

# NETWORKS CASE STUDY

Chat Server - Group 17

May 2021

## **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

# **TODAY'S PRESENTATION**

## **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

b.) Client: -

- i.) Windows (ex. mIRC, Bersirc, ChatZilla, etc.,)
- ii.) Linux (ex. Xchat, HexChat, etc.,)
- iii.) Mac OS X (ex. IRCCle, 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

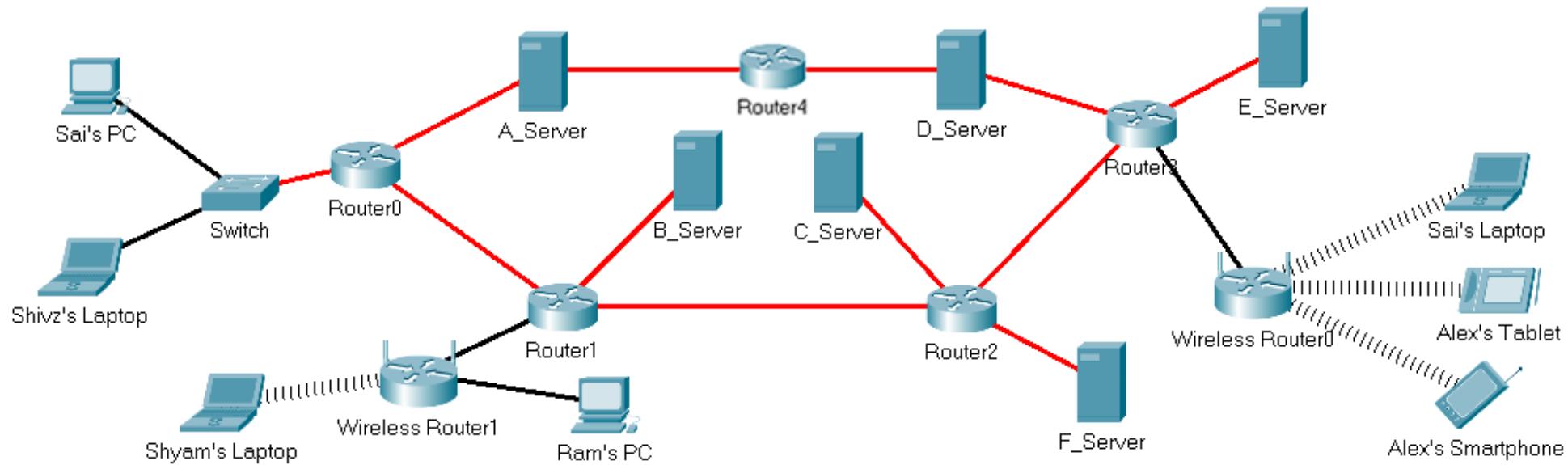
The traditional IRC does not require any cloud services but a few modern-day IRC's are using a cloud-based approach and are quite successful. Ex. Amazon AWS used by IRCCloud (an IRC application using modern approaches), Azure, etc.,

# Programming Languages

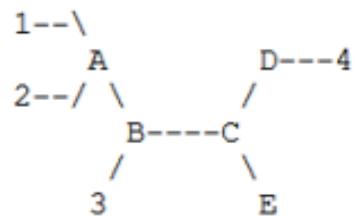


C, C++, Python, Java, mIRC scripting language (msl)  
Lisp, etc.,

# ARCHITECTURE DIAGRAM



A Small Sample IRC Network



Servers: A, B, C, D, E

Clients: 1, 2, 3, 4

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.

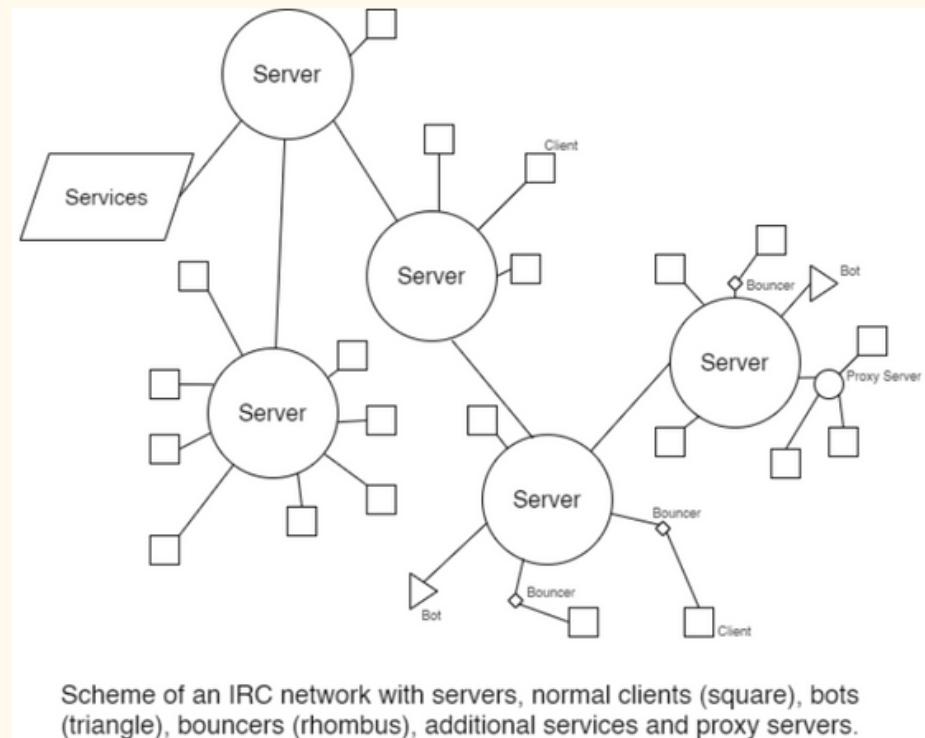
When an user sends a message, it is relayed to the server, which then holds the message and then sends it to any user who is connected to the server. Sometimes given the size of the userbase one server might not be able to accommodate all the users and so multiple servers might be needed for one room, 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.

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 = $100\% * (\text{transferred} - \text{retransmitted}) / \text{transferred}$ Network Loss = $100 - \text{Efficiency}$
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 = $\text{Message size} / \text{Bandwidth}$
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 are 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 its 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

```
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())

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

```
import socket, threading

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

def receive():
    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()
```

▲ 11 ✅

# OUTPUTS

The image shows six Command Prompt windows arranged in a grid, illustrating a multi-user chat application. The windows are labeled as follows:

- Top-left: Command Prompt - py Chat\_Client.py
- Top-middle: Command Prompt - py Chat\_Server.py
- Top-right: Command Prompt - py Chat\_Client.py
- Middle-left: Command Prompt - py Chat\_Client.py
- Middle-middle: Command Prompt - py Chat\_Client.py
- Middle-right: Command Prompt - py Chat\_Client.py

The chat session details are as follows:

- User A (Sai Tharun):** Enters the chat and starts a conversation with the server.
- Server:** Starts the chat server on port 6000.
- User B (Sudharshan):** Enters the chat and greets the server.
- Server:** Greets user B.
- User C (Tanya):** Enters the chat and greets the server.
- Server:** Greets user C.
- Admin:** Enters the chat and maintains rules.
- Users:** Engage in a discussion about being here together and maintaining rules.

# 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

```
# 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], ")")

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

```
file(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)

file(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 : □
```

# Thank You

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