NETWORKS CASE STUDY

Chat Server - Group 17

May 2021

TODAY'S PRESENTATION

WHAT WE'LL DISCUSS

Problem Definition
Why Networking is Required for the Application
Operating Systems used
Architecture diagram
Performance Parameters
Analytical Reasoning
Bibliography
Contributions
Code Snippets
Go back n Code snippets

PROBLEM STATEMENT

To understand the working of networks in a chat server,

PROBLEM DEFINITION

Chat Server is a platform where like-minded people can collaborate, without having the need to authenticate themselves in a place (called chat rooms), and discuss about any topic with privacy. The individual chat servers are connected to each other and thus people can hop from one room to another with ease.

WHY NETWORKING IS REQUIRED FOR THE APPLICATION

- To allow users to share information and communicate with each other.
- To help like-minded people collaborate with each other.
- To help maintain privacy and establish communication anonymously.
- To have a reliable communication channel i.e a TCP(Transmission Control Protocol), etc.,
- To secure the application and provide remote access to users

OPERATING SYSTEMS USED

a.) Server: -
- i.) Windows (ex. mIRC, etc.,)
- ii.) Linux (ex. Xchat)
- iii.) Mac OS X
- iv.) Chrome OS

b.) Client: -

- i.) Windows (ex. mIRC, Bersirc, ChatZilla, etc.,)
- ii.) Linux (ex. Xchat, HexChat, etc.,)
- iii.) Mac OS X (ex. IRCle, ChatZilla, Fire, X-Chat Aqua, Conversation, etc.,)
- iv.) Android (ex. AndChat, Holo IRC, IRC Cloud, etc.,)
- v.) IOS (ex. IglooIRC, Palaver IRC, Colloquy, etc.,)
- vi.) Chrome OS (ex. CIRC, Byrd, etc.,)

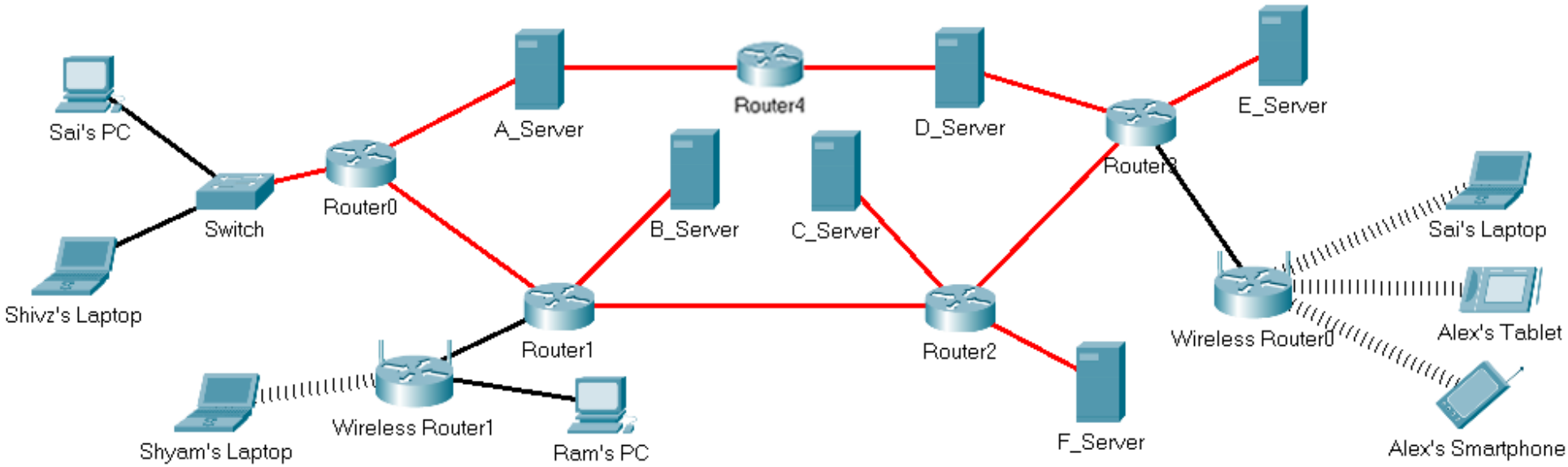
Cloud Services

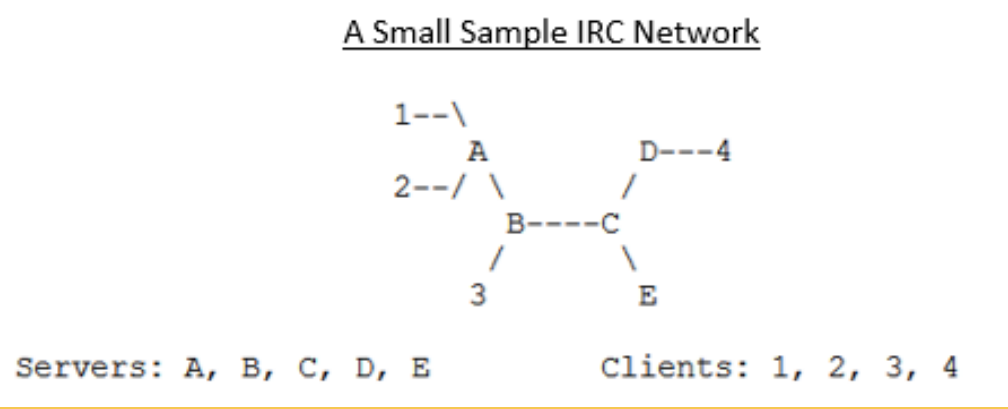
The traditional IRC does not require any cloud services but a few modern-day IRC's are using a cloud-based approach and are quite successful. Ex. Amazon AWS used by IRCCloud (an IRC application using modern approaches), Azure, etc,.

Programming Languages

C, C++, Python, Java, mIRC scripting language (mSL)
Lisp, etc.,

ARCHITECTURE DIAGRAM

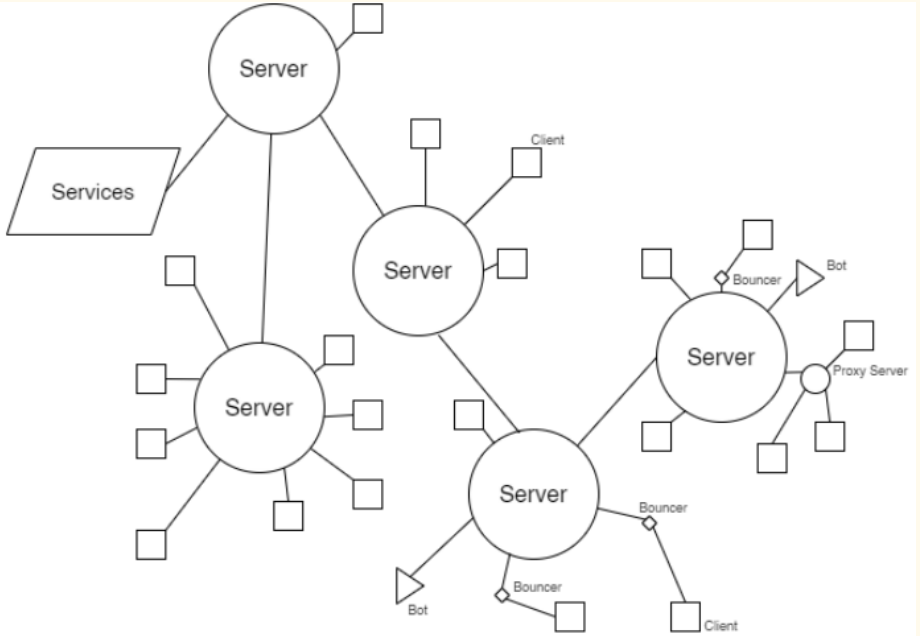




All the devices are connected to each other through routers that rely the two and fro communication.
Server is connected to clients, bouncers, bots, proxy servers, different types of services, etc.,.

Explanation of devices and flows

What we saw in the previous slides and to the right is a simple Chat Server Architecture diagram.
When an user sends a message, it is relayed to the server, which then holds the message and then sends it to any user who is connected to the server. Sometimes given the size of the userbase one server might not be able to accommodate all the users and so multiple servers might be needed for one room, in which case the messages will be sent to the other servers too, from where it will be shown to the users who are connected to that server.
Server is the place where the chat rooms are hosted and where the data is stored.



Scheme of an IRC network with servers, normal clients (square), bots (triangle), bouncers (rhombus), additional services and proxy servers.

The routing algorithm used in this case is Static Routing.
In static routing, static routing table is constructed to determine the path through which the packets are to be sent.
The Advantage of using static routing is that it consumes less bandwidth when compared to dynamic routing as no CPU cycles are-used in route calculation and communication.
Thus, it is a model that is both expandable and fault-resistant.

PERFORMANCE PARAMETERS

Parameter	Meaning w.r.t chat server	Formula
Bandwidth	Bandwidth for chat server is calculated as the max rate of transfer of information(messages,files,etc) from client to server.	Expressed as bits per second (bps), modern network links have greater capacity, which is typically measured in millions of bits per second (megabits per second, or Mbps) or billions of bits per second (gigabits per second, or Gbps).
Throughput	It is the rate of successful message communication over the IRC protocol.	Expressed as bits per second (bps), modern network links have greater capacity, which is typically measured in millions of bits per second (megabits per second, or Mbps) or billions of bits per second (gigabits per second, or Gbps).

