**Network Topologies, Hamming Code and CRC**

1. **RING TOPOLOGY**

**CODE:**

#Create a simulator object

set ns [new Simulator]

#Routing Protocol used is Distance Vector

$ns rtproto DV

#Open the nam trace file

set nf [open ringTopology.nam w]

$ns namtrace-all $nf

#Define a 'finish' procedure

proc finish {} {

 global ns nf

 $ns flush-trace

 #Close the trace file

 close $nf

 #Executenam on the trace file

 exec nam ringTopology.nam &

 exit 0

}

#Create nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

set n4 [$ns node]

set n5 [$ns node]

#Create links between the nodes

$ns duplex-link $n0 $n1 1Mb 10ms DropTail

$ns duplex-link $n1 $n2 1Mb 10ms DropTail

$ns duplex-link $n2 $n3 1Mb 10ms DropTail

$ns duplex-link $n3 $n4 1Mb 10ms DropTail

$ns duplex-link $n4 $n5 1Mb 10ms DropTail

$ns duplex-link $n5 $n0 1Mb 10ms DropTail

$ns duplex-link-op $n0 $n1 orient right-up

$ns duplex-link-op $n1 $n2 orient right-down

$ns duplex-link-op $n2 $n3 orient down

$ns duplex-link-op $n3 $n4 orient left-down

$ns duplex-link-op $n4 $n5 orient left-up

$ns duplex-link-op $n5 $n0 orient up

#Create a TCP agent and attach it to node n0

set tcp0 [new Agent/TCP]

$tcp0 set class\_ 1

$ns attach-agent $n1 $tcp0

#Create a TCP Sink agent (a traffic sink) for TCP and attach it to node n3

set sink0 [new Agent/TCPSink]

$ns attach-agent $n3 $sink0

#Connect the traffic sources with the traffic sink

$ns connect $tcp0 $sink0

# Create a CBR traffic source and attach it to tcp0

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize\_ 500

$cbr0 set interval\_ 0.01

$cbr0 attach-agent $tcp0

#Schedule events for the CBR agents

$ns at 0.5 "$cbr0 start"

# Break link n2---n3 at 1.3ms

$ns rtmodel-at 1.3 down $n2 $n3

$ns at 4.5 "$cbr0 stop"

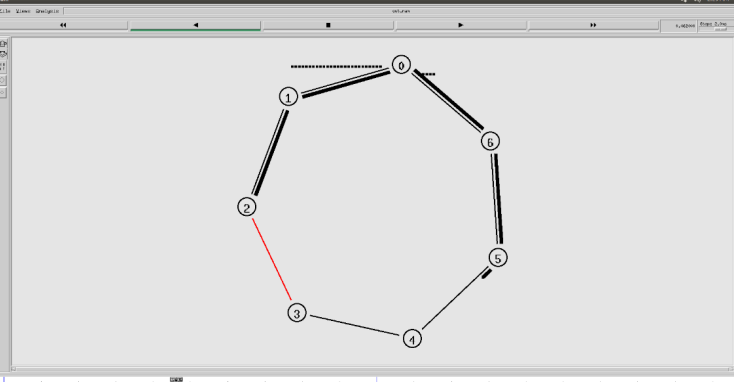
#Call the finish procedure after 5 seconds of simulation time

$ns at 5.0 "finish"

#Run the simulation

$ns run

**OUTPUT:**



1. **STAR TOPOLOGY**

**CODE:**

#Create a simulator object

set ns [new Simulator]

#Routing Protocol used is Distance Vector

$ns rtproto DV

#Open the nam trace file

set nf [open out.nam w]

$ns namtrace-all $nf

#Define a 'finish' procedure

proc finish {} {

 global ns nf

 $ns flush-trace

 #Close the trace file

 close $nf

 #Executenam on the trace file

 exec nam out.nam &

 exit 0

}

# Creating five nodes

set node1 [$ns node]

set node2 [$ns node]

set node3 [$ns node]

set node4 [$ns node]

set node5 [$ns node]

# Creating a hub

set hub [$ns node]

# Creating point-to-point connections from nodes to hub

$ns duplex-link $node1 $hub 100Mb 20ms DropTail

$ns duplex-link $node2 $hub 100Mb 20ms DropTail

$ns duplex-link $node3 $hub 100Mb 20ms DropTail

$ns duplex-link $node4 $hub 100Mb 20ms DropTail

$ns duplex-link $node5 $hub 100Mb 20ms DropTail

# Creating a TCP agent and attaching it to node 1

set tcp0 [new Agent/TCP]

$tcp0 set class\_ 1

$ns attach-agent $node1 $tcp0

# Creating a TCP Sink agent for TCP and attaching it to node 3

set sink0 [new Agent/TCPSink]

$ns attach-agent $node3 $sink0

# Connecting the traffic sources with the traffic sink

$ns connect $tcp0 $sink0

# Creating a CBR traffic source and attach it to tcp0

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize\_ 500

$cbr0 set interval\_ 0.05

$cbr0 attach-agent $tcp0

# Schedule events for the CBR agents

$ns at 0.5 "$cbr0 start"

$ns at 5.5 "$cbr0 stop"

