COMPUTER NETWORKS LAB

NAME: ARITRA DATTA

ROLL NO: 002010501054

CLASS: BCSE – III

GROUP: A2

ASSIGNMENT: 2

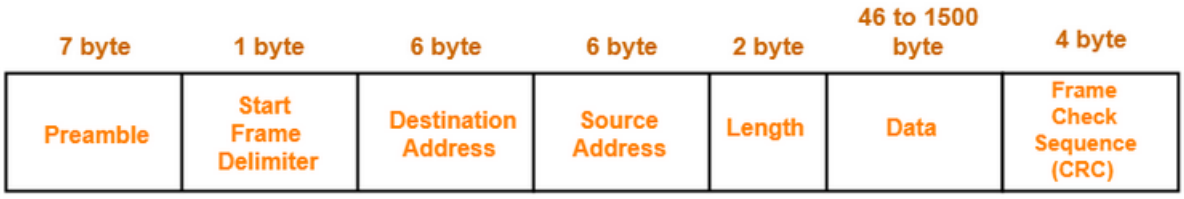
DEADLINE: 28th August, 2022

Problem Statement: Implement three data link layer protocols, Stop and Wait, Go Back N Sliding Window and Selective Repeat Sliding Window for flow control. Sender, Receiver and Channel all are independent processes. There may be multiple Transmitter and Receiver processes, but only one Channel process. The channel process introduces random delay and/or bit error while transferring frames. Define your own frame format or you may use IEEE 802.3 Ethernet frame format.

Date of Submission: 5 th September, 2022

DESIGN

The Ethernet IEEE 802.3 Frame Format is defined as follows.



In my implementation, I have taken the “Data” part to be of 46 bytes.

Preamble: - Pattern of alternating 0’s and 1’s.

Start of Frame Delimiter: - 10101011

CRC Polynomial used: - 10000010011000001000111010110111

The “Length” part has been divided to contain the type of packet (data, ACK or NAK) and the sequence number of the packet. Both of these are of size 1 byte each.

Sender reads in the message to be sent from the input file, makes a packet out of it (as per the IEEE 802.3 Ethernet Frame Format), adds redundant bits, converts it into frames and sends it to the channel. Multiple senders can be connected to the channel. The sender keeps a copy of the sent frame according to various protocols so that it can be resent if needed later. The noisy channel randomly introduces error, delay or it may even drop the frame. If the frame reaches the receiver, it checks the frame for any sort of error. If everything is fine, it sends an ACK (Acknowledgement), else it may either discards the packet or send an NAK (in case of Selective Repeat ARQ). The ACKs/NAKs pass through the channel and reach the sender. The sender then checks it and either sends the next frames or resends the previous (lost in transit) frames. Also, according to the specific policy, the frames may be resent by the sender based on some timeout policy. The receiver, when it receives the correct frames, writes them to an output file. Messages can be written in order with the help of the sequence numbers contained in each packet. At the end, this file contains the total message received.

The CRC error detection mechanism and the mechanism to inject errors have been incorporated from the previous assignment.

Protocol	**Window Size**	
	Receiver	Sender
Stop-And-Wait ARQ	1	1
Go-Back-N ARQ	1	$2^m - 1$
Selective Repeat ARQ	2^{m-1}	2^{m-1}

Here, m is the number of bits needed to represent the maximum sequence number in of the packets.

IMPLEMENTATION

STOP N WAIT PROTOCOL

SERVER.py

```python
import socket
import random

HEADERSIZE = 10
PROBABILITY = 0.3
ACK = 0
SEQ = 1
PORT = 1234

# Create a socket
# socket.AF_INET - address family, IPv4, some otehr possible are AF_INET6, AF_BLUETOOTH, AF_UNIX
# socket.SOCK_STREAM - TCP, conection-based, socket.SOCK_DGRAM - UDP, connectionless, datagrams, socket.SOCK_RAW - raw IP packets
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


server_socket.bind((socket.gethostname(), PORT))

server_socket.listen(5)

client_socket, addr = server_socket.accept()
print("Connection established from " + str(addr))


def injectError(data) :
    prob = random.uniform(0,1)
    if prob < PROBABILITY:
        data = '1' + str(random.randint(0,8)) + data[2:]
    return data

def extract_frame(data):
    return {'pkt_type' : int(data[0]),'frame_id' : int(data[1]),'res' : data[2:]}

def createFrame(pkt_type, seq_no, msg):
    message = str(pkt_type) + str(seq_no) + msg
    #message = injectError(message)
    return f"{len(message):<{HEADERSIZE}}" + message
```

```python
def receive_message(client_socket):
    try:
        msg_header = client_socket.recv(HEADERSIZE)
        if not len(msg_header):
            return False

        msg_len = int(msg_header.decode('utf-8').strip())
        data = client_socket.recv(msg_len).decode('utf-8')
        return data

    except:
        return False

frame_no = 0
recv = []

while True:

    client_response = receive_message(client_socket)
    if client_response is False:
        break

    client_response = extract_frame(client_response)


    if client_response['pkt_type'] == SEQ and client_response['frame_id'] == frame_no:
        recv.append(client_response['res'])
        prob = random.uniform(0,1)
        if prob > PROBABILITY:
            print("Packet " + str(frame_no) + " Received")
            print("ACK Sent\n")
            client_socket.send(createFrame(ACK,frame_no,"ACK").encode('utf-8'))
            frame_no = frame_no + 1
        else:
            recv.pop()
            print("ACK for Packet "+str(frame_no) + " Lost!!\n")
    else :
        print("Invalid Packet")
        print("Frame " + str(frame_no) + " not Received\n")

recv_data = ""
for d in recv:
    recv_data = recv_data + d
print("Data Received : " + recv_data)
server_socket.close()
```

