Internet chatting in the TCP/IP network

COURSEWORK 14ELC004-COMPUTER NETWORKS

Designed by: Dr Alex Gong and Mr Zhao Tian

2014/15

1. Introduction

The coursework develops an internet chatting tool box using Python programming. The Internet chatting becomes ever popular in social media, often providing a more efficient way of connecting people together than traditional methods such as the email. There exist several kinds of network chatting: some are through severs (e.g. Google Chat), and some are totally ad hoc (e.g. FireChat).

In this coursework, we develop a simple Internet chatting tool box based on TCP/IP protocols.

2. Install the Python in your computer

This coursework suggests use Python. Python is a free and widely used programming language. You may also use other programming languages such as C, C++ etc.

The official Python website is https://www.python.org. Download and install Python 2.7. You shall see "Python 2.7" in your computer. Open the GUI tool "IDLE (Python GUI)", and then you can edit and run your Python codes.

Students are required to self-learn the Python. There are many learning resources either on-line or in textbooks. One recommended website to study Python is

http://www.tutorialspoint.com/python/python_networking.htm

where you can also find the Python server and client codes. Strongly suggest students go through the above website before start the project.

3. System model

The system model of the Internet chatting tool box in this coursework is shown in Fig. 1.

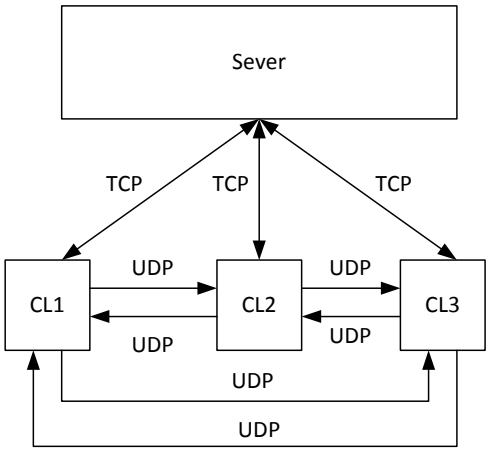

Fig. 1. System model of the chatting tool box

In this system, there are one sever and a number of clients ('CL'). The chatting tool box operates in the following way:

- The server is always on-line. The server maintains a *client address table* containing the UDP socket addresses of all on-line clients.
    - socket = <IP address : port number>
    - The socket of the server is known to all clients.

- A client can either be on-line or off-line. When a client goes on-line,
    - the client informs the server of its socket address used for UDP connection with other clients.
    - the server adds the UDP socket of this client in its *client address table*.
    - The server sends *the client address table* back to the client.
        - the client thus also maintain an *client address table* containing the UDP sockets of other on-line clients.
        - the client frequently access the server to update its *client address table.*
    - the communications between the server and clients are based on the TCP, because the exchanged socket addresses are 'important' information.

- If, for example, CL1 want to contact CL2
    - From its *client address table*, CL1 obtains the UDP socket address of CL2.
    - CL1 sends a request to CL2, asking for the user name of the CL2.
    - CL2 sends back an acknowledge information including user name etc.
    - CL1 associates the user name and UDP socket address of the CL2.
    - Communications between clients are based on UDP for best flexibility.
    - Start chatting.

- If a client goes off-line, it informs the server and the sever removes the UDP address of this client from its *client address table*.

4. The server program

The server program is given to students, but students are free to modify the codes. Running "server_gui.py" popes up the server dialogue window as is shown in Fig. 2.

