

**University Of Dhaka  
Department of Computer Science & Engineering**

CSE-2213: Data and Telecommunication Lab

2nd Year 2nd Semester

Session: 2022-2023

**Exp No. 3:** Implementation of CRC for Error Detection and Single-Bit Error Correction

**Submitted by:**

1. Farhan Bin Rabbani (Roll: FH-35)
2. Tamal Kanti Sarker (Roll: JN-39)

**Submitted to:**

1. Dr. Md. Mustafizur Rahman, Professor, Dept. of CSE, DU
2. Mr. Palash Roy, Lecturer, Dept. of CSE, DU

**Lab Report 3**

**Lab Report Name:**  Implementation of CRC for Error Detection and Single-Bit Error Correction

**Introduction:** Cyclic Redundancy Check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. CRC uses a generator polynomial, available on both the sender and receiver sides. It treats the data as a binary polynomial and divides it by a generator polynomial. The remainder of this division is called the CRC code, which is appended to the data before transmission. After transmission, if the binary XOR division of the transmitted data and the generator polynomial yields the remainder zero, then there’s no error. Any other remainder suggests that an error is present. CRC also helps us locate and fix single-bit errors through Hamming Distance techniques. However, the same cannot be said for multiple-bit errors.

**Objectives:**

• To understand the theory and operation of CRC (Cyclic Redundancy Check).

• To implement a CRC encoding and decoding mechanism using polynomial division.

• To simulate data transmission, detect errors using CRC, and apply Hamming Distance for locating single-bit error positions.

• To evaluate CRC’s effectiveness in ensuring data integrity.

**Algorithms/Pseudocode:**

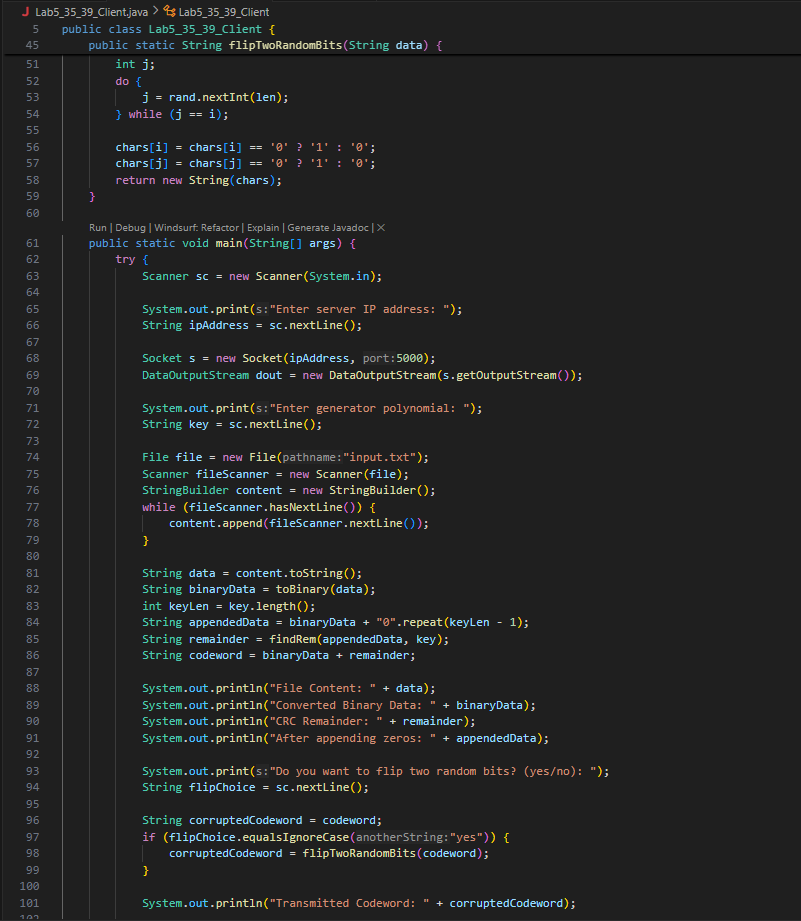
**Client-Side Algorithm**

* + We create a Socket object that takes the IP address and port number as input. We also create an object of the DataOutputStream class, which is used to send data to the server side.
  + We open and read content from a file (e.g., input.txt). The text file contains ASCII character data.
  + We write a function toBinary() to convert the character string to a binary representation (ASCII to 8-bit binary).
  + We input the generator polynomial (a binary string). We also append (k − 1) zeros to the original binary data, where k is the length of the generator.
  + We write a findRem() function to perform modulo-2 division on the appended data using the generator polynomial (XOR-based bitwise division performed by another function named xor()).
  + After obtaining the remainder, which is the CRC code, we append the CRC code to the original binary data to form the **codeword**.
  + We ask the user if they want to simulate errors. If “yes”, we use a flipTwoRandomBits() function to flip two random bits in the codeword and store it as the corrupted codeword.
  + We send the **original codeword, corrupted codeword,** and **generator polynomial** to the server through the socket.

