Name: Joaquim Miguel Conceição Espada

Login: con0004

Date: 27th September of 2016

Report of measuring

Title: Error Detection

Task: Learn and understand the methods used to detect errors in data transmission

Scheme:

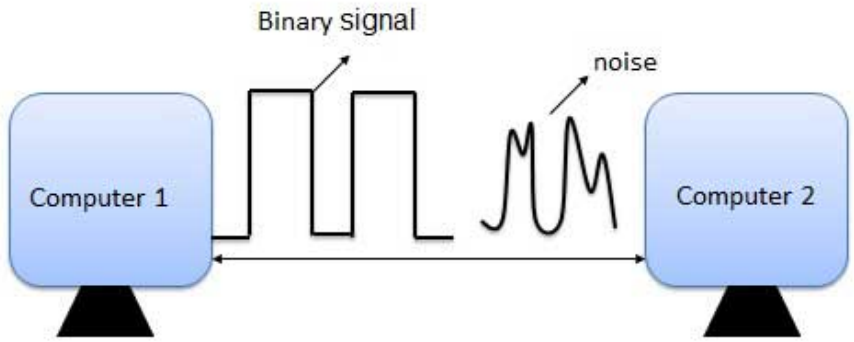

Illustration 1: Data transmission with noise wave

Introduction to problematics:

When data is transmitted often is scrambled by a channel noise and so errors maybe introduced during the transmission from the sender and receiver.
To avoid this situations there are used some techniques and methods that allow detecting some of this kind of errors although they can't detect all of them. This techniques, usually, add additional bits to the original data allowing to detect if an error as occurred during the transmission of the message.
We will study several techniques check sum, parity check, repetition code and CRC.
In final part of the class a simulation of some of this methods was implemented in Octave.

Elaboration:

Parity Check
Parity Check is the simplest method for detecting error. In this method a bit is added to a string of binary code (original data), this bit can be 1 or 0 depending which type of parity used even or odd.

Even parity → Even parity means the number of 1's in the given word including the parity bit should be even (2,4,6,....).

Odd parity → Odd parity means the number of 1's in the given word including the parity bit should be odd (1,3,5,....).

Example:

7 bits of data	(count of 1-bits)	8 bits including parity	
		even	odd
0000000	0	0000000**0**	0000000**1**
1010001	3	1010001**1**	1010001**0**
1101001	4	1101001**0**	1101001**1**
1111111	7	1111111**1**	1111111**0**

This isn't a very reliable technique because the data can changed by the noise and the error will not be detected, because the number of "1" is counted and then is added a "0" or "1" depending in the type of parity used. The noise could flip a pair number of bits and the data will be corrupted and the algorithm will not be detect the error.

Example:

Transmitted Message → 1001
Transmitted Message with even parity → 10010
Received code with error but it isn't detected → 1**1**01**1**

Repetition Code

Repetition Code is one of the most basic techniques and it is used in satellite communication.
In this technique, data is transmitted multiple times. The receiver reads enough of these repetitions until a clear unanimity emerges as to the value of the message.

Perhaps the most popular repetition code is "triple modular redundancy", sending the same message 3 times.

To send *n* data bits with *k* repetitions, *nk* bits need to be transmitted.

Example

A repeated message of 3 bits that is sent 3 times. When it is received, each message is different:

- 101
- 001
- 100

Because the received message wasn't exactly the same int the 3 times, there was detected that some errors occurred.

- 101
- 001
- 101

In this case the unanimity emerges with the message value “101“.

Checksum

Checksum is a small piece of data from a block of digital data with purpose of detecting errors that may have been introduced during transmission. Usually the piece of data passes through a checksum function or algorithm returning a checksum word. When data is transmitted the checksum sequence can be corrupted too.

Parity byte or parity word
This is the simplest checksum algorithm which breaks the data in several “words” with a fixed *n* number of bits and then computes the exclusive or (XOR) between them. The computed word is added to the data as an extra word. The receiver computes XOR of all words and if the result isn’t a word with *n* zeros, the receiver knows a transmission error occurred.

With this checksum, any transmission error which flips a single bit of the message, or an odd number of bits, will be detected as an incorrect checksum. However, an error which affects two bits will not be detected if those bits lie at the same position in two distinct words. Also swapping of two or more words will not be detected. If the affected bits are independently chosen at random, the probability of a two-bit error being undetected is 1/*n*.