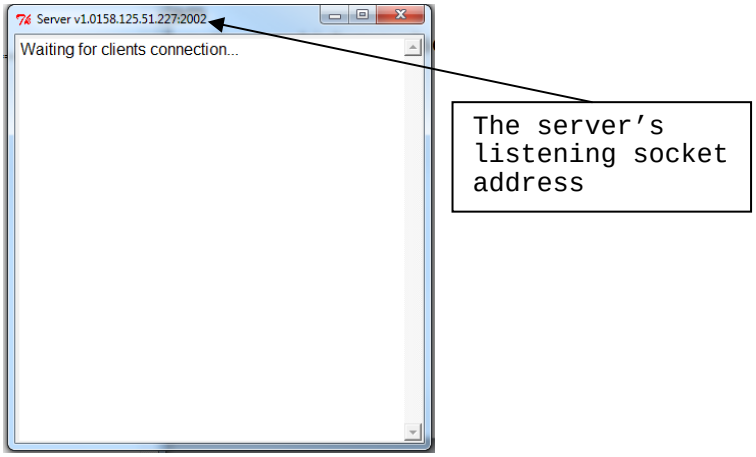

Fig. 2. Server dialogue window

The server's listening socket address is shown on the window's top bar, where the IP address is obtained as that of the computer running the server program:

```
HOST = socket.gethostbyname(socket.gethostname())
```

and the port number is manually fixed at "2002" (You can of course choose another port number) as:

```
PORT = 2002
```

The listening socket of the server is known to all clients. The listening socket is set for the TCP connection as:

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

where "AF_INET" indicates that it is for IPV4, and "SOCK_ STREAM" indicates that the socket is used for TCP connection.

The socket 's' is always open and continuously listening to all clients:

```
conn, addr = s.accept()
```

This passively accepts TCP client connection, waiting until connection arrives, where

- 'addr' returns the TCP socket of the connecting client, where 'addr[0]' and 'addr[0]' are the corresponding IP address and port number respectively.

- 'conn' is an automatically generated TCP socket at the server associated with 'addr' for TCP data transmission, or a TCP connection between 'addr' in the client and 'conn' in the server is established. Note, the server's listening socket 's' is not used for data transmission.

The server dialogue window initially shows "Waiting for clients connection". This indicates that the server's listening socket is open and waiting to be connected by clients. When a client visits the server, the server takes the following actions:

(1) The server receives data as

```
info = conn.recv(1024)
```

where the maximum number for receiving is set as1024 bytes (you can change the setting). Note again it is the socket 'conn' that receives data. The format of the received message is

```
{type: IP address: UDP port number : user ID}
```

(2) Obtains the UDP socket address of the connecting client as

```
conn_addr = (addr[0],dpkg(info)[2])
```

where 'addr[0]' and 'dpkg(info)[2]' gives of the IP address and the UDP port number of the connecting client respectively. 'dpkg(info)' de-packs the received message into the format as

```
{Type, IP address, port number, user ID}
```

(3) If 'type=R', the message is for the client asking for connection/information. The server takes the following actions:

(3.1) Update 'server_log' to display the message from the connecting client on the dialogue window as

```
Update(server_Log, 'got request from :'+info[2:])
```

where is 'server_log' is a text window located on the dialogue window 'root'.

(3.2) Send the UDP address of other on-line clients to the connecting client as

```
conn.send('-'+pkg( 'R' , ADDR[0], ADDR[1]))
```

(3.3) Add the connecting client's UDP socket address in 'client_list' as

```
client_list[conn_addr] = client_ID
```

where 'client_list' is the "*clients address table*" defined as

```
client_list={(IP, Port):ID}
```

This results in the sending message with the format as

```
{- IP address A : port number A - IP address B: port number B …}
```

where '-' is used to separate addresses. 'pkg('R' , ADDR[0], ADDR[1])' packs the sending addresses into the format as

```
{R: IP address : port number }
```

(3.4) Output a file containing the UDP addresses as

```
update_log_file(info)
```

(4) If 'type=K', the message is for the client to inform the server that the client go off-line. The server then remove the UDP sockets from 'client_list' and display the information in the server dialogue window.

```
if conn_addr in client_list.keys():
        Update(server_Log, 'receive offline from:'+info[2:])
        del client_list[conn_addr]
```

5. The client program

The basic client codes are given as "client_gui.py". The students need to complete the client program.

