```
import socket, sys, os, signal, getopt, msgpack, types, pyaudio
sys.path.append('/home/pi/Documents/18500/usb_4_mic_array')
from collections import deque
from string import *
from threading import *
from tuning import Tuning
import usb.core
import usb.util
import time

dev = usb.core.find(idVendor=0x2886, idProduct=0x0018)
mic_tuning = Tuning(dev)

#socket params
SERVER_IP =  '18.221.2.198' #aws public ip
PORT = 12345
CHUNK = 2000
p = pyaudio.PyAudio()

#Test variables
numPacketSent = 0
numPacketReceived = 0
timeOut = 0.0
timeIn = 0.0
latency = 0.0
latenyTotal = 0.0
avgLatency = 0.0
droppedPacketTime = 600
stopSending = False
finishPacketTest = False

'''
Send "Hello Server" to server
and then wait to receive
"Hello from Server" back
'''
def testBasicConnection():
    sock = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
    print('made socket')
    sock.sendto(str.encode("Hello Server") ,(SERVER_IP, PORT))
    print('sent hello')

    data, addr = sock.recvfrom(CHUNK)
    print('data received: ')
    print(data.decode())
    sock.close()

def startClient(processedAudioQIn, isDroppedPacket, isLatency):
```

```
    global processedAudioQ
    global stream
    global p
    global finishPacketTest
    global stopSending

    processedAudioQ = processedAudioQIn
    connect()
    if isLatency:
        testLatency(processedAudioQ)

    if isDroppedPacket:
        finishPacketTest = False
        stopSending = False
        droppedPacketThread = Thread(target=testDroppedPacket, args=(processedAudioQ,))
        droppedPacketThread.start()
        time.sleep(droppedPacketTime)
        stopSending = True
        time.sleep(1)
        finishPacketTest = True
        time.sleep(.1)
        droppedPacketThread.join()

        #print result
        print('----------DROPPED PACKET TEST RESULTS----------')
        print('Test ran for ' + str(droppedPacketTime) + ' seconds')
        print('Packets Sent ' + str(numPacketSent))
        print('Packets Received ' + str(numPacketReceived))
        print('Dropped Packet Rate '+ str((numPacketSent - numPacketReceived) / numPacketSent))
        print('----------END DROPPED PACKET TEST----------')

    stream.stop_stream()
    stream.close()
    p.terminate()

def connect():
    # set up socket
    global s
    global micID
    global stream
    global p
    try:
        s = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
        print("socket created")
    except socket.error:
        print("Could not make socket. Error: ")
        sys.exit()

    #connect
```

```
    #get mic ID
    f = open("micID.txt", "r")
    micID = f.read()

    #set non blocking socket
    s.setblocking(False)

    stream = p.open(
        format = pyaudio.paInt16,
        channels= 1,
        rate = 16000,
        input = True,
        frames_per_buffer = 512,
        input_device_index=1,
        stream_callback=callback,
    )

    stream.start_stream()

def callback(in_data, frame_count, time_info, status):
    global processedAudioQ

    #put audio in queue for signal processing
    processedAudioQ.append((in_data, mic_tuning.direction))

    return(None, pyaudio.paContinue)

def testLatency(processedAudioQ):
    global numPacketSent
    global numPacketReceived
    global timeOut
    global timeIn
    global latency
    global latenyTotal
    global avgLatency
    global stream
    global finishPacketTest

    packet_index = 0
    resTime = [0.0] * 500

    #Start sending and receiving
    print('----------START LATENCY TEST----------')

    try:
        while True:
            #Getting new data
            data = [b'', -1, '']
            gotData = False
```

```
try:
    dataByte, addr = s.recvfrom(CHUNK)
    gotData = True

except BlockingIOError:
    gotData = False

#decode data
unpackedData = False
if gotData:
    try:
        data = msgpack.loads(dataByte)
        unpackedData = True
    except:
        unpackedData = False

lenQ = len(processedAudioQ)
if lenQ > 0:
    (data, doa)  = processedAudioQ.pop()
    outMsg = [b'',0,'']
    outMsg[0] = data
    outMsg[1] = packet_index
    outMsg[2] = micID[0]
    timeOut = time.time()

    if packet_index < 500:
        print('\nPacket sent @ time ' + str(timeOut))
        resTime[packet_index] = timeOut
        s.sendto(msgpack.dumps(outMsg), (SERVER_IP, PORT))
