COMP312
UDP-Based Chatting System Report

Lecturer: Luo Xiapu

CHEN Yunkun 13103214D

ZHENG Hongyi 13104036D

LIU Xin 15007974X

Contribution Table	
Chen Yunkun	Design & Implementation of Server & Client
Zheng Hongyi	Design & Implementation of Server & Client
Liu Xin	Design & Implementation of User Interface

Table of Contents

1. **Design**

1.1 General design

This UDP-based chatting system is implemented using Java. It is designed into two parts, namely Client and Server. Server is deployed on Microsoft Azure.

Each client can connect to server. Then the messages sent by connected client will be forwarded to all client connected by server. Client can also initialize P2P chatting, provided the IP address of other client is known.

1.2 Design of Client

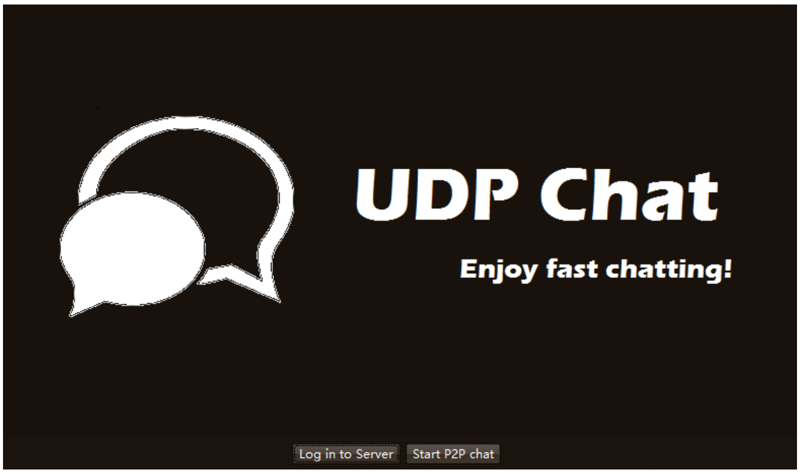
The programme of Client is designed into two parts, namely module and UI. Module handles the internal mechanism of the chatting system. While ui builds a graphic user interface.

1.2.1 Design of Client Module

Module consists three parts, namely ClientControl, ClientRece, and ClientSend. ClientSend sends message to server or another client (in P2P chatting); ClientRece receives messages; ClientControl is the interface between ui and the internal mechanism. It captures the user input from UI then deploys ClientSend to send the message; and collect the message received by ClientRece and shows them on UI.

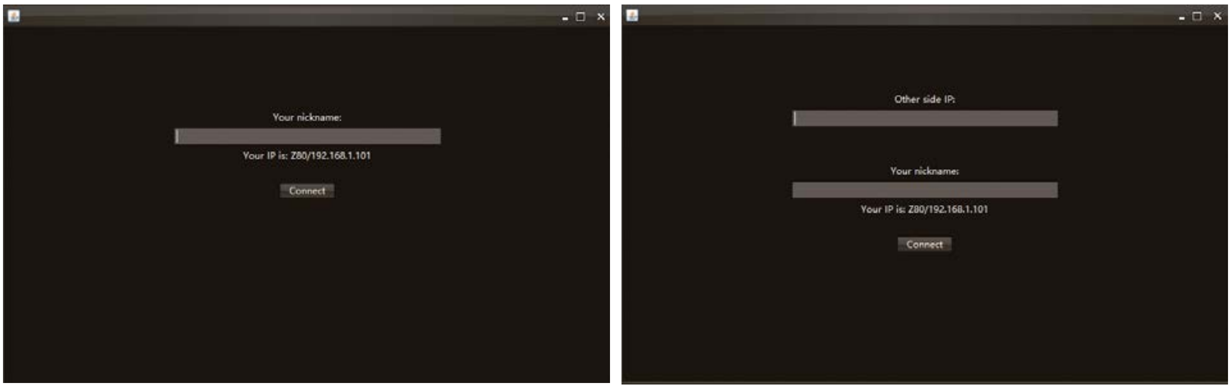
1.2.2 Design of Client UI

UI consists three pages.



