Software System Lab - Project

FastChat

Chaitanya Garg (210050039)
Jennisha Agrawal (210050074)
Atharv Kshirsagar (210050025)

Telepound

Database

Database has 3 tables.

- clientinfo -

Coloumn name	data type	details
username	text	unique id
password	text	strong password
public_n	text	pubic key n value
public_e	text	pubic key e value
private_d	text	private key d value
private_p	text	private key p value
private_q	text	private key q value
salt	text	salt to encrypt keys

Database

- undelivered -

Coloumn name	data type	details
time	double precision	time of sending message
touser	text	user message is directed to
message	text	message to be sent

- groupinfo -

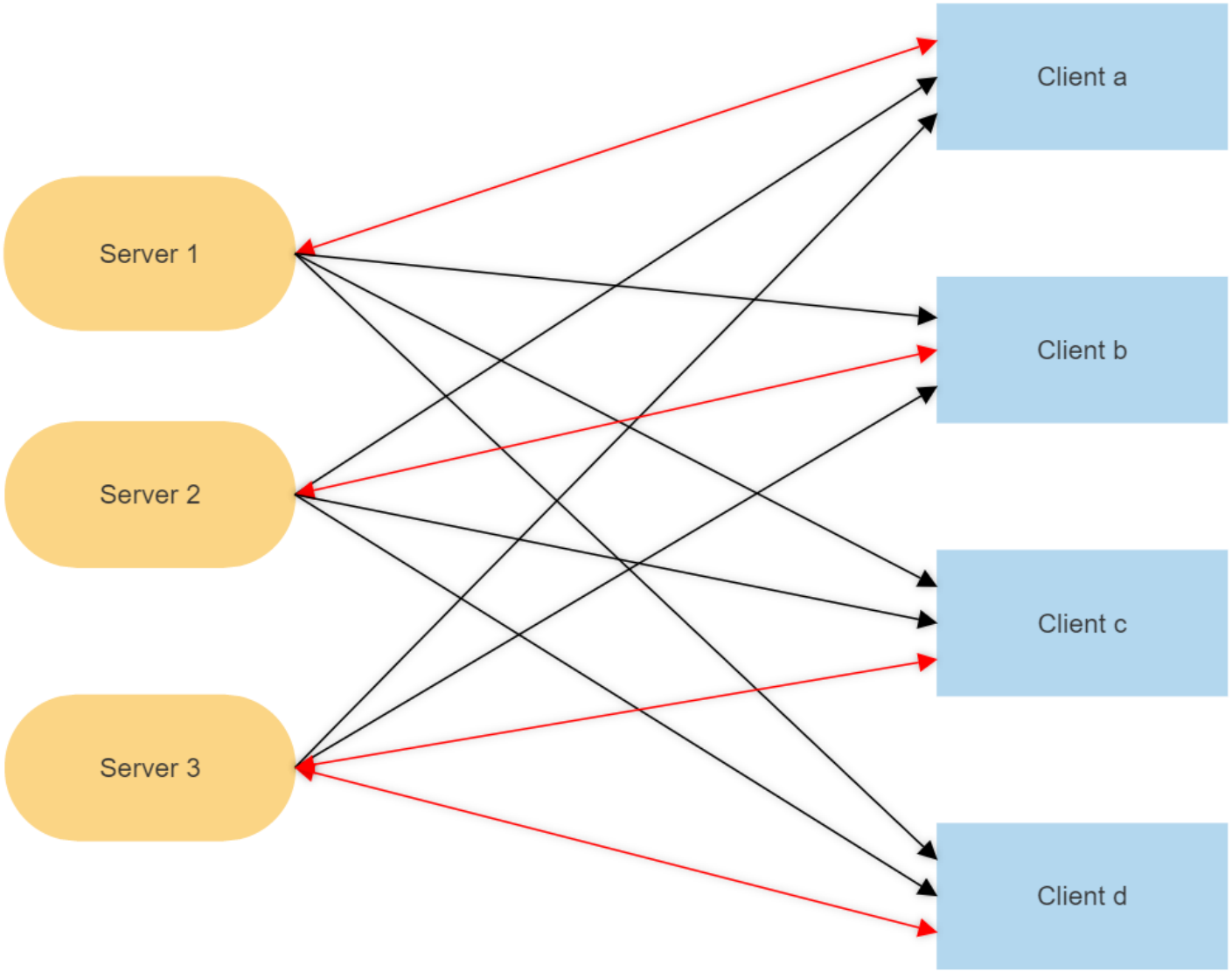
Coloumn name	data type	details
groupname	text	The name of group, unique id
admin	text	The username of group admin/creator
members	text[ ]	username of all group members

