Chatterbox

The following API calls will be implemented as part of UDP based distributed chat system:

MESSAGE TYPE	**FUNCTIONALITY**
JOIN	Handles new members in chat, if successful returns NOTICE to everyone that new member added the chat and sends LIST to the new node, If the join failed we display a error message
ELEC	Implements a election algorithm If the leader dies to detect the new leader and the response will be REPL from the newly elected leader and NTCE to everyone about the new leader
CHAT	Handles simple chat messages, REPL type acknowledgement if delivered
PING	Leader checks if all nodes are alive by pinging them at regular intervals, response is of type REPL, if the leader didn't get the reply then it considers the node is dead and send NTCE to everyone
REPL	Indicates successful delivery of message, can serve as a generic acknowledgement
LIST	Sends list of all nodes (Users) on chat system currently
NTCE	If a node joins a chat or leaves a chat the leader sends a notice to everyone and they update their local list and in case of leader dies the new leader sends a NTCE to everyone to update the current leader in their local list

The above api calls types are sent using different servers or appending corresponding case numbers to the message itself, which will be explained in detail.

The list of data structures we are planning to use for implementing the multi-threaded chat functionality:

Data Structure	Functionality
map<key, value> memberInfo	The map contains user information i.e key - ipaddress:portNo,Value - vector of UserName and leaderOrNotInfo
BlockingQueue<message> holdbackQueue	This queue will handle all the pre-processing for chat messages in the client side until the message is delivered to the leader.
BlockingQueue <message> sequencerQueue	This is the main sequencer queue which will be sending the chat messages to all the live nodes (Users) in the system after pre-processing.
BlockingQueue<message> printQueue	This queue handles the message to be printed on every client in the chat

Limitations:

- The messages exchanged over network is of type character array and are limited to size 2048 bytes
- Username is char array of size 32 bytes
- We used the ‘-’ and ‘:’ as a delimiters in most of the cases,so the chat works in most of the cases, but we prefer eliminating those in the chat messages

UDP Framework:

1) ‘OPEN’ a new ChatterBox:
    - ➢ The program runs in a main loop (while(!shutdown)) and does the following: Creates a new socket connection and listens for any new incoming messages
    - ➢ The creator of the chat will add himself to the userInfo map and elect himself as the leader of the ChatterBox by default

2) ‘JOIN’ an existing ChatterBox:
    - ➢ Case 1: Contact the current leader’s ip:port
        - The leader can directly add the new member to the memberInfo map if the ‘JOIN’ request is successful
        - The leader then sends the status message to all the members in the chat that a new member is added and passes a map of updated memberInfo with the information of the new member
        - All nodes then send ‘REPL’ to the leader to confirm that all nodes have same set of userInfo map.

- ➢ Case 2: Contact any other node's ip:port
    - ● The node which receives the 'JOIN' request by the new member then passes on the information (ip address and port number) of new member to the elected leader of the chat
    - ● Then the leader will add the new member to the userInfo map if the JOIN request is successful
    - ● The leader then sends the status message to all the members in the chat that a new member is added and passes the information of the new member which will be updated in the clients memberInfo map
    - ● All nodes then send 'REPL' back to the leader to confirm that all nodes have same set of userInfo map.

3) Regular 'CHAT' messages which need to be sent to centralized sequencer:
    - ➢ Initially all the chat messages are added to the holdBackQueue we initially read the front element in holdBackQueue and send it to the sequencer which add it to the sequencer main queue (sequencerQueue). We then wait for the 'REPL' from the sequencer, if successful then we pop the message from holdBackQueue and if failed then we re-transmit the same message

4) Finally, Sequencer messages to be displayed on the chat window of each user:
    - ➢ The first message from the Queue is sent to all the nodes to be printed on the display window
    - ➢ All the nodes sent a 'REPL' message back to the leader.
    - ➢ If the leader gets acknowledgement from all the clients then the transmission is successful and the first message is popped out of the sequencerQueue. And if the number is not equal then the message is retransmitted to particular node twice, even then we couldn't deliver we move on to the next client

5) The Heartbeat Mechanism to find the alive nodes or the dead leader
    - ➢ The leader keeps pinging all the clients for every 5 seconds and if the client is alive it responds back that he is alive, it keeps happening for all the clients in a regular interval and the leader didn't get a response from a client then we declares he is dead and sends a notice to everyone that the particular client is dead and everyone remove the client from their memberInfo map.

