P2P Chat Application

Extended Feature Design and Implementation

We choose “shout” as our extended feature in the project. In our design of shout function, a peer can use the shout command only if he is currently in a room that he is maintaining or a room that is maintained by other peers. In our implementation of the decentralized chat application, “” is not treated as a room. In other words, a peer needs to join in a room whose name is not an empty string locally or needs to first connect to another peer and join in a room maintained by the peer to send chat message or to issue the shout command. The shout command allows a peer currently in a room to broadcast a message to all peers that are currently in a room and are reachable from the peer in the network structure through single or multiple connections. Starting from the peer that issues the shout command, there are two kinds of situations.

Situation 1

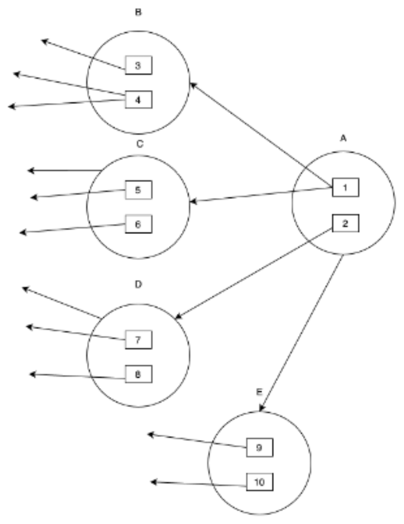


Figure 1

As shown in Figure 1, The shouting peer A does not connect to any peer in the network as a client. In our implementation, we consider peer A to be a self-connected peer which means that it is connected to itself as a client without establishing socket connection and can execute local commands, like “#join” to join a room of its own. In this case, we only need to broadcast the shout message in one direction. As the example shown in Figure 1, peer A is a self-connected peer maintaining two rooms: room 1 and room 2, and it uses the shout command to deliver a message. In this case, peer A will first deliver the shout message to peer B, peer C and peer D who are in room 1 and room 2 of peer A and to peer E who is connected to peer A but not in any room of peer A. After peer B, peer C and peer D receives the shout message, they first process and display it locally and relay the original shout message of peer A to all their connected clients, while peer E does not display it and relays the shout message of peer A to all his connected clients since he is not in any room. We stop propagating the shout message after all peers reachable from peer A have received the shout message.

Situation 2

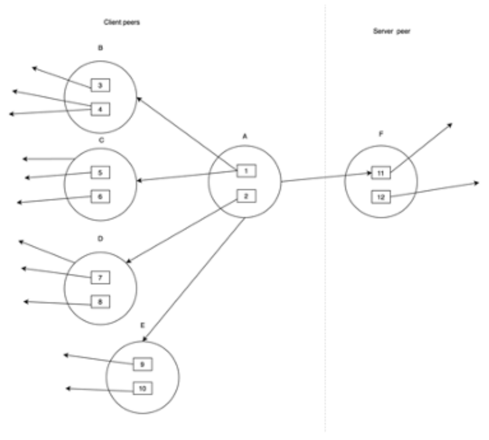


Figure 2

As shown in Figure 2, the shouting peer A is connected to peer F as a client while having peer B, peer C and peer D connected to it as clients. We first propagate the shouting message in the left direction. In other words, peer A will first send the shout message to its clients and then let clients relay the shout message to their clients. After that, peer A will send the shout message to its server peer F. Then we will check if peer F is self-connected. If peer F is self-connected, peer F will simply relay the message to its client peers. If peer F is not self-connected, peer F will first relay the shout message to its server peer and then relay the message to its client peers. We stop propagating the shout message after all peers reachable from peer A have received the shout message.

Shout protocol

Shout protocol	
type	“shout”
identity	Identity of the shouting peer, example: “127.0.0.1:4444 shouted”
content	Shout content, example: “hi!”
identifier	Identifier of the shout regarding the identity, example: “9856”

In our implementation, we use a separate protocol for the shout command. In shout protocol, the type will always be “shout”. For the identity, if the peer does not connect to another peer as client, the peer will use its local address and the listening port in the identity. If the peer is connected to another peer as a client, the peer will use its local address and the source port in the identity. For the content, it is any string message that the peer wants to shout. For the identifier, it is used to identify a shout message in combination with the identity. Every peer in the network will maintain a hash map which maps the identity of the shouter to the identifier which is a randomly generated number between 0 and 10,000 that is used to identify a shout of the identity. If a peer receives a shout message that contains an identifier that is different to the identifier of the identity stored in the peer’s hash map, the peer will update its hash map and relay the shout message. However, if two identifiers are the same, which means that the peer has already processed and relayed the shout message, the peer will do nothing.

