Implementation of Bit Stuffing and De-Stuffing using Java Socket Programming

April 24, 2025

Objective

To implement bit stuffing and de-stuffing using Java Socket Programming to simulate sender and receiver communication over a network.

Theory

Bit Stuffing

Bit stuffing is the process of inserting non-information bits into data to break up bit patterns in the data that might otherwise be interpreted as control information. It is often used in protocols like HDLC.

Rule: After every 5 consecutive 1s in the data, insert a 0.

De-Stuffing

De-stuffing is the reverse process, where the inserted 0s are removed on the receiving side after detecting 5 consecutive 1s.

Java Implementation

We simulate two systems:

- **Sender (Client)**: Applies bit stuffing and sends the frame.
- **Receiver (Server)**: Receives the stuffed frame and applies de-stuffing.

Sender.java (Client)

```
1 import java.io.*;
2 import java.net.*;
3 import java.util.Scanner;
4
5 public class Sender {
6     public static String bitStuff(String data) {
```

```
        StringBuilder stuffed = new StringBuilder();
        int count = 0;
        for (char bit : data.toCharArray()) {
            if (bit == '1') {
                count++;
                stuffed.append('1');
                if (count == 5) {
                    stuffed.append('0');
                    count = 0;
                }
            } else {
                stuffed.append('0');
                count = 0;
            }
        }
        return stuffed.toString();
    }

    public static void main(String[] args) throws IOException {
        Socket s = new Socket("localhost", 6666);
        DataOutputStream dout = new DataOutputStream(s.getOutputStream
            ());

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter binary data: ");
        String data = sc.nextLine();

        String stuffed = bitStuff(data);
        System.out.println("Stuffed Data: " + stuffed);

        dout.writeUTF(stuffed);
        dout.flush();
        dout.close();
        s.close();
    }
}
```

Listing 1: Sender.java

Receiver.java (Server)

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

public class Receiver {
    public static String bitDeStuff(String data) {
        StringBuilder destuffed = new StringBuilder();
        int count = 0;
        for (int i = 0; i < data.length(); i++) {
            char bit = data.charAt(i);
            if (bit == '1') {
                count++;
                destuffed.append('1');
            } else {
                if (count == 5) {
                    // skip stuffed '0'
```

```
                    count = 0;
                    continue;
                } else {
                    destuffed.append('0');
                    count = 0;
                }
            }
        }
        return destuffed.toString();
    }

    public static void main(String[] args) throws IOException {
        ServerSocket ss = new ServerSocket(6666);
        Socket s = ss.accept();
        DataInputStream dis = new DataInputStream(s.getInputStream());

        String stuffed = dis.readUTF();
        System.out.println("Received Stuffed Data: " + stuffed);

        String original = bitDeStuff(stuffed);
        System.out.println("De-Stuffed Data: " + original);

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

Listing 2: Receiver.java

Sample Input and Output

- **Input (Sender)**: 0111111011
- **Stuffed (Sender)**: 01111101011
- **Received (Receiver)**: 01111101011
- **De-Stuffed (Receiver)**: 0111111011

Conclusion

This implementation demonstrates how bit stuffing and de-stuffing techniques can be used in network communication, ensuring synchronization and frame boundary management.