Name: Bhaswata Sarkar

Computer Science and Engineering, 3rd Year, 1st Semester

Section – A3

Roll no. : 001910501080

Computer Networks Assignment

Assignment #1

Deadline: 12-08-2021

Submission date: 17-08-2021

Problem:

Design and implement an error detection module.

Design and implement an error detection module which has four schemes namely LRC, VRC, Checksum and CRC. Please note that you may need to use these schemes separately for other applications (assignments). You can write the program in any language. The Sender program should accept the name of a test file (contains a sequence of 0,1) from the command line. Then it will prepare the data frame (decide the size of the frame) from the input. Based on the schemes, codeword will be prepared. Sender will send the codeword to the Receiver. Receiver will extract the dataword from codeword and show if there is any error detected. Test the same program to produce a PASS/FAIL result for following cases.

(a) Error is detected by all four schemes. Use a suitable CRC polynomial (list is given in next page).

(b) Error is detected by checksum but not by CRC.

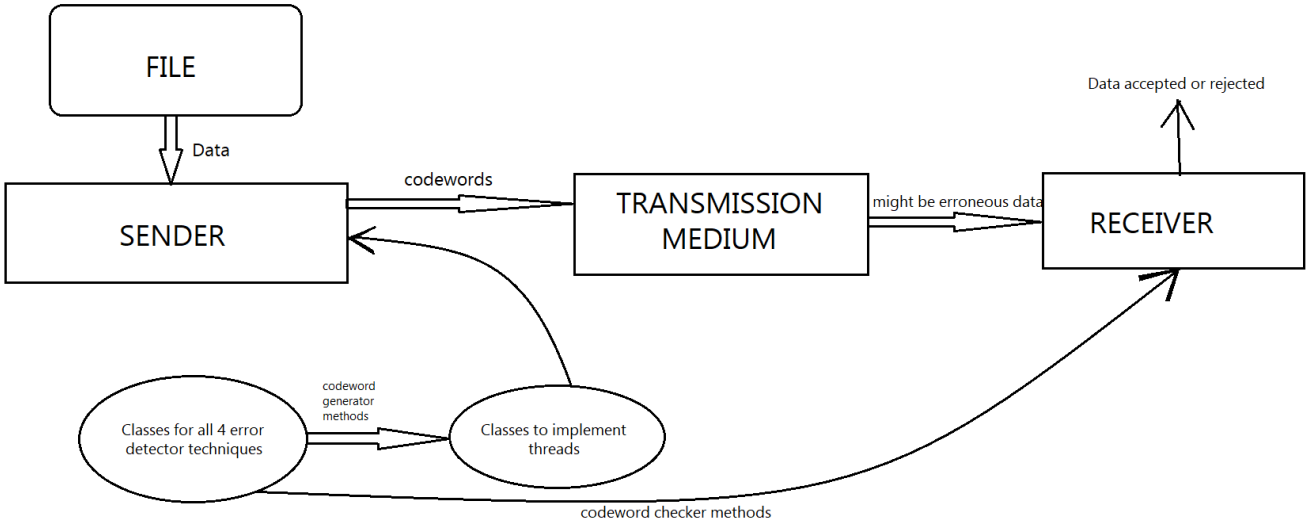
(c) Error is detected by VRC but not by CRC.

[Note: Inject error in random positions in the input data frame. Write a separate method for that.]

Purpose of the program:

4 different error detection techniques are implemented in the program. Those are VRC(Vertical Redundancy Check), LRC(Longitudinal Redundancy Check), Checksum and CRC(Cyclic Redundancy Check). Different test cases will be applied with different type of errors to check how much efficient those methods are.

Procedural organization of the program:



In my program the data will be collected from the file which will be processed in SENDER. All the error detection methods are implemented differently. 4 classes are separately made to implement thread so that all of the 4 codewords from different techniques will be created parallelly. After creating the codewords , these will be sent in Transmission Medium. Here error as per required by the user are added. Then the RECEIVER will check the erroneous data and decide is the data is to be accepted or rejected. As per the result we will find how the methods are performing and how much reliable they are.

