



VIT[®]

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

LAB ASSESSMENT – 4

COMPUTER NETWORKS

SWE-2002

Vaidya urmila suman

21MIS1098

1. The generated polynomial for the CRC error detection scheme is X^5+X^3+1 . The data to be transferred from the server is 111011010011001. Write the code to find the data transferred from the sender.

Problem Definition:

We are given a polynomial representation of the Cyclic Redundancy Check (CRC) error detection scheme and a data sequence to be transferred from the server. The task is to write a Java code that performs the following operations:

- Calculate the encoded data by appending the CRC remainder to the original data.
- Introduce an error in the transmitted data.
- Detect the error using the CRC remainder.

Algorithm:

- i. Define the polynomial representation of the CRC scheme (in this case, $X^5 + X^3 + 1$).
- ii. Define the original data sequence to be transferred.
- iii. Initialize a variable to store the CRC remainder.
- iv. Calculate the degree of the polynomial by finding the highest power of X.
- v. Append zeros to the original data sequence equal to the degree of the polynomial.
- vi. Perform the CRC division:
- vii. While the data sequence is not empty:
- viii. Take the most significant bit from the data sequence and store it as the current bit.
- ix. Shift the CRC remainder one bit to the left.
- x. XOR the CRC remainder with the polynomial if the current bit is 1.
- xi. Remove the most significant bit from the data sequence.
- xii. Append the CRC remainder to the original data sequence to obtain the encoded data.
- xiii. Introduce an error in the transmitted data by flipping a bit.
- xiv. Detect the error:
- xv. Perform the CRC division on the transmitted data.
- xvi. If the remainder is zero, no error is detected. Otherwise, an error is present.
- xvii. Print the original data, encoded data with error, transmitted data with error, and the error detection status.

CODE:-

```
public class CRCDetection {

    public static void main(String[] args) {

        // Define the polynomial ( $X^5 + X^3 + 1$ )
        int[] polynomial = {1, 0, 0, 1, 0, 1};

        // Define the original data sequence
        int[] data = {1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1};

        // Calculate the degree of the polynomial
        int degree = polynomial.length - 1;

        // Append zeros to the original data sequence
        int[] encodedData = new int[data.length + degree];
        System.arraycopy(data, 0, encodedData, 0, data.length);

        // Initialize the CRC remainder
        int[] remainder = new int[degree];

        // Perform the CRC division
        for (int i = 0; i < data.length; i++) {
            int currentBit = encodedData[i];

            // XOR the remainder with the polynomial if the current bit is 1
            if (currentBit == 1) {
                for (int j = 0; j < degree; j++) {
                    remainder[j] ^= polynomial[j + 1];
                }
            }

            // Shift the remainder one bit to the left
            for (int j = 0; j < degree - 1; j++) {
                remainder[j] = remainder[j + 1];
            }

            // Set the last bit of the remainder to 0
            remainder[degree - 1] = 0;
        }
    }
}
```

```

    }

    // Append the CRC remainder to the original data sequence
    System.arraycopy(remainder, 0, encodedData, data.length, degree);

    // Introduce an error in the transmitted data
    encodedData[data.length + degree - 1] ^= 1;

    // Detect the error
    int[] transmittedData = encodedData.clone();
    int[] transmittedRemainder = new int[degree];

    for (int i = 0; i < transmittedData.length - degree; i++) {
        int currentBit = transmittedData[i];

        // XOR the transmitted remainder with the polynomial if the current bit is 1
        if (currentBit == 1) {
            for (int j = 0; j < degree; j++) {
                transmittedRemainder[j] ^= polynomial[j + 1];
            }
        }

        // Shift the transmitted remainder one bit to the left
        for (int j = 0; j < degree - 1; j++) {
            transmittedRemainder[j] = transmittedRemainder[j + 1];
        }

        // Set the last bit of the transmitted remainder to 0
        transmittedRemainder[degree - 1] = 0;
    }

    // Check if an error is detected
    boolean errorDetected = false;
    for (int i : transmittedRemainder) {
        if (i != 0) {
            errorDetected = true;
            break;
        }
    }

