on the standard 8-bit ASCII codebook. An extra bit is added in order to make space for the Algorithm to generate extra codewords as it parses the text. For the table that demonstrates how this is done, See Table IV.

Result: We get the following 9-bit codewords

[97][98][99][256][258][257][259][262][261][264][260][266][263][99]

This means that the message will take 9*14 = 126 bits to transmit. The same message would have taken 8*36=288 bits with ASCII. We see a dramatic 66.25% reduction in the message to be transmitted.

Decoding Lempel Ziv The decoder needs no knowledge of the transmitter's codebook as once the starting point is known (in this case ASCII), the decoder can generate its own dictionary dynamically and accurately decode the message.

C. Channel Coding Concepts

In Channel Coding, the goal is the exact opposite of Source Coding. Whereas the goal in Source Coding is to maximize the efficiency of the message to be transmitted by compression, Channel Coding aims to maximize the integrity of the message at the receiver. This is usually done by adding redundancy to the code so that it can withstand errors that may be generated over the channel.

1) Binary Symmetric Channel: In a Binary Symmetric Channel (BSC), the probablililty of flipping a bit, independent of other bits is assumed to be less than half i.e. $P(of\ flipping\ a\ bit) = \epsilon (< \frac{1}{2})$, so if we send N bits over our communications link we expect to receive $N.\epsilon$ bits at the receiver. We can find this probablility(ϵ by sending a long string of known bits over a channel, because of ***law of large numbers:*** *The average results from a large number of trials tends toward the expected value as more trials are performed.* In a BSC, the Probability of interpreting a bit b as $1 - b = \epsilon$.

A BSC is a simple model that can help us understand the intuition of a channel. However, real world channels can exhibit much more complex behaviours. One of these behaviours is burst errors, where the error is dependent on the success of the previous bit. We shall come back to this later in this paper.

2) A Naive solution to channel coding: The easiest and most naive solution to channel coding will be to use repitition, We shall repeat each bit n times and use majority rule to decide the correct bit at the receiver. The problem with repittion code is that we can only get $\frac{1}{n}$ amount of information across. So if we repeat each bit 3 times, we only use 1/3 of the channel.

We have a bitstream of n redundant bits attached to each message bit, so if n is odd the message can be decoded because the majority vote decides which bit was intended. On the other hand, if n is even then the receiver has to arbitrarily decide what the intended bit. Repitition code can achieve an exponential decrease in errors with each increase in redundant bits n but this comes at the cost of extra overhead (remember that our code rate is 1/n).

We shall explore two ideas that will play a central role in understanding Channel Coding: *(1)*Embedding messages to achieve structural separation *(2)*Parity (linear) computations over the message bit

3) Hamming Distance: The Hamming Distance is the minimum disance between any two codewords in a coding scheme. We can help the receiver to better detect errors by spreading out our codewords.

A coding scheme with a minimum hamming distance of D can detect any error pattern of D-1 or fewer errors. We can correct more errors by further increasing the space so that the valid codewords in our coding scheme are never too close to each other. For example if $S = 0, 1$, we can devise a 3 bit coding scheme from the set $\{000, 001, 010, 011, 100, 101, 110, 111\}$ so now we interpret the message to be 0 if we receive 000, 001, 100, 010 or 1 if we receive 011, 110, 101, 111. Hence we can correct an error if we know that only one error occured. This is visualized in figure 2

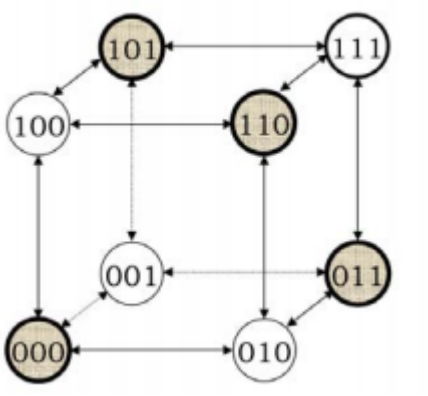


Fig. 2. Code with Hamming distance of 3

For a certain hamming distance d, we can detect $(d-1)$ errors and correct $floor(\frac{d-1}{2})$ errors. If we want to send 4-bit messages with $d = 3$ and single bit error correction, we need a (7,4) hamming code