Design:

Data = 32 bits

Packets = 4 packets of length 8 bit

errordetection.techniques package

FRAMES.java:

Class to store store a list of data packets and frames

VRC.JAVA,LRC.java,CHECKSUM.java,CRC.java:

These files are under the errordetection.techniques package. They contain the methods for creating the codewords and also checking the codewords if they are erroneous or not.

network package

Sender.java:

This file contains the VRCsender, LRCsender, CHECKSUMsender and CRCsender class to create the threads. It also contains the the Sender class which will create all the codewords.

TransmissionMedium.java:

This file is used to inject different type of errors as per requirement.

Receiver.java:

This file will receive and check the codewords if they are correct or needed to be discarded.

User.java:

For user to implement the error with test cases as per required for analysis.

Input format: A file containing some character sequence of '0' and '1'

Output format: As per required for analysis

Implementation:

- Frames

```
public class FRAMES{//Each string of the frame can be considered as packets

    public ArrayList<String> list;


    public FRAMES(){
         list = new ArrayList<String>();
    }
}
```

Data will be stored in this FRAMES class instances.

VRC

- Function taking argument FRAMES object as list of datawords and returning another FRAMES object as list of codewords

```
//take a data as FRAMES(argument) and return the codewords considering each packet of the FRAMES
//using VRC
public static FRAMES createCodeword(FRAMES dataword_list)
{
    //VRC
    int i,j,count;

    FRAMES codeword_list = new FRAMES();

    for(i=0;i<dataword_list.list.size();i++){
        count = 0;
        for(j=0;j<dataword_list.list.get(0).length();j++)
        {
            if(dataword_list.list.get(i).charAt(j)=='1')
            {
                count++;
            }
        }
        if (count%2==0)//redundancy is added to each packet
            codeword_list.list.add(dataword_list.list.get(i)+'0');
        else
            codeword_list.list.add(dataword_list.list.get(i)+'1');
    }

    return codeword_list;//returning list of codewords
}
```

- Function taking argument FRAMES object as list of codewords and returning Boolean as per error is detected or not

```java
//take FRAMES as argument(list of codewords) and returning boolean
public static boolean checkCodeword(FRAMES codeword_list)
{
    int i,j,count;

    for(i=0;i<codeword_list.list.size();i++) {
        count = 0;
        for(j=0;j<codeword_list.list.get(i).length();j++)
        {
            if(codeword_list.list.get(i).charAt(j)=='1')
            {
                count++;
                //System.out.println(count);
            }
        }

        if (count%2==1)
            return false;//error is detected in a packet
}
    return true;//no error detected
```

LRC

- Function taking argument FRAMES object as list of datawords and returning another FRAMES object as list of codewords

```java
//take a data as FRAMES(argument) and return the codewords considering each packet of the FRAMES
//using LRC
    public static FRAMES createCodeword(FRAMES dataword_list)
    {
        int i,j,count;

        StringBuffer tempstr = new StringBuffer();

        for(j=0;j<dataword_list.list.get(0).length();j++) {
            count = 0;
            for(i=0;i<dataword_list.list.size();i++) {
                if(dataword_list.list.get(i).charAt(j) == '1')
                    count++;;
            }

            if (count%2==0)
                tempstr.append('0');
            else
                tempstr.append('1');
    }
        FRAMES codeword_list = new FRAMES();

        for(i=0;i<dataword_list.list.size();i++) {
            codeword_list.list.add(dataword_list.list.get(i));
        }

        codeword_list.list.add(tempstr.toString());
        return codeword_list;//returning codeword list
```