PERFORMANCE PARAMETERS

Parameter	Meaning w.r.t chat server	Formula
Packet Loss	Packet loss occurs when one or more packets of data send by the client travelling across the network is not received by the server.	Efficiency = 100% * (transferred - retransmitted) / transferred Network Loss = 100 - Efficiency
Transmission time	The time required for transmission of a message depends on the size of the message and the bandwidth of the channel.	Transmission time = Message size / Bandwidth
Queuing Delay	Time spent by the data packet waiting in the queue before it is taken for execution is called queuing delay.	It depends on the size of the queue and number of packets arriving at a point of time into the queue so there is no formula to calculate
Jitter	Jitter is defined as the variation in time delay for the data packets sent over a network. This variable represents an identified disruption in the normal sequencing of data packets.	Latency=sum of all delays To measure Jitter, we take the difference between samples, then divide by the number of samples (minus 1).

ANALYTICAL REASONING

1. What are the likely problems users can face?
Ans. Users can face issues while the initial set up and in accessing the channels for the first time. They have to go through the guidelines that are set up for the channel operators to prevent themselves from being kicked out of the channel.

2. How does the application tackle the above-mentioned problems?
Ans. For the problem with network congestion, we can reduce it through network segmentation and content delivery network, we can have a upper limit over the number of people who can connect to the server at a time.

3. Is the application flexible according to the customer?
Ans. The application can be made flexible according to the user base the application is being developed for. We can have it as a terminal operated application, or a web application, or we can also implement it as a GUI based standalone application.

ANALYTICAL REASONING

4. What is the architecture used to implement IRC?
A. Servers are connected in spanning tree fashion architecture. In this, each server is connected to several others, which may be another server/set of servers or a number of clients. Server-Server connection is used for server-server communication (relying messages), to increase the capacity and length of the network (to have a greater ease and to adjust traffic flow properly) while Server-Client is used for communication purposes (the main objective for building the application). When you connect to server, first you have to choose specific channel to join and choose user name to identify yourself when you at chat. Your message is sent from client software on your PC to IRC server to which you are connected. Then message is sent from one server to other servers where all users on this channel are logged on.

5. Is there a limit on the number of channels a user can join/create?
A. There is no restriction on the number of channels a user can join or create, but then the server hosting the channels should have the required capacity to accommodate them i.e., the number of channels a user can create and host is limited by the hardware capacity of the server system hosting the channels.

ANALYTICAL REASONING

6. Is it costly to implement and run a chat server?
A. Implementing a chat server is relatively cheap and can be done by anyone with basic knowledge of networking concepts. Cost depends on the number of clients you wish to be able to accommodate in your server and the purpose of creating it. So, the cost can be anywhere between a few dollars to a few hundred dollars. The majority cost incurred is only during creation of the chat server, once establish the operational costs are very minimal.

7. Does it have privacy features?
A. The application is built for privacy. Basic privacy features are already there but many other privacy centric features can be added based on the user base the application is being developed for.

8. Can I communicate with another client one on one?
A. That is not the main feature of the application but then we can add such a functionality too. It is possible to implement such a functionality wherein the server does not display the message sent by the client on it's wall but instead relays it back to the client it was intended to reach.

9. Is the application secure?
A. Yes, the application is secure with basic security features in place. Incase more security features are required they can always be implemented based on the requirement.

ANALYTICAL REASONING

10. Is the application scalable?
A. Yes, the application can be scaled according to the future requirements. We can add extra hardware to the server to scale it or we can add another server to support the first one (can expand to any size this way) by adequately splitting the traffic between the servers and a few other methods are also possible to help us scale the application.

11. Can I send anything other than text like pictures, audio, video or other files?
A. IRC was built for text (chat) based communication and so has protocols only meant for that but it can always be expanded to accommodate files of various formats others than text. To accomplish that we would have to add in some more protocols and do some additions in the server code, possibly we would have to provide addition methods like compressing, resampling, etc., to prevent huge files sent by client from clogging the bandwidth and the memory of the server.

12. Why such a solution has been proposed to tackle the problems?
A. This solution has been proposed to facilitate easy communication between people with minimal training and infrastructure required from the user end. This solution also provides privacy to the user and promotes freedom of speech along with various other benefits. It's easy to implement, scale up and so can accommodate a large number of users thereby facilitating easy collaboration between people with common ideas from anywhere across the globe. Since the messages are stored in the server, a user can access the messages from anywhere irrespective of the location, and device being used even with a low internet bandwidth.

CONCLUSION

Thus a chat server has been built to provide a platform for remote users to communicate with each other in a secure manner. The multiple server architecture also enables multiple chat rooms.
IRC (Internet Relay Chat) Protocol has been used is a protocol for facilitating text communication and this application can be extended to provide video and image transfers as well.

BIBLIOGRAPHY

REFERENCE LINKS:-

https://www.hjp.at/doc/rfc/rfc2810.html
https://dl.acm.org/doi/pdf/10.17487/RFC1459 (RFC1459)

RESEARCH RELATED TO CASE STUDY:-

Hossein Rouhani Zeidanloo, “New Approach for Detection of IRC and P2P Botnets”,International Journal of Computer and Electrical Engineering, Vol.2, No.6, December, 2010

Sicong Shao; Cihan Tunc; Pratik Satam; Salim Hariri , et.al “Real-Time IRC Threat Detection Framework” , 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)

CONTRIBUTIONS

A. SAI THARUN

CB.EN.U4CSE18250

DOCUMENTATION, CODE

SUDARSHAN M.S

CB.EN.U4CSE18258

DIAGRAM, CODE

TANYA K

CB.EN.U4CSE18259

DOCUMENTATION, DIAGRAM

CODE SNIPPETS - SERVER SIDE

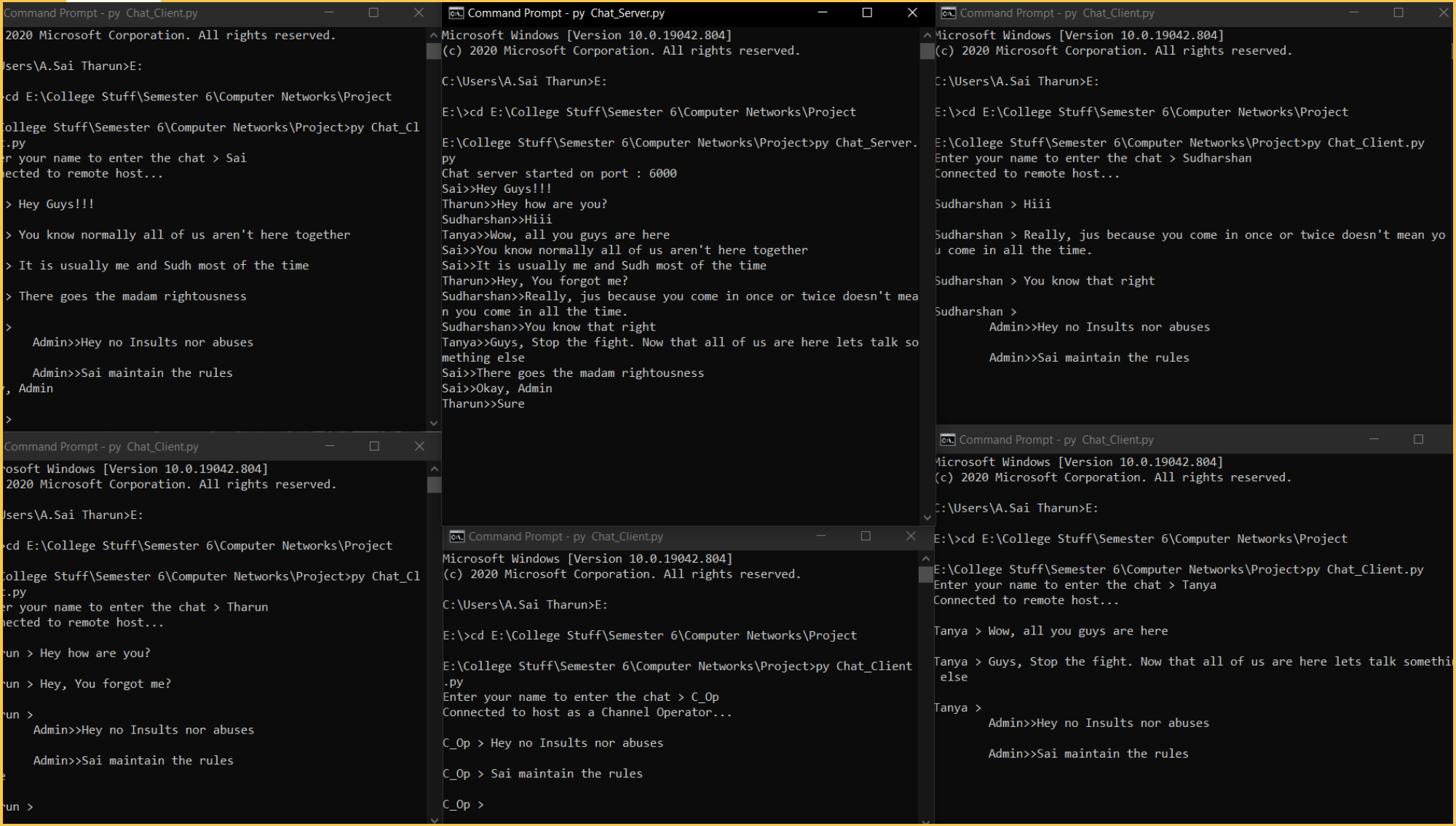
```python
import socket, threading

def accept_client():
    while True:
        cli_sock, cli_add = ser_sock.accept()
        CONNECTION_LIST.append(cli_sock)
        thread_client = threading.Thread(target = msgrecieve_usr, args=[cli_sock])
        thread_client.start()

def msgrecieve_usr(cli_sock):                                   # Recieving the msg
    while True:
        try:
            data = cli_sock.recv(1024)
            if data:
                name = data.decode()
                uname = name.split('>>')[0]
                if uname == 'C_Op':
                    msg = 'Admin'+'>>'+name.split('>>')[1]
                    b_usrs(cli_sock, msg)
                else:
                    m_usr(cli_sock, data)
        except Exception as x:
            print(x)
            break

def m_usr(cs_sock, msg):                                        # Printing the msg sent by the user
    for client in CONNECTION_LIST:
        if client == cs_sock:
            print(msg.decode())
```

```python
def b_usrs(cs_sock, msg):                                       # Broadcasting the msg sent by the admin
    for client in CONNECTION_LIST:
        if client != cs_sock:
            client.send(msg.encode())


if __name__ == "__main__":
    CONNECTION_LIST = []
    ser_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)   # Creating a socket

    HOST = 'localhost'                                          # Binding a socket
    PORT = 6000
    ser_sock.bind((HOST, PORT))

    ser_sock.listen(1)
    print('Chat server started on port : ' + str(PORT))

    thread_ac = threading.Thread(target = accept_client)
    thread_ac.start()
```