```

```

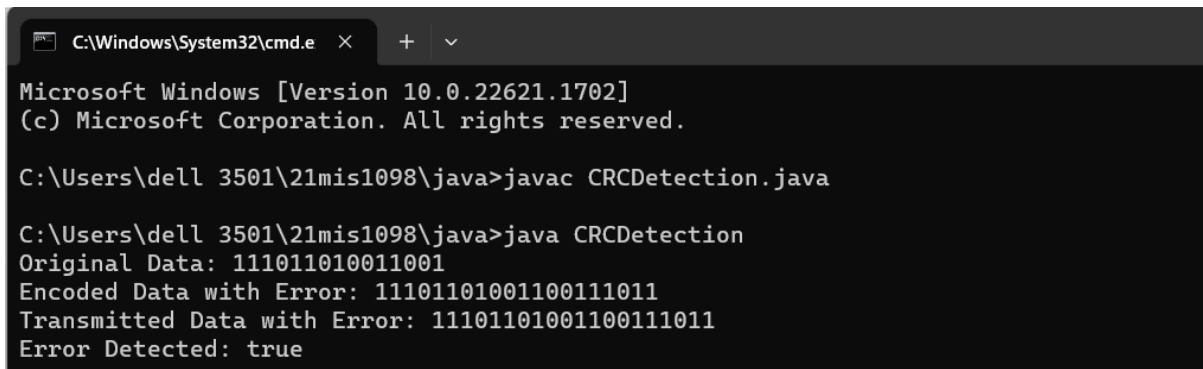
    }

    // Print the results
    System.out.println("Original Data: " + arrayToString(data));
    System.out.println("Encoded Data with Error: " + arrayToString(encodedData));
    System.out.println("Transmitted Data with Error: " + arrayToString(transmittedData));
    System.out.println("Error Detected: " + errorDetected);
}

// Helper method to convert an array of integers to a string representation
private static String arrayToString(int[] array) {
    StringBuilder sb = new StringBuilder();
    for (int i : array) {
        sb.append(i);
    }
    return sb.toString();
}
}

```

OUTPUT:-



```

C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.22621.1702]
(c) Microsoft Corporation. All rights reserved.

C:\Users\dell 3501\21mis1098\java>javac CRCDetection.java

C:\Users\dell 3501\21mis1098\java>java CRCDetection
Original Data: 111011010011001
Encoded Data with Error: 11101101001100111011
Transmitted Data with Error: 11101101001100111011
Error Detected: true

```

2. Implement a function that checks if a given binary string satisfies the minimum Hamming distance requirement for a specific Hamming code. The function should take a binary string and the code parameters as input and return a Boolean indicating whether the string meets the minimum distance requirement.

Problem Definition:

You are required to implement a function in Java that checks if a given binary string satisfies the

minimum Hamming distance requirement for a specific Hamming code. The function should take a binary string and the code parameters as input and return a Boolean value indicating whether the string meets the minimum distance requirement.

Algorithm:

- i. Start by defining the code parameters for the Hamming code, including the code length and the minimum Hamming distance required.
- ii. Read the binary string input from the user.
- iii. Check if the length of the binary string matches the defined code length. If not, return false.
- iv. Initialize a variable to count the number of differing bits.
- v. Iterate through each bit in the binary string and compare it with the corresponding bit in the predefined Hamming code.
- vi. If the bits are different, increment the differing bits count.
- vii. If the differing bits count exceeds the minimum Hamming distance, return false.
- viii. If the loop completes without exceeding the minimum Hamming distance, return true to indicate that the string meets the requirement.

