

CIS-505 (Software Systems)

Distributed Chat System

Final Report

Group Members

Kiran George Vetteth
Prathik Prakash
Rishabh Gulati

Compilation

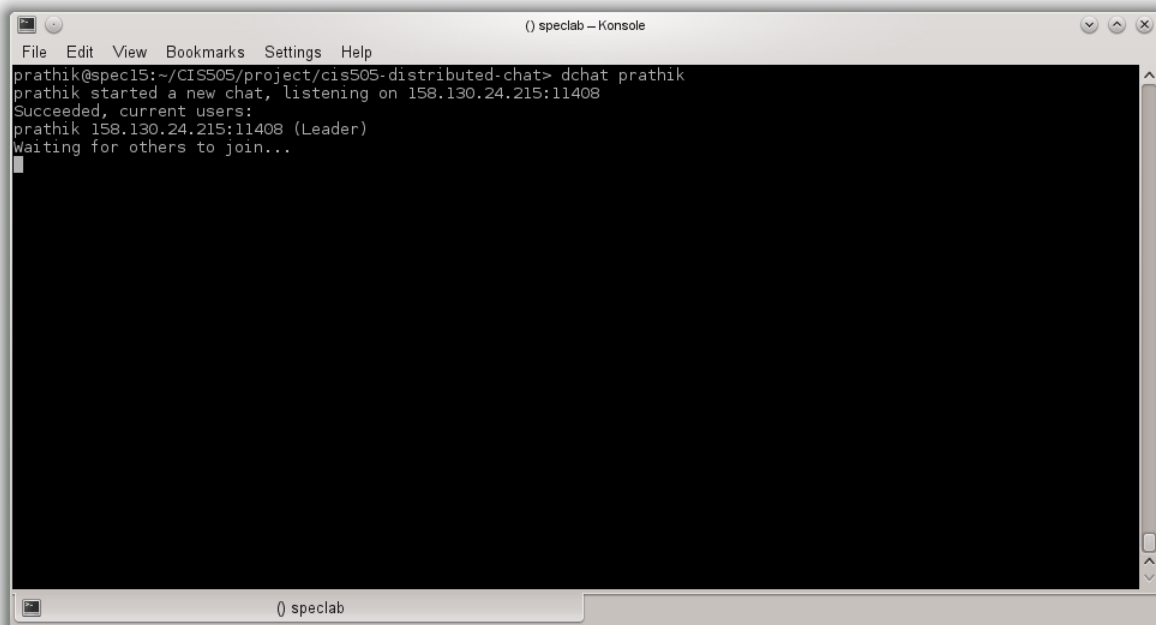
Type the following command to get a dchat executable from the files submitted

```
g++ main.cpp -pthread -std=c++11 -o dchat
```

Working

Distributed chat system can be initiated with the command line argument
dchat username

This begins the chat system with “username” as the leader.

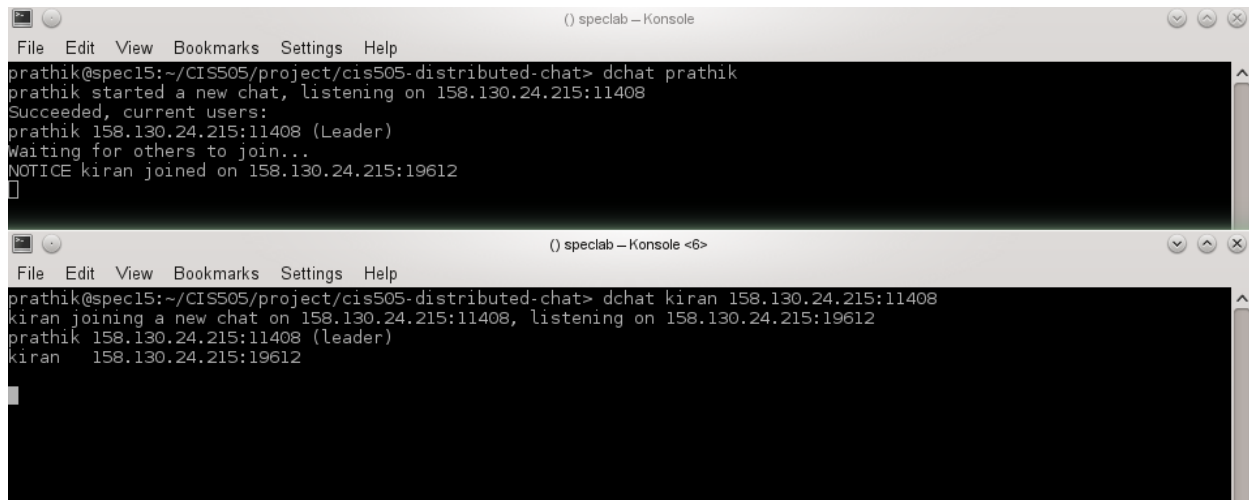


```
() speclab - Konsole
File Edit View Bookmarks Settings Help
prathik@spec15:~/CIS505/project/cis505-distributed-chat> dchat prathik
prathik started a new chat, listening on 158.130.24.215:11408
Succeeded, current users:
prathik 158.130.24.215:11408 (Leader)
Waiting for others to join...
```