CODE SNIPPETS - CLIENT SIDE

```python
import socket, threading

def send(uname):                                                    # Sending the msg to the server
    while True:
        msg = input(f'\n{uname} > ')
        data = uname + '>>' + msg
        cli_sock.send(data.encode())

def receive():                                                      # Recieving the broadcast msg from the server
    while True:
        data = cli_sock.recv(1024)
        print('\n\t'+ str(data.decode()))

if __name__ == "__main__":
    cli_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)    # Creating a socket

    HOST = 'localhost'                                              # Connecting to the socket
    PORT = 6000
    cli_sock.connect((HOST, PORT))

    uname = input('Enter your name to enter the chat > ')
    if uname == 'C_Op':
        print('Connected to host as a Channel Operator...')
    else:
        print('Connected to remote host...')

    thread_send = threading.Thread(target = send, args=[uname])
    thread_send.start()

    thread_receive = threading.Thread(target = receive)
    thread_receive.start()
```

OUTPUTS


Command Prompt - py Chat_Client.py
2020 Microsoft Corporation. All rights reserved.
Users\A.Sai Tharun>E:
cd E:\College Stuff\Semester 6\Computer Networks\Project
ollege Stuff\Semester 6\Computer Networks\Project>py Chat_Cl
.py
r your name to enter the chat > Sai
ected to remote host...
> Hey Guys!!!
> You know normally all of us aren't here together
> It is usually me and Sudh most of the time
> There goes the madam rightousness
>
Admin>>Hey no Insults nor abuses
Admin>>Sai maintain the rules
, Admin
>
Command Prompt - py Chat_Server.py
Microsoft Windows [Version 10.0.19042.804]
(c) 2020 Microsoft Corporation. All rights reserved.
C:\Users\A.Sai Tharun>E:
E:\>cd E:\College Stuff\Semester 6\Computer Networks\Project
E:\College Stuff\Semester 6\Computer Networks\Project>py Chat_Server.
py
Chat server started on port : 6000
Sai>>Hey Guys!!!
Tharun>>Hey how are you?
Sudharshan>>Hiii
Tanya>>Wow, all you guys are here
Sai>>You know normally all of us aren't here together
Sai>>It is usually me and Sudh most of the time
Tharun>>Hey, You forgot me?
Sudharshan>>Really, jus because you come in once or twice doesn't mea
n you come in all the time.
Sudharshan>>You know that right
Tanya>>Guys, Stop the fight. Now that all of us are here lets talk so
mething else
Sai>>There goes the madam rightousness
Sai>>Okay, Admin
Tharun>>Sure
Command Prompt - py Chat_Client.py
Microsoft Windows [Version 10.0.19042.804]
(c) 2020 Microsoft Corporation. All rights reserved.
C:\Users\A.Sai Tharun>E:
E:\>cd E:\College Stuff\Semester 6\Computer Networks\Project
E:\College Stuff\Semester 6\Computer Networks\Project>py Chat_Client
.py
Enter your name to enter the chat > C_Op
Connected to host as a Channel Operator...
C_Op > Hey no Insults nor abuses
C_Op > Sai maintain the rules
C_Op >
Command Prompt - py Chat_Client.py
Microsoft Windows [Version 10.0.19042.804]
(c) 2020 Microsoft Corporation. All rights reserved.
C:\Users\A.Sai Tharun>E:
E:\>cd E:\College Stuff\Semester 6\Computer Networks\Project
E:\College Stuff\Semester 6\Computer Networks\Project>py Chat_Client.py
Enter your name to enter the chat > Sudharshan
Connected to remote host...
Sudharshan > Hiii
Sudharshan > Really, jus because you come in once or twice doesn't mean yo
u come in all the time.
Sudharshan > You know that right
Sudharshan >
Admin>>Hey no Insults nor abuses
Admin>>Sai maintain the rules
Command Prompt - py Chat_Client.py
Microsoft Windows [Version 10.0.19042.804]
(c) 2020 Microsoft Corporation. All rights reserved.
C:\Users\A.Sai Tharun>E:
E:\>cd E:\College Stuff\Semester 6\Computer Networks\Project
E:\College Stuff\Semester 6\Computer Networks\Project>py Chat_Client.py
Enter your name to enter the chat > Tanya
Connected to remote host...
Tanya > Wow, all you guys are here
Tanya > Guys, Stop the fight. Now that all of us are here lets talk somethi
else
Tanya >
Admin>>Hey no Insults nor abuses
Admin>>Sai maintain the rules
Command Prompt - py Chat_Client.py
osoft Windows [Version 10.0.19042.804]
2020 Microsoft Corporation. All rights reserved.
Users\A.Sai Tharun>E:
cd E:\College Stuff\Semester 6\Computer Networks\Project
ollege Stuff\Semester 6\Computer Networks\Project>py Chat_Cl
.py
r your name to enter the chat > Tharun
ected to remote host...
un > Hey how are you?
un > Hey, You forgot me?
un >
Admin>>Hey no Insults nor abuses
Admin>>Sai maintain the rules
un >

GO BACK N CODE SNIPPETS – SERVER SIDE

```
# Receiver.py
import time, socket, sys
import random

print("\nWelcome to Chat Room\n")
print("Initialising....\n")
time.sleep(1)

s = socket.socket()
shost = socket.gethostname()
ip = socket.gethostbyname(shost)
print(shost, "(", ip, ")\n")
host = input(str("Enter server address: "))
name = input(str("\nEnter your name: "))
port = 1234
print("\nTrying to connect to ", host, "(", port, ")\n")
time.sleep(1)
s.connect((host, port))
print("Connected...\n")

s.send(name.encode())
s_name = s.recv(1024)
s_name = s_name.decode()
print(s_name, "has joined the chat room\nEnter [e] to exit chat room\n")

while True:
    m=s.recv(1024)
    m=m.decode()
    k=s.recv(1024)
    if(k=='Left Chat Room!'):
        print("Exiting connection")
        break
```

```
else:
    k=k.decode()
    k=int(k)
    i=0
    a=""
    b=""
    f=random.randint(0,1)
    message=""
    while i!=k:
        f=random.randint(0,1)
        if(f==0):
            b="ACK Lost"
            message = s.recv(1024)
            message = message.decode()
            s.send(b.encode())

        elif(f==1):
            b="ACK "+str(i)
            message = s.recv(1024)
            message = message.decode()
            s.send(b.encode())
            a=a+message
            i=i+1



print("The message received is :", m)
print("Waiting for next message..... ")
```

GO BACK N CODE SNIPPETS – CLIENT SIDE

```python
# Sender.py
import time, socket, sys

def decimalToBinary(n):
    return n.replace("0b", "")

def binarycode(s):
    a_byte_array = bytearray(s, "utf8")

    byte_list = []

    for byte in a_byte_array:
        binary_representation = bin(byte)
        byte_list.append(decimalToBinary(binary_representation))

    #print(byte_list)
    a=""
    for i in byte_list:
        a=a+i
    return a

print("\nWelcome to Chat Room\n")
print("Initialising....\n")
time.sleep(0.5)

s = socket.socket()
host = socket.gethostname()
ip = socket.gethostbyname(host)
port = 1234
s.bind((host, port))
print(host, "(", ip, ")\n")
name = input(str("Enter your name: "))
```