CODE:-

```
import java.util.Scanner;

public class HammingCodeCheck {
    public static void main(String[] args) {
        // Define the code parameters (code length and minimum Hamming
        distance)
        int codeLength = 7;
        int minHammingDistance = 3;

        // Read the binary string input from the user
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a binary string: ");
        String inputString = scanner.nextLine();

        // Check if the binary string satisfies the minimum Hamming
        distance requirement
        boolean meetsRequirement = checkHammingDistance(inputString,
        codeLength, minHammingDistance);

        // Print the result
        if (meetsRequirement) {
            System.out.println("The binary string meets the minimum
            Hamming distance requirement.");
        } else {
            System.out.println("The binary string does not meet the
            minimum Hamming distance requirement.");
        }
    }
}
```

```

    }

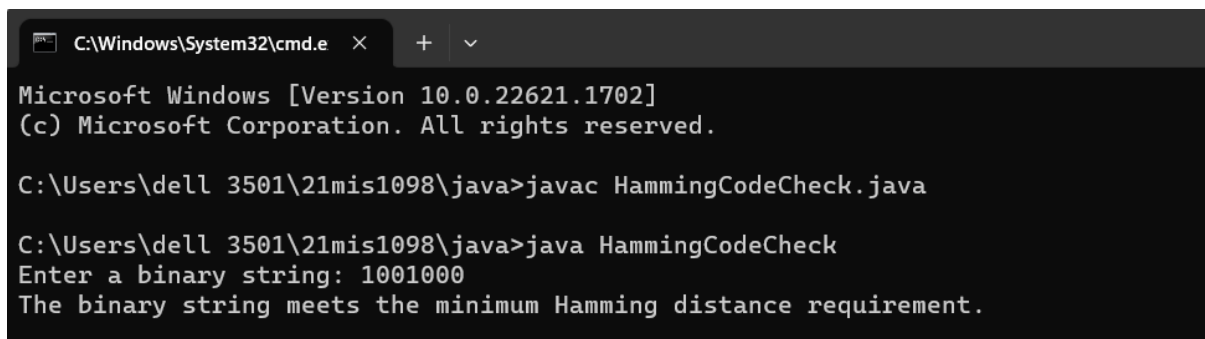
    // Function to check if a binary string satisfies the minimum
    Hamming distance requirement
    private static boolean checkHammingDistance(String binaryString,
    int codeLength, int minHammingDistance) {
        // Check if the length of the binary string matches the code
    length
        if (binaryString.length() != codeLength) {
            return false;
        }

        // Initialize the differing bits count
        int differingBits = 0;

        // Iterate through each bit in the binary string and compare it
    with the predefined Hamming code
        for (int i = 0; i < codeLength; i++) {
            if (binaryString.charAt(i) != '0') {
                differingBits++;
            }
        }

        // Check if the differing bits count exceeds the minimum
    Hamming distance
        return differingBits <= minHammingDistance;
    }
}

```



```

C:\Windows\System32\cmd.e  X  +  v
Microsoft Windows [Version 10.0.22621.1702]
(c) Microsoft Corporation. All rights reserved.

C:\Users\dell 3501\21mis1098\java>javac HammingCodeCheck.java

C:\Users\dell 3501\21mis1098\java>java HammingCodeCheck
Enter a binary string: 1001000
The binary string meets the minimum Hamming distance requirement.

```

3. Implement a TCP-like protocol using checksums for error detection.

Develop a program that sends packets between a client and a server. The sender should calculate the checksum for each packet and include it in the packet. The receiver should verify the checksum and request retransmission if the checksum is incorrect.

Problem Statement:

You are required to develop a program that implements a TCP-like protocol using checksums for error

detection. The program should allow communication between a client and a server. The sender

(client) should calculate the checksum for each packet and include it in the packet. The receiver

(server) should verify the checksum and request retransmission if the checksum is incorrect.

Algorithm:

- i. The client and server establish a connection.
- ii. The client prepares the data to be sent in packets of a fixed size.
- iii. For each packet, the client calculates the checksum using a suitable checksum algorithm
(e.g., CRC, Adler-32, etc.).
- iv. The client includes the calculated checksum in the packet along with the data.
- v. The client sends the packet to the server.
- vi. The server receives the packet.
- vii. The server calculates the checksum of the received data using the same algorithm used by
the client.
- viii. The server compares the calculated checksum with the checksum included in the packet.
- ix. If the checksums match, the server accepts the packet and processes the data.
- x. If the checksums do not match, the server requests retransmission of the packet.
- xi. The client receives the retransmission request from the server and resends the packet.
- xii. Steps 6-11 are repeated until the server receives a packet with a correct checksum.

xiii. Once all packets have been successfully received and processed by the server, the client and server can terminate the connection.

