ASSIGNMENT 1

Name: Imon Raj
Class: BCSE III
Roll: 002010501098
Section: A3
Subject: Computer Networks Lab Report

PROBLEM STATEMENT: 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 (not limited to).

DESIGN: In this assignment we are going to implement different error detection modules. It will have functionalities of LRC, VRC, CRC, CHECKSUM techniques.

We will use Java as a programming language to create the modules of this assignment. We will use Object Oriented Programming approach to solve the problems.

Basically, we will have three important classes – **Sender**, **Receiver**, **Channel** and one helper class **BinaryArithmetic.**

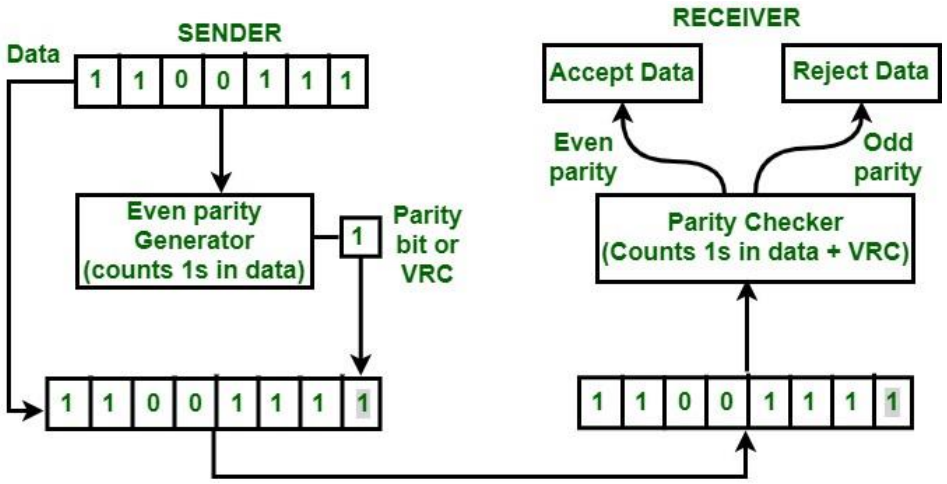
- **Sender** class will have different functions for loading the raw data from a specified file, creating codewords using CRC, LRC, VRC, CheckSum methods and sending them to the channel.
- **Channel** class will simulate the transmission impairments by flipping any number of bits in the codeword sent by the Sender. It will use Java's random function to generate the random integer.
- **Receiver** class will have similar methods as the Sender to validate the codeword received from the Channel. If the channel had inserted any error, the receiver may or may not detect it.

DIFFERENT ERROR DETECTION TECHNIQUES:

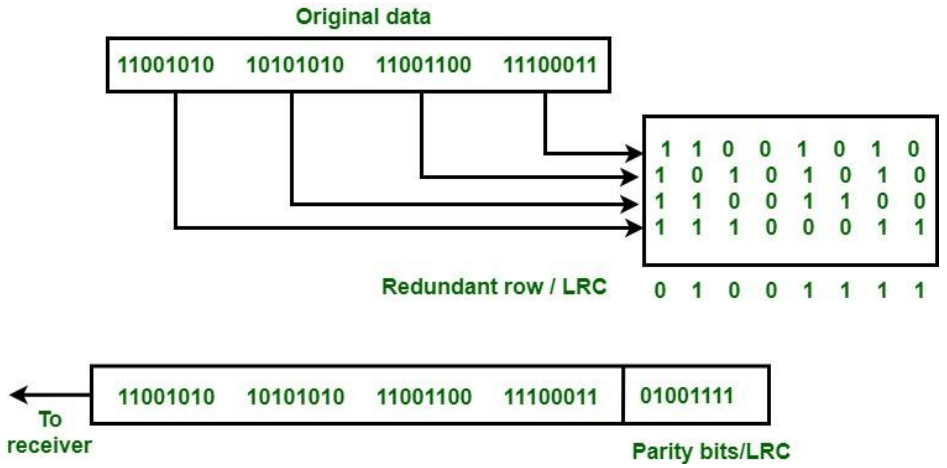
The sender and receiver are bound to maintain certain rules and protocols during data communication. Sender prepares the codeword using any of the techniques and the receiver uses the same rule to detect the error. Now the different techniques are described below briefly –

1. Vertical Redundancy Check(VRC): In this error detection technique, a redundant bit called parity bit is appended to every data unit so that

total number of 1's in the unit (including parity bit) becomes even. At the receiver, all received bits are checked through even parity checking function. If it counts even 1's data unit passes. If it counts odd number of 1's, it means error has been introduced in the data somewhere. Hence receiver rejects the whole data unit.



2. Longitudinal Redundancy Check(LRC): In this error detection technique, the sender divides the complete dataword into multiple packets of same size and then, putting them one below another, parity bits are calculated which finally generates a parity group same as data packet size and this extra packet is then added to dataword to form the codeword. The receiver follows the same process and if it gets all zeros then it detects no error, otherwise there was error during transmission.



