# Bansilal Ramnath Agarwal Charitable Trust’s

Vishwakarma Institute of Technology, Pune-37

*(Autonomous Institute of Savitribai Phule Pune University)*



**Department of Computer Engineering**

|  |  |
| --- | --- |
| **Division** | **CS TY B** |
| **Batch** | **3** |
| **GR no.** | **12320165** |
| **Roll no.** | **83** |
| **Name** | **Komal Mahadev Potdar** |

**Assignment No. 3**

**Title:** Write a program for error detection and correction for 7/8 bits ASCII codes using Hamming Codes or CRC. Demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer to peer mode. Further extend it to real implementation of CRC over Ethernet standard.

**CRC**

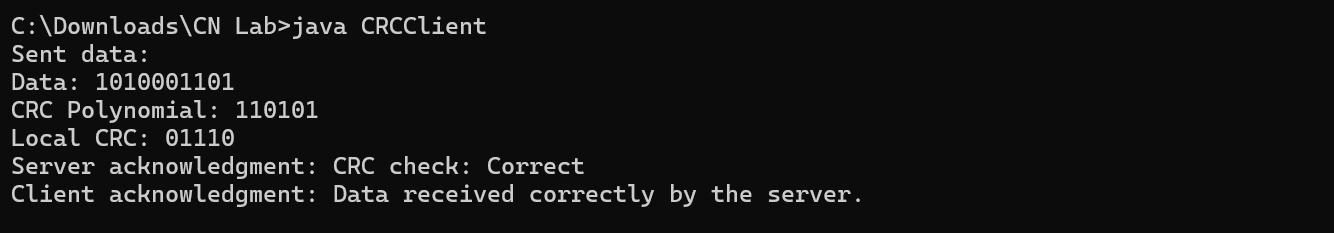
**Client**

|  |
| --- |
| import java.io.\*;  import java.net.\*;  public class CRCClient {      public static void main(String[] args) {          String hostname = "127.0.0.1"; // Use localhost or IP address for testing          int port = 12346; // Ensure this port matches with the receiver's port          try (Socket socket = new Socket(hostname, port)) {              OutputStream output = socket.getOutputStream();              PrintWriter writer = new PrintWriter(output, true);              InputStream input = socket.getInputStream();              BufferedReader reader = new BufferedReader(new InputStreamReader(input));              // Define the data and CRC polynomial              String data = "1010001101";              String crcPolynomial = "110101";                // Calculate CRC locally              String localCRC = calculateCRC(data, crcPolynomial);              System.out.println("Sent data:");              System.out.println("Data: " + data);              System.out.println("CRC Polynomial: " + crcPolynomial);              System.out.println("Local CRC: " + localCRC);                // Send data, CRC polynomial, and the calculated CRC to the receiver              writer.println(data);              writer.println(crcPolynomial);              writer.println(localCRC);              // Print the server's acknowledgment response              String response = reader.readLine();              System.out.println("Server acknowledgment: " + response);              // Client prints the acknowledgment              if ("CRC check: Correct".equals(response)) {                  System.out.println("Client acknowledgment: Data received correctly by the server.");              } else {                  System.out.println("Client acknowledgment: Data received incorrectly by the server.");              }          } catch (UnknownHostException ex) {              System.out.println("Server not found: " + ex.getMessage());          } catch (IOException ex) {              System.out.println("I/O error: " + ex.getMessage());          }      }      public static String calculateCRC(String data, String crcPolynomial) {          int dataLength = data.length();          int polynomialLength = crcPolynomial.length();          StringBuilder paddedData = new StringBuilder(data);          for (int i = 0; i < polynomialLength - 1; i++) {              paddedData.append('0');          }          String current = paddedData.substring(0, polynomialLength);          for (int i = polynomialLength; i <= paddedData.length(); i++) {              if (current.charAt(0) == '1') {                  current = xor(current, crcPolynomial);              } else {                  current = xor(current, "000000".substring(0, polynomialLength));              }              if (i < paddedData.length()) {                  current = current.substring(1) + paddedData.charAt(i);              } else {                  current = current.substring(1);              }          }          return current;      }      public static String xor(String a, String b) {          StringBuilder result = new StringBuilder();          for (int i = 0; i < a.length(); i++) {              result.append(a.charAt(i) == b.charAt(i) ? '0' : '1');          }          return result.toString();      }  } |