- Database has two roles :
  - Postgres(default) : Role for server and load balancer, has all priviledges.
  - Client : Role for client, can only select information.

Login and Sign-Up

- Authorization happens in client itself.
- Sign-Up
  - Checks if username already exits
  - If username is new then sign-up request request is sent to load balancer which stores new users in database
- login
  - Checks if given credentials are correct
  - Maximum 5 incorrect attempts are allowed
  - If correct login requests is sent to load balancer which stores new users in database
- Load balancer sends a assigned server and list of all servers to the client
- Client connects to all server

Multiple Servers



Load Balancing

- Load Balancer maintains a priority queue of servers based on some load factors
- While login/sign-up load balancer assigns server with least load factor at the time to the clients.
- Client recieve messages from all servers but can send message only to assigned server.
- After every fixed number of messages request to assign new server is sent to load balancer (number of messages can be varied to optimize performance)
- When new server connects , its information is sent to every client and they all connects to the server

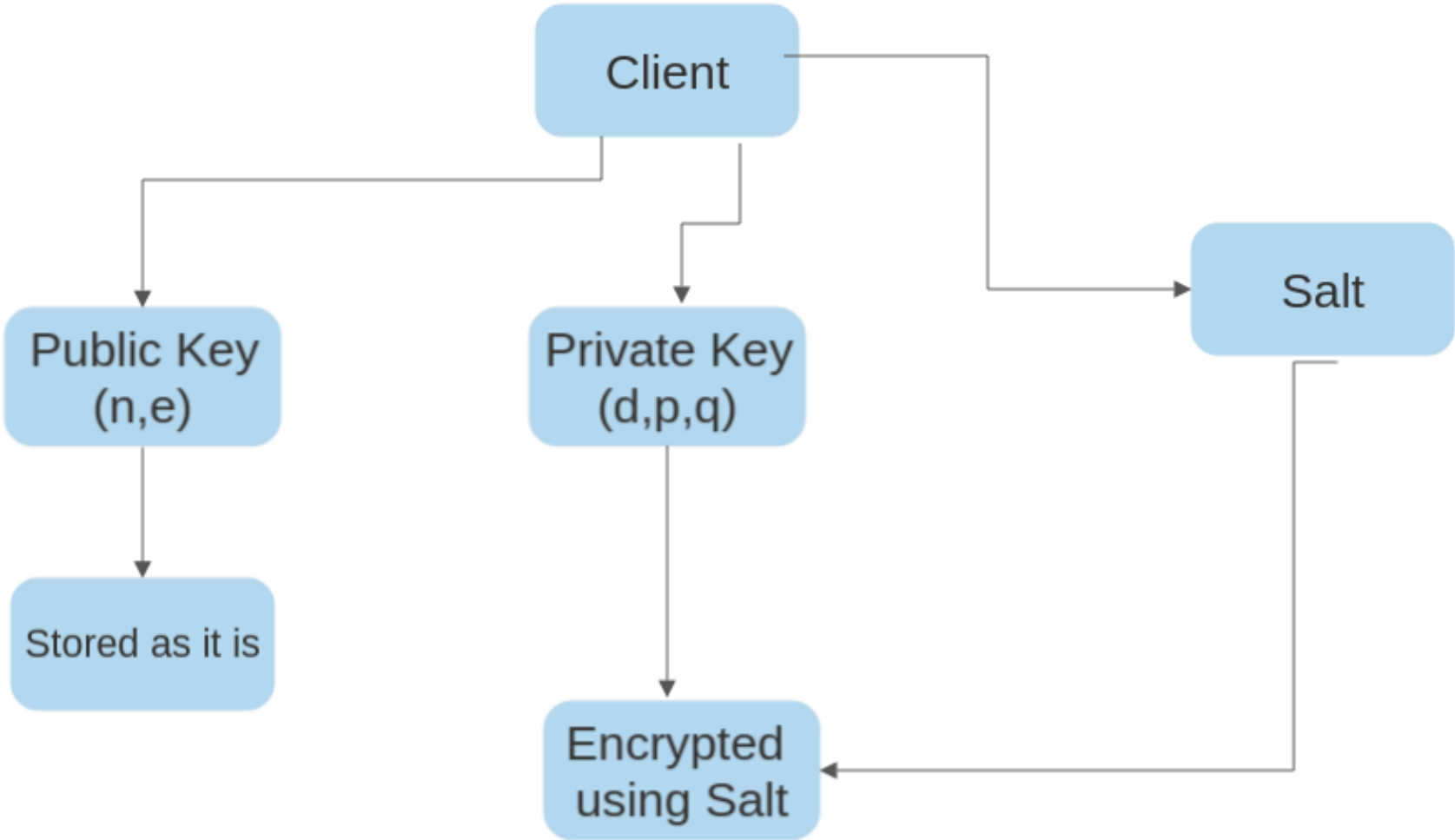
Sending Messages

- All communications is through json objects.
- Read receipt are sent to message sender.