GO BACK N CODE SNIPPETS – CLIENT SIDE

```
s.listen(1)
print("\nWaiting for incoming connections...\n")
conn, addr = s.accept()
print("Received connection from ", addr[0], "(", addr[1], ")\n")

s_name = conn.recv(1024)
s_name = s_name.decode()
print(s_name, "has connected to the chat room\nEnter [e] to exit chat room\n")
conn.send(name.encode())

while True:
    message = input(str("Me : "))
    conn.send(message.encode())
    if message == "[e]":
        message = "Left chat room!"
        print("You left the chat room")
        conn.send(message.encode())
        print("\n")
        break
    message=binarycode(message)
    f=str(len(message))
    conn.send(f.encode())
    i=0
    j=0
    j=int(input("Enter the window size -> "))
    b=""
    j=j-1
    f=int(f)
    k=j
    while i!=f:
        if(i==k):
            break
```

GO BACK N CODE SNIPPETS – CLIENT SIDE

```
ile(i!=(f-j)):
  conn.send(message[i].encode())
  b=conn.recv(1024)
  b=b.decode()
  print(b)
  if(b!="ACK Lost"):
      time.sleep(1)
      print("Acknowledgement Received!\nThe sliding window is in the range "+(str(i+1))+" to "+str(k+1)+"\n Now sending the next packet")
      i=i+1
      k=k+1
      time.sleep(0.5)
  else:
      time.sleep(0.5)
      print("Acknowledgement of the data bit is LOST!\n The sliding window remains in the range "+(str(i+1))+" to "+str(k+1)+"\n Duplicate detected! Removing duplicates
      time.sleep(0.5)
ile(i!=f):
  if(i==k):
      break
  conn.send(message[i].encode())
  b=conn.recv(1024)
  b=b.decode()
  print(b)
  if(b!="ACK Lost"):
      time.sleep(0.5)
      print("Acknowledgement Received!\n The sliding window is in the range "+(str(i+1))+" to "+str(k)+"\n Now sending the next packet")
      i=i+1
      time.sleep(0.5)
  else:
      time.sleep(0.5)
      print("Acknowledgement of the data bit is LOST!\n The sliding window remains in the range "+(str(i+1))+" to "+str(k)+"\n Duplicate detected! Removing duplicates")
      time.sleep(0.5)
```

OUTPUTS

```
PS F:\ChatServer-ARQ> py receiver.py

Welcome to Chat Room

Initialising....

MSSudarshanPC ( 192.168.1.7 )

Enter server address: 192.168.1.7

Enter your name: sai tharun

Trying to connect to  192.168.1.7 ( 1234 )

Connected...

sudarshan has joined the chat room
Enter [e] to exit chat room

The message received is : a
Waiting for next message.....
```

OUTPUTS

```
PS F:\ChatServer-ARQ> py sender.py

Welcome to Chat Room

Initialising....

MSSudarshanPC ( 192.168.1.7 )

Enter your name: sudarshan

Waiting for incoming connections...

Received connection from  192.168.1.7 ( 59039 )

sai tharun has connected to the chat room
Enter [e] to exit chat room

Me : a
Enter the window size -> 2
ACK 0
Acknowledgement Received!
The sliding window is in the range 1 to 2
 Now sending the next packet
ACK Lost
Acknowledgement of the data bit is LOST!
 The sliding window remains in the range 2 to 3
 Duplicate detected! Removing duplicates
ACK 1
Acknowledgement Received!
The sliding window is in the range 2 to 3
 Now sending the next packet
ACK Lost
Acknowledgement of the data bit is LOST!
 The sliding window remains in the range 3 to 4
 Duplicate detected! Removing duplicates
ACK 2
Acknowledgement Received!
The sliding window is in the range 3 to 4
 Now sending the next packet
ACK Lost
Acknowledgement of the data bit is LOST!
 The sliding window remains in the range 4 to 5
 Duplicate detected! Removing duplicates
ACK 3
Acknowledgement Received!
The sliding window is in the range 4 to 5
 Now sending the next packet
ACK Lost
Acknowledgement of the data bit is LOST!
 The sliding window remains in the range 5 to 6
 Duplicate detected! Removing duplicates
ACK Lost
Acknowledgement of the data bit is LOST!
 The sliding window remains in the range 5 to 6
 Duplicate detected! Removing duplicates
ACK 4
Acknowledgement Received!
The sliding window is in the range 5 to 6
 Now sending the next packet
ACK Lost
Acknowledgement of the data bit is LOST!
 The sliding window remains in the range 6 to 7
 Duplicate detected! Removing duplicates
ACK 5
Acknowledgement Received!
The sliding window is in the range 6 to 7
 Now sending the next packet
ACK Lost
Acknowledgement of the data bit is LOST!
 The sliding window remains in the range 7 to 7
 Duplicate detected! Removing duplicates
ACK 6
Acknowledgement Received!
 The sliding window is in the range 7 to 7
 Now sending the next packet
Packets sent successfully
Me :
```

SELECTIVE REPEAT ARQ CODE SNIPPETS – CLIENT SIDE LOGIC

```python
class Client(Node):
    def __init__(self, mode, port, timeout, windowSize, name):
        super(Client, self).__init__(mode, port, name)
        self.packetList = []
        self.lower = 0  # lower and upper to handle sliding window
        self.upper = windowSize - 1
        self.windowSize = windowSize
        self.timeout = timeout
        print('Client node started')

    def readFromFile(self, filename):
        self.debug('Reading and parsing from file into list')
        with open(filename) as f:
            seqNumber = 0
            while True:
                readChar = f.read(1)
                if not readChar:
                    break
                # normalize all possible numbers
                sequenceNumber = str(seqNumber)
                if(len(str(seqNumber)) == 1):
                    sequenceNumber = "0000"+str(seqNumber)
                elif(len(str(seqNumber)) == 2):
                    sequenceNumber = "000" + str(seqNumber)
                elif(len(str(seqNumber)) == 3):
                    sequenceNumber = "00" + str(seqNumber)
                elif(len(str(seqNumber)) == 4):
                    sequenceNumber = "0" + str(seqNumber)
                serialized = sequenceNumber+":"+str(readChar)
                seqNumber += 1

                self.packetList.append(Packet(serialized))
            if len(self.packetList) < 10001:
                self.packetList.append(Packet('10001:z'))
        self.debug('Reading done. Total ' +
                   str(len(self.packetList))+' nodes to send.')

    def sendPacket(self, index):
        packet = self.packetList[index].getSerialized()
        self.debug('Sending packet: ' + packet + ' to server')
        self.packetList[index].send()
        self.sock.send(packet.encode('UTF-8'))
```

SELECTIVE REPEAT ARQ CODE SNIPPETS – CLIENT SIDE LOGIC

```
def checkTime(self):
    i = self.lower
    self.debug('Checking for timeout...')
    while(i <= self.upper):
        if self.packetList[i].getAckRcvd() is False and self.timeout < time.time() % 60-self.packetList[i].getSent():
            self.sendPacket(i)
        i += 1

def sendWindow(self):
    i = self.lower
    while i <= self.upper:
        self.sendPacket(i)
        i += 1

def slideWindowBy(self, number):
    if(self.upper+1 < len(self.packetList)):
        self.debug('Sliding client\'s window ' +
                   str(number) + ' position(s)')
        self.lower += number
        self.upper += number
        self.sendPacket(self.upper)

def slideWindow(self):
    i = self.lower
    while i <= self.upper:
        if(self.packetList[i].getAckRcvd() == True):
            self.slideWindowBy(1)
        else:
            break
        i += 1

def recieveAck(self):
    while True:
        self.debug('Recieving ACK...')
        ack = None
        self.sock.settimeout(10)
        ack = self.sock.recv(5)
        if ack is not None:
            if int(ack) == 10001:
                self.debug(
                    'No more ACKs to recieve. Shutting down....')
                break
            self.packetList[int(ack)].setAckRcvd(True)
        self.slideWindow()
        self.debug('Recieved ACK message for packet '+ack.decode('UTF-8'))
        self.checkTime()
        self.sock.settimeout(None)

def runClientNode(self, filename):
    start = time.time() % 60
    self.connect()
    self.readFromFile(filename)
    self.sendWindow()
    self.recieveAck()
    self.closeSocket()
    self.debug('Finished.')
    print(time.time() % 60 - start)
```

