Computer Networks

Assignment 1

Md Sahil
001710501029
BCSE-III

Objectives

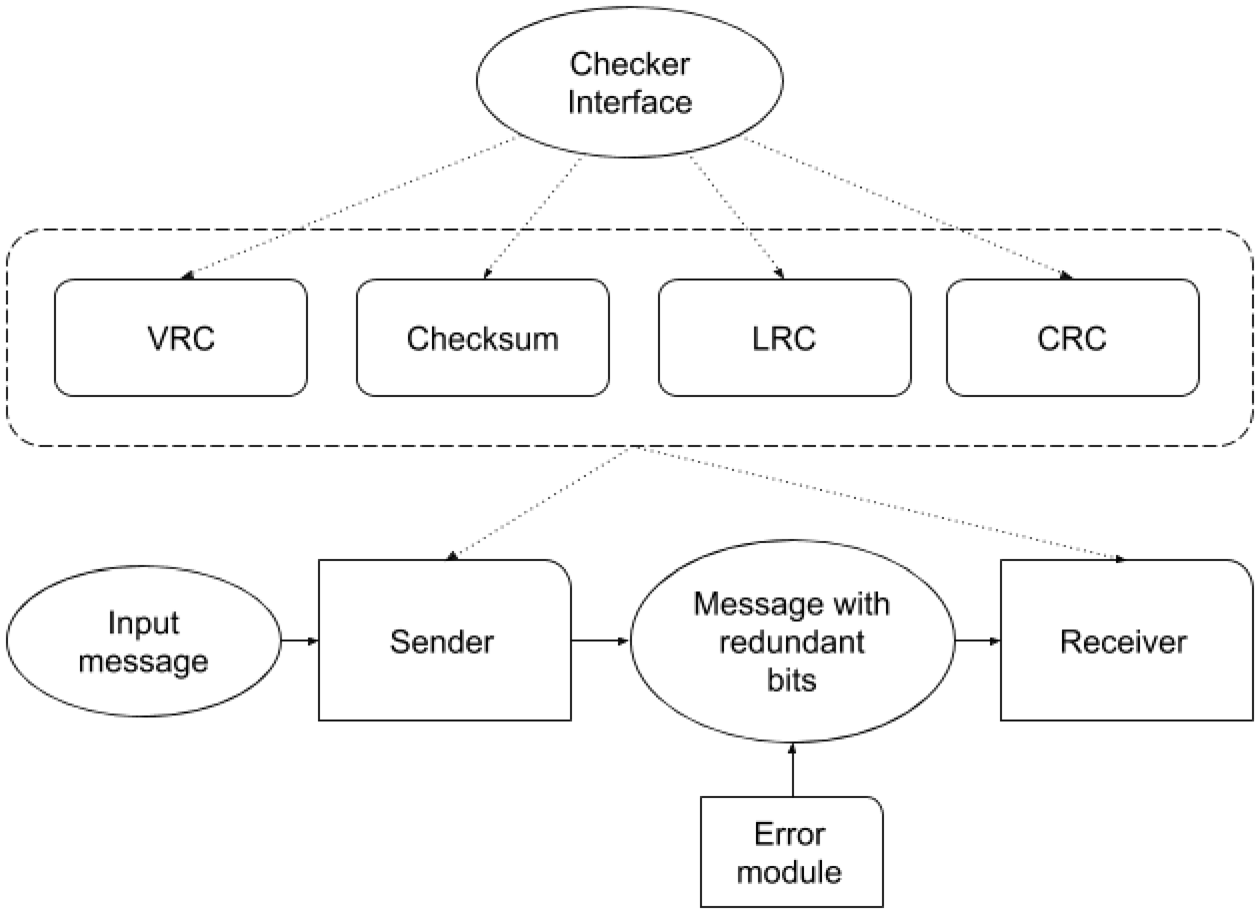
- To design and implement an error detection module.
- Error detection modules to be used are LRC, VRC, Checksum and CRC.
- To Test the program for the following cases:
    - Error is detected by all four schemes
    - Error is detected by Checksum but not by CRC
    - Error is detected by VRC but not by CRC
- To design an error injection module to inject random errors in input frames to test the cases.

Design

Purpose of the program

The program provides an implementation of the four error detection methods, namely LRC, VRC, Checksum and CRC. The program provides a simulation of a data transmission taking place over a network. When data is transmitted from a network it may get subjected to electrical interference, radiation, power surges on either end, or wear and tear on the cable, and a lot more factors. All these disturbances can cause errors in transmission. The data link layer handles these errors by adding extra information about the message when the data is transferred. When the data is received at the receiver’s end, the receiver matches the message with the extra information to check if the message contains any errors in it.

