**Batch: A-3 Roll No.: 16010122104**

**Experiment / assignment / tutorial No. 5**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

**Experiment No.:5**

|  |
| --- |
| **TITLE:** Flow control Mechanism: Selective Repeat ARQ Sliding Window Protocol using Socket programming |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:** Implementation of Flow Control Mechanism: Stop and Wait ARQ / Go-Back- N

/ Selective Repeat Sliding Window Protocol ARQ using sockets.

**Expected Outcome of Experiment:**

**CO:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. A. S. Tanenbaum, “Computer Networks”, Pearson Education, Fourth Edition
2. B. A. Forouzan, “Data Communications and Networking”, TMH, Fourth Edition

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

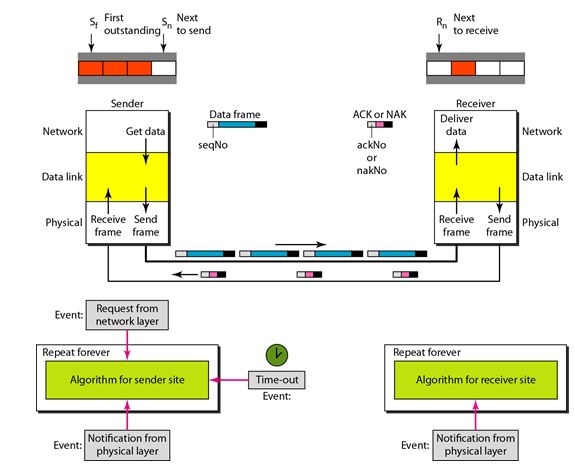
**Pre-Lab/ Prior Concepts:**

Java Socket Programming, Flow Control, Go-Back-Stop and Wait

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**New Concepts to be learned:** Window Flow Control **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Design of Go-Back-N ARQ**

****

1. Take data from user about how many bit windows is case of go back n and selective repeat.
2. Generate frames randomly and show the transmission
3. Generate the random number for the frame to be lost.
4. For Go – Back – N transmit all the frames after that number till max number
5. For Selective repeat transmit the selected frame which is not received by the receiver.

**IMPLEMENTATION: (**printout of code)

Client

import asyncio

import websockets

import json

import logging

logging.basicConfig(level=logging.INFO, format=' - %(message)s')

class Client:

    def \_\_init\_\_(self, window\_size=4):

        self.window\_size = window\_size

        self.base = 0

        self.next\_seq\_num = 0

        self.buffer = {}

        self.unacked\_frames = set()

        self.total\_frames = 13

        self.nak\_buffer = set()

    async def send\_frame(self, websocket, seq\_num, data, is\_retransmission=False):

        frame = {

            "type": "DATA",

            "seq\_num": seq\_num,

            "data": data,

            "is\_retransmission": is\_retransmission

        }

        await websocket.send(json.dumps(frame))

        if is\_retransmission:

            logging.info(f"Retransmitted lost frame {seq\_num}")

        else:

            logging.info(f"Sent frame {seq\_num}")

        self.unacked\_frames.add(seq\_num)

    async def start(self):

        uri = "ws://localhost:8000"

        async with websockets.connect(uri) as websocket:

            data\_to\_send = list(range(self.total\_frames))

            logging.info("Starting initial transmission phase")

            while self.next\_seq\_num < len(data\_to\_send):

                if len(self.unacked\_frames) < self.window\_size:

                    await self.send\_frame(websocket, self.next\_seq\_num, data\_to\_send[self.next\_seq\_num])

                    self.next\_seq\_num += 1

                try:

                    response = await asyncio.wait\_for(websocket.recv(), timeout=0.1)

                    response\_data = json.loads(response)

                    if response\_data["type"] == "ACK":

                        ack\_num = response\_data["ack\_num"]

                        logging.info(f"Received ACK for frame {ack\_num}")

                        if ack\_num in self.unacked\_frames:

                            self.unacked\_frames.remove(ack\_num)

                    elif response\_data["type"] == "NAK":

