Forward Error Correction in Underwater Acoustic Sensor Networks

Margarita Otochkina, Undergraduate Student, School of Computer Science, Carleton University

I. Introduction

Underwater Acoustic Sensor Networks (UASNs) suffer from long delays, variable reliability, and narrow bandwidth. The communication in such an environment is assumed to be of low data rate and short range. The project considers Forward Error Correction (FEC) as a method to improve communication in UASNs. Using different Error Correcting Codes (ECC), the goal is to evaluate the performance of FEC in simulated Arctic-like conditions using a GNU Radio and MATLAB model developed by Traboulsi.[1]

The project's background is described in Section II. It includes the project's environment (underwater communication, GNU Radio), and the description of the simulation. Lastly, the section describes ECCs and how they work.

The test design is described in Section III, including the modified version of an originally provided flow diagram for the simulation, and two FEC diagrams.

The evaluation technique is defined in the Section IV. The section provides the criteria for the evaluation, details of the simulation adapted for the evaluation, and performance results. The results are compared to the base case.

We conclude in Section V. The section also includes the discussion of the future work.

II. Background

This section describes the background information required for the project. It defines the underwater communication and how water affects the quality of UASNs, explains what GNU Radio is, how the simulation of an Arctic-like environment works, and gives a details background of ECCs considered in the project.

A. Underwater Communication

Underwater Acoustic Sensor Networks (UASNs) is a promising technology for numerous applications including monitoring of the undersea environment for pollution reduction.

Underwater communications are based on acousic waves, which are generated by mechanical vibrations at a frequency f (in Hertz). Because of the elasticity of water, acousic waves create acoustic pressure, detectable by a sensor called *hydrophone*. Acoustic waves travel at much shorter speeeds, comparing to electromagnetic waves, which makes their wavelength relatively much shorter.

The factors affecting UASNs are absorption, geometrical spreading, multipath propagation, and noise. Water absorbs acoustic waves, and the degree of absorption depends on the chemical content of the water. Geometrical spreading is a result of the distance from the source increasing as sound waves travel away. Multipath signals are created as a result of changes of propagation direction caused by varying sound speed. Lastly, site-specific noises and ambient noises affect underwaver communication. Site-specific noises depend georgraphically, and include noises made by breaking ice and sea creatures. Sources of ambient noise are turbulence, shipping, waves, and thermal. [2]

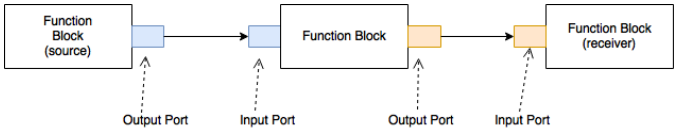

Fig. 1. GNU Radio flow diagram.

As a result, massages sent in UASNs suffer from high bit error rate (BER) and are assumed to be of low data rate, and small size. The goal of the project is to evaluated the performance of FEC in such an environment.

B. GNU Radio

The implementation and testing of FEC is mostly done in the GNU Radio environment. GNU Radio is an open-source software for designing, simulating, and deploying real-world radio systems. GNU Radio framework provides a comprehensive library of processing blocks. The blocks are combined together in a flow graph via ports for creating complex signal processing applications. The applications are generated as a Python file from grc flow graph. [3]

As mentioned earlier, GNU Radio Companion flow diagrams are built using blocks, where each block represents a function. Each block have parameters and ports. Ports are used for connecting blocks, and represented as colored rectangles on a function block. The color of a port depends from the data structure the port uses:

1) Pink: Byte
2) Orange: Float
3) Grey: PDU Message
4) Blue: Complex

Usually, blocks have two ports: one port outputs the data, the other port takes the data. Each block is connected by arrors pointing from the output function to the input function.

Fig. 1 shows a design of a flow diagrams in GNU Radio. Each block represents a function, ports of which are colored according to their data types. For connecting two blocks, the

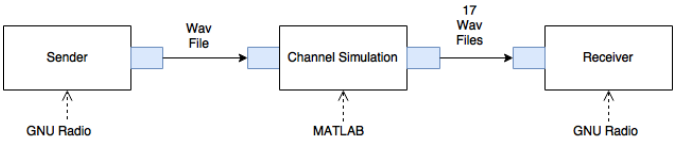

Fig. 2. Simulation architecture.

output port of the source block must be of the same color as the input port of the sink block. The blocks are connected to each other by arrors. The diagram closes on the receiver with only the input port.

GNU Radio contains FEC API. The library contains three types of FEC encoders/decoders (regular, tagged, and asynchronous), and six types of ECC: Dummy (or no ECC), LDPC, CC, POLAR, Repetition, and TPC. All ECCs are compared using the BER test provided as an FEC example. Using this test, CC and POLAR codes are chosen for the test as the codes achieving better BER. The codes' backgrounds are provided later in this section.

C. Simulation

The test is based on the simulation of an Arctic-like environment developed by by A.Traboulsi.[1] The simulation is separated into three steps which are presented on Fig. 2:

1) First, the sender in GNU Radio reads a message , modulates the data using FSK modulator, and produces wav file;
2) Then, MATLAB generates 17 wav files using the GNU Radio wav file, environment files provided with the simulation, and BELLHOP software;
3) Lastly, the receiver in GNU Radio takes wav files MATLAB, demodulates them, and outputs the result.

Figure 2 illustrates the design of the simulation, where sender and receiver are used in the GNU Radio environment, and channel simulation is done in MATLAB.

The simulation focuses on two categories of environments: with different ice cover percentages (0%, 20%, 40%, and 80%) and distance (5 km and 8 km) between sender and receiver. The simulation includes 100 different environments of each type.

