**22BCE0476**

**AMAN CHAUHAN**

****

BCSE308P - Computer Networks Lab

LAB REPORT

DA-3 FLOW CONTROL

**4. Flow Control**

1. Stop and wait ARQ

2. Go back N ARQ

3. Selective repeat ARQ

**1. Stop and wait ARQ protocol:**

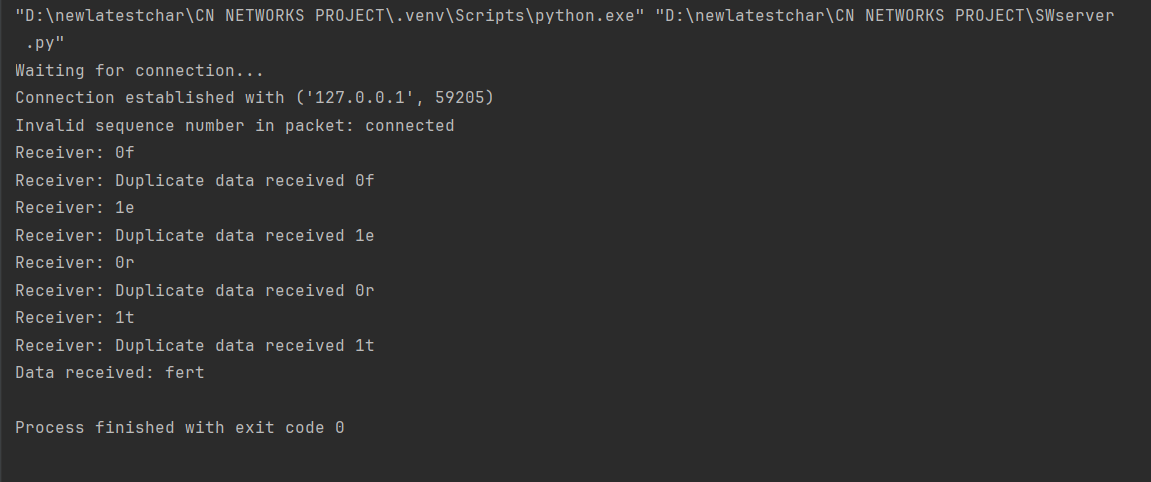
**Server:**

import socket  
import pickle  
  
  
def server():  
 # Create a TCP/IP socket  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1) # Allow address reuse  
 server\_socket.bind(('localhost', 2005)) # Use localhost and port 2005  
 server\_socket.listen(1)  
 print("Waiting for connection...")  
  
 connection, client\_address = server\_socket.accept()  
 print("Connection established with", client\_address)  
  
 sequence = 0  
 data = ""  
 i = 0  
  
 try:  
 while True:  
 packet = pickle.loads(connection.recv(1024)) # Receive packet from client  
 if packet == 'end':  
 break  
  
 # Ensure the packet starts with a valid sequence number  
 try:  
 packet\_sequence = int(packet[0]) # First character should be the sequence number  
 if packet\_sequence == sequence:  
 data += packet[1:]  
 sequence = 1 - sequence # Alternate sequence (0 -> 1 or 1 -> 0)  
 print(f"Receiver: {packet}")  
 ack = str(sequence)  
 else:  
 print(f"Receiver: Duplicate data received {packet}")  
 ack = str(1 - sequence)  
 except ValueError:  
 print(f"Invalid sequence number in packet: {packet}")  
 ack = str(sequence) # In case of error, keep the same sequence  
  
 # Send acknowledgment back to client  
 connection.sendall(ack.encode())  
 except Exception as e:  
 print(f"Server error: {e}")  
 finally:  
 print("Data received:", data)  
 connection.close()  
 server\_socket.close()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 server()

**Client:**

import socket  
import pickle  
import time  
  
  
def client():  
 # Create a TCP/IP socket  
 client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 client\_socket.settimeout(5) # Set a timeout for socket operations (5 seconds)  
  
 try:  
 client\_socket.connect(('localhost', 2005)) # Ensure server is up before connecting  
  
 packet = input("Enter data to send: ")  
 sequence = 0  
 i = 0  
 n = len(packet)  
  
 # Sending initial connection message  
 client\_socket.sendall(pickle.dumps("connected"))  
 ack = client\_socket.recv(1024).decode()  
 print("Receiver > ", ack)  
  
 while i < n:  
 # Format packet: send sequence number followed by the character  
 msg = str(sequence) + packet[i]  
 client\_socket.sendall(pickle.dumps(msg))  
 print(f"Sender: Sent {msg}")  
  
