



PROJECT REPORT ON

“Live audio streaming application using socket programming with python.”

Computer Communication Network

Department of ECE

PES University

Bangalore

Faculty in charge

PROF. PRAJEESHA

Submitted by :

Aishwarya T [PES1UGXXXXXX] Akshay
Kumar N [PES1UGXXXXXX]

Sem : V SEMESTER

Sec : F SECTION

CONTENTS:

SI No	Description	Page No:
1	PROBLEM STATEMENT	
2	INTRODUCTION	
3	BLOCK DIAGRAM/ FLOW CHART	
4	PROCEDURE	
5	HARDWARE/ SOFTWARE CODE	
6	RESULT (WITH SCREEN SHOT IF SIMULATION) OR PICTURES (IF HARDWARE)	
7	CONCLUSION	
8	FUTURE SCOPE	
9	REFERENCES	
10	APPENDIX (IF ANY LIKE RFC DOCS, DATA SHEET etc)	

OBJECTIVE/ PROBLEM STATEMENT

To build a live audio streaming application using socket programming with python

INTRODUCTION

Streaming is the continuous transmission of audio files from a server to a client. In simpler terms, streaming is what happens when consumers watch TV or listen to podcasts on Internet-connected devices. With streaming, the media file being played on the client device is stored remotely, and is transmitted a few seconds at a time over the Internet.

SOCKET PROGRAMMING

A *socket* is a communications connection point (endpoint) that you can name and address in a network. Socket programming shows how to use socket APIs to establish communication links between remote and local processes.

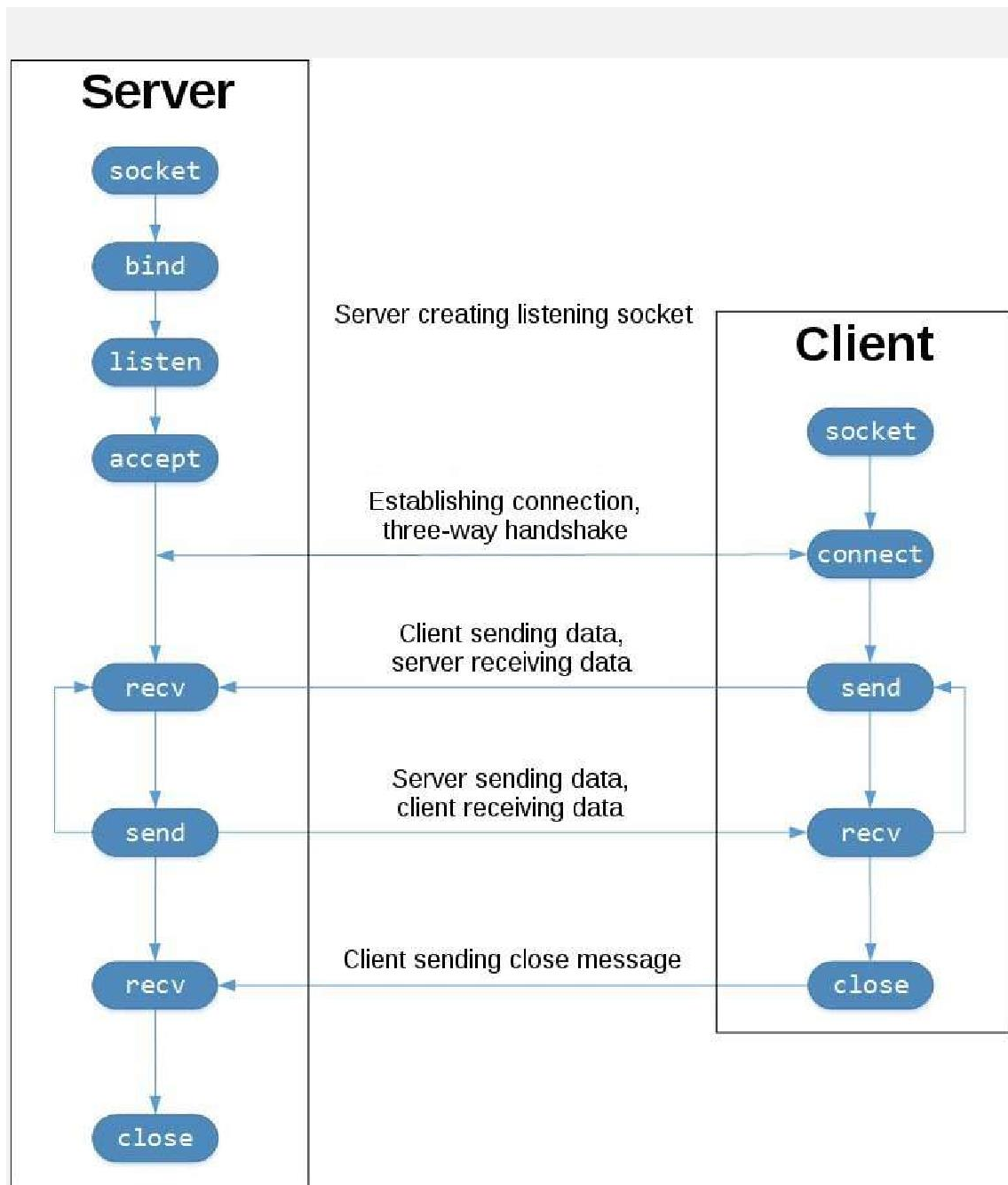
The processes that use a socket can reside on the same system or different systems on different networks. Sockets are useful for both stand-alone and network applications. Sockets allow you to exchange information between processes on the same machine or across a network, distribute work to the most efficient machine, and they easily allow access to centralized data. Socket application program interfaces (APIs) are the network standard for TCP/IP.