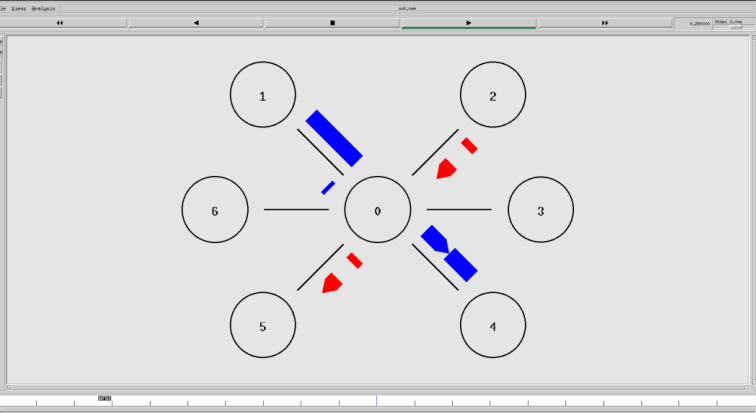
# Here we call the finish procedure after 10 seconds of simulation time

$ns at 10.0 "finish"

# Finally, run the simulation

$ns run

**OUTPUT:**

****

1. **MESH TOPOLOGY:**

**CODE:**

#Create a simulator object

set ns [new Simulator]

#Routing Protocol used is Distance Vector

$ns rtproto DV

#Open the nam trace file

set nf [open mesh.nam w]

$ns namtrace-all $nf

#Define a 'finish' procedure

proc finish {} {

 global ns nf

 $ns flush-trace

 #Close the trace file

 close $nf

 #Executenam on the trace file

 exec nam mesh.nam &

 exit0

}

# Creating five nodes

set node1 [$ns node]

set node2 [$ns node]

set node3 [$ns node]

set node4 [$ns node]

set node5 [$ns node]

# Creating point-to-point connections between all nodes

$ns duplex-link $node1 $node2 100Mb 20ms DropTail

$ns duplex-link $node1 $node3 100Mb 20ms DropTail

$ns duplex-link $node1 $node4 100Mb 20ms DropTail

$ns duplex-link $node1 $node5 100Mb 20ms DropTail

$ns duplex-link $node2 $node3 100Mb 20ms DropTail

$ns duplex-link $node2 $node4 100Mb 20ms DropTail

$ns duplex-link $node2 $node5 100Mb 20ms DropTail

$ns duplex-link $node3 $node4 100Mb 20ms DropTail

$ns duplex-link $node3 $node5 100Mb 20ms DropTail

$ns duplex-link $node4 $node5 100Mb 20ms DropTail

# Creating a TCP agent and attaching it to node 1

set tcp0 [new Agent/TCP]

$tcp0 set class\_ 1

$ns attach-agent $node1 $tcp0

# Creating a TCP Sink agent for TCP and attaching it to node 3

set sink0 [new Agent/TCPSink]

$ns attach-agent $node3 $sink0

# Connecting the traffic sources with the traffic sink

$ns connect $tcp0 $sink0

# Creating a CBR traffic source and attach it to tcp0

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize\_ 500

$cbr0 set interval\_ 0.05

$cbr0 attach-agent $tcp0

# Schedule events for the CBR agents

$ns at 0.5 "$cbr0 start"

$ns at 5.5 "$cbr0 stop"

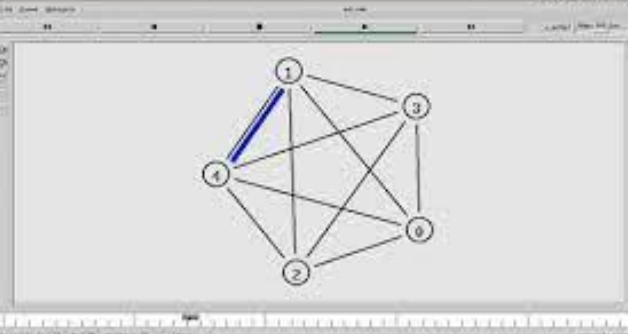
# Here we call the finish procedure after 10 seconds of simulation time

$ns at 10.0 "finish"

# Finally, run the simulation

$ns run

**OUTPUT:**

****

**Hamming Code**

**Code in python**

# Hamming Code in Python

# By - Pranav Singh Mahara

#       21BBS0188

option=int(input('\nChose option\n1 for generating hamming code \n2 for finding error in hamming code\nEnter your choice : \n'))

# Section to generate hamming code

if(option==1):

    print('Enter the data bits : ')

    d=input()

    data=list(d)

    data.reverse()

    c,ch,j,r,h=0,0,0,0,[]

    while ((len(d)+r+1)>(pow(2,r))):

        r=r+1

    for i in range(0,(r+len(data))):

        p=(2\*\*c)

        if(p==(i+1)):

            h.append(0)

            c=c+1

        else:

            h.append(int(data[j]))

            j=j+1

    for parity in range(0,(len(h))):

        ph=(2\*\*ch)

        if(ph==(parity+1)):

            startIndex=ph-1

            i=startIndex

            toXor=[]

            while(i<len(h)):

                block=h[i:i+ph]

                toXor.extend(block)

                i+=2\*ph

            for z in range(1,len(toXor)):

                h[startIndex]=h[startIndex]^toXor[z]

            ch+=1

    h.reverse()

    print('Hamming code generated would be : ', end="")

    print(int(''.join(map(str, h))))

# Section to detect error

elif(option==2):

    print('Enter the hamming code received : ')

    d=input()

    data=list(d)

    data.reverse()

    c,ch,j,r,error,h,parity\_list,h\_copy=0,0,0,0,0,[],[],[]

    for k in range(0,len(data)):

        p=(2\*\*c)

        h.append(int(data[k]))

        h\_copy.append(data[k])

        if(p==(k+1)):

            c=c+1

    for parity in range(0,(len(h))):

        ph=(2\*\*ch)

        if(ph==(parity+1)):

            startIndex=ph-1

            i=startIndex

            toXor=[]

            while(i<len(h)):

                block=h[i:i+ph]

                toXor.extend(block)

                i+=2\*ph

            for z in range(1,len(toXor)):

                h[startIndex]=h[startIndex]^toXor[z]

            parity\_list.append(h[parity])

            ch+=1

    parity\_list.reverse()

    error=sum(int(parity\_list) \* (2 \*\* i) for i, parity\_list in enumerate(parity\_list[::-1]))

    if((error)==0):

        print('There is no error in the hamming code received.')