4) Simple code parity check: In Parity calculation, We want to make sure a bitrstream always has an even number of 1s and we do this by adding a parity bit

D. Linear Block codes

We have K message bits and 2^K messages. We have n code bits and 2^n possible messages but we can only use 2^K codewords. Our goal here is to unambiguosly detect errors

In Linear block codes, we take blocks of K message bits encoded into n message bits with thte sum of the two codewords also being a codeword

$$C_{1..n} = D_{1..K}.G$$

where C is the codeword D is the message and G is the generator matrix. All operations are in GF(2) (i.e -x=x)

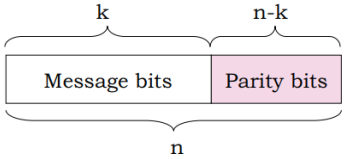


Fig. 3. Data bits and Parity Bits

1) Rectangular parity codes: An example of linear block code: We want even parity on each row and column P= sum

D_1	D_2	D_3	D_4	P_1
D_5	D_6	D_7	D_8	P_2
D_9	D_{10}	D_{11}	D_{12}	P_3
P_4	P_5	P_6	P_7	P

Fig. 4. (16,8) Rectangular Parity Code with parity bits highlighted in pink

of data bits in column or row Adding all the bits on each row/column should give us zero

The generator matrix $G_{(K*n)} = [I_{K*K}|A_{K*(n-k)}]$

2) Single error correction codes: Hamming distance of rectangular code is

$$Code\ rate = \frac{rc}{rc + (r + c)}$$

rc here is the number of message bits while (r+c) is the number of parity bits (overall parity)

Example: (12,7) code: This code can detect two errors but only correct one

E. Syndrome Decoding

We have a K dimension subspace of vector space n and anything outside K is an obvious error. Parity equation: $P_j = \sum_{i=1}^{K} D_i a_{ij}$; Parity relation: $P_j + \sum_{i=1}^{K} D_i a_{ij} = 0$

Syndrome decoding is an efficient way of decoding linear block codes. For single bit errors, if rectangular word has zero or one error the decoder will return the correct transmitted message So, in a (7,4) code where

$$P_1 = D_1 + D_2 + D_3$$

$$P_2 = D_1 + D_3 + D_4$$

$$P_3 = D_2 + D_3 + D_4$$

we can simplify this, since we are in GF(2)

$$0 = P_1 + D_1 + D_2 + D_3$$

$$0 = P_2 + D_1 + D_3 + D_4$$

$$0 = P_3 + D_2 + D_3 + D_4$$

We can compare each receive n bit word to each valid codeword (2^k) and correct any one that is a Hamming distance of 1 away from a valid code but dis doesn't exploit the code's linear characteristics. We do that with syndromes $H.c^T = 0$ for any received word r without errors $H.r^T = 0$

$$H.(C + error)^T = S$$

S is our syndrome

Process

1) Compute all syndromes $H.E^T = S$
2) Compute received word syndrome
3) Compare received word syndrome to our valid syndromes and apply error correction $C = R = E_c$

This can also work for independent multi bit errors but we also want to correct burst errors (correlated multi-bit errors)

1) Burst errors: So far, we have imagined a Binary Symmetric channel (where one error does not affect other bits). Imagine the channel has a 'good' and 'bad' state so, $P_{error,b} > P_{error,g}$ We want to turn a B-bit error burst in to into B single bit errors

2) Interleaving: Instead of sending blocks row by row we send blocks column by columns so we can spread a multi bit error into many single bit errors which we can then detect and correct accordingly

3) Framing: How do we know when a new block starts? We do thins with framing A frame = Synchronous pattern + interleaved codeword block The synchronous pattern has to be unique and it can't be protected by Error correcting code, so we might lose frames every now and then.