Type of sockets are suitable for our objective

The type of sockets that are most suitable are **TCP sockets** as the Transmission Control Protocol (TCP):

- **Is reliable:** packets dropped in the network are detected and retransmitted by the sender.
- **Has in-order data delivery:** data is read by your application in the order it was written by the sender.

FLOW CHART



PROCEDURE:

Task: Step by Step

- At server side, launch the shell script to configure the tc setting for controlling the network condition
- At server side, read the wave file
- At both side, use TCP/UDP to deliver audio signal
- At receiver side, call aplay to real-time play the received audio signal
- At receiver side, save and merge the received packets as the output file ratexx_tcp/udp.wav
- For TCP, log the time-stamp of each received packet
- For UDP, drop the out-of-order packets and pad '0' bits for the lost packets
- Finally, evaluate the quality of the received signal
- For TCP, calculate the per-byte delay
- For UDP, calculate the PSNR

1. Import Socket Library

To use a socket object in your program, start off by importing the socket library. No need to install it with a package manager, it comes out of the box with Python.

```
import socket
```

1.

2. Build Socket Objects

Now we can create socket objects in our code.

```
1. sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

This code creates a socket object that we are storing in the “sock” variable. The constructor is provided a family and type parameter respectively. The family parameter is set to the default value, which is the Address Format Internet.

The type parameter is set to Socket Stream, also the default which enables “sequenced, reliable, two-way, connection-based byte streams” over TCP.

3. Open and Close Connection

Once we have an initialized socket object, we can use some methods to open a connection, send data, receive data , and finally close the connection.

```
1. ## Connect to an IP with Port, could be a URL
2. sock.connect(['0.0.0.0', 8080])
3. ## Send some data, this method can be called multiple times
4. sock.send("Twenty-five bytes to send")
5. ## Receive up to 4096 bytes from a peer
6. sock.recv(4096)
7. ## Close the socket connection, no more data transmission
8. sock.close()
```

Python Socket Client Server

Now that we know a few methods for transmitting bytes, let's create a client and server program with Python.

```
1. import socket
2. serv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
3. serv.bind(('0.0.0.0', 8080))
4. serv.listen(5)
5. while True:
6.     conn, addr = serv.accept()
7.     from_client = ""
8.     while True:
9.         data = conn.recv(4096)
10.        if not data: break
11.        from_client += data
12.        print from_client
13.        conn.send("I am SERVER<br>")
14.        conn.close()
15. print 'client disconnected'
```

Python Socket Client

Here is the `client` socket demo code.