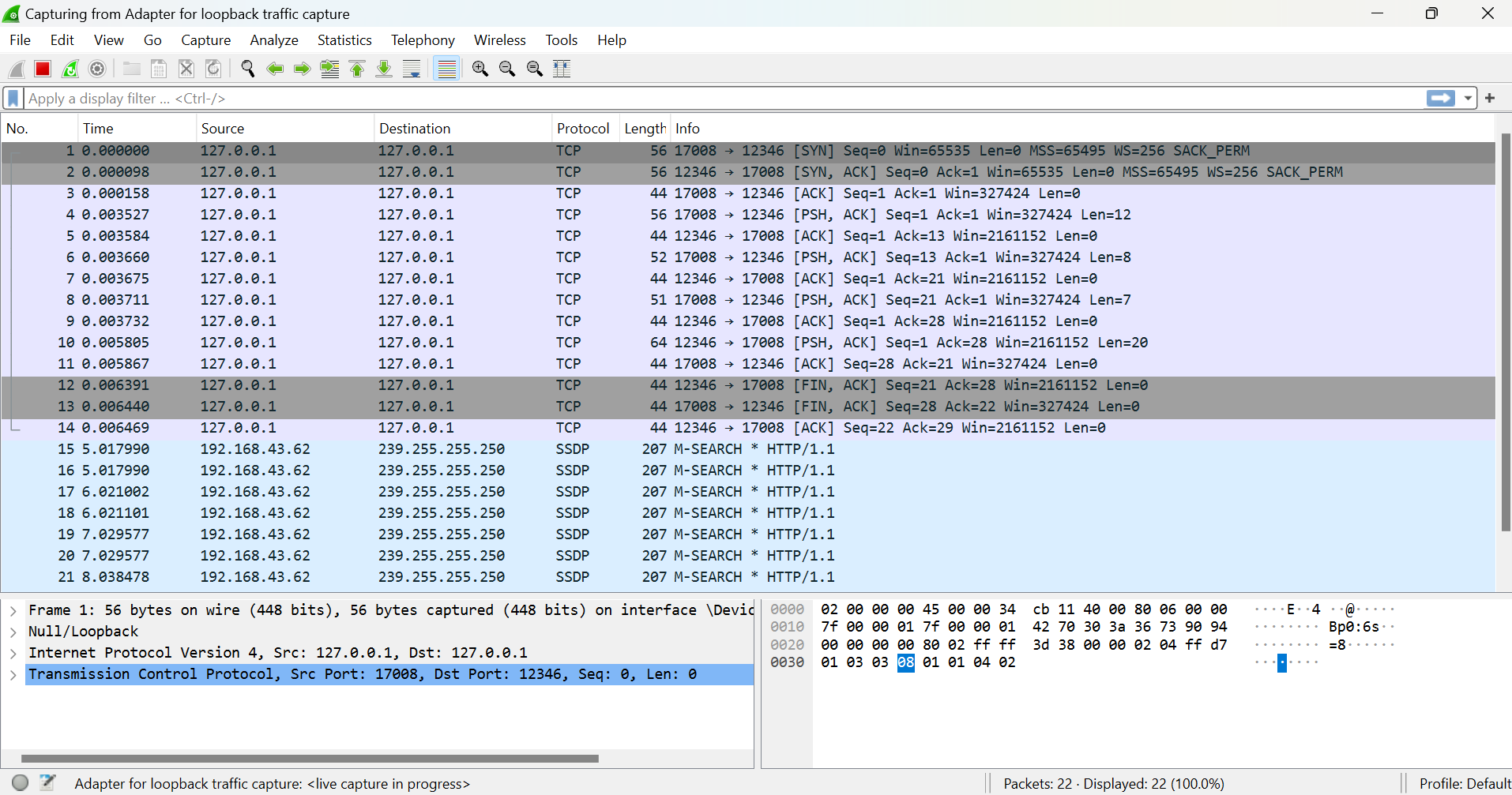
**Server**

|  |
| --- |
| import java.io.\*;  import java.net.\*;  public class CRCServer {      public static void main(String[] args) {          int port = 12346; // Ensure this port matches with the sender's port          try (ServerSocket serverSocket = new ServerSocket(port)) {              System.out.println("Receiver is listening on port " + port);              while (true) {                  Socket socket = serverSocket.accept();                  System.out.println("Sender connected");                  new ServerThread(socket).start();              }          } catch (IOException ex) {              System.out.println("Server exception: " + ex.getMessage());              ex.printStackTrace();          }      }  }  class ServerThread extends Thread {      private Socket socket;      public ServerThread(Socket socket) {          this.socket = socket;      }      public void run() {          try (InputStream input = socket.getInputStream();               BufferedReader reader = new BufferedReader(new InputStreamReader(input));               OutputStream output = socket.getOutputStream();               PrintWriter writer = new PrintWriter(output, true)) {              // Read the data and CRC polynomial from the sender              String data = reader.readLine();              String crcPolynomial = reader.readLine();              // Calculate CRC              String receivedCRC = reader.readLine();              String calculatedCRC = calculateCRC(data, crcPolynomial);              // Server prints the received data and CRC              System.out.println("Received data:");              System.out.println("Data: " + data);              System.out.println("CRC Polynomial: " + crcPolynomial);              System.out.println("Received CRC: " + receivedCRC);              System.out.println("Calculated CRC: " + calculatedCRC);              // Send back acknowledgment if data is correct or not              String acknowledgment;              if (calculatedCRC.equals(receivedCRC)) {                  acknowledgment = "CRC check: Correct";              } else {                  acknowledgment = "CRC check: Incorrect";              }              writer.println(acknowledgment);              // Server prints the acknowledgment              System.out.println("Acknowledgment sent to client: " + acknowledgment);          } catch (IOException ex) {              System.out.println("Server exception: " + ex.getMessage());              ex.printStackTrace();          }      }      public static String calculateCRC(String data, String crcPolynomial) {          int dataLength = data.length();          int polynomialLength = crcPolynomial.length();          StringBuilder paddedData = new StringBuilder(data);          for (int i = 0; i < polynomialLength - 1; i++) {              paddedData.append('0');          }          String current = paddedData.substring(0, polynomialLength);          for (int i = polynomialLength; i <= paddedData.length(); i++) {              if (current.charAt(0) == '1') {                  current = xor(current, crcPolynomial);              } else {                  current = xor(current, "000000".substring(0, polynomialLength));              }              if (i < paddedData.length()) {                  current = current.substring(1) + paddedData.charAt(i);              } else {                  current = current.substring(1);              }          }          return current;      }      public static String xor(String a, String b) {          StringBuilder result = new StringBuilder();          for (int i = 0; i < a.length(); i++) {              result.append(a.charAt(i) == b.charAt(i) ? '0' : '1');          }          return result.toString();      }  } |