4) Summary of channel coding steps:

- Break message into K-bit blocks
- Add redundancy (parity bits)
- Interleave to protect against burst errors
- Add unique pattern of Synchronous bits
- Bit stuff message to make sure message can never contain synchronous bit. for example if our synchronous bit is a pattern on six 1s (111111), we make sure that whever we encounter 5 consequtive 1s in our message we add a zero bit to it. The decoder does the opposite by removing the zeros to read and decode the message properly
- Finally, the new bitstream is sent to the transmitter

F. Convolutional Codes

Convolutional codes use a sliding window to transmit only the parity bits. It is so named because it uses the convolution operation to generate its codes.

$$P_i[n] = (\sum_{j=0}^{K-1} g_i[j].x[n-j])mod2$$

where g_i is the set of weights and K is the maximum window (constraint length)

We can view convolutional coding in two ways

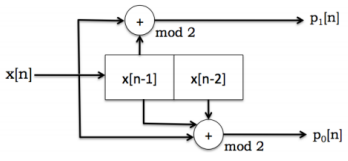


Fig. 5. Shift Register view of convolutional code

1) Shift register view: Register size is K and it overwrites/discards previous values as it goes along

2) State machine view: The state machine view is identical for all codes of length K. It always has 2^{K-1} states

G. The Trellis diagram

It is a convinient way of viewing the decoding task and understanding the changes in the state machine over time. See Figure 6.

H. The Viterbi Algorithm

The Viterbi Algorithm is an elegant method of decoding convolutional codes invented by Andrew Viterbi in 1957 [12]. The Viterbi algorithm is widely used in modern disk drives and speech recognition systems.

1) Soft and hard decoding: When the parity bits arrive at the receiver, they come as an analog signal and the receiver digitizes this signal by first sampling the signal and then demapping it (comparing it to the treshold voltage). This digitized signal is then sent to the decoder

In Hard desicion decoding, we proceed to decode the message from the received bit sequence without any accompanying information from the receiver

In Soft desicion decoding, the decoder is given the sampled analog signal instead (or in addition to) the demapped bitstream from the receiver. This information is then used to decode the message.

2) How the Viterbi Decoder works: We start by assigning a branch metric of 0 for all four states. The path metric is calculated by finding the Hamming distance between each state transition and the received codeword, this value is then added to the previous branch metric. We do this for all eight paths in the time and we kill the branch with the higher path metric. This now becomes the branch metric for the new time t+1. We then repeat until we get to the end of the received bit sequence. Finally, we traceback from the lowest branch metric in our last branch to the zero state in our first branch. The is carried in our final traced back path as the information bit accompanying each state transition (path). Thus our message can be decoded.

3) Illustration: A message (1001100) encoded with convolutional coding and has the parity bit stream of 11 10 11 11 01 01 11. The parity bits are received at the other end and the receiver now needs to decode the original message. The trellis diagram below illustrates this

Explaining figure7:

- The state transitions with their accompanying information bits are in black while the parity bitstream that we want to decode is at the bottom (blue) in pairs
- The Hamming distance is in blue; The Hamming distance is added to previous branch metric
- We put a red X in front of the killed off branch
- New branch metric is in Red
- And finally we have the traceback path in orange

I. Turbo Coding

Turbo codes [11] are a class of high performance Forward Error Correction code and were the first to approach the Shannon limit for channel capacity.