                        nak\_num = response\_data["nak\_num"]

                        logging.info(f"Received NAK for frame {nak\_num}")

                        self.nak\_buffer.add(nak\_num)

                        if nak\_num in self.unacked\_frames:

                            self.unacked\_frames.remove(nak\_num)

                except asyncio.TimeoutError:

                    continue

                except websockets.exceptions.ConnectionClosed:

                    return

            if self.nak\_buffer:

                logging.info(f"Starting retransmission phase for lost frames: {sorted(self.nak\_buffer)}")

                for seq\_num in sorted(self.nak\_buffer):

                    await self.send\_frame(websocket, seq\_num, data\_to\_send[seq\_num], is\_retransmission=True)

                    while True:

                        try:

                            response = await websocket.recv()

                            response\_data = json.loads(response)

                            if response\_data["type"] == "ACK" and response\_data["ack\_num"] == seq\_num:

                                logging.info(f"Retransmitted frame {seq\_num} acknowledged")

                                break

                        except websockets.exceptions.ConnectionClosed:

                            return

            logging.info("Transmission complete")

            await websocket.send(json.dumps({"type": "COMPLETE"}))

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

    client = Client(window\_size=4)

    asyncio.run(client.start())

Server

import asyncio

import websockets

import json

import random

import logging

logging.basicConfig(level=logging.INFO, format='%(message)s')

class Server:

    def \_\_init\_\_(self, window\_size=4):

        self.window\_size = window\_size

        self.base = 0

        self.received\_buffer = {}

        self.lost\_frames = set()

        self.total\_frames = 13

        self.delivered\_frames = []

    def deliver\_frames(self):

        """Deliver frames in order"""

        while self.base in self.received\_buffer:

            frame\_data = self.received\_buffer.pop(self.base)

            self.delivered\_frames.append(frame\_data)

            logging.info(f"Delivered frame {self.base} with data {frame\_data}")

            self.base += 1

    async def handle\_connection(self, websocket):

        logging.info("Client connected")

        while True:

            try:

                message = await websocket.recv()

                frame = json.loads(message)

                if frame["type"] == "COMPLETE":

                    logging.info("Received transmission complete signal")

                    break

                seq\_num = frame["seq\_num"]

                is\_retransmission = frame.get("is\_retransmission", False)

                if not is\_retransmission:

                    if random.random() < 0.2:  # 20% chance of frame loss

                        logging.info(f"Frame {seq\_num} lost")

                        self.lost\_frames.add(seq\_num)

                        response = {

                            "type": "NAK",

                            "nak\_num": seq\_num

                        }

                        await websocket.send(json.dumps(response))

                        continue

                logging.info(f"Received {'retransmitted ' if is\_retransmission else ''}frame {seq\_num}")

                self.received\_buffer[seq\_num] = frame["data"]

                response = {

                    "type": "ACK",

                    "ack\_num": seq\_num

                }

                await websocket.send(json.dumps(response))

                self.deliver\_frames()

            except websockets.exceptions.ConnectionClosed:

                logging.info("Client disconnected")

                break

        logging.info("\nFinal Delivery Report:")

        logging.info(f"Total frames delivered: {len(self.delivered\_frames)}")

        logging.info(f"Frames that were lost and retransmitted: {sorted(self.lost\_frames)}")

        logging.info(f"Final delivered sequence: {self.delivered\_frames}")

    async def start(self):

        async with websockets.serve(self.handle\_connection, "localhost", 8000):