- Function taking argument FRAMES object as list of codewords and returning Boolean as per error is detected or not

```
    //take the list of frames as argument and checks if its a perfect codeword list
    public static boolean checkCodeword(FRAMES codeword_list) {

        int i,j,count;

        //StringBuffer tempstr = new StringBuffer();

        for(j=0;j<codeword_list.list.get(0).length();j++) {
            count = 0;
            for(i=0;i<codeword_list.list.size();i++) {
                if(codeword_list.list.get(i).charAt(j) == '1')
                    count++;;
            }

            if (count%2==1)
                return false;
    }
        return true;

    }
```

CHECKSUM

- Function taking argument FRAMES object as list of datawords and returning another FRAMES object as list of codewords

```
    //take a data as FRAMES(argument) and return the codewords considering each packet of the FRAMES
    public static FRAMES createCodeword(FRAMES dataword_list)
    {
int i,j,carry=0;

        StringBuffer sum = new StringBuffer();
        //StringBuffer checksum = new StringBuffer();
        for(j=dataword_list.list.get(0).length()-1;j>=0;j--)
            sum.append('0');
        //System.out.println(sum);//////////////////////


        //implementing addition of binary numbers
        for(j=dataword_list.list.get(0).length()-1;j>=0;j--) {
            for(i=0;i<dataword_list.list.size();i++) {
                if(dataword_list.list.get(i).charAt(j) == '1')
                    carry++;
            }

            if (carry%2==0) {
                sum.setCharAt(j, '0');
                carry = carry/2;
                }
            else{
                sum.setCharAt(j, '1');
                carry = carry/2;
                }
            }

        //System.out.println(sum);//////////////////////
```

```
    //adding the carry generated to the sum
    if(carry!=0) {
        for(j=sum.length()-1;j>=0;j--){
            carry = carry + Character.getNumericValue(sum.charAt(j));

        if (carry%2==0) {
            sum.setCharAt(j, '0');
            carry = carry/2;
            }
        else{
            sum.setCharAt(j, '1');
            carry = carry/2;
            }
        }}

    //System.out.println(sum);

        //checksum by inverting the sum
        for(j=sum.length()-1;j>=0;j--) {
            if(sum.charAt(j)=='0')
                sum.setCharAt(j, '1');
            else
                sum.setCharAt(j, '0');
        }


        FRAMES codeword_list = new FRAMES();
        for(i=0;i<dataword_list.list.size();i++) {
            codeword_list.list.add(dataword_list.list.get(i));
        }

        codeword_list.list.add(sum.toString());
        return codeword_list;//returning codeword list

}
```

- Function taking argument FRAMES object as list of codewords and returning Boolean as per error is detected or not

```
public static boolean checkCodeword(FRAMES dataword_list) {

    int i,j,carry=0;

    StringBuffer sum = new StringBuffer();
    //StringBuffer checksum = new StringBuffer();
    for(j=dataword_list.list.get(0).length()-1;j>=0;j--)
        sum.append('0');


    for(j=dataword_list.list.get(0).length()-1;j>=0;j--) {
        for(i=0;i<dataword_list.list.size();i++) {
            if(dataword_list.list.get(i).charAt(j) == '1')
                carry++;
        }

        if (carry%2==0) {
            sum.setCharAt(j, '0');
            carry = carry/2;
            }
        else{
            sum.setCharAt(j, '1');
            carry = carry/2;
            }
        }

    //System.out.println(sum);//////////////////////
```

```
    if(carry!=0) {
        for(j=sum.length()-1;j>=0;j--){
            carry = carry + Character.getNumericValue(sum.charAt(j));

        if (carry%2==0) {
            sum.setCharAt(j, '0');
            carry = carry/2;
            }
        else{
            sum.setCharAt(j, '1');
            carry = carry/2;
            }
        }}

        for(j=sum.length()-1;j>=0;j--)
        {
            if(sum.charAt(j)=='0')
                return false;
        }

    return true;

}
```

CRC

- This function is to calculate the XOR of two numbers

```
private static String xor(String str1, String str2) {//length of str1 and str2 must be same
```