For each wav file generated by GNU Radio sender flow graph, MATLAB generates 17 wav files for each environment. Each wav file has different SNR ranging from -50 to 30 dB.

The Section III contains the description of GNU Radio sender and receiver flow graphs, as well as the description of other modification made to the original simulation.

D. GFSK Modulator and Demodulator

Frequency-Shift Keying (FSK) Modulation is a modulation scheme for transmitting digital signals. FSK modulation widely used in underwater communications due to its robustness. The scheme works by transmitting digital data using different frequencies, where each frequency represents different binary pattern.

GNU Radio provides FSK modulation scheme as GFSK Mod and GFSK Demod functions. GFSK mod has three parameters:

samples per symbol stands for samples per baud;

Sensitivity is defined by a formula:

$$S = \pi \cdot \textit{Modulation\_Index} \ / \ \textit{Samples\_Per\_Symbol}$$

where modulation index is a value between 0 and 1.

BT is a bandwidth by symbol duration.

GFSK Demod has six parameters:

1) Samples per symbol represent samples per baud;
2) Gain mu controls rate of mu adjustment;
3) Mu is a fractional delay between 0 and 1;
4) Omega relative limit sets maximum variation in omega;
5) Freq error sets a bit rate error as a fraction.

E. Packet Encoder and Decoder

GNU Radio provides packet encoder and decoder blocks as part of "Packet Operation" library. Packet encoder packetizes a message by wrapping it into a packet of a size equall to the payload length adding a header, access code, and preamble. Size of a header is twice the payload length.

Packet encoder consist of six variables:

1) Samples per symbol is set to 8 by default, minimum value is 2;
2) Bits per symbol is 1;
3) Preamble and access code can be left empty for default values;
4) Pad for USRP is set to no;
5) Payload length is the size of a packet;

Packet decoder takes unpacked data and looks for the access code. Once decoder finds the access code, it reads the header, finds the payload length, and extracts the payload. Packet decoder block takes two parameters; Access code should be the same as in encoder, or left empty for the default value. Threshold should be left as -1 for the default value.

Packet encoder and decoder blocks are used with GMSL, PSK, QAM modulator and demodulator blocks. Additionally, packet encoder performs data whitening by using whitening transformation. Whitening is done by transforming a vector of random variables with a known covariance matrix into a set of new variables whose covariance is the identity matrix meaning that they are uncorrelated and all have variance.

In this project, the test uses a version modified by M. Barbeau. [4] The new version detects length tag, eliminating the need for the stream to tagged stream converter before the packet encoder and decoder. CRC in the new version is disabled.

F. Convolutional Code

CC is a widely used class of ECCs. It generates parity bits using a sliding window. A sliding window technique combines subsets of bits in the window using modulo 2 additions, or exclusive-or operation. The windows size is determined by the code's constraint length k. [5]

The number of parity bits per window and window sliding depends on CC rate. Given a rate m/n, CC produces n parity bits, sliding the window forward by m bits at a time.

GNU Radio provides CC encoder and decoder definitions. Typically, CC encoder uses $k = 7$, $rate = 1/2$, and $polynomials = [109, 79]$. The polynomials are constructed as:

$$109 : b(1101101) -> 1 + x + x^3 + x^4 + x^6$$
$$79 : b(1001111) -> 1 + x^3 + x^4 + x^5 + x^6$$

Additionally, blocks include four modes that specify encoder's behavior:

CC STREAMING: mode expects an uninterrupted flow of samples into the encoder, and the output stream is continually encoded. This mode is the only mode for this decoder that has a history requirement because it requires $rate \cdot (K - 1)$ bits more to finish the decoding properly. This mode does not work with any deployments that do not allow history.
CC TERMINATED: is a mode designed for packet-based systems. This mode adds $rate \cdot (k - 1)$ bits to the output as a way to help flush the decoder.
CC TAILBITING: is another packet-based method. Instead of adding bits onto the end of the packet, this mode will continue the code between the payloads of packets by pre-initializing the state of the new packet based on the state of the last packet for $(k - 1)$ bits.
CC TRUNCATED: a truncated code always resets the registers to the start_state between frames.

In GNU Radio, CC achieves one of the best performances along the other ECCs. Performance of CC is discussed in the Section IV.

G. POLAR Code

1) Definition: Polar code is a linear block ECC, first mentioned by Arikan [6]. Arikan defines it as code based on the idea of channel polarization. Code sequences with channel polarization achieve the symmetric capacity $I(W)$ of any given binary-input discrete memoryless channel (B-DMC) W.

Channel polarization operation consists of two phases: channel combining and channel splitting. The first phase combines copies of a given B-DMC W in a recursive manner producing a vector channel:

$$W_N : X^N -> Y^N$$

where N can be any power of two, $N = 2^n, n >= 0$.

The second phase splits the vector channel W_N back into a set of N binary-input coordinate channels:

$$W_N^{(i)} : X -> Y^N \cdot X^{i-1}, 1 < i < N.$$

The idea of polar coding is to use channel polarization for creating a coding system that access each coordinate channel $(W_N)^{(i)}$ individually and send data only through channels with $Z(W_N^{(i)})$ is near 0. $Z(W_N^{(i)})$ is a rate of polarization where:

$$Z(W_N^{(i)}) = \sum_{y_1^N \in Y^N} \sum_{u_1^{i-1} \in X^{i-1}} \sqrt{W_N^{(i)}\left(y_1^N, u_1^{i-1}|0\right) W_N^{(i)}\left(y_1^N, u_1^{i-1}|1\right)}$$

and (y_1^N, u_1^{i-1}) denotes the output of $W_N^{(i)}$.