 # Wait for acknowledgment  
 try:  
 ack = client\_socket.recv(1024).decode()  
 if ack == str(sequence):  
 print(f"Receiver > Packet {msg} received")  
 i += 1  
 sequence = 1 - sequence # Alternate sequence (0 -> 1 or 1 -> 0)  
 else:  
 print("Timeout, resending packet...")  
 time.sleep(2) # Simulate timeout before retransmitting the packet  
 except socket.timeout:  
 print("Timeout occurred, resending packet...")  
 time.sleep(2) # Simulate timeout before retransmitting the packet  
  
 # Send 'end' to signal completion  
 client\_socket.sendall(pickle.dumps("end"))  
 print("All data sent. Exiting.")  
 except Exception as e:  
 print(f"Client error: {e}")  
 finally:  
 client\_socket.close()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 client()

**Output:**

**Server output:** ****

**Client output:**

**A screenshot of a computer program

Description automatically generated**

**2. Go-Back-N protocol:**

**Server:**

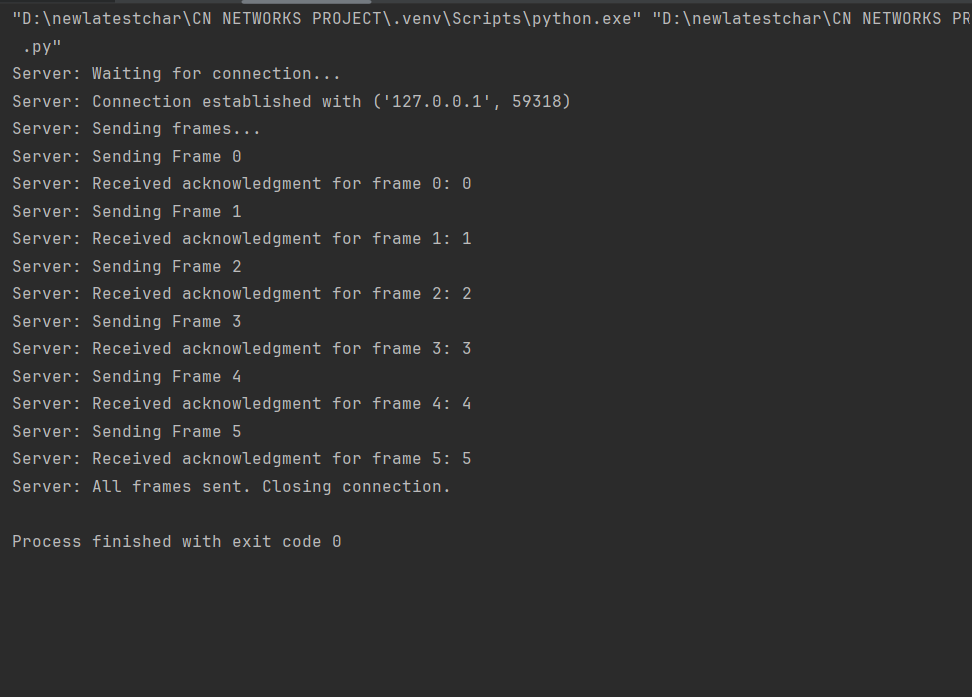
import socket  
import time  
  
  
def server():  
 # Set up the server to listen for connections  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 server\_socket.bind(('localhost', 5000))  
 server\_socket.listen(1)  
  
 print("Server: Waiting for connection...")  
 client\_socket, client\_address = server\_socket.accept()  
 print(f"Server: Connection established with {client\_address}")  
  
 # Set up input and output streams  
 in\_stream = client\_socket.makefile('rb') # Buffered input stream  
 out\_stream = client\_socket.makefile('wb') # Buffered output stream  
  
 # Read the number of frames to send  
 num\_frames = int(in\_stream.readline().decode().strip())  
 # Read the error flag (0 - no error, 1 - error)  
 error\_choice = int(in\_stream.readline().decode().strip())  
  
 print("Server: Sending frames...")  
  