Fig. 6. A Trellis Diagram

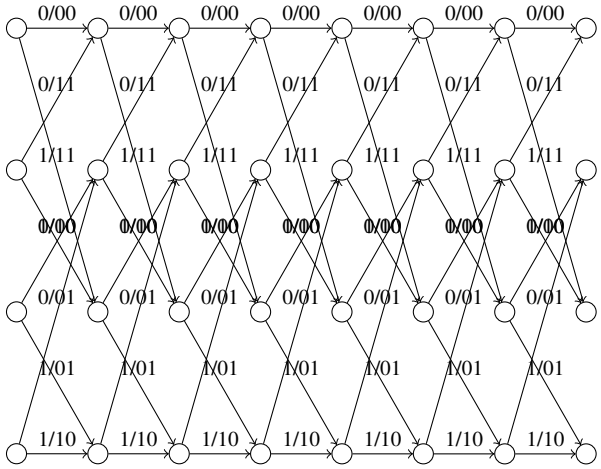
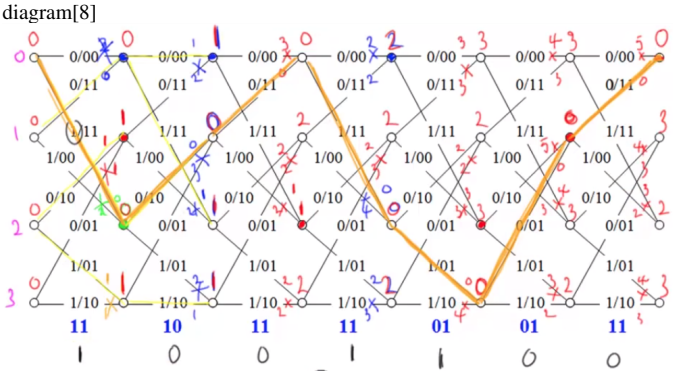


Fig. 7. Decoding a message using the Viterbi Algorithm and a trellis diagram[8]



1) Encoder: The turbo encoder is composed of two identical Convolutional encoders

2) Decoder: The decoder is quite similar to the encoder Uses two elementary decoders connected in series. Decoder 1 uses a soft decision approach. It does this by caluclating the Logirthm of Likelihood Ratio (LLR). This is represented by $\Lambda(d_k)$

$$\Lambda(d_k) = log\frac{p(d_k = 1)}{p(d_k = 0)}$$

The *a posteriori probability* (APP) of interpreting a received data bit d_k as i is $p(d_k = i)$, $i \epsilon 0, 1$

This probability is then fed into Decoder 2 which uses a hard descion approach and calculates a decoded bit taking the LLR into account

An interleaver is placed between the two decoders to scatter burst errors The viterbi algorithm cannot calculate the APP so is not used in Decoder 1 (we use an algorithm like BCJR instead), however it can be used to make the hard desicion in Decoder 2.

The big innovation in turbo coding is the use of probability data to interpret the intended bit at the receiver.

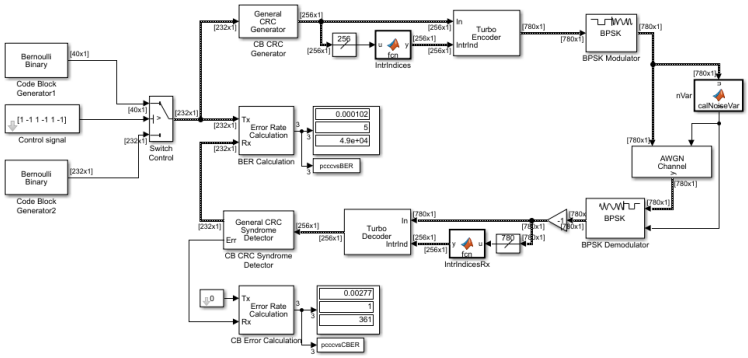
III. Convolutional Coding in Practice

The LTE description document [5] defines two types of channel coding to be used in LTE systems. We shall explore them and compare their performance using SIMULINK models to simulate...

A. Variable size Parallel Concantenated Convolutional (aka Turbo) Coding

1) : Our **Aim** is to explore the performance of a Turbo coding scheme in a full communications simulation.