```
1. import socket
2. client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
3. client.connect(('0.0.0.0', 8080))
4. client.send("I am CLIENT<br>")
5. from_server = client.recv(4096)
6. client.close()
7. print from_server
```

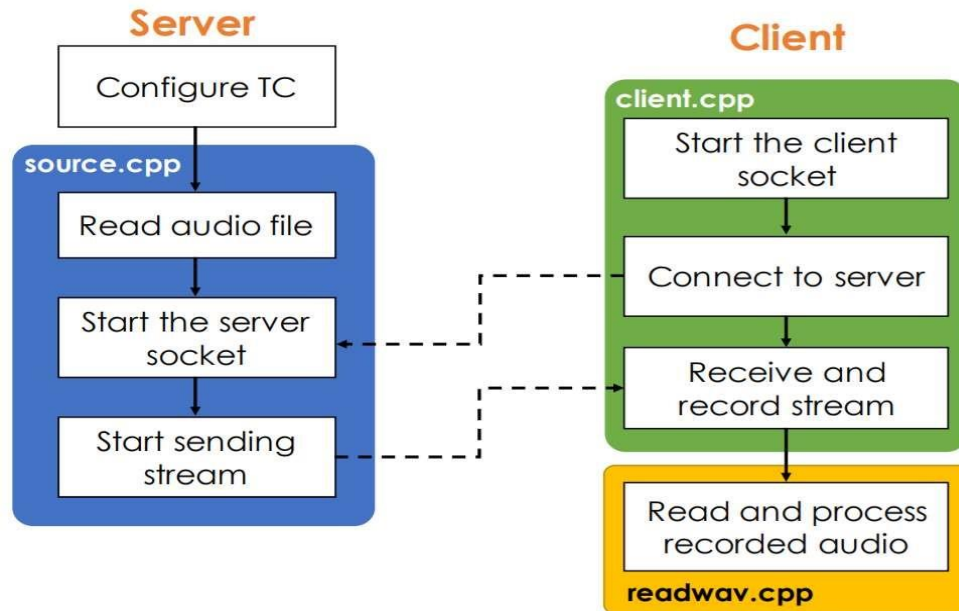
How Does It Work?

This client opens up a socket connection with the server, but only if the server program is currently running. To test this out yourself, you will need to use 2 terminal windows at the same time.

Next, the client sends some data to the server: I am CLIENT

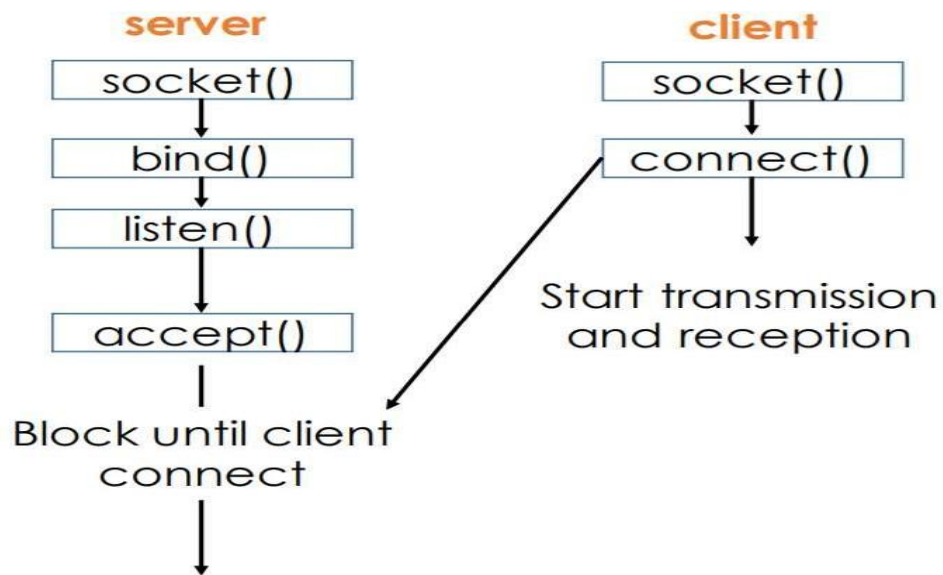
Then the client receives some data it anticipates from the server.

Task diagram



Socket programming

TCP socket



SHELL SCRIPTS

Server.py

```
import socket, cv2, pickle,
struct, time import pyshine as
ps

mode = 'send'
name = 'SERVER TRANSMITTING AUDIO'
audio, context =
ps.audioCapture(mode=mode) #
ps.showPlot(context,name)