To add clients to this particular chat, we need to type the following command line argument,

dchat client1 leaderIP:leaderPort

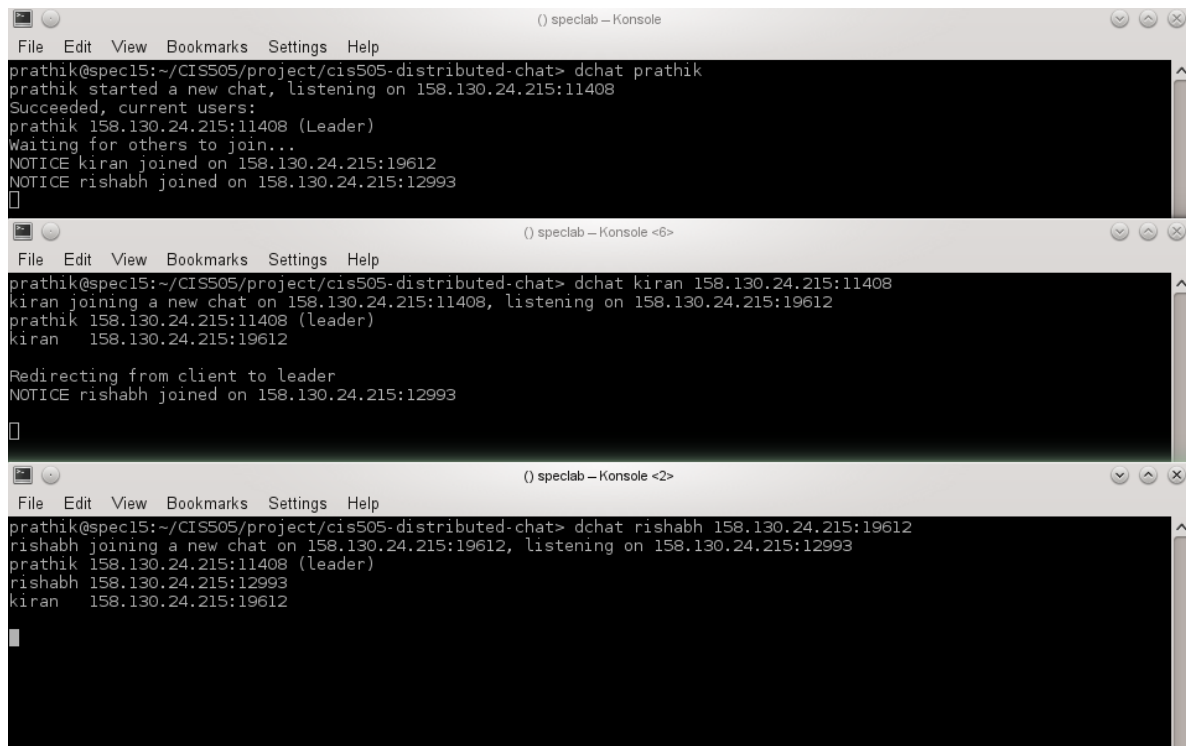
where client1 is the name of the client who wants to be in the existing chat and leaderIP and leaderPort are the IP address and port of the the leader who is already in a chat.



```
prathik@spec15:~/CIS505/project/cis505-distributed-chat> dchat prathik
prathik started a new chat, listening on 158.130.24.215:11408
Succeeded, current users:
prathik 158.130.24.215:11408 (Leader)
Waiting for others to join...
NOTICE kiran joined on 158.130.24.215:19612
[]

prathik@spec15:~/CIS505/project/cis505-distributed-chat> dchat kiran 158.130.24.215:11408
kiran joining a new chat on 158.130.24.215:11408, listening on 158.130.24.215:19612
prathik 158.130.24.215:11408 (leader)
kiran 158.130.24.215:19612
[]
```