- This function will return the remainder after performing the binary division(for CRC)

```
//modulo-2 division for crc
private static String binaryDivision(String dividendarg){//return remainder
    String dividend = new String(dividendarg);
    int i, len = divisor.length();
    /*for(i=0;i<len-1;i++){
        dividend = dividend + '0';
    }*/

    for(i=0;i<dividend.length()-len+1;i++){
        String temp = new String(dividend.substring(i, i+len));
        if(temp.charAt(0)=='1') {
            dividend = dividend.substring(0,i) + CRC.xor(temp, divisor) + dividend.substring(i+len);
        }
        }
    return dividend.substring(dividend.length()-len+1);//return remainder

}
```

- Function taking argument FRAMES object as list of datawords and returning another FRAMES object as list of codewords

```
//take a data as FRAMES(argument) and return the codewords considering each packet of the FRAMES
//CRC
public static FRAMES createCodeword(FRAMES dataword_list)
{
    int i,j;
    FRAMES codeword_list = new FRAMES();
    //System.out.println(dataword_list.list);
    codeword_list.list = new ArrayList<String>(dataword_list.list);

    for(i=0;i<4;i++) {
        for(j=0;j<CRC.divisor.length()-1;j++) {
        codeword_list.list.set(i,codeword_list.list.get(i).toString()+"0");
        };
    }
    //System.out.println(codeword_list.list);

    for(i=0;i<codeword_list.list.size();i++){
        codeword_list.list.set(i, dataword_list.list.get(i)+binaryDivision(codeword_list.list.get(i)));
    }
    //System.out.println(codeword_list.list);
    return codeword_list;//returning codeword
}
```

- Function taking argument FRAMES object as list of codewords and returning Boolean as per error is detected or not

```
public static boolean checkCodeword(FRAMES codeword_list) {

    int i,j;
    FRAMES temp = new FRAMES();

    for(i=0;i<codeword_list.list.size();i++){
        j=0;
        temp.list.add(binaryDivision(codeword_list.list.get(i)));
        while(j<divisor.length()-1)
        {
            if(temp.list.get(i).charAt(j)=='1')
                return false;
            j++;
        }
    }
    return true;
```

- Class VRCsender will take the list of datawords and the run() method will use VRC object to create dataword in a thread. The getCodewordList() method will return the codeword.

```
class VRCsender implements Runnable{

    private FRAMES dataword_list = new FRAMES();
    private FRAMES codeword_list = new FRAMES();
    VRCsender(FRAMES dataword_list){

    public void run() {

//return the list of codewords
    FRAMES getCodewordList()
}
```

Similarly classes LRCsender, CHECKSUMsender, CRCsender are also present.

Sender

```
    Sender(String filename){
}
```

The constructor of sender class takes the filename as argument and will create the dataword packets from the stream of ‘1’s and ‘0’ are written in the file. Total 32 bits data is created by padding ‘0’s before if needed and then 4, 8 bit data packets are created to use as datawords.

TransmissionMedium

```
public class TransmissionMedium{

    void pushBitError(Sender s)

    void pushBurstError(Sender s,int size,char c)//c='0' or '1'

    void putCustomError(Sender s,int packetnumber, int bitnumber) throws IndexOutOfBoundsException{
}
```

Different methods for injecting different type of errors in the codeword stored in the Sender object.

Receiver

```
public class Receiver{
    Receiver(Sender s){
}
```

Receiver will check the codewords in the sender class are erroneous or not.

Testing program for special cases:

CRC polynomial used: CRC-4-ITU (x^4+x+1)

Test cases

(a) Error is detected by all four schemes.

Data used: 11100111110011111100011000011

4^{th} bit of 3^{rd} packet and 6^{th} bit of 2^{nd} packet is reversed for all 4 codewords.