SELECTIVE REPEAT ARQ CODE SNIPPETS – CLIENT SIDE LOGIC

```
def checkTime(self):
    i = self.lower
    self.debug('Checking for timeout...')
    while(i <= self.upper):
        if self.packetList[i].getAckRcvd() is False and self.timeout < time.time() % 60-self.packetList[i].getSent():
            self.sendPacket(i)
        i += 1

def sendWindow(self):
    i = self.lower
    while i <= self.upper:
        self.sendPacket(i)
        i += 1

def slideWindowBy(self, number):
    if(self.upper+1 < len(self.packetList)):
        self.debug('Sliding client\'s window ' +
                   str(number) + ' position(s)')
        self.lower += number
        self.upper += number
        self.sendPacket(self.upper)

def slideWindow(self):
    i = self.lower
    while i <= self.upper:
        if(self.packetList[i].getAckRcvd() == True):
            self.slideWindowBy(1)
        else:
            break
        i += 1

def recieveAck(self):
    while True:
        self.debug('Recieving ACK...')
        ack = None
        self.sock.settimeout(10)
        ack = self.sock.recv(5)
        if ack is not None:
            if int(ack) == 10001:
                self.debug(
                    'No more ACKs to recieve. Shutting down....')
                break
            self.packetList[int(ack)].setAckRcvd(True)
        self.slideWindow()
        self.debug('Recieved ACK message for packet '+ack.decode('UTF-8'))
        self.checkTime()
        self.sock.settimeout(None)

def runClientNode(self, filename):
    start = time.time() % 60
    self.connect()
    self.readFromFile(filename)
    self.sendWindow()
    self.recieveAck()
    self.closeSocket()
    self.debug('Finished.')
    print(time.time() % 60 - start)
```

SELECTIVE REPEAT ARQ CODE SNIPPETS – CLIENT SIDE LOGIC

```
def decode(self, generator):
    temp = ""
    print("\nManchester Encoding                : \n" + str(self.bits))
    yVal = [int(x) for x in list(self.bits)]
    temp = []
    for val in yVal:
        if val == 0:
            temp.append(-1)
        else:
            temp.append(1)
    yVal = temp
    self.manchesterYVal = yVal
    temp = ""
    for i in range(0, len(self.bits), 2):
        s = self.bits[i: i + 2]
        if s == "01":
            temp += "1"
        elif s == "10":
            temp += "0"
    self.bits = temp
    tempOriginalYVal = [int(x) for x in self.bits]
    for y in tempOriginalYVal:
        self.originalYVal.append(y)
        self.originalYVal.append(y)
    print("\nOriginal Encoding With CRC Remainder     : \n" + str(self.bits))
    choice = input(
        "\nPress 1 for delibertely inserting error bits yes else give any other input : ")
    if choice == "1":
        temp = list(self.bits)

        for i in range(5):
            index = random.randint(0, len(self.bits) - 1)
            if temp[index] == "0":
                temp[index] = "1"
            elif temp[index] == "1":
                temp[index] = "0"
        self.bits = "".join(temp)
```

SELECTIVE REPEAT ARQ CODE SNIPPETS – CLIENT SIDE CALLING FUNCTION

```python
from object1 import Client


def runClient():
    print("Welcome to Client...")
    winsize = int(input("Enter a window size: "))
    client = Client(1, 10000, 10, winsize, '[CLIENT]:')
    client.runClientNode('hello.txt')


if __name__ == '__main__':
    runClient()
```

SELECTIVE REPEAT ARQ CODE SNIPPETS – SERVER SIDE LOGIC

```
class Server(Node):
    def __init__(self, mode, port, timeout, windowSize, name):
        super(Server, self).__init__(mode, port, name)
        self.packetList = [None]*10000  # por arreglar
        self.upper = windowSize - 1
        self.lower = 0
        self.windowSize = windowSize
        self.timeout = timeout
        print('Server node started')

    def writeresult(self):
        with open("hello.txt", 'r') as f1, open("result.txt", 'w+') as f2:
            f2.write(f1.read())
        with open("unsentmsg.txt", 'w+') as f:
            f.write(str(unsentlist))

    def exportResults(self):
        result = ""
        for x in self.packetList:
            if x is None:
                break
            result += x.getCharacter()
        with File('sentmsg.txt', 'w') as f:
            f.write(str(result))
```

SELECTIVE REPEAT ARQ CODE SNIPPETS – SERVER SIDE LOGIC

```
def decode(self, generator):
    temp = ""
    print("\nManchester Encoding                : \n" + str(self.bits))
    yVal = [int(x) for x in list(self.bits)]
    temp = []
    for val in yVal:
        if val == 0:
            temp.append(-1)
        else:
            temp.append(1)
    yVal = temp
    self.manchesterYVal = yVal
    temp = ""
    for i in range(0, len(self.bits), 2):
        s = self.bits[i: i + 2]
        if s == "01":
            temp += "1"
        elif s == "10":
            temp += "0"
    self.bits = temp
    tempOriginalYVal = [int(x) for x in self.bits]
    for y in tempOriginalYVal:
        self.originalYVal.append(y)
        self.originalYVal.append(y)
    print("\nOriginal Encoding With CRC Remainder     : \n" + str(self.bits))
    choice = input(
        "\nPress 1 for delibertely inserting error bits yes else give any other input : ")
    if choice == "1":
        temp = list(self.bits)

        for i in range(5):
            index = random.randint(0, len(self.bits) - 1)
            if temp[index] == "0":
                temp[index] = "1"
            elif temp[index] == "1":
                temp[index] = "0"
        self.bits = "".join(temp)
```

SELECTIVE REPEAT ARQ CODE SNIPPETS – SERVER SIDE LOGIC

```
def recieve(self):
    cont = True
    while cont:
        self.debug('Waiting connection from middle node...')
        connection, address = self.sock.accept()
        try:
            while True:
                buf = connection.recv(7)
                if buf:
                    buf = buf.decode('UTF-8')
                    print(buf)
                    seqNumber, character = buf.split(':', 1)
                    if seqNumber == '' or int(seqNumber) == 10000:
                        cont = False
                        self.debug('Done. Closing socket.')
                        break
                    print('Received: ' + buf)
                    if self.packetList[int(seqNumber)] is None:
                        self.packetList[int(seqNumber)] = Packet(buf)
                    self.debug('Sending ['+seqNumber+'] ACK...')
                    connection.send(seqNumber.encode())
                else:
                    break
        finally:
            connection.close()
            self.exportResults()

def runServerNode(self):
    self.writeresult()
    self.bind()
    self.listen()
    self.recieve()
```

SELECTIVE REPEAT ARQ CODE SNIPPETS – SERVER SIDE CALLING FUNCTION

```python
from object1 import Server


def runServer():
    print("Welcome to Server")
    winsize = int(input("Enter a window size: "))
    server = Server(1, 10001, 1000, winsize, '[SERVER]:')
    server.runServerNode()


if __name__ == '__main__':
    runServer()
```

SELECTIVE REPEAT ARQ CODE SNIPPETS – INTERMEDIATE SIDE CALLING FUNCTION

```python
from object1 import Middle
import threading
import queue


def run1(debug, por, q1, q2, prob, kill):
    # From client to middle
    m = Middle(debug, por, q1, q2, prob, '[CLIENT-MIDDLE]:', kill)
    m.runClientMiddle()


def run2(debug, por, q1, q2, prob, kill):
    # From middle to server
    m = Middle(debug, por, q1, q2, prob, '[MIDDLE-SERVER]:', kill)
    m.runMiddleServer()


if __name__ == '__main__':
    debug = input("Set debug state [1=debug mode/0=standard mode]: ")
    kill = input(
        "Do you want to kill packets?(only in debug mode)[1=yes/0=no]: ")
    portClient = input(
        "Set the port for the client to connect(must be the same port the client uses): ")
    portServer = input(
        "Set the port binded with the server (must be the same port the server uses): ")
    prob = input("Set packet loss probability[0.0 - 1.0]: ")
    winSize = input("Set window size (same as client and server window):")
    winSize = int(winSize)
    debug = int(debug)
    portServer = int(portServer)
    portClient = int(portClient)
    prob = float(prob)
    queue1 = queue.Queue(winSize)
    queue2 = queue.Queue(winSize)

    t1 = threading.Thread(target=run1, args=(
        debug, portClient, queue1, queue2, prob, kill))
    t2 = threading.Thread(target=run2, args=(
        debug, portServer, queue2, queue1, prob, kill))

    t1.start()
    t2.start()
```

SELECTIVE REPEAT ARQ OUTPUTS

```
PS E:\6th_sem\Networks\Project\SRARQ\SRARQ> py receiver.py
Welcome to Server
Enter a window size: 5
Server node started
('127.0.0.1', 10001)
[SERVER]: Server binded to ('127.0.0.1', 10001)
[SERVER]:Waiting connection from middle node...
00000:t
Received: 00000:t
[SERVER]:Sending [00000] ACK...
00001:h
Received: 00001:h
[SERVER]:Sending [00001] ACK...
00002:i
Received: 00002:i
[SERVER]:Sending [00002] ACK...
00003:s
Received: 00003:s
[SERVER]:Sending [00003] ACK...
00004:
Received: 00004:
[SERVER]:Sending [00004] ACK...
00005:i
Received: 00005:i
[SERVER]:Sending [00005] ACK...
00006:s
Received: 00006:s
[SERVER]:Sending [00006] ACK...
00007:
Received: 00007:
[SERVER]:Sending [00007] ACK...
00008:a
Received: 00008:a
[SERVER]:Sending [00008] ACK...
00009:
Received: 00009:
[SERVER]:Sending [00009] ACK...
00010:t
Received: 00010:t
[SERVER]:Sending [00010] ACK...
00011:e
Received: 00011:e
[SERVER]:Sending [00011] ACK...
00012:s
Received: 00012:s
[SERVER]:Sending [00012] ACK...
00013:t
Received: 00013:t
[SERVER]:Sending [00013] ACK...
00014:
```

```
Received: 00014:
[SERVER]:Sending [00014] ACK...
00015:f
Received: 00015:f
[SERVER]:Sending [00015] ACK...
00016:i
Received: 00016:i
[SERVER]:Sending [00016] ACK...
00017:l
Received: 00017:l
[SERVER]:Sending [00017] ACK...
00018:e
Received: 00018:e
[SERVER]:Sending [00018] ACK...
10000:z
[SERVER]:Done. Closing socket.
```

