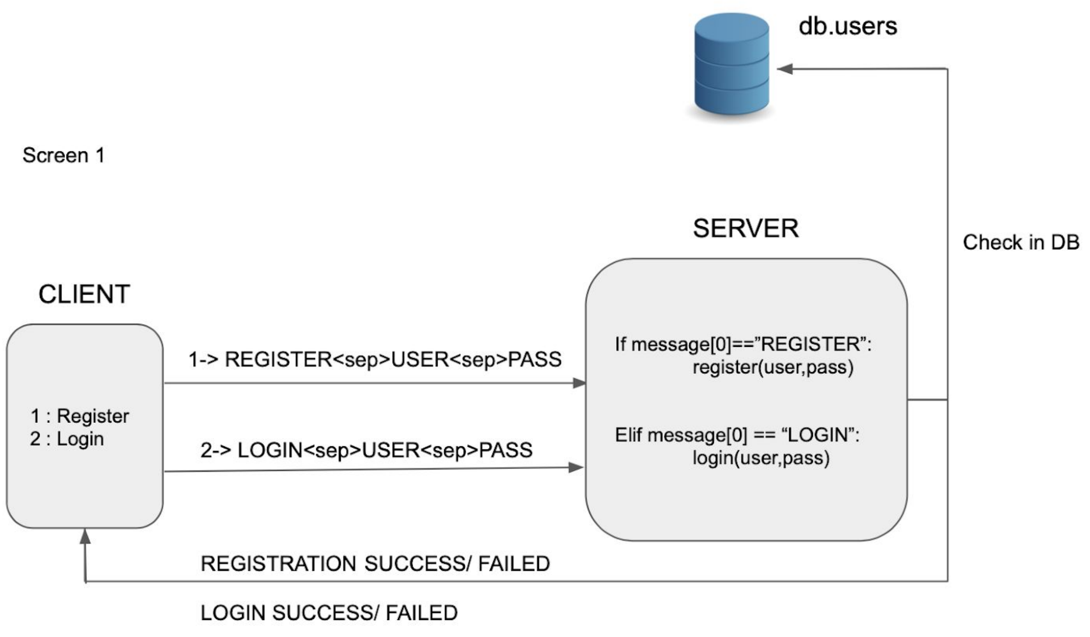
CS 433 COMPUTER NETWORKS PROJECT REPORT

Anubhav Jain (17110021)
Harshil Jain (17110060)
Rohit Patil (17110126)

Network Communication Paradigm


db.users
Screen 1
SERVER
Check in DB
CLIENT
1 : Register
2 : Login
1-> REGISTER<sep>USER<sep>PASS
2-> LOGIN<sep>USER<sep>PASS
If message[0]=="REGISTER": register(user,pass)
Elif message[0] == "LOGIN": login(user,pass)
REGISTRATION SUCCESS/ FAILED
LOGIN SUCCESS/ FAILED

Client Feature	Message to server	Server Functions	Server Response
Post Tweet	POST TWEET<SEP>"tweet"	post_tweet()=> Extracts hashtags saves tweet,hashtags,user in db.tweets	If success: TWEET POST SUCCESS Else: TWEET POST FAILED
Main feature: Show my profile ------------------------ Subfeature: 1 : Delete tweet	PROFILE --- 1 . {type:DEL	get_tweets() =>gets tweet from that particular user from db.tweets,sends them as a dictionary dump object	{ Tweet dictionary } ------------------------------------- If success: DEL TWEET

2 : Back	TWEET,id:tweet_id}
	
	2 . {type:BACK}
		-- 	
		del_tweet()=>deletes tweet from db.tweets
			SUCCESS
			Else:
			  DEL TWEET FAILED

