Department of Computer Science and Engineering
University of Dhaka

Title:

Implementation of CRC for Error Detection and Single-Bit Error Correction

CSE 2213: Data and Telecommunication Lab

Batch-29 (2nd Year 2nd Semester 2025)

Course Instructors

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

Submitted by-

Sumaiya Rahman Soma(SK-07)
Tasnova Shahrin(SK-51)

1.Introduction

In digital communication and data storage, it's important to make sure that the data remains correct. When data is sent from one place to another, it can sometimes get changed because of noise or interference. To check if any errors happened, one of the most commonly used and efficient methods is called Cyclic Redundancy Check (CRC).

CRC works by dividing the original data using a fixed rule called a polynomial. The result of this division is called the remainder. This remainder is added to the original data before sending it, and this combined part is called the checksum.

When the data is received, the receiver performs the same division again. If the new remainder is zero, it means the data was received correctly. If it’s not zero, it means an error occurred during transmission.

CRC is mainly used for error detection, but it can also help in correcting single-bit errors by trying to flip each bit one by one and checking if the checksum becomes valid again. However, multiple-bit errors (burst errors) cannot be corrected this way — CRC can only detect them, not fix them.

2.Objectives

- To implement CRC-based error detection over a client-server communication system.
- To simulate both single-bit and burst errors during transmission.
- To detect and attempt correction of single-bit errors on the receiver side.
- To analyze the performance of different CRC polynomial generators (CRC-8, CRC-10, CRC-16, CRC-32).

3.Algorithms/Pseudocode

CRC Generation (Client Side):

- Read input data from file.
- Convert each character of the input text to its 8-bit binary representation.
- Append (n-1) zero bits to the end of the binary data, where n is the length of the generator polynomial.
- Perform binary division (modulo-2) on the extended binary string using the given generator polynomial.
- Take the remainder from the division and append it to the original binary data to form the final codeword.
- Send the codeword and the generator polynomial to the server using socket communication.

CRC Detection and Correction (Server Side):

- Receive codeword and generator polynomia**l** from the client.
- Select the type of error to simulate (single,burst,none).
- If an error is chosen, modify the codeword accordingly:
    - For a single-bit error: flip one random bit.
    - For a burst error: flip 4 bits in a row.
    - For none: Keep the codeword unchanged.
- Perform modulo-2 division on the received codeword using the generator polynomial.
- If the remainder is 0,the data is valid and there is no error.
- If the remainder contains one or more '1's, an error is detected in the transmission.
- To correct the error, the server tries flipping each bit in the received codeword one by one and recalculates the CRC remainder for each flipped version.
- If flipping a particular bit results in a zero remainder, that bit is considered a candidate for correction, and the search stops immediately upon the first such

bit found.

- If a single candidate bit is found, the server flips that bit in the codeword to correct the single-bit error and outputs the corrected codeword.
- If no bit flip leads to a zero remainder, the server concludes that single-bit correction is not possible or there is multiple bit error and reports the error as uncorrectable.

4.Implementation:

4.1 Client Side

Client.java > Client > main(String[])

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

public class Client {

    public static String xor(String a, String b) {
        String result = "";
        for (int i = 1; i < b.length(); i++) {
            if (a.charAt(i) == b.charAt(i)) {
                result += '0';
            } else {
                result += '1';
            }
        }
        return result;
    }

    public static String mod2Div(String dividend, String divisor) {
        int curr = divisor.length();
        String tmp = dividend.substring(beginIndex:0, curr);

        while (curr < dividend.length()) {
            if (tmp.charAt(index:0) == '1') {
                tmp = xor(divisor, tmp) + dividend.charAt(curr);
            } else {
                tmp = tmp.substring(beginIndex:1) + dividend.charAt(curr);
            }
            curr++;
        }

        if (tmp.charAt(index:0) == '1') {
            tmp = xor(divisor, tmp);
        } else {
            tmp = tmp.substring(beginIndex:1);
        }

        return tmp;
    }
```

```java
Run | Debug
public static void main(String[] args) {
    try {
        Socket socket = new Socket(host:"localhost", port:5000);
        System.out.println(x:"Client connected to the server on Handshaking port 5000");

