SOCKET PROGRAMMING SERIES

How to send audio data using socket programming in Python

24 Feb 2021 · 19 mins read

Hi! Let's say we have an audio file (.wav), and we want to send it to the client so that the client can listen the stream as a playback in real-time. For this purpose we require PyAudio and socket programming. PyAudio enriches Python bindings for PortAudio, the cross-platform audio I/O library. We will first make codes for the TCP and then go on with the UDP. But before that, please install PyAudio. If you can't install it using pip installer, then please go this link and download the `.whl` according to your Python version. For instance if you are using Python 3.6 then you need to download this `PyAudio-0.2.11-cp36-cp36m-win_amd64.whl` . After that go to the location of this download and open up the power shell or terminal and use the command below:

```
pip3.6 install PyAudio-0.2.11-cp36-cp36m-win_amd64.whl
```

The audio data normally consists of 2 channels, which means the data array will be of shape (CHUNK,2), where CHUNK is the number of data samples representing the digital sound. We commonly put CHUNK as 1024. Below are the TCP based server client codes to provide these data CHUNKS from a server to a client. For more details on socket programming please visit our previous tutorials.

Here is the server side code, we assume that you already have wave (.wav) audio file in the same directory as this server.py file. Please run the server.py at one computer and accordingly provide your host_ip to it.

TCP SOCKET VERSION

server.py

Copy code to clipboard

```
# This is server code to send video and audio frames over TCP

import socket
import threading, wave, pyaudio,pickle,struct

host_name = socket.gethostname()
host_ip = '192.168.1.102'#  socket.gethostbyname(host_name)
print(host_ip)
port = 9611

def audio_stream():
    server_socket = socket.socket()
    server_socket.bind((host_ip, (port-1)))

    server_socket.listen(5)
    CHUNK = 1024
    wf = wave.open("temp.wav", 'rb')

    p = pyaudio.PyAudio()
    print('server listening at',(host_ip, (port-1)))


    stream = p.open(format=p.get_format_from_width(wf.getsampwidth()),
                    channels=wf.getnchannels(),
                    rate=wf.getframerate(),
                    input=True,
                    frames_per_buffer=CHUNK)



    client_socket,addr = server_socket.accept()

    data = None
    while True:
```

```
                data = wf.readframes(CHUNK)
                a = pickle.dumps(data)
                message = struct.pack("Q",len(a))+a
                client_socket.sendall(message)

t1 = threading.Thread(target=audio_stream, args=())
t1.start()
```

Usage:

```
python server.py
```

On the same or second computer please run the code below as:

```
python client.py
```

client.py

Copy code to clipboard

```
# Welcome to PyShine
# This is client code to receive video and audio frames over TCP

import socket,os
import threading, wave, pyaudio, pickle,struct
host_name = socket.gethostname()
host_ip = '192.168.1.102'#  socket.gethostbyname(host_name)
print(host_ip)
port = 9611
def audio_stream():

        p = pyaudio.PyAudio()
        CHUNK = 1024
        stream = p.open(format=p.get_format_from_width(2),
                                        channels=2,
                                        rate=44100,
                                        output=True,
                                        frames_per_buffer=CHUNK)

        # create socket
        client_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        socket_address = (host_ip,port-1)
        print('server listening at',socket_address)
        client_socket.connect(socket_address)
        print("CLIENT CONNECTED TO",socket_address)
        data = b""
        payload_size = struct.calcsize("Q")
        while True:
                try:
                        while len(data) < payload_size:
                                packet = client_socket.recv(4*1024) # 4K
                                if 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)
                        stream.write(frame)

                except:

                        break

        client_socket.close()
        print('Audio closed')
        os._exit(1)

t1 = threading.Thread(target=audio_stream, args=())
t1.start()
```

If everything goes well you will listen the good quality sound at the client side.