(Assume indexing is from 0)

All of them managed to reject the data and hence error is detected by all four schemes.

```
     Datawords generated : [00011100, 11111001, 11111000, 11000011]


     VRC frames sent: [000111001, 111110010, 111110001, 110000110]
     LRC frames sent: [00011100, 11111001, 11111000, 11000011, 11011110]
CHECKSUM frames sent: [00011100, 11111001, 11111000, 11000011, 00101101]
     CRC frames sent: [000111000010, 111110011110, 111110001101, 110000111100]


After injecting error


     VRC frames received : [000111001, 111110010, 111110101, 110010110]
     LRC frames received : [00011100, 11111001, 11111010, 11001011, 11011110]
CHECKSUM frames received : [00011100, 11111001, 11111010, 11001011, 00101101]
     CRC frames received : [000111000010, 111110011110, 111110101101, 110010111100]


Data rejected by VRC method
Data rejected by LRC method
Data rejected by CHECKSUM method
Data rejected by CRC method
```

(b) Error is detected by checksum but not by CRC.

Data used: 11000110

The 1st, 3rd, 4th, 5th, 6th, 7th bit of the 3rd packet has error. But CRC fails to detect the error but CHECKSUM successfully rejects the error.

```
     Datawords generated : [00000000, 00000000, 00000000, 11000110]


     VRC frames sent: [000000000, 000000000, 000000000, 110001100]
     LRC frames sent: [00000000, 00000000, 00000000, 11000110, 11000110]
CHECKSUM frames sent: [00000000, 00000000, 00000000, 11000110, 00111001]
     CRC frames sent: [000000000000, 000000000000, 000000000000, 110001100011]


After injecting error


     VRC frames received : [000000000, 000000000, 000000000, 100110010]
     LRC frames received : [00000000, 00000000, 00000000, 10011001, 11000110]
CHECKSUM frames received : [00000000, 00000000, 00000000, 10011001, 00111001]
     CRC frames received : [000000000000, 000000000000, 000000000000, 100110010011]


Data accepted by VRC method
Data rejected by LRC method
Data rejected by CHECKSUM method
Data accepted by CRC method
```

(c) Error is detected by VRC but not by CRC.

Data used: 10011

Some burst error converted all the ‘1’s to ‘0’s. Now VRC is able detect the error but not CRC.

```
     Datawords generated : [00000000, 00000000, 00000000, 00010011]


     VRC frames sent: [000000000, 000000000, 000000000, 000100111]
     LRC frames sent: [00000000, 00000000, 00000000, 00010011, 00010011]
CHECKSUM frames sent: [00000000, 00000000, 00000000, 00010011, 11101100]
     CRC frames sent: [000000000000, 000000000000, 000000000000, 000100110000]


After injecting error


     VRC frames received : [000000000, 000000000, 000000000, 000000001]
     LRC frames received : [00000000, 00000000, 00000000, 00000000, 00010011]
CHECKSUM frames received : [00000000, 00000000, 00000000, 00000000, 11101100]
     CRC frames received : [000000000000, 000000000000, 000000000000, 000000000000]


Data rejected by VRC method
Data rejected by LRC method
Data rejected by CHECKSUM method
Data accepted by CRC method
```

Result and Analysis:

- Bit error:
  Example:
  Data used: 1100110011011101111

```
     Datawords generated : [00000000, 00000011, 00110010, 11101111]


     VRC frames sent: [000000000, 000000110, 001100101, 111011111]
     LRC frames sent: [00000000, 00000011, 00110010, 11101111, 11011110]
CHECKSUM frames sent: [00000000, 00000011, 00110010, 11101111, 11011010]
     CRC frames sent: [000000000000, 000000110101, 001100101001, 111011110001]


After injecting error


     VRC frames received : [000000000, 000000110, 001110101, 111011111]
     LRC frames received : [00000000, 00000011, 00110010, 11101111, 11010110]
CHECKSUM frames received : [00000000, 00000011, 00110010, 11101111, 01011010]
     CRC frames received : [000000000000, 000010110101, 001100101001, 111011110001]


Data rejected by VRC method
Data rejected by LRC method
Data rejected by CHECKSUM method
Data rejected by CRC method
```

```
Bit error analysis :

Successfully identified by VRC out of 100000 : 100000
Successfully identified by LRC out of 100000 : 100000
Successfully identified by CHECKSUM out of 100000 : 100000
Successfully identified by CRC out of 100000 : 100000
```

Bit error is successfully identified 100% by all methods.