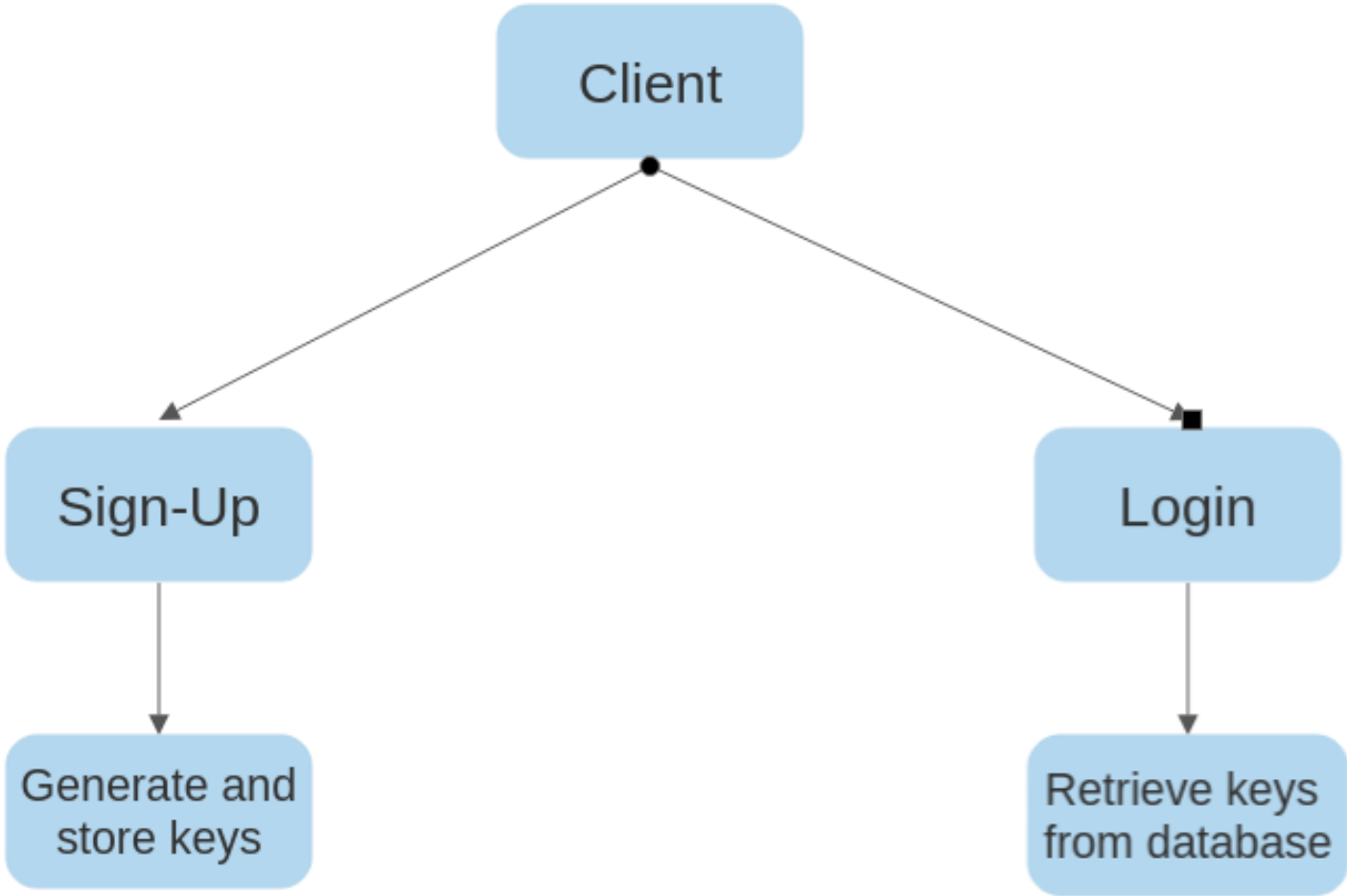
Storing messages for offline clients

- If the recipient is offline then the message is stored in the undelivered table of database.
- When a user logins all of its undelivered message are retrieved and sent to it.