Main feature: Show User feed ------------------------ Subfeature: 1: Retweet 2. Back	ALLUSERFEED -- 1 . {type:RETWEET,user:user,tweet:tweet} 2 . {type:BACK}	feed_display() => get all tweets from db.tweets and sends to client as dictionary dump object -------------------------------------- retweet()=> saves the tweet for given user in db.tweets	<user list> ----------------------------------- If success: RETWEET SUCCESS Else: RETWEET FAILED

Client Feature	Message to server	Server Functions	Server Response

Main feature:
 	
Search Tweets
----------------------- 	
Subfeatures:
1 : search and retweet
2 : show trending hashtags
	1 . SEARCH TWEETS <sep>text
	
	To retweet :
	{type:RETWEET,user:user,tweet:tweet}
	To get back :
	{type:BACK}
	-- 	
	2. "TRENDING HASHTAGS"
		search_tweets()
		--------------------------------------- 	
		
		retweet()
		
		--------------------------------------- 	
		
		get_trending_hashtags()
			{Search results dictionary}
			----------------------------------- 	
			RETWEET SUCCESS
			RETWEET FAILED
			----------------------------------- 	
			Hashtags as list dump

Main feature: Show followers ------------------------ Subfeature: Remove follower	“SHOW FOLLOWERS” -- “REMOVE FOL”	show_followers() ---------------------------- unfollow(): removes user from corresponding fields in db	<Followers results list> -------------------------------- REMOVE FOLLOWER SUCCESS REMOVE FOLLOWER FAILED
Main feature: Show followings ------------------------ Subfeature: Unfollow user	“SHOW FOLLOWINGS” -- “UNFOL USER”	show_followings() ---------------------------- unfollow(): removes user from corresponding fields in	<Followings results list> -------------------------------- REMOVE FOLLOWING

		db	SUCCESS REMOVE FOLLOWING FAILED

Client Feature	Message to server	Server Functions	Server Response
Main feature: Show all users Subfeature: follow / unfollow given user	SHOW USERS ------------------------------ ---------- {type:FOL USER, username} {type:UNFOL USER, username}	show_users(): returns list of all users except current with online/offline status ------------------------------ -------- follow() / unfollow() : Adds/removes from corresponding fields in db	<user list> -------------------- FOL SUCCESS FOL FAILED --------------------- UNFOL SUCCESS UNFOL FAILED

Explanation

1. **Post Tweet:** The client and server responses and the functions involved for the "Post Tweet" feature are as follows. The client initially requests to post the tweet to the server. The client thus sends the message *POST TWEET<SEP>"tweet"* to the server. The server function involved is the post_tweet() routine which extracts hashtags, saves the tweets and users in the database. The server finally sends the message *TWEET POST SUCCESS* if it is successful in storing the tweet in the database and *TWEET POST FAILED* if it is not successful.

2. **Show my Profile:** The client and server responses and the functions involved for the "Post Tweet" feature are as follows. This feature is responsible for showing all the tweets posted by the user till date. The server function involved is get_tweet() routine which gets the tweets from that particular user from db.tweets and sends them as a dictionary dump object to the client. The tweets are displayed and then there is an option to delete the tweet or go back to the previous screen. For deleting the tweet, the client sends a dictionary to the server of the form *{type:DEL TWEET,id:tweet_id}* and the function involved is the del_tweet() routine on the

server which looks for that particular tweet_id in the database and deletes it from the database. It sends a message *DEL TWEET SUCCESS* if it is successfully able to delete the tweet and *DEL TWEET FAIL* otherwise.

3. **Show User Feed:** The client and server responses and the functions involved for the "Show User Feed" feature are as follows. The feature shows the tweets tweeted by the users which the user follows arranged chronologically in descending order in time. The server function involved is feed_display() which gets all tweets from db.tweets and sends them to the client as a dictionary dump object. There are two options available here - retweet and to go back to the previous screen. For retweeting a tweet, the client sends a dictionary to the server of the form *{type:RETWEET,user: user,tweet:tweet}* and the function involved is retweet() routine on the server which saves the tweet for the given user under the flag RETWEET=True and the user from which it was tweeted. It sends a message *RETWEET SUCCESS* if it is successfully able to delete the tweet and *RETWEET FAIL* otherwise.