Fig. 8. SIMULINK model of Variable Sized Turbo Coding in a Communications system - courtesy of Mathworks[3]



2) Setup: Since we are using variable size Turbo coding, we can test two different code lengths in one simulation (based on the control signal). The code block lengths were chosen to simulate a wide range of the possible lengths in the LTE description [5] so a small codeblock, a medium length block and the longest block(6144bits) were chosen. Different number of decoding iterations were performed for each codeblock length. Below is the list of the parameters used for testing:

- 2 iterations, 64/256 bits
- 4 iterations, 64/256 bits
- 6 iterations, 64/256 bits
- 2 iterations, 1024/2048 bits
- 4 iterations, 1024/2048 bits
- 6 iterations, 1024/2048 bits
- 1 iteration, 4096/6144 bits
- 2 iterations, 4096/6144 bits
- 4 iterations, 4096/6144 bits
- 6 iterations, 6144 bits

Other parameters

- Constraint length of 4
- Code generator was a [13 15] (octal) code generator
- The simlation uses BPSK modulation over an AWGN channel
- CRC generator used is 24 bit CRC for FEC
- Variable code block length that toggles between two code lengths based on a control signal
- 361 code blocks are sent in each simulation epoch
- Turbo decoder that employs the a posteriori probability and uses i number of iterations

The Bit Error Rate gives us the ratio of errors to total bits while The Code block error rate give us the ratio of bad code blocks to total (361) code blocks in each simulation run. The Code block error rate (CBER) is more important here because the nature of Turbo codes means that we want to miniimize the amount of bad codeblocks at the receiver not neccesarily the individual bits (as a succesful block means any bit errors can be corrected)

3) Results - Tables : See Tables I&II:

Fig. 9. Turbo Code results - BER vs SNR

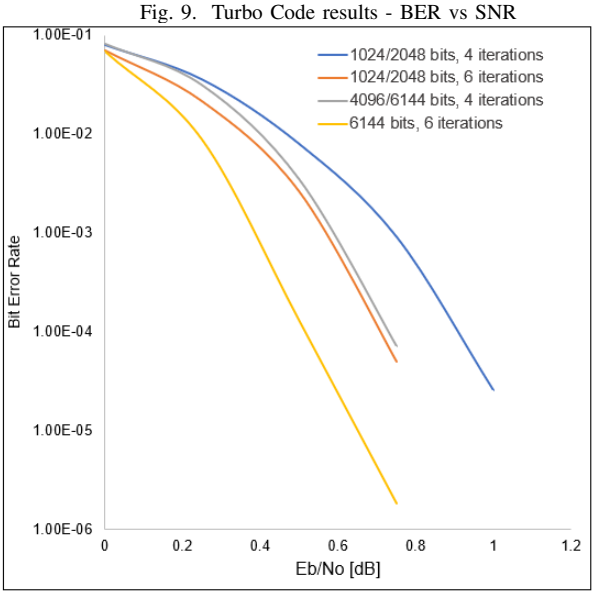
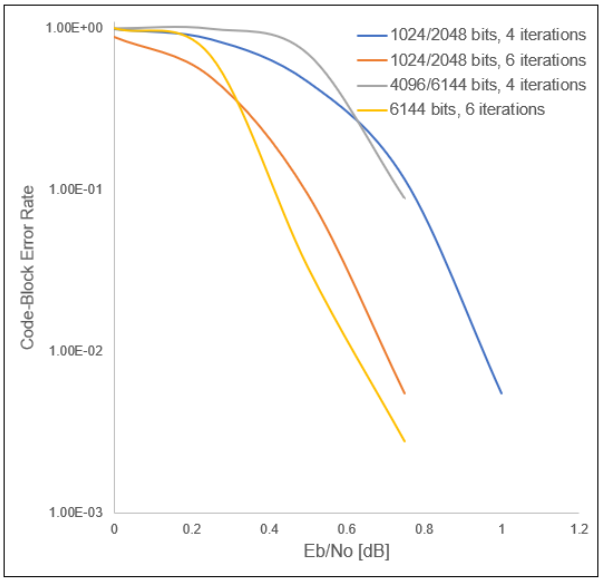