**Output:**

****

****

****

**Hamming Code**

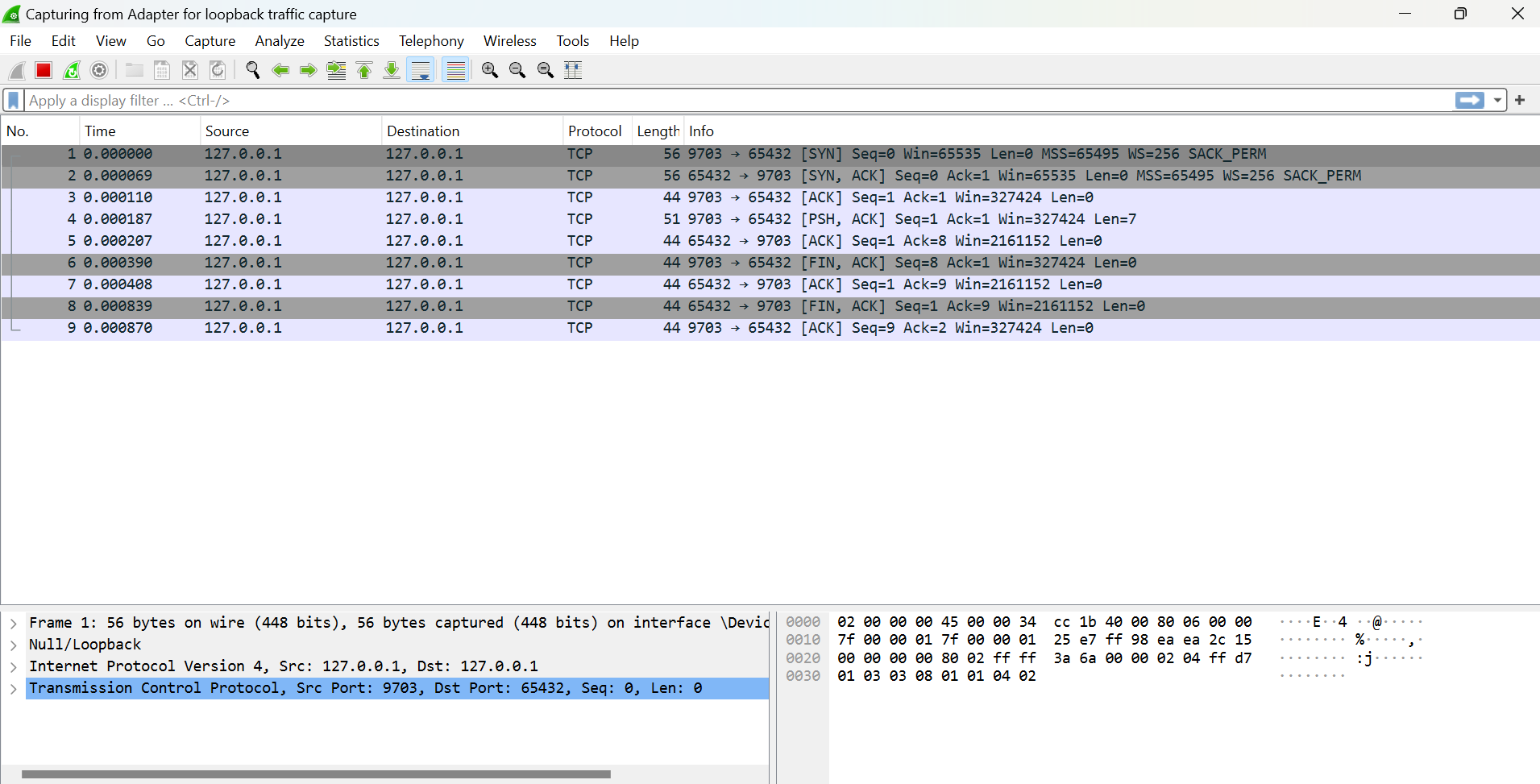
**Sender**

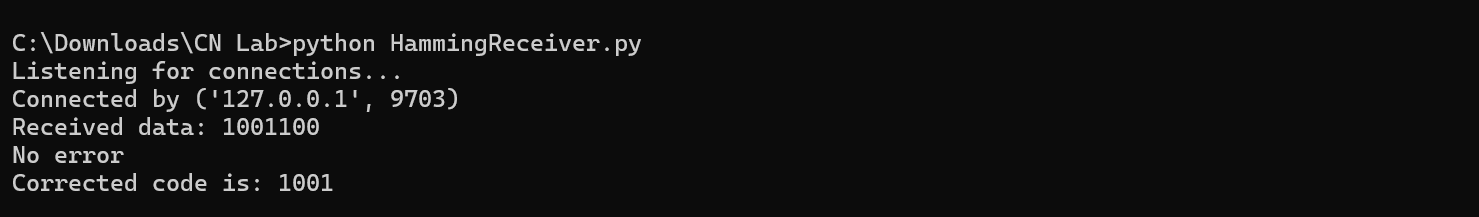
|  |
| --- |
| import socket  def calculate\_parity\_bits(data\_bits, r):      for i in range(r):          val = 0          for j in range(1, len(data\_bits) + 1):              if j & (2\*\*i) == (2\*\*i):                  val = val ^ int(data\_bits[-1 \* j])          data\_bits = data\_bits[:len(data\_bits) - (2\*\*i)] + str(val) + data\_bits[len(data\_bits) - (2\*\*i) + 1:]      return data\_bits  def hamming\_code(data):      m = len(data)      r = 1      while (2\*\*r < m + r + 1):          r = r + 1      j = 0      k = 1      hamming\_code = ''      for i in range(1, m + r + 1):          if i == 2\*\*j:              hamming\_code = hamming\_code + '0'              j = j + 1          else:              hamming\_code = hamming\_code + data[-1 \* k]              k = k + 1      hamming\_code = hamming\_code[::-1]      hamming\_code = calculate\_parity\_bits(hamming\_code, r)      return hamming\_code  # Sender function  def sender():      server\_address = ('localhost', 65432)      data = "1001"      encoded\_data = hamming\_code(data)      print("Encoded data:", encoded\_data)        with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:          s.connect(server\_address)          s.sendall(encoded\_data.encode('utf-8'))          print('Data sent')  if \_\_name\_\_ == "\_\_main\_\_":      sender() |

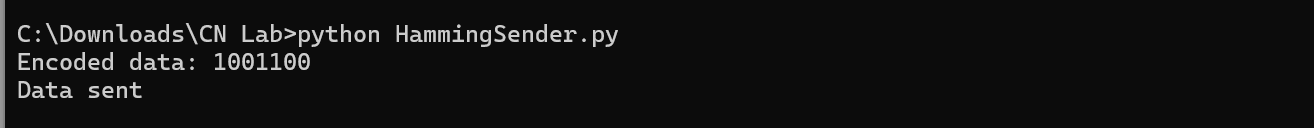
**Receiver**

|  |
| --- |