Shout and Challenges

Security

All shout messages are sent between peers in plain text without using any encryption and decryption technique. We could suffer from man-in-the-middle attack and the attacker can simply read the message in plain text. One possible solution to this is to use java’s SSLSocket instead of normal socket to communicate securely with ciphers provided by SSLSocket for message encryption and decryption. What’s more, the code can be easily modified by the user. For example, a user can easily modify the code and make the peer keep shouting to flood other peers in the network, in which case other peers can’t work properly anymore. We should not allow the code to be modified by any user.

Failure handling

Our shout achieves error masking. The shouting peer will not see anything happens in other peers during the shout. If another peer not in a room receives the shout, it will silently relay the shout. If it is the second time for a peer to receive the same shout, the peer will simply ignore it. What’s more, our shout command does not support error recovery. If a peer happens to disconnect from the network before the shout being delivered to it, the peer will not be able to see the message ever again. One possible solution is that we can use Java Message Service to ensure reliable message delivery. A Message Queue will be responsible to hold messages before the peer is ready.

Decentralized model

A decentralized Structured Topology (DHT) is composed of individual peers, as figure [3] shows, also it is the network this section refers to.

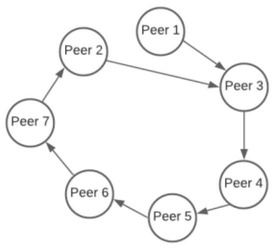
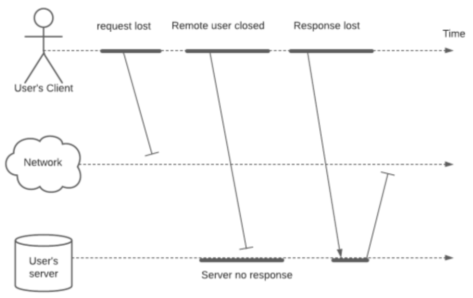


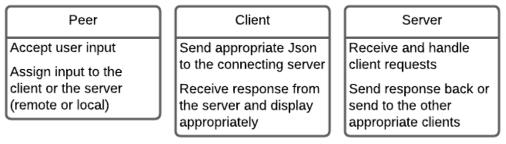
Figure [3]

Peers can directly communicate with other peers and can have a view of the whole network. Besides the complexity that one peer connects to other peers, the number of incoming connections increases the complexity, peer 3 is O(2) while peer 1 has a running server that has no connection. The network cannot guarantee the time or reachable data package transmission. For instance, there could be three possible reasons that the request has not responded, as the figure[4] shown.



Figure[4]

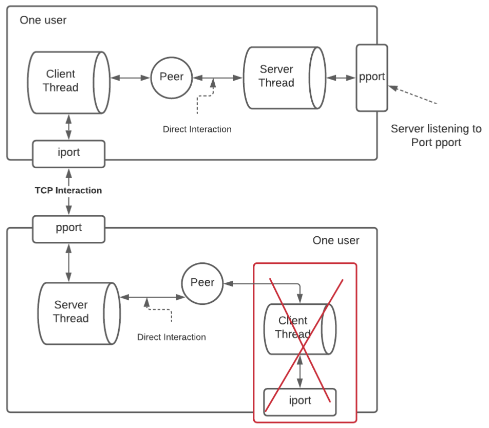
There are three main components in each chat application that users open, we will refer the chat application to the "user" in this case. The "peer" in the "user" refers to an interactive interface.