Fig. 10. Turbo Code results - CBER vs SNR



4) Results - Graphs: See Figs. 9&10:

5) Discussion of Results: From the results, there are two main things to be observed: *(1)* Larger Codeblocks outperform the Smaller blocks by a good margin *(2)* More Decoder iterations result in lower BER and subsequently lower CBER. One thing to be noted however is that decoder iterations come at the cost of increased overhead

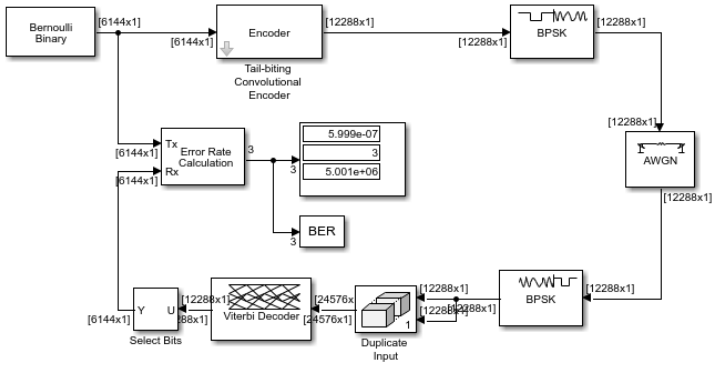
The best performing code tested was the 6144bit code with 6 decoder iterations, followed by the 1024/2048-bit code with 6 iterations.

Only the top 4 performing codes are plotted on the graph for the sake of clarity but all the results can be seen in the table.

B. Tail-Biting Convolutional Coding

1) : In Tailbiting codes, the encoder starts and ends at the same state but this state remains unknown to the decoder [7]. Our **Aim** is to explore the performance of Tail-Biting convolutional Coding scheme in a communications network

Fig. 11. SIMULINK model of Tailbiting Convolutional Code in a Communications system - courtesy of Mathworks[4]



2) Setup:

- Constraint length of 7
- Two different code generators were used: A half-rate [133 171] and a $\frac{1}{3}$ rate generator [133 171 165]
- A tailbiting convolutional encoder was used with a viterbi decoder at the receiver
- The simulation was carried out using BPSK modulation over an AWGN channel

3) Results - Table: See Table III:

4) Results - Graphs: See Fig. 12:

5) Observations & Discussion of Results: From the results, we can see that the code rate is far more important than the codeblock length for achieving better results (all 1/3 codes outperform 1/2 codes). However, we must be reminded that although the 1/3 code outperforms the 1/2 code, the 1/2 code is 50% faster than the 1/3 code so this must be taken into consideration when designing channel coding schemes. The LTE standard is 1/3 code and 6144 bits codeword length.

IV. Conclusion

We set out to understand channel coding starting with the first principles and concepts, exploring the intuition behind the channel coding concepts and finally building up to Turbo codes which are used in modern communications. We were

TABLE I
TURBO CODE RESULTS (BER)

Eb/No [dB]	64/256 bits			1024/2048 bits			4096/6144 bits			6144 bits
	i=2	*i=4*	*i=6*	*i=2*	*i=4*	*i=6*	*i=1*	*i=2*	*i=4*	*i=6*
0	0.101898	0.66482	0.077796	0.105637	0.080043	0.069867	0.137585	0.106017	0.080526	0.067481
0.25	0.075347	0.053082			0.035753	0.021307			0.030749	0.00861
0.5	0.051878	0.029653	0.025102	0.047952	0.007971	0.002587	0.118824		0.003427	0.000127
0.75	0.032	0.013245		0.024072	0.000922	4.95E-05			7.12E-05	1.81E-06
1	0.017592	0.006898	0.006102	0.009444	2.57E-05		0.061345	0.008383		
1.25	0.008714			0.003004				0.002134		
1.5	0.004102	0.001776	0.001592	0.000679				0.000397		
1.75	0.001796			0.00013				4.95E-05		
2	0.000735	0.000122	0.000102	2.93E-05			0.011663	5.44E-06		
2.25	0.000245	8.16E-05		3.67E-06				1.09E-06		
2.5	4.08E-05									
4							1.36E-05			
4.5							4.89E-06			