➢ If the client didn't receive a ping from the current leader for more than two cycle period then the client comes to a conclusion that the current leader is dead and calls for the next mechanism Election

6) The Election mechanism follows the dead leader:

➢ If the client detects the leader is dead it pings the highest port in the memberInfo and if the client who detected the leader is dead and he has the highest Ipaddress:PortNo combination then he declares himself as a leader and send notification to everyone and starts the pinging process and all other threads

➢ In case the current highest port is also dead then we will come to know in 12 seconds, which is the two cycle period + some delay, then we ping the next highest member in the map and the process continues

Functions in the files we created:

Udp-server.cpp - Server-Udp

we created different objects of the Server-Udp and used them to perform communication without any hindrance between communication in two different threads

*server_Udp::receive_String(struct sockaddr_in clientaddr,char *buf, size_t BUFSIZE) -*

This function is called to send a string from a current node to the respective node in need

*server_Udp::send_String(struct sockaddr_in clientaddr,char *buf, size_t BUFSIZE) -*

Vice Versa

blockingQueue.h

We built a class where queue as its member and the push will happen in the queue only If the queue is not full and pop happens only when the queue is not empty, if that's not the case we created a lock in such a way that it waits until any of the above case becomes a possibility in the push() and pop() functions.

member.cpp

void addMember (string key, string username, bool lead)

Where it is used:

Used to add new member to the memberInfo map, when a leader gets a request that the new client wanna join the chat, he adds the new member to it's own memberInfo map and also sends a notification to all the other clients so they can add to their own memberInfo map

char* getListOfMembers ()

Where it is used:

It is used to create a char* of all the memberInfo, when the new user is created the leader use this function to pass the map info as a string

void deleteMember (string entry)
Where it is used:
In case if the client leaves the chat the leader sends notification about the crash and we remove the particular node from the map and same in case of dead leader

string getUserName(string key)
Where it is used:
It is used while printing the chat, the message of the corresponding client will be printed with the their name preceding the chat

void getMapFromString (char * input)
Where it is used:
When the new client chat joins the chat, the leader sends the memberInfo map as string, we used this function to get the map details from the string and populate the local memberInfo

string* listOfUsers()
Where it is used:
It is also used for the printing purpose when the new client joins the chat, we use this function to display the current members in the chat

string Leader()
Where it is used:
This function returns the leader of the current chat

void newLeader (string entry)

Where it is used:

This function is used to change the current leader to the new leader info passed, this function is called when the current leader dies and the new leader has been elected

Threads Used in dchat

pthread_create(&th1,NULL,dispatcher,(void*) ipaddress);
pthread_create(&th2,NULL,latest_join,(void*) ipaddress);

Takes care of the client who is requesting to join the current chat via leader or through other clients

pthread_create(&th3,NULL,print_handler,(void*) ipaddress);

Takes care of printing the chat message on all the clients in order

pthread_create(&th4,NULL,&check_live_members,(void*) &args);

Takes care of the pinging the clients for each 5 seconds in order to detect the dead client (this thread is ran only by the leader)

pthread_create(&th5,NULL,deadLeader_handler,(void*) ipaddress);

This threads reply back for the ping from the leader and also to detect whether the leader is alive or not

pthread_create(&th6,NULL,chat_send,(void*) ipaddress);

This thread just keep checking if there is input message from the client, if so, then it sends the message to the leader to add it to the sequencerQueue, if not keep trying to communicate for three time, then move on to the next message

pthread_create(&th7,NULL,join_send,(void*) ipaddress);

Takes care of the client who wanna join the existing chat, in the beginning

Extra Credits:
We have implemented the following extra credits as part of Distributed Chat system:

1) <u>Encryption:</u>

All messages exchanged over the network will go through a substitution cipher algorithm, where we are moving the value for each character by 5 ascii value (e.g *hi will be encrypted as mn*) and the encrypted message is printed on the leader side and the decrypted message will be outputted on all the other clients side

2) <u>Traffic control:</u>

If the sequencer queue is overwhelmed by messages sent mostly by one member, (for example if a single client sends 200 messages in 5 seconds, we made that particular client to sleep for a second and for next 1 second it can only send 10 messages in 0.1 second interval and after that it gets back to it's own speed in that way we can give chance to other slower clients to transmit the message over the network) we use traffic control.