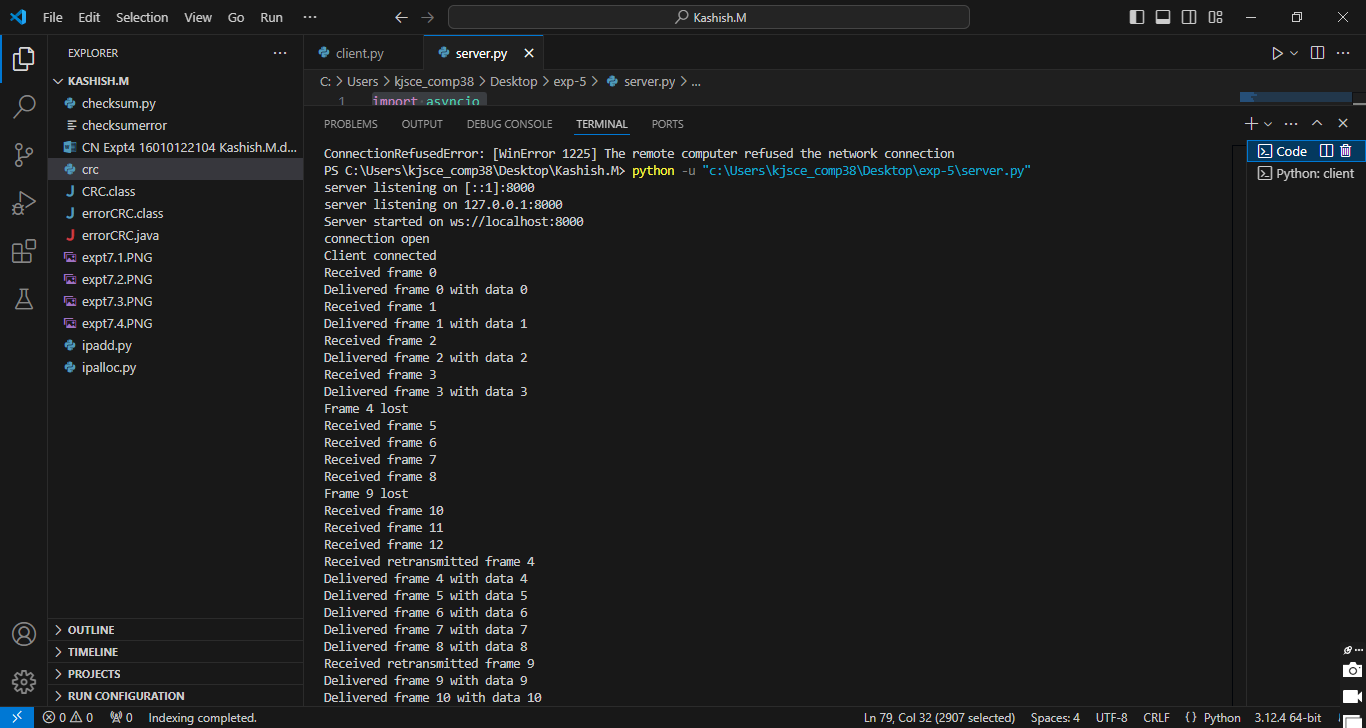
            logging.info("Server started on ws://localhost:8000")