Structure of the program



The four error checker classes are VRC, Checksum, LRC, CRC. All these classes implements the Checker interface. These classes provide methods for redundant bit generation at the senders end and also for valid message checking at the receivers end.

- The Checker interface contains two main methods, ***String generate(String)*** and ***boolean check(String).*** The first method takes an input message as an argument, divides it into frames and adds the redundant bits. The second method, takes a message at the receivers end as the input and checks for the errors.
- The ***VRC, CRC, LRC, Checksum*** classes provide implementation for these methods.The frame size if passed in as an argument for the constructor of these classes. In the case of CRC, there is an extra argument that specifies the generator polynomial.
- All the above classes are packed into the ***errorChecking*** package. The ***Driver*** class contains the main function to run the program.

Code snippets

Checksum Implementation:

```
public class CheckSum implements Checker {
    private int framesize;
    private String addBinaryString(String s1, String s2){
        if(s1.length()!=framesize || s2.length()!=framesize){
            System.out.println("checksum addition bit size error");
            System.exit(0);
        }
        int n1 = Integer.parseInt(s1,2);
        int n2 = Integer.parseInt(s2,2);
        int sum = n1 + n2;
        String result = Integer.toBinaryString(sum);
        if(result.length()<framesize) {
            int d = framesize - result.length();
            String padding = "";
            for(int i=0;i<d;i++) padding += '0';
            result = padding + result;
        }
        // One's complement addition ( we add the carry)
        if(result.length()>framesize){
            String carry = result.substring(0,result.length() - framesize);
            result = result.substring(result.length() - framesize, result.length());
            n1 = Integer.parseInt(result,2);
            n2 = Integer.parseInt(carry,2);
            result = Integer.toBinaryString(n1 + n2);
        }
        if(result.length()<framesize) {
            int d = framesize - result.length();
            String padding = "";
            for(int i=0;i<d;i++) padding += '0';
            result = padding + result;
        }
        return result;
    }

    public CheckSum(int f) {
        framesize = f;
    }
    public String generator(String input) {
        String paddingDummy = "";
        String checksum = "";
        for(int i=0;i<framesize;i++) {
            paddingDummy += '0';
            checksum += '0';
        }
        input = input + paddingDummy.substring((input.length() % framesize));
        String output = "";
        while(!input.isEmpty()) {
            String frame = input.substring(0,framesize);
            input = input.substring(framesize,input.length());
            output = output + frame;
            checksum = addBinaryString(checksum,frame);
        }
        checksum = Integer.toBinaryString(~(Integer.parseInt(checksum,2)));
        checksum = checksum.substring(checksum.length()-framesize,checksum.length());
        System.out.println("Checksum:" + checksum);
        return output + checksum;
    }
    public boolean check(String input) {
        String checksum = "";
        for(int i=0;i<framesize;i++)
            checksum += '0';
        String correct = new String(checksum);

        while(!input.isEmpty()) {
            String frame = input.substring(0,framesize);
            input = input.substring(framesize,input.length());
            checksum = addBinaryString(checksum,frame);
        }

        checksum = Integer.toBinaryString(~(Integer.parseInt(checksum,2)));
        checksum = checksum.substring(checksum.length()-framesize,checksum.length());
        System.out.println("Checksum:" + checksum);

        if(checksum.equals(correct))
            return true;
        else
            return false;
    }
}
```

CRC Implementation:

```
public class CRC implements Checker{
    int frameSize;
    int rBitSize;
    int mBitSize;
    String div;
    CRC (int fs, String d) {
        frameSize = fs;
        div = d;
        rBitSize = d.length() - 1;
        mBitSize = frameSize - rBitSize;
        if(mBitSize <= 0) {
            System.out.println("CRC error: frame size must be larger than redundant bits");
            System.exit(0);
        }
    }
    private char xor(char a, char b) {
        if(a == '1' && b == '1') {
            return '0';
        } else if(a == '0' && b == '0') {
            return '0';
        }
        return '1';
    }
    public String generator(String input) {
        // Applying zero padding to make frames evenly sized
        //System.out.println(div);
        String paddingDummy = "";
        for(int i=0;i<mBitSize;i++)
            paddingDummy += '0';
        input = input + paddingDummy.substring((input.length() % mBitSize));
        String output = "";
        // Iterate per frames
        // Adding redundant bits to each frame
        while( input.length() != 0 ) {
            String s = input.substring(0,mBitSize);
            s = s + paddingDummy.substring(0,rBitSize);
            String r = div;
            String r_new = "";
            input = input.substring(mBitSize,input.length());
            for(int i = 0; i < mBitSize - 1; i++) {
                r_new = "";
                for(int j = 0; j < rBitSize + 1; j++) {
                    r_new += xor(r.charAt(i),s.charAt(i+j));
                }
                r = r_new;
                r = r.substring(1,r.length()) + s.charAt(i+rBitSize);
            }
            output += s.substring(0,s.length() - r_new.length()) + r_new;
        }
        check(output);
        return output;
    }
    public boolean check(String input) {
        String r = div;
        String r_new = "";
        String correct = "";
        for(int i=0;i<r.length();i++)
            correct += '0';
        while( input.length() != 0 ) {
            String s = input.substring(0,frameSize);
            r = div;
            r_new = "";
            input = input.substring(frameSize,input.length());
            for(int i = 0; i < mBitSize - 1; i++) {
                r_new = "";
                for(int j = 0; j < rBitSize + 1; j++) {
                    r_new += xor(r.charAt(i),s.charAt(i+j));
                }
                r = r_new;
                r = r.substring(1,r.length()) + s.charAt(i+rBitSize);
            }
        }
        if(r.equals(correct))
            return true;
        else
            return false;
    }
}
```