- Burst error:

Burst error is generated at random position of the data 01111001011111111011111111111100 after creating all the packets. So some consecutive bits at random position is to be converted all ‘0’s.

CRC polynomial : 10011 (x^4+x+1)

Example:

```
     Datawords generated : [00011100, 11111001, 11111000, 11000011]


     VRC frames sent: [000111001, 111110010, 111110001, 110000110]
     LRC frames sent: [00011100, 11111001, 11111000, 11000011, 11011110]
CHECKSUM frames sent: [00011100, 11111001, 11111000, 11000011, 00101101]
     CRC frames sent: [000111000010, 111110011110, 111110001101, 110000111100]


After injecting error


     VRC frames received : [000111001, 111110010, 100000001, 110000110]
     LRC frames received : [00011100, 11111001, 11111000, 11000011, 11000000]
CHECKSUM frames received : [00011100, 11100000, 11111000, 11000011, 00101101]
     CRC frames received : [000111000010, 111110011110, 110000001101, 110000111100]


Data accepted by VRC method
Data rejected by LRC method
Data rejected by CHECKSUM method
Data rejected by CRC method
```

Here VRC fails to catch the error. (Error is underlined)

- Analyzing burst error occurring single time :

Percentage(%) of success

Size of Burst error	VRC	LRC	CHECKSUM	CRC
3 - bit	77.83	100	100	100
8 - bit	80.51	100	85.21	93.74
15 - bit	74.853	97.524	100	93.69
20 - bit	80.74	97.50	97.56	93.92
22 - bit	80.52	97.52	97.41	91.62

- Analyzing burst error occurring two times. Primarily a burst error affecting 5-bits already occurred. The next burst error is injected varying the size.

Percentage(%) of success

Size of 2nd Burst error	VRC	LRC	CHECKSUM	CRC
5 - bit	79.01	99.08	98.15	98.47
8 - bit	82.46	99.87	96.05	97.42
14 - bit	84.75	99.27	99.52	97.24
20 - bit	82.40	98.18	98.38	96.08
25 - bit	88.47	98.45	99.79	94.19

Conclusion:

- From the above graphs it is clear that VRC is the least efficient scheme. VRC is totally unpredictable. If in total even number of bits are affected then VRC will fail. So it will totally depend on the data how the ‘0’s and ‘1’s are distributed. As per the result it will not exactly depend how big the burst error is.

- LRC fails when even number of bits are affected, all in same position of different data packets. But burst error affects consecutive bits (different position). On an overall we can see burst error of size less than the packet cannot create any false positive because in this case it is impossible two affect bits of same position in different data packets. However burst errors of large scale or multiple burst errors can create the situation. In that case on overall average larger the errors lesser the efficiency of the scheme. The overall performance is pretty good.