    elif((error)>=len(h\_copy)):

        print('Error cannot be detected.')

    else:

        print('Error is in',error,'bit')

        if(h\_copy[error-1]=='0'):

            h\_copy[error-1]='1'

        elif(h\_copy[error-1]=='1'):

            h\_copy[error-1]='0'

            print('After correction hamming code is : ')

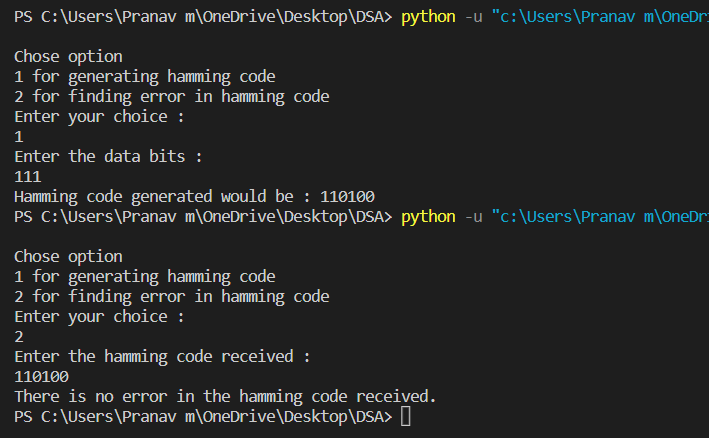
        h\_copy.reverse()

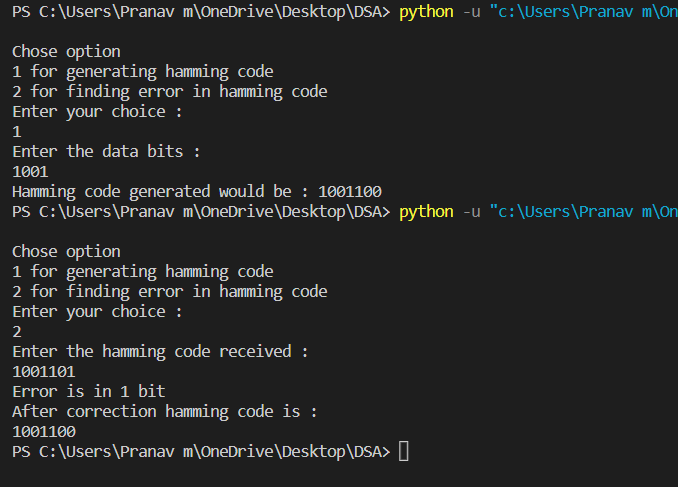
        print(int(''.join(map(str, h\_copy))))

else:

    print('Option entered does not exist')

**SCREENSHOT**





**CRC Code**

**Code in python**

# python code for crc error detecion

# done by Pranav Singh Mahara

def remainder(x, y):

    result = []

    for i in range(1, len(y)):

        if x[i] == y[i]:

            result.append('0')

        else:

            result.append('1')

    return ''.join(result)

def func(dividend, divisor):

    pick = len(divisor)

    tmp = dividend[0: pick]

    while pick < len(dividend):

        if tmp[0] == '1':

            tmp = remainder(divisor, tmp) + dividend[pick]

        else:

            tmp = remainder('0'\*pick, tmp) + dividend[pick]

        pick += 1

    if tmp[0] == '1':

        tmp = remainder(divisor, tmp)

    else:

        tmp = remainder('0'\*pick, tmp)

    checkword = tmp

    return checkword

def CRC(data, key):

    l\_key = len(key)

    appended\_data = data + '0'\*(l\_key-1)

    remainder = func(appended\_data, key)

    codeword = data + remainder

    print("Remainder : ", remainder)

    print("Encoded Data (Data + Remainder) : ",

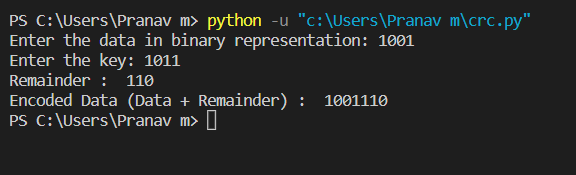
          codeword)

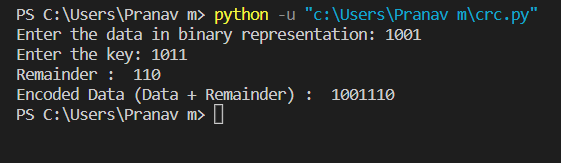
data = input("Enter the data in binary representation: ")

key = input("Enter the key: ")

CRC(data,key)

**Output of code**





xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

**Part -1**

**Create Socket Server with Multiple Clients in Python**

Server Code

import socket

import os

from \_thread import \*

ServerSideSocket = socket.socket()

host = socket.gethostname()

port = 2004

ThreadCount = 0

try:

    ServerSideSocket.bind((host, port))

except socket.error as e:

    print(str(e))

print('Socket is listening..')