        System.out.println("Client's Communication Port: " + socket.getLocalPort());
        System.out.println(x:"Client is connected");
        DataOutputStream out = new DataOutputStream(socket.getOutputStream());

        BufferedReader br = new BufferedReader(new FileReader(fileName:"input.txt"));
        String data = br.readLine();
        br.close();

        System.out.println("File content: " + data);

        String binaryData = "";
        for (char ch : data.toCharArray()) {
            binaryData += (String.format(format:"%8s", Integer.toBinaryString(ch)).replace(oldChar:' ', newChar:'0'))
        }

        System.out.println("Converted Binary Data: " + binaryData);

        Scanner scanner = new Scanner(System.in);
        System.out.print(s:"Enter CRC type (8 / 10 / 16 / 32): ");
        int type = scanner.nextInt();
        scanner.close();

        String generator = "";
        if (type == 8) {
            generator = "100000111"; // CRC-8
        } else if (type == 10) {
            generator = "11000110011"; // CRC-10
        } else if (type == 16) {
            generator = "1100000000000101"; // CRC-16
        } else if (type == 32) {
            generator = "100000100110000010001110110110111"; // CRC-32
        } else {
            System.out.println(x:"Invalid CRC type selected.");
            return;
        }

        String appendedData = binaryData;
        for (int i = 0; i < generator.length() - 1; i++) {
            appendedData += '0';
        }
```

```java
            System.out.println("After Appending zeros Data to Divide: " + appendedData);

            String remainder = mod2Div(appendedData, generator);
            System.out.println("CRC Remainder: " + remainder);

            String codeword = binaryData + remainder;
            System.out.println("Codeword to send: " + codeword);

            PrintWriter output = new PrintWriter(socket.getOutputStream(), autoFlush:true);
            output.println(codeword);
            output.println(generator);

            socket.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

4.2 Server Side

Server.java > Server > mod2div(String, String)

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

public class Server {

    public static String xor(String a, String b) {
        String result = "";
        for (int i = 1; i < b.length(); i++) {
            result += (a.charAt(i) == b.charAt(i)) ? '0' : '1';
        }
        return result;
    }

    public static String mod2div(String data, String key) {
        int keylen = key.length();
        String part = data.substring(beginIndex:0, keylen);
        int curr = keylen;

        while (curr < data.length()) {
            if (part.charAt(index:0) == '1') {
                part = xor(key, part) + data.charAt(curr);
            } else {
                part = part.substring(beginIndex:1) + data.charAt(curr);
            }
            curr++;
        }

        if (part.charAt(index:0) == '1') {
            part = xor(key, part);
        } else {
            part = part.substring(beginIndex:1);
        }

        return part;
    }

    public static String flip(String data, int index) {
        char[] chars = data.toCharArray();

        if (chars[index] == '0')
            chars[index] = '1';
        else
            chars[index] = '0';
        return new String(chars);
    }
```

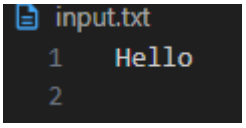
```java

    public static String simulateError(String data, String type) {
        Random rand = new Random();

        if (type.equals(anObject:"single")) {
            int index = rand.nextInt(data.length());
            System.out.println("Simulated single bit flip at position: " + index);
            return flip(data, index);
        } else if (type.equals(anObject:"burst")) {
            int burstLength = 4; // fixed length burst
            int start = rand.nextInt(data.length() - burstLength + 1);
            System.out.println("Simulated burst error from position: " + start + " to " + (start + burstLength - 1));
            for (int i = 0; i < burstLength; i++) {
                data = flip(data, start + i);
            }
            return data;
        }

        return data;
    }

    Run | Debug
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(port:5000)) {
            System.out.println(x:"Server is connected at port no: 5000");
            System.out.println(x:"Server is connecting");
            System.out.println(x:"Waiting for the client...");

            try (
                    Socket clientSocket = serverSocket.accept();
                    BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
                    Scanner sc = new Scanner(System.in)) {
                System.out.println("Client request accepted at port no: " + clientSocket.getPort());

                String received = reader.readLine();
                String generator = reader.readLine();

                System.out.println("Received Codeword: " + received);
                System.out.println("Using Generator Polynomial: " + generator);

                System.out.print(s:"Do you want to simulate error (single/burst/none): ");
                String type = sc.nextLine().toLowerCase();

                if (!type.equals(anObject:"none")) {
                    received = simulateError(received, type);
                }
```

.

