MANCHESTER 1824

The University of Manchester

# School of Computer Science
# COMP28512: Mobile Systems
# Semester 2: 2018-19
# Laboratory Task 3
# Bit-error Control

## 1. Introduction

Noise is a fundamental problem with radio transmission since it causes bit-errors when digital information is received. Transmitting at higher power is an obvious way of making sure a signal is received with fewer bit-errors, since the noise power may then be made negligible in comparison to the signal power. But high power signals carry further and cause interference over a wider range, thus making the re-use of frequency bands some distance away more difficult. They also more quickly discharge the battery in a mobile device, and maybe cause unnecessary radiation. It is advantageous, therefore, to find ways of reducing the power of mobile phone transmissions and dealing with the bit-errors that will then be caused by the noise. This highlights the need for bit-error control.

The effect of bit-errors on text, within emails for example, could be catastrophic if they are not detected and eliminated by various mechanisms. These mechanisms include the use of:

- parity checks, check-sums and cyclic redundancy checks (CRC) which are extra bits inserted by the transmitter to allow the receiver to detect the presence of bit-errors when they occur.
- forward error correction (FEC) which requires the transmitter to insert even more extra bits into the message to allow bit-errors to be corrected at the receiver,
- automatic re-transmission reQuest (ARQ) for messages that are found to have bit-errors that cannot be corrected at the receiver.

For speech and music, the effect of bit-errors is not always catastrophic. Mobile phones can produce audible distortion when bit-errors are occurring, but they can continue working, with less than ideal sound output. Similarly, the visual distortion to images and video can be localised and made non-catastrophic if a little care is taken with the source encoding. Compressed digitised speech, image and video data is more sensitive to bit-errors than uncompressed data. Bit-error control is a vital aspect of most radio communication systems since it allows communication to take place over radio channels that would otherwise be considered unsatisfactory. It is also a transmission power and radiation saving measure as mentioned earlier.

Bit-errors are easiest to deal with when they are evenly spaced out in time, and not too many of them occur at once. 'Burst' errors affect sequences of adjacent bits in a data-stream. In a ‘b-bit burst’ of bit-errors, many erroneous bits may be concentrated within a sequence of b-bits. Thereafter, at least for a while, there may be no bit-errors at all. This does not mean that all bits within the burst are wrong, but generally we will not know which bits are wrong. We only know that there will be a lot of them. To deal with burst errors, data is usually interleaved, resulting in a scattering of the error bits across a wide range of the data.

The words ‘encoder’ and ‘decoder’ are used a little differently in this task. In Tasks 1 and

2, 'encoder' meant 'source encoder' for digitising and compressing a source of speech or music. Now it can also mean a 'forward error control' encoder. Similarly, a 'decoder' can be a source decoder or an FEC decoder. A typical radio transmission scheme will contain a 'source encoder' followed by an 'FEC encoder'. The receiver will contain an 'FEC decoder' followed by a source decoder.

# 2. Support Software

There is some support software provided within comp28512_utils.py. Before starting any experiments, read the following notes.

### 2.1. Converting strings and arrays of samples to arrays of bits

When investigating bit-error control in Python, it is a good idea to convert strings of characters and arrays of quantised samples to arrays containing just ones and zeros, or their Boolean equivalent. Call these 'bit-arrays'. The function 'bytes_to_bit_array' in comp28512_utils.py converts a string of bytes (which may be ASCII characters) to a NumPy array of Boolean types which we may call a 'bit-array'. If S = "BBC", bytes_to_bit_array(S) gives us the 24 element bit-array: 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 1 . The Python function 'numpy_array_to_bit_array' converts a NumPy array of data, which may represent speech or music samples, to a bit-array.

### 2.2. Converting arrays of bits back to strings or samples

The function 'bit_array_to_bytes' converts a bit-array back to a string of bytes.

The function bit_array_to_numpy_array is also available in comp28512_utils.py to convert back to an array of numbers.