        numPacketSent = numPacketSent + 1
        packet_index = packet_index + 1

if unpackedData and isinstance(data[0], bytes) and data[1] != -1 :
    # play output audio
    timeIn = time.time()
    latency = timeIn - resTime[data[1]]
    latenyTotal = latency + latenyTotal
    avgLatency = latenyTotal / numPacketSent
    numPacketReceived = numPacketReceived + 1
    print('Packet received @ time ' + str(timeIn))
    print('Latency ' + str(latency))

#allows for some dropped packets
if packet_index >= 500 and (numPacketReceived + 2) == numPacketSent:
    print('----------LATENCY TEST RESULTS----------')
    print('Number packets sent ' + str(numPacketSent))
    print('Average latency ' + str(avgLatency))
    print('----------END LATENCY TEST----------')
    break
```

```
    except KeyboardInterrupt:
        stream.stop_stream()
        stream.close()
        p.terminate()

def testDroppedPacket(processedAudioQ):
    global numPacketSent
    global numPacketReceived
    global stream

    #Start sending and receiving
    print('----------START DROPPED PACKET TEST----------')

    numPacketSent = 0
    numPacketReceived = 0

    try:
        while True:
            #Getting new data
            data = [b'', -1, '']
            gotData = False
            try:
                dataByte, addr = s.recvfrom(CHUNK)
                gotData = True

            except BlockingIOError:
                gotData = False

            #decode data
            unpackedData = False
            if gotData:
                try:
                    data = msgpack.loads(dataByte)
                    unpackedData = True
                except:
                    unpackedData = False

            if len(processedAudioQ) > 0:
                (data, doa)  = processedAudioQ.pop()

                outMsg = [b'',0,'']
                outMsg[0] = data
                outMsg[1] = doa
                outMsg[2] = micID[0]

                if not stopSending:
                    s.sendto(msgpack.dumps(outMsg), (SERVER_IP, PORT))
                    numPacketSent = numPacketSent + 1
```

```
            if unpackedData and isinstance(data[0], bytes) and data[1] != -1 :
                # play output audio
                numPacketReceived = numPacketReceived + 1

            if finishPacketTest:
                break

    except KeyboardInterrupt:
        stream.stop_stream()
        stream.close()
        p.terminate()


'''
Program args
ClientTest.py
-b | --basic : run test basic connection
-l | --latency :  run the latency test
-d | --droppedPacket : run the dropped packet test
'''
def main(argv):

    isLatency = False
    isDroppedPacket = False
    workers = 0
    outputfile = ''
    try:
        opts, args = getopt.getopt(argv,"hbld",["basic","latency","droppedPacket"])
    except getopt.GetoptError:
        print ("ClientTest.py\n" +
                            "-b | --basic : run test basic connection\n" +
                            "-l | --latency :  run the latency test\n" +
                            "-d | --droppedPacket : run the dropped packet test\n")
        sys.exit(2)
    for opt, arg in opts:
        if opt == '-h':
            print ("ClientTest.py\n" +
                                "-b | --basic : run test basic connection\n" +
                                "-l | --latency :  run the latency test\n" +
                                "-d | --droppedPacket : run the dropped packet test\n")
            sys.exit()
        # elif opt in ("-w", "--workers"):
        #     workers = arg
        elif opt in ("-l", "--latency"):
            isLatency = True
        elif opt in ("-d", "--droppedPacket"):
            isDroppedPacket = True
        elif opt in ("-b", "--basic"):
```

```
            testBasicConnection()
            sys.exit()

    # if workers >= 0 and workers <= 5:
    #     password = getpass("Please enter your password: ")
    #     # sudo requires the flag '-S' in order to take input from stdin
    #     proc = Popen(("sudo -S stress --io " + str(workers)).split(), stdin=PIPE, stdout=PIPE, stderr=PIPE)
    #     # Popen only accepts byte-arrays so you must encode the string
    #     proc.communicate(password.encode())

    startClient(deque(), isDroppedPacket, isLatency)

if __name__ == "__main__":
   main(sys.argv[1:])
```