Assignment 1:

Due Date: Wednesday February 2nd, 11:59 p.m.

Objectives:

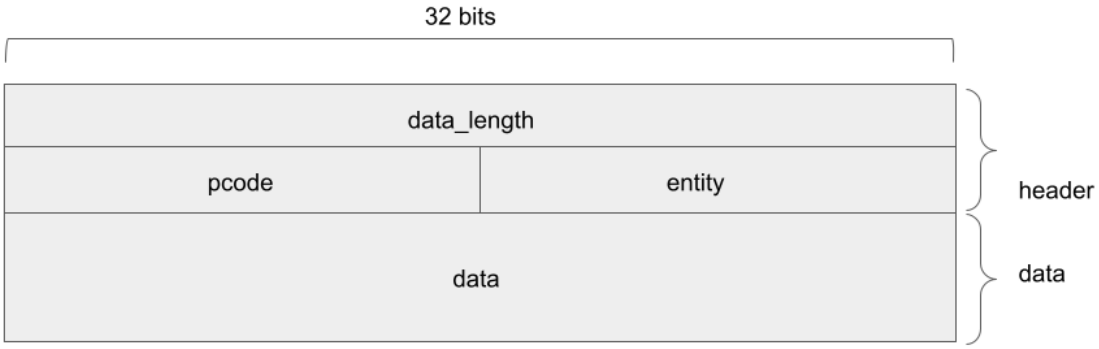
- Socket programming in python using stream sockets (TCP) and datagram sockets (UDP)
- Implementing a simple protocol that resembles many protocols you will be learning throughout this course
- Get to know many terminologies used in computer networks

General Instruction

- You are allowed to work in groups of at most two students. It is okay if you want to work on your own.
- **Start early.** This assignment has lots of work.
- Start with the `UDPServer.py` and `UDPClient.py` files provided in the course. You will be implementing a protocol consisting of four phases. The first two phases use UDP protocol to communicate and the last two phases use TCP protocol.
- You must use python version 3.6 for this assignment.

Description

In this assignment, you will develop a client application and a server application. The client and server communicate using a specific protocol named **FourPhase**. The client and server communicate according to the protocol specifications. The protocol consists of four phases. In the first two phases, the client and the server use UDP protocol to communicate and in the last two phases they use TCP protocol for communication. The client and server communicate by sending packets. The general format of a packet is shown in the following figure:



As you see a packet consists of a **header** and a **data** part. The header is located in the initial part of the packet and is 8 bytes. The first four bytes of the header, **data_length**, is the length of the packet (Note: this length does not include header length, it only specifies the length of the data).
The **pcode** is the code generated and sent by the server in the previous phase of the protocol. For the first phase the pcode is zero. The client must extract the code sent by the server in each phase and use that code in the next phase. The server verifies that the client is following the protocol and it closes the connection in case of any deviation from the protocol. In the first phase of the protocol pcode is zero.
The next two bytes are the **entity**. The entity specifies whether it is the client or the server that is sending the packet. For the client always use 1 and for the server always use 2.
The **data** part could be of any size, what is important is that the packet size should be divisible by 4. In other words, whenever making a packet, the data part must be padded until the packet length is divisible by 4.

Server

The server application starts running on UDP port 12000 and listens for incoming packets. The server expects to receive a packt in the format specified above. The specific value that goes into each field will be specified for each phase of the protocol.