SELECTIVE REPEAT ARQ OUTPUTS

```
PS E:\6th_sem\Networks\Project\SRARQ\SRARQ> py intermediate.py
Set debug state [1=debug mode/0=standard mode]: 0
Do you want to kill packets?(only in debug mode)[1=yes/0=no]: 0
Set the port for the client to connect(must be the same port the client
 uses): 10000
Set the port binded with the server (must be the same port the server u
ses): 10001
Set packet loss probability[0.0 - 1.0]: 0.0
Set window size (same as client and server window):5
('127.0.0.1', 10000)
[MIDDLE-SERVER]: Client connected to ('127.0.0.1', 10001)
[CLIENT-MIDDLE]: Server binded to ('127.0.0.1', 10000)
[CLIENT-MIDDLE]:Recieved: 00000:t
[CLIENT-MIDDLE]:No ACK to forward right now.
[CLIENT-MIDDLE]:Recieved: 00001:h
[CLIENT-MIDDLE]:No ACK to forward right now.
[CLIENT-MIDDLE]:Recieved: 00002:i
[CLIENT-MIDDLE]:Retrieved. Forwarding ACK:[00000] to client
[CLIENT-MIDDLE]:Recieved: 00003:s
[CLIENT-MIDDLE]:Retrieved. Forwarding ACK:[00001] to client
[CLIENT-MIDDLE]:Recieved: 00004:
[CLIENT-MIDDLE]:Retrieved. Forwarding ACK:[00002] to client
[CLIENT-MIDDLE]:Recieved: 00005:i
[CLIENT-MIDDLE]:Retrieved. Forwarding ACK:[00003] to client
[CLIENT-MIDDLE]:Recieved: 00006:s
[CLIENT-MIDDLE]:Retrieved. Forwarding ACK:[00004] to client
[CLIENT-MIDDLE]:Recieved: 00007:
[CLIENT-MIDDLE]:Retrieved. Forwarding ACK:[00005] to client
[CLIENT-MIDDLE]:Recieved: 00008:a
[CLIENT-MIDDLE]:Retrieved. Forwarding ACK:[00006] to client
[CLIENT-MIDDLE]:Recieved: 00009:
[CLIENT-MIDDLE]:Retrieved. Forwarding ACK:[00007] to client
[CLIENT-MIDDLE]:Recieved: 00010:t
[CLIENT-MIDDLE]:Retrieved. Forwarding ACK:[00008] to client
[CLIENT-MIDDLE]:Recieved: 00011:e
[CLIENT-MIDDLE]:Retrieved. Forwarding ACK:[00009] to client
[CLIENT-MIDDLE]:Recieved: 00012:s
[CLIENT-MIDDLE]:No ACK to forward right now.
[CLIENT-MIDDLE]:Recieved: 00013:t
[CLIENT-MIDDLE]:Retrieved. Forwarding ACK:[00010] to client
[CLIENT-MIDDLE]:Recieved: 00014:
[CLIENT-MIDDLE]:Retrieved. Forwarding ACK:[00011] to client
[CLIENT-MIDDLE]:Recieved: 00015:f
[CLIENT-MIDDLE]:Retrieved. Forwarding ACK:[00012] to client
[CLIENT-MIDDLE]:Recieved: 00016:i
[CLIENT-MIDDLE]:Retrieved. Forwarding ACK:[00013] to client
[CLIENT-MIDDLE]:Recieved: 00017:l
[CLIENT-MIDDLE]:Retrieved. Forwarding ACK:[00014] to client
[CLIENT-MIDDLE]:Recieved: 00018:e
[CLIENT-MIDDLE]:Retrieved. Forwarding ACK:[00015] to client
[CLIENT-MIDDLE]:Recieved: 10000:z
[CLIENT-MIDDLE]:Retrieved. Forwarding ACK:[00016] to client
Exception in thread Thread-1:
Traceback (most recent call last):
```

SELECTIVE REPEAT ARQ OUTPUTS

```
PS E:\6th_sem\Networks\Project\SRARQ\SRARQ> py sender.py
Welcome to Client...
Enter a window size: 5
Client node started
[CLIENT]: Client connected to ('127.0.0.1', 10000)
[CLIENT]:Reading and parsing from file into list
[CLIENT]:Reading done. Total 20 nodes to send.
[CLIENT]:Sending packet: 00000:t to server
[CLIENT]:Sending packet: 00001:h to server
[CLIENT]:Sending packet: 00002:i to server
[CLIENT]:Sending packet: 00003:s to server
[CLIENT]:Sending packet: 00004:  to server
[CLIENT]:Recieving ACK...
b'00000'
<class 'str'>
In else stmt
b'00000'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00005:i to server
[CLIENT]:Recieved ACK message for packet 00000
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00001'
<class 'str'>
In else stmt
b'00001'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00006:s to server
[CLIENT]:Recieved ACK message for packet 00001
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00002'
<class 'str'>
In else stmt
b'00002'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00007:  to server
[CLIENT]:Recieved ACK message for packet 00002
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00003'
<class 'str'>
In else stmt
b'00003'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00008:a to server
[CLIENT]:Recieved ACK message for packet 00003
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00004'
<class 'str'>
In else stmt
b'00004'
[CLIENT]:Sliding client's window 1 position(s)
```

```
b'00005'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00010:t to server
[CLIENT]:Recieved ACK message for packet 00005
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00006'
<class 'str'>
In else stmt
b'00006'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00011:e to server
[CLIENT]:Recieved ACK message for packet 00006
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00007'
<class 'str'>
In else stmt
b'00007'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00012:s to server
[CLIENT]:Recieved ACK message for packet 00007
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00008'
<class 'str'>
In else stmt
b'00008'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00013:t to server
[CLIENT]:Recieved ACK message for packet 00008
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00009'
<class 'str'>
In else stmt
b'00009'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00014:  to server
[CLIENT]:Recieved ACK message for packet 00009
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00010'
<class 'str'>
In else stmt
b'00010'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00015:f to server
[CLIENT]:Recieved ACK message for packet 00010
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00011'
<class 'str'>
```

SELECTIVE REPEAT ARQ OUTPUTS

```
b'00005'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00010:t to server
[CLIENT]:Recieved ACK message for packet 00005
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00006'
<class 'str'>
In else stmt
b'00006'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00011:e to server
[CLIENT]:Recieved ACK message for packet 00006
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00007'
<class 'str'>
In else stmt
b'00007'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00012:s to server
[CLIENT]:Recieved ACK message for packet 00007
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00008'
<class 'str'>
In else stmt
b'00008'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00013:t to server
[CLIENT]:Recieved ACK message for packet 00008
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00009'
<class 'str'>
In else stmt
b'00009'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00014:  to server
[CLIENT]:Recieved ACK message for packet 00009
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00010'
<class 'str'>
In else stmt
b'00010'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00015:f to server
[CLIENT]:Recieved ACK message for packet 00010
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00011'
<class 'str'>
```

```
b'00012'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00017:l to server
[CLIENT]:Recieved ACK message for packet 00012
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00013'
<class 'str'>
In else stmt
b'00013'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 00018:e to server
[CLIENT]:Recieved ACK message for packet 00013
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00014'
<class 'str'>
In else stmt
b'00014'
[CLIENT]:Sliding client's window 1 position(s)
[CLIENT]:Sending packet: 10000:z to server
[CLIENT]:Recieved ACK message for packet 00014
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00015'
<class 'str'>
In else stmt
b'00015'
[CLIENT]:Recieved ACK message for packet 00015
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
b'00016'
<class 'str'>
In else stmt
b'00016'
[CLIENT]:Recieved ACK message for packet 00016
[CLIENT]:Checking for timeout...
[CLIENT]:Recieving ACK...
Traceback (most recent call last):
  File "E:\6th_sem\Networks\Project\SRARQ\SRARQ\object1.py", line 149,
in recieveAck
    ack = self.sock.recv(5)
socket.timeout: timed out
```