3) <u>Fair Queuing:</u>

______We implemented Fair queueing by a hash function (mod of 5), where the port number of the whose message to be added to the sequencer main queue will be divided by 5 and found the mod value of the port number. Instead of using one sequencer queue we used 5 blocking queues as sequencer queues.

```
BlockingQueue <message> sequencer0Queue
BlockingQueue <message> sequencer1Queue
BlockingQueue <message> sequencer2Queue
BlockingQueue <message> sequencer3Queue
BlockingQueue <message> sequencer4Queue
```

Based on the value of hash_fun we add the message to the corresponding queue and we ran different threads of print_handler for the above queues and the threads will be executed in a interval and will be blocked until that particular queue has a value by using the concept of blockingqueue class we built, this function provides fair queueing by giving chance for all the port values, as we generated the port values for all the queue randomly, mostly the probability distribution of the messages in the above 5 queues will be equal.

4) <u>Decentralized total ordering:</u>

We implemented the decentralized algorithm where there is no leader and all the clients transfer the message to all the clients in the chat and anyone can join the chat via any client and the map in all the members in chat will be updated accordingly and the chat messages are transferred in such a way that we chose a random sequence number Pg = max(Pq g , Ag)+1 and send it to all the clients, the clients who wanna print the message at the same time proposes a sequence number whoever has the highest sequence number will be chosen and he sends YES-sequenceNumber <m,t> to everyone so that they know there is someone has higher priority than him, which is t and the Ag of the current user is updated =max(Ag, t) and it tries until it gets the acknowledgement and the process continues, we used a random number generator for this purpose so when it reaches highest value at some point we changed the Ag back to 0 and the process continues

5) <u>Priority message:</u>

______We implemented Priority message in the original version of the code, where join, chat and print everything happens in their own thread, in case the leader is dead we gave highest priority to the election mechanism, until the new leader is detected upcoming notices and the chat messages are added to their respective queues and printed after the leader is elected

6) <u>GUI:</u>

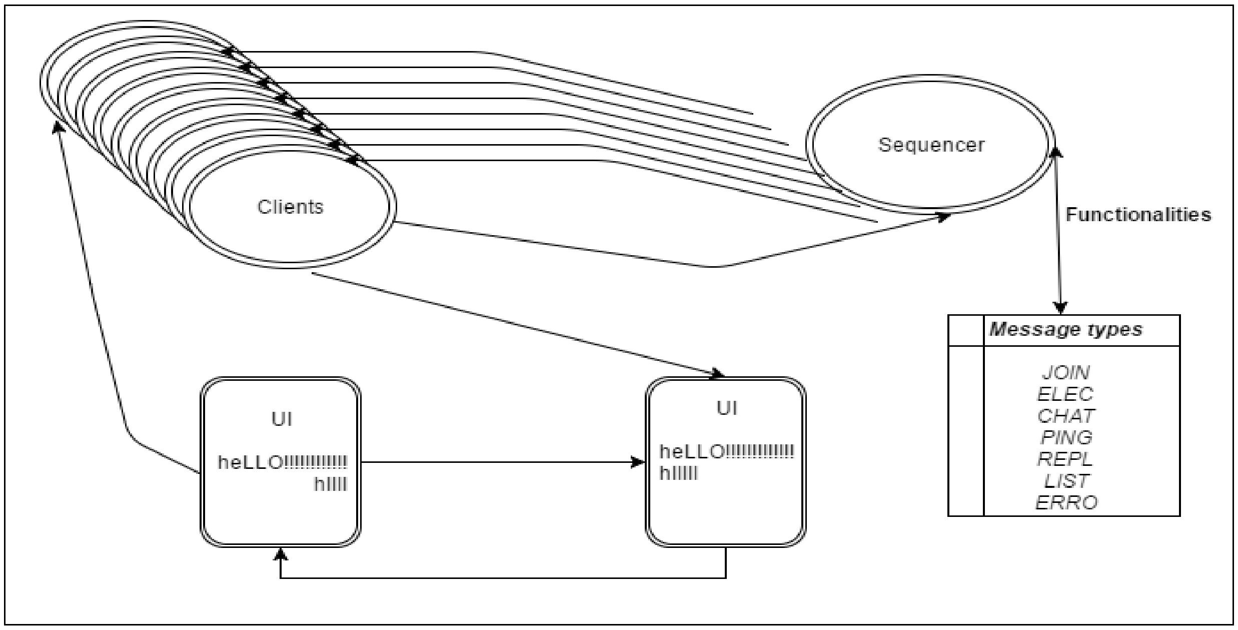
We tried implementing UI component of the chat by constantly redirecting standard input and output of dchat to python tk-inter library. The UI chat message is captured using Entry component and in real time updating UI from stdout. However, we could not get it to work in the given time frame, hence turning in partial code. Kindly accept.