```
            String remainder = mod2div(received, generator);
            System.out.println("Calculated Remainder: " + remainder);

            boolean hasError = false;
            for (char c : remainder.toCharArray()) {
                if (c == '1') {
                    hasError = true;
                    break;
                }
            }

            if (!hasError) {
                System.out.println(x:"No error detected in transmission.");
            } else {
                System.out.println(x:"Error detected in transmission!");

                int found = -1;// no index found

                for (int i = 0; i < received.length(); i++) {
                    String flipped = flip(received, i);
                    String testRemainder = mod2div(flipped, generator);

                    if (!testRemainder.contains(s:"1")) {
                        if (found == -1) {
                            found = i;
                            break; // first index found
                        }
                    }
                }

                if (found >= 0) {
                    String corrected = flip(received, found);
                    System.out.println("Single-bit error corrected at position: " + found);
                    System.out.println("Corrected Codeword: " + corrected);
                } else {
                    System.out.println(x:"No single-bit correction possible. Error uncorrectable.");
                }
            }
        }

        System.out.println(x:"Server connection closed.");

    } catch (IOException e) {
        System.err.println("Error: " + e.getMessage());
    }
```

5.Result Analysis:

Input File



Test Case 1 : CRC-8 with single bit error

Client Output:

```
PS C:\Users\rahma\OneDrive\Desktop\Telecom> java Client
Client connected to the server on Handshaking port 5000
Clientâ??s Communication Port: 58683
Client is connected
File content: Hello
Converted Binary Data: 0100100001100101011011000110110001101111
Enter CRC type (8 / 10 / 16 / 32): 8
After Appending zeros Data to Divide: 0100100001100101011011000110110001101111100000000
CRC Remainder: 11110110
Codeword to send: 0100100001100101011011000110110001101111111110110
```

Server Output

```
PS C:\Users\rahma\OneDrive\Desktop\Telecom> java Server
Server is connected at port no: 5000
Server is connecting
Waiting for the client...
Client request accepted at port no: 58683
Received Codeword: 0100100001100101011011000110110001101111111110110
Using Generator Polynomial: 100000111
Do you want to simulate error (single/burst/none): single
Simulated single bit flip at position: 41
Calculated Remainder: 01000000
Error detected in transmission!
Single-bit error corrected at position: 41
Corrected Codeword: 0100100001100101011011000110110001101111111110110
Server connection closed.
```

In this test case, the client sent the codeword generated using CRC-8 for the message "Hello". The server simulated a single-bit error by flipping a random bit at position 41. Upon receiving the corrupted codeword, the server performed

CRC division and detected a non-zero remainder (`01000000`), indicating an error.

The server then attempted single-bit error correction by flipping each bit one by one. It successfully identified and corrected the flipped bit at position 41. The corrected codeword matched the original, proving that CRC-8 was able to detect and correct a single-bit error in this case.

Test Case 2 : CRC-8 with multiple bit error

Server Output:

```
PS C:\Users\rahma\OneDrive\Desktop\Telecom> java Server
Server is connected at port no: 5000
Server is connecting
Waiting for the client...
Client request accepted at port no: 58805
Received Codeword: 0100100001100101011011000110110001101111111110110
Using Generator Polynomial: 100000111
Do you want to simulate error (single/burst/none): burst
Simulated burst error from position: 9 to 12
Calculated Remainder: 10000010
Error detected in transmission!
No single-bit correction possible. Error uncorrectable.
Server connection closed.
```

In this test case, the server received the codeword generated using CRC-8. A burst error was simulated by flipping 4 consecutive bits (from position 9 to 12). The server detected a non-zero CRC remainder (10000010), confirming that an error occurred during transmission.

However, since CRC is primarily designed for error detection, not correction of multiple-bit errors, the server could not identify any single-bit fix that resulted in a valid codeword. As a result, it reported the error as uncorrectable.

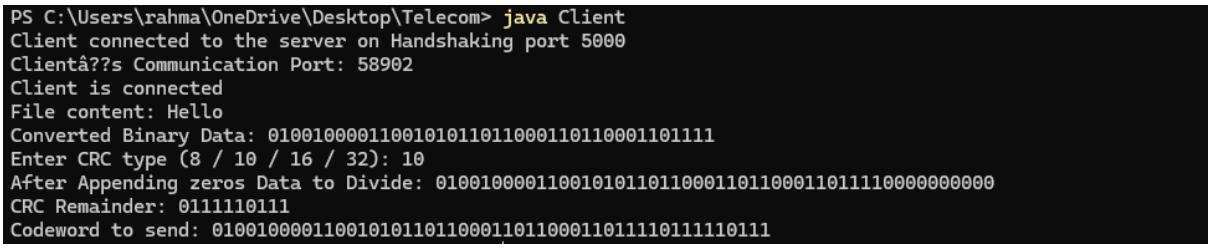
Test Case 3: CRC-8 with no bit error