 # Create a list to track the acknowledgment for each frame  
 ack\_received = [False] \* num\_frames  
  
 # Simulate the sending and acknowledgment of frames  
 for i in range(num\_frames):  
 frame = f"Frame {i}"  
 print(f"Server: Sending {frame}")  
 out\_stream.write(frame.encode() + b'\n')  
 out\_stream.flush()  
  
 # Wait for acknowledgment or handle frame error  
 if error\_choice == 0: # No error  
 ack = int(in\_stream.readline().decode().strip())  
 print(f"Server: Received acknowledgment for frame {i}: {ack}")  
 if ack != i:  
 print(f"Server: Frame {i} not acknowledged. Resending...")  
 # Resend the frame  
 out\_stream.write(frame.encode() + b'\n')  
 out\_stream.flush()  
 time.sleep(1) # Simulate delay  
 ack\_received[i] = True  
 else: # Simulate error  
 ack = int(in\_stream.readline().decode().strip())  
 if ack == -1: # Negative acknowledgment  
 print(f"Server: Discarded frame {i}, sending negative acknowledgment.")  
 out\_stream.write(b'-1\n')  
 out\_stream.flush()  
 else:  
 ack\_received[i] = True  
  
 # Close the connection after sending all frames  
 print("Server: All frames sent. Closing connection.")  
 client\_socket.close()  
 server\_socket.close()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 server()

**Client:**

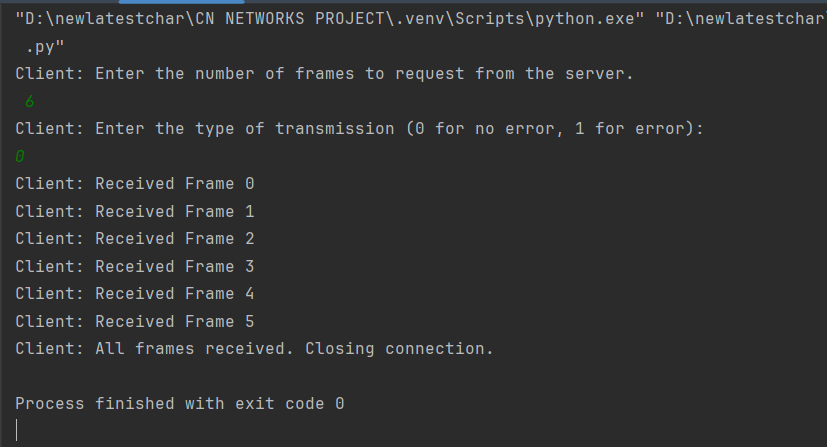
import socket  
  
  
def client():  
 # Set up the client to connect to the server  
 client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 client\_socket.connect(('localhost', 5000))  
  
 # Set up input and output streams  
 in\_stream = client\_socket.makefile('rb') # Buffered input stream  
 out\_stream = client\_socket.makefile('wb') # Buffered output stream  
  
 print("Client: Enter the number of frames to request from the server.")  
 num\_frames = int(input())  
 out\_stream.write(f"{num\_frames}\n".encode())  
 out\_stream.flush()  
  
 print("Client: Enter the type of transmission (0 for no error, 1 for error):")  
 error\_choice = int(input())  
 out\_stream.write(f"{error\_choice}\n".encode())  
 out\_stream.flush()  
  
 # Simulate the receipt of frames and send acknowledgments  
 expected\_ack = 0  
 for i in range(num\_frames):  
 frame = in\_stream.readline().decode().strip()  
 print(f"Client: Received {frame}")  
  
 # If no error, send the correct acknowledgment  
 if error\_choice == 0:  
 out\_stream.write(f"{i}\n".encode()) # Acknowledge frame i  
 else: # Simulate error for testing  
 if i % 2 == 0: # Discard every other frame  
 print(f"Client: Discarding {frame}")  
 out\_stream.write(b"-1\n") # Negative acknowledgment  
 else:  
 out\_stream.write(f"{i}\n".encode()) # Acknowledge frame i  
 out\_stream.flush()  
  
 print("Client: All frames received. Closing connection.")  
 client\_socket.close()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 client()