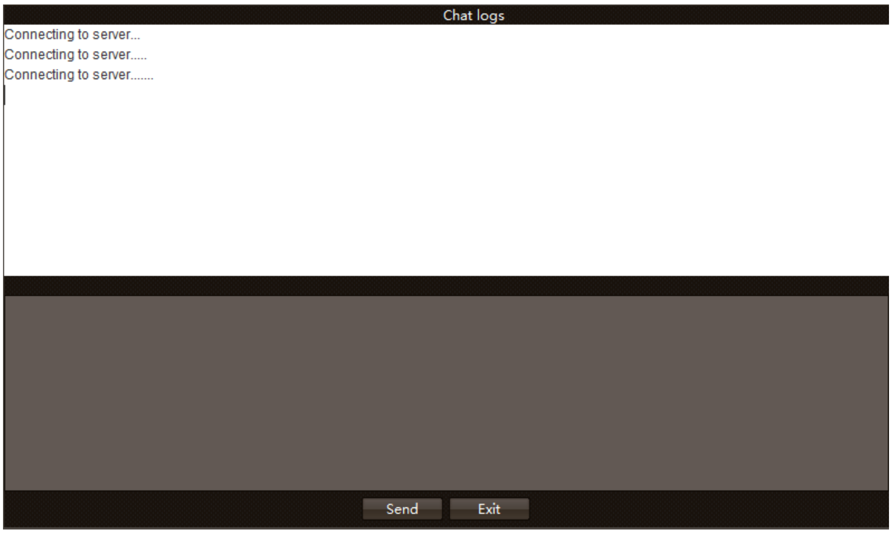
First one is login page.

User can choose to chat through server or chat with another client directly, provided IP address of other side is known.



Second page is initialization page.

If user chose chatting through server, he or she should input a nickname. If P2P chatting is chosen, the IP address other side should also be inputted. The IP address of user's current computer will also be shown on this page. Notice that the port numbers of client and server are automatically chosen and configured by programme, hence user has no need to concern about them.



```
Connecting to server...
Connecting to server.....
Connecting to server......
```

The third page is chatting page.

The upper area means to show the chat logs. The lower area is user input board. There are two button, one for sending message and one for return to login page.

1.3 Design of Server

Server does not have graphic user interface, as it is hidden from user. The internal mechanism is separated into three parts, namely ServerControl, ServerSend, and ServerRece.

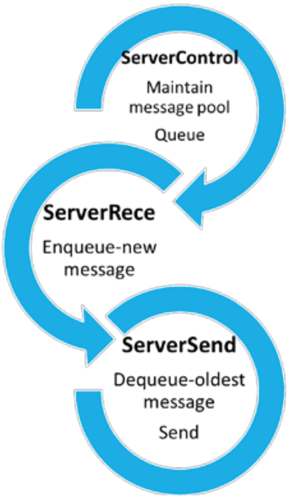
ServerRece keeps receiving message from clients then stores messages into message pool. If an INITIAL message was received, ServerRece will notice ServerControl to add a new client into client list. If a DISCONNECT message was received, ServerRece will notice ServerControl to delete this client from client list. ServerSend gets each message from message pool and forwards them to each connected client. While ServerControl coordinates the ServerSend and ServerRece and also maintains the message pool and client list.

2. Implementation (difficulties and solutions)

2.1 Message Forwarding Mechanism

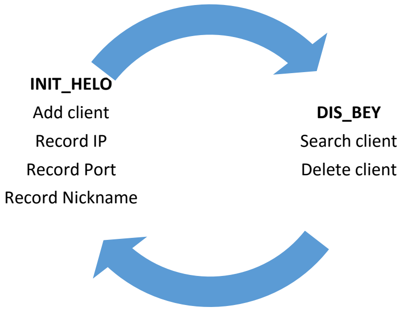
The first problem we encountered is how to handle the incoming and outgoing messages in sever-client mode.

After discussion, we designed a message pool, which is implemented by using a queue. The message pool is maintain by ServerControl and can be accessed by both ServerRece and ServerSend. The ServerRece en-queues a new message each when receiving. The ServerSend keeps de-queuing and sending message. As one of the character of queue is first in first out, the sequence of message will not get disordered. Below is the demonstration graph of message forwarding mechanism.



2.2 Multithreading

After the design of message forwarding mechanism, we found that sending and receiving message might happened at the same time, hence single thread cannot fulfilled the function required by chatting system. Hence we learned and adopted multithreading technology.



Both ServerSend and ServerRece run on their individual thread. ClientRece also runs on its own thread. For ClientSend, as it will only be triggered by user's mouse click, there is ran on default thread.

The methods in class MessageQueue are synchronized, hence the message can be handled appropriately.

2.3 Client registration mechanism

The last problem we encountered is the client registration mechanism, as client are connected and disconnected into the server dynamically.

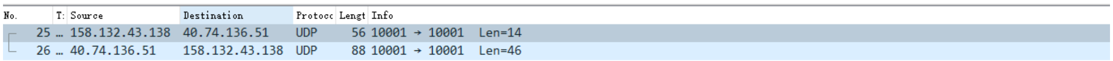
After discussion, we designed a simple protocol.