The next part of this subsection describes the implementation of POLAR in GNU Radio.

2) Implementation in GNU Radio:

Polar code for GNU Radio is implemented by J. Demel.[7] He developed Polar encoder as two blocks: POLAR Encoder Definition and POLAR code Configuration.

POLAR Encoder Definition is build using six variables: *Packed Bits, Parallelism, Block Size (N), # Info Bits (K), Frozen Bit Values*. *Packed bits* option chooses between two different encoders. Packed-bit mode uses a generic bit shift encoder which operates on packed bytes. *Block Size* is a size of a polar code and must be of a power of two. *# Info Bits* is the number of bits in a packet with value less than Block size.

Frozen bit positions and values are calculated POLAR code configurator. By default, the encoder uses an all-zero frozen bit word. Code configurator calculates values for a specific block size and channel mode with its defining parameters. The defining parameter is a design-SNR that is as a result converted to the parameter for the channel model. Configurator expects *# Info Bits (K)* parameter. Frozen bit positions and values are returned as the k most reliable position after the synthetic channel parameters calculation.

Polar decoder is defined by two blocks as well: POLAR Decoder SC Definition, and POLAR code Configurator. Parameters for decoder are exactly the same as for encoder.

POLAR code is another powerfull ECC provided by GNU Radio. It's perfomance will be evaluated in the Section IV.

III. Implementation and Testing

This section provides a detailed description of the simulation adapted for the goal of the project. The details include GNU Radio sender and receiver diagrams, test requirements, and the test process itself.

A. Test Definition

The test is based on a simulation of an Arctic-like environment developed by Traboulsi.[1] The original simulation tests five environment types, each differs by ice cover percentage and distance.

Traboulsi's original design is a bit-stream oriented, while FEC requires a packet communication model. That's why the orignal design of GNU Radio flow diagrams is augmented with a packet encoder and decoder. The original updated diagrams are considered as a base-line case for the test. The results from FEC tests are compared to the results of the base case.

The test is broken into three cases. The first case mesures and evaluates the performance of a base case. The second case evaluates Extended Tagged FEC with CC as an ECC. The third case evaluates POLAR code used in Asynchronous FEC.

Each test case with the GNU Radio sender diagram that reads a message from a data file and returns a wav file. Using environment files from the simulation and GNU Radio wav file, MATLAB generates 17 wav files for every environment. Finally, the GNU Radio receiver reads MATLAB wav files one by one and returns the decoded message.

This section explains the design implemented in GNU Radio environment. This includes the receiver diagram, and the sender diagram for each test case.

B. Implementation

The implementation in the GNU Radio environment includes two diagrams: the FSK sender and receiver. This subsection first gives the detailed review of the sender diagrams, then of the receiver diagrams. In order to avoid repetition, some parts are omitted. The subsection starts with the base case diagram, moving to the CC diagram, and finishing with the POLAR diagram.

1) Base Case Diagram

Base case diagram is built of the original FSK diagrams without FEC added. First, the section explains the design of a sender diagram, then of a receiver diagram.

FSK Sender:

The graph starts with the file source function reading a message from a source file.[Fig. 3] The file source reads the message once from a location specified as “File:” argument. As a result, the function returns a byte stream.

Then, the packet encoder takes the byte stream and packetize it. The payload length is specified prior the processing, and should be exactly of a size of a message. So if the file size is 2 bytes, payload for packet encoder should be 2 as well.

GFSK Mod takes the packet, modulates it, and outputs a stream of complex values. GFSK behaviour is specified by arguments where:

1) Samples per symbol variable = 8;
2) Sensitivity = 1;
3) BT = 0.35.

The second part of the sender [Fig. 4] starts with a rational resample that interpolates samples by 1000. The function takes and outputs complex values. Frequency Xlating Finite Impulse Response (FIR) filer takes interpolated samples and performs frequency translation with a sample rate 48,000 samples/seconds. Taps value is speficied as a Band-pass filter:

$$firdes.band\_pass(0.5, samp\_rate, carrier - sideband, carrier + sideband, transistion)$$

where:

1) samp_rate = 48,000;
2) carrier = 8,000;

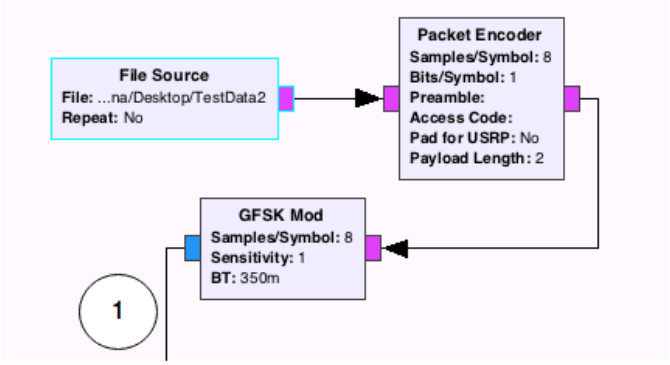

Fig. 3. FSK Sender: first part.