**Output:**

**Server output:**

****

**Client output:**

****

**3. Selective repeat ARQ protocol:**

**Server:**

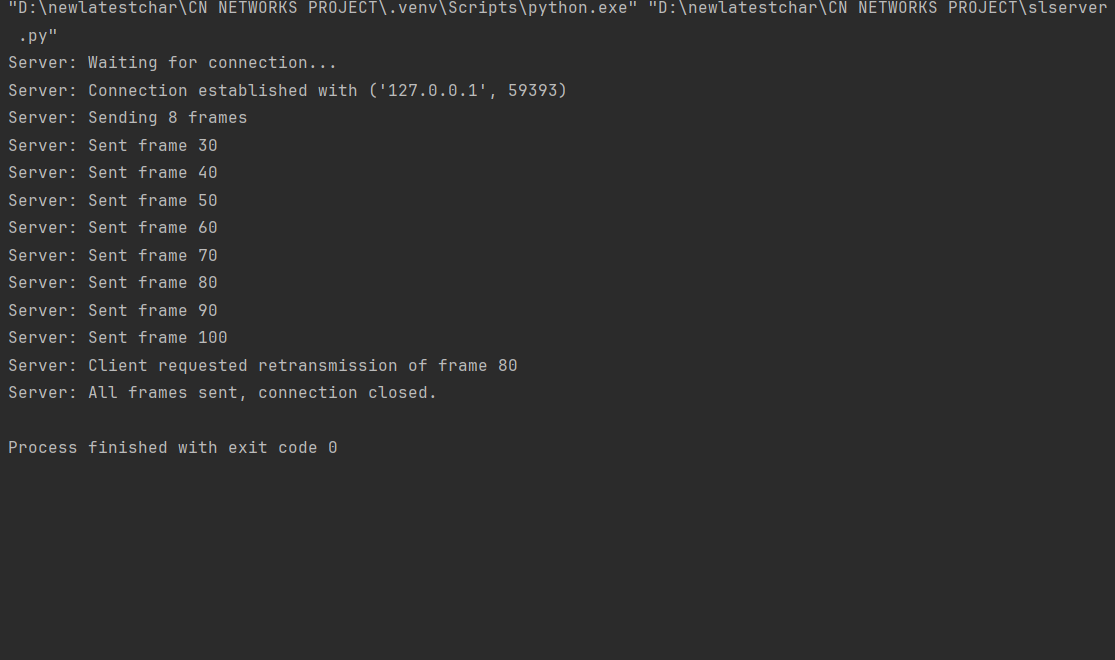
import socket  
import time  
  
  
def server():  
 # Set up the server to listen for connections  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 server\_socket.bind(('localhost', 8011))  
 server\_socket.listen(1)  
  
 print("Server: Waiting for connection...")  
 client\_socket, client\_address = server\_socket.accept()  
 print(f"Server: Connection established with {client\_address}")  
  
 # Set up input and output streams  
 in\_stream = client\_socket.makefile('rb') # Buffered input stream  
 out\_stream = client\_socket.makefile('wb') # Buffered output stream  
  
 # Define frames to send  
 frames = [30, 40, 50, 60, 70, 80, 90, 100]  
 print(f"Server: Sending {len(frames)} frames")  
  
 # Send the number of frames  
 out\_stream.write(f"{len(frames)}\n".encode())  
 out\_stream.flush()  
  
 # Send frames one by one  
 for frame in frames:  
 out\_stream.write(f"{frame}\n".encode())  
 out\_stream.flush()  
 print(f"Server: Sent frame {frame}")  
  
 # Wait for the client to request retransmission  
 frame\_to\_retransmit = int(in\_stream.readline().decode().strip())  
 print(f"Server: Client requested retransmission of frame {frames[frame\_to\_retransmit]}")  
  
 # Retransmit the requested frame  
 out\_stream.write(f"{frames[frame\_to\_retransmit]}\n".encode())  
 out\_stream.flush()  
  
 print("Server: All frames sent, connection closed.")  
 client\_socket.close()  
 server\_socket.close()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 server()

**Client:**

import socket  
  
  
def client():  
 # Set up the client to connect to the server  
 client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 client\_socket.connect(('localhost', 8011))  
  
 # Set up input and output streams  
 in\_stream = client\_socket.makefile('rb') # Buffered input stream  
 out\_stream = client\_socket.makefile('wb') # Buffered output stream  
  
 # Receive the number of frames  
 num\_frames = int(in\_stream.readline().decode().strip())  
 print(f"Client: Number of frames to receive: {num\_frames}")  
  
 # Receive the frames  
 received\_frames = []  
 for \_ in range(num\_frames):  
 frame = int(in\_stream.readline().decode().strip())  
 print(f"Client: Received frame {frame}")  
 received\_frames.append(frame)  
  
 # Simulate an error by discarding one of the frames  
 received\_frames[5] = -1 # Discard the frame at index 5  
  
 # Print the frames  
 for i, frame in enumerate(received\_frames):  
 if frame == -1:  
 print(f"Client: Frame {i} is missing or corrupted. Requesting retransmission.")  
 out\_stream.write(f"{i}\n".encode())  
 out\_stream.flush()  
 else:  
 print(f"Client: Frame {i} received successfully.")  
  
 # Receive the retransmitted frame  
 retransmitted\_frame = int(in\_stream.readline().decode().strip())  
 print(f"Client: Received retransmitted frame: {retransmitted\_frame}")  
  
 print("Client: All frames received, quitting.")  
 client\_socket.close()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 client()

**Output:**

**Server output:**

****

**Client output:**

A screenshot of a computer program

Description automatically generated

A piece of paper with writing on it

Description automatically generated

A paper with writing on it

Description automatically generated