Once the basic chat system has been set up between a leader and a client, the other users can join the chat by using IP address and port of any member of the chat including the chat. If the leader has been contacted, then the client is directly added to the leader client info map. If any other client is contacted with a request to join the chat, the client redirects the requested program towards the leader, who adds him to the chat.



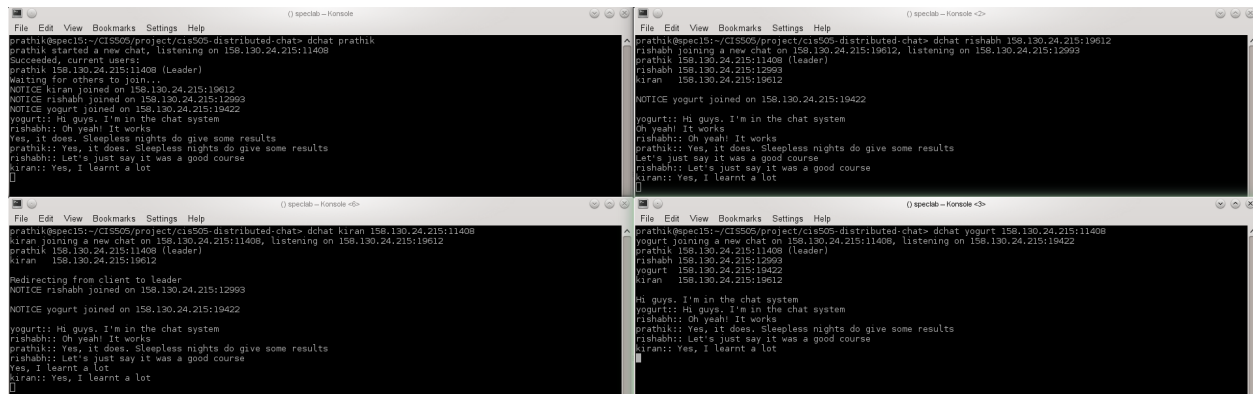
```
prathik@spec15:~/CIS505/project/cis505-distributed-chat> dchat prathik
prathik started a new chat, listening on 158.130.24.215:11408
Succeeded, current users:
prathik 158.130.24.215:11408 (Leader)
Waiting for others to join...
NOTICE kiran joined on 158.130.24.215:19612
NOTICE rishabh joined on 158.130.24.215:12993
[]

prathik@spec15:~/CIS505/project/cis505-distributed-chat> dchat kiran 158.130.24.215:11408
kiran joining a new chat on 158.130.24.215:11408, listening on 158.130.24.215:19612
prathik 158.130.24.215:11408 (leader)
kiran 158.130.24.215:19612

Redirecting from client to leader
NOTICE rishabh joined on 158.130.24.215:12993
[]

prathik@spec15:~/CIS505/project/cis505-distributed-chat> dchat rishabh 158.130.24.215:19612
rishabh joining a new chat on 158.130.24.215:19612, listening on 158.130.24.215:12993
prathik 158.130.24.215:11408 (leader)
rishabh 158.130.24.215:12993
kiran 158.130.24.215:19612
[]
```