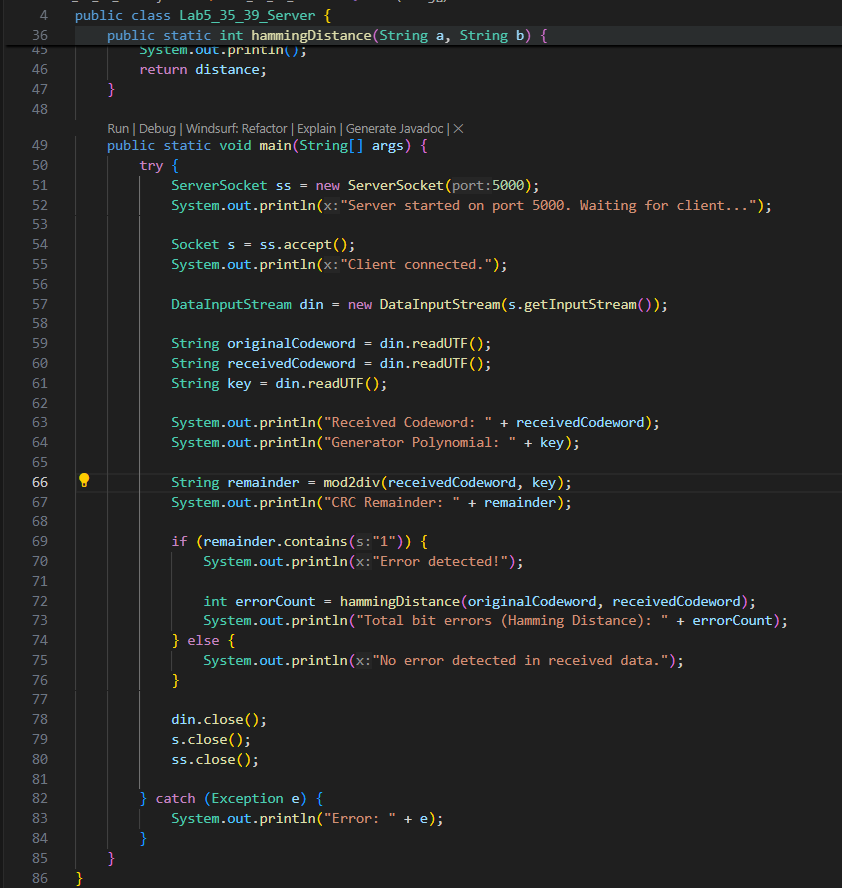
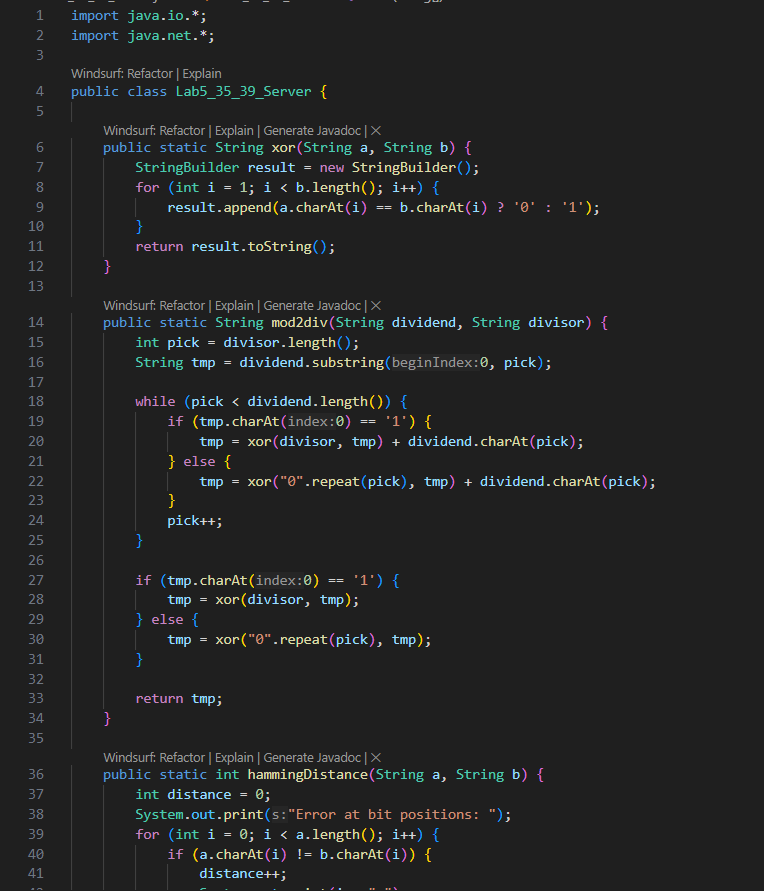
**Server-Side Algorithm**