Every client has 2 ports:

- *TCP socket*: for communication with the server
  - The IP address is obtained as that of the computer running the client program.
  - The port number is automatically generated: when the client sends TCP request to the server's listening socket, a port number is automatically generated.

- *UDP socket:* for communication with other clients.
  - The IP address is obtained as that of the computer running the client program.
  - The port number is user specified. Note: if you test your program in one computer, you must assign different ports for different clients.
  - This is also the UDP socket which is stored in the server's *client address table*.

The client program for every client is the same. The client program runs two 'threads' simultaneously, corresponding to communications with the server and other clients respectively.

5.1. Thread for server connection - `class Server_Alive(threading.Thread):`

There are 3 cases that a client visits the server

- Case 1: when a client goes on-line, it informs the server of its UDP socket and obtains the UDP socket addresses of other on-line clients.
- Case 2: a client frequently (e.g. every 30 seconds) visits the server for the latest UDP socket addresses of other on-line clients in '`client_list`'.
- Case 3: when a client goes off-line, it informs the server and the server removes the UDP socket addresses from '`client_list`'.

When the client program (client_gui.py) is running, it first pops up a "request dialogue window" as is shown in Fig. 3.

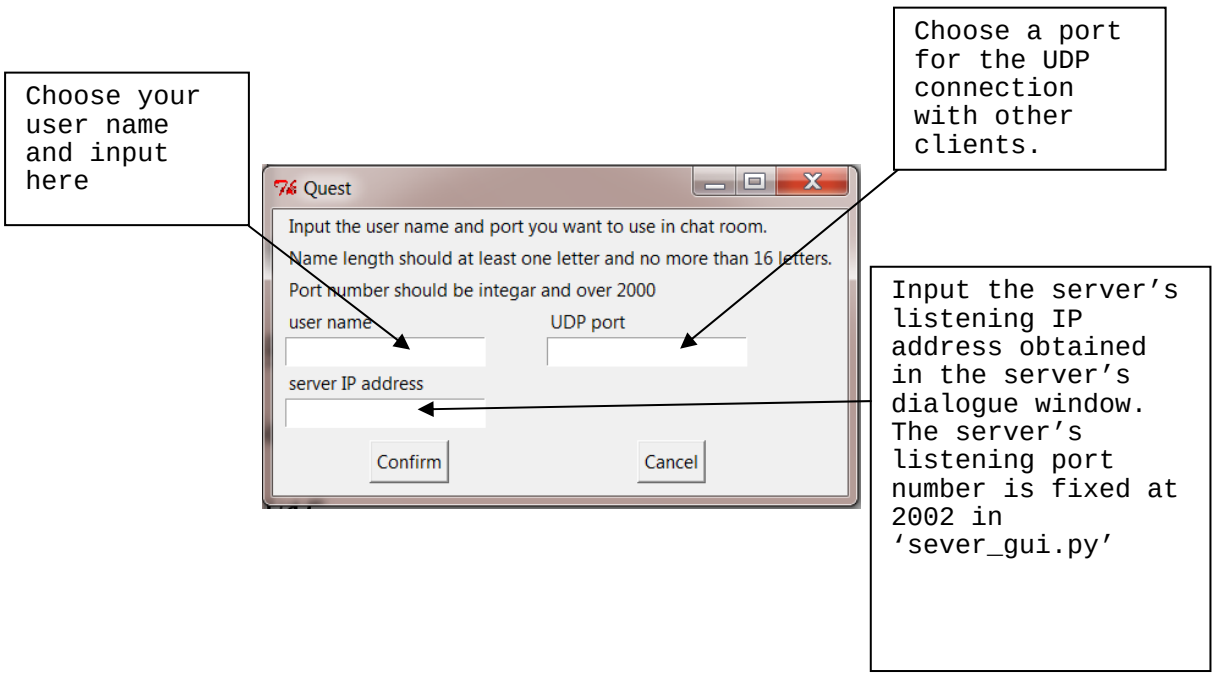

Fig. 3. Client's request dialogue window

In the client's request dialogue window, user needs to manually input 'user name', the port number used for the UDP connection with other clients, and the server IP address. The 'user name' and 'UDP port' are specified by the user, and the 'server IP address' is obtained in the server's dialogue window.