ServerSideSocket.listen(5)

import socket, threading

class ClientThread(threading.Thread):

    def \_\_init\_\_(self,clientAddress,clientsocket):

        threading.Thread.\_\_init\_\_(self)

        self.csocket = clientsocket

        print ("New connection added: ", clientAddress)

    def run(self):

        print ("Connection from : ", clientAddress)

        #self.csocket.send(bytes("Hi, This is from Server..",'utf-8'))

        msg = ''

        while True:

            data = self.csocket.recv(2048)

            msg = data.decode()

            if msg=='bye':

              break

            print ("from client", msg)

            self.csocket.send(bytes(msg,'UTF-8'))

        print ("Client at ", clientAddress , " disconnected...")

LOCALHOST = "127.0.0.1"

PORT = 8080

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

server.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

server.bind((LOCALHOST, PORT))

print("Server started")

print("Waiting for client request..")

while True:

    server.listen(1)

    clientsock, clientAddress = server.accept()

    newthread = ClientThread(clientAddress, clientsock)

    newthread.start()

Client Code

import socket

SERVER = "127.0.0.1"

PORT = 8080

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

client.connect((SERVER, PORT))

client.sendall(bytes("This is from Client",'UTF-8'))

while True:

    in\_data =  client.recv(1024)

    print("From Server :" ,in\_data.decode())

    out\_data = input()

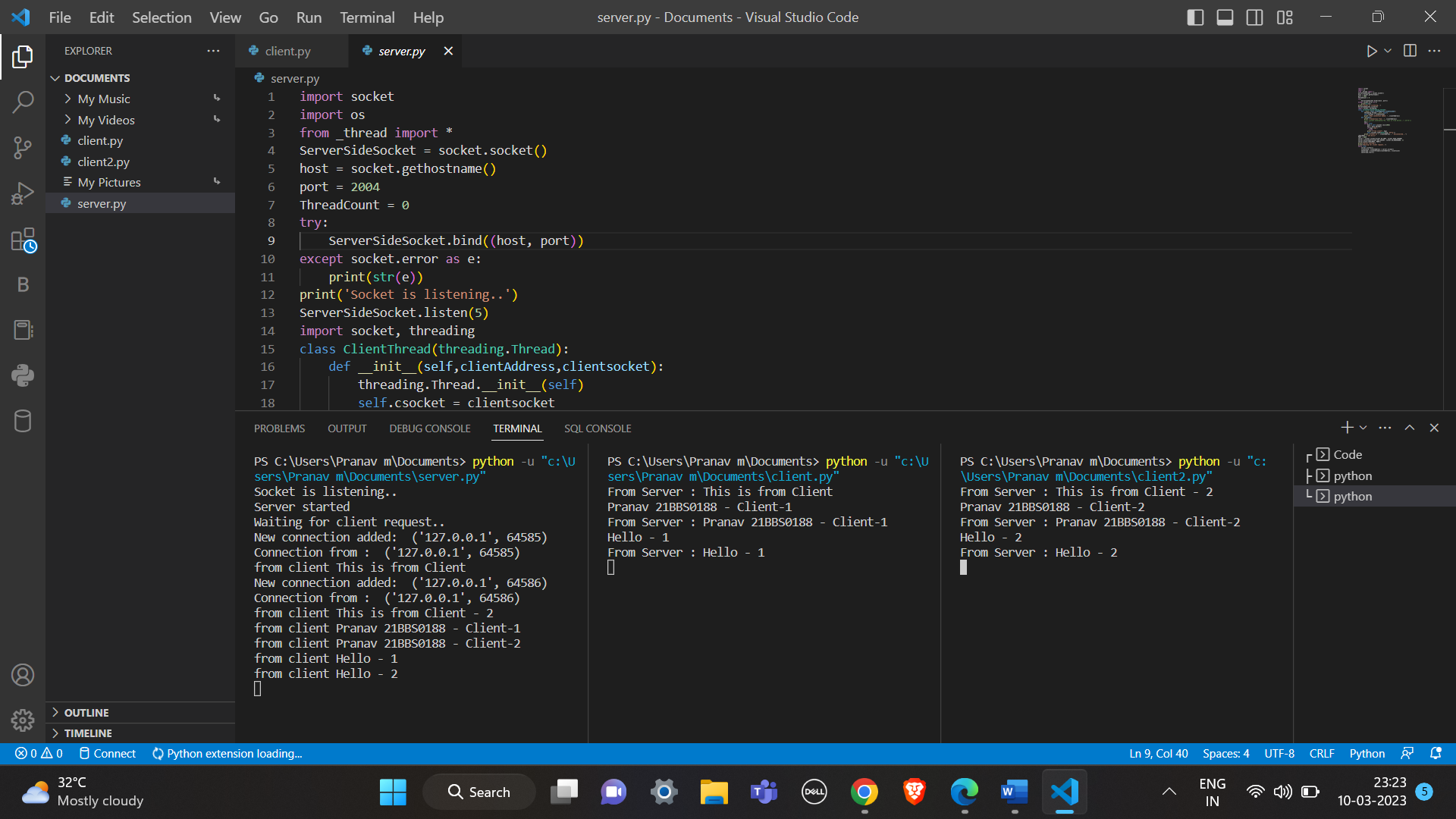
    client.sendall(bytes(out\_data,'UTF-8'))

    if out\_data=='bye':

