6650 Final Project Report - P2P Chat Room

Team members: Mengyun Wang, Wenhao Ge, Jing Wang

Summary:

This project is aimed to build a distributed chat room application. The structure of the chat application is designed based on the peer-to-peer networks, so it is less expensive to set connections between users. All the users are firstly registered with the directory server to be online and then connected with each other using the stored information using a directory service. Any online user can host a room and others can join a room by host name to initialize a group chat. The one to one private chat within the group is supported by adding a "@<name>" sign before message. A fault tolerance mechanism is also implemented by transferring the host position to other alive members in the same chat room when the host user fails in order to ensure stable communication for the rest of the group members. The application uses a GUI interface.

Introduction:

The chat room service is widely used in our life, and the application is meant to be a simple version of that service.

The application have implemented following functions:

- Register and deregister a user (online/offline).
- A user can see all the online users and rooms using directory service.
- A user can see the number of people in a room without a connection.
- An online user can host a chat room on a selected port.
- An online user can join any chat room by the host name.
- A user in a chat room can see all other members in the chat room.
- A user in a chat room can send messages to all the other users in that room.
- A user in a chat room can send private messages to one member of the room.
- A user can leave a chat room while online and rejoin other rooms.

Design approach:

Created Peer-to-Peer application that consists of following components:

1. Peer Node registration and deregistration:

All the users can do registration(online) or deregistration(offline) to be visible or invisible to other users. Users have to be firstly online to host a room or join a room. Uses in a room have to disconnect to leave the room to be offline. The online and offline functions is a simple demo for user login and logout.

2. Room Discovery via Directory Service:
The application utilizes a directory service to synchronize the information of all the online users and hosted rooms. Any update of a user is online, a user is offline, a user hosts a room and a user leaves a room would be sent to the directory server to be recorded. The users can request such information using the directory service to be online to join any of the alive chat rooms.

3. Peer to Peer networks
Peer-to-peer networking is a distributed application architecture that partitions tasks or workloads between peers. Our chat application is a p2p system, and peer nodes are equally privileged, equipotent participants in the application. Any peer node can be used as a server to host a room to be connected by other nodes or be as a client to peer nodes hosting a room. The peer node hosting a room is also a client of it's own server. A peer hosting a room can stop hosting and leave a room. A client peer node can leave the room, disconnecting the server and then host it's own room to be a server. All the nodes in the p2p system can change freely between the role being a server or a client.

4. Unicast/Multicast Group Communication
Unicast/Multicast distribution feature is added to the project. Users can send messages to a group of users. They can also send private messages to a specific user, and these private messages will not interfere with other users. The private message format is: "@<name>" + space + message