Press 'Confirm`, and then you go to the client's main chatting window as is shown in Fig. 4.

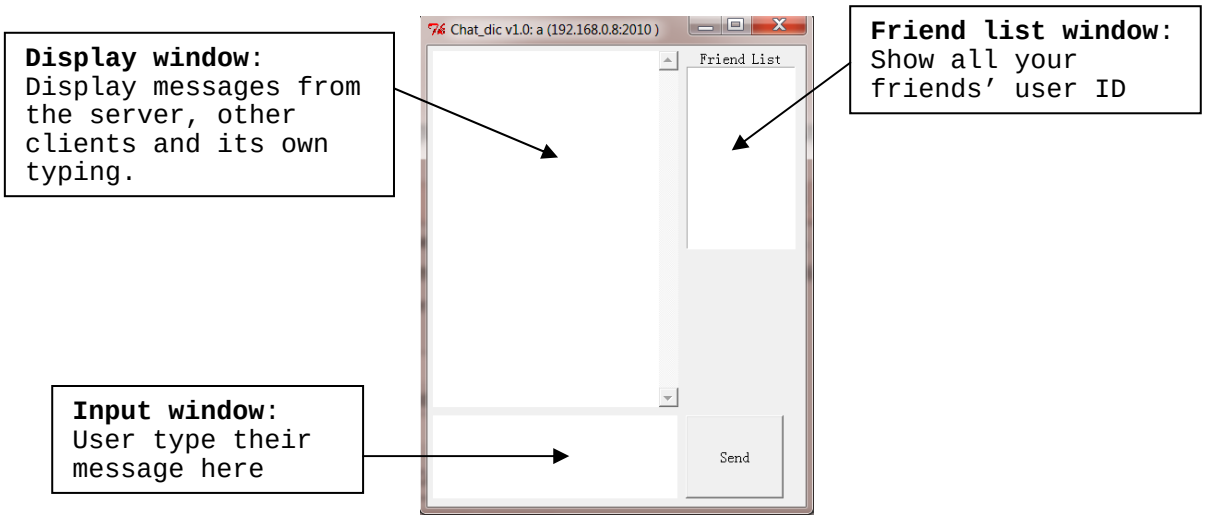

Fig. 4. Client's chatting window

An on-line client visits the server's listening socket as

```
connect_server(myID, myPort, serverIP, serverPort, self.Type)
```

In the function 'connect_server', the client opens TCP connection with the server as

```
s = socket(AF_INET, SOCK_STREAM)
s.connect((SERVER_ADD, SERVER_PORT))
```

The client then sends its UDP socket address to the server as

```
s.send(pkg(Type, myIP, myPort, myID))
```

Note that the TCP transmission socket at the client is automatically generated and we don't need to specify.

If 'Type == 'R'', the client is asking for the information of the UDP address table 'client_list' from the server. And then the client will later receives the message from the server as:

```
data = s.recv(1024)
```

When the client is on-line, it frequently requests the server for the latest 'client_list' as:

```
while self.flag:
    time.sleep(30)
    connect_server(myID, myPort, serverIP, serverPort, self.Type)
```

In this case, it requests every 30 seconds. You can set other times.

If 'Type == 'K'', the client goes off-line and informs the server to removes its UDP socket

```
s.close()
```

5.2. Thread for connection between clients - `class Receiving(threading.Thread)`

In this thread, the client opens a socket to receive the message from others. The basic structure of this class is defined, but the students need to complete the remaining.