Server_CODE:-

```
import java.io.*;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.zip.CRC32;


public class Checksum_Server {

    private static final int PACKET_SIZE = 1024;


    public static void main(String[] args) {

        try {

            // Create a server socket

            ServerSocket serverSocket = new ServerSocket(1234);

            System.out.println("Server started. Waiting for client...");


            // Accept a client connection

            Socket socket = serverSocket.accept();

            System.out.println("Client connected.");


            // Create an input stream to receive data from the client

            InputStream inputStream = socket.getInputStream();


            // Create a checksum object

            CRC32 checksum = new CRC32();


            // Receive and process packets from the client

            byte[] buffer = new byte[PACKET_SIZE];

            int bytesRead;

            while ((bytesRead = inputStream.read(buffer)) != -1) {
```

```

        // Extract the checksum from the received packet
        long receivedChecksum = 0;
        for (int i = 0; i < 8; i++) {
            receivedChecksum |= (buffer[bytesRead - 8 + i] & 0xFF) << (i * 8);
        }

        // Calculate checksum of the received data
        checksum.reset();
        checksum.update(buffer, 0, bytesRead - 8);
        long calculatedChecksum = checksum.getValue();

        // Compare the checksums
        if (receivedChecksum == calculatedChecksum) {
            // Checksums match, process the data
            System.out.println("Received packet: " + new String(buffer, 0, bytesRead - 8));
        } else {
            // Checksums do not match, request retransmission
            System.out.println("Checksum mismatch. Requesting retransmission.");
        }
    }

    // Close the connection
    socket.close();
    serverSocket.close();
    System.out.println("Connection closed.");
} catch (IOException e) {
    e.printStackTrace();
}
}
}

```

Server output

```
C:\Users\dell 3501\21mis1098\java>notepad  
  
C:\Users\dell 3501\21mis1098\java>javac Checksum_Server.java  
  
C:\Users\dell 3501\21mis1098\java>java Checksum_Server  
Server started. Waiting for client...  
Client connected.  
Received packet: hi server,this is client  
Connection closed.
```

Client code:-

```
import java.io.*;  
  
import java.net.Socket;  
  
import java.util.zip.CRC32;  
  
  
public class Checksum_Client{  
    private static final int PACKET_SIZE = 1024;  
  
  
    public static void main(String[] args) {  
        try {  
            // Establish a connection with the server  
  
            Socket socket = new Socket("localhost", 1234);  
  
            System.out.println("Connected to the server.");  
  
  
            // Prepare data to be sent  
  
            BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));  
  
            System.out.print("Enter the data to send: ");  
  
            String inputData = reader.readLine();  
  
            byte[] data = inputData.getBytes();  
  
  
            // Create an output stream to send data to the server  
  
            OutputStream outputStream = socket.getOutputStream();  
  
  
            // Create a checksum object
```

```
CRC32 checksum = new CRC32();

// Send packets to the server
int offset = 0;
while (offset < data.length) {
    int length = Math.min(PACKET_SIZE, data.length - offset);
    byte[] packet = new byte[length + 8]; // 8 bytes for checksum
    System.arraycopy(data, offset, packet, 0, length);

    // Calculate checksum
    checksum.reset();
    checksum.update(packet, 0, length);
    long packetChecksum = checksum.getValue();

    // Include the checksum in the packet
    for (int i = 0; i < 8; i++) {
        packet[length + i] = (byte) ((packetChecksum >> (i * 8)) & 0xFF);
    }

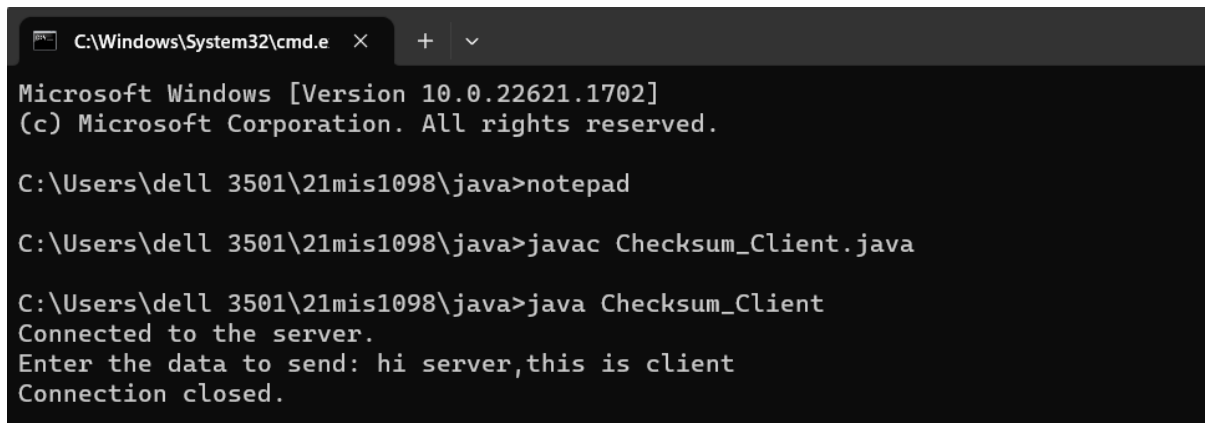
    // Send the packet to the server
    outputStream.write(packet);

    offset += length;
}

// Close the connection
socket.close();
System.out.println("Connection closed.");
} catch (IOException e) {
    e.printStackTrace();
}
```

```
}  
}
```