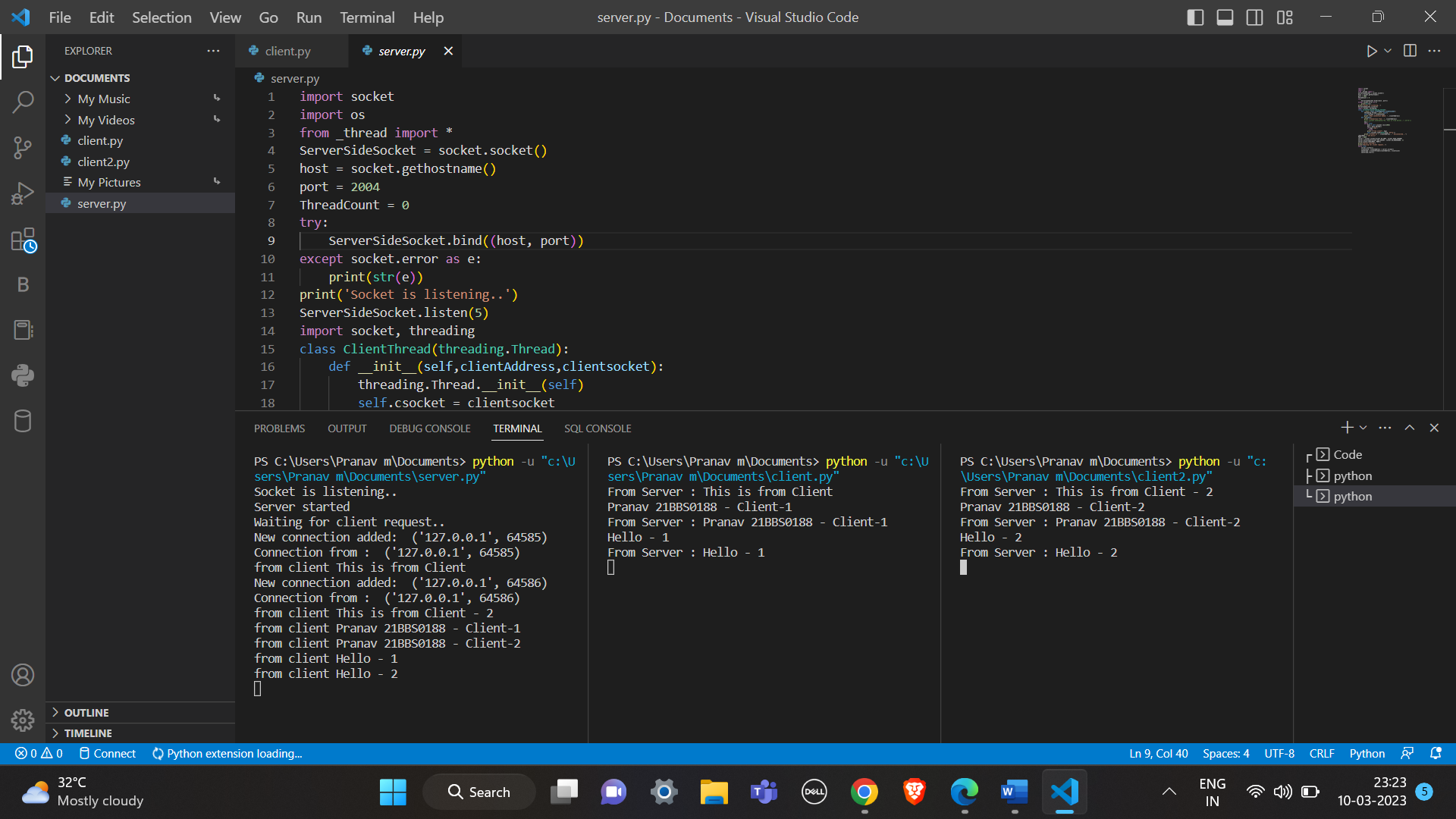
        break

client.close()