CLIENT.py

```
import socket
import random
import time

PORT = 1234
PROBABILITY = 0.3
HEADERSIZE = 10
PACKET_SIZE = 8
TIMER = 5
ACK = 0
SEQ = 1

sender = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
sender.connect((socket.gethostname(),PORT))

def injectError(data) :
    prob = random.uniform(0,1)
    print(prob)
    if prob < PROBABILITY:
        data = '0' + str(random.randint(0,8)) + data[2:]
    return data

def extract_frame(data):
   return {'pkt_type' : int(data[0]),'frame_id' : int(data[1]),'res' : data[2:]}


def createFrame(pkt_type, seq_no, msg):
    message = str(pkt_type) + str(seq_no) + msg
    # message = injectError(message)
    return f"{len(message):<{HEADERSIZE}}" + message

def createDataPackets(data) :
    data_frames = []
    l = len(data)
    pkt_no,i = 0,0
    while i < l:
        data_frames.append(createFrame(SEQ,pkt_no,data[i:min(i+PACKET_SIZE,l)]).encode('utf-8'))
```

```
        data_frames.append(createFrame(SEQ,pkt_no,data[i:min(i+PACKET_SIZE,l)]).encode('utf-8'))
        pkt_no = pkt_no + 1
        i = i + PACKET_SIZE
    print("Total Packets = " + str(pkt_no)+"\n")
    return data_frames,pkt_no

def receive_message(client_socket):
    client_socket.settimeout(TIMER)
    try:
        msg_header = client_socket.recv(HEADERSIZE)

        if not len(msg_header):
            return False

        msg_len = int(msg_header.decode('utf-8').strip())

        data = client_socket.recv(msg_len).decode('utf-8')
        client_socket.settimeout(None)
        return data,False

    except:
        client_socket.settimeout(None)
        return False,True


data = input("Enter the data to be sent : ")
frames,last_packet = createDataPackets(data)
frame_no = 0
ack_recv = True

while True:

    if ack_recv:
        sender.send(frames[frame_no])
        print("Frame "+ str(frame_no) + " sent\n")
```

```
        sender.send(frames[frame_no])
        print("Frame "+ str(frame_no) + " sent\n")


    receiver_res,TIMEOUT = receive_message(sender)


    if TIMEOUT:
        print("Timeout Occured")
        print("Resending Frame " + str(frame_no) + "\n")
        ack_recv = True
        continue


    if receiver_res is False:
        break

    receiver_res = extract_frame(receiver_res)


    if receiver_res['pkt_type'] == ACK and receiver_res['frame_id'] == frame_no :
        print("ACK Received\nReady to send Frame : " + str(frame_no+1)+"\n")
        ack_recv = True
        frame_no = frame_no + 1
    else:
        ack_recv = False
        print("Invalid ACK Received\n")

    if frame_no == last_packet:
        break


print("All Packets sent")
sender.close()
```

OUTPUT

```
E:\NETWORK\Flow Control\Stop N Wait>python server.py
Connection established from ('192.168.1.7', 53475)
ACK for Packet 0 Lost!!

Packet 0 Received
ACK Sent

Packet 1 Received
ACK Sent

Packet 2 Received
ACK Sent

Packet 3 Received
ACK Sent

Data Received : hello how are you, I am fine

E:\NETWORK\Flow Control\Stop N Wait>
```

RECEIVER

```
E:\NETWORK\Flow Control\Stop N Wait>python client.py
Enter the data to be sent : hello how are you, I am fine
Total Packets = 4

Frame 0 sent

Timeout Occured
Resending Frame 0

Frame 0 sent

ACK Received
Ready to send Frame : 1

Frame 1 sent

ACK Received
Ready to send Frame : 2

Frame 2 sent

ACK Received
Ready to send Frame : 3

Frame 3 sent

ACK Received
Ready to send Frame : 4

All Packets sent

E:\NETWORK\Flow Control\Stop N Wait>
```