### 2.3. Simulating the effect of evenly distributed or bursty bit-errors

Given a bit_array containing just ones and zeros (or their Boolean equivalent), introducing evenly distributed bit-errors with a given bit-error probability (between 0 and 1) is achieved by calling the function 'insert_bit_errors' which is in comp28512_utils.py, and is listed below:

```
def insert_bit_errors(data, bit_error_probability):
    # Insert bit errors with given probability into the bit-array 'data'
    errors = np.random.uniform(size=data.size) < bit_error_probability
    # Generates a bit-array (of Booleans) which we 'excusive-or' with data
    return errors ^ data
```

Introducing bursty bit-errors with a given bit-error probability is also possible.

### 2.4. Reading PESQ scores into an array

Remember the function introduced in Task 1 for reading PESQ scores into an array after a series of experiments. It will be useful again for this Task.

### 2.5. Waterfall graph for minimum shift keying (msk)

Assume the radio channel used to transmit a required bit-stream has a certain bandwidth, say 30,000 Hz. This radio channel will be affected by noise at the receiver, and it is common to measure this noise in watts per Hz. This is 'power spectral density'. If the noise power is $300\times10^{-9}$ watts, the power spectral density, $N_0$, will be $(300\times10^{-9})\div30000 = 10\times10^{-12}$ watts/Hz.

Assume your mobile phone is using 1 watt of power to send speech at128 kbits per second. If the battery holds 18000 joules, how many minutes of talk time will be possible before it expires, neglecting energy consumed by other functions of the phone? Burning 18000 joules at 1 joule/s (watt) gives 18000 seconds. Therefore the answer is 5 hours.

The average energy per bit at the transmitter is then:

1 joule per second ÷ ($128\times10^3$ bits per second) = $(1/128)\times10^{-3}$ joules per bit.

If we now assume that there is a 50 dB (i.e. a factor $10^{-5}$) power loss over the channel from transmitter to receiver (including the effect of transmitting and receiving antennas), the average energy per bit ($E_b$) at the receiver can be calculated.

It is: $(1/128)\times10^{-3} \div 10^5 = (1/128)\times10^{-8}$ joules per bit.

Now we can calculate the value of $E_b/N_0$ at the receiver. $E_b/N_0$ is a very widely used measure of signal to noise ratio. The higher its value, the greater the received signal power in comparison to the noise power at the receiver. It is often converted to dB form by calculating $10\times\log_{10}(E_b/N_0)$ dB.

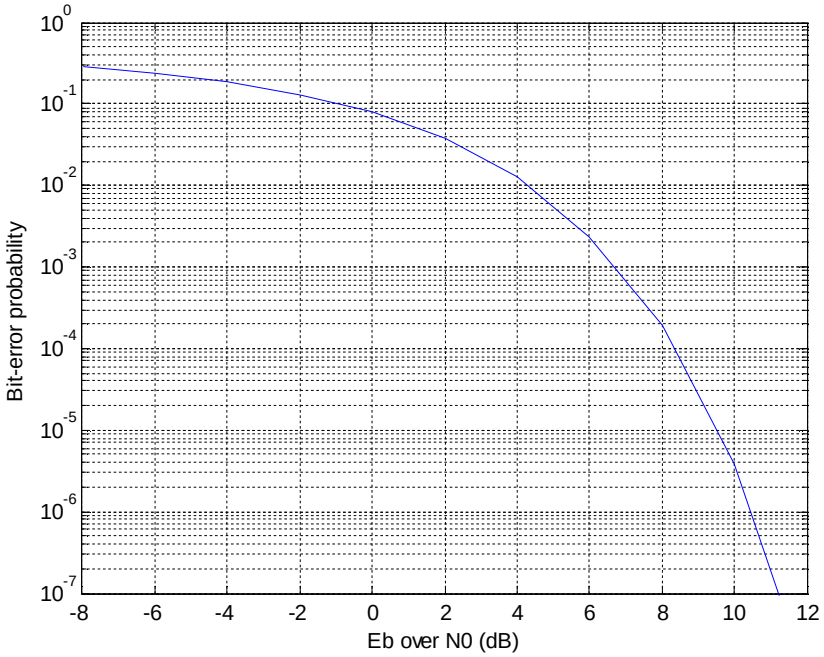
Assuming the modulation method is 'binary minimum shift keying' (binary msk), as used in 2G mobile telephony, a well known formula, given in Lab_support Lecture 3, allows us to calculate the expected bit-error probability given the value of $E_b/N_0$. The formula is:

$$beP = 0.5\times erfc(\sqrt{E_b / N_0})$$

where $E_b/N_0$ is not in dB but in (average joules/bit) / (watts/Hz). The 'complementary error function' erfc(x) = 1 – erf(x) where erf(x) is known as 'the error function'.