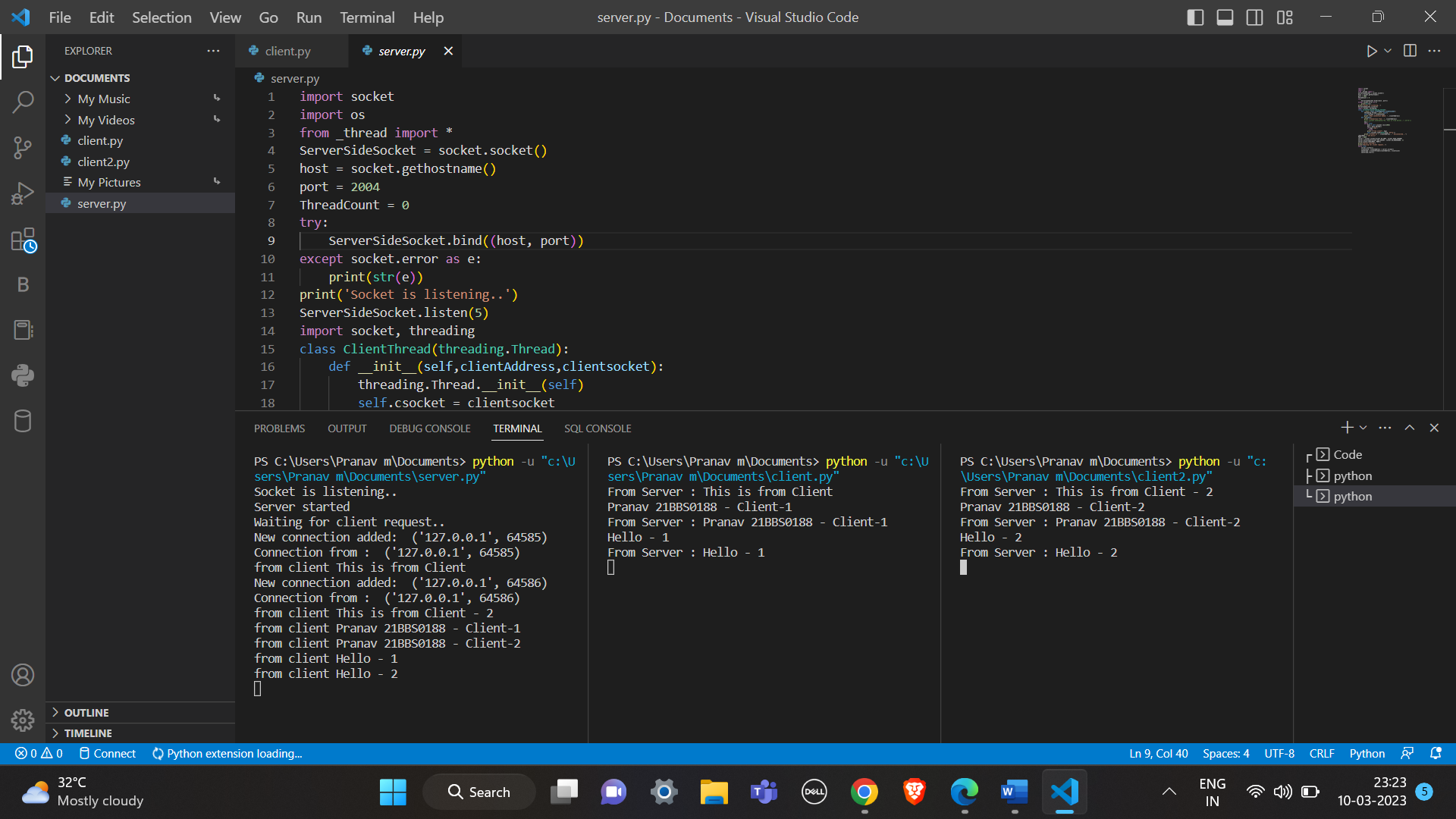
Output



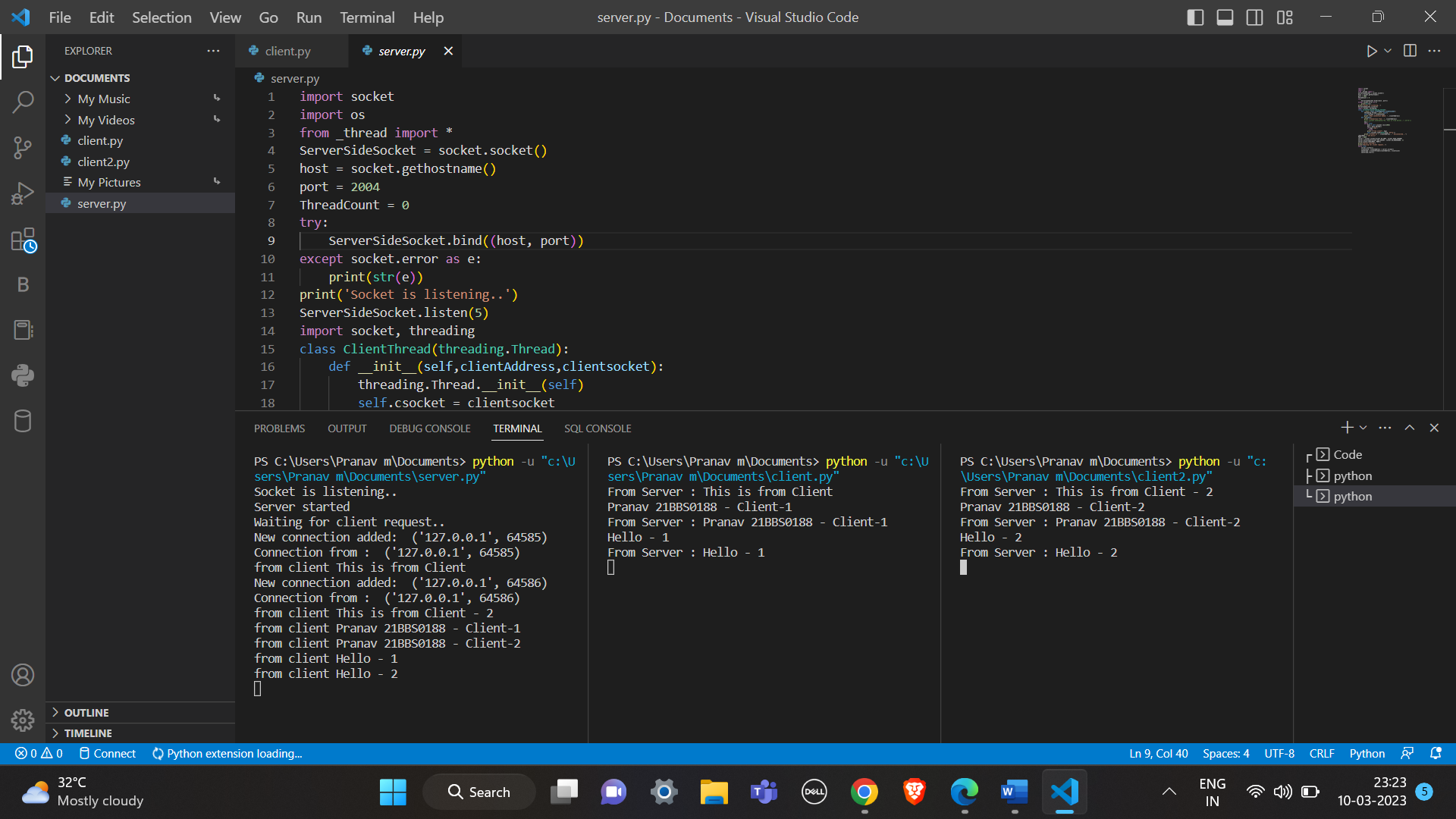
Server



Client-1



Client-2



**Part -2**

**Control Flow**

Stop and Wait Protocol Code

import time

def sender(message):

    print("Sender: Message to send: ", message)

    while True:

        time.sleep(1)

        print("Sender: Sending message to receiver...")