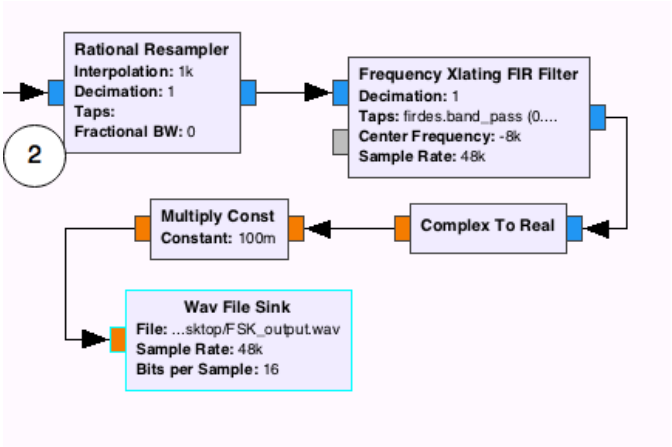

Fig. 4. FSK Sender: second part.

3) transition = 50;
4) sideband = 800;
5) center_frequency = -carrier = -8,000;

Lastly, the result is converted from complex to float value, and written to a wav file using wav file sink function. The result wav file is then used in MATLAB.

FSK Receiver:

Once MATLAB generated all wav file, the test returns back to the GNU Radio environment. The receiver flow diagram starts with the wav file source function that reads given wav file.[Fig. 5] For the test, the file is repeated until the program terminates. Wav file source returns wav file as float values, which the diagram converts to complex.

The complex values are processed through two filters: low-pass filter and band-pass filter,where Band-pass filer is:

$$firdes.band\_pass(1, samp\_rate, carrier - sideband, carrier + sideband, transistion)$$

Low-pass filter is:

$$firdes.low\_pass(1, samp\_rate, sideband, transistion)$$

and:

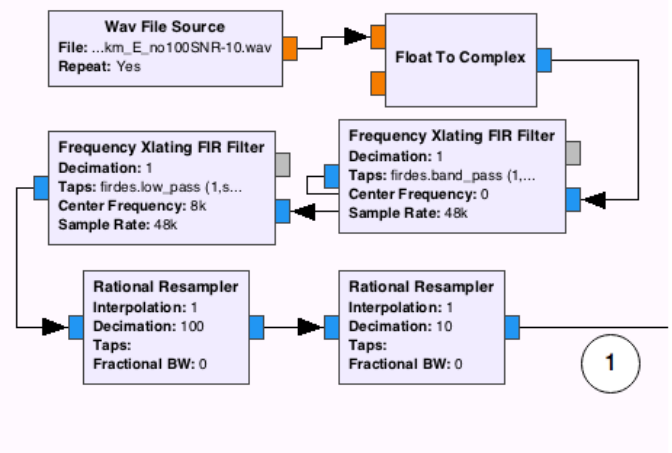

Fig. 5. FSK Receiver: first part.

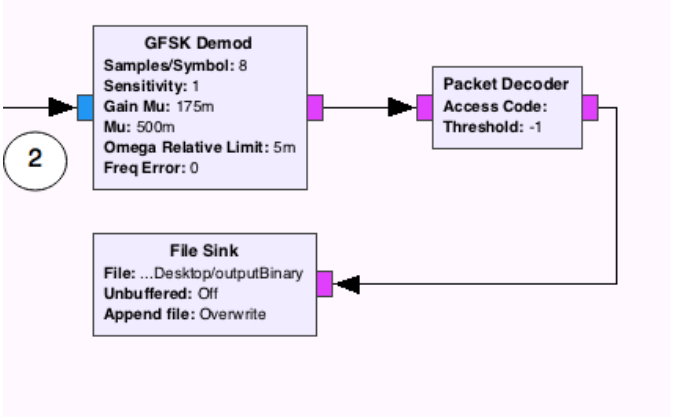

Fig. 6. FSK Receiver: second part.

1) samp_rate = 48,000;
2) carrier = 8,000;
3) transition = 50;
4) sideband = 800;
5) center_frequency = -carrier = -8,000;

The result is decimated using two rational resample blocks. First block decimates the stream by 100, second by 10, which in total gives 1000 - the interpolation rate used in the sender.

Then, GFSK demodulator takes the complex stream.[Fig. 6] As parameters, GFSK uses:

1) sensitivity = 1;
2) gain mu = 0.175;
3) mu = 0.5;
4) omega relative limit = 0.005;

Demodulator then passes packed byte stream to the packet decoder, which reads the header and returns the message (payload). Often, if SNR is too low, the packet decoder will not return the message. The reason is if the header is corrupted, the decoder cannot read it, and cannot find the payload. This fact is mentioned later in the Section IV.

2)FEC CC Diagram

The sender and receiver diagrams with FEC do not differ much from the base case ones. The section only discusses the modified part of the diagrams with FEC added, and ECC

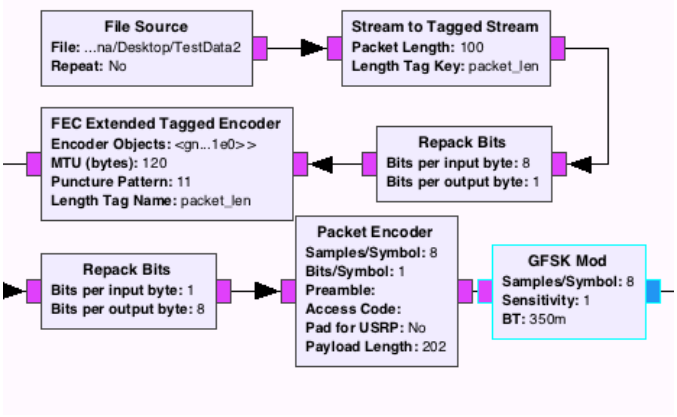

Fig. 7. FSK Sender with FEC CC.

descriptions. The section first describes the sender, then the receiver.