UDP SOCKET VERSION

Alright, lets do the same things as above but this time using UDP socket. The process of reading audio data should be streamed smoothly, in case of UDP. The `wf.readframes(CHUNK)` returns at most CHUNK frames of audio, as a bytes object. These bytes are then sent to the client address using `server_socket.sendto(data,client_addr)` . If we don't put enough wait, the receiver will overload and the reliability that all samples are properly sent, will be highly compromised and could even result in no sound or exceptin raised. We need to put a wait using `time.sleep()` , before sending the next CHUNK of audio data. Since we know that the sample rate of audio is 44100 samples per second, so it means that the time for one sample to send is `1/sample_rate` . In this context, all the samples in a CHUNK would require a time of `CHUNK/sample_rate` . Therefore, after sending a CHUNK we will put a `time.sleep(CHUNK/sample_rate)` . Here is the server side code:

server.py

Copy code to clipboard

```
# This is server code to send video and audio frames over UDP

import socket
import threading, wave, pyaudio, time

host_name = socket.gethostname()
host_ip = '192.168.1.102'#  socket.gethostbyname(host_name)
print(host_ip)
port = 9633
# For details visit: www.pyshine.com

def audio_stream_UDP():

    BUFF_SIZE = 65536
    server_socket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
    server_socket.setsockopt(socket.SOL_SOCKET,socket.SO_RCVBUF,BUFF_SIZE)

    server_socket.bind((host_ip, (port)))
    CHUNK = 10*1024
    wf = wave.open("temp.wav")
    p = pyaudio.PyAudio()
    print('server listening at',(host_ip, (port)),wf.getframerate())
    stream = p.open(format=p.get_format_from_width(wf.getsampwidth()),
                    channels=wf.getnchannels(),
                    rate=wf.getframerate(),
                    input=True,
                    frames_per_buffer=CHUNK)

    data = None
    sample_rate = wf.getframerate()
    while True:
        msg,client_addr = server_socket.recvfrom(BUFF_SIZE)
        print('GOT connection from ',client_addr,msg)

        while True:
            data = wf.readframes(CHUNK)
            server_socket.sendto(data,client_addr)
            time.sleep(0.8*CHUNK/sample_rate)



t1 = threading.Thread(target=audio_stream_UDP, args=())
t1.start()
```

Since in UDP there is no handshake, so at the receiver side, we have to store each received datagram in a queue. For this purpose our queue will serve as a buffer of some size (let's say 100). On a thread we will continue to receive the UDP datagram of audio data. On the other hand, in a while loop we will playback the sound. A time.sleep(5), will provide 5 second delay at the receiver side to fill the buffer, you can change it according to your requirements.

client.py

Copy code to clipboard

```
import socket
import threading, wave, pyaudio, time, queue

host_name = socket.gethostname()
host_ip = '192.168.1.102'#  socket.gethostbyname(host_name)
print(host_ip)
port = 9633
# For details visit: www.pyshine.com
q = queue.Queue(maxsize=2000)

def audio_stream_UDP():
        BUFF_SIZE = 65536
        client_socket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
        client_socket.setsockopt(socket.SOL_SOCKET,socket.SO_RCVBUF,BUFF_SIZE)
        p = pyaudio.PyAudio()
        CHUNK = 10*1024
        stream = p.open(format=p.get_format_from_width(2),
                                        channels=2,
                                        rate=44100,
                                        output=True,
                                        frames_per_buffer=CHUNK)

        # create socket
        message = b'Hello'
        client_socket.sendto(message,(host_ip,port))
        socket_address = (host_ip,port)

        def getAudioData():
                while True:
                        frame,_= client_socket.recvfrom(BUFF_SIZE)
                        q.put(frame)
                        print('Queue size...',q.qsize())
        t1 = threading.Thread(target=getAudioData, args=())
        t1.start()
        time.sleep(5)
        print('Now Playing...')
        while True:
                frame = q.get()
                stream.write(frame)

        client_socket.close()
        print('Audio closed')
        os._exit(1)



t1 = threading.Thread(target=audio_stream_UDP, args=())
t1.start()
```

Please checkl that your MacOS is configured with maximum datagram size as:

```
sudo sysctl -w net.inet.udp.maxdgram=65535
```

The above UDP max buffer size may reduce back to 9216 upon restart. So please run the above above command again if required. Usage: on the server side run:

```
python server.py
```