When client connects to server, a message began with “INIT_HELO” will be sent to server automatically. When the client disconnect from the server, a message began with “DIS_BEY” will be sent to server automatically. For the server, it will extract the each message when receiving. If it is started with “INIT_HELO”, it will record the IP address of client and add it into client list. If it is started with “DIS_BEY”, it will search the client and delete it from client list. The port numbers of server and client are chosen and configured by programme automatically, hence there is no need for more operation on port numbers.

3. Data

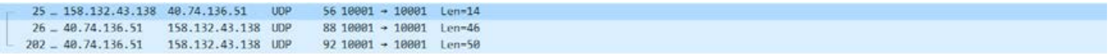
Followings are screenshot of packets captured by Wireshark.

Local client logs in:



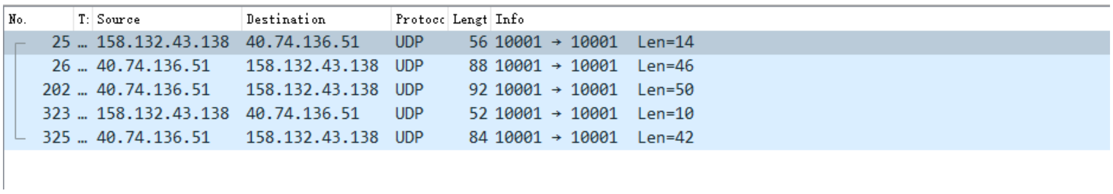
```
No.    T: Source          Destination     Protoco Lengt Info
   25 … 158.132.43.138  40.74.136.51    UDP      56 10001 → 10001  Len=14
   26 … 40.74.136.51    158.132.43.138  UDP      88 10001 → 10001  Len=46
```

Remote client logs in:



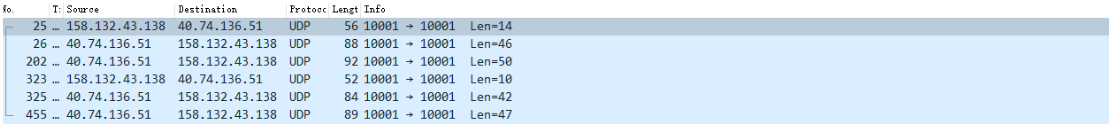
```
   25 … 158.132.43.138  40.74.136.51    UDP      56 10001 → 10001  Len=14
   26 … 40.74.136.51    158.132.43.138  UDP      88 10001 → 10001  Len=46
  202 … 40.74.136.51    158.132.43.138  UDP      92 10001 → 10001  Len=50
```

A message is sent by local client and forwarded back by server:



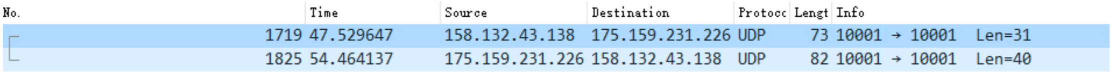
```
No.    T: Source          Destination     Protoco Lengt Info
   25 … 158.132.43.138  40.74.136.51    UDP      56 10001 → 10001  Len=14
   26 … 40.74.136.51    158.132.43.138  UDP      88 10001 → 10001  Len=46
  202 … 40.74.136.51    158.132.43.138  UDP      92 10001 → 10001  Len=50
  323 … 158.132.43.138  40.74.136.51    UDP      52 10001 → 10001  Len=10
  325 … 40.74.136.51    158.132.43.138  UDP      84 10001 → 10001  Len=42
```

A message sent by remote client is forwarded by server:



```
No.    T: Source          Destination     Protoco Lengt Info
   25 … 158.132.43.138  40.74.136.51    UDP      56 10001 → 10001  Len=14
   26 … 40.74.136.51    158.132.43.138  UDP      88 10001 → 10001  Len=46
  202 … 40.74.136.51    158.132.43.138  UDP      92 10001 → 10001  Len=50
  323 … 158.132.43.138  40.74.136.51    UDP      52 10001 → 10001  Len=10
  325 … 40.74.136.51    158.132.43.138  UDP      84 10001 → 10001  Len=42
  455 … 40.74.136.51    158.132.43.138  UDP      89 10001 → 10001  Len=47
```

Send & receive message in P2P chatting:



```
No.          Time       Source           Destination      Protoco Lengt Info
     1719 47.529647  158.132.43.138   175.159.231.226  UDP      73 10001 → 10001  Len=31
     1825 54.464137  175.159.231.226  158.132.43.138   UDP      82 10001 → 10001  Len=40
```