REPORT

Shubham(IWM2017004), Aman Gupta(IWM2017006)

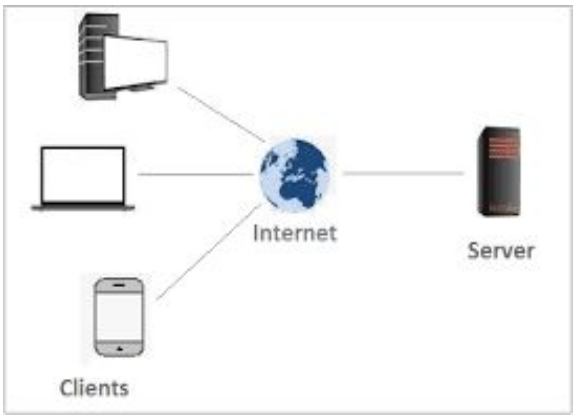
We used a client-server model for our application . In this model computers such as servers provide the network services to the other computers such as clients to perform user based tasks.

Client

- A client is a program that runs on the local machine requesting service from the server. A client program is a finite program means that the service started by the user and terminates when the service is completed.

Server

- A server is a program that runs on the remote machine providing services to the clients. When the client requests for a service, then the server opens the door for the incoming requests, but it never initiates the service.
- Server program is an infinite program means that when it starts, it runs infinitely unless the problem arises. The server waits for the incoming requests from the clients. When the request arrives at the server, then it responds to the request.



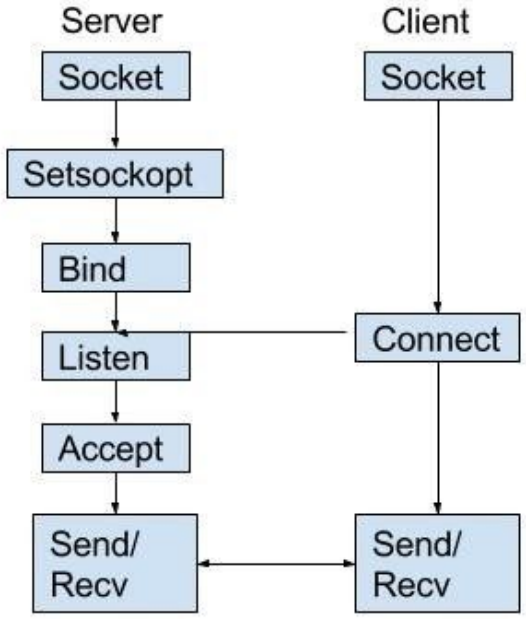
The client establishes a connection to the server over a local area network (LAN) or wide-area network (WAN), such as the Internet. Once the server has fulfilled the client's request, the connection is terminated.

Client-Server Protocols

We used TCP/IP protocol suite to communicate with servers. TCP is a connection-oriented protocol, which means a connection is established and maintained until the application programs at each end have finished exchanging messages. It determines how to break application data into packets that networks can deliver, sends packets to and accepts packets from the network layer, manages flow control and handles retransmission of dropped or garbled packets as well as acknowledgement of all packets that arrive.

Socket Programming

We used Socket programming in python to implement client-server model , Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while another socket reaches out to the other to form a connection. Server forms the listener socket while the client reaches out to the server.



Hardware

- They can compromise on a single processor, 8GB RAM but have to remember that when they grow, then the server will also need to be upgraded/ replaced.
- **Space** :- Initially we have taken 1TB Space.(Note : According to size of client -server model you increase or decrease it).
- **Ram**:-At least 256 MB of RAM. The amount of RAM needed depends on the number of concurrent client connections.(Note : According to size of client -server model you increase or decrease it).

- Display monitor, Keyboard, Mouse as required.
- Webcam or any camera device to connect to monitor.
- Microphone and speaker for audio streaming.
- Any computer that is to be connected to a network, needs to have a **network interface card (NIC).**Most modern computers have these devices built into the motherboard, but in some computers you have to add an extra expansion card.
- **Network Cable :-** To connect together different devices to make up a network, you need cables.Cables are still used in most networks, rather than using only wireless, because they can carry much more data per second, and are more secure (less open to hacking).
- **Hub :-** A hub is a device that connects a number of computers together to make a LAN.