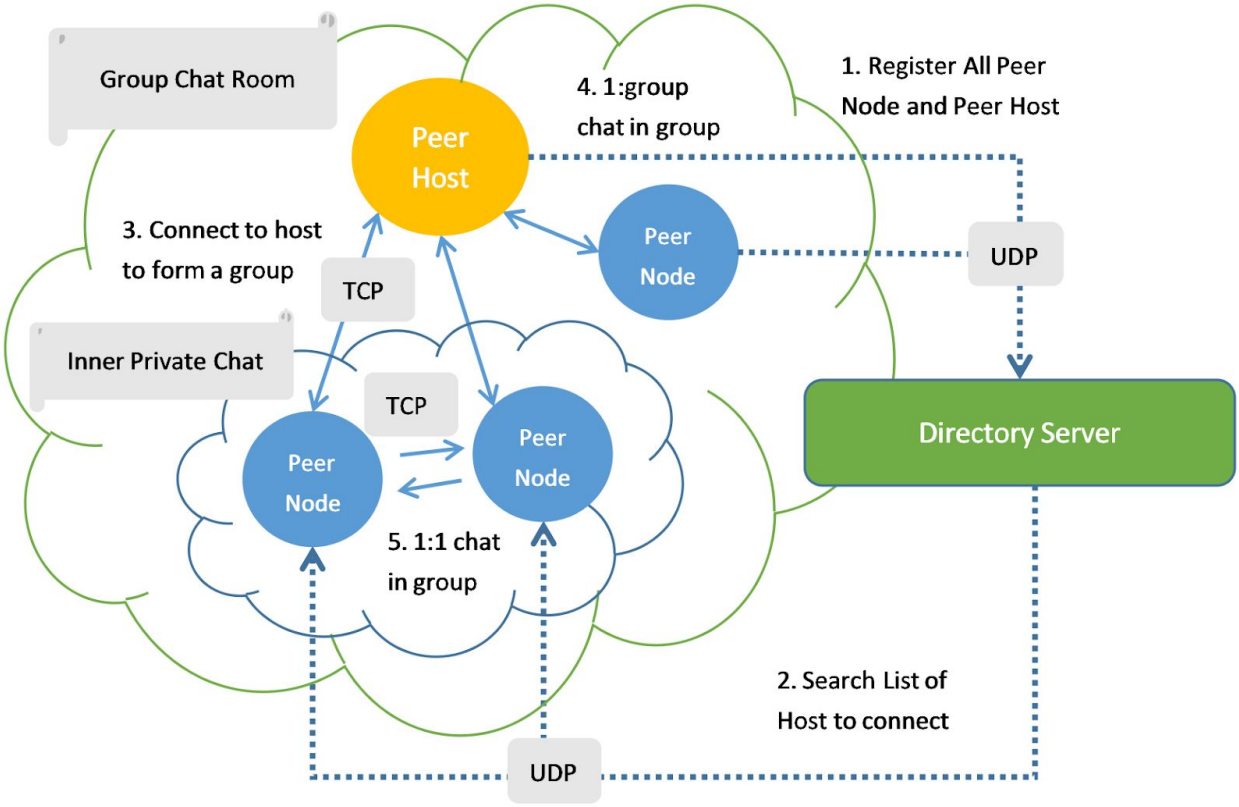
5. Fault Tolerance
Fault tolerance is an important feature to maintain reliable communication within a group. The application implemented a fault tolerance mechanism to allow consistent communication when the host peer node (server) fails. The aim of the fault tolerance algorithm is to handle a peer server failure while all the members in that room are still connected with the failed server. In real life, it handles the situation when the host's computer suddenly shuts down or the application breaks. The mechanism tolerates a server failure by transferring the host position to next alive members in the same group (chat room) to maintain the communication of the rest group members. The chat history of alive members is still saved in the chat room interface and the list of online users would be updated without the failed host peer node. The mechanism is aimed to make the application perform like a normal client leaving, rather than server failure.

6. GUI interface
Our chat application uses a GUI interface utilizing the Java Swing library. The directory server, peer hosting server and chatroom all have their separate interfaces to be able to

monitor any change in the system. We use the GUI buttons to make the user to be online and offline, to host a room, to join a room and to leave a room. The interface provides a section to display all online users and hosts or members of a room.

Chat Workflow

1. All the running peer nodes need to be online in the first place.
2. An online node requests a directory server to send all registered users and hosts.
3. According to returning results, the online node selects a room to join.
4. Aftering successfully joining a group chat room, a node can send and receive messages with the whole group.
5. Aftering successfully joining a group chat room, a node can do 1:1 private chat within the group using “@” annotation.



Key Implementation:

Unicast/Multicast Distribution:

When a user types messages in the chat box, my code checks to see the content of the user’s messages. if my code finds that the message starts with @ symbol, then the code checks to see if there are any spaces exist in the message. if the spaces exist,

then defines the substring between the second character and the character before the first space to be a String object called ‘name’. And check to see if this ‘name’ object exists in the arrayList called ‘onlineUsers’. if this ‘name’ object exists in the arrayList, which means the user wants to send this private message to a specific user. then the code will call a function called ‘tellOnePerson’. In the ‘tellOnePerson’ function, the ‘clientMessage’ arrayList uses the index to find out the specific user that the private message will be sent to. Then this private message is sent to that user and this message is displayed on that user’s chat box.