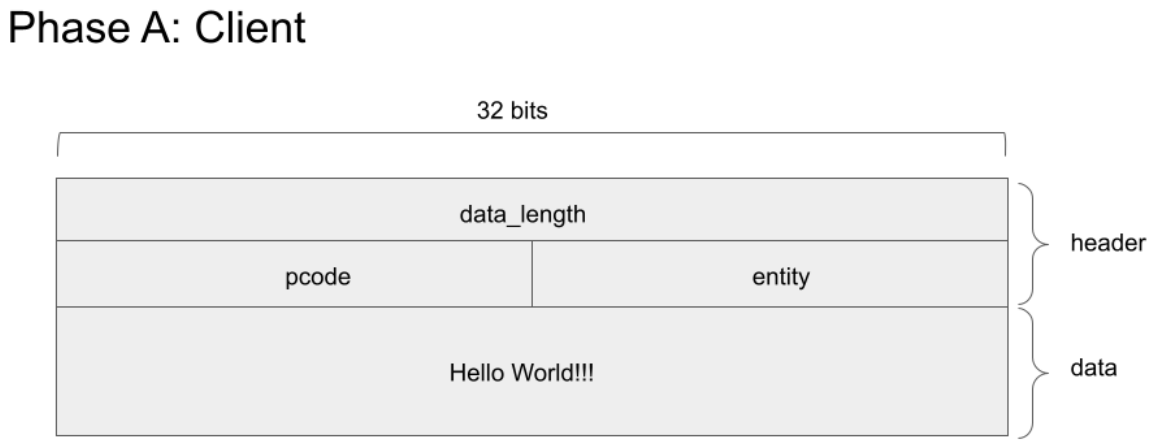
The server checks to make sure that the packet has a header and follows the specified format and its size is divisible by four. Any number should be an unsigned short or an unsigned integer (will be specified). The server will expect the characters to be in network order (big-endian).
The server will close the connection to the client in the following cases:

- The server does not receive any packet from the client for 3 seconds
- A packet is received with a size not divisible by 4
- Unexpected header, data, data_length is received
- The length of packet, length of data, data, data_length, pacode or entity is not what is expected in the specific phase of the protocol.
- Unsigned short number and unsigned integer numbers are not used
- The characters are not aligned in network order (big-endian)

In the following, the packet details for each phase of the protocol is described:

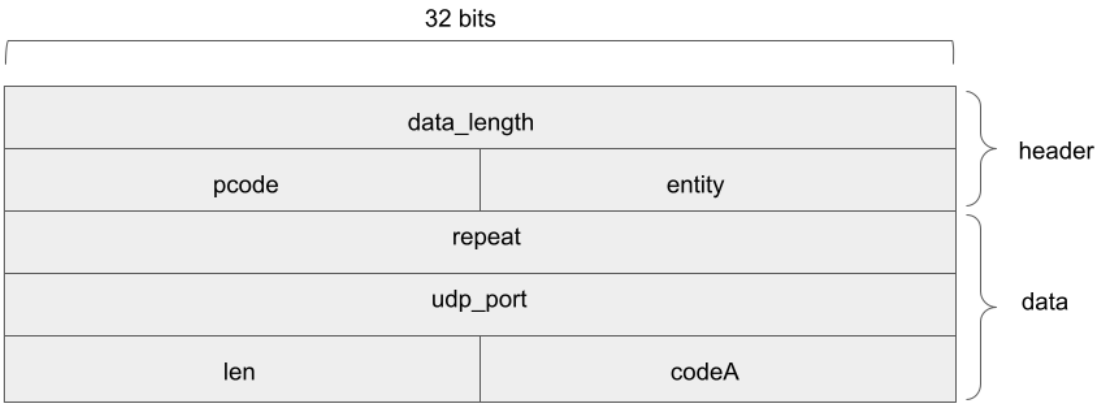
Phase A

Client: The client sends a single packet to the server using UDP protocol. The format of the packet is as specified in the following figure. The data part contains the string Hello World!!!. As you see in the following picture, it does not have quotation marks and it has three exclamation marks at the end. The data_length field should contain the size of string (in bytes). The **pcode** is zero and the **entity** is Client. Make sure the length of the packet is divisible by 4..