4. **Search Tweets:** The client and server responses and the functions involved for the "Search Tweets" feature are as follows. To search a tweet, the client sends the message to the server of the form *SEARCH TWEETS<sep> text*. The routine involved for searching the tweets at the server side is search_tweets() and it returns the results in the form of a dictionary to the client. There are two subfeatures available here, retweet and show top 5 trending hashtags. The retweet functionality works in the same way as discussed above. For showing the top 5 trending hashtags, the client sends the message *TRENDING HASHTAGS* to the server and the server uses the get_trending_hashtags() routine to search all the tweets posted by all users in the database and then find the count of the hashtags. Finally, the top 5 hashtags based on the count are sent to the client as a list dump and it displays it to the user.

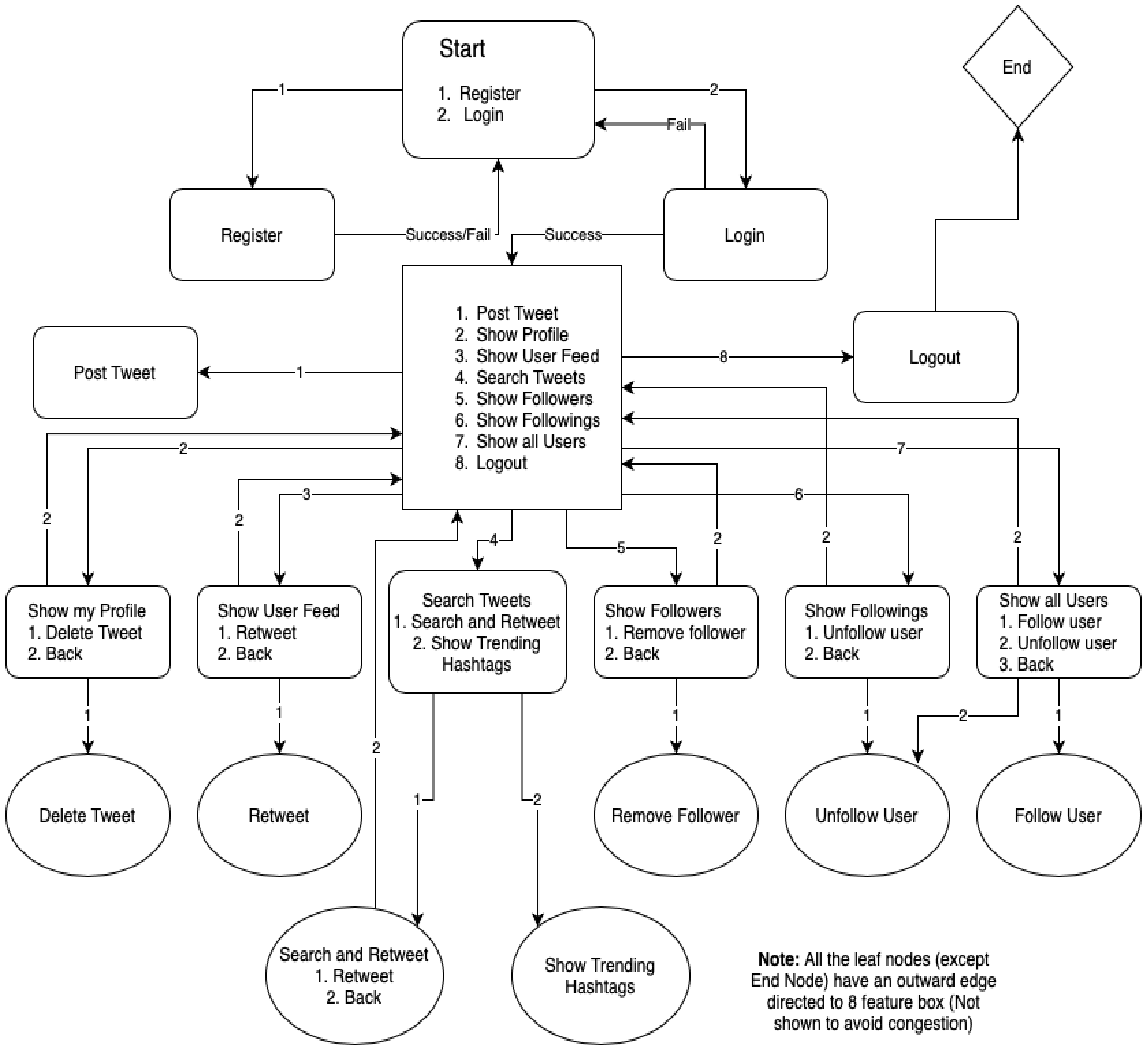
5. **Show Followers:** The client and server responses and the functions involved for the "Show Followers" feature are as follows. To search a tweet, the client sends the message to the server of the form *SHOW FOLLOWERS.* The