Fault tolerance:

A hosting server failure is represented by closing the peerServer interface, which also kills the clientChat interface of the host node. When a host fails, all the members in the same room would realize the failure when the readline() function of the connected TCP client socket returns a null value.

A client peer node is implemented using ChatMain class and it stores all the members in the group chat when joining a room. Using the pre-stored timely ordered member names stored in preUserList list, we can get the name of the next client joining the room after the host as the new host. When the new host realizes the failure of the old host, it stops listening to the host and initializes a new PeerServer object on it’s localhost as the new host server. Also, it informs the DirectoryServer object of the host change and disconnection of the old host. The DirectoryServer would update it’s onlineUsers and hostUsers list by deleting the old host on onlineUsers and setting the hostName and numbers of members of previous host to current host.

Except the new host, other ChatMain nodes would wait for 100 ms and reconnect to the new host’s IP and port. After the whole process of rebuilding the PeerServer and all the connections of the group, the chat server remains the same functionalities and the previously failed host can rejoin the room by normal process. Since we didn’t close any initChatClient interfaces, the chat history remains. The clients may not notice the disconnection and host changing while chatting, and only see the previous host have been disconnected from the chat room.

Running Results:

How To Compile Application:

Compile all the java files using command javac filename.java

> javac PeerServer.java DirectoryServer.java ChatMain.java

How to run application:

- Execute the jar files in the packet, DirectoryServer.jar and ChatMain.jar.
  > java -jar DirectoryServer.jar

> java -jar ChatMain.jar

- Run class files created after compiling the java files, the execute with command:
  > java DirectoryServer
  > java ChatMain
- Run the files directly from IDE after importing them.

Using the Application steps:

(It is rather complex running the application, please follow strictly with instructions)

- Execute the DirectoryServer.jar file first and click the “start” button to start the directory service, then execute multiple ChatMain.jar files. (Each execution of ChatMain.jar is a start of a peer node).
- In the Chat User interface, enter a username in the textbox then click the “Go Online” button to join the directory server.
- You may click the “Go Offline” button to test if required.
- Once you have joined the directory server you can now host a chat room.
- Enter the port for which you want to host a chat room, and click the “Host” button.
- Two new windows should appear, (“Peer2PeerServer” and “Chat Client”) and you should be connected in a chat room, capable of typing or disconnecting.
- Clicking disconnect will return you to the main menu where you will have to input your information again.
- At this point, execute another ChatMain.jar. Enter a new username and click the button “Go Online”.
- Click the “query for peers” button, the Host Servers and Users online fields should be populated with online hosts and online users’ list.
- Once you see a user hosting a room in the Host Servers text field, (format should be: username R:X where X is the number of members in the room) you can now join that specific hosted room.
- Enter the username of the person you see in the Host Servers field into the text field next to the “Join” button. Click the “Join” button.
- You should now be in a chat room with that user, and be able to chat with him by entering a message and clicking the “Send” button.
- For private 1:1 communication of the one of the Online Users, add “@<name>”, a space before the message and send. Only the user with the stated name would receive that message and should “(private)” annotation.

Fault Tolerance Demo:

- Using the application as previously stated, start 3 Chat User interfaces and join the 3 members in the same chat room. One of them would host a room.

- Close the Peer2PeerServer interface or the host's Chat Client interface, which means an overall fail of the host peer node.
- You would see a new Peer2PeerServer interface immediately pop up and from the chat history with some host change log like following:
  Host failed!
  Tim is the new host!!
  Tim: has connected.
  Bob: has connected.
  The failed user is also now removed from the Online Users and we have a new host server running on the side of the next user in the chat room.
- You can start a new ChatMain to rejoin the host to the same room, but now the host name would change to the current host's name.

Key Notes:

- You must be connected to the directory server using the "Go Online" button to use the "Host" , "Join" and "Query For Peers" buttons.
- You must use the "Query For Peers" button and get all necessary information appearing in the text fields in order to use the "Join" button.
- Number of current members of a chat room is shown in the Host Servers text field next to the username (R:X).