        receiver\_response = receiver(message)

        if receiver\_response == "ACK":

            print("Sender: Receiver ACKed the message.")

            break

        else:

            print("Sender: Receiver NAKed the message. Resending...")

            continue

def receiver(message):

    print("Receiver: Message received: ", message)

    time.sleep(1)

    if is\_corrupt():

        print("Receiver: Message corrupted. Sending NAK.")

        return "NAK"

    else:

        print("Receiver: Message OK. Sending ACK.")

        return "ACK"

def is\_corrupt():

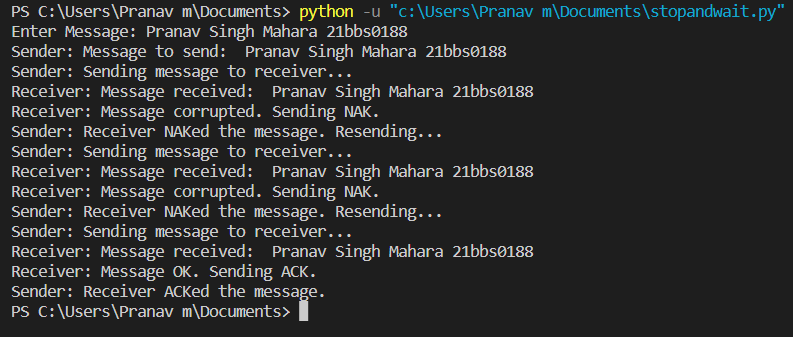
    import random

    return random.random() < 0.5

x=input("Enter Message: ")

sender(x)

Output



Selective ARQ Code

import socket

import threading

import time

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

server\_socket.bind(('0.0.0.0', 1234))

server\_socket.setblocking(False)

packet\_size = 1024

timeout = 1.0

seq\_num = 0

ack\_num = 0

window\_size = 4

window = {}

def send\_packet(seq\_num):

    packet = str(seq\_num).encode()

    server\_socket.sendto(packet, client\_address)

    start\_time = time.time()

    window[seq\_num] = (packet, start\_time)

def receive\_ack(ack\_num):

    while True:

        try:

            ack\_packet, address = server\_socket.recvfrom(packet\_size)

            ack\_seq\_num = int(ack\_packet.decode())

            if ack\_seq\_num == ack\_num:

                return True

        except socket.error:

            for seq\_num, (packet, start\_time) in window.items():

                if time.time() - start\_time > timeout:

                    send\_packet(seq\_num)

            return False

while True:

    data, client\_address = server\_socket.recvfrom(packet\_size)

    seq\_num = int(data.decode())

    if seq\_num >= ack\_num and seq\_num < ack\_num + window\_size:

        server\_socket.sendto(data, client\_address)

        ack\_num = seq\_num + 1

        for i in range(seq\_num, ack\_num):

            window.pop(i, None)

        for i in range(ack\_num, min(ack\_num + window\_size, seq\_num + window\_size)):

            threading.Thread(target=receive\_ack, args=(i,)).start()

        for i in range(ack\_num + len(window), min(seq\_num + window\_size, ack\_num + window\_size)):

            send\_packet(i)

Go Back N ARQ Code

import random

def transmission(i, N, tf):

    tt = 0

    while i <= tf:

        z = 0

        for k in range(i, min(i + N, tf+1)):

            print(f"Sending Frame {k}...")

            tt += 1

        for k in range(i, min(i + N, tf+1)):

            f = random.randint(0, 1)

            if not f:

                print(f"Acknowledgment for Frame {k}...")

                z += 1

            else:

                print(f"Timeout!! Frame Number : {k} Not Received")

                print("Retransmitting Window...")

                break

        print()

        i += z

    print(f"Total number of frames which were sent and resent are: {tt}")

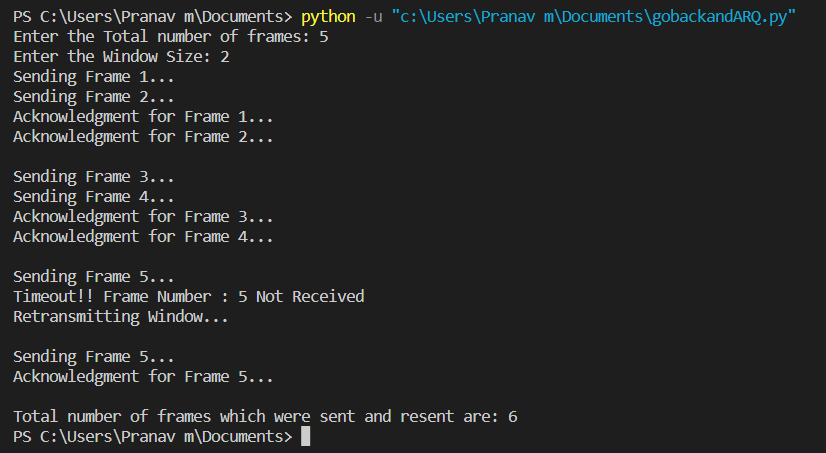
tf = int(input("Enter the Total number of frames: "))

N = int(input("Enter the Window Size: "))

i = 1

transmission(i, N, tf)

Output



xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

**Classful Addressing**

This code uses the Ip address module to create an Ip address object and then determines the class of the IP address based on its first octet. It also checks if the IP address is private, loopback, link-local, multicast, or reserved.