SENDER

GO BACK N PROTOCOL

CLIENT.py

```
import socket
import random
import time

PORT = 1234
HEADERSIZE = 10
PACKET_SIZE = 8
TIMER = 5
WINDOW_SIZE = 4
PACKET_SIZE = 8
ACK = 0
SEQ = 1
PROBABILITY = 0.2


sender = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
sender.connect((socket.gethostname(),PORT))

def injectError(data) :
    err = random.randint(0,1)
    if err:
        data = '0' + str(random.randint(0,8)) + data[2:]
    return data

def extract_frame(data):
    ind = PACKET_SIZE
    msg = ""
    if(data[ind] == '/'):
        msg = data[0:ind]
        ind = ind + 1
    else:
        msg = data[0:ind].rstrip()
        msg = msg[:-1]

    return {
            'pkt_type' : int(data[ind]),

            'packet_no' : int(data[ind+1:]),

            'res' : msg,
            }


def createFrame(pkt_type, seq_no, msg):
    msg = msg + "/"
    msg = msg = f"{msg:<{PACKET_SIZE}}"
    message = msg + str(pkt_type) + str(seq_no)
    return f"{len(message):<{HEADERSIZE}}" + message
```

```
def createDataPackets(data) :
    data_frames = []
    l = len(data)
    pkt_no,i = 0,0
    while i < l:
        data_frames.append(createFrame(SEQ,pkt_no,data[i:min(i+PACKET_SIZE,l)]).encode('utf-8'))
        pkt_no = pkt_no + 1
        i = i + PACKET_SIZE
    print("Total Packets = " + str(pkt_no)+"\n")
    return data_frames,pkt_no

def receive_message(client_socket):
    client_socket.settimeout(TIMER)
    try:
        msg_header = client_socket.recv(HEADERSIZE)

        if not len(msg_header):
            return False

        msg_len = int(msg_header.decode('utf-8').strip())

        data = client_socket.recv(msg_len).decode('utf-8')
        client_socket.settimeout(None)
        return data,False

    except:
        client_socket.settimeout(None)
        return False,True

data = input("Enter the data to be sent : ")
frames,last_packet = createDataPackets(data)
WINDOW_SIZE = min(WINDOW_SIZE,last_packet)
base_pkt,last_sent = 0,0

while True:

    while last_sent - base_pkt < WINDOW_SIZE  and  last_sent < last_packet :
        prob = random.uniform(0,1)
        if prob > PROBABILITY:
            sender.send(frames[last_sent])
            print("Packet "+ str(last_sent) + " Sent\n")
        else:
            print("Packet "+ str(last_sent) + " Lost!!\n")
        last_sent = last_sent + 1

    receiver_res,TIMEOUT = receive_message(sender)
```

```
    if TIMEOUT:
        print("Resending packets in the current window -> ["+str(base_pkt)+","+str(last_sent-1)+"] : ")
        for packet in range(base_pkt,last_sent):
            print("Packet "+ str(packet) + " Sent")
            sender.send(frames[packet])
        print()
    else:
        receiver_res = extract_frame(receiver_res)
        if receiver_res['packet_no'] != base_pkt or receiver_res['pkt_type'] != ACK:
            print("Invalid ACK Received\n")
        else:
            print("ACK Received for Packet : " + str(base_pkt))
            base_pkt = base_pkt + 1

    if base_pkt  == last_packet :
        break

print("\nAll Packets sent")
sender.close()
```

SERVER.py

```
import socket
import random

HEADERSIZE = 10
PACKET_SIZE = 8
PORT = 1234
ACK = 0
SEQ = 1
PROBABILITY = 0.3

# Create a socket
# socket.AF_INET - address family, IPv4, some otehr possible are AF_INET6, AF_BLUETOOTH, AF_UNIX
# socket.SOCK_STREAM - TCP, conection-based, socket.SOCK_DGRAM - UDP, connectionless, datagrams, socket.SOCK_RAW - raw IP p
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


server_socket.bind((socket.gethostname(), PORT))

server_socket.listen()

client_socket, addr = server_socket.accept()
print("Connection established from " + str(addr[0]))


def injectError(data) :
    err = random.randint(0,1)
    if err:
        data = '1' + str(random.randint(0,8)) + data[2:]
    return data

def extract_frame(data):
    ind = PACKET_SIZE
    msg = ""
    if(data[ind] == '/'):
        msg = data[0:ind]
        ind = ind + 1
    else:
        msg = data[0:ind].rstrip()
        msg = msg[:-1]
    return {
            'pkt_type' : int(data[ind]),

            'packet_no' : int(data[ind+1:]),

            'res' : msg,
            }
```

```python
def createFrame(pkt_type, seq_no, msg):
    msg = msg + "/"
    msg = msg = f"{msg:<{PACKET_SIZE}}"
    message = msg + str(pkt_type) + str(seq_no)
    return f"{len(message):<{HEADERSIZE}}" + message


def receive_message(client_socket):
    try:
        msg_header = client_socket.recv(HEADERSIZE)
        if not len(msg_header):
            return False

        msg_len = int(msg_header.decode('utf-8').strip())
        data = client_socket.recv(msg_len).decode('utf-8')
        return data

    except:
        return False

recv = {}
ack_to_Sent = 0

# frame[i] = packet, packet = length + "--"(header) + "data --"(pkt_size)+type+seq_no
while True:

    client_response = receive_message(client_socket)
    if client_response is False:
        break

    client_response = extract_frame(client_response)

    packet_recv_no = client_response['packet_no']

    if client_response['pkt_type'] == SEQ  and  packet_recv_no == ack_to_Sent :
        recv[packet_recv_no] = client_response['res']
        print("Packet "+ str(ack_to_Sent) + " Received")
        prob = random.uniform(0, 1)
        if prob > PROBABILITY:
            client_socket.send(createFrame(ACK,ack_to_Sent,"ACK").encode('utf-8'))
            print("ACK sent for : " + str(ack_to_Sent) + "\n")
            ack_to_Sent = ack_to_Sent + 1
        else:
            print("ACK for Packet " + str(ack_to_Sent) + " Lost!!\n")
    else:
        print("Invalid Packet Seq No - "+ str(packet_recv_no))
        print("Packet Discarded due not in order\n")
```

```python
recv_data = """
for frame in sorted(list(recv.keys())):
    recv_data += recv[frame]

print("Received data : " + recv_data)
server_socket.close()
```

