DistChat - Demo 5

Team 7

Change Log (Updated November 24, 2016)

- Changed Server Chain model to the Server Network
- Removed mention of clients using ***groupID*** to identify chatrooms
- Added a “login” component
- Updated protocol to match new changes
- Added a database section

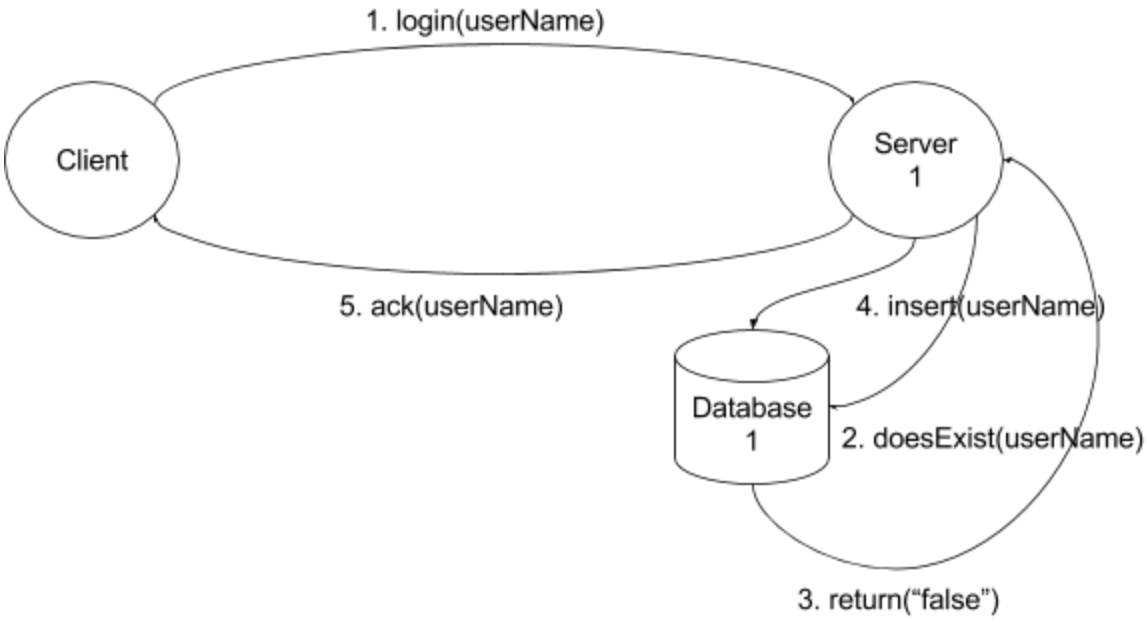
System Design

The system will use three different levels of interactions: Client-to-Server, Database, and the Server Network. DistChat is planned to be a closed system where all processes are made by the developers with no third-party applications. The system will be asynchronous meaning that all processes use a logical clock that is modified based on particular events; in this case receiving packets.

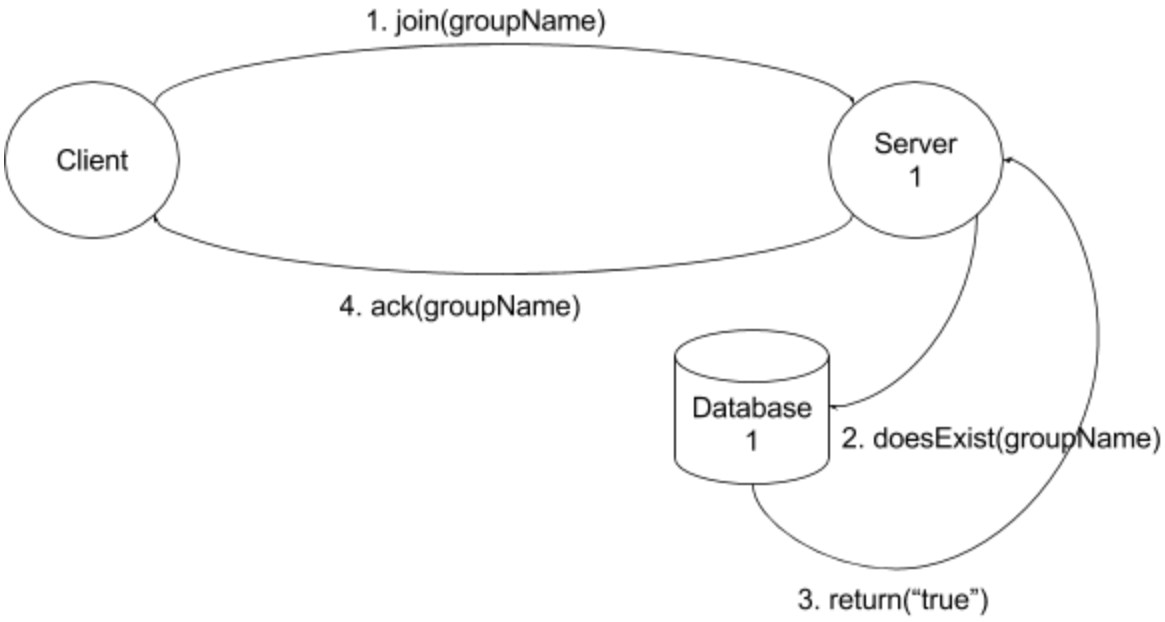
Client-to-Server:

Client employs UDP/IP to send messages (aimed at a particular chat room) using an ***add*** request to the server that will then place the message into a database using an ***insert*** command. Clients begin their interaction with a server using a ***login*** command that returns a token that is used to make sure that Client usernames remain unique. Tokens are lost after a period of inactivity which releases the username for others to use. Clients may use the ***create*** command to create a new chat room. ***Join*** on the other hand allows the user to join an existing chatroom. In both cases the server will either send an ***ack*** message to the client to confirm that the request has been received and accepted or an ***error*** message.

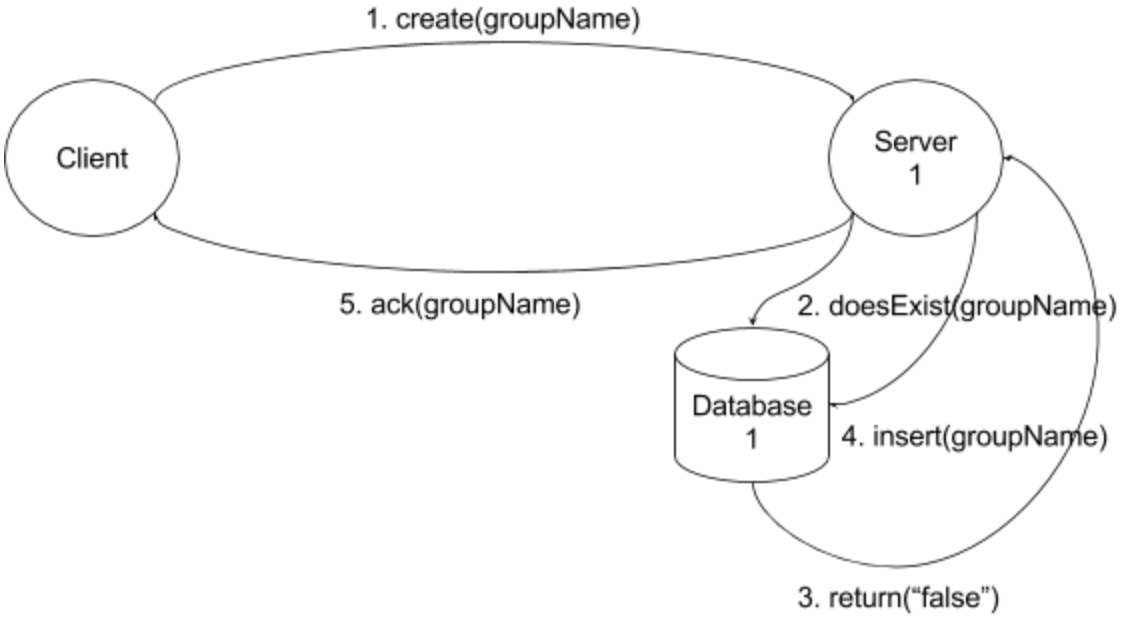
Client-To-Server:
Logging-In



Client-To-Server:
Join a Chat Room

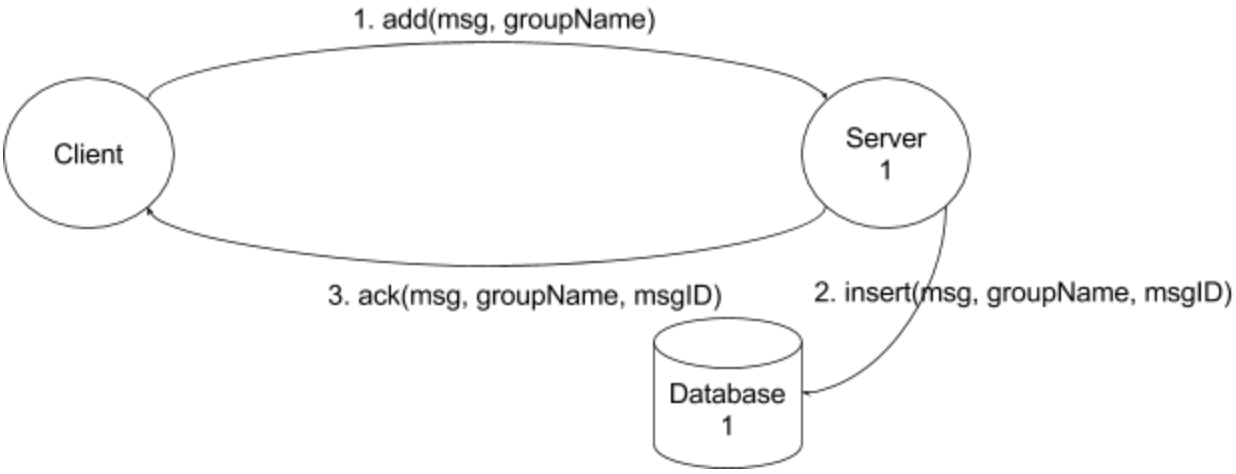


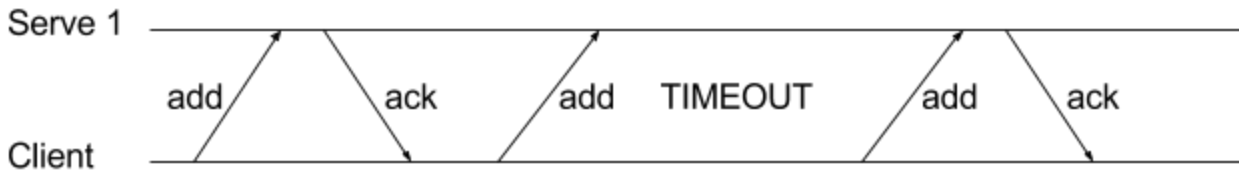
Client-To-Server:
Create a Chat Room



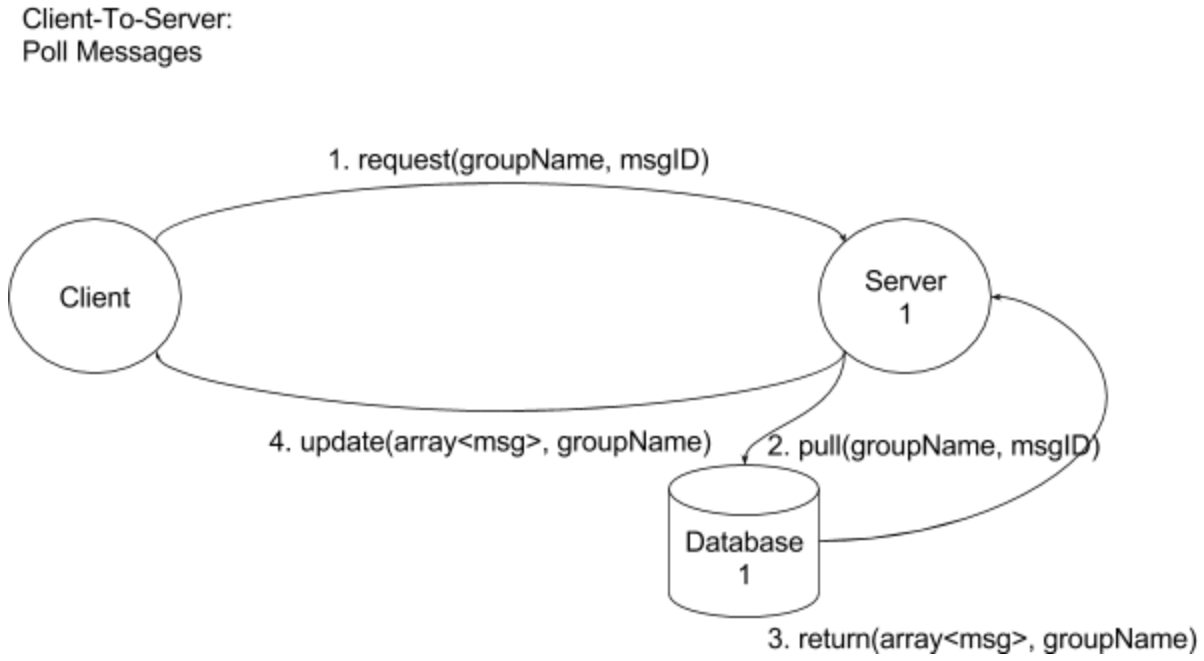
After a client sends an ***add*** request, an ***ack*** will be sent from the server to the client to show that the message has been received. If no ***ack*** has been received from the server the client will resend the ***add*** request again after a defined timeout period.