FSK Sender with FEC

The difference between the base-case diagram and FEC version is the part between the file source and the packet encoder. [Fig. 7]

When file source function reads the message, instead of passing it to the packet encoder, it send the message to the FEC encoder. The message has to be tagged first, where the tag is its length. The tag is added by the stream to tagged stream function. The message also has to be unpacked due to the fact that encoder accepts only unpacked data. Unpacking is done using the repack bits function. The values for unpacking are:

1) bits per input bytes = 0;
2) bits per output byte = 1.

FEC Encoder takes and returns an unpacked tagged byte stream. The FEC function has four parameters, where MTU bytes is the maximum size of data that the encoder accepts at a time, puncture pattern is 11 by default, and length tag name is the same variable as the one assigned to the data by the Stream to Tagged stream block. Encoder object is a name of a block that defines ECC. The encoding depends from a ECC specified.

The ECC used in this test is called CC.[Fig. 8] GNU Radio block containing the definition of CC is called CC Encoder Definition. The function's behaviour is specified by six arguments:

1) Parallelism = 0.
2) Frame bits = MTU · 8;
3) Constraint length (K) = 7;
4) Rate inverse = 2;
5) Polynomials = [109, 79];
6) Streaming behavior = terminating.

The unpacked byte stream from the encoder is packed back using repack bits function with settings opposite to packing. The resulted data is passed to the packet encoder. Since FEC increases the size of a message, the size of a message has to

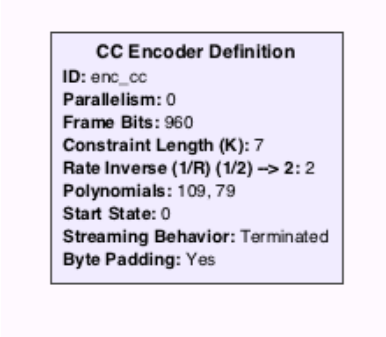

Fig. 8. CC encoder definition.

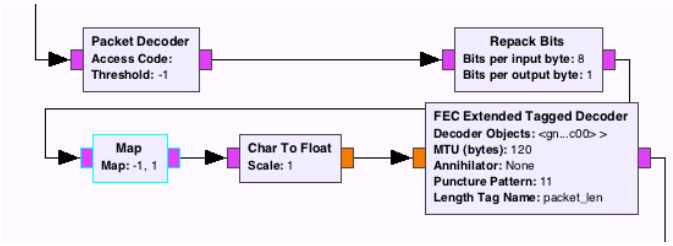

Fig. 9. CC FSK Receiver.

be checked and specified as a payload length in the packet encoder.

The rest of the diagram is the same as in the base case.

FSK Receiver with FEC

In FSK Receiver, FEC is added between packet encoder and file sink. [Fig. 9] The byte stream from the packed decoder is unpacked using repack bits function. The message then has to be converted from byte to float, because FEC decoder accepts only float values. FEC decoder reads the float stream and decodes it using ECC. All settings for the CC decoder, except for annihilatort, has to be the same as for the encoder. Annihilator is set to None by default.

CC decoder definition is exactly the same as the encoder one. The values for block's parameter has to be the same as in CC encoder definition.

As a result, FEC Decoder outputs the unpacked byte stream, which is then packed using the repack bits function. The resulted data is the message.

3) FEC POLAR Diagram

As well as CC diagrams, POLAR diagrams differ from the base case only by a part with FEC integrated. This subsection will only discuss the integrated part with ECC definitions, starting with the sender.

FSK Sender with FEC Polar

FEC Asynchronous encoder is added between file source and packet encoder. Asynchronous encoder both accepts and returns packed data, so no packing required. However, it takes only PDU messages. The byte stream first is tagged, and then

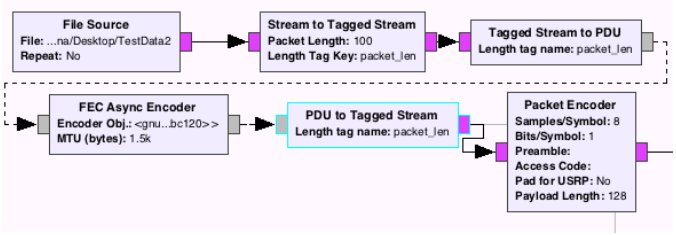

Fig. 10. POLAR FSK Sender.

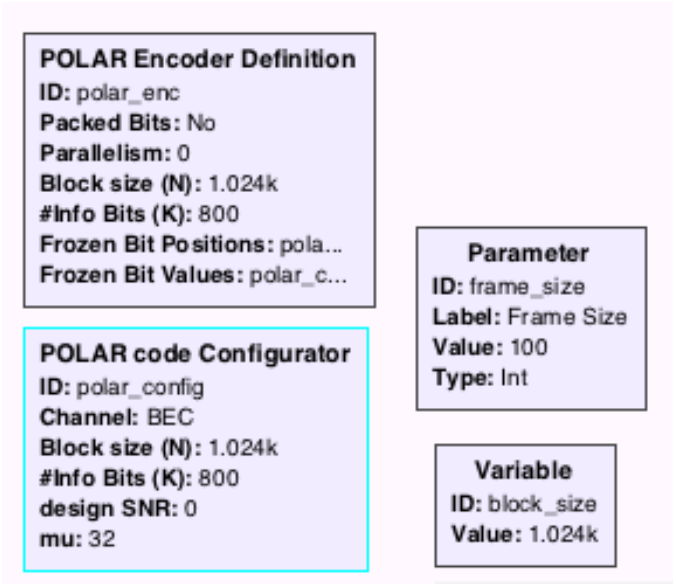

Fig. 11. POLAR encoder definition.

converted to PDU using tagged stream to PDU function.