SELECTIVE REPEAT ARQ CODE SNIPPETS – INTERMEDIATE SIDE CALLING FUNCTION

```
def decode(self, generator):
    temp = ""
    print("\nManchester Encoding                : \n" + str(self.bits))
    yVal = [int(x) for x in list(self.bits)]
    temp = []
    for val in yVal:
        if val == 0:
            temp.append(-1)
        else:
            temp.append(1)
    yVal = temp
    self.manchesterYVal = yVal
    temp = ""
    for i in range(0, len(self.bits), 2):
        s = self.bits[i: i + 2]
        if s == "01":
            temp += "1"
        elif s == "10":
            temp += "0"
    self.bits = temp
    tempOriginalYVal = [int(x) for x in self.bits]
    for y in tempOriginalYVal:
        self.originalYVal.append(y)
        self.originalYVal.append(y)
    print("\nOriginal Encoding With CRC Remainder     : \n" + str(self.bits))
    choice = input(
        "\nPress 1 for delibertely inserting error bits yes else give any other input : ")
    if choice == "1":
        temp = list(self.bits)

        for i in range(5):
            index = random.randint(0, len(self.bits) - 1)
            if temp[index] == "0":
                temp[index] = "1"
            elif temp[index] == "1":
                temp[index] = "0"
        self.bits = "".join(temp)
```

DETECTING ERRORS BY FINDING CHECKSUM

```
class DataLinkLayer():
    def __init__(self, bits, generator):
        self.bits = bits
        self.keyLength = len(generator)
        self.appendedData = self.bits + "0" * (self.keyLength - 1)
        self.generator = generator

    def CRCdetectError(self):
        divisor = self.generator
        divident = self.appendedData
        numBits = len(self.generator)
        subpartSubstring = self.appendedData[0: numBits]

        while numBits < len(self.appendedData):
            if subpartSubstring[0] == '1':
                subpartSubstring = self.XOR(
                    self.generator, subpartSubstring) + self.appendedData[numBits]

            else:
                subpartSubstring = self.XOR(
                    '0'*numBits, subpartSubstring) + divident[numBits]
            numBits += 1
        if subpartSubstring[0] == '1':
            subpartSubstring = self.XOR(divisor, subpartSubstring)
        else:
            subpartSubstring = self.XOR('0' * numBits, subpartSubstring)
        checksum = subpartSubstring
        return checksum

    def XOR(self, messagePartition, generator):
        self.xor = ""
        for bit1, bit2 in zip(messagePartition, generator):
            if bit1 == bit2:
                self.xor = self.xor + "0"
            else:
                self.xor = self.xor + "1"
        return self.xor[1:]
```

CRC CODE SNIPPETS – SERVER SIDE – ENCODING LOGIC

```
def decode(self, generator):
    temp = ""
    print("\nManchester Encoding                : \n" + str(self.bits))
    yVal = [int(x) for x in list(self.bits)]
    temp = []
    for val in yVal:
        if val == 0:
            temp.append(-1)
        else:
            temp.append(1)
    yVal = temp
    self.manchesterYVal = yVal
    temp = ""
    for i in range(0, len(self.bits), 2):
        s = self.bits[i: i + 2]
        if s == "01":
            temp += "1"
        elif s == "10":
            temp += "0"
    self.bits = temp
    tempOriginalYVal = [int(x) for x in self.bits]
    for y in tempOriginalYVal:
        self.originalYVal.append(y)
        self.originalYVal.append(y)
    print("\nOriginal Encoding With CRC Remainder     : \n" + str(self.bits))
    choice = input(
        "\nPress 1 for delibertely inserting error bits yes else give any other input : ")
    if choice == "1":
        temp = list(self.bits)

        for i in range(5):
            index = random.randint(0, len(self.bits) - 1)
            if temp[index] == "0":
                temp[index] = "1"
            elif temp[index] == "1":
                temp[index] = "0"
        self.bits = "".join(temp)
```

OUTPUTS
-CLIENT SIDE

```
PS E:\6th_sem\Networks\Socket-Programming-master> py client.py
Welcome Client
Enter the generator function in 0s and 1s : 10000
Received message is  1001011001101010100101101001100110010110010110101001011001011010100101100101010110010101
1001101010101010

Manchester Encoding                 :
1001011001101010100101101001100110010110010110101001011001011010100101100101010110010101100110101010101010

Original Encoding With CRC Remainder     :
0110100001100101011011000110110001101111011101000000

Press 1 for delibertely inserting error bits yes else give any other input :

CRC Remainder           :
0000

No Error in bits transmitted from current frame.

Original Encoding Without CRC Remainder :
0110100001100101011011000110110001101111011101000000

Decoded Frame Message     : hellot
---------------------------------------
Received message is  1001011001101010100101101001100110010101101001101001011010011001101010101010
```

OUTPUTS
-CLIENT SIDE

```
Manchester Encoding                  :
1001011001101010100101101001100110010101101001101001011010011001101010 10

Original Encoding With CRC Remainder     :
0110100001100101011100100110010100 00

Press 1 for delibertely inserting error bits yes else give any other input :

CRC Remainder            :
0000

No Error in bits transmitted from current frame.

Original Encoding Without CRC Remainder :
011010000110010101110010011001 01

Decoded Frame Message     : here
------------------------------------

Final Decoded Message : hellothere
------------------------------------
```

OUTPUTS – SERVER SIDE

```
PS E:\6th_sem\Networks\Socket-Programming-master> py server.py
Socket created and binded to port number 10000
Socket listening.

Enter the message you want to send : hellothere

----------------------------------------------------------------

Example of a generator function = x^3 + x + x^0 = "1" + "0" + "1" + "1" = 1011
Example of a generator function = x^2 + x = "1" + "0" + "0" = 110

----------------------------------------------------------------



Enter the generator function in (0s, 1s) : 10000

Waiting for client...
Enter frame size6

Sending Frame Number 0

Client connection received ('127.0.0.1', 50565)

Original Message Bits                 :
011010000110010101101100011011000110111101110100

CRC Value                             :
0000

Original Message Bits With CRC Value  :
011010000110010101101100011011000110111101101000000

Manchester Encoding                   :
1001011001101010100101101001100110010110010110101001011001011010100101100101010110010101100110101010101

Encoded input to    : 1001011001101010100101101001100110010110010110101001011001011010100101100101010
11001101010101010
```

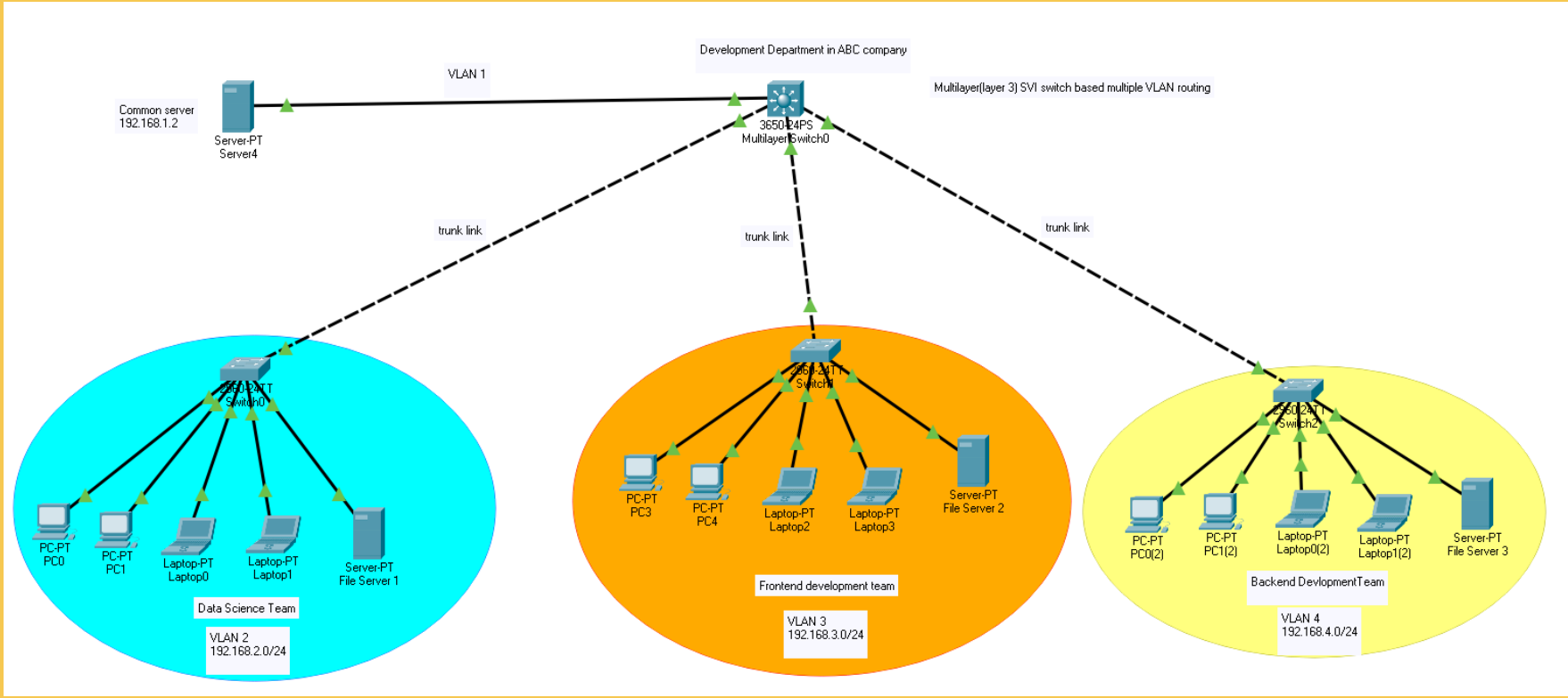
OUTPUTS
- SERVER SIDE

```
Sending Frame Number 1

Client connection received ('127.0.0.1', 50569)

Original Message Bits                :
01101000011001010111001001100101

CRC Value                            :
0000

Original Message Bits With CRC Value  :
01101000011001010111001001100101 0000

Manchester Encoding                  :
10010110011010101001011010011001100101011010011010010110100110011010101 0

Encoded input to    : 10010110011010101001011010011001100101011010011010010110100110011010101 0
-------------------------------------
```