OUTPUT

```
Connection established from 192.168.1.16
Invalid Packet Seq No - 1
Packet Discarded due not in order

Invalid Packet Seq No - 2
Packet Discarded due not in order

Packet 0 Received
ACK for Packet 0 Lost!!

Invalid Packet Seq No - 1
Packet Discarded due not in order

Invalid Packet Seq No - 2
Packet Discarded due not in order

Invalid Packet Seq No - 3
Packet Discarded due not in order

Packet 0 Received
ACK sent for : 0

Packet 1 Received
ACK sent for : 1

Packet 2 Received
ACK sent for : 2

Packet 3 Received
ACK sent for : 3

Packet 4 Received
ACK for Packet 4 Lost!!

Invalid Packet Seq No - 5
Packet Discarded due not in order

Packet 4 Received
ACK sent for : 4

Packet 5 Received
ACK sent for : 5

Received data : Hellooo how are you let's hope everyone is safe
```

RECEIVER

```
E:\network\Go Back N>python client.py
Enter the data to be sent : Hellooo how are you let's hope everyone is safe
Total Packets = 6

Packet 0 Lost!!

Packet 1 Sent

Packet 2 Sent

Packet 3 Lost!!

Resending packets in the current window -> [0,3] :
Packet 0 Sent
Packet 1 Sent
Packet 2 Sent
Packet 3 Sent

Resending packets in the current window -> [0,3] :
Packet 0 Sent
Packet 1 Sent
Packet 2 Sent
Packet 3 Sent

ACK Received for Packet : 0
Packet 4 Sent

ACK Received for Packet : 1
Packet 5 Sent

ACK Received for Packet : 2
ACK Received for Packet : 3
Resending packets in the current window -> [4,5] :
Packet 4 Sent
Packet 5 Sent

ACK Received for Packet : 4
ACK Received for Packet : 5

All Packets sent
```

SENDER

SELECTIVE REPEAT

CLIENT.py

```
import socket
import random
import time

PORT = 1234
HEADERSIZE = 10
PACKET_SIZE = 8
TIMER = 5
WINDOW_SIZE = 4
PACKET_SIZE = 8
ACK = 0
SEQ = 1
NAK = 2
PROBABILITY = 0.6


sender = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
sender.connect((socket.gethostname(),PORT))

def injectError(data) :
    err = random.randint(0,1)
    if err:
        data = '0' + str(random.randint(0,8)) + data[2:]
    return data

def extract_frame(data):
    ind = PACKET_SIZE
    msg = ""
    if(data[ind] == '/'):
        msg = data[0:ind]
        ind = ind + 1
    else:
        msg = data[0:ind].rstrip()
        msg = msg[:-1]

    return {
            'pkt_type' : int(data[ind]),

            'packet_no' : int(data[ind+1:]),

            'res' : msg,
            }


def createFrame(pkt_type, seq_no, msg):
    msg = msg + "/"
    msg = msg + f"{msg:<{PACKET_SIZE}}"
    message = msg + str(pkt_type) + str(seq_no)
    return f"{len(message):<{HEADERSIZE}}" + message
```

```
def createDataPackets(data) :
    data_frames = []
    l = len(data)
    pkt_no,i = 0,0
    while i < l:
        data_frames.append(createFrame(SEQ,pkt_no,data[i:min(i+PACKET_SIZE,l)]).encode('utf-8'))
        pkt_no = pkt_no + 1
        i = i + PACKET_SIZE
    print("Total Packets = " + str(pkt_no)+"\n")
    return data_frames,pkt_no

def receive_message(client_socket):
    client_socket.settimeout(TIMER)
    try:
        msg_header = client_socket.recv(HEADERSIZE)

        if not len(msg_header):
            return False

        msg_len = int(msg_header.decode('utf-8').strip())

        data = client_socket.recv(msg_len).decode('utf-8')
        client_socket.settimeout(None)
        return data,False

    except:
        client_socket.settimeout(None)
        return False,True

data = input("Enter the data to be sent : ")
frames,last_packet = createDataPackets(data)
WINDOW_SIZE = min(WINDOW_SIZE,last_packet)
base_pkt,last_sent = 0,0
nack_pkt_sent = -1
nack_recv = False

#send length
len_msg = str(last_packet)
len_msg = f"{len(len_msg):<{HEADERSIZE}}" + len_msg
sender.send(len_msg.encode('utf-8'))

while True:
```

```
    while last_sent - base_pkt < WINDOW_SIZE  and  last_sent < last_packet :
        prob = random.uniform(0,1)
        if prob > PROBABILITY:
            sender.send(frames[last_sent])
            print("Packet "+ str(last_sent) + " Sent\n")
        else:
            print("Packet "+ str(last_sent) + " Lost!!\n")
        last_sent = last_sent + 1

    receiver_res,TIMEOUT = receive_message(sender)

    if TIMEOUT:
        send_pkt = base_pkt
        if nack_recv:
            send_pkt = nack_pkt_sent
        print("Timeout Occured\nResending packet " + str(send_pkt) + "\n")
        print("Packet "+ str(send_pkt) + " Sent")
        sender.send(frames[send_pkt])
        print()
    else:
        receiver_res = extract_frame(receiver_res)
        packet_no = receiver_res['packet_no']
        if receiver_res['pkt_type'] == ACK:
            print("Received ACK for Packet : " + str(packet_no))
            if packet_no - base_pkt + 1 == WINDOW_SIZE:
                print("Entire Window ["+str(base_pkt)+","+str(packet_no)+"] Received\n")
            else :
                print()
            base_pkt = packet_no + 1
            nack_recv = False

        elif receiver_res['pkt_type'] == NAK:
            if packet_no >= base_pkt and packet_no < last_packet:
                print("NACK  Received for Packet " + str(packet_no) + "\n")
                print("Resending packet " + str(packet_no) + "\n")
                print("Packet "+ str(packet_no) + " Sent")
                sender.send(frames[packet_no])
                nack_pkt_sent = packet_no
                nack_recv = True
                print()
            else :
                print("Invalid NACK Received\n")


    if base_pkt  == last_packet :
        break

print("\nAll Packets sent")
sender.close()
```