Example of a successful message

Sender
Message → NO

N → 4E → 0100 1110
O → 4F → 0100 1111
Checksum sequence→ 0000 0001

Receiver

N→4E → 0100 1110
O→4F → 0100 1111
Checksum sequence→ 0000 0001
0000 0000 → No errors found

CRC

The cyclic redundancy check is a error detecting technique usually used in digital networks and storage devices.

This method algorithm is based in a generic polynomial *G(x)*, can be used with several bits (2,4,8,16,32,…,2^*n*), is very fast to detected errors and can be implemented by hardware or software. The hardware implementation is faster.

The polynomial *M(x)* is represented by the initial message added to the parity bits and must be divisible by *G(x)*.

The receiver tries to $M(x) \div G(x)$ and if the reminder $R(x) \neq 0$ a error was found.

Successful Example

Sender

Message → 10111011
G(X) = x^4 + x + 1 → 10011
M(x) = $Message \times x^r$ → 10111011 **0000**
R(x) of $M(x) \div G(x)$ = 01111
Now the message is ready to sent → 10111011 **1111**

Receiver
Received message → 10111011 **1111**
R(x) of $Recieved\ Message \div G(x)$ = 0 → The transmission was a success

Unsuccessful Example

Sender

Message → 10111011
G(X) = x^4 + x + 1 → 10011
M(x) = $Message \times x^4$ → 10111011 **0000**
R(x) of $M(x) \div G(x)$ = 01111
Now the message is ready to sent → 10111011 **1111**

Receiver
Received message with error → 10111**0**11111
R(x) of $Recieved\ Message \div G(x)$ = 00110 → unsuccessful transmission

Octave Simulation Code

CRC

```
octave:8> m = [1 0 0 0 0 0 1] # message in binary
m =

  1  0  0  0  0  0  1

octave:9> g =[1 0 0 1 1] # generator polynomial G(x) = x^4+x+1
g =

  1  0  0  1  1

octave:10> xr = [1 0 0 0 0] #  x^4
xr =
```

```
  1   0   0   0   0

octave:11> mxr = conv(xr,m) # multiplication of message and    x^4
mxr =

  1   0   0   0   0   0   1   0   0   0   0

octave:12> [q,r]=deconv (mxr,g) # save in vector q the quotient and on r the remainder of the
polynomial division
q =

  1   0   0  -1  -1   0   2

r =

  0   0   0   0   0   0   0   2   1  -2  -2

octave:13> r=mod(r,2) # compute the modulo of r and 2
r =

  0   0   0   0   0   0   0   0   1   0   0   → CRC Sequence
```

Repetition Code

```
octave:16> m(repmat(1:length (m),1,2)) # equivalent to repmat (m,1,2) copies the m two times in one
row
ans =

  1   0   0   0   0   0   1   1   0   0   0   0   0   1
```

Parity Check

```
octave:18> xor(1,0)
ans =  1
octave:19> ans = 0; disp(m);for i=1:length (m) xor (m(i),ans);endfor;disp(ans) # script that computes
xor between all bits of the message
  1   0   0   0   0   0   1
0
```

Conclusion:

After studying the followings techniques: Parity Check, Repetition Code, Checksum and CRC I realized that CRC is the most versatile,reliable and accurate.

The parity check is the simplest method for error detection but it isn't very accurate because the noise could flip a pair number of bits and the data will be corrupted and the algorithm will not be detect the error.

The repetition code is a basic technique that relies on sending multiple times the same message. To send *n* data bits with *k* repetitions, *nk* bits need to be transmitted. The receiver must read enough of these repetitions until a clear unanimity emerges as to the value of the message.

Regarding the Checksum the message when transmitted the checksum sequence and words can be corrupted by noise. This algorithm can't detect error if two bits that lie at the same position in two distinct words are changed.

References:

- Notes from Ing. Pavel Nevlud classes
- https://www.tutorialspoint.com/computer_logical_organization/error_codes.htm
- https://en.wikipedia.org/wiki/Parity_bit
- https://en.wikibooks.org/wiki/Data_Coding_Theory/Repetition_Codes
- https://en.wikipedia.org/wiki/Cyclic_redundancy_check
- https://en.wikipedia.org/wiki/Checksum#Modular_sum
- http://www.di.ubi.pt/~pprata/sdtf/Tk_codigosErros_Manuela.pdf