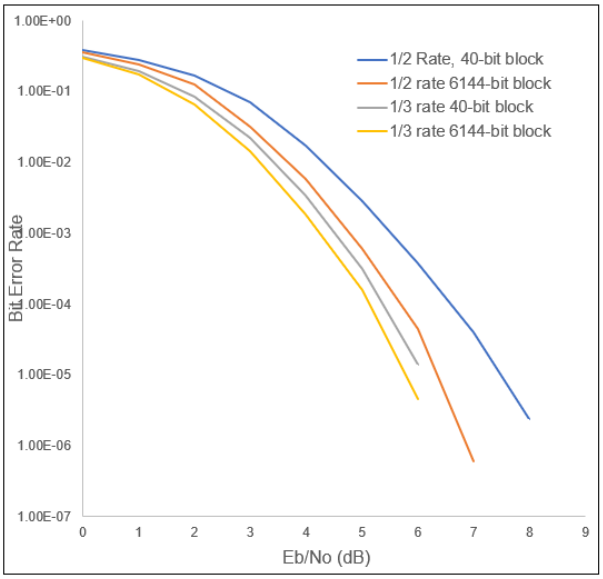
TABLE II
TURBO CODE - CODE-BLOCK ERROR RATES

Eb/No [dB]	64/256 bits			1024/2048 bits			4096/6144 bits			6144 bits
	i=2	*i=4*	*i=6*	*i=2*	*i=4*	*i=6*	*i=1*	*i=2*	*i=4*	*i=6*
0	0.867036	0.66482	0.598338	1	0.98338	0.883657	1	1	1	1
0.25	0.792244	0.526316			0.850416	0.498615			0.99446	0.67313
0.5	0.695291	0.379501	0.304709	1	0.465374	0.094183	1		0.698061	0.033241
0.75	0.545706	0.216066		0.972299	0.116343	0.00554		1	0.088643	0.00277
1	0.382271	0.121884	0.102493	0.850416	0.00554		1	0.99723		
1.25	0.235457			0.540166				0.911357		
1.5	0.121884	0.036011	0.033241	0.221607				0.465374		
1.75	0.049861			0.052632				0.099723		
2	0.030471	0.00554	0.00277	0.01662			1	0.01108		
2.25	0.00554	0.00277		0.00277			1	0.00277		
2.5	0.00277						1			
4							0.027701			
4.5							0.01108			

TABLE III
TAILBITING CODE SIMULATION RESULTS

Eb/No [dB]	1/2 Rate, 40-bit	1/2 rate 6144-bit	1/3 rate 40-bit	1/3 rate 6144-bit
0	3.89E-01	0.365723	3.11E-01	0.300293
1	0.281536	0.246322	1.96E-01	0.172472
2	0.169121	0.124439	0.086237	6.50E-02
3	0.071222	0.032267	0.022007	0.014606
4	0.017428	0.005929	0.003449	0.001842
5	0.002832	0.00059	0.000319	0.00016
6	0.000382	4.46E-05	1.38E-05	4.60E-06
7	4.00E-05	6.00E-07		
8	2.40E-06			

Fig. 12. Tailbiting results - BER vs SNR



able to achieve that knowledge and test out the principles using SIMULINK models, this helped clarify why certain standards are used (like 6144 bit codeblocks for all LTE communications and 1/3-coderates for tailbiting codes).

ACKNOWLEDGMENT

I would like to thank Dr. Veselin Rakocevic for his support and guidance during the course of this brief research project.

.
.
.
.
.