Software

- Python:- You need Python 3.7.x or above.
- Module :- Pyaudio, Wave, Socket, Thread, cv2.

Operating System

- Linux , Windows , Mac.

Protocol

- IPV4 :- At the Network layer we used IPV4 protocol.pv4 (Internet Protocol version 4) Internet address is 32 bit integers. The IP stands for Internet Protocol.
- TCP :- At Transport Layer we used TCP (Transmission Control Protocol) .
- ARP (Address Resolution Protocol):- Functions performed at this level include encapsulation of IP datagrams into the frames transmitted by the network and mapping of IP addresses to the physical addresses used by the network (provided by ARP protocol).
- FTP:File Transfer Protocol is used while opening , closing, reading and writing of file.

Network

- LAN(Local Area Network) : IN LAN we used Ethernet.

FILE TRANSFER

Below is the algorithm to send a file(Audio,Video,text) from a local server to a local client. To let the server interact with multiple clients, we used multi-threading

Server :-

- Import all the required libraries and initialize the variable port, ip,buffer_size.
- Create a socket, IPv4 Protocol is used with Tcp (connection oriented) Communication type.
- After creation of the socket, bind function binds the socket to the address and port.
- Listen puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection.
- Accept extracts the first connection request on the queue of pending connections.
- When the connection is established a new thread for this client is made .
- Input is taken for the file which is to be transferred. Then that file is open and its descriptor is saved in f .
- Stream of buffer_size data is transferred to the client. Then the file is closed.

```
import socket
from threading import Thread

TCP_IP = 'localhost'
TCP_PORT = 9001
BUFFER_SIZE = 1024

// Socket creation
tcpsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tcpsock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
tcpsock.bind((TCP_IP, TCP_PORT)) // Socket Binding
threads = []

class ClientThread(Thread):

//thread Initialization
    def __init__(self,ip,port,sock):
        Thread.__init__(self)
        self.ip = ip
        self.port = port
        self.sock = sock
        print " New thread started for "+ip+":"+str(port)


    def run(self):
        filename=sys.stdin.readline()   // input file to be transfered
        f = open(filename,'rb')    // open a file
        while True:
            l = f.read(BUFFER_SIZE)
            while (l):
                self.sock.send(l)
                #print('Sent ',repr(l))
                l = f.read(BUFFER_SIZE)
            if not l:
                f.close()
                self.sock.close()
                break
```

```
while True:
    tcpsock.listen(5)
    print ("Waiting for incoming connections...")
    (conn, (ip,port)) = tcpsock.accept()
    print ('Got connection from ', (ip,port))
    newthread = ClientThread(ip,port,conn)
    newthread.start()
    threads.append(newthread)
```

Client

- Import all the required libraries.
- Create a socket, IPv4 Protocol is used with Tcp (connection oriented) Communication type.
- Connects the socket with Ip and port number.
- After the connection is formed a new file named receive_file is created and opened and file descriptor will be stored in f.
- All the stream of buffer_size data will be stored in the received_file as the file data is transferred as stream of buffer_size data by the server.

```
import socket
TCP_IP = 'localhost'
TCP_PORT = 9001
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
with open('received_file', 'wb') as f:  # create a new file & open it.
    print ('file opened')
    while True:
        #print('receiving data...')
        data = s.recv(BUFFER_SIZE)
        print('data=%s', (data))
        if not data:
            f.close()
            print 'file close()'
            break
        # write data to a file
        f.write(data)

print('Successfully get the file')
s.close()
print('connection closed')
```

CHAT

Below algorithm setup Chat Room server and allow multiple clients to connect to it using a client-side script.