Server: The server receives the packet sent by the client, verifies the packet, and responds with a packet that has the following format:

Phase A: Server



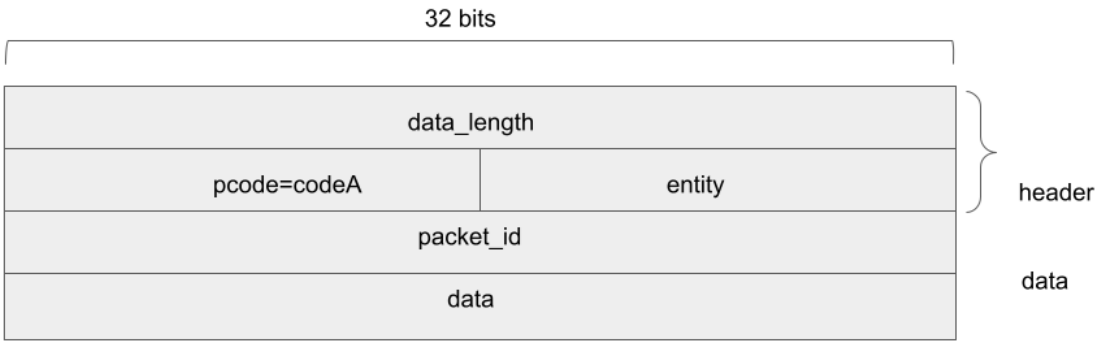
repeat is a random integer (4 bytes) between 5 and 20
udp_port is a random integer (4 bytes) between 20000 and 30000
len is a short random number between 50 and 100
codeA is a short random number between 100 and 400

The server then starts listening on the **udp_port** just sent to the client.

Phase B:

Client: The client **reliably** sends **repeat** UDP packets to the server on **udp_port**. Each UDP packet has the following format:

Phase B: Client



pcode is set to **codeA** received in the previous phase. The entity is Client. The **packet_id** is an integer identifying the packet. The first packet sent by the client has packet_id set to zero and the last packet has it set to repeat-1. **The data part consists of 0's. The data part should be an integer (packet_id), which is 4 bytes, as well as bytearray(len) where len is sent from the server in the previous phase. Note that all the sizes are in bytes. For example, if len=70, then data will be 74 bytes and to make it divisible by 4, we add two bytes using bytearray(2) and it will be 76 bytes. Each value in the data is 0.** The len sent from the server specifies the number of bytes of data.The data part must be padded to make sure the packet length is divisible by four. The **data_length** field should contains the length of data (len + if any padding needed) as well as length of packet_id (4 bytes)

Server:

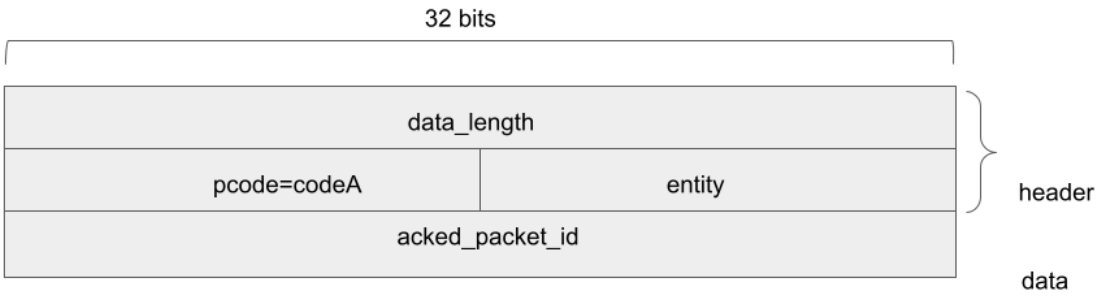
Step B-1

Upon receiving a packet, the server verifies the packet. In addition the server checks:

- The packet has correct length (remember **len** is adjusted to make sure packet length is divisible by 4 and also the 4 bytes for packet_id should be considered)
- The server sends an ack for each packet before receiving the next
- The first four bytes of the data part of the packet is the integer number identifying the packet.
- The packets arrive in order

For each received packet, the server randomly decides to send an acknowledgement packet:

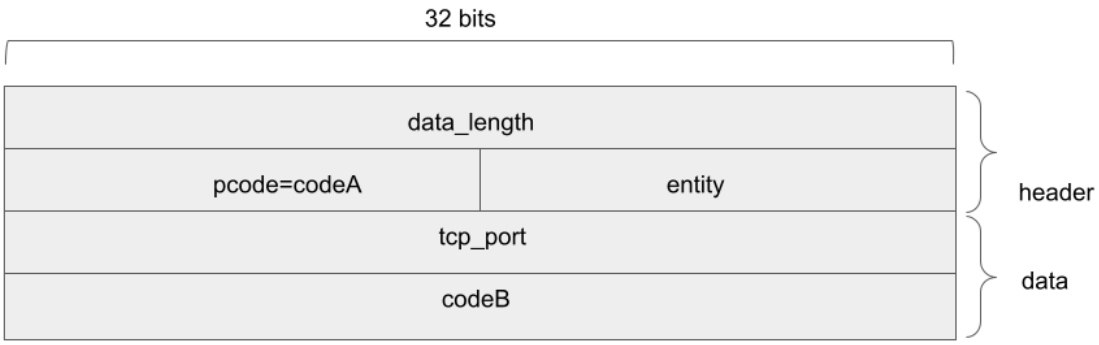
Phase B-1: Server



To have reliable communication, the client must receive ack packets from the server for all **repeat** packets sent to the server. If the server does not send acknowledgement for a packet, the client resends those packers and the server should receive the same packet (same packet_id) again. This process will continue until the server acknowledges the packet. The client should use a retransmission interval of at least .5 seconds.

Step B-2. Once the server receives all the **repeat** packets, it sends a packet to the client having the following format:

Phase B-2: Server



The **tcp_port** and **codeB** are random numbers generated by the server. Then the server starts listening on this **tcp_port** waiting for connection from the client.

tcp_port is a random integer (4 bytes) between 20000 and 30000
codeB is a short random number between 100 and 400

Phase C:

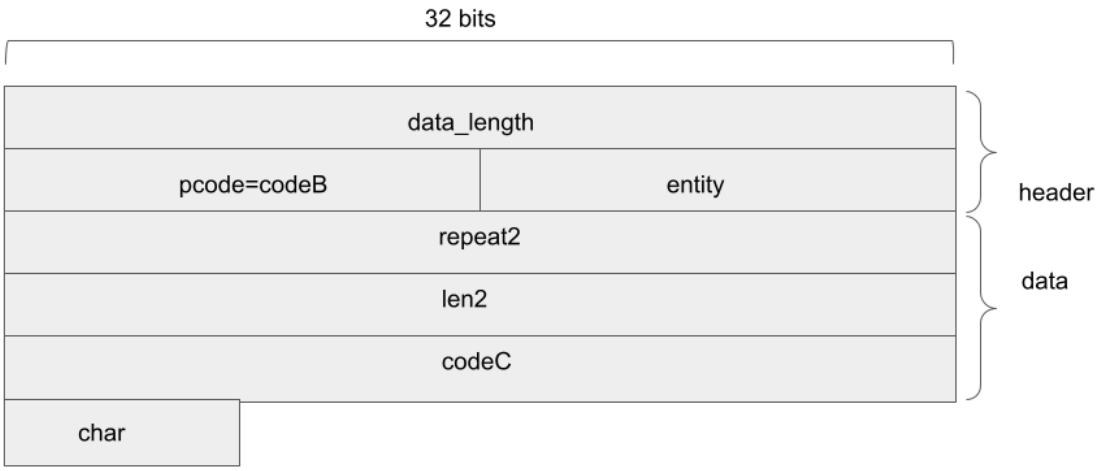
client:

The client connects to the server on **tcp_port** and opens a TCP connection to the server. The **tcp_port** is received from the server in the previous phase. **Before connecting, make sure to wait a bit (by calling time.sleep) because the server may have not found the time to start listening.**

Server:

The server sends a packet to the client which has the following format:

Phase C: Server

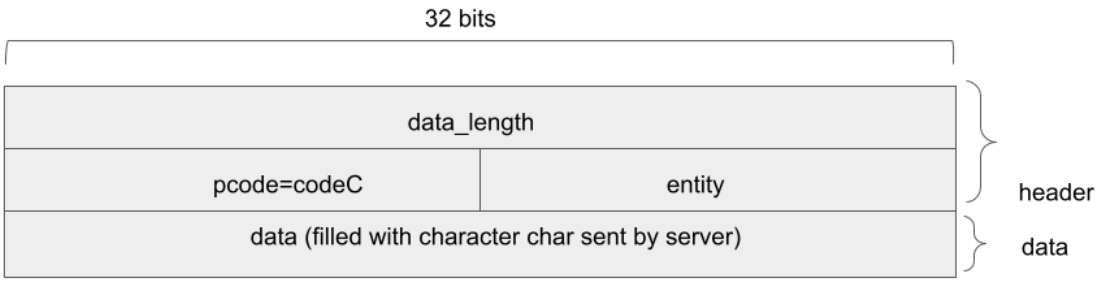


repeat2 is a random integer (4 bytes) between 5 and 20
len2 is a short random number between 50 and 100
codeC is a short random number between 100 and 400
Char: any of the 26 characters in English alphabet: A to Z

Phase D:

The client sends **repeat2** packets to the server.

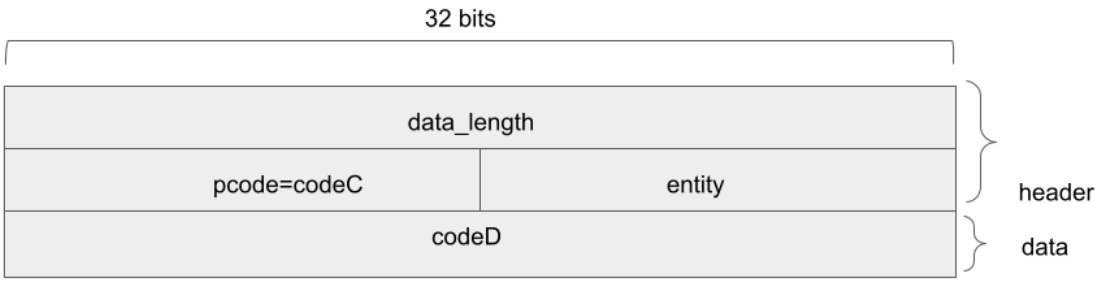
Phase D: Client



The length of data in each packet sent by the client to the server is len2 (the length is adjusted to make sure packet length is divisible by 4), and each byte is set to **char** sent by the server in the previous phase.

Once the server receives all the packets, it responds by sending a packet to the client that has the following format. The **codeD** is a random integer between 100 and 400.

Phase D: Server



Submission Instructions

- Create one file named **`server.py`**
- Create one file name **`client.py`**
- Put the above two files in a folder named **a1_x_y** where x is your username and y is your assignment partner username (e.g., if I were to develop this assignment with my partner having username barn407, then I would name my project as **a1_mrudafshani_barn407**)
- Zip the above folder (e.g., a1_mrudafshani_barn407.zip) and submit the zip file to the appropriate dropbox folder.
- **Do NOT submit an eclipse project!**
- **NOTE: Make sure to use python version 3.6 or higher.**
- **DO NOT USE python 2.7.**
- Make sure to put your names (as well as your partner's name) at the top of both the server.py and client.py.
- Make sure to put your names on top of both python files
- **Make sure to have one submission per group**

Tips

- When using the TCP socket and testing your application (server program), you may run the code several times and you may encounter the following error:

```
OSError: [Errno 98] Address already in use
```

To prevent this you can set a socket flag named SO_REUSEADDR. Look here for more information.

- `struct.pack()` and `struct.unpack()` are needed to create packets in the specified format. Look here for more information. You need to use **I** for formatting unsigned integers and **H** for formatting unsigned short, and **!** to make sure the characters are in big-endian order
- You can use **settimeout** function of the socket library to define a timeout for a socket

Grading

To test your code, I will test your client against my server and your server against my client.
[5 points] Following Submission Instruction
Server:
[10 points] Close connection when client sends faulty packets or timeout
[10 points] Phase A
[10 points] Phase B
[10 points] Phase C
[15 points] Phase D

BONUS point [20 points] Multi-process server: Server handling multiple clients

Client:
[10 points] Phase A
[10 points] Phase B
[10 points] Phase C
[10 points] Phase D

A large portion of your mark will be deducted if your code does not run.
Try to develop and run your code step by step. Follow the hints and tips provided