# Socket Create
server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)host_ip = '10.20.200.145'
port = 4982
backlog = 5
socket_address = (host_ip, port)
print('STARTING SERVER AT',
socket_address, '...')
server_socket.bind(socket_address)
server_socket.listen(backlog)

while True:
    client_socket, addr =
server_socket.accept() print('GOT
CONNECTION FROM:', addr)
    if client_socket:

        while (True):
            frame = audio.get()

            a = pickle.dumps(frame)
            message = struct.pack("Q",
len(a)) + a
            client_socket.sendall(message)

    else:
        break
```

```
client_socket.close()
```

Client.py

```
import socket, cv2, pickle,
structimport pyshine as ps

mode = 'get'
name = 'CLIENT RECEIVING AUDIO'
audio,          context          =
ps.audioCapture(mode=mode)
ps.showPlot(context, name)

# create socket
client_socket    =          socket.socket(socket.AF_INET,
socket.SOCK_STREAM)host_ip = '10.20.200.145'
port = 4982

socket_address   =   (host_ip,   port)
client_socket.connect(socket_address)
print("CLIENT      CONNECTED      TO",
socket_address)data = b""
payload_size     =
struct.calcsize("Q")   while
True:
    while len(data) < payload_size:
        packet = client_socket.recv(4 *
        1024) # 4Kif not packet: break
        data += packet
        packed_msg_size =
        data[:payload_size]   data =
        data[payload_size:]
        msg_size = struct.unpack("Q", packed_msg_size)[0]

    while len(data) < msg_size:
        data += client_socket.recv(4 *
        1024)   frame_data =
        data[:msg_size]
        data = data[msg_size:]
        frame =
        pickle.loads(frame_data)
        audio.put(frame)