Server

- Firstly import all the required libraries.
- Create a socket with tcp communication type and ipv4 protocols.
- We take an ip and port through the command prompt so we check if the argument is 3 or not , if not 3 then exit.
- Then we bind the socket with port number and ip address.and call listen and accept.
- When connection is formed with the client , a client is added in a list and a new thread will be started for that client.
- Messages received from the client will be broadcasted to each client.

```
import socket
import select
import sys
from thread import *

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

if len(sys.argv) != 3:
    print ("Correct usage: script, IP address, port number")
    exit()

IP_address = str(sys.argv[1])
Port = int(sys.argv[2])

server.bind((IP_address, Port))
server.listen(5)    //listens for 100 active connections
list_of_clients = []


def clientthread(conn, addr):
    # sends a message to the client whose user object is conn
    conn.send("Welcome to this chatroom!")
    while True:
          try:
                message = conn.recv(2048)
                if message:
                    print ("<" + addr[0] + "> " + message)
                    # Calls broadcast function to send message to all
                    message_to_send = "<" + addr[0] + "> " + message
                    broadcast(message_to_send, conn)
                else:
                    #if connection broken
                    remove(conn)
            except:
                Continue
```

```
#we send the message to all  clients except sender client.
def broadcast(message, connection):
    for clients in list_of_clients:
        if clients!=connection:
            try:
                clients.send(message)
            except:
                clients.close()

                # if the link is broken, we remove the client
                remove(clients)

#this simply removes the object  from the list.
def remove(connection):
    if connection in list_of_clients:
        list_of_clients.remove(connection)


while True:

    #Accepts a connection request and stores socket object for that user as
     conn and Ip address of that client as addr.
    conn, addr = server.accept()

    #Maintains a list of clients for ease of broadcasting
        a message to all available people in the chatroom
    list_of_clients.append(conn)

    # prints the address of the user that just connected
    print (addr[0] + " connected")

    # creates and individual thread for every user that connects
    start_new_thread(clientthread,(conn,addr))

conn.close()
server.close()
```

Client

- Import all the required libraries.
- Create a socket, IPv4 Protocol is used with Tcp (connection oriented) Communication type.
- Connects the socket with Ip and port number.
- After the connection is formed, if the message comes from the server then it will be printed and if the message is typed it will be sent to the server.

```
import socket
import select
import sys

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
if len(sys.argv) != 3:
        print ("Correct usage: script, IP address, port number")
        exit()
IP_address = str(sys.argv[1])
Port = int(sys.argv[2])
server.connect((IP_address, Port))

while True:
        # maintains a list of possible input streams
        sockets_list = [sys.stdin, server]

        #Either the message is send by server to clients
           or client will take input and send it to server.
        read_sockets,write_socket, error_socket =
        select.select(sockets_list,[],[])

        for socks in read_sockets:
                if socks == server:
                        message = socks.recv(2048)
                        print (message)
                else:
                        message = sys.stdin.readline()
                        server.send(message)
                        sys.stdout.write("<You>")
                        sys.stdout.write(message)
                        sys.stdout.flush()
server.close()
```

AUDIO STREAM

Below algorithm shows how the audio streaming happens using the client server model.

Server

- First of all we import sockets which is necessary.
- Then we made a socket object and reserved a port on our pc.
- After that we binded our server to the specified port. Passing an empty string means that the server can listen to incoming connections from other computers as well. If we would have passed 127.0.0.1 then it would have listened to only those calls made within the local computer.
- After that we put the server into listen mode.5 here means that 5 connections are kept waiting if the server is busy and if a 6th socket tries to connect then the connection is refused.
- To do the audio stream we used , PyAudio provides Python bindings for PortAudio, the cross-platform audio I/O library. With PyAudio, you can easily use Python to play and record audio on a variety of platforms, such as GNU/Linux, Microsoft Windows, and Apple Mac OS X / macOS.
- Pyaudio has a different mode we used callback mode, PyAudio will call a specified callback function (2) whenever it needs new audio data (to play) and/or when there is new (recorded) audio data available. Note that PyAudio calls the callback function in a separate thread. The function has the following signature callback(<input_data>, <frame_count>, <time_info>, <status_flag>) and must return a tuple containing frame_count frames of audio data and a flag signifying whether there are more frames to play/record.
- Start processing the audio stream using pyaudio.Stream.start_stream() (4), which will call the callback function repeatedly until that function returns pyaudio.paComplete.
- At last we make a while loop and start to accept all incoming connections and close those connections after a thank you message to all connected sockets.

```
This is the server side program for Audio Stream.


function PyAudio()
{
        This will intiate pyaudio module.
        It will not return anything.
}

function socket( argument 1 , argument 2)
{
        The first parameter is AF_INET and the second one is SOCK_STREAM.
        AF_INET refers to the address family ipv4.
        The SOCK_STREAM means connection oriented TCP protocol.
}
function bind (argument 1 , argument 2)
{
        binds server to a specific ip and port so that
        it can listen to incoming requests on that ip and port
}
function callback(argument 1 , argument 2 , argument 3, argument 4)
{
        PyAudio will call a specified callback function
        henever it needs new audio data (to play) and/or when
         there is new (recorded) audio data available.
        callback(<input_data>, <frame_count>, <time_info>, <status_flag>)
        status flag tells whether there are more frames to play/record.
        returns frame_count  frames of audio data.
}

function open(argumen1 argumen2 argumen3 argumen4 argument 5)
{
        FORMAT = pyaudio.paInt16
        CHANNELS = 1
        RATE = 44100
        INPUT = TRUE
        CHUNK = 4096

        This will open the stream for recording.
}

function start_stream()
{
        starts the stream.
}
```

```
try:
    while True:
        readable, writable, errored = select.select(read_list, [], [])
        for s in readable:
            if s is serversocket:
                (clientsocket, address) = serversocket.accept()
                read_list.append(clientsocket)
                print "Connection from", address
            else:
                data = s.recv(1024)
                if not data:
                    read_list.remove(s)
except KeyboardInterrupt:
    pass


serversocket.close()
# stop Recording
stream.stop_stream()
stream.close()
audio.terminate()
```