SERVER.py

```
import socket
import random

HEADERSIZE = 10
PACKET_SIZE = 8
WINDOW_SIZE = 4
PORT = 1234
ACK = 0
SEQ = 1
NAK = 2
PROBABILITY = 0.3

# Create a socket
# socket.AF_INET - address family, IPv4, some otehr possible are AF_INET6, AF_BLUETOOTH, AF_UNIX
# socket.SOCK_STREAM - TCP, conection-based, socket.SOCK_DGRAM - UDP, connectionless, datagrams, socket.SOCK_RAW - raw IP packets
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


server_socket.bind((socket.gethostname(), PORT))

server_socket.listen()

client_socket, addr = server_socket.accept()
print("Connection established from " + str(addr[0]))


def injectError(data) :
    err = random.randint(0,1)
    if err:
        data = '1' + str(random.randint(0,8)) + data[2:]
    return data

def extract_frame(data):
    ind = PACKET_SIZE
    msg = ""
    if(data[ind] == '/'):
        msg = data[0:ind]
        ind = ind + 1
    else:
```

```
def createFrame(pkt_type, seq_no, msg):
    msg = msg + "/"
    msg = msg + f"{msg:<{PACKET_SIZE}}"
    message = msg + str(pkt_type) + str(seq_no)
    return f"{len(message):<{HEADERSIZE}}" + message


def receive_message(client_socket):
    try:
        msg_header = client_socket.recv(HEADERSIZE)
        if not len(msg_header):
            return False

        msg_len = int(msg_header.decode('utf-8').strip())
        data = client_socket.recv(msg_len).decode('utf-8')
        return data

    except:
        return False

total_packets = int(receive_message(client_socket))

recv = dict()
last_seen, Rn, end_window = -1,0,min(total_packets,WINDOW_SIZE)
marked = []
for i in range(total_packets+1):
    marked.append(False)

NakSent = False
# frame[i] = packet, packet = length + "--"(header) + "data --"(pkt_size)+type+seq_no
while True:

    client_response = receive_message(client_socket)
    if client_response is False:
        break

    client_response = extract_frame(client_response)

    packet_recv_no = client_response['packet_no']

    if Corrupted(client_response):
        if not NackSent:
            # send Nack for Rn
            NakSent = True
            print("NACK Sent for packet : " + str(Rn))
            client_socket.send(createFrame(NAK,Rn,"NAK").encode('utf-8'))
        continue
```

```
if client_response['pkt_type'] == SEQ :
    last_seen = max(last_seen,packet_recv_no)
    marked[packet_recv_no] = True
    if packet_recv_no == Rn:
        recv[packet_recv_no] = client_response['res']
        print("Packet "+ str(Rn) + " Received")
        if last_seen == Rn :
            #Received in order
            client_socket.send(createFrame(ACK,Rn,"ACK").encode('utf-8'))
            print("ACK sent for : " + str(Rn) + "\n")
            Rn = Rn + 1
            # prob = random.uniform(0, 1)
            # if prob > PROBABILITY:
            #     client_socket.send(createFrame(ACK,Rn,"ACK").encode('utf-8'))
            #     print("ACK sent for : " + str(Rn) + "\n")
            #     Rn = Rn + 1
            # else:
            #     print("ACK for Packet " + str(Rn) + " Lost!!\n")
        else:
            NakSent = False
            nak_packet = end_window
            for pkt in range(Rn,end_window):
                if not marked[pkt]:
                    nak_packet = pkt
                    break
            Rn = nak_packet
            if nak_packet == end_window:
                #Send Ack for current window and shift to new window
                client_socket.send(createFrame(ACK,end_window-1,"ACK").encode('utf-8'))
                print("ACK Sent for Packet " + str(end_window-1) + "\nCurrent Window Received\n")
            else:
                #Send NAK
                NakSent = True
                client_socket.send(createFrame(NAK,nak_packet,"NACK").encode('utf-8'))
                print("NAK Sent for Packet " + str(nak_packet) + "\n")\

    else:
        #Received out of order
        recv[packet_recv_no] = client_response['res']
        print("Received Packet : " + str(packet_recv_no) +  " Out of Order")
        if not NakSent :
            end_window = min(total_packets,Rn + WINDOW_SIZE)
            NakSent = True
            client_socket.send(createFrame(NAK,Rn,"NAK").encode('utf-8'))
            print("NAK Sent for Packet " + str(Rn) + "\n")\
```

```
    else:
        print("Invalid Packet Seq No - "+ str(packet_recv_no))
        print("Packet Discarded due not in order\n")


recv_data = ""
sequence = sorted(list(recv.keys()))
for seq_no in sequence:
    recv_data += recv[seq_no]

print("Received data : " + recv_data)
server_socket.close()
```