On the client side please run:

```
python client.py
```

A little bit advanced UDP method

Thanks to Ethan Chocron, who tested the above code and commented that “regarding the UDP audio stream, the code works perfectly as long as the queue size is greater than 1. I understood why. I think it has to do with the quality and power of the computer but the rate that the data is sent and received is slower than the rate at which it is written on the stream, thus when the queue is 1, it plays, waits (no sound is heard) and then plays again, the difference between the rates is very small, but makes a big difference in the quality of the sound”. The main problem is that how to maintain the queue size greater than 1.

over-utilized and get emptied, instead it will continue to increase at a much lower rate, and keeping the max size of queue to 2000 or above. The second solution is to have a feedback from the receiver, this feedback will correct the transmission rate. The third is prior knowledge of data to allocate the queue size accordingly, just like the streaming media players do, as a buffering mechanism. The `print(wf.getnframes())` will show the total number of frames in the audio file, and the respective queue size will be `print(wf.getnframes()/CHUNK)` . In the codes below, we will use the third solution, which seems much better and suits well for multiple machines. The idea is to send the size of audio frames to the client, so that the client can decide its maximum qsize, and then go on loading the buffer at the client size.

server.py

Copy code to clipboard

```
# This is server code to send video and audio frames over UDP

import socket
import threading, wave, pyaudio, time
import math
host_name = socket.gethostname()
host_ip = '192.168.1.104'#  socket.gethostbyname(host_name)
print(host_ip)
port = 9633
# For details visit: www.pyshine.com

def audio_stream_UDP():

        BUFF_SIZE = 65536
        server_socket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
        server_socket.setsockopt(socket.SOL_SOCKET,socket.SO_RCVBUF,BUFF_SIZE)

        server_socket.bind((host_ip, (port)))
        CHUNK = 10*1024
        wf = wave.open("temp.wav.wav")
        p = pyaudio.PyAudio()
        print('server listening at',(host_ip, (port)),wf.getframerate())
        stream = p.open(format=p.get_format_from_width(wf.getsampwidth()),
                                        channels=wf.getnchannels(),
                                        rate=wf.getframerate(),
                                        input=True,
                                        frames_per_buffer=CHUNK)

        data = None
        sample_rate = wf.getframerate()
        while True:
                msg,client_addr = server_socket.recvfrom(BUFF_SIZE)
                print('[GOT connection from]... ',client_addr,msg)
                DATA_SIZE = math.ceil(wf.getnframes()/CHUNK)
                DATA_SIZE = str(DATA_SIZE).encode()
                print('[Sending data size]...',wf.getnframes()/sample_rate)
                server_socket.sendto(DATA_SIZE,client_addr)
                cnt=0
                while True:

                        data = wf.readframes(CHUNK)
                        server_socket.sendto(data,client_addr)
                        time.sleep(0.001) # Here you can adjust it according to how fast you want to sen
                        print(cnt)
                        if cnt >(wf.getnframes()/CHUNK):
                                break
                        cnt+=1

                break
        print('SENT...')

t1 = threading.Thread(target=audio_stream_UDP, args=())
t1.start()
```

client.py

Copy code to clipboard

```
host_name = socket.gethostname()
host_ip = '192.168.1.104'#  socket.gethostbyname(host_name)
print(host_ip)
port = 9633
# For details visit: www.pyshine.com


def audio_stream_UDP():
        BUFF_SIZE = 65536
        client_socket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
        client_socket.setsockopt(socket.SOL_SOCKET,socket.SO_RCVBUF,BUFF_SIZE)
        p = pyaudio.PyAudio()
        CHUNK = 10*1024
        stream = p.open(format=p.get_format_from_width(2),
                                        channels=2,
                                        rate=44100,
                                        output=True,
                                        frames_per_buffer=CHUNK)

        # create socket
        message = b'Hello'
        client_socket.sendto(message,(host_ip,port))
        DATA_SIZE,_= client_socket.recvfrom(BUFF_SIZE)
        DATA_SIZE = int(DATA_SIZE.decode())
        q = queue.Queue(maxsize=DATA_SIZE)
        cnt=0
        def getAudioData():
                while True:
                        frame,_= client_socket.recvfrom(BUFF_SIZE)
                        q.put(frame)
                        print('[Queue size while loading]...',q.qsize())

        t1 = threading.Thread(target=getAudioData, args=())
        t1.start()
        time.sleep(5)
        DURATION = DATA_SIZE*CHUNK/44100
        print('[Now Playing]... Data',DATA_SIZE,'[Audio Time]:',DURATION ,'seconds')
        while True:
                frame = q.get()
                stream.write(frame)
                print('[Queue size while playing]...',q.qsize(),'[Time remaining...]',round(DURATION),'s
                DURATION-=CHUNK/44100
        client_socket.close()
        print('Audio closed')
        os._exit(1)



t1 = threading.Thread(target=audio_stream_UDP, args=())
t1.start()
```