Code

print("Pranav Singh Mahara - 21BBS0188")

def findClass(ip):

    if (ip[0] >= 0 and ip[0] <= 127):

        return "A"

    elif (ip[0] >= 128 and ip[0] <= 191):

        return "B"

    elif (ip[0] >= 192 and ip[0] <= 223):

        return "C"

    elif (ip[0] >= 224 and ip[0] <= 239):

        return "D"

    else:

        return "E"

def separate(ip, className):

    if (className == "A"):

        print("Network Address is : ", ip[0])

        print("Host Address is : ", ".".join(ip[1:4]))

    elif (className == "B"):

        print("Network Address is : ", ".".join(ip[0:2]))

        print("Host Address is : ", ".".join(ip[2:4]))

    elif (className == "C"):

        print("Network Address is : ", ".".join(ip[0:3]))

        print("Host Address is : ", ip[3])

    else:

        print("In this Class, IP address is not divided into Network and Host ID")

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

    ip = input("Enter IP Address : ")

    ip = ip.split(".")

    ip = [int(i) for i in ip]

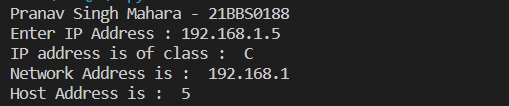
    networkClass = findClass(ip)

    print("IP address is of class : ", networkClass)

    ip = [str(i) for i in ip]

    separate(ip, networkClass)

Output



**Classless Addressing**

In this example, we use the Ip network function to create an Ip address object that includes the IP address and the prefix length as a single string separated by a forward slash, such as "172.16.0.0/25". We then use the prefixlen attribute of the Ip address object to determine the prefix length and the class of the IP address.

Code

print("Pranav Singh Mahara - 21BBS0188")

import ipaddress

ip\_net = ipaddress.IPv4Network('172.16.0.0/25')

print(f"Number of subnets: {ip\_net.num\_addresses//(2\*\*(32-ip\_net.prefixlen))}")

print(f"Number of hosts per subnet: {2\*\*(32-ip\_net.prefixlen)-2}")

subnet0 = ip\_net[0]

print(f"\nFirst subnet:")

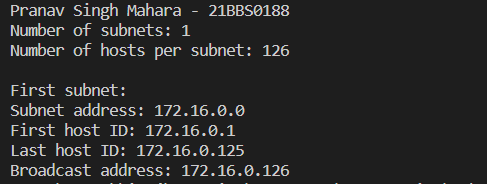
print(f"Subnet address: {subnet0}")

print(f"First host ID: {subnet0+1}")

print(f"Last host ID: {subnet0+(2\*\*(32-ip\_net.prefixlen)-3)}")

print(f"Broadcast address: {subnet0+(2\*\*(32-ip\_net.prefixlen)-2)}")

Output



**SELECTIVE ARQ**

Selective Repeat ARQ (Automatic Repeat Request) is a protocol used for reliable data transmission in computer networks. It allows for retransmission of only those packets that were lost or corrupted during transmission, rather than retransmitting the entire sequence of packets.

CODE:

import socket

import threading

import time

# Define constants for packet size, buffer size, and timeout

PACKET\_SIZE = 1024

BUFFER\_SIZE = 1024

TIMEOUT = 1

# Define server IP address and port number

SERVER\_IP = '127.0.0.1'

SERVER\_PORT = 8080

# Define client IP address and port number

CLIENT\_IP = '127.0.0.1'

CLIENT\_PORT = 8081

# Define sequence number and window size

SEQ\_NUM = 0

WINDOW\_SIZE = 4

# Define a list to store received packets

rcv\_packets = [None] \* WINDOW\_SIZE

# Define a function to send packets to the server

def send\_packets(sock):

    global SEQ\_NUM

    while True:

        # Check if the window is full

        if None not in rcv\_packets:

            time.sleep(0.1)

            continue

        # Send packets in the window

        for i in range(WINDOW\_SIZE):

            if rcv\_packets[i] is None:

                # Get user input for packet data

                data = input('Enter data for packet: ')

                pkt = str(SEQ\_NUM).encode() + b' ' + data.encode()

                sock.sendto(pkt, (SERVER\_IP, SERVER\_PORT))

                print(f'Sent packet with SEQ={SEQ\_NUM}')

                SEQ\_NUM = (SEQ\_NUM + 1) % WINDOW\_SIZE

                time.sleep(0.1)

# Define a function to receive packets from the server

def recv\_packets(sock):

    global rcv\_packets

    while True:

        # Receive packets from the server

        pkt, \_ = sock.recvfrom(PACKET\_SIZE)

        seq\_num = int(pkt.split()[0])

        print(f'Received packet with SEQ={seq\_num}')

        # Check if the packet is within the window

        if rcv\_packets[seq\_num] is None:

            rcv\_packets[seq\_num] = pkt

        # Send an ACK for the packet

        ack\_pkt = str(seq\_num).encode() + b' ACK'

        sock.sendto(ack\_pkt, (SERVER\_IP, SERVER\_PORT))

        print(f'Sent ACK for packet with SEQ={seq\_num}')

# Create a socket for the client

client\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

client\_sock.bind((CLIENT\_IP, CLIENT\_PORT))

# Create threads for sending and receiving packets

send\_thread = threading.Thread(target=send\_packets, args=(client\_sock,))

recv\_thread = threading.Thread(target=recv\_packets, args=(client\_sock,))

send\_thread.start()

recv\_thread.start()

# Wait for the threads to finish

send\_thread.join()

recv\_thread.join()

Output