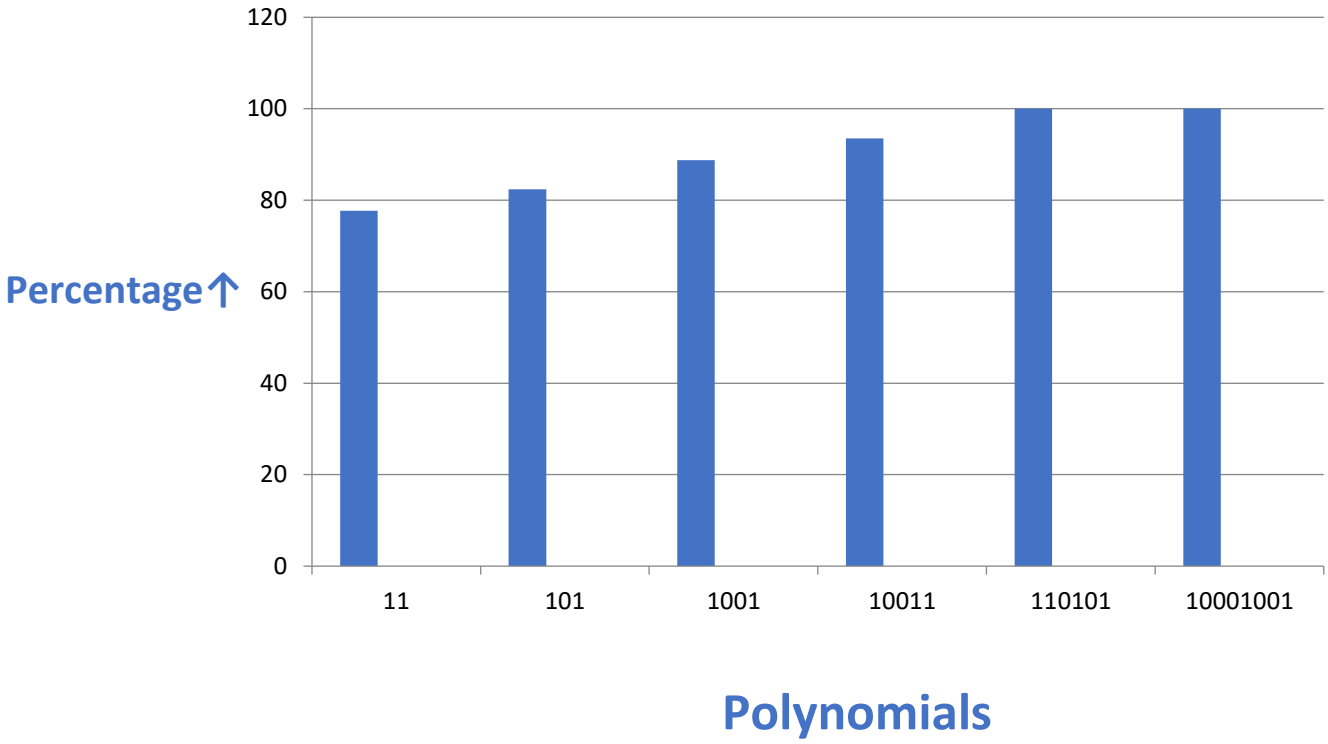
- Pushing more errors is making CRC less efficient. But very large errors are producing more wrong results with respect to LRC. The redundancy in CRC is created from 1 data packet only for that particular packet but in LRC for each data packet multiple redundant bits are created and checked later. This makes the LRC scheme more reliable and less error prone.

- Checksome is appearing unpredictable and can fluctuate from its efficiency in large scale in certain situations.

Analyzing efficiency of different CRC polynomials:

Data used: 01111001011111111011111111111100

CRC polynomial(in binary)	Percentage(%) of success
11	77.69
101	82.37
1001	88.74
10011	93.53
110101	100
10001001	100



Conclusion: Clearly using larger CRC polynomials is providing better success rate.

Possible improvement:

- This project would have been better if was implemented using socket.
- Invalid inputs should have been handled.
- More different test cases might improve the analysis.
- Making this code more compact and maintainable was possible with respect to this one.

Discussion:

This analysis shows how the different schemes is performing but of course doesn't reflect the bigger picture. The sample space of analysis is small and this reflects only a small part of the possible problems or usefulness of the schemes. So drawing a general conclusion with this analysis is not perfect.

Comments:

This assignment was fairly difficult for me. Choosing suitable test cases was a challenge.