Previous
‹ How to send audio and video...

Next
How to easily install OpenC... ›

Useful Posts

- Fourty important tips to write better python code Published on March 28, 2023
- What is SVD Published on March 2, 2023
- How to make a chatGPT like application Published on February 23, 2023
- Interview questions for python programming jobs Published on February 16, 2023
- Python coding tips for faster and better software development Published on February 14, 2023
- Good python coding examples for the software development Published on February 12, 2023
- What are NP problems Published on February 12, 2023
- How to sort a list using Quicksort algorithms Published on February 10, 2023
- How to make an image to text classifier application Published on September 17, 2022
- How to extract text from image in Python Published on September 4, 2022

- Learn Python Tips and Tricks Part 03 Published on July 25, 2022
- Learn Python Tips and Tricks Part 02 Published on July 14, 2022
- Learn Python Tips and Tricks Part 01 Published on May 28, 2022
- UDP Single server to multiple clients Published on May 9, 2022
- Video streaming and Car Control in Python Published on May 3, 2022
- How to send video over UDP socket and save it as MP4 at client side Published on May 3, 2022
- FAQs about PyQt5 Published on April 19, 2022
- Transfer video over sockets from multiple clients and save at server side with a name Published on April 17, 2022
- How to install TVM on MAC OS Published on February 27, 2022
- PytQt5 terminal console Published on February 3, 2022
- PytQt5 Video and Audio GUI with start and stop buttons Published on January 17, 2022
- Test your audible frequency range in Python Published on December 12, 2021
- How to split a pdf into pages in Python Published on August 4, 2021
- How to parse XML file and save the data as CSV Published on July 8, 2021
- Video and Text chat in Python Published on July 7, 2021
- How to perform online video processing from the client's camera Published on June 6, 2021
- How to stream multiple videos on an HTML webpage Published on May 14, 2021
- How to stream video and bidirectional text in socket programming Published on May 4, 2021
- How to easily stream picamera video over wifi with Raspberry Pi Published on April 16, 2021
- How to configure Raspberry Pi in Ad hoc wifi mode Published on April 14, 2021
- How to easily stream webcam video over wifi with Raspberry Pi Published on April 14, 2021
- How to make a simple webcam video recorder GUI in PyQt5 Published on April 8, 2021
- How to open and show the SQL database file in a PyQt5 GUI Published on April 8, 2021
- Interactive Matplotlib GUI with data cursors Published on April 5, 2021
- Basics about SQL database in Python Published on April 5, 2021
- How to make an image to text GUI in Python Published on March 26, 2021
- How to easily install OpenCv in Raspberry Pi boards Published on March 17, 2021
- How to send audio data using socket programming in Python Published on February 24, 2021
- How to send audio and video using socket programming in Python Published on February 20, 2021
- How to send video using UDP socket in Python Published on February 17, 2021
- How to make a Matplotlib and PyQt5 based GUI to plot a CSV file data Published on January 31, 2021
- Play Rock Paper Scissors Game using PyQt5 GUI Published on January 29, 2021
- Lipstick color picker GUI in PyQt5 Published on January 21, 2021
- How to send audio from a client computer to a server over the wifi Published on January 17, 2021
- How to visualize Earthquakes in Python Published on January 16, 2021
- How to publish-subscribe video using socket programming in Python Published on January 1, 2021
- How to send and receive live audio using socket programming in Python Published on December 23, 2020
- How to get audio frames from the microphone device Published on December 14, 2020
- How to deploy Python video processing application on the server Published on December 3, 2020
- Working with multiple threads in PyQt5 Published on November 21, 2020
- PytQt5 Live Audio GUI with start and stop buttons Published on November 19, 2020
- PytQt5 GUI design to plot Live audio data from Microphone Published on November 13, 2020
- PytQt5 GUI design and Video processing with OpenCV Published on November 7, 2020
- OpenCV and Real time streaming protocol (RTSP) Published on November 1, 2020
- A simple cache-server to broadcast video to clients Published on October 29, 2020
- How to make an interactive PSO algorithm in Python Published on October 22, 2020
- Transfer video over sockets from multiple clients Published on October 16, 2020
- Add text with transparent rectangle on an image Published on October 10, 2020
- How to make an image to text converter GUI in Python Published on October 8, 2020
- How to make OpenCV and PyQt5 based GUI for image processing applications Published on October 1, 2020
- How to make a Matplotlib and PyQt5 based GUI Published on October 1, 2020
- How to make a Matplotlib and PyQt5 based GUI with drag and drop the CSV file Published on October 1, 2020
- How to make a calculator GUI in python with PyQt5 Published on September 22, 2020
- Pandas dataframe with hexadecimal and ascii values Published on September 2, 2020
- Free Audio Video Screen Recorder for Windows 10 Published on September 2, 2020
- Socket programming to send and receive webcam video Published on September 1, 2020
- Faster and accurate object tracking in Python Published on September 1, 2020
- How to track Mario in Python Published on September 1, 2020
- What are yield and return statements in Python Published on August 1, 2020
- How to automatically arrange the desktop icons Published on April 18, 2020
- How to plot realtime frame rate of a web camera Published on April 8, 2020
- How to make screen recorder in PyQt5 Published on April 1, 2020
- How to make a real time voice plot Published on March 3, 2020