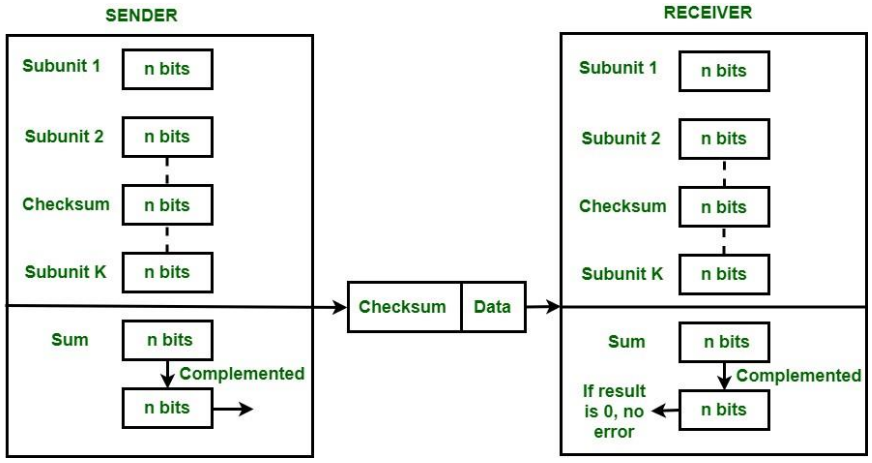
3. Cyclic Redundancy Check(CRC): This is another error detection technique used during data communication. CRC uses Generator Polynomial which is available on both sender and receiver side. An example generator polynomial is of the form like x3 + x + 1. This generator polynomial represents key 1011. Then we append n zero bits at the end of the dataword where n is the degree of the generator polynomial. After that, modulus 2 division is done to this bit sequence to collect the final remainder, which is finally appended to the dataword to create the codeword to be transferred. Receiver uses the same technique to find the final remainder

and if the remainder has all zero bits, then no error is detected and otherwise there is some error in the received bit sequence and it is rejected.

Example – Consider data – 100100 and Polynomial – 1101, final code to be sent - 100100001

```
          111101
       ____________
1101  | 100100000
        1101
        ----
         1000
         1101
         ----
          1010
          1101
          ----
           1110
           1101
           ----
            0110
            0000
            ----
             1100
             1101
             ----
              001
              ---
```

4. CheckSum: This is another way of adding redundancy to the end of dataword for error detection purpose. In this process dataword is divided into list of packets of a pre-defined same size and then they are added in binary arithmetic, any carry generated during this process is added to the sum. The final sum is complemented using one's complement and this is the checksum for that dataword. This is appended to the dataword to send. Receiver uses the same way and if it gets a checksum with all zero bits then no error is detected and if not, then there is error in the received code and thereby rejected.



IMPLEMENTATION(Various Functions Used):

loadData(): Used to load any binary data from file using java.io.*

.

createCheckSumData() & **validateCheckSumData():** First one used in sender class and second one at Receiver class. Both does the similar work of calculating the checksum given a binary sequence and data-packet size. Firstly the whole binary sequence is divided into a group of data-packet-sized strings. This list of binary strings are then added using binary arithmetic in the BinaryArithmetic class.

BinaryArithmetic class contains two necessary functions for this operation – addValues() and addTwoValues(). The answer returned by the addValues() function is complemented to finalize the checksum. In case of Sender it just appends the checksum to the data and in case of Receiver it validates the checksum to detect any error.

createCRCdata() & validateCRCdata(): First one is present in Sender class and second one is at Receiver class. Both does the work of calculating CRC, where polynomial is given. For the division, it again uses a function from BinaryArithmetic class which takes divisor and dividend(appended zeros to the actual sequence), does the division and finally returns the remainder which is nothing but the CRC for a given binary sequence. Sender appends this CRC to raw data and Receiver in its case validates the CRC to detect any error.

createLRCdata() & validateLRCdata(): Both does the work of calculating LRC for a given binary sequence. Binary sequence is divided into multiple groups of same size as the decided data-packet size. Considering them putting one below another, parity bits are calculated longitudinally, taking even parity as convention. Finally, we get the LRC. LRC used by Sender to append to the data and, Receiver validates the calculated LRC using even-parity only.

createVRCdata() & validateVRCdata(): This scheme implementation is easiest of all. One parity bit is added to every data packet considering the even parity in that packet. Receiver also validates each packet whether even parity is maintained or not.

introduceRandomError(): This function present in Channel class simulates the noise or impairments which are present in the actual data-communication scenario. It takes a random number of errors to be inserted and those many errors are inserted into the binary sequence in random indexes by flipping the bits. This function internally uses a function randomInteger() which returns a random integer within the given start and end points.

xOr(): Present in BinaryArithmetic class. It does xOr of two bits.

addTwoValues(): Present in BinaryArithmetic class. Adds two binary strings and if any carry is generated added to the sum to get the final answer.

addValues(): This is a function to add list of binary strings using the function mentioned above.

Comments:

This assignment indirectly helped me to polish my concepts of OOP in Java. Instead of files, socket or pipe programming could be used to implement the simulation for data communication.

Single bit errors can be detected by all the schemes. In case of burst error schemes may or may not detect them. As an example – 10110 is data then for VRC parity will be 1 so sent code will be 101101 and if two bits flip to become 001111, then this error couldn't be detected by VRC. Similarly – for LRC, if multiple bits in same longitudinal line flip in such a way that evenparity doesn't violates then, no error is detected. In CRC, if width of bits flipped is more than the degree of polynomial then, error may or may not be detected. In case of checksum, suppose two data packets increased and decreased by same amount because of flipped bits then, no error is detected by checksum.