6. Coursework requirement

In this coursework project, the students are given the full codes of the server and the basic codes of the clients. The students are required to complete the following task.

6.1. One-to-one chatting (compulsory).

An on-line client can send the text message to another on-line client based on the UDP (**the system should at least contain 3 clients**). It includes the following functions (or students can define other functions):

- The UDP socket is defined as

  ```
  socket(AF_INET, SOCK_DGRAM)
  ```

  where "AF_INET" indicates that it is for the IPV4, and "SOCK_DGRAM" indicates that the socket is used for UDP connection.

- Find the "user name" of the neighbouring clients. In the given server program, the server only provides the sockets of the on-line clients, but does not give the corresponding user names. When a client want communicate with another client, it can send "request" to other on-line clients. When a receiving client receives the request, it shall send back "acknowledge" to the sender with its user name and other information.

- Maintain a "friend list" of all on-line clients with user names.

- Define a structure of the data between clients. The UDP segment contains "head" and "data". The "head" is automatically maintained the Python program when a UDP socket is used. In this course, the students need to design the structure of the "data" part to deal with different type to transmission such as "user name request", "acknowledge", "chatting data" and secure transmission if it is necessary. For example:
  - If a receiving client receives a "request for user name", it then automatically sends back its user name (or other information such as address).
  - If the receiving data is standard "chatting data", it is then displayed on the screen.
  - For some important data, it is also possible to establish a link (similar to the TCP) with automatic acknowledge.

- A client can talk to any other clients in the "friend list".

6.2. Group chatting (optional).

This part is optional. A group of on-line clients can exchange text message among each other so that every client in the group can see the message from others.

6.3. Some important issues

- You need to decide the ports for the server and all clients. The ports available for safely use are: *2000 – 5000*.

- When you test the system, you can use only one computer, because you can assign different ports for server and clients. In such case, remember to use different file names for different clients.

    - Even if your computer is off-line, you may still test your program by using the localhost '127.0.0.1'.

- It is suggested you use own laptop (or home computer) to do the project, as Python may not be installed in the school computers.

7. Coursework marking scheme

The coursework mark allocation:

- System realization: 60%
- Report: 40%.

It is expected that the report will each be about 6-8 sides of A4 in size. However, the report must **not exceed** 15 sides of A4 under any circumstances.

While there is no fixed format for the report, the report shall contain (but not limit to) the following parts:

- Title
    - Include your name, the name of your group members, and the file names of the program in the first page.
- Introduction:
    - The purpose of this coursework.
    - Comments on the relation between the coursework project and the computer network background studied in the module.
    - Possibly a brief overview of some of the popular on-line chatting tools, including their functions, implementation, pros & cons etc.
    - Some background about Python.
- System design.
    - System model.
    - Functions in the system.
        - Beside the basic function of exchanging text message among on-line 'friends' are there any other extra functions in the system?
    - The implementation details.
        - For example, how the name and IP/port number of other on-line users are obtained? What kinds of packets are there in the system?
- System performance.
    - Some examples in running the system.
- Conclusion
    - Summary
    - How can the designed system be used in practice and further improved?
        - The similarities with some of the commercial on-line chatting systems.
        - What kinds of more function are necessary for the coursework project implemented in practice?
        - How can it be transferred to the application in mobile phones?

8. Submission

Students will work in groups, but **individual** reports will be required from each student.

Each group contains 4 members of students. Because there are 27 students in total, one group can have 3 members. **Please email me your choice of group by 27 Feb 2015.**

It is expected that the report will each be about 7-9 sides of A4 in size. However, the report must **not exceed** 15 sides of A4 under any circumstances.

You need to submit the following before the deadline:

- One individual report to the school office.
- One zipped file containing all programs (including the client and server codes). On-line submission in Learn.

The deadlines

Report and codes submission: **1st May 2015, 16:00pm**.