OUTPUT

RECEIVER

```
E:\NETWORK\Flow Control\Selective Repeat>python server.py
Connection established from 192.168.1.7
Received Packet : 1 Out of Order
NAK Sent for Packet 0

Received Packet : 2 Out of Order
Packet 0 Received
NAK Sent for Packet 3

Packet 3 Received
ACK sent for : 3

Packet 4 Received
ACK sent for : 4

Packet 5 Received
ACK sent for : 5

Received data : hello hope everything is going good and fine

E:\NETWORK\Flow Control\Selective Repeat>
```

SENDER

```
E:\NETWORK\Flow Control\Selective Repeat>python client.py
Enter the data to be sent : hello hope everything is going good and fine
Total Packets = 6

Packet 0 Lost!!

Packet 1 Sent

Packet 2 Sent

Packet 3 Lost!!

NACK  Received for Packet 0

Resending packet 0

Packet 0 Sent

NACK  Received for Packet 3

Resending packet 3

Packet 3 Sent

Received ACK for Packet : 3
Entire Window [0,3] Received
```

```
Packet 4 Lost!!

Packet 5 Lost!!

Timeout Occured
Resending packet 4

Packet 4 Sent

Received ACK for Packet : 4

Timeout Occured
Resending packet 5

Packet 5 Sent

Received ACK for Packet : 5


All Packets sent

E:\NETWORK\Flow Control\Selective Repeat>
```