The resulted PDU message is passed to the encoder. The encoder has two parameters: MTU and encoder object. [Fig. 10]

Encoder object is a ECC definition in a separate function block. For POLAR, the encoder definition consists of two blocks. Frame size should be of the same as a message. Block size must be a multiple of two, and less than # info bits. # Info Bits is a frame size multiplied by 8. [Fig. 11]

The encoded PDU message is converted back to the tagged byte stream and processed by the packed encoder.

FSK Receiver with FEC Polar

FEC block in the receiver is located between packet decoder and file sink. Before passing data to FEC decoder, the message is converted from a byte stream to float, and then from float to PDU. [Fig. 12]

POLAR definition consist of two blocks: POLAR Decoder SC Definition, and POLAR code Configurator. The settings

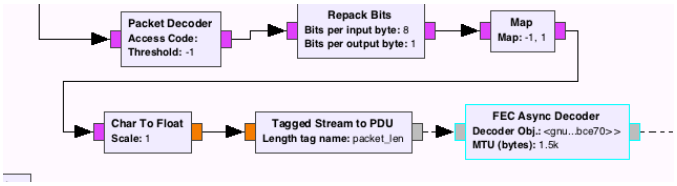

Fig. 12. POLAR FSK Receiver.

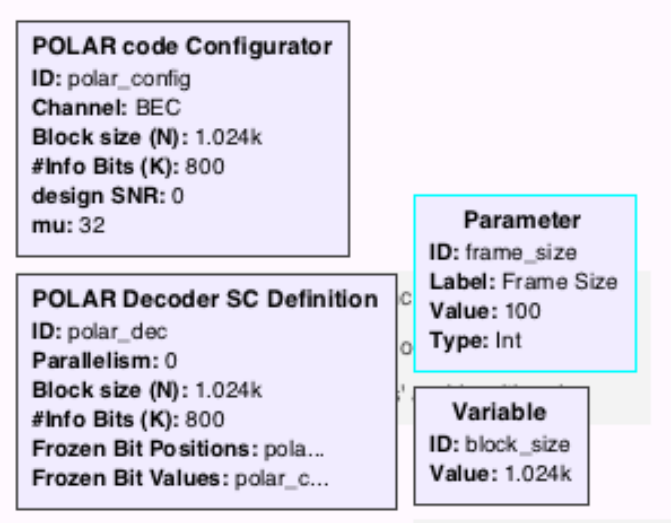

Fig. 13. POLAR decoder definition.

for POLAR decoder have to be the same as for the encoder. [Fig. 13]

C. Simulation Details

The simulation is structured in the same for every test case. First, the message stored in a file is read by the file source of the sender diagram. The sender processes the file and generates wav file used by MATLAB.

MATLAB requires two folders to generate files: the environment forlder and BELLHOP functions. The test simulates the environment with 5km_80pc_ice. The environment folder contains 100 environment files and pskProcessing.m MATLAB file. Wav file generated by GNU Radio has to be stored in the environment folder. pskProcessing function uses BELLHOP software, which is stored in "Matlab/ReadWrite" folder of a software library called "Acoustic Toolbox".

When all the requirements are satisfied, MATLAB calls pskProcessing. The function takes environment file number, number of seconds in a wav file, and the type of modulation (always 'GFSK'). pskProcessing has been modified for the test, so that instead of generating files for a range of environment files, it generates files for one specific environment file passed to the function.

Once wav files are generated, only the ones with SNR -10, -15, and -20 are used for the test. In the receiver, wav source block repeats the message. This way, the same message can be tested many time until a certain number of packets is generated, or a period of time passed.

Once the message is received and processed, the result is saved as a text file. The Python script called "counter" counts the number of packets received in total and the number of packets saisfying the test requirement. The details for the evaluation are specified in the next section.

IV. Evaluation

Performance of FEC in UASN is evaluated by two criterias:

1) Size of a message after encoding;
2) Packet Error Rate (PER) per Signal to Noise Ratio (SNR)

Every ECC increases the size of a message by some increase rate. For example, Repeptition code sends the message three times, increasing the total message size by three. Other ECCs also affect the message size by different degree. For UASN, the message size has to stay as small as possible. Thus, the first criteria is to find ECC with the smallest or most optimal increase rate.

The second criteria is PER per SNR. The test is done by sending a small message n times. In this case it makes more sense to measure PER instead of a bit error rate (BER). For PER, every packet that has at least one error is counted as an error. Lost packets are considered as erroneous as well. SNR is a measure that compares the level of a desired signal to the level of background noise. Comparing PER to SNR helps to determine how much the noise affects the message, and how much FEC improves it. The smaller PER the better the performance of FEC.

Performance of FEC will be evaluated as how much ECC increases the size of a message and decreases PER.

For the test, the simulation sends a message of size two bytes. The receiver processes the message from the sender 512 times for 40 minutes. The wav file source in the receiver repeats message every second, and processes every message for up to four seconds. Therefore, it takes approximatelly 35 minutes to process 512 messages. Five minutes to include the cases when the message is process for more than four seconds.

The message content is:

d0 22

The received message has to be exactly the same. Any difference is considered as an error.

SNR range used in the test is only -10, -15, and -20. Every sender and receiver has been tested with higher SNR, however most of the time anything higher than -10 have 0% packet rate. -20 dB is the lowest SNR packet decoder can process, and -10 dB is the highest SNR where the receiver can still produce errors.

The test was performed with five environment files: 1, 25, 50, 75, 100.

Each test is evaluated according to the mean and standard error (SE). The results are compared determining which test achieves the best performance.

The evaluation starts with comparing the increase rate for the message size. Then, the test results are evaluated.

