Lab Report-03

CSE 2213: Data and Telecommunication Lab
Batch: 29/2nd Year 2nd Semester 2024

Report's Title: Implementation of CRC for Error Detection and Single-Bit Error Correction.

Submitted By:

Suraya Jannat Mim (Roll: 17)
Anisha Tabassum (Roll: 19)

Course Instructors:

Dr. Md. Mustafizur Rahman (MMR)
Mr. Palash Roy (PR)

1. Introduction

In digital communication and storage systems, data integrity is crucial. Data can be corrupted due to noise, signal loss, or hardware failure. To ensure the correctness of transmitted or stored data, error detection and correction techniques are widely used.

Cyclic Redundancy Check (CRC) is one of the most efficient and commonly used methods for error detection. It uses polynomial division to generate a checksum which is appended to the data. At the receiver end, the same CRC algorithm is applied to check whether the data has been altered. Although CRC is primarily an error detection method, with proper implementation and under specific conditions, it can also be used to correct **single-bit errors**.

2. Objectives

- To understand the working principle of CRC in error detection.
- To implement CRC for detecting single-bit and multiple-bit errors.
- To attempt single-bit error correction using CRC.
- To analyze the effectiveness of different CRC generator polynomials.

3. Algorithms / Pseudocode

Algorithm for CRC Error Detection and Single-Bit Error Correction

1. **Sender Side:**
    - Input the data bits.
    - Input the generator polynomial (divisor).
    - Append `n-1` zeros to the data, where `n` is the length of the divisor.
    - Perform modulo-2 division (XOR) of data by the divisor.
    - Append the CRC remainder (checksum) to the original data.
    - Send the codeword (data + checksum).
2. **Receiver Side:**
    - Receive the codeword.
    - Perform modulo-2 division using the same generator polynomial.
    - If remainder is zero: No error detected.
    - If remainder is non-zero: Error detected.
    - Try flipping one bit at a time to detect and correct single-bit error (re-calculate CRC each time).
    - If corrected: Display corrected codeword.
    - If not corrected: Report uncorrectable error.

4. Implementation

Server side:

Part1:

```java
import java.io.*;
import java.net.*;

public class server2 {
    public static void main(String[] args) throws IOException {
        int port = 6009;
        System.out.println("Server is connected at port no: " + port);
        System.out.println("Server is connecting");
        System.out.println("Waiting for the client");

        ServerSocket ss = new ServerSocket(port);
        Socket s = ss.accept();

        System.out.println("Client request is accepted at port no: " + s.getPort());
        System.out.println("Server's Communication Port: " + s.getLocalPort());

        DataInputStream dis = new DataInputStream(s.getInputStream());

        String original = dis.readUTF();
        String received = dis.readUTF();
        String generator = dis.readUTF();

        System.out.println("Received Codeword: " + received);


        String padded = received + "0".repeat(generator.length() - 1);
        String remainder = calculateCRC(padded, generator);
        System.out.println("Calculated Remainder: " + remainder);

        if (remainder.matches("0+")) {
            System.out.println("No error detected in transmission.");
        } else {
            System.out.println("Error detected in transmission!");
            int errors = countBitErrors(original, received);
            System.out.println("Estimated number of bit errors: " + errors);
        }
```

Part2:

```
        dis.close();
        s.close();
        ss.close();
    }

    public static String calculateCRC(String input, String generator) {
        int k = generator.length();
        char[] data = input.toCharArray();
        char[] gen = generator.toCharArray();

        for (int i = 0; i <= data.length - k; i++) {
            if (data[i] == '1') {
                for (int j = 0; j < k; j++) {
                    data[i + j] = (data[i + j] == gen[j]) ? '0' : '1';
                }
            }
        }

        return new String(data).substring(data.length - (k - 1));
    }

    public static int countBitErrors(String a, String b) {
        int count = 0;
        for (int i = 0; i < Math.min(a.length(), b.length()); i++) {
            if (a.charAt(i) != b.charAt(i)) {
                count++;
            }
        }
        return count;
    }
}
```

Client Side:

Part1:

```java
import java.io.*;
import java.net.*;
import java.util.*;

public class client2 {
    public static void main(String[] args) throws IOException {
        String ip = "192.168.0.109";
        int port = 6009;
        String generator = "1101";

        Socket s = new Socket(ip, port);
        System.out.println("Client connected to the server on Handshaking port " + port);
        System.out.println("Client's Communication Port: " + s.getLocalPort());
        System.out.println("Client is Connected");


        BufferedReader input = new BufferedReader(new FileReader("input.txt"));
        String txt = input.readLine();
        input.close();
        System.out.println("File Content: " + txt);

        StringBuilder sb = new StringBuilder();
        for (char c : txt.toCharArray()) {
            sb.append(String.format("%8s", Integer.toBinaryString(c)).replace(' ', '0'));
        }
        String binaryData = sb.toString();
        System.out.println("Converted Binary Data: " + binaryData);


        int k = generator.length();
        String padded = binaryData + "0".repeat(k - 1);
        System.out.println("After Appending zeros Data to Divide: " + padded);


        String crc = calculateCRC(padded, generator);
        System.out.println("CRC Remainder: " + crc);


        String codeword = binaryData + crc;
        System.out.println("Transmitted Codeword to Server: " + codeword);


        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of bits to flip for error simulation: ");
        int flipCount = sc.nextInt();
```

Part2:

```
        String corrupted = flipMultipleBits(codeword, flipCount);
        System.out.println("Corrupted Codeword (Bit flipped): " + corrupted);


        DataOutputStream dos = new DataOutputStream(s.getOutputStream());
        dos.writeUTF(codeword);
        dos.writeUTF(corrupted);
        dos.writeUTF(generator);

        dos.close();
        s.close();
    }

    public static String calculateCRC(String input, String generator) {
        int k = generator.length();
        char[] data = input.toCharArray();
        char[] gen = generator.toCharArray();

        for (int i = 0; i <= data.length - k; i++) {
            if (data[i] == '1') {
                for (int j = 0; j < k; j++) {
                    data[i + j] = (data[i + j] == gen[j]) ? '0' : '1';
                }
            }
        }

        return new String(data).substring(data.length - (k - 1));
    }

    public static String flipMultipleBits(String s, int numBitsToFlip) {
        Random r = new Random();
        char[] chars = s.toCharArray();
        int length = s.length();
        Set<Integer> flippedIndices = new HashSet<>();

        while (flippedIndices.size() < numBitsToFlip) {
            int i = r.nextInt(length);
            if (!flippedIndices.contains(i)) {
                chars[i] = (chars[i] == '0') ? '1' : '0';
                flippedIndices.add(i);
            }
        }

        return new String(chars);
    }
}
```

5. Result Analysis

Server:

```
PS C:\Users\mimro\alien\Java-Practice-> javac server2.java
PS C:\Users\mimro\alien\Java-Practice-> java server2
Server is connected at port no: 6009
Server is connecting
Waiting for the client
Client request is accepted at port no: 64018
Server's Communication Port: 6009
Received Codeword: 0100111001000101110011000100110001001111001
Calculated Remainder: 110
Error detected in transmission!
Estimated number of bit errors: 3
```

Client:

```
PS C:\Users\mimro\alien\Java-Practice-> javac client2.java
PS C:\Users\mimro\alien\Java-Practice-> java client2
Client connected to the server on Handshaking port 6009
Client?s Communication Port: 64018
Client is Connected
File Content: HELLO
Converted Binary Data: 0100100001000101010011000100110001001111
After Appending zeros Data to Divide: 0100100001000101010011000100110001001111000
CRC Remainder: 001
Transmitted Codeword to Server: 0100100001000101010011000100110001001111001
Enter number of bits to flip for error simulation: 3
Corrupted Codeword (Bit flipped): 0100111001000101110011000100110001001111001
```

6. Discussion

CRC is powerful for detecting both single-bit and burst errors. Its efficiency depends on the polynomial used:

- CRC successfully **detects single-bit errors** using a standard generator like `1101`.
- It also detects **most multiple-bit errors**, though detection depends on the generator and error pattern.
- In this implementation, the **client allows choosing how many bits to flip**, making it flexible for testing.
- The **server detects errors using CRC** and estimates the number of flipped bits by comparing original and received codewords.
- **CRC cannot correct errors**, especially multiple-bit ones—it is strictly for detection.
- In real systems, the receiver doesn't have the original codeword, so **bit comparison is only useful for simulation**.
- The experiment demonstrates the **effectiveness of CRC** in identifying errors and highlights its limitations in error correction.

---

7. Learning and Difficulties

Learnings:

- Practical implementation of modulo-2 division.
- Role of CRC polynomials in determining error detection capabilities.
- Client-server socket communication for simulating real-world transmission.

Difficulties Faced:

- Bitwise XOR implementation during CRC division.
- Managing byte-string conversion in socket communication.
- Detecting multiple-bit errors but failing to correct them due to CRC limitations.

---

8. Conclusion

This lab experiment successfully demonstrated the power of CRC in detecting errors and correcting single-bit errors. Through the implementation, we learned the limitations of CRC in correcting multiple-bit errors and how polynomial selection affects the detection capability. This experiment lays a strong foundation for understanding more advanced error correction methods in communication systems.