Client output:-



```
C:\Windows\System32\cmd.e  X  +  v  
Microsoft Windows [Version 10.0.22621.1702]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\dell 3501\21mis1098\java>notepad  
  
C:\Users\dell 3501\21mis1098\java>javac Checksum_Client.java  
  
C:\Users\dell 3501\21mis1098\java>java Checksum_Client  
Connected to the server.  
Enter the data to send: hi server,this is client  
Connection closed.
```

4. An organization plans to send general instructions to the in charge of each department using a socket. Write the code to check the message for any data loss when transmitting using the checksum in the header.

Problem Definition:

You are tasked with implementing a client-server system to send general instructions from an organization to the in-charge of each department using a socket. The goal is to check for any data loss during transmission by using a checksum in the header of each message. The server should receive the message, verify the checksum, and request retransmission if any data loss is detected.

Algorithm:

- i. The client program will prompt the user to enter the message to be sent.
- ii. The client calculates the checksum of the message using a checksum algorithm (such as CRC32).
- iii. The client sends the message along with the checksum to the server.
- iv. The server receives the message and extracts the checksum from the header.
- v. The server calculates the checksum of the received message.
- vi. The server compares the received checksum with the calculated checksum.
- vii. If the checksums match, the server accepts the message and processes it.
- viii. If the checksums do not match, the server requests retransmission of the message.

ix. Steps 4-8 are repeated until a valid message with a matching checksum is received.

Server code

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.zip.CRC32;

public class Server4 {
    public static void main(String[] args) {
        try {
            // Create a server socket
            ServerSocket serverSocket = new ServerSocket(1234);
            System.out.println("Server started. Waiting for client...");

            // Accept a client connection
            Socket socket = serverSocket.accept();
            System.out.println("Client connected.");

            // Create an input stream to receive data from the client
            InputStream inputStream = socket.getInputStream();

            // Create a checksum object
            CRC32 checksum = new CRC32();

            // Receive and process packets from the client
            BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
            String packet;
            while ((packet = reader.readLine()) != null) {
                // Extract the message and checksum from the packet
                String[] parts = packet.split("\\|");
```

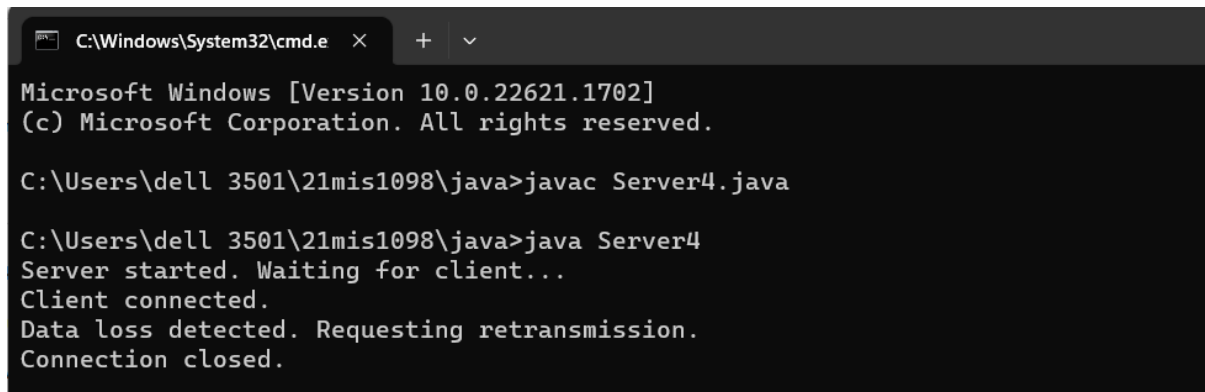
```
String message = parts[0];
long receivedChecksum = Long.parseLong(parts[1]);