A. Size

For the evaluation, let us define the increase rate as R_i. Clearly, the base case simulation does not suffer from the increasing message size. The size between the file source and packet encoder stays the same.

R_i using CC depends from the message itself. For small messages, $R_i = 3$. However, the bigger the message gets, the smaller R_i. For example, a message of size 100 bytes has:

$$R_i = 2.02$$

TABLE I
NO FEC: PER MEAN RESULTS.

SNR (dB)	-20	-15	-10
μ PER	99.2%	99.88%	61.8 %

a message of size 1000 bytes has:

$R_i = 2.002$

That makes CC more usefull in the cases with large messages.

POLAR ECC shows lower increase rate comparing to CC. For a small message of size two bytes:

$R_i = 2$

For bigger messages, the increase rate get smaller. Message of size 100 bytes has:

$R_i = 1.28$

Thus, comparing two ECCs, POLAR has lower increase rate comparing to CC, but higher comparing to no FEC at all.

B. PER per SNR

This subsection provides the mean and standard error for results achieved during each test case. The results of FEC cases are compared to the base case. Only mean results are used for the evaluation. Full results are available in Appendix A. The section starts with no FEC test results, and ends with POLAR.

1) Base Case Results

The test produces five samples for each SNR using different environment files. The result of the test highly depends from the environment file, and while for one environment the test can return 0% PER, for the other it can be 100%.

The evaluation starts with calculating the mean value μ for each SNR. Since the population tested is small, the mean is calculated using the formula:

$\mu = \sum_{i=0}^{N} X_i/N$

where N is the population size, and sum is the sum of all samples in the population.

Table I contains the mean results for the base case. The results are mostly consistent and show very high PER. In increasing SNR PER decreases. For the lowest $SNR = -20$ dB, PER is almost 100%. Most of the time the received messages are either decoded incorrectly, or the packed decoder cannot read the header. For $SNR = -15$, PER slightly increases. Mostly, the packet decoder cannot read the header, and does not return any messages. During $SNR = -10$, the test results show the expected improvement. The average PER drops to 62%, meaning in average 38 messages out of 100 are received successfully.

Table 1. No FEC PER mean results

Standard error (SE) is calculated with the fomula:

TABLE II
NO FEC: STANDARD ERROR.

SNR (dB)	-20	-15	-10
SE	0.39%	0.03%	9.39 %

TABLE III
CC: PER MEAN RESULTS.

SNR (dB)	-20	-15	-10
μ PER	86.2 %	48.1%	40.04%

$SE = \sigma/\sqrt{N}$

where σ is the standard deviation.

While SE for $SNR = -20$ and -15 is small, it gets larger for $SNR = -10$.[Table II] The reason is because the results differ from environment files, the difference between two environment files can be up to 100%. Due to the fact, the error gets larger.

Overall, the base case does not provide good results: PER does not go lower than 60%. The next subsection describes the results of the test performed with CC ECC.

2) FEC with CC ECC Results

Comparing to the base case, CC demonstrates immideate improvement. The results on the Table III show that PER at $SNR = -20$ is 13% lower than base case, PER at $SNR = -15$ is twice lower than for $SNR = -20$, and the result for $SNR = -10$ is just slightly lower than for $SNR = -15$.

SE however is way larger comparing to the base case., as shown on the Table IV. The reason is the same: different environment files produce different results. Yet, CC still corrects the errors better even in the worst cases comparing to the base case.

The next subsection analyses the results of POLAR ECC.

3) POLAR Results

POLAR demonstrates the best PER vs SNR performance comparing to all other tests.[Table V] Particularly, it has the lowest PER for any SNR, especially for $SNR = -10$.

SE is slightly larger comparing to other test due to the

TABLE IV
CC: STANDARD ERROR.

SNR (dB)	-20	-15	-10
SE	3.33%	6.31%	6.26 %

TABLE V
POLAR: PER MEAN RESULTS.

SNR (dB)	-20	-15	-10
μ PER	70.7 %	13.2%	40.0%

TABLE VI
POLAR: STANDARD ERROR.

SNR (dB)	-20	-15	-10
SE	7.3%	5.3%	9.7%

packet decoder performance.[Table VI] POLAR either shows perfect results, or no results at all if the packet decoder cannot read headers.

C. Summary

According to the results, POLAR achieves the best performance comparing to the base case and CC. PER using POLAR is way lower, yet the size of a message is affected not as much as using CC. The performance of CC is almost twice better than the performance of the base case design. Fig. 14 illustrates the results of each test comparing to each other. The x-axis prepresents the SNR, while the y-axis represents the PER. The lower the line the better as the goal of the project is to reduce PER leaving the transferred message as small as possible,

Therefore, POLAR is a good way to improve PER, as well as BER, in UASNs.

V. CONCLUSION

The goal of the project was to evaluate the performance of FEC for underwater communications. The evaluation was done using two criteria: message size and PER vs. SNR. Because underwater communications suffer from high BER, FEC was considered as a method of improving it. However, FEC increases the size of messages. The question was "Does this overhead translate into better PER?".

Three test cases were performed: the base case without FEC, FEC diagram with CC ECC, and FEC with POLAR ECC. The results demonstated that the POLAR code achieves the best performance. It increases the message size, however the overhead is smaller than for CC. The increase rate is at most two.

The future work includes the improvement of framing techniques, as the packet encoder and decoder have their imperfections. FEC implementations in GNU Radio can also be improved to add more features and increase the performance.