Following is a diagram which shows 4 chats using the same system,



```
prathik@spec15:~/CIS505/project/cis505-distributed-chat> dchat prathik
prathik started a new chat, listening on 158.130.24.215:11408
Succeeded, current users:
prathik 158.130.24.215:11408 (leader)
Waiting for others to join...
NOTICE kiran joined on 158.130.24.215:19612
NOTICE rishabh joined on 158.130.24.215:12993
NOTICE yogurt joined on 158.130.24.215:19422
yogurt:: Hi guys, I'm in the chat system
rishabh:: Oh yeah! It works
Yes, it does. Sleepless nights do give some results
prathik:: Yes, it does. Sleepless nights do give some results
rishabh:: Let's just say it was a good course
kiran:: Yes, I learnt a lot
]

prathik@spec15:~/CIS505/project/cis505-distributed-chat> dchat rishabh 158.130.24.215:19612
rishabh joining a new chat on 158.130.24.215:19612, listening on 158.130.24.215:12993
prathik 158.130.24.215:11408 (leader)
rishabh 158.130.24.215:12993
kiran 158.130.24.215:19612
NOTICE yogurt joined on 158.130.24.215:19422
yogurt:: Hi guys, I'm in the chat system
rishabh:: Oh yeah! It works
prathik:: Yes, it does. Sleepless nights do give some results
Let's just say it was a good course
rishabh:: Let's just say it was a good course
kiran:: Yes, I learnt a lot
]

prathik@spec15:~/CIS505/project/cis505-distributed-chat> dchat kiran 158.130.24.215:11408
kiran joining a new chat on 158.130.24.215:11408, listening on 158.130.24.215:19612
prathik 158.130.24.215:11408 (leader)
kiran 158.130.24.215:19612
Redirecting from client to leader
NOTICE rishabh joined on 158.130.24.215:12993
NOTICE yogurt joined on 158.130.24.215:19422
yogurt:: Hi guys, I'm in the chat system
rishabh:: Oh yeah! It works
prathik:: Yes, it does. Sleepless nights do give some results
rishabh:: Let's just say it was a good course
Yes, I learnt a lot
kiran:: Yes, I learnt a lot
]

prathik@spec15:~/CIS505/project/cis505-distributed-chat> dchat yogurt 158.130.24.215:11408
yogurt joining a new chat on 158.130.24.215:11408, listening on 158.130.24.215:19422
prathik 158.130.24.215:11408 (leader)
rishabh 158.130.24.215:12993
yogurt 158.130.24.215:19422
kiran 158.130.24.215:19612
Hi guys, I'm in the chat system
yogurt:: Hi guys, I'm in the chat system
rishabh:: Oh yeah! It works
prathik:: Yes, it does. Sleepless nights do give some results
rishabh:: Let's just say it was a good course
kiran:: Yes, I learnt a lot
]
```

Data Structures

- **Maps**

We have used the following 5 maps to keep track of leader, client information and their messages.

On the client end we have 3 maps,

```
//key --- client message sequence number, value --- message payload
extern map<int,string> client_send_map;
```

This map is used to send messages to the leader. To deal with the possibility that messages will be lost, re-ordered, or duplicated by the network the serialised string packet is indexed by its unique sequence number. The entry from the map is deleted only when the ack is received for a particular message.

```
//key --- leader message sequence number, value --- message payload
extern map<int,string> client_receive_map;
```

This map receives messages from the leader_send_receive map and displays it on the console. The sequence number, indexing the received serialised packet, ensures global sequencing which is set by the leader.

```
//key --- clientID, value --- message payload
extern map<string,clientinfo> client_info_map;
```

This map is used to make sure that every client has information about all other clients in the network as well as the current leader, this is used when leader re-election is to be initiated when the initial leader exits/crashes.

The leader uses 2 maps to keep track of clients and for ordering of messages

```
//key --- leader message sequence number, value --- message payload
extern map<int,string> leader_send_receive_map;
```

This is the sequencer map which is used to send to and receive messages from, all the clients and the leader himself. The chat messages which are sent from the clients are received and sequenced as they come in and a global sequence number is used to index each message to maintain ordering.

```
//key --- clientID, value --- message payload
extern map<string,clientinfo> leader_client_info_map;
```

This map is used by the leader to keep track of all the clients in the chat. Also this map is periodically sent to all the clients which copy them to client_info_map.

