Engr 523 - Project 1 - UDP Server

Due: 11:59pm, Janurary 18, 2019 Version: 2019.6

Goal

This project will help you review Internet-enabled communication through a UDP socket. We will be using python to transfer a WAV (uncompressed audio) file over a UDP socket. We will then serve the wave file using a HTTP (TCP) interface.

# Setup

We will be using a random AWS cloud machine to act as our "cloud server" for our devices. For convenience, this machine can be accessed at [iot.lukefahr.org](http://iot.lukefahr.org).

**You will need an account on this machine before you can begin working. Please send an email to Subhojit <susom@iu.edu> with the subject "IOT KEY" with an RSA public key attached named 'username.pub' (where username is replaced with your** [**username@iu.edu**](mailto:username@iu.edu) **name).**

Once you have an account, you are allowed to log in and run TCP and UDP servers that open any port above 10,000. We recommend your program(s) open two ports: one for accepting incoming audio data from the photon, and another for making that audio data available via HTML website. To avoid port conflicts, please post your port selection to Piazza.

Please do not use this server or ports for anything other than class related projects. We told the department we wouldn’t do anything stupid with this server, so please help us keep that commitment so we can do this again next year.

# Background

## WAV Files

WAV (or Waveform Audio File) is a audio file format. In simple terms, a wave file is a series of numbers that represents a sound. We encourage you to read more about WAV files on Wikipedia. For now, it is sufficient for you to know how to open, close, and read the file.

For these examples, we will be using the **‘elephant.wav**’ file found on the website under the ‘**Downloads**’ quick link.

### Loading WAV files

In Python3, a wave file can be loaded with this code:

import librosa

ys, sr = librosa.core.load(‘elephant.wav’, mono=True, sr=16000)

This will yield a series of numbers, ys, and a sampling rate, sr. Later the sampling rate will be important, but for now, we can mostly ignore it.

### Inspecting WAV files

If you are curious about what values make up a wav file, you can simply print out the ‘ys’ and find out:

print (ys)

You should get something that looks like this:

array([-0.19870029, -0.23709884, -0.2184751 , ..., 0.04333525,

0.11380226, 0.14801326], dtype=float32)

This tells us that the python array is of 32-bit floats (dtype=float32), and it starts listing off the first few (-0.19870029, -0.23709884, -0.2184751) and last few (0.04333525, 0.11380226, 0.14801326) values.

### Saving WAV Files

A wave file can be saved with this code:

import librosa

librosa.output.write('elephant2.wav', ys, sr )

This saves those numbers (ys) back into a WAV file, and names it ‘elephant2.wav’.

Note: Newer versions of librosa use the ‘write\_wav’ function, i.e.:

librosa.output.write\_wav('elephant2.wav', ys, sr )

### Converting to/from Bytes

Python is a bit wired in that it by default it stores all data in ‘lists’. However, for network transmissions, we need the data in ‘bytes’. Luckily, we can convert between the two:

Convert a series of floats to/from bytes:

import numpy

import struct

a = numpy.array([-0.19870029, -0.23709884, -0.2184751], dtype=numpy.float32)

b = struct.pack('<'+'f'\*len(a), \*a)

a2 = struct.unpack('<'+'f'\*int(len(b)/4), b)

The array a is the first three values from the above WAV file. Here, I’m converting it into an array of bytes, b, using the struct.pack command. Note that len(b) = 4 \* len(a). b is a 12 element array of 8-bit bytes, where as a is a 3 element array of 32-bit bit words. b is now ready for network transmissions. To convert it back into an array, we use struct.unpack. Recall that a 32-bit float requires 4 bytes. Therefore, len(b/4) is the number of 32-bit (4-byte) floats. Unfortunately, python doesn’t know that this division will result in a whole number, so we have to cast it as an int, ie int(len(b)/4).

## UDP Interfaces

To transmit the WAV data, we’re going to use a UDP socket.

### UDP Client

To start a UDP client, the following python3 code is helpful. It will create a connect to a remote machine on port 10000 and send a string.

import socket

port = 10000

server\_address = ('iot.lukefahr.org', port)

s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

data = b'This is the message.'

s.sendto(data, server\_address)

s.close()

### UDP Server

To start a UDP server, the following python3 code is helpful. It will create a socket and start listening on port 10000. It waits until someone connects, receives the string, and prints it out.

import socket

port = 10000

s = socket.socket( socket.AF\_INET, socket.SOCK\_DGRAM, socket.IPPROTO\_UDP)

s.bind( ('0.0.0.0', port))

raw, addr = s.recvfrom(1024)

print (raw)