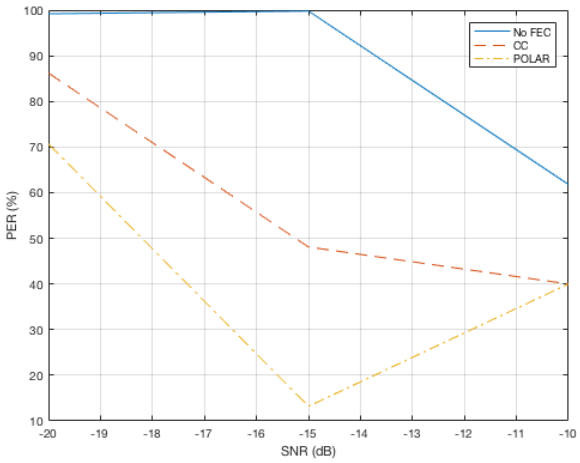

Fig. 14. PER vs SNR.

VI. APPENDIX A

A. Full Test results

No FEC:

Environment 1:
1) -20 db: 25 out of 512 received; PER = 95.1%
2) -15 db: 1 out of 512 received; PER = 99.8%
3) -10 db: 512 out of 512 received: PER = 0.0%

Environment 25:
1) -20 db: 0 out of 512 received; PER = 100.0%
2) -15 db: 0 out of 512 received; PER = 100.0%
3) -10 db: 0 out of 512 received: PER = 100.0%

Environment 50:
1) -20 db: 0 out of 512 received; PER = 100.0%
2) -15 db: 2 out of 512 received; PER = 99.6%
3) -10 db: 462 out of 512 received: PER = 9.8%

Environment 75:
1) -20 db: 0 out of 512 received; PER = 100.0%
2) -15 db: 0 out of 512 received; PER = 100.0%
3) -10 db: 4 out of 512 received: PER = 99.2%

Environment 100:
1) -20 db: 0 out of 512 received; PER = 100.0%
2) -15 db: 0 out of 512 received; PER = 100.0%
3) -10 db: 0 out of 512 received: PER = 100.0%

CC:

Environment 1:
1) -20 db: 0 out of 512 received; PER = 100.0%
2) -15 db: 512 out of 512 received; PER = 0.0%
3) -10 db: 511 out of 512 received: PER = 0.2%

Environment 25:
1) -20 db: 201 out of 512 received; PER = 60.7%
2) -15 db: 272 out of 512 received; PER = 46.9%
3) -10 db: 512 out of 512 received: PER = 0.0%

Environment 50:
1) -20 db: 136 out of 512 received; PER = 73.4%
2) -15 db: 272 out of 512 received; PER = 46.9%
3) -10 db: 512 out of 512 received: PER = 0.0%

Environment 75:

1) -20 db: 0 out of 512 received; PER = 100.0%
2) -15 db: 272 out of 512 received; PER = 46.9%
3) -10 db: 0 out of 512 received: PER = 100.0%

Environment 100:

1) -20 db: 0 out of 512 received; PER = 100.0%
2) -15 db: 6 out of 512 received; PER = 99.8%
3) -10 db: 0 out of 512 received: PER = 100.0%

Polar:

Environment 1:

1) -20 db: 512 out of 512 received; PER = 0.0%
2) -15 db: 512 out of 512 received; PER = 0.0%
3) -10 db: 512 out of 512 received: PER = 0.0%

Environment 25:

1) -20 db: 124 out of 512 received; PER = 75.8%
2) -15 db: 512 out of 512 received; PER = 0.0%
3) -10 db: 0 out of 512 received: PER = 100.0%

Environment 50:

1) -20 db: 112 out of 512 received; PER = 78.1%
2) -15 db: 512 out of 512 received; PER = 0.0%
3) -10 db: 511 out of 512 received: PER = 0.2%

Environment 75:

1) -20 db: 0 out of 512 received; PER = 100.0%
2) -15 db: 174 out of 512 received; PER = 66.0%
3) -10 db: 1 out of 512 received: PER = 99.8%

Environment 100:

1) -20 db: 1 out of 512 received; PER = 99.8%
2) -15 db: 512 out of 512 received; PER = 0.0%
3) -10 db: 512 out of 512 received: PER = 0.0%

REFERENCES

[1] Ahmad Traboulsi, *Software-Defined Underwater Acoustic Modems for Arctic-like Environment*. Carleton University, Ottawa, 2016 https://curve.carleton.ca/system/files/etd/f9016df7-cac9-46be-8576-9e08456ecabf/etd_pdf/fbcc8469ce1f4fe40c1c473a5cc57614/traboulsi-softwaredefinedunderwateracousticmodemsfor.pdf.

[2] Michel Barbeau, *Wireless Mobile Communications, Networks and Security* : Manuscript. Carleton University, Ottawa, 2016.

[3] *GNU Radio*. gnuradio.org, 2016.

[4] Michel Barbeau, Margarita Otochkina, and Ahmad Traboulsi, *Frequency-Shift Keying (FSK) Transceiver, Packet/Encoder Decoder and Forwad Error Correction (FEC).*. Carleton University, 2016 https://github.com/michelbarbeau/gr-splash.

[5] *Convolutional Coding, Lecture 8*, 6.02 Draft MIT , Fall 2010 http://web.mit.edu/6.02/www/f2010/handouts/lectures/L8.pdf

[6] Erdal Arkan, *Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels*. in IEEE Transactions on Information Theory, vol. 55, no. 7, pp. 3051-3073, July 2009.

[7] Johannes Demel, *Polar Codes for Software Radio*. Karlsruhe Institute of Technology, June 2015 https://github.com/jdemel/socis-2015-gr-results/blob/master/proposal.pdf.