show_followers() routine at the server returns the list of followers to the client in the form of a list. There is an option to remove a follower and the corresponding message from the client to the server is *REMOVE FOL* which removes the user from the corresponding fields in the database. The server sends a message *REMOVE FOLLOWER SUCCESS* if it is successfully able to remove the followers and *REMOVE FOLLOWER FAILED* otherwise.

6. **Show Following:** The client and server responses and the functions involved for the “Show Following” feature are as follows. To search a tweet, the client sends the message to the server of the form *SHOW FOLLOWING.* The show_followings() routine at the server returns the list of followers to the client in the form of a list. There is an option to unfollow a user and the corresponding message from the client to the server is *UNFOL USER* which removes the user from the corresponding fields in the database. The server sends a message *REMOVE FOLLOWING SUCCESS* if it is successfully able to remove the followers and *REMOVE FOLLOWING FAILED* otherwise.

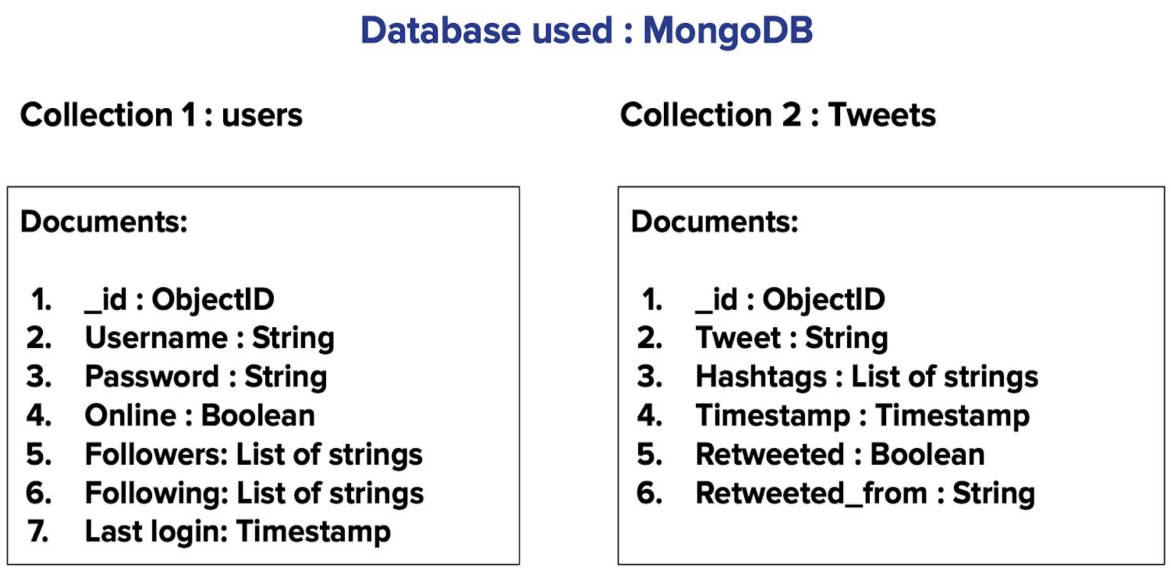
7. **Show all users:** The client and server responses and the functions involved for the “Show all users” feature are as follows. To show all users, the client sends a message *SHOW USERS* to the server. The show_users() routine at the server returns a list of all users except the current user who has loginned with the online/offline status. The server sends this list as a dump to the client which then shows it to the user. Here there are two subfeatures: following a user or unfollowing a user. For following a user the client sends a dictionary of the form *{type:FOL USER, username}* to the server and the server routine follow() adds the corresponding fields in the database and sends the message *FOL SUCCESS* if successful and *FOL FAILED* otherwise. For UNfollowing a user the client sends a dictionary of the form *{type:UNFOL USER, username}* to the server and the server routine follow() removes the corresponding fields in the database and sends the message *UNFOL SUCCESS* if successful and *UNFOL FAILED* otherwise.