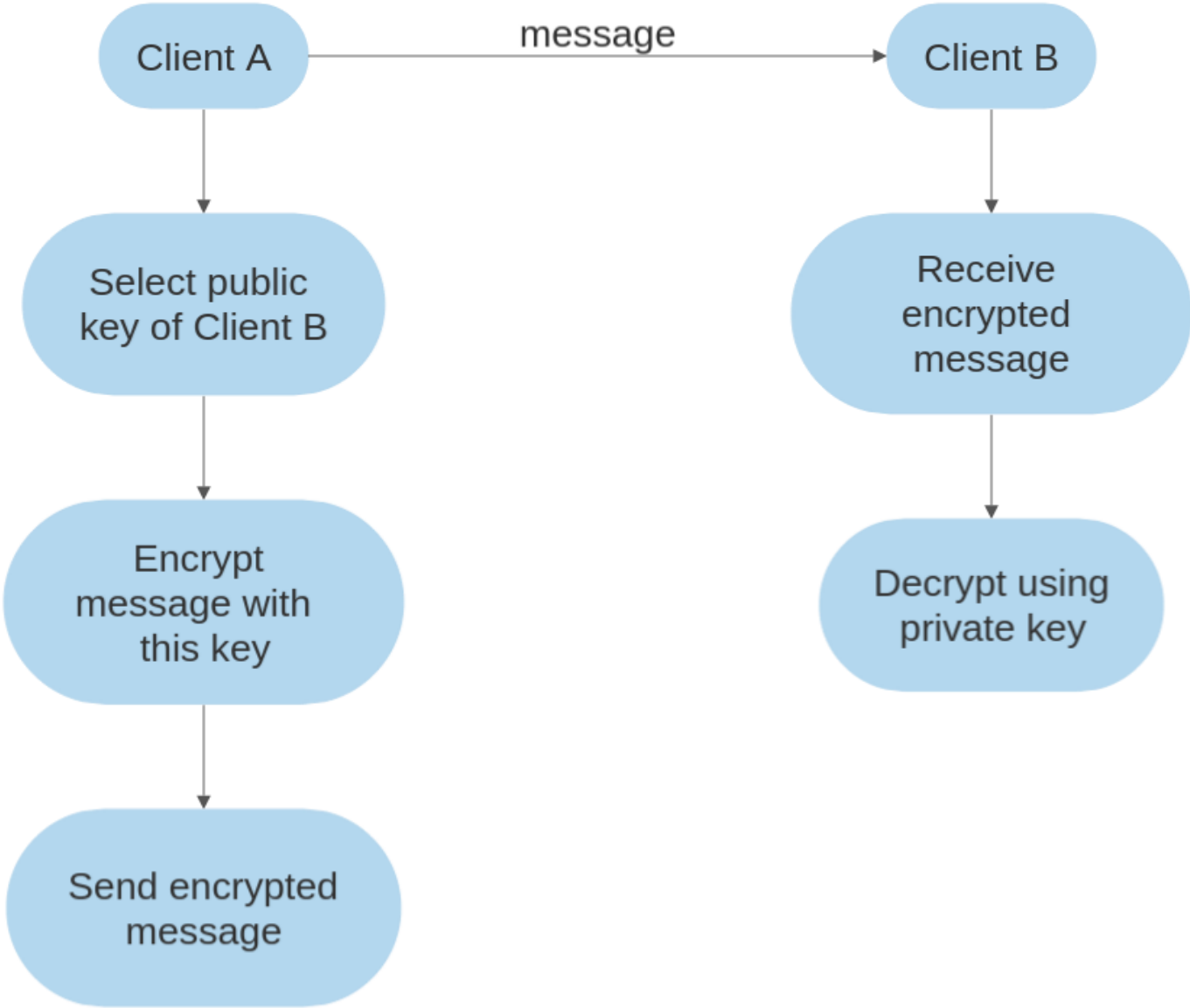
Encryption


Client
Public Key
(n,e)
Private Key
(d,p,q)
Salt
Stored as it is
Encrypted
using Salt

Encryption


Client
Sign-Up
Login
Generate and store keys
Retrieve keys from database

Encryption


Client A
message
Client B
Select public key of Client B
Receive encrypted message
Encrypt message with this key
Decrypt using private key
Send encrypted message

Group Chat

- Some commands are preserved as keywords, ('CREATE GROUP', 'ADD MEMBER', 'REMOVE MEMBER', 'LEAVE GROUP')
- The Group Name is also a unique id. Meaning username and groupname, combined, are unique.
- On Creating group, A group, with only 1 user is created and stored in the groupinfo table.
- This user is the admin, and he can add remove members by using ADD MEMBER, REMOVE MEMBER command.
- Group Messages are encrypted individually for all members.
- Any member can leave group using Leave Group command.
- If the group admin Leaves the group, or removes himself, then the group is deleted.

Performance Analysis

We have tried our best to complete this task but could not complete it.
We have uploaded the runner.py code to generate output logs, but it gives many errors which we were not able to debug in time.
It might be irrelevant, but we can see theoretically how this should perform.

- Assume there are n clients (say 10) and m servers (say 2).
- Statistically each server should have n/m (=5) clients to whom this server is assigned, so load on each server is O(n/m).
- The load balancer is called every k messages (say 5) to reassign server by each client, so its load should be O(n/k).
- Reassigning is only benificial if many clients go offline (because then only load on servers might reduce and become uneven).
- So based on client behavious, k an be set such that server reassigning is neither too frequent, nor do the servers have very varying loads.
- With increasing n, we need to only make n/m constant, then load on servers would remain unchanged, and k would need to be increased to make load on balancer small.

Performance Analysis

So for given number of servers, if the number of clients (n) increases, then best strategy would be to increase k with it.
This is because the average load on a server would definitely increase, but we would like to keep load on the load balancer under control as well. And Since the number of clients is large, statistically the leaving clients would be equally distributed, so across servers load would have little variation, so reassignment of servers can be less frequent.

If We can increase number of servers, even if memory and cpu usage of each server is a bit more limited, the result would be more favorable.
This is because in our implementation, most of the computational part is done in client itself, using client resources, so the main work on server is to manage channels of communications. So if we have more servers, the messages would be more distributed and would not have to wait less for other messages to be handled. This would of course have a optimal cpu and memory power, below which servers would become slow.

Performance Analysis

Note that this is mainly because most of the computational part and case handling is done on client side itself, so less resources at server are needed for computation.
In implementations were most computation is done at server, the strategy might change and we would like to give more resources to the servers to make the computations fast.