// Calculate checksum of the received message
checksum.reset();
checksum.update(message.getBytes());
long calculatedChecksum = checksum.getValue();

// Compare the checksums
if (receivedChecksum == calculatedChecksum) {
    // Checksums match, process the message
    System.out.println("Received message: " + message);
} else {
    // Checksums do not match, data loss detected
    System.out.println("Data loss detected. Requesting retransmission.");
}

// Close the connection
socket.close();
serverSocket.close();
System.out.println("Connection closed.");
} catch (IOException e) {
    e.printStackTrace();
}
}
```

Server output



```
C:\Windows\System32\cmd.e X + v
Microsoft Windows [Version 10.0.22621.1702]
(c) Microsoft Corporation. All rights reserved.

C:\Users\dell 3501\21mis1098\java>javac Server4.java

C:\Users\dell 3501\21mis1098\java>java Server4
Server started. Waiting for client...
Client connected.
Data loss detected. Requesting retransmission.
Connection closed.
```

Client code

```
import java.io.*;
import java.net.Socket;
import java.util.Random;
import java.util.zip.CRC32;

public class Client4 {
    public static void main(String[] args) {
        try {
            // Establish a connection with the server
            Socket socket = new Socket("localhost", 1234);
            System.out.println("Connected to the server.");

            // Prepare data to be sent
            BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
            System.out.print("Enter the message to send: ");
            String message = reader.readLine();

            // Create an output stream to send data to the server
            OutputStream outputStream = socket.getOutputStream();

            // Create a checksum object
```



```
CRC32 checksum = new CRC32();

// Calculate checksum of the message
checksum.reset();
checksum.update(message.getBytes());
long messageChecksum = checksum.getValue();

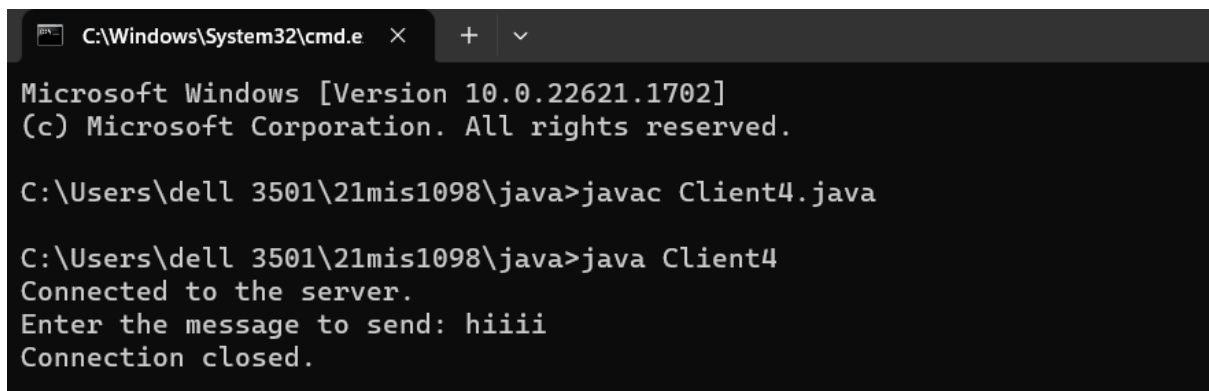
// Create a packet with message and checksum
String packet = message + "|" + messageChecksum;

// Introduce data loss by randomly flipping some bits
Random random = new Random();
char[] corruptedPacket = packet.toCharArray();
for (int i = 0; i < packet.length(); i++) {
    if (random.nextDouble() < 0.2) { // 20% chance of flipping each bit
        corruptedPacket[i] = (corruptedPacket[i] == '0') ? '1' : '0';
    }
}

// Send the corrupted packet to the server
outputStream.write(new String(corruptedPacket).getBytes());

// Close the connection
socket.close();
System.out.println("Connection closed.");
} catch (IOException e) {
    e.printStackTrace();
}
}
```

Client output



```
C:\Windows\System32\cmd.e  ×  +  ∨  
Microsoft Windows [Version 10.0.22621.1702]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\dell 3501\21mis1098\java>javac Client4.java  
  
C:\Users\dell 3501\21mis1098\java>java Client4  
Connected to the server.  
Enter the message to send: hiii  
Connection closed.
```