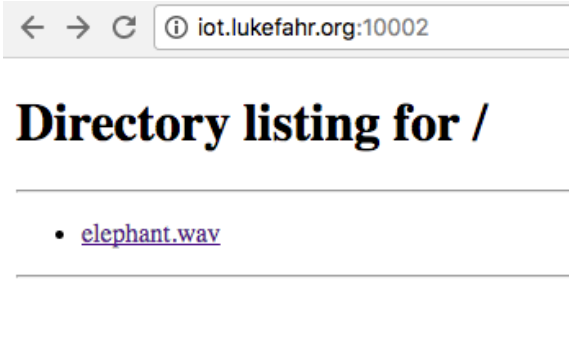
s.close()

## HTTP Web Interfaces

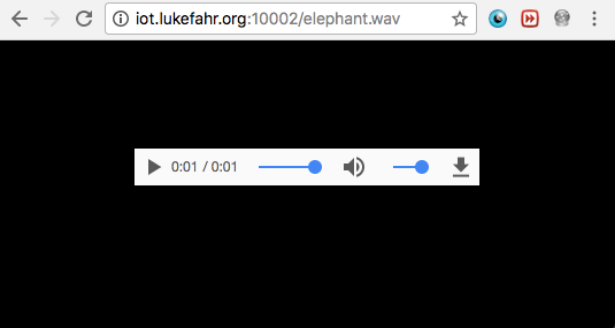
Python provides a built-in simple web server interface. While your cloud application will probably need to be more complex, simple forms of this server can be accessed directly from the command line:

python3 -m http.server 10002

That will serve a listing of everything in the current directory. If your directory contains only ‘elephant.wav’, it should look similar to this:



You can use this interface to find and play your audio files over a web browser:



# Assignment Description

Your assignment is to create three python files, client.py and server.py, and http.py.

## server.py udp\_port

This file should receive an incoming WAV file over a UDP socket, and save it to a WAV file. For now, it only needs to receive 1 incoming transmission of 16000 values (64000 bytes). Once those values have been received, it should store the received data to a wav file on the local folder.

It should accept one command line argument, udp\_port. This is the port that the server will attempt to listen on to receive incoming data.

## client.py udp\_port wav\_file

This file should load a WAV file, connect to the UDP server and transmit the WAV file.

It should accept two command line argument, udp\_port and wav\_file. udp\_port is the port on the server (iot.lukefahr.org) that the server is listening on. wav\_file is the name of the file to be transmitted.

## http.py tcp\_port

This file should create a HTTP interface to serve the contents of the local folder to a website. This will allow you to use a web browser to play the contents of the received wav file.

It should also accept one command line arguments, tcp\_port. tcp\_port is the port on which to listen for incoming http requests from web browsers.

# Grading

The grading for this assignment will be based on 4 milestones:

* Correctly transmitting WAV data via UDP (30%)
* Correctly receiving WAV data via UDP (30%)
* Correctly serving HTTP data via TCP (20%)
* End-to-end system with correctly formatted audio data playable via website (20%)

The points for these milestones are all dependent on correctly following the submission instructions.

# Submission

All code should be developed in a private IU Github repository where *the users susom and lukefahr are added as collaborators*. You must convey which version of your code is your “submission” by submitting the link to the commit on IU Github. This link will look similar to

<https://github.iu.edu/SOIC-Digital-Systems/Spring-2017/commit/269670d11d10decb79905ff3cb4ba456a9c928c0>

and when followed should lead to the commit summary page.

The submitted commit must contain:

* Your code, obviously. ;)
* A Readme.md. This will specify a set of instructions for how to build and/or run the project, and we suggest that the process be automated via makefiles or similar tool. The readme must also contain the name and email of your partner.
* A video of you demonstrating your working code. This should be included directly into the Github repo. If multiple videos exist, the Readme.md should contain instructions on which video to view for grading.

The video should contain the following steps. Here blue indicates shell (or bash) commands.

1. SSH into iot.lukefahr.org (optional for video)
2. Execute ls in a client folder containing only client.py and elephant.wav
3. Execute ls in a server folder containing only server.py and http.py. This folder should NOT contain a WAV file
4. Execute python3 server.py (plus arguments) in the server folder
5. Execute ls on the server folder, showing no WAV files
6. Execute python3 client.py (plus argument) in the client folder
7. Execute ls in the server folder, this time showing the received WAV file
8. Execute python3 http.py (plus arguments)
9. Use a web browser (Firefox, Chrome, Safari, Edge, etc) to display the contents of your server folder over HTTP
10. Play the received WAV file either directly in the browser (Chrome) or by downloading the WAV file and playing with a media play (Firefox, Safari, Edge, etc).