Text

Description automatically generated

**Activity based**

**Project Report on**

**Computer Network**

**Submitted to Vishwakarma University, Pune**

**Under the Initiative of**

**Contemporary Curriculum, Pedagogy, and Practice (C2P2)**

**By**

**Gargi Chitale Devanshu Katiyar Sakshi Tosniwal**

**SRN No : 202101776 SRN No: 202101210 SRN No:202101398**

**Roll No : 42 Roll No: 22 Roll No: 30**

**Div : E Div: E Div: E**

**Third Year Engineering**

**Department of Computer Engineering**

**Faculty of Science and Technology**

**Academic Year**

**2023-2024**

**Project Statement :**

Simulation of Datalink layer framing methods using Bit stuffing

**Problem Description :**

Simulation of Datalink layer framing methods using Bit stuffing

Bit stuffing is a data encoding technique used in computer networks and data communication systems to ensure data integrity and synchronization between sender and receiver.

Project Parts:

1. Implement Bit Stuffing: Develop a program or script to encode data using bit stuffing, including inserting extra bits based on specific rules (e.g., adding '0' bits after every five consecutive '1' bits) and simulate data transmission through a network or communication channel, with a printed display of stuffed data.

2. Bit Destuffing: Simulate the reception of stuffed data at the receiver's end, including receiving data over the network or communication channel. Write code for destuffing, removing extra bits (e.g., '0' bits) to recover the original data, and ensure the decoded data matches the initially stuffed data, displaying results using print statements.

3. Testing and Verification

a. Testing: Test your bit stuffing and destuffing implementation with different data patterns. Verify that data is correctly stuffed, transmitted, received, and destuffed.

b. Error Handling: Implement error handling to account for cases where the data might be incorrectly received or bits might be lost during transmission.

c. Validation: Validate that the destuffed data matches the original data, indicating that the bit stuffing and destuffing processes are functioning correctly.

d. Documentation and Presentation: Document your simulation, including the code, data patterns used for testing, and the results of your validation. Prepare a presentation that explains the bit stuffing process and how it is used to ensure data integrity in real-world communication systems.

**Theory:**

Bit destuffing is a technique used to recover the original data from a bit-stuffed stream. In the provided client-server communication code, bit destuffing is implemented to ensure that the received data can be correctly interpreted. The focus is on implementing bit destuffing in both the ClientThread and SecondPage classes.

Changes Made:

**ClientThread Class:**

**Bit Destuffing Method:**

Implemented the bit\_destuffing method to perform bit destuffing on received binary data.

**Modification in run Method:**

Applied bit destuffing to the received binary data before interpreting it.

**SecondPage Class:**

**Enhanced send\_message Method:**

No direct changes to the send\_message method, but it's crucial to be aware of the destuffing process when receiving data.

Execution Flow with Bit Destuffing:

**ClientThread Execution:**

Connects to the server and enters a loop to receive binary data.

Applies bit destuffing to the received binary data before interpreting it.

**SecondPage Interaction:**

Allows users to input messages through a QTextEdit widget.

Converts sender and receiver IDs to binary and converts each word to binary with bit stuffing before sending messages to the server.

**Recommendations and Considerations:**

**Testing:**

Thoroughly test the bit destuffing implementation to ensure it works correctly in various scenarios.

**Documentation and Comments:**

Update comments and documentation to reflect the changes made for bit destuffing.

**Error Handling:**

Include error handling mechanisms to deal with potential issues during the bit destuffing process.

**User Feedback:**

Provide feedback to the user about the received and destuffed data.

**Code :**

import socket

from PyQt5.QtWidgets import QApplication, QMainWindow, QTextEdit, QVBoxLayout, QWidget

from PyQt5.QtCore import QThread, pyqtSignal

import sys

receiver\_id = 211

sender\_id = 211

flag = '01111110'

class ServerThread(QThread):

    message\_received = pyqtSignal(str)

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

    def bit\_destuffing(self, data):

        destuffed\_data = ""

        consecutive\_ones = 0

        for bit in data:

            if bit == '1':

                consecutive\_ones += 1

            else:

                consecutive\_ones = 0

            destuffed\_data += bit

            # Handle destuffing: if 5 consecutive '1's are followed by '0', skip the stuffed '0'

            if consecutive\_ones == 5 and bit == '0':

                destuffed\_data = destuffed\_data[:-1]

        return destuffed\_data

    def process\_word(self, word):

        # Process the complete message (destuffing, removing header, etc.)

        destuffed\_data = self.bit\_destuffing(word)

        clean\_data = destuffed\_data[16:].replace(flag, '')

        # Emit the clean data for printing

        self.message\_received.emit(f"Received data after removing flag and header: {clean\_data}")