* We create a ServerSocket that listens on port 5000 and waits for a connection request from the client. Once connected, we accept the request using the accept() method and establish communication.
* We create a DataInputStream object to receive the original codeword, the received codeword (possibly corrupted), and the generator polynomial from the client.
* We implement a mod2div() function to perform XOR-based CRC checking by dividing the received codeword using the provided generator polynomial.
* If the remainder contains any ‘1’, we conclude that an error occurred during transmission; otherwise, the data is considered error-free.
* If an error is detected, we compute the Hamming distance between the original and received codewords by comparing them bit by bit.
* We count and display the number of mismatched bits (i.e., bit errors) and print their positions.

**Implementation:**

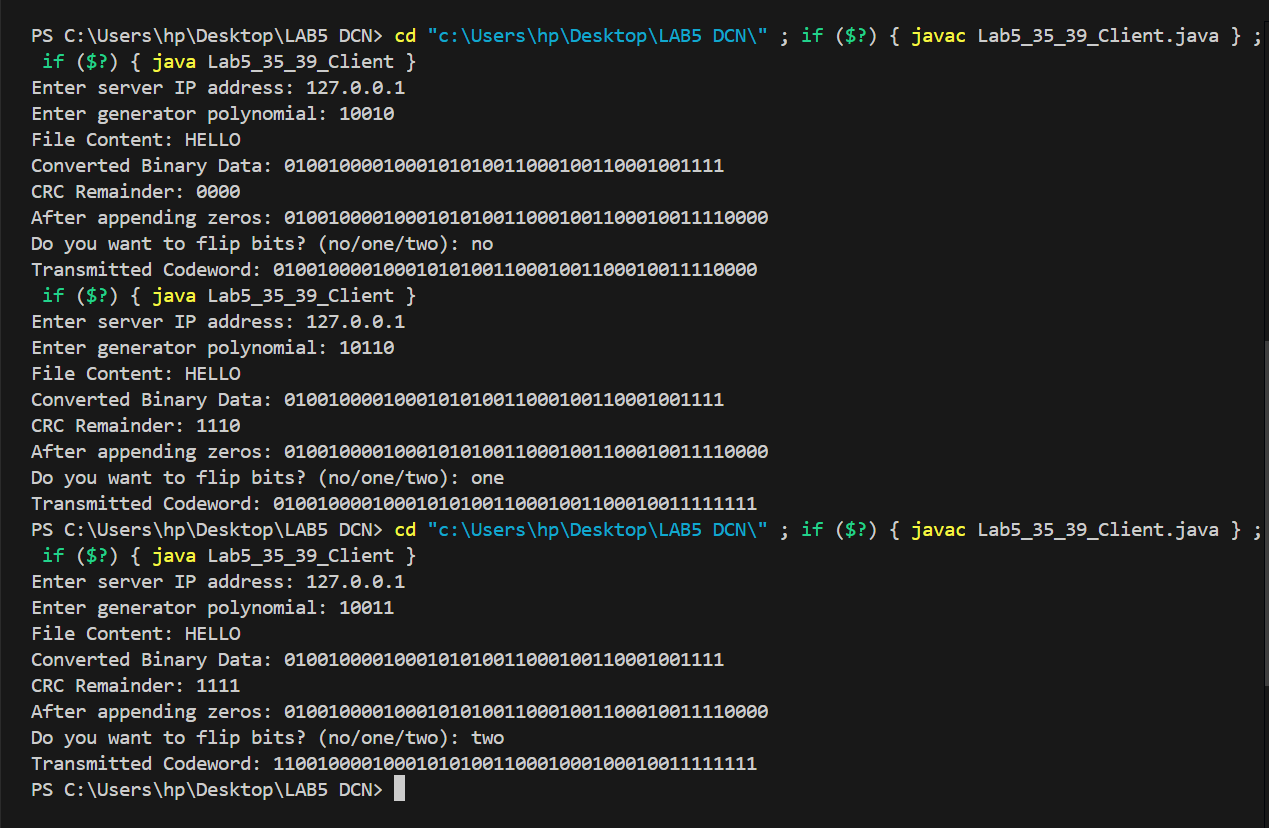
****

**Fig:** Client-Side

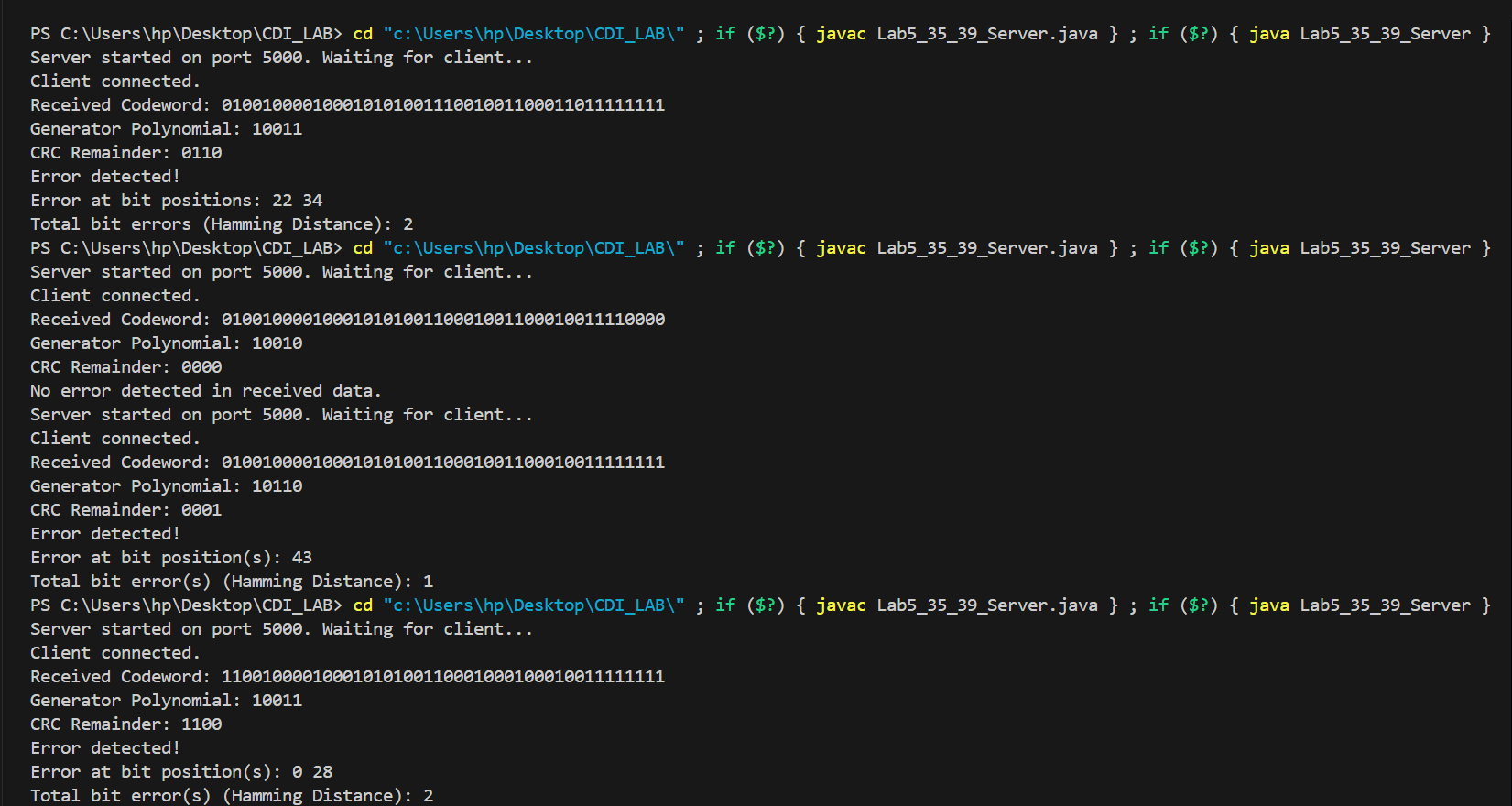
****

**Fig:** Server-Side

**Result Analysis:**

****

**Fig:** Client-Side



**Fig:** Server-Side

**Discussion:**

CRC is highly effective in detecting single-bit and burst errors, but it does not offer built-in correction for multiple-bit errors. Its reliability largely depends on the choice of the generator polynomial:

* A well-selected polynomial helps detect the most common types of transmission errors.
* If the polynomial includes the factor (x + 1), CRC can always detect any odd number of bit errors.
* It can also detect burst errors up to the length of the generator polynomial.

However, multiple-bit errors may still go undetected or be misinterpreted without additional mechanisms.  
By combining CRC with Hamming distance, the system can detect andlocalize errors, but correction is still limited to single-bit errors only.

**Learning and Difficulties:**

 Bitwise XOR division must be correctly implemented, otherwise the CRC code resulting from that division will be wrong.

 The function used in the server-side code that flips random bits for error injection should be properly implemented.

 Correct conversion from text to binary is essential; any mistake in binary representation will lead to incorrect CRC calculations.

 When comparing codewords using Hamming distance, both strings must be of equal length; otherwise, the comparison may result in runtime errors or inaccurate error detection.

**Conclusion:**

We were successful in the implementation of CRC for Error Detection and Single-Bit Error Correction.