It may be shown that $Q(x) = 0.5\times\text{erfc}(x/\sqrt{2})$ is the probability of a sample of Gaussian noise with mean value zero and variance 1, being greater than x. It follows that Q(x/P) is the probability of a sample of zero-mean Gaussian noise of power P watts being greater than x volts. Q(x) as defined here is often (confusingly) also referred to as a complementary error function. But it is the easiest of these error functions to understand, I think. If x is some threshold above which there will be a bit-error, then Q(x/P) or the equivalent erfc expression will give you the bit-error probability.

The error function 'erf' is provided in Python, and probably erfc is provided also. Plotting the formula for beP against $Eb/N_0$ (see below) allows you to deduce the bit-error probability for binary msk for any given value of $E_b/N_0$.




Waterfall graph for msk modulation

### 2.6. CRC8 cyclic redundancy check

The principle of a CRC check was discussed in Lecture 5. It is a method of detecting bit-errors, and is often used with ARQ. If 'xa' is an array of bits to be transmitted or received as a bit-array, we need a function which generates an 8-bit CRC check as another bit-array. This CRC8 check is generally appended to the message bits, thus increasing the bit-array length by 8 bits. In this task, the generator polynomial is required to be $g(x) = x^8 + x^2 + x + 1$. There are two ways of checking whether the bits are correct at the receiver:

(1) Calculate CRC8 for xa without the appended 8 bits. If all 8 bits of this CRC8 match the appended eight bits, it is likely that the received bits are correct.

(2) Calculate CRC8 for all the received bits (including the appended ones). If the result is {0 0 0 0 0 0 0 0}, it is likely that the received bits are correct.

Remember that since the CRC8 bits are transmitted over the noise-affected channel, along with the array xa, they may also have bit-errors.

## 3. Experiments to be carried out

### Part 3.1 Effect of increasing transmit power on speech quality [4 marks]

Write a Python program to read the narrowband speech from 'NarrobandSpeech8k.wav' and to play out the sound without bit-errors. Then demonstrate the speech quality obtained when the transmission power is 1 watt, there is a 50 dB power loss over the channel from transmitter to receiver and $N_0 = 10 \times 10^{-12}$ watts/Hz at the receiver. Do this by introducing evenly distributed bit-errors with the appropriate bit-error probability. Produce a clearly labelled sound-bar for demonstrating the quality of speech produced and obtain its PESQ score, taking as a reference 'NarrobandSpeech8k.wav. Then demonstrate how the speech quality would be improved by increasing the power of the transmissions from 1 watt to 1.35 watts. To do this you will need to calculate the new bit-error probability. Again listen to the speech and give its new PESQ score. Calculate how increasing the power from 1 watt to 1.35 watt affects the talk-time on the mobile phone.

Questions:

1. Explain how the appropriate bit-error probability is calculated in Python for a transmission power of 1 watt.
2. Show how the required graph of bit-error probability against Eb/No (in dB) is produced in Python.
3. What is the bit-error probability when the transmission power is 1 watt?
4. Explain how the evenly distributed bit-errors are introduced.
5. What is the new bit-error probability (for 1.35 watts) and how was this calculated?
6. What is the PESQ score for the narrowband speech transmitted with 1.35 watts?
7. How is the talk time affected?
8. Explain in about one sentence how battery life and bit-error rates are connected.

### Part 3.2: Effect of increasing bit-error probability on narrow-band speech without FEC scheme [4 marks]

Demonstrate the effect of introducing evenly distributed bit-errors on narrow-band speech. Generate 13 versions of the speech file '*Narroband8kspeech.wav*', with gradually increasing bit-error probability: starting from 0.00001 and ending at 0.01. Use a logarithmic scale for the increasing bit-error probability. For each value of bit-error probability, your program should produce a correctly labelled sound-bar for playing out the damaged speech and it

should be compared, using PESQmain, with an error-free version with exactly the same number of samples. A graph of PESQ score against bit-error probability should then be then plotted. The graph should have a logarithmic scale for the bit-error probability and a linear scale for the PESQ scores.

use plot(x,y)

Questions:

1. How did you decide how to choose increasing values of bit-error probability bearing in mind that your graph should have a logarithmic horizontal scale.
2. Comment on the effect of the bit-errors on the sound and how this changes as the bit-error probability increases.
3. Do the PESQ scores agre with your perception of the sound?