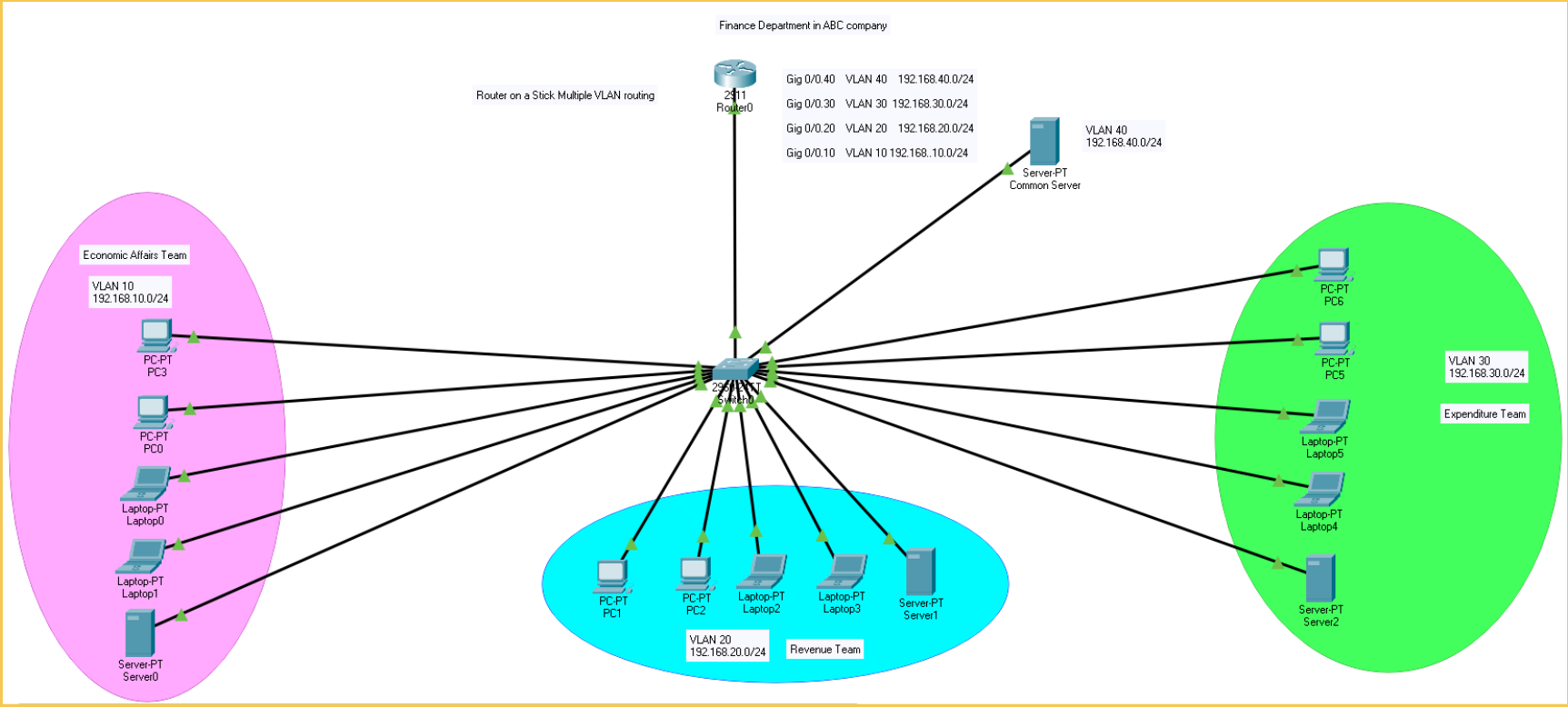
OUTPUTS
- SERVER SIDE

```
Sending Frame Number 1

Client connection received ('127.0.0.1', 50569)

Original Message Bits                :
01101000011001010111001001100101

CRC Value                            :
0000

Original Message Bits With CRC Value  :
01101000011001010111001001100101 0000

Manchester Encoding                  :
10010110011010101001011010011001100101011010011010010110100110011010101 0

Encoded input to    : 10010110011010101001011010011001100101011010011010010110100110011010101 0
-------------------------------------
```

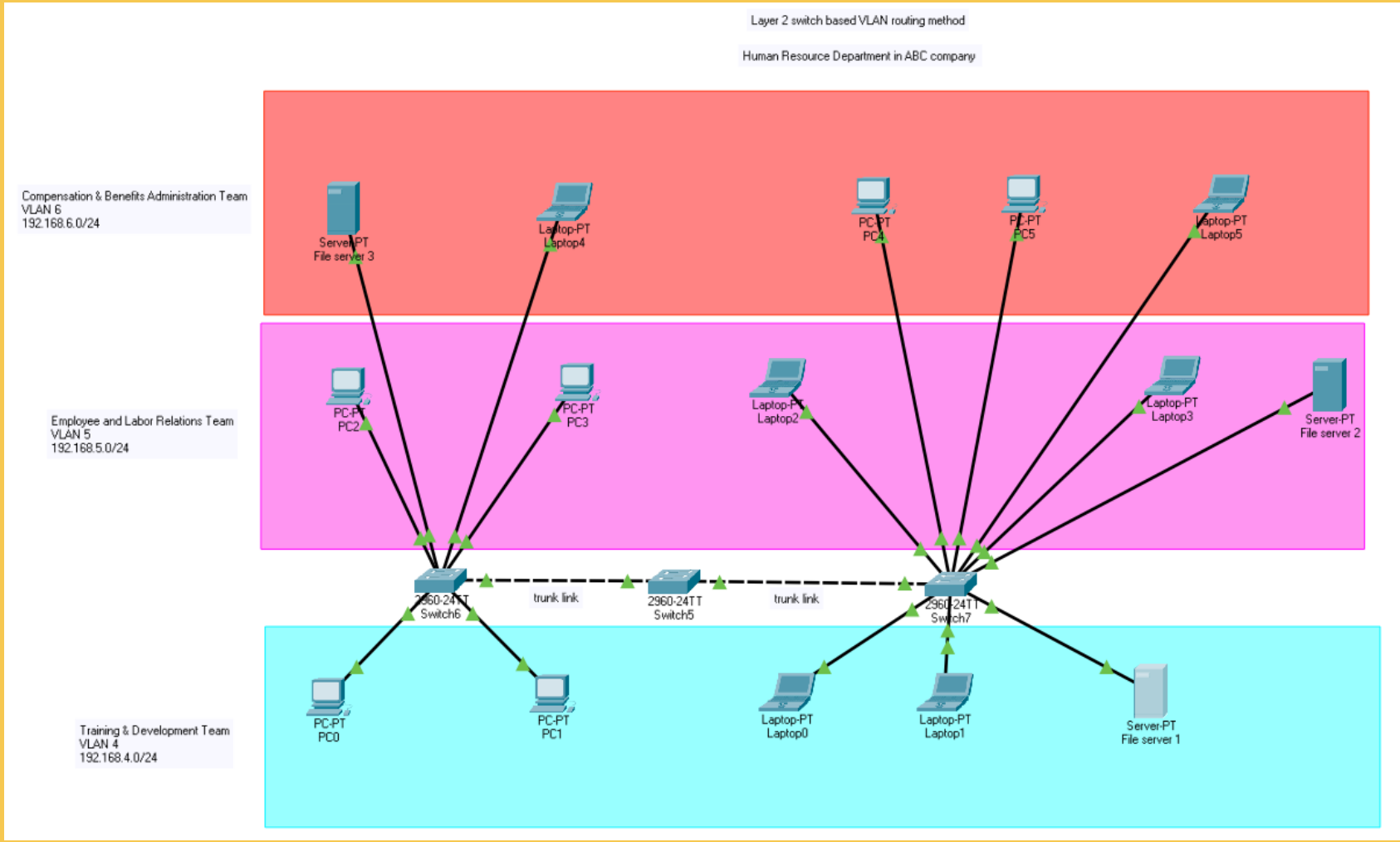
PACKET TRACER DIAGRAMS – DEVELOPMENT DEPARTMENT



PACKET TRACER DIAGRAMS – FINANCE DEPARTMENT



PACKET TRACER DIAGRAMS – HR DEPARTMENT



Thank You

Hope you enjoyed the presentation as much as we enjoyed making and presenting it.