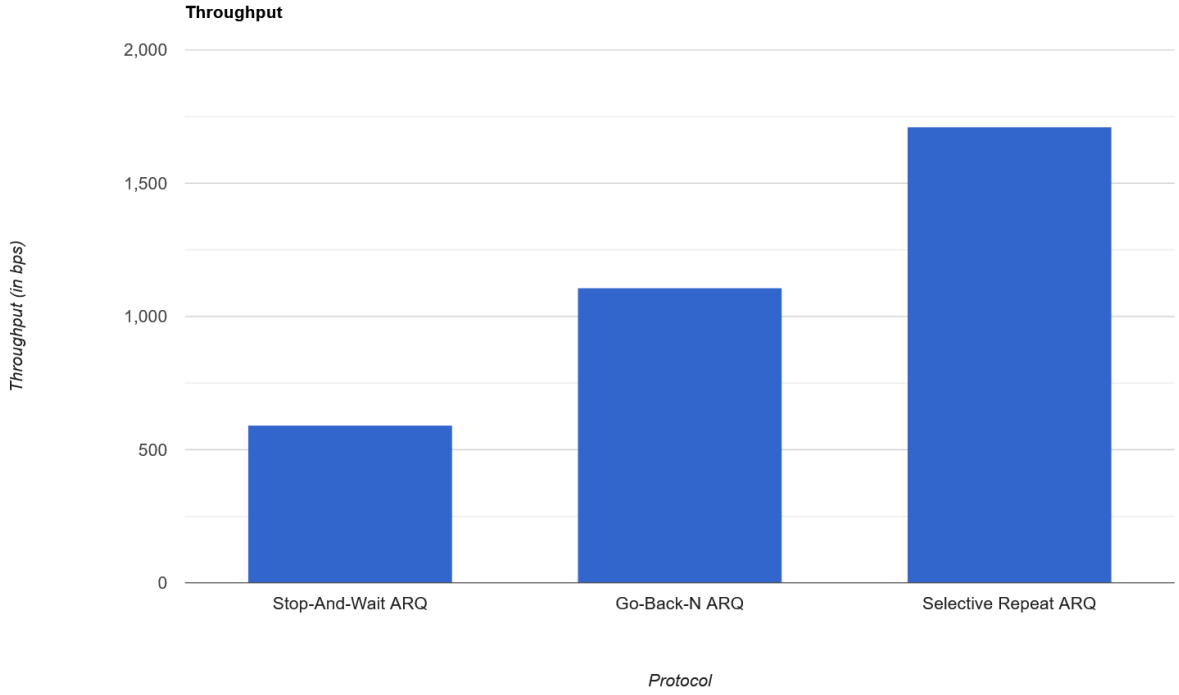
RESULTS

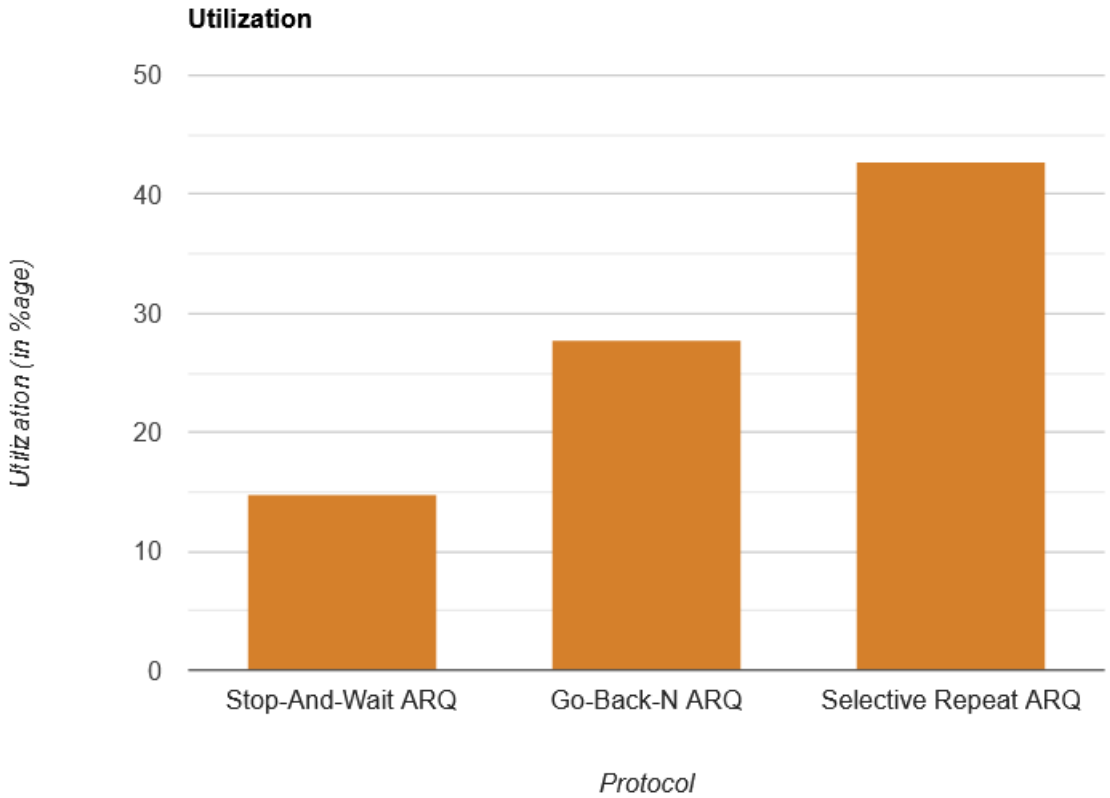
The assumed bandwidth was 4000bps.