**Part 3.3: Effect of increasing bit-error probability on narrow-band speech with a (3,1) repetition FEC scheme [4 marks]**

The function of an FEC encoder is to add 'forward error control' (FEC) information to the bit-stream to allow some bit-errors to be corrected at the receiver. In this case, the FEC strategy is simply to send each speech sample three times. The number of required bits is multiplied by 3, therefore it is a 'one third rate' (3,1) FEC encoder sending 3 bits (or bytes) for every one bit (or byte). This is a rather inefficient FEC scheme, but it serves as an example.
Introduce the simple (3,1) repetition FEC scheme. Simulate a 'majority voting' procedure that may be introduced at the receiver to correct bit-errors. Observe the new graph of PESQ score against bit-error probability that is produced and compare it with the previous version.
To do the comparison efficiently, plot both graphs on the same axes.

Questions:

1. Demonstrate and comment on any improvement in the narrow-band speech quality that is obtained at the expense of the 3-fold increase in the bit-rate. How would you summarise the degree of improvement?
2. What is the effect of the 3-fold increase in bit-rate on the transmit power and on the energy required to transmit the speech segment?
3. If we reduce the energy per bit by a factor of 3, the transmit power becomes what it was before? Consider the example in Section 2.5 where $E_b/N_0$ was 8.93 dB. It would now become $8.93 - 10\times\log_{10}(3) = 8.93 - 4.8$ dB$= 4.3$ dB. From the msk waterfall graph above, the beP now becomes about 0.01. How well does the (3,1) rep coder work at beP=0.01?
4. What other method could be used to reduce the energy required to transmit the speech sentence without reducing the energy per bit? Think about previous Tasks.

**Part 3.4 Apply ARQ to a text message [4 marks]**

Append a CRC8 cyclic redundancy check (with generator polynomial $g(x) = x^8 + x^2 + x + 1$) to a bit-array representing the following text:
"Smart-phones are mobile games consoles and mp3 players that you can also use for telephone calls."
Simulate its transmission over a channel with evenly distributed bit-errors with a bit-error probability ranging from 0.0001 to 0.1 on a logarithmic scale. Convert the characters that are received back to a string and print out the string along with an indication as to whether the CRC8 passes or fails.
If the CRC8 fails, there will likely be incorrect characters in the received message, which are unacceptable. The traditional form of ARQ (Type 1) would discard this message and request a re-transmission in the hope that it might be correct. If the re-transmission also fails its CRC8 check, it is also discarded and a third transmission is then requested. This process continues, and in some cases up to 9 re-transmissions may occur until the process is

abandoned. Implement a demonstration of this form of ARQ.

Questions:

1. If the CRC8 check succeeds, can we deduce that there are no bit-errors?
2. If the CRC8 check fails, can we deduce that there are some bit-errors?
3. Explain how your results demonstrate the effect of evenly distributed bit-errors and ARQ with various values of bit-error probability.
4. For what values of bit-error probability did the ARQ process fail completely?

**Part 3.5: Applying a modified form of ARQ to a text message [4 marks]**

Consider whether you find ARQ, as used above, an efficient approach. Could you improve it by combining failed retransmissions? Such an approach is called 'Chase combining'. Devise a mechanism for combining up to 3 failed retransmissions and demonstrate its effect.

Questions:

1. What could the ARQ mechanism above do if a numerical or string array is received correctly but bit-errors have occurred in the CRC8 check-bits? How often might this occur?
2. State whether you consider ARQ as used in Part 2.4 an efficient method, and explain the reasons for your answer.
3. Explain your mechanism for combining failed retransmissions.
4. Demonstrate the effect of combining 3 failed transmissions.
5. At what value of beP does the new ARQ process fail completely? Is there any improvement over what we got before?
6. If combining failed transmissions turns out to be a good idea, why is it not used in practice, for example in wifi (you need to look this up).

Modified by Barry 24/2/19 _ 1pm