        # Convert binary data to ASCII

        ascii\_text = self.binary\_to\_ascii(clean\_data)

        self.message\_received.emit(f"Received data as ASCII: {ascii\_text}")

    def run(self):

        server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

        host = "127.0.0.1"

        port = 12346

        server.bind((host, port))

        server.listen(1)

        self.message\_received.emit("Server is waiting for connection...")

        conn, addr = server.accept()

        self.message\_received.emit(f"Connected to: {addr}")

        received\_data = ""  # To store all the received words

        data\_buffer = ""   # To accumulate partial data

        while True:

            data = conn.recv(1024).decode()

            if not data:

                break

            self.message\_received.emit(f"Received data from client: {data}")

            # Accumulate the data until the flag is detected

            data\_buffer += data

            while flag in data\_buffer:

                # Find the index of the flag

                flag\_index = data\_buffer.find(flag)

                # Extract the complete message up to the flag

                complete\_message = data\_buffer[:flag\_index]

                # Process each word in the complete message

                words = complete\_message.split(flag)

                for word in words:

                    if word:

                        self.process\_word(word)

                # Remove the processed data from the buffer

                data\_buffer = data\_buffer[flag\_index + len(flag):]

        conn.close()

    def binary\_to\_ascii(self, binary\_data):

        # Split the binary data into 8-bit chunks

        binary\_chunks = [binary\_data[i:i + 8] for i in range(0, len(binary\_data), 8)]

        # Convert each 8-bit chunk to decimal

        decimal\_values = [int(chunk, 2) for chunk in binary\_chunks]

        # Replace non-printable ASCII characters with '.'

        printable\_characters = [chr(value) if 32 <= value <= 126 else '.' for value in decimal\_values]

        # Join the characters to form the final ASCII text

        ascii\_text = ''.join(printable\_characters)

        return ascii\_text

class ServerWindow(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Server")

        self.setGeometry(100, 100, 600, 400)

        self.central\_widget = QWidget()

        self.setCentralWidget(self.central\_widget)

        self.layout = QVBoxLayout()

        self.server\_output = QTextEdit()

        self.server\_output.setReadOnly(True)

        self.layout.addWidget(self.server\_output)

        self.central\_widget.setLayout(self.layout)

        self.server\_thread = ServerThread()

        self.server\_thread.message\_received.connect(self.update\_server\_output)

        self.server\_thread.start()

    def update\_server\_output(self, message):

        self.server\_output.append(message)

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    server\_window = ServerWindow()

    server\_window.show()

    sys.exit(app.exec\_())

import socket

from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton, QTextEdit, QVBoxLayout, QWidget

from PyQt5.QtCore import QThread, pyqtSignal, Qt

import sys

receiver\_id = 211

sender\_id = 211

flag = '01111110'

class ClientThread(QThread):

    message\_received = pyqtSignal(str)

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

    def convert\_id\_to\_binary(self, id\_value):

        # Convert ID to binary representation

        return format(id\_value, '08b')

    def bit\_stuffing(self, data):

        stuffed\_data = ""

        consecutive\_ones = 0

        for bit in data:

            if bit == '1':

                consecutive\_ones += 1

            else:

                consecutive\_ones = 0

            stuffed\_data += bit

            if consecutive\_ones == 5:

                stuffed\_data += '0'

                consecutive\_ones = 0

        return stuffed\_data

    def run(self):

        global client

        client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

        host = "127.0.0.1"

        port = 12346

        try:

            client.connect((host, port))

        except ConnectionRefusedError:

            self.message\_received.emit("Failed to connect: Connection refused.")

            return

        except Exception as e:

            self.message\_received.emit(f"Failed to connect: {str(e)}")

            return

        while True:

            data\_to\_send = input("Enter words to be sent (Enter 'q' to quit): ")

            if data\_to\_send.lower() == 'q':

                break

            # Convert sender and receiver IDs to binary

            sender\_binary = self.convert\_id\_to\_binary(sender\_id)

            receiver\_binary = self.convert\_id\_to\_binary(receiver\_id)

            # Form the header by concatenating sender and receiver IDs

            complete\_header = flag + sender\_binary + receiver\_binary

            # Convert each word to binary and apply bit stuffing

            for word in data\_to\_send.split():

                words\_binary = self.bit\_stuffing(''.join(format(ord(x), '08b') if len(x) == 1 and x != ' ' else '') for x in word)

                # Combine flag, header, sender and receiver IDs, and stuffed data for each word

                complete\_data = complete\_header + words\_binary + flag

                # Logic to send complete binary data to the server

                client.send(complete\_data.encode())

        client.close()

class SecondPage(QWidget):

    def \_\_init\_\_(self, client\_thread, parent=None):

        super(SecondPage, self).\_\_init\_\_(parent)

        self.client\_thread = client\_thread

        self.layout = QVBoxLayout()

        self.input\_text = QTextEdit()

        self.send\_button = QPushButton("Send Message")

        self.send\_button.clicked.connect(self.send\_message)

        self.layout.addWidget(self.input\_text)

        self.layout.addWidget(self.send\_button)

        self.binary\_output = QTextEdit()

        self.binary\_output.setReadOnly(True)

        self.layout.addWidget(self.binary\_output)

        self.setLayout(self.layout)

    def send\_message(self):

        message = self.input\_text.toPlainText()

        print("Message sent to server:", message)