| import socket  def detect\_error(hamming\_code, r):      n = len(hamming\_code)      res = 0      for i in range(r):          val = 0          for j in range(1, n + 1):              if j & (2\*\*i) == (2\*\*i):                  val = val ^ int(hamming\_code[-1 \* j])          res = res + val \* (10\*\*i)      return int(str(res), 2)  def receiver\_function(data):      r = 1      while (2\*\*r < len(data) + 1):          r += 1      error\_position = detect\_error(data, r)      if error\_position == 0:          print("No error")      else:          print("Error is at position:", error\_position)          if data[error\_position - 1] == '0':              data = data[:error\_position - 1] + '1' + data[error\_position:]          else:              data = data[:error\_position - 1] + '0' + data[error\_position:]      j = 0      k = 1      corrected\_code = ''      for i in range(1, len(data) + 1):          if i == 2\*\*j:              j += 1          else:              corrected\_code = corrected\_code + data[-1 \* i]      print("Corrected code is:", corrected\_code[::-1])  # Receiver function  def receiver():      server\_address = ('localhost', 65432)        with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:          s.bind(server\_address)          s.listen()          print('Listening for connections...')            conn, addr = s.accept()          with conn:              print('Connected by', addr)              data = conn.recv(1024).decode('utf-8')              print("Received data:", data)              receiver\_function(data)  if \_\_name\_\_ == "\_\_main\_\_":      receiver() |

**Output:**

****

****

****