TABLE IV
LEMPEL-ZIV IMPLEMENTATION

current letter	current codeword	next letter	next codeword	lookup	In Dict(T/F)	Add to Dict	Dict Index	Output
a	97	b	98	ab	F	ab	256	97
b	98	c	99	bc	F	bc	257	98
c	99	a	97	ca	F	ca	258	99
a	97	b	98	ab	T	-	-	-
b	98	c	99	abc	F	abc	259	256
c	99	a	97	ca	T	-	-	-
a	97	b	98	cab	F	cab	260	258
b	98	c	99	bc	T	-	-	-
c	99	a	97	bca	F	bca	261	257
a	97	b	98	ab	T	-	-	-
b	98	c	99	abc	T	-	-	-
c	99	a	97	abca	F	abca	262	259
a	97	b	98	ab	T	-	-	-
b	98	c	99	abc	T	-	-	-
c	99	a	97	abca	T	-	-	-
a	97	b	98	abcab	F	abcab	263	262
b	98	c	99	bc	T	-	-	-
c	99	a	97	bca	T	-	-	-
a	97	b	98	bcab	F	bcab	264	261
b	98	c	99	bc	T	-	-	-
c	99	a	97	bca	T	-	-	-
a	97	b	98	bcab	T	-	-	-
b	98	c	99	bcabc	F	bcabc	265	264
c	99	a	97	ca	T	-	-	-
a	97	b	98	cab	T	-	-	-
b	98	c	99	cabc	F	cabc	266	260
c	99	a	97	ca	T	-	-	-
a	97	b	98	cab	T	-	-	-
b	98	c	99	cabc	T	-	-	-
c	99	a	97	cabca	F	cabca	267	266
a	97	b	98	ab	T	-	-	-
b	98	c	99	abc	T	-	-	-
c	99	a	97	abca	T	-	-	-
a	97	b	98	abcab	T	-	-	-
b	98	c	99	abcabc	F	abcabc	268	263
c	99	-	-	c	T	-	-	96

REFERENCES

[1] Shannon, C. (1948). A Mathematical Theory of Communication. Bell System Technical Journal, 27(4), pp.623-656.

[2] Hartley, R. (1928). Transmission of Information1. Bell System Technical Journal, 7(3), pp.535-563.

[3] Uk.mathworks.com. (2018). Parallel Concatenated Convolutional Coding: Turbo Codes- MATLAB & Simulink- MathWorks United Kingdom. [online] Available at: https://uk.mathworks.com/help/comm/examples/parallel-concatenated-convolutional-coding-turbo-codes.html [Accessed 25 Apr. 2018].

[4] Uk.mathworks.com. (2018). Tail-Biting Convolutional Coding- MATLAB & Simulink- MathWorks United Kingdom. [online] Available at: https://uk.mathworks.com/help/comm/examples/tail-biting-convolutional-coding.html [Accessed 25 Apr. 2018].

[5] 3GPP TS 36.212 v10.8.0, "3rd Generation partnership project; Technical specification group radio access network; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (Release 10)", 2013-06.

[6] Huffman, D. (1952). A Method for the Construction of Minimum-Redundancy Codes. Proceedings of the IRE, 40(9), pp.1098-1101.

[7] Ancha, D. (2008). Implementation of Tail-Biting Convolutional Codes. Master of Science in Electrical Engineering. West Virginia University.

[8] Janak Sodha (2018). Viterbi Algorithm. [online] Available at: https://www.youtube.com/watch?v=iPh-HKZyWN4 [Accessed 25 Apr. 2018].

[9] Ziv, J. and Lempel, A. (1978). Compression of individual sequences via variable-rate coding. IEEE Transactions on Information Theory, 24(5), pp.530-536.

[10] (2018). Introduction to EECS II: Digital Communication Systems. [online] MIT OpenCourseWare. Available at: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-02-introduction-to-eecs-ii-digital-communication-systems-fall-2012/index.htm [Accessed 25 Apr. 2018].

[11] Berrou, C. and Glavieux, A. (1996). Near optimum error correcting coding and decoding: turbo-codes. IEEE Transactions on Communications, 44(10), pp.1261-1271.

[12] Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Transactions on Information Theory, 13(2), pp.260-269.