- **Structures**

We have used the following structures to send and receive service and chat messages

```
// Client map structure sent and recieved
typedef struct sockaddr_in clientsock;
typedef struct
{
    string name;
    int DeadCount;
    bool IsAlive;
    int ACK_No;
    int SEQ_No;
    string CurrentLeaderID;
    clientsock client;
}clientinfo;
```

- name is a string which contains either leader or client name
- DeadCount is to keep track if the client is alive or not
- IsAlive tells the condition of the client to the recipient
- ACK_No and SEQ_No is used to keep track of global ordering of messages
- CurrentLeaderID contains the Current Leaders ID

- client is the sockaddr_in of the client

```
typedef struct
{
    string LeaderUID;
    string ClientUID;
    string GroupUID;
}UID;
```

- LeaderUID is the leaderID which is a combination of IP and Port in the form of IP:Port
- ClientUID is same as leaderID but for a client
- GroupUID is the reverse of LeaderUID

```
typedef struct
{
    string Msg_Type;
    string client_name;
    int Seq_No;
    int Ack_No;
    UID Access;
    clientsock client;
    string user_message;
}user;
```

- Msg_Type tells if the message is a chat message or a service message. Also there are various type of service messages which will be explained further.
- client_name is the name of the client
- Ack_No and Seq_No is used to keep track of global ordering of messages
- UID is the structure which was explained earlier
- client is the sockaddr_in of the client
- user_message can either have chat message or any status messages which are transmitted with service messages.

//Structure used to select(switch) function based on Msg_Type

```
typedef struct
{
    string Msg_Type;
    string Serialized_payload;
}recievedmsg;
```

- This structure is to quickly ascertain what type of message has been received and what needs to be done with that message.
- Serialized_payload contains the serialised message as received from the socket

Message Types

- JOIN: joining client to node.
 - Contains
 - Requesting Client name
 - Requesting ClientID
 - Requesting Client Sequence number
- JOIN_ACK_MEMBER: member to joining client containing leader address/port
 - Contains
 - Client name
 - ACK No = Requesting Client sequence number
 - LeaderID
 - LeaderIP
 - LeaderPort
- JOIN_ACK_LEADER: leader to joining client
 - Contains
 - Requesting Client Name;
 - ACK No = Requesting Client sequence number
 - Leader UID
 - LeaderIP
 - LeaderPort
- MULTICAST_REQUEST: message to the leader from client
 - Contains
 - leader name
 - Ack No. = Requesting Client Seq No.
 - Message
- MULTICAST_REQUEST_ACK: leader to message initiating member
 - Contains
 - Requesting client UID
 - ACK No = Requesting Client sequence number
- MULTICAST_BROADCAST: leader to ALL members
 - Contains
 - leader sequence number
 - Message
- MULTICAST_BROADCAST_ACK: Clients to leader
 - Contains
 - client name
 - ACK No = leader sequence number

- SERVICE: leader to all members containing active clients list
 - Contains
 - Map which contains information of all the active clients
- LEADER: Send by the client to the newly elected leader
 - Contains
 - Leader UID
- LEADERACK: Every Acknowledges the SERVICE message
 - Contains
 - client name
- NOTICE: To alert the changes in the system to every user in the chat system
 - Contains
 - Member name
 - Member port and IP
 - Event - entry or exit