```
PS C:\Users\rahma\OneDrive\Desktop\Telecom> java Server
Server is connected at port no: 5000
Server is connecting
Waiting for the client...
Client request accepted at port no: 58834
Received Codeword: 0100100001100101011011000110110001101111111110110
Using Generator Polynomial: 100000111
Do you want to simulate error (single/burst/none): none
Calculated Remainder: 00000000
No error detected in transmission.
Server connection closed.
```

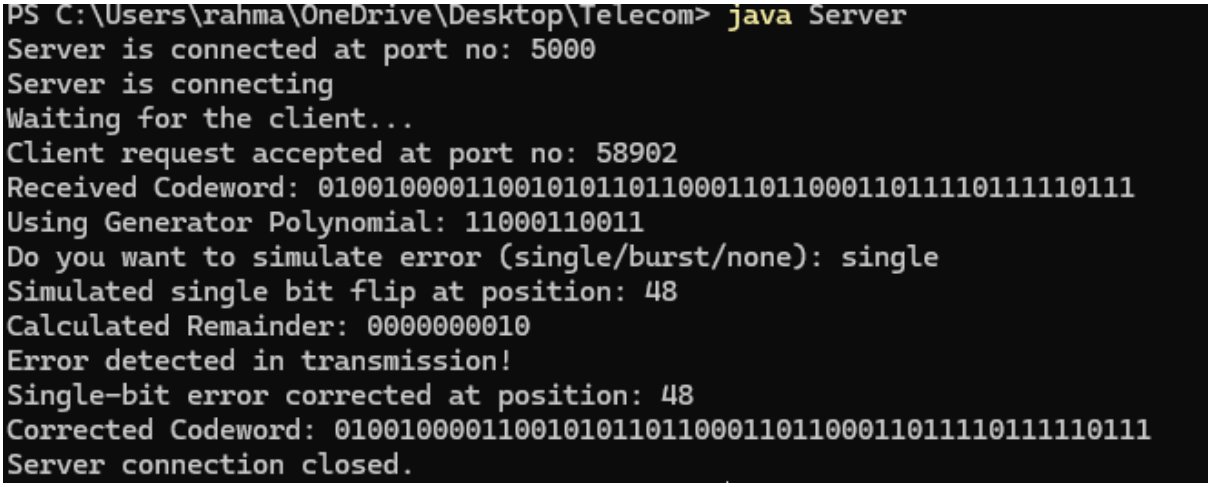
In this test, the server received the codeword along with the CRC-8 generator polynomial without simulating any error. Upon performing the modulo-2 division, the CRC remainder was `00000000`, indicating no error in the received data. Therefore, the transmission was considered valid and error-free.

Test Case 4: CRC-10 with single bit error

Client Side:



Server Side:



In this test case, the client sent the codeword generated using CRC-10 for the message "Hello". The server simulated a single-bit error by flipping a random bit at position 48. Upon receiving the corrupted codeword, the server performed CRC division and detected a non-zero remainder (`0000000010`), indicating an error.

The server then attempted single-bit error correction by flipping each bit one by one. It successfully identified and corrected the flipped bit at position 48. The corrected codeword matched the original, proving that CRC-10 was able to detect and correct a single-bit error in this case

Test Case 5 : CRC-10 with multiple bit error

Server Side:

```
PS C:\Users\rahma\OneDrive\Desktop\Telecom> java Server
Server is connected at port no: 5000
Server is connecting
Waiting for the client...
Client request accepted at port no: 58933
Received Codeword: 0100100001100101011011000110110001101111010111110111
Using Generator Polynomial: 11000110011
Do you want to simulate error (single/burst/none): burst
Simulated burst error from position: 27 to 30
Calculated Remainder: 0011100001
Error detected in transmission!
No single-bit correction possible. Error uncorrectable.
Server connection closed.
```

In this test case, the server received the codeword generated using CRC-10. A burst error was simulated by flipping 4 consecutive bits (from position 27 to 30). The server detected a non-zero CRC remainder (0011100001), confirming that an error occurred during transmission.

However, since CRC is primarily designed for error detection, not correction of multiple-bit errors, the server could not identify any single-bit fix that resulted in a valid codeword. As a result, it reported the error as uncorrectable.

Test Case 6: CRC-10 with no bit error

```
PS C:\Users\rahma\OneDrive\Desktop\Telecom> java Server
Server is connected at port no: 5000
Server is connecting
Waiting for the client...
Client request accepted at port no: 58969
Received Codeword: 0100100001100101011011000110110001101111010111110111
Using Generator Polynomial: 11000110011
Do you want to simulate error (single/burst/none): none
Calculated Remainder: 0000000000
No error detected in transmission.
Server connection closed.
```

In this test, the server received the codeword along with the CRC-10 generator polynomial without simulating any error. Upon performing the modulo-2 division, the CRC remainder was 0, indicating no error in the received data. Therefore, the transmission was considered valid and error-free.

Test Case 7: CRC-16 with single bit error

Client Side

```
PS C:\Users\rahma\OneDrive\Desktop\Telecom> java Client
Client connected to the server on Handshaking port 5000
Client Communication Port: 58996
Client is connected
File content: Hello
Converted Binary Data: 01001000011001010110110001101100011011111
Enter CRC type (8 / 10 / 16 / 32): 16
After Appending zeros Data to Divide: 0100100001100101011011000110110001101111000000000000000
CRC Remainder: 1011011111000110
Codeword to send: 0100100001100101011011000110110001101111101101111100011
```

Server Side

```
PS C:\Users\rahma\OneDrive\Desktop\Telecom> java Server
Server is connected at port no: 5000
Server is connecting
Waiting for the client...
Client request accepted at port no: 58996
Received Codeword: 0100100001100101011011000110110001101111101101111100011
Using Generator Polynomial: 11000000000000101
Do you want to simulate error (single/burst/none): single
Simulated single bit flip at position: 38
Calculated Remainder: 1000000000001111
Error detected in transmission!
Single-bit error corrected at position: 38
Corrected Codeword: 0100100001100101011011000110110001101111101101111100011
Server connection closed.
```

In this test case, the client sent the codeword generated using CRC-16 for the message "Hello". The server simulated a single-bit error by flipping a random bit at position 38. Upon receiving the corrupted codeword, the server performed CRC division and detected a non-zero remainder (1000000000001111), indicating an error.

The server then attempted single-bit error correction by flipping each bit one by one. It successfully identified and corrected the flipped bit at position 38. The corrected codeword matched the original, proving that CRC-16 was able to detect and correct a single-bit error in this case

Test Case 8 : CRC-16 with multiple bit error

Server Side:

```
PS C:\Users\rahma\OneDrive\Desktop\Telecom> java Server
Server is connected at port no: 5000
Server is connecting
Waiting for the client...
Client request accepted at port no: 59070
Received Codeword: 0100100001100101011011000110110001101111101101111l000110
Using Generator Polynomial: 11000000000000101
Do you want to simulate error (single/burst/none): burst
Simulated burst error from position: 21 to 24
Calculated Remainder: 0000000001100110
Error detected in transmission!
No single-bit correction possible. Error uncorrectable.
Server connection closed.
```

In this test case, the server received the codeword generated using CRC-16. A burst error was simulated by flipping 4 consecutive bits (from position 21 to 24). The server detected a non-zero CRC remainder, confirming that an error occurred during transmission.

However, since CRC is primarily designed for error detection, not correction of multiple-bit errors, the server could not identify any single-bit fix that resulted in a valid codeword. As a result, it reported the error as uncorrectable.

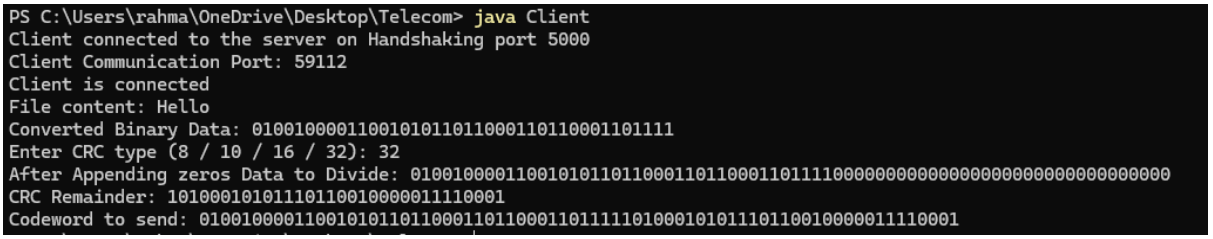
Test Case 9: CRC-16 with no bit error