Client-To-Server:
Send Messages to a Chat room





Client will poll the server for new messages ***request*** and if there are any new messages then the server will send the messages to the client ***update***. ***msgID*** will be used to determine the state of the client in relation to the chatroom. So if Client has ***msgID*** = 4 in ***groupID*** = 1 while the database has ***msgID*** = 10 in ***groupID*** = 1, then we know that the Client is behind by 6 messages and will need to be updated. The polling is expected to be automated on a fixed time. It is not expected for the client to be responded to every poll.



Fault tolerance:

servers will simply not care if clients have fallen or have returned; their relationship is strictly transactional. Clients on the other hand will be provided a list of IP address of every server so if a client is unable to connect to one server they will simply move on to the next. If no server responds within a reasonable period of time, then the application will return an error statement in relation to a failure in connecting and re-attempt to communicate with the servers. Realistically getting the list of IP addresses would involve communicating with a DNS server, but for the purpose of this course the list of IP addresses may be hardcoded, provided in the command line, or via a text file (current implementation)

Client-To-Server Protocol:

<send> ::= <add> | <request> | <join> | <create> | <login>
<receive> ::= <update> | <ack> | <error>

<add> ::= “add”<newline><msg><groupName><newline>
<request> ::= “request”<newline><groupName><msgID><newline>
<join> ::= “connect”<newline><groupName><newline>
<create> ::= “open”<newline><groupName><newline>
<login> ::= “login”<newline><userName><newline>

<update> ::= “poll”<newline><msgArray><groupName><newline>
<ack> ::= “ok”<newline><ackMessage><newline>
<error> ::= “error”<newline><errorMessage><newline>

<groupName> ::= *String
<msgID> ::= *int
<msgArray> ::= array of <msg> and <msgID>
<msg> ::= *String
<ackMessage> ::= <msgAck> | <joinAck> | <createAck>
<msgAck> ::= <msg><groupName><msgID>
<joinAck> ::= <groupName>
<createAck> ::= <groupName>
<newline> ::= \r?\n
<errorMessage> ::= *String

The ***send*** format is for all messages sent from the client to the server while the ***receive*** format are for all messages sent from the server to the client.

H2 Database:

A separate browser based entity will serve as a database. Each server will have access to one database (meaning a one-to-one relationship).

H2 Database Fault Tolerance:

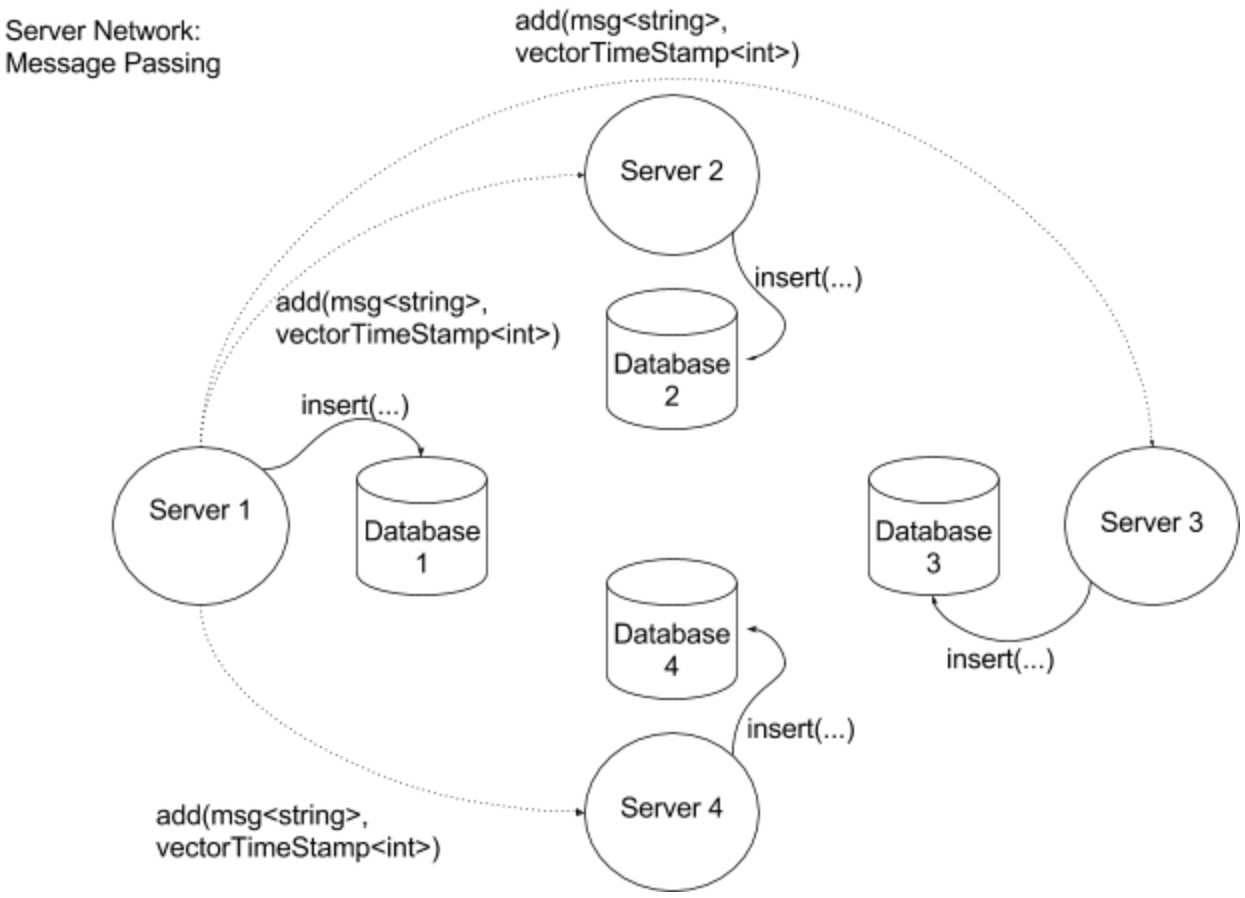
If a server falls, the database will sit idle and isolated until its server or a replacement connects to it. If the database falls, then the server connected to it will receive an exception that will trigger a self-termination as the server will only be capable of broadcasting messages, but no storage. This will allow clients to attempt to connect to a working server.

H2 Database Protocol:

There is no formal protocol for the H2 Database at this time.

Server Network (Active Replication):

Servers will act as peers in this group. Servers will receive messages from clients using the above Client-to-Server interaction. These messages will be assigned (by the client) to a particular chat room ***add(msg, groupName)***. The server will then add this message to its database. After, the server will IP-Multicast the message to its peers. The peers will then add the message to their respective databases. Employing IP-Multicast may result in some messages being lost, but this is considered an acceptable loss. Alternatively R-Multicast may be employed, but there is a significant bandwidth cost. As well, messages may not always appear in the same order i.e. server 1 may update its client with message 1 then message 2, but server 2 may update its client with message 2 then message 1 (causal ordering); this is considered acceptable.



Server Network Fault Tolerance:

If a server falls, the group will simply not care as clients will simply move on to another server and servers are never aware of each other. When a server returns it would learn that it has fallen behind when it receives a multicast. Here the server will request the sender of the datagram to pass along any messages it has missed.

Server Network Protocol:

<send> ::= <add> | <catchUp>
<receive> ::= <ack>

<add> ::= “add”<newline><msgArray><vectorTimeStamp><newline>
<catchUp> ::= “ask”<newline><vectorTimeStamp><newline>

<ack> ::= “ok”<newline><msgArray><vectorTimeStamp><newline>

<newline> ::= \r?\n
<msgArray> ::= array of <msg> and <msgID>

<msg> ::= *String
<msgID> ::= *int
<vectorTimeStamp> ::= vector of type int

The ***send*** format is employed by servers sending messages to the network or attempting to catch up after failure while the ***receive*** format is used by servers that help a lagged server.

Optional Features

- Encryption of messages through salts and hashes