Client

- First of all we make a socket object.
- Then we connect to localhost on port 12345 (the port on which our server runs)
- Then we initialize pyaudio to stream audio.
- Now open the stream for audio streaming.
- And lastly we receive data from the server and close the connection.

```
This is the client side program for Audio Stream.



function socket( argument 1 , argument 2)
{
        The first parameter is AF_INET and the second one is SOCK_STREAM.
        AF_INET refers to the address family ipv4.
        The SOCK_STREAM means connection oriented TCP protocol.
}

function connect (argument 1 , argument 2)
{
        connects server to a specific ip and port so that
        it can send requests on that ip and port
}
function PyAudio()
{
        This will intiate pyaudio module.
        It will not return anything.
}

function open(argumen1 argumen2 argumen3 argumen4 argument 5)
{
        FORMAT = pyaudio.paInt16
        CHANNELS = 1
        RATE = 44100
        INPUT = TRUE
        CHUNK = 4096

        This will open the stream for recording.
}


try:
    while True:
        data = s.recv(CHUNK)
        stream.write(data)
except KeyboardInterrupt:
    pass

print('Shutting down')
s.close()
stream.close()
audio.terminate()
```

VIDEO STREAM

Below algorithm shows how the video streaming happens using the client server model.

Server

```
import pickle
import socket
import struct

import cv2

HOST = ''
PORT = 8089

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print('Socket created')

s.bind((HOST, PORT))
print('Socket bind complete')
s.listen(10)
print('Socket now listening')

conn, addr = s.accept()

data = b''
payload_size = struct.calcsize("L")

while True:


    while len(data) < payload_size:
        data += conn.recv(4096)

    packed_msg_size = data[:payload_size]
    data = data[payload_size:]
    msg_size = struct.unpack("L", packed_msg_size)[0]


    while len(data) < msg_size:
        data += conn.recv(4096)

    frame_data = data[:msg_size]
    data = data[msg_size:]


    frame = pickle.loads(frame_data)


    cv2.imshow('frame', frame)
    cv2.waitKey(27)
```

- The server side script will establish a socket and bind it to an IP address and port specified by the user.
- The script will then stay open and receive connection requests, and will append respective socket objects to a list to keep track of active connections.
- After that it will retrieve message size received from the client.
- Then on the basis of message size , it will retrieve all data.
- Extract frames of video from data using pickle library's loads function . which basically convert data into frames.
- Then using open_cv2 we display the video.

Client

- Create a socket, IPv4 Protocol is used with Tcp (connection oriented) Communication type.
- Connects the socket with Ip and port number.
- Capture video frame through open_cv2 .
- Using pickle library's dump function serializes the frame.
- First send a message size to the server.
- Then send the final data.

```
import cv2
import numpy as np
import socket
import sys
import pickle
import struct

cap=cv2.VideoCapture(0)
clientsocket=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
clientsocket.connect(('localhost',8089))

while True:
    ret,frame=cap.read()

    data = pickle.dumps(frame)


    message_size = struct.pack("L", len(data))


    clientsocket.sendall(message_size + data)
```