```
PS C:\Users\rahma\OneDrive\Desktop\Telecom> java Server
Server is connected at port no: 5000
Server is connecting
Waiting for the client...
Client request accepted at port no: 59102
Received Codeword: 0100100001100101011011000110110001101111101101111l000110
Using Generator Polynomial: 11000000000000101
Do you want to simulate error (single/burst/none): none
Calculated Remainder: 0000000000000000
No error detected in transmission.
Server connection closed.
```

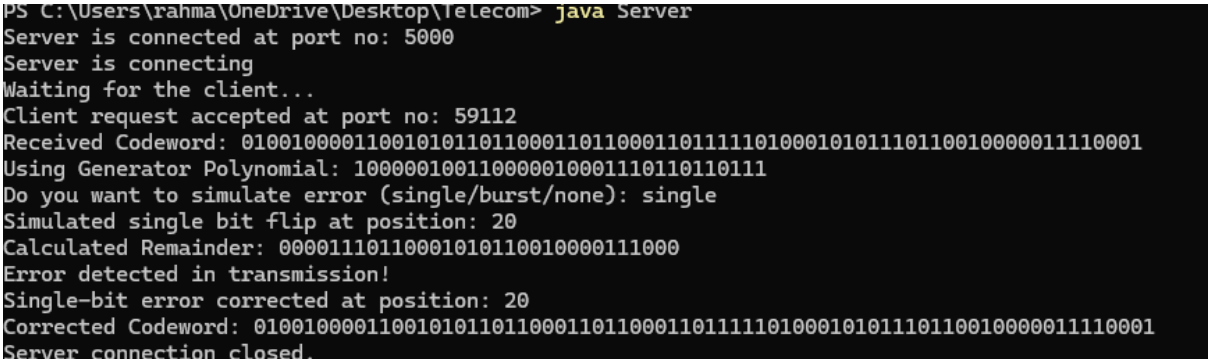
In this test, the server received the codeword along with the CRC-16 generator polynomial without simulating any error. Upon performing the modulo-2 division, the CRC remainder was 0, indicating no error in the received data. Therefore, the transmission was considered valid and error-free.

Test Case 10: CRC-32 with single bit error

Client Side:



Server Side:



In this test case, the client sent the codeword generated using CRC-32 for the message "Hello". The server simulated a single-bit error by flipping a random bit at position 20. Upon receiving the corrupted codeword, the server performed CRC division and detected a non-zero remainder , indicating an error.

The server then attempted single-bit error correction by flipping each bit one by one. It successfully identified and corrected the flipped bit at position 20. The corrected codeword matched the original, proving that CRC-32 was able to detect and correct a single-bit error in this case.

Test Case 11: CRC-32 with multiple bit error

Server Side:

```
PS C:\Users\rahma\OneDrive\Desktop\Telecom> java Server
Server is connected at port no: 5000
Server is connecting
Waiting for the client...
Client request accepted at port no: 59138
Received Codeword: 01001000011001010110110001101100011011111010001010111011001000001111000 1
Using Generator Polynomial: 10000010011000001000111011011 0111
Do you want to simulate error (single/burst/none): burst
Simulated burst error from position: 11 to 14
Calculated Remainder: 11001110000100011101000101110101
Error detected in transmission!
No single-bit correction possible. Error uncorrectable.
Server connection closed.
```

In this test case, the server received the codeword generated using CRC-32. A burst error was simulated by flipping 4 consecutive bits (from position 11 to 14). The server detected a non-zero CRC remainder, confirming that an error occurred during transmission.

However, since CRC is primarily designed for error detection, not correction of multiple-bit errors, the server could not identify any single-bit fix that resulted in a valid codeword. As a result, it reported the error as uncorrectable.

Test Case 12: CRC-32 with no bit error

Server Side:

```
PS C:\Users\rahma\OneDrive\Desktop\Telecom> java Server
Server is connected at port no: 5000
Server is connecting
Waiting for the client...
Client request accepted at port no: 59147
Received Codeword: 01001000011001010110110001101100011011111010001010111011001000001111000 1
Using Generator Polynomial: 10000010011000001000111011011 0111
Do you want to simulate error (single/burst/none): none
Calculated Remainder: 00000000000000000000000000000000
No error detected in transmission.
Server connection closed.
```

In this test, the server received the codeword along with the CRC-32 generator polynomial without simulating any error. Upon performing the modulo-2 division, the CRC remainder was 0, indicating no error in the received data. Therefore, the transmission was considered valid and error-free.

6.Discussion:

Cyclic Redundancy Check (CRC) is a widely used technique for error detection in digital communication. The effectiveness of CRC largely depends on the choice of the generator polynomial, which determines its ability to detect different types of errors.

Detection of Single-Bit Errors:

Most standard CRC polynomials (like CRC-8, CRC-10, CRC-16, etc.) are designed to detect all single-bit errors, as long as the polynomial has at least two non-zero terms. In our implementation, we successfully detected and corrected single-bit errors using a brute-force approach, where each bit was flipped one-by-one and checked until the correct one was found. This is only feasible for single-bit errors due to the low complexity of the process.

Detection of Multiple-Bit Errors:

CRC is also effective in detecting burst errors, especially when the burst length is less than or equal to the degree of the generator polynomial. In our simulation, we introduced burst errors by flipping four consecutive bits. The CRC remainder turned out non-zero, indicating an error which shows the CRC's capability to detect such cases. However, we could not correct them, which highlights one of the core limitations of CRC.

CRC was designed for multiple bit detection, not correction. The polynomial division process creates a checksum (remainder) that validates the integrity of the entire codeword. However, if multiple bits are flipped, the remainder changes in ways that do not clearly identify which bits were affected. In our implementation, we used a brute-force correction method for single-bit errors by trying all possible flips.However,It's hard to fix multiple-bit errors because there are too many possible changes, different errors can look the same, and CRC can't tell which bits are wrong.

Comparative Analysis of CRC Polynomials:

- CRC-8 can detect all single-bit errors and burst errors up to 8 bits. It's best for small data and where the chance of error is low.
- CRC-10 offers a balance between simplicity and better error detection than CRC-8. It is used in some communication systems (e.g., ATM networks). It can also detect burst errors up to 10 bits
- CRC-16 is stronger than CRC-8and CRC-10. It works better for longer messages and burst errors. It is commonly used in systems like USB.
- CRC-32 is very robust. It detects most types of common errors in long messages and is widely used in Ethernet, ZIP files, PNG images, etc.

Each polynomial's strength lies in its degree and structure, which influences the detection range. However, none of the methods can correct burst or multi-bit errors without external error-correction codes.

7.Learning and Difficulties:

During this lab, we learned how CRC detects errors in data transmission and how different generator polynomials affect its error detection ability. By implementing modulo-2 division and simulating bit flips, we better understood the error detection process.

The biggest challenge was implementing single-bit error correction by flipping bits one at a time and checking the CRC. We also studied why CRC can detect but cannot correct multiple-bit errors. It took some time for us to set up and debug the socket communication between client and server.

8.Conclusion:

In conclusion, CRC is a powerful and efficient method for detecting errors in digital communication. This experiment demonstrated how CRC detects both single-bit and burst errors effectively. While single-bit error correction is possible using brute-force flipping, multiple-bit error correction is not feasible with CRC alone.

Choosing the right generator polynomial is important for maximizing error detection based on message length and error types. Overall, this lab helped solidify the practical understanding of CRC and its limitations in error correction.