Compiling the file:

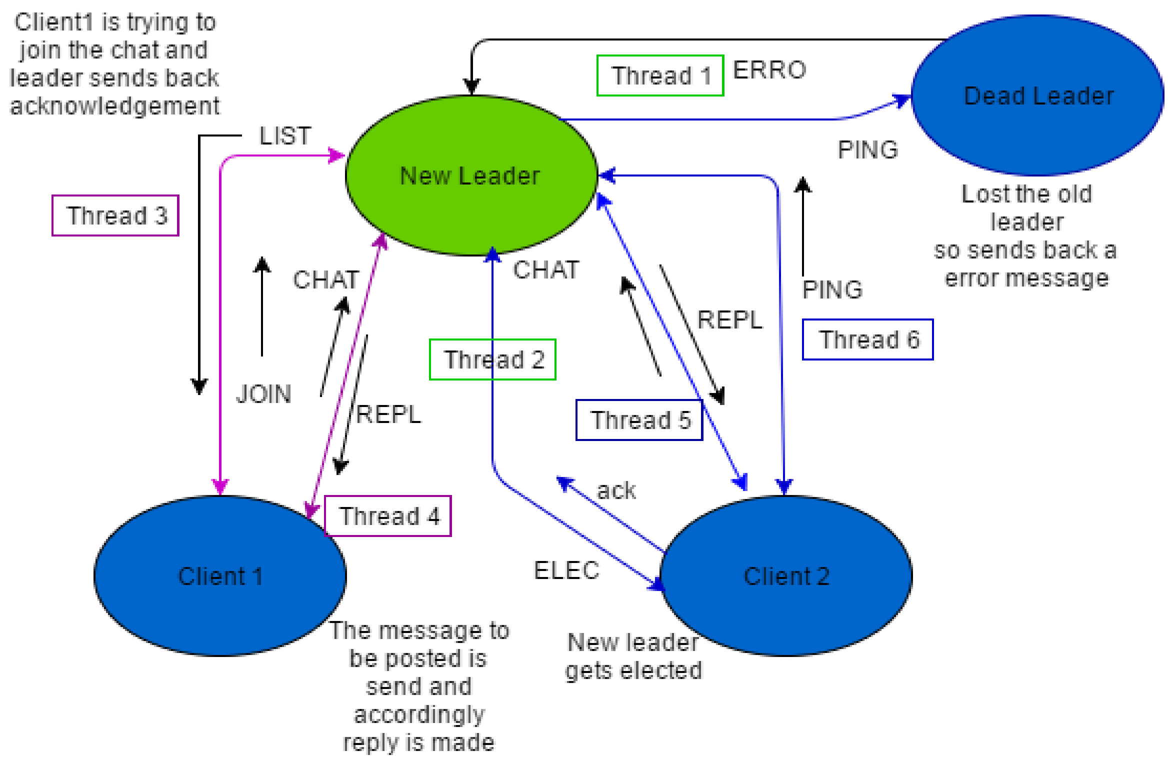
```
actual: \
compileActual
    ./dchat bob

compileActual: \
dchat.cpp
    g++ -std=c++11 dchat.cpp -o dchat -lpthread -lboost_system

encrypted: \
compileE
    ./dchat bob

compileE: \
dchat.cpp
    g++ -std=c++11 dchat_encryption.cpp -o dchat -lpthread -lboost_system

fairQ: \
compileQ
```

```
    ./dchat bob

compileQ: \
dchat.cpp
    g++ -std=c++11 dchat_fairqueuing.cpp -o dchat -lpthread -lboost_system

dec: \
compileD
    ./dchat bob

compileD: \
dchat.cpp
    g++ -std=c++11 dchat_decentralized.cpp -o dchat -lpthread -lboost_system

traffic: \
compileT
    ./dchat bob

compileT: \
dchat.cpp
    g++ -std=c++11 dchat_trafficcontrol.cpp -o dchat -lpthread -lboost_system
```

The Basic flowchart of ChatterBox



Architecture



Client1 is trying to join the chat and leader sends back acknowledgement
LIST
Thread 3
New Leader
Thread 1
ERRO
Dead Leader
PING
Lost the old leader so sends back a error message
CHAT
CHAT
JOIN
REPL
Thread 2
REPL
PING
Thread 6
Thread 5
ack
Thread 4
ELEC
Client 1
Client 2
The message to be posted is send and accordingly reply is made
New leader gets elected