        # Convert the sender and receiver IDs to binary

        sender\_binary = format(sender\_id, '08b')

        receiver\_binary = format(receiver\_id, '08b')

        # Convert each word in the input message to binary and apply bit stuffing

        for word in message.split():

            words\_binary = self.client\_thread.bit\_stuffing(''.join(format(ord(x), '08b') if len(x) == 1 and x != ' ' else '') for x in word)

            # Combine flag, header, sender and receiver IDs, and stuffed data for each word

            complete\_data = flag + sender\_binary + receiver\_binary + words\_binary + flag

            # Display stuffed binary data in the GUI, including the flag, header, and stuffed data for each word

            header\_output = f"Header (including flag): {flag + sender\_binary + receiver\_binary + flag}\n"

            complete\_output = header\_output + f"Binary Equivalent (after bit stuffing) for each word: {words\_binary}\n"

            self.binary\_output.append(complete\_output)

            # Logic to send complete binary data to the server

            client.send(complete\_data.encode())

class ClientWindow(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Client")

        self.setGeometry(100, 100, 600, 400)

        self.client\_thread = ClientThread()

        self.central\_widget = QWidget()

        self.setCentralWidget(self.central\_widget)

        self.layout = QVBoxLayout()

        self.connect\_button = QPushButton("Connect to Server")

        self.connect\_button.clicked.connect(self.start\_client)

        self.layout.addWidget(self.connect\_button)

        self.stuffed\_data\_output = QTextEdit()

        self.stuffed\_data\_output.setReadOnly(True)

        self.layout.addWidget(self.stuffed\_data\_output)

        self.client\_output = QTextEdit()

        self.client\_output.setReadOnly(True)

        self.layout.addWidget(self.client\_output)

        self.central\_widget.setLayout(self.layout)

        self.second\_page = SecondPage(self.client\_thread, self)

        self.client\_thread.message\_received.connect(self.update\_server\_output)

    def update\_server\_output(self, message):

        self.client\_output.append(message)

    def start\_client(self):

        self.client\_output.clear()

        self.stuffed\_data\_output.clear()

        self.client\_thread.start()

        self.switch\_to\_second\_page()

    def switch\_to\_second\_page(self):

        self.setCentralWidget(self.second\_page)

    def closeEvent(self, event):

        # Override closeEvent to ensure the client socket is closed

        global client

        client.close()

        event.accept()

if \_\_name\_\_ == "\_\_main\_\_":

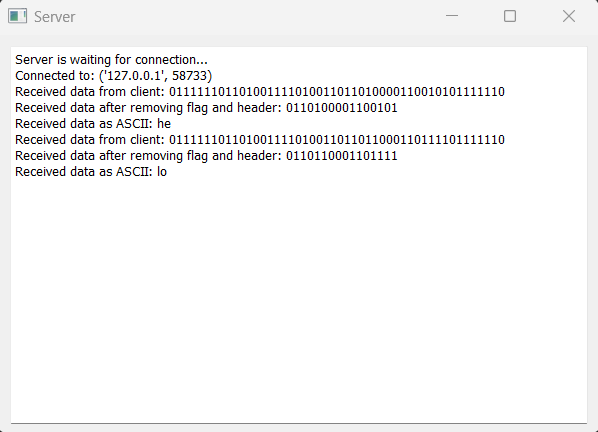
    app = QApplication(sys.argv)

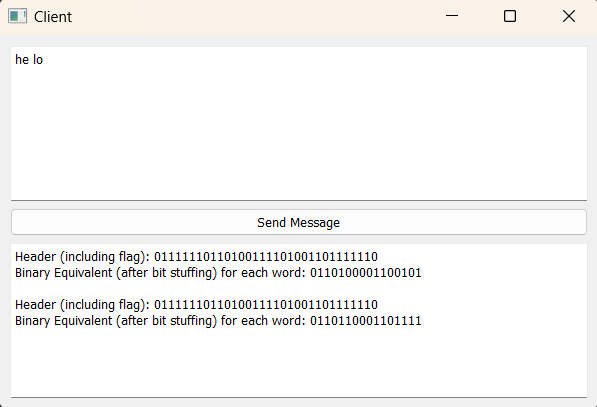
    client\_window = ClientWindow()

    client\_window.show()

    sys.exit(app.exec\_())

**Output:**

****

****

**Conclusion :**

The implementation of bit destuffing in the code allows for the correct interpretation of received data. The modified **ClientThread** class now incorporates bit destuffing logic, providing a more robust solution for handling incoming binary data. Testing and documentation are essential steps to ensure the correctness and clarity of the code.