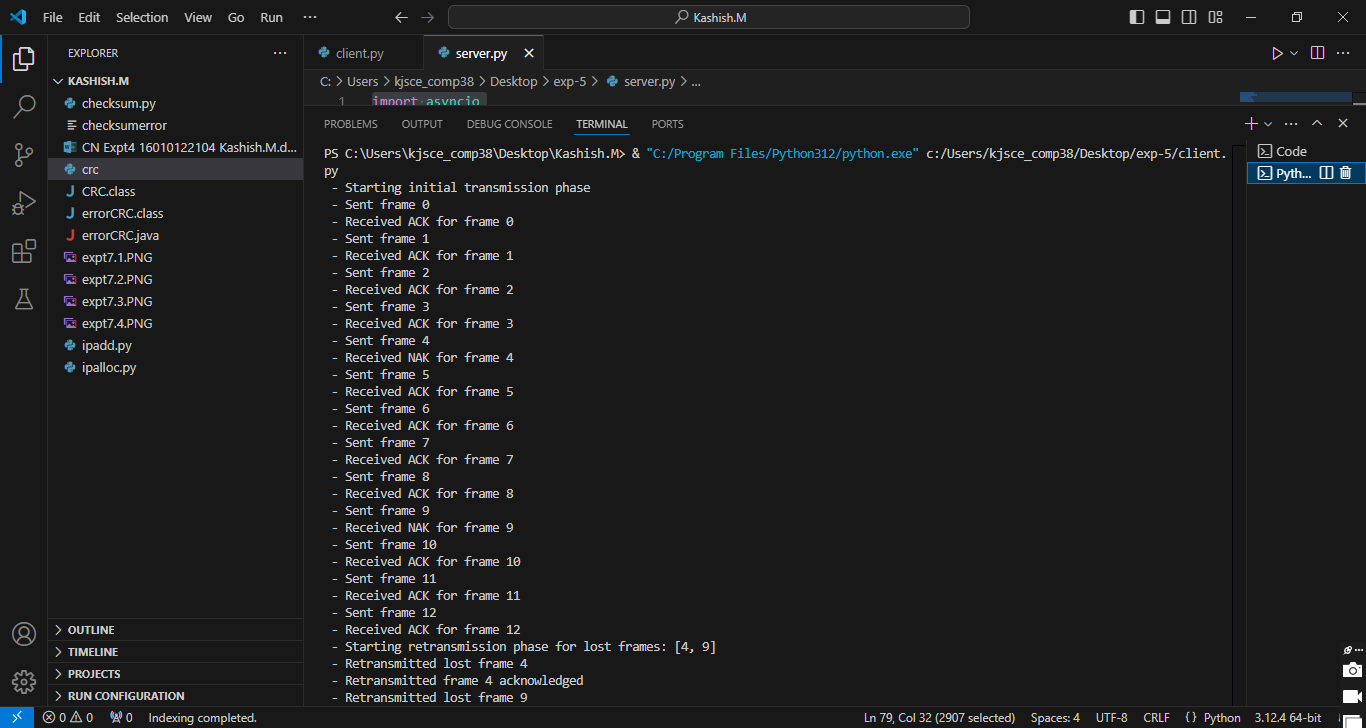
            await asyncio.Future()

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

    server = Server(window\_size=4)

    asyncio.run(server.start())





**CONCLUSION:**

We implemented Flow control Mechanism: Selective Repeat ARQ Sliding Window Protocol using Socket programming.

**Post Lab Questions**

1. Compare Go-Back-N and Stop and Wait.

Ans:

 **Stop-and-Wait**:

* **Transmission**: Sends one frame at a time and waits for an acknowledgment (ACK) before sending the next.
* **Efficiency**: Low efficiency, as the sender remains idle during the waiting period.
* **Error Handling**: If a frame is lost or an ACK is not received, the sender retransmits the frame.
* **Window Size**: The window size is 1.

 **Go-Back-N**:

* **Transmission**: Sends multiple frames before waiting for an acknowledgment (up to a window size).
* **Efficiency**: Higher efficiency, as the sender can transmit multiple frames without waiting after each one.
* **Error Handling**: If a frame is lost, all subsequent frames are discarded and retransmitted starting from the lost frame.
* **Window Size**: The window size is greater than 1 (up to 2n−12^n - 12n−1).

1. What is Flow Control and why it is necessary?

Ans:

 **Definition**: Flow control is a technique used in data communication to ensure that the sender does not overwhelm the receiver by sending data faster than the receiver can process it.

 **Necessity**:

* **Prevents Buffer Overflow**: Ensures that the receiver's buffer doesn’t overflow due to excessive incoming data.
* **Maintains Transmission Efficiency**: Helps maintain a steady flow of data without unnecessary retransmissions due to lost data.
* **Synchronizes Transmission Speed**: Matches the data transmission rate of the sender with the processing capability of the receiver.

1. The maximum window size for data transmission using the selective reject protocol with n-bit frame sequence numbers is  
   a) 2n            b) 2n-1                    c) 2n-1                   d)2n-2

Ans:

The maximum window size for data transmission using the selective reject (selective repeat) protocol with **n-bit** frame sequence numbers is:

* **Answer: c) 2n−12^{n-1}2n−1**

**Date:21/10/2024 Signature of Faculty In-charge**