Finite State Machine for Client/Server



Note: All the leaf nodes (except End Node) have an outward edge directed to 8 feature box (Not shown to avoid congestion)

Database Schema



Database used : MongoDB

Collection 1 : users

Documents:

1. _id : ObjectID
2. Username : String
3. Password : String
4. Online : Boolean
5. Followers: List of strings
6. Following: List of strings
7. Last login: Timestamp

Collection 2 : Tweets

Documents:

1. _id : ObjectID
2. Tweet : String
3. Hashtags : List of strings
4. Timestamp : Timestamp
5. Retweeted : Boolean
6. Retweeted_from : String

Feature Checklist

Basic Features

- ✓ Any Client/user is able to register and set up an account with Mini-Tweet.
- ✓ Client can login, get the updates and logout.
- ✓ Client can search for registered users, follow/unfollow any users and remove followers.
- ✓ The application supports users to post tweets, and categorize the tweets with specific hashtags.

Advanced Features

- ✓ Hashtags: Users are allowed to search and display tweets under specific hashtags. Show the Top 5 trendings hashtags.
- ✓ Can determine the list of active/online followers/followings.

- ✓ Retweet: Supporting users to use other users' tweets and post the retweets.
- ✓ Scaling to a concurrent server that can handle several client requests (Multithreading - A new thread is created for each client)

Security Features

- ✓ Users are able to authenticate with the server before trying to access any of the features.
- ✓ When a user is prompted for a Login password, the user input for the password is obscured/masked.

Dependencies

mongodb
mininet
python libraries : mininet, pymongo, stdiomask

Commands

Important instructions:
You have to create a database named ‘minitweet’ in mongodb
To do so:
run the script **db_initiate.py** to add some dummy users and tweets in the database.

To run the application on ubuntu terminal:

```
python server.py <ip>
python client.py <server_ip>
```

For example:
Mongodb should be running (use sudo mongod command)
In terminal 1: run

```
python server.py localhost
```

In terminal 2: run

```
python client.py localhost
```

To run application on mininet

1) Run mininet using -x flag (for Xterm terminals for each hosts)
2) Run mongod on the mininet terminal
3) Run server.py on h1 ‘s xterm terminal
4) Run client.py on h2’s xterm terminal

For example,
mininet -x
mininet > h1 mongod

On h1’s xterm terminal, (h1’s IP is by default 10.0.0.1)
python server.py 10.0.0.1

On h2’s xterm terminal,
python client.py 10.0.0.1

To run application on mininet using python script (Mininet library for python, taking stdin input for client using .txt files)

In client.py,
(stdiomask library has been used to hide entered password, this library does not work well with stdin input from .txt files)
Comment lines 52 and 74
Uncomment lines 53,54,75,76

```
cd testing
sudo python script_exp.py
```