Peer	Client	Server
Accept user input	Send appropriate Json to the connecting server	Receive and handle client requests
Assign input to the client or the server (remote or local)	Receive response from the server and display appropriately	Send response back or send to the other appropriate clients

Figure[5]

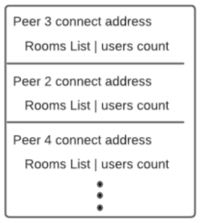
The structure is illustrated in the below figure[6].



Figure[6]

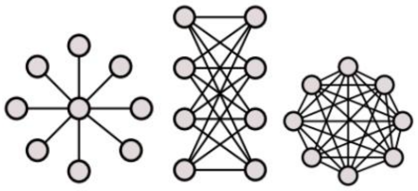
The user can have no connection with other users so that there is no client part as the red cross. In this case, peer will execute commands by directly interacting with the server which is local commands. On the contrary, the command executed by the server belonging to other users will be the remote command. Therefore, when a user starts a chat application, it also starts its own server. The server will be available as long as the user is online.

The application will first check if there is an internet connection by establishing a socket connection with bing.com. If the socket connects succeed, the application will use an internet connection IP. On the contrary, it will use localhost as IP. Each peer only can have one official connection with other peers. Search network command can be used to discover the whole network. As the figure [7] shows the result after peer 1 request.



Figure[7]

Then, the peer could establish a new connection to one of the addresses and the system will automatically shut down the old connection. During the search, with the target address list conducted by breadth-first search, the peer will establish and close TCP connection with each of them which is running in the backend. The search could require a long time to process in terms of the network scale and is limited by reachability between machines. The network could become one centralized network if all users connect to one user so that the user is able to receive all information of the whole network, as the first pattern in the figure[8].



Figure[8]

There is a maximum limit of socket connection of one operating system. The workload is multiplicatively increased when new and close connections. Especially for commands such as shout and search network, the time required until reaching the furthest node in the third pattern in figure[] will be faster over time as the node will update the mapping table so that it can know which route is currently fastest, similar as Chrod. Since there is no socket connection when the user is only in local mode (no connection with other servers), the request and response have no delay caused by the socket.

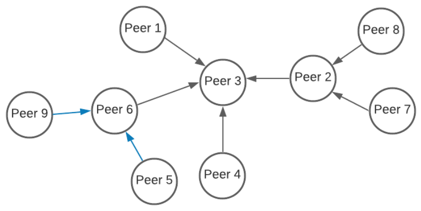
Failure Model

Network issues always surprise people, such as shark attack network cable. The possible network failure in chat application as follows:

- Request may be lost (user cut the network cable)
- Request in the waiting lane, cannot be sent immediately (the network or receiver is already overloaded)
- Remote peer server invalid (the remote server crashed)
- Remote peer server temporarily has no response (remote server could pause for a second)
- Remote peer server processed the request, package lost during sending back (could be the wrong configuration of network switch)
- Remote peer server processed the request, delay on replying (the network or sender is overloaded)

Assuming the network is reliable. The user will be notified if the socket connection is lost. All requests need the response from the server on the application level. For instance, TCP confirms one data package has been sent to the target node but the application may crash before it finishes the process. Then the user will try to reconnect and make the final statement that the node is invalid after reconnecting timeout.

The network could become separated when peer has no income connection and lost connection with other servers. As figure[9] shown, peer 9 and peer 5 will be separate if they only know the address of peer 6.



Figure[9]

Assume we know the mean time of one peer failure. Search network before the mean time so that the user can have other peer options.

In the case of the server of one user crashing or being terminated, connecting clients will try to reconnect. When one client connects, the server stores the server address of that incoming client connection in a list. If the server recovering time is longer than the clients reconnect waiting time, the server will send notifications to the clients based on the list.

The application uses TCP to guarantee in-order delivery. TCP stream supports flow control and re-transmits lost data package, it is biased on reliability compared to UDP message. If TCP does not receive acknowledgement in a timeout range, it considers the data package lost and automatically re-transmits. This process is transparent to the application and will bring extra delays such as waiting for the timeout to expire, waiting for acknowledgement of re-transmits data package.

Word Interruption

Server: the server thread of the chat application of one node (user) in peer-to-peer network described in the model section
Client: the client thread of the chat application of one node (user) in peer-to-peer network described in the model section