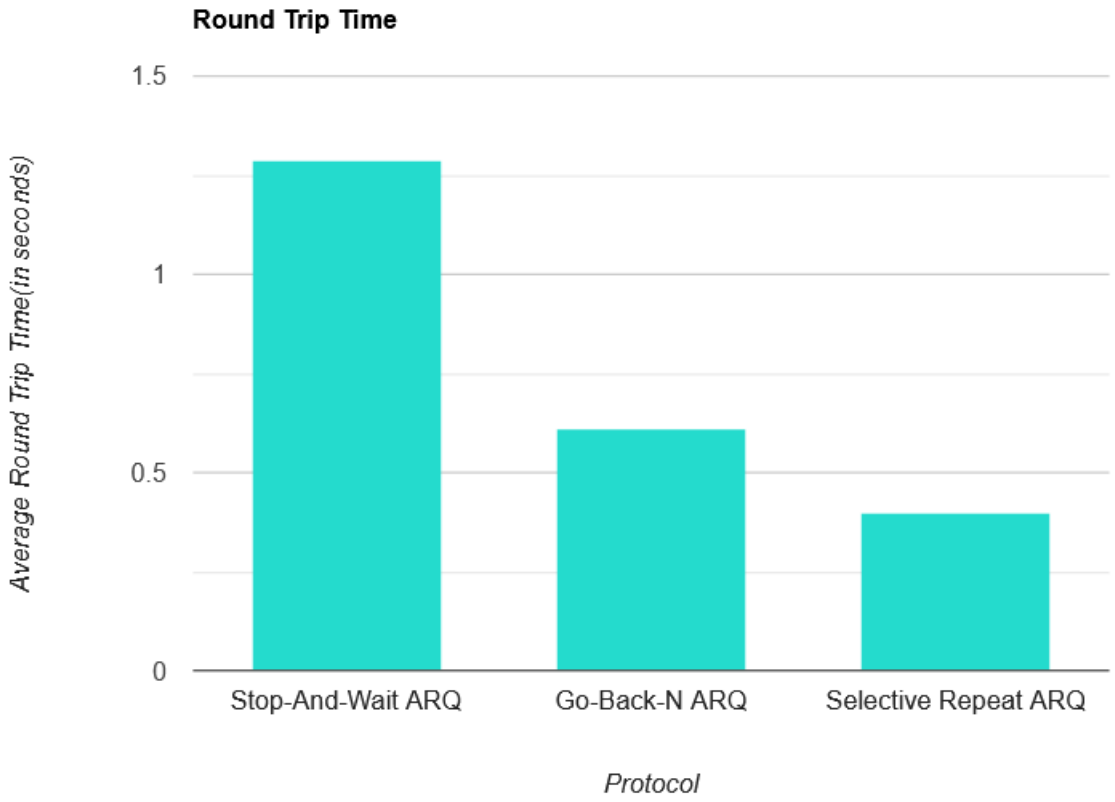
	Stop-And-Wait ARQ					
	Effective Frames Sent	Total Frames Sent	Total Time Taken	Throughput	Utilization	Average time for transmission of a Packet
	16	27	0.449397	341	8.54	1.685239
	16	28	0.466057	329	8.24	1.747712
	16	17	0.11019	1393	34.85	0.413214
	16	22	0.287949	533	13.34	1.079807
	16	26	0.413903	371	9.28	1.552136
AVERAGE	16	24	0.3454992	593.4	14.85	1.2956216
			(in minutes)	(in bits per second)	(in %age)	(in seconds)

	Go-Back-N ARQ					
	Effective Frames Sent	Total Frames Sent	Total Time Taken	Throughput	Utilization	Average time for transmission of a Packet
	16	25	0.068804	2232	55.81	0.258014
	16	44	0.182762	840	21.01	0.685358
	16	45	0.196717	780	19.52	0.737687
	16	45	0.19531	786	19.66	0.732411
	16	44	0.169846	904	22.61	0.636922
AVERAGE	16	40.6	0.1626878	1108.4	27.722	0.6100784
			(in minutes)	(in bits per second)	(in %age)	(in seconds)

	Selective Repeat ARQ					
	Effective Frames Sent	Total Frames Sent	Total Time Taken	Throughput	Utilization	Average time for transmission of a Packet
	16	26	0.098219	1563	39.1	0.36832
	16	33	0.14629	1049	26.25	0.548586
	16	16	0.046945	3271	81.8	0.176044
	16	33	0.147066	1044	26.12	0.551271
	16	29	0.094503	1625	40.63	0.354388
AVERAGE	16	27.4	0.1066046	1710.4	42.78	0.3997218
			(in minutes)	(in bits per second)	(in %age)	(in seconds)


Throughput
2,000
1,500
1,000
500
0
Throughput (in bps)
Stop-And-Wait ARQ
Go-Back-N ARQ
Selective Repeat ARQ
Protocol


Utilization
50
40
30
20
10
0
Utilization (in %age)
Stop-And-Wait ARQ
Go-Back-N ARQ
Selective Repeat ARQ
Protocol



Stop-And-Wait is memory efficient as the sequence numbers are only 0 and 1 and thus, keeps a copy of just 1 sent frame. If the channel is thick and long, the potential of the channel is wasted because we are just waiting for an ACK from the receiver, whereas we could have sent a few packets lined up next too at the same time. This would boost the throughput to a great extent.

The need to utilize more of the channel brings us to **Go-Back-N ARQ**, where we send many frames before waiting for ACK. This ensures that many frames are in transit at the same time, which is desired when the bandwidth-delay product is high. But here the receiver needs to accept the frames in order. So a timer is maintained on the sender side to resend the frames, in case the frame or ACK was lost during transit and thus the frame was either not acknowledged or the sender didn't receive the ACK. Whenever such happens, all the frames from the last acknowledged frames are resent by the sender.

In **Selective Repeat ARQ**, multiple frames are in transit and the channel is also utilized well. The improvement here is that the receiver can accept the frames in any order. It just needs to make sure that the data is delivered to the file accurately. As a result, the frames within a window can be acknowledged in any order. 1 NAK can inform regarding the last missing packet and 1 ACK can serve as ACK for the previously received ACKs as well because an ACK is transferred only when the frames are converted in order to message and delivered to the file. This releases contention on the channel. But the out-of-order hack necessitates individual timers, so more memory overhead is present on the sender side and special care must be given to synchronization issues.

COMMENTS

From this assignment, I understood the implementation of various flow control protocols in great detail and also the challenges faced in implementing them. I learned a great deal about socket programming and multithreading as well.