client_socket.close()
```

Traffic Control

- User-space utility program used to configure the Linux kernel packet scheduler
- Shape traffic by
 - Shaping: limited the transmission rate (egress only)
 - Scheduling: scheduling the packet to different “class” (egress only)
 - Policing: deal with reception traffic
- Dropping: If traffic exceeding a set bandwidth, drop the packet (both ingress and egress)

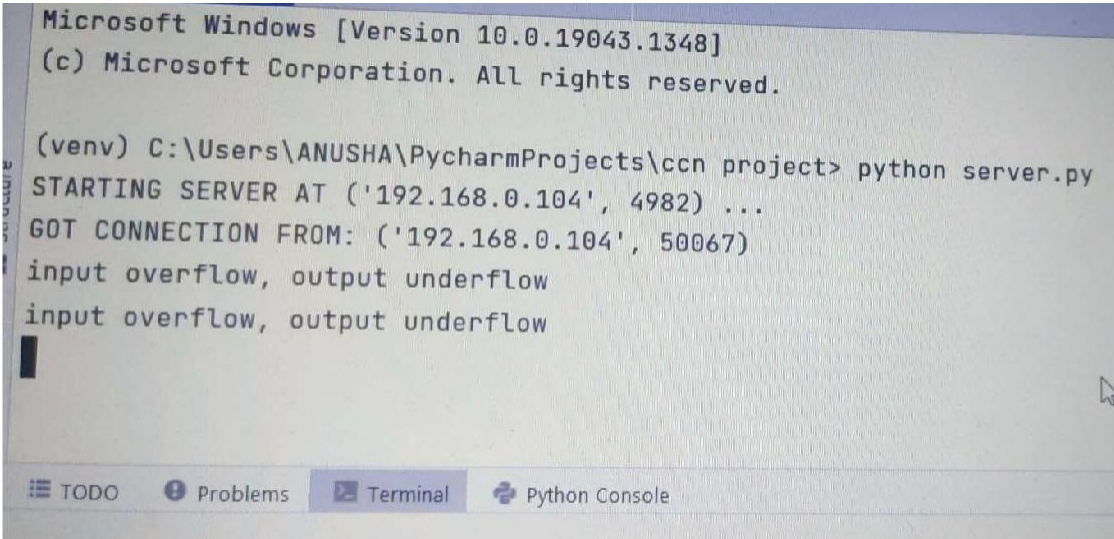
Traffic Control: How to

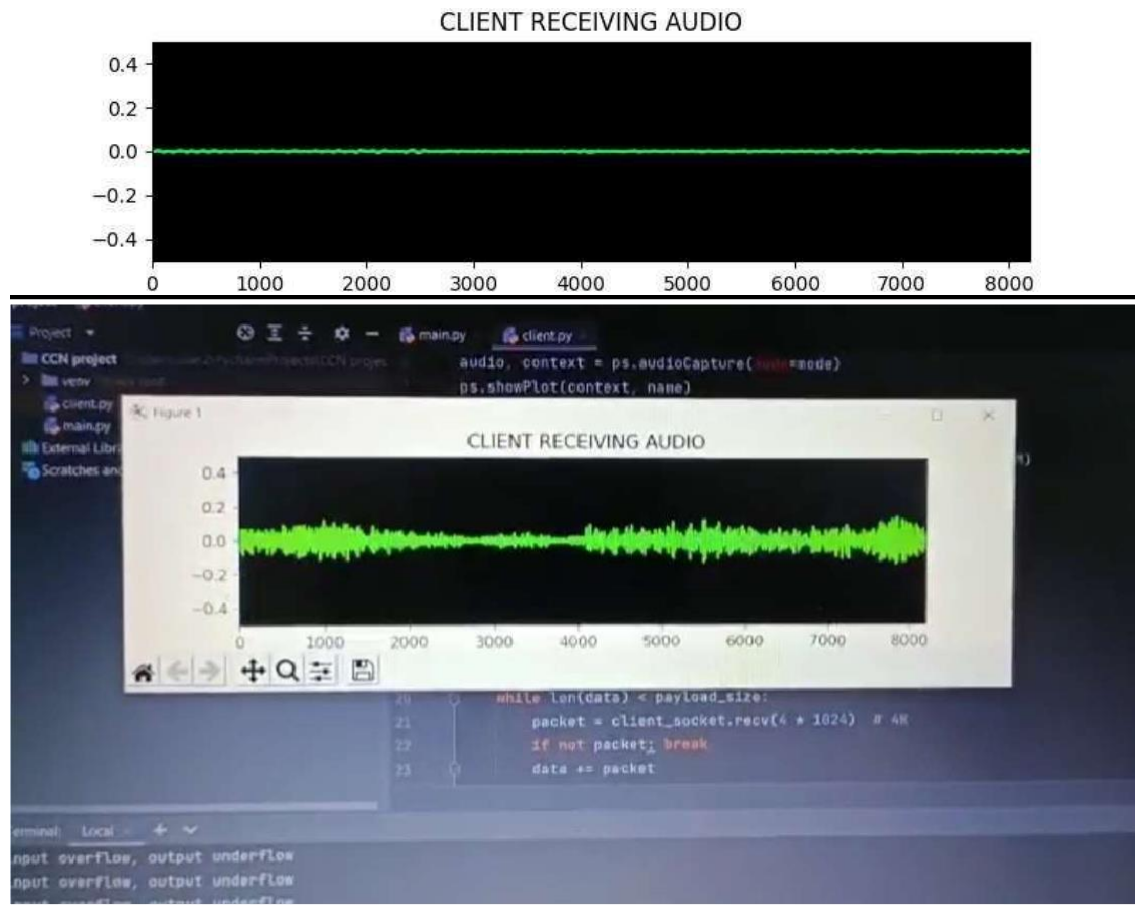
- qdisc
 - Short for “queueing discipline”
 - It is elementary to understanding traffic control
 - class
 - Qdisc contains classes
 - A qdisc may prioritize certain kinds of traffic by dequeuing from certain classes
 - filter
 - A filter is used by a classful qdisc to determine in which class a packet will be enqueued.

RESULTS

```
Microsoft Windows [Version 10.0.19043.1348]
(c) Microsoft Corporation. All rights reserved.

(venv) C:\Users\ANUSHA\PycharmProjects\ccn project> python server.py
STARTING SERVER AT ('192.168.0.104', 4982) ...
GOT CONNECTION FROM: ('192.168.0.104', 50067)
input overflow, output underflow
input overflow, output underflow
█
```





CONCLUSION

We Developed network applications in Python by using sockets, threads, and Web services. This software is portable, efficient, and easily maintainable for large number of clients. Our developed web-based live audio streaming is unique in its features and more importantly easily customizable. Typically, programs running on client machines make requests to programs on a server Machine. These involve networking services provided by the transport layer. The most widely used transport protocols on the Internet are TCP (Transmission control Protocol) and UDP (User Datagram Protocol). TCP is a connection-oriented protocol providing a reliable flow of data between two computers

REFERENCES :

- <https://pyshine.com/How-to-send-audio-from-PyAudio-over-socket/>
- <https://www.twilio.com/docs/voice/tutorials/consume-real-time-media-stream>
- <https://stackoverflow.com/-to-play-audio>

APPENDIX IF ANY