# Lab 4

More sockets….

***Goal:*** The goal of this lab is to give you more practice using sockets, particularly using headers and dealing with non-text data.

In this lab, you will start with two files: server.py and client.py. ***You should only change client.py. Do not make changes to server.py.*** These files implement a simple protocol for downloading music. This protocol has two commands and two responses:

list – Gets a list of available songs from the server.

Command: list

Payload: none

list reply – This is the type of the reply message from the server, which includes a list of available songs

Command: list reply

Payload: serialized list of available songs

get – Requests a specific song from the server

Command: get

Payload: song name

get reply – Returns the requested song

Command: get reply

Payload: song data (binary)

Of these 4 messages, three are already implemented: list, list reply, and get reply. Our protocol includes the length of the message set as a 4 byte header, which greatly simplifies dealing with the socket.

**TODO:** Your job in this lab is to implement the get request, and to save the song data you get back in a file.

To send the request:

1. Create a new command class instance
2. Set the command to “get”
3. Set the payload to the name of the song you want to retrieve
4. Use pickle to serialize the command class. This is your message (see my example code)
5. Get the length of the message and pack it into a network neutral format with struct.pack() (see the example code)
6. Send the message length over the socket (it’s 4 bytes)
7. Send the message over the socket

To process the reply:

1. Read the first 4 bytes of the reply from the socket.
2. Unpack these bytes into an int using struct.unpack(). This is the reply message length
3. Read the message in, using the reply message length from 9.
4. Unserialize it to a Command class instance using pickle (see my example code)
5. Save the payload in a **binary** file

Submit client.py **ONLY.**