Test cases

VRC

```
Sender:------
Input :       1001010110100011
Output :      100101010101000110
Receiver:------(without error)
Received input: 100101010101000110
True

Sender:------
Input :       1001010110100011
Output :      100101010101000110
Receiver:------(without error)
Received input: 000101010101000110
False

Sender:------
Input :       1001010110100011
Output :      100101010101000110
Receiver:------(with error)
Received input: 111101010101000110
True
```

In the third test case VRC is unable to detect the error.

LRC

```
Sender:------
Input :        1001010110100011
Frames:
10010101
10100011
00110110
Output :       100101011010001100110110
Receiver:------(without error)
Received input: 100101011010001100110110
True

Sender:------
Input :        1001010110100011
Frames:
10010101
10100011
00110110
Output :       10101011010001100110110
Receiver:------(with error)
Received input: 110111011010001100110110
False

Sender:------
Input :        1001010110100011
Frames:
10010101
10100011
00110110
Output :       100101011010001100110110
Receiver:------(with error)
Received input: 000101010010001100110110
True
```

Here in the second case LRC is able to detect the errors but it fails to detect the errors in the 3rd case.

Checksum

```
Sender:------
Input :        1001010110100011
10010101
10100011
00110110
Output :       100101011010001100110110
Receiver:------
Received input: 100101011010001100110110
True


Sender:------
Input :        1001010110100011
Checksum:11000110
Output :       100101011010001111000110

Inflicting error
Errors at:3,12,

Receiver:------
Received input: 100001011010101111000110
Checksum:00001000
False
```

CRC (7,4) using generator- 1101

```
Sender:------
Input :        1001010110100011
Output :       1001011010101110100000011100
Receiver:------
Received input: 1001011010101110100000011100
True

Sender:------
Input :        1001010110100011
Output :       1001011010101110100000011100
Inflicting error
Errors at:12,20,
Receiver:------
Received input: 1001011010100110100010011100
False
```

Analysis of edge cases

Error is detected by checksum and not by CRC

Input: 10101011

CRC Generated code word: 1010001 1011100
Received code word at receiver: 101**1**1**0**0 1011100
Error at position: 4th, 5th and 7th bits

DATA WORD	GENERATED DATA WORD	RECEIVED DATE WORD	RECEIVED CODE WORD	VALID/INVALID
1010	1010001	101**1**1**0**0	1011	VALID
1011	1011100	1011100	1011	VALID

The above data recognized as valid by CRC

Checksum:
Framesize 4

```
 1010
 1011
10101
```

Addinng carry:

```
0101
   1
0110
```

Generated code word: 101010110110
Received code word at receiver: 101**1**0**0**010110
Error at position: 4th, 5th and 7th bits

```
 1011
 0001
 0110
10010
```

The above data is recognized as invalid by Checksum

Error is detected by VRC and not by CRC

Input: 10101011

CRC Generated code word: 1010001 1011100
Received code word at receiver: 101**110**0 1011100
Error at position: 4th, 5th and 6th bits

DATA WORD	GENERATED DATA WORD	RECEIVED DATE WORD	RECEIVED CODE WORD	VALID/INVALID
1010	1010001	101**110**0	1011	VALID
1011	1011100	1011100	1011	VALID

The above data recognized as valid by CRC

VRC Generated code word: 10100011
Received code word at receiver: 101**110**0
Error at position: 4th, 5th and 6th bits
The last parity bit is 0 bit the parity of the data word is 1
The above data is recognized as invalid by VRC

Comments

This was a very insightful assignment. We got to know a lot about error checking taking place networking :-) .