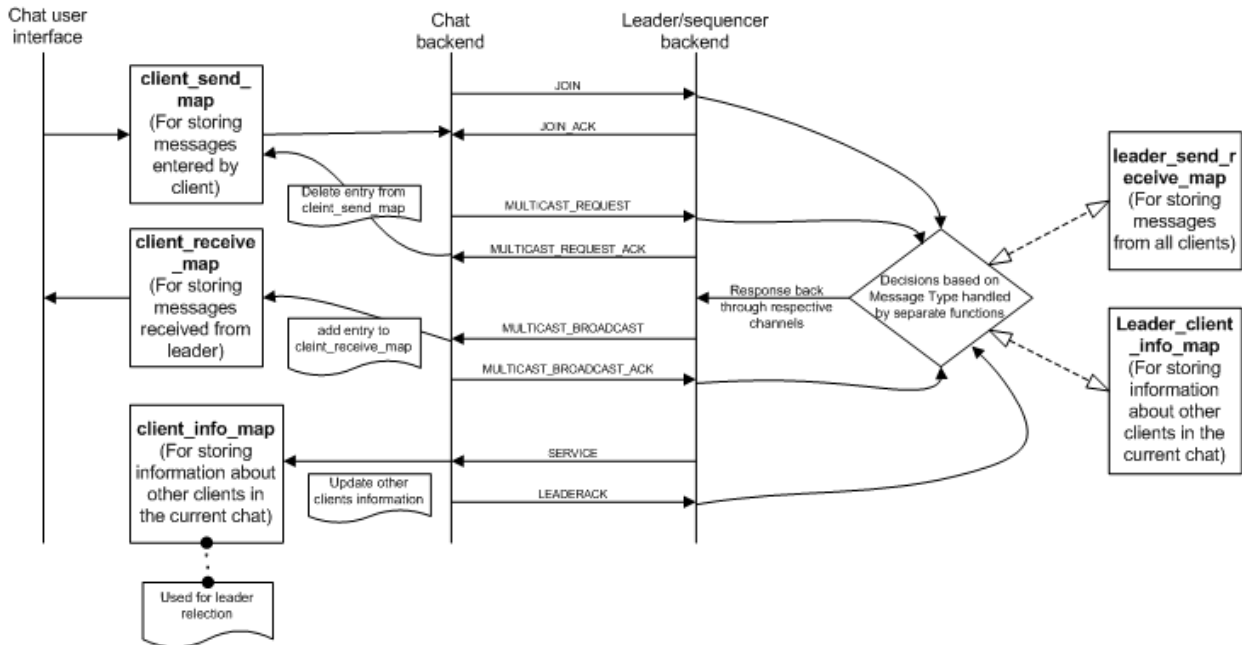
Few observations about the distributed chat system.

- Control characters when entered sends a blank space to all the users in the chat.
- Ctrl+D (EOF) ends the chat properly
- Ctrl+C and Ctrl+Z exits the chat abruptly
- Incase there is heavy load on the network/system, there might be a case where we would need to wait for a few seconds after leader re-election to avoid loss of messages.

System flow

- Client enters the command JOIN for joining the chat which is sent via client_send_map, it is received by the leader, which stores the client info in the leader_client_info_map and sends a JOIN_ACK back
- Once this connection is setup the chat messages that are typed in by client go into the client_send_map which is sequenced using the appropriate sequence number and a MULTICAST_REQUEST message is sent to the leader, the leader stores the received value in the leader_send_receive_map and sends a MULTICAST_REQUEST_ACK back to client receiving which the entry corresponding to that sequence is erased from the map.
- To broadcast the messages, the leader has to check the messages in the leader_send_receive_map and use a global sequencing number which maintains the global ordering, once the clients receive this message, they take in the messages that come in, put it on their client_receive_map and send a MULTICAST_BROADCAST_ACK to the leader, on receiving all the ACKs from all the clients, the leader deletes it's entry.
- Other than the regular chat messages, there are various service messages which are sequenced with a keyword SERVICE and if the leader services it, it sends back a LEADERACK

Systems Design & Message Flow



Leader Election (Sample flow)

2) Rishabh sends LEADER message to Prathik to be the next leader since his ID is the greatest between them

3) Prathik, once he receives the LEADER message now acts like a leader and continues the program flow.

NOTE :

=>In case there were more than 2 users left when the leader died, all would have found the same leader with the highest ID and pinged him to be the leader

=>Rishabh finds out if the leader is elected when he starts getting the SERVICE messages from the new leader(in this case Prathik)

=>Under normal test conditions the leader Election works very smoothly and converges really fast (i.e. less than 1 sec) In case of heavy traffic and higher system usage, thread scheduling is slower and the converging takes slightly longer.

Existing client
Rishabh (ID - 100)



Existing client
Prathik(ID - 200)



~~Existing client **Kiran (ID - 300)**
(sequencer/leader)~~

1) Kiran dies



Flow Diagram - Client joining through existing client in the chat