- How to read and write in xlsx file using python Published on February 3, 2020
- Playing piano with Python Published on February 2, 2020
- Lab4 Training regression model and Epochs Published on August 14, 2019
- Lab3 Train and Test Keras Model Published on August 13, 2019
- Lab2 How to make a basic multilayer Keras model Published on August 10, 2019
- Lab1 Keras Basic Model Published on September 17, 2018
- Basic Coding in TensorFlow Published on September 17, 2018
- With Speech, control the MS Power Point Presentation Published on September 17, 2018
- TensorFlow Basics Published on September 13, 2018
- Installing Pytorch in Windows (GPU version) Published on September 6, 2018
- Installing Pytorch in Windows (CPU version) Published on September 5, 2018
- Importance of One Hot Encoding Published on September 3, 2018
- How to install OpenCV and Python in windows Published on September 1, 2018
- How to install protobuf on Mac OS Published on August 21, 2018

What do you think?

14 Responses

Upvote Funny Love Surprised Angry Sad

3 Comments Login ▾

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS

Name

3 Share Best Newest Oldest

ddgg
8 months ago

Thank you for this amazing post!
I have been trying to tweak a few variables to achieve the lowest latency possible, which I guess can be pretty low in a localhost environment. However removing the 5 seconds wait time and reducing the CHUNK size down to 1024 still results in an audible latency. Does the BUFF_SIZE and the sleep time of the server_socket.sendto() have to do with it? Are they related to each other?

0 0 Reply • Share ›

Alex G
8 months ago

I'm trying to run the server script on AWS Fedora instance and I'm getting:

```
OSError: [Errno -9996] Invalid input device (no default output device)
```

Do you know why it needs a device? All I'm trying to do is to stream a wav file.

Thanks.

0 0 Reply • Share ›

Sassenach
2 years ago

Is there a way to decode a pyaudio stream on an android app ? using java I mean because I want to stream audio from raspberrypi microphone (server) to an android client (or multiple clients) and the android side will self coded as a minimal android app

0 0 Reply • Share ›

Subscribe Privacy Do Not Sell My Data