Class 3, Group 13

Diogo Samuel Fernandes
Hugo Guimarães
Paulo Ribeiro
Telmo Baptista

Large Scale Distributed Systems

Supervised by
Carlos Baquero-Moreno
Pedro Ferreira Souto

Camellia Social Network

Peer-to-peer network using Kademlia

INDEX

Technologies

- Python
  - Simple Term Menu
  - Asyncio
  - ZeroMQ (with Tornado)
  - Kademlia
  - NTPLib
  - PyCryptodome

FUNCTIONALITIES

- Register / Login / Logout
  - Create a new account or login with an existing one, and logout from the system
- Follow / Unfollow
  - Subscribe or unsubscribe to another user's post, receiving its updates whenever a new post occurs
- Post Message
  - Publish a new message that becomes visible to all your followers
- View Timeline
  - See your timeline, composed of the posts of the users that you follow and your own posts
- View Profile
  - See your own profile, with information about the people you follow, your followers and how many posts you have made
- Recommendation System
  - Get suggestions of users based on people you follow

Kademlia Network

Kademlia is a distributed hash table for decentralized peer-to-peer computer networks.
In order to enter the network, a node must connect to a currently active node.

- Every node in our network can be considered a bootstrap node.
- It is assumed that our network always has at least 2 active nodes.
- The network stores information regarding its users, except their timeline and associated posts, which are stored locally.

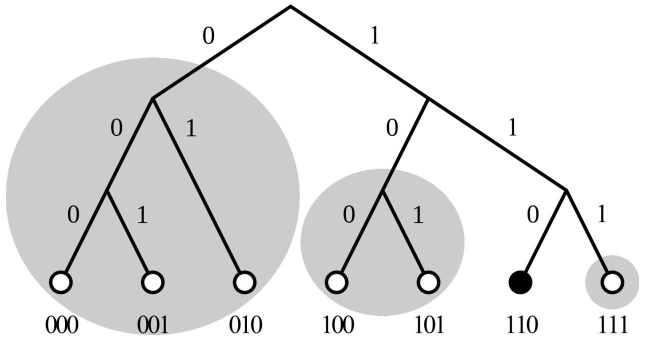


Fig. 1 - Example of a Kademlia Network

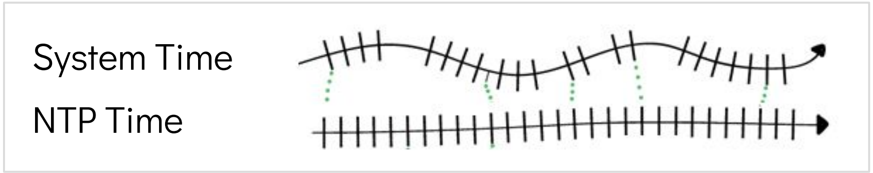
Information Stored on Kademlia

private	public	connections
username:private	**username:public**	**username:connections**
salt	followers	public_key_n
hash_password	following	public_key_e
		listening_ip
		listening_port

Order of Posts

Since Camellia is a Timeline App, we need to ensure that posts follow an order. At first, we started to search for ways to do this, like vector clocks and interval tree clocks. However, these alternatives had too much complexity without a significant benefit because posts that are replies to another will have a big delay between them.

To ensure the order of posts, Twitter uses SnowflakeID. The first 41 bits of this ID are a timestamp. The following bits represent a machine ID, being followed by the sequence number of message, to allow creation of multiple messages in the same millisecond.



Network Time Protocol was used to synchronize the clocks of each peer once an hour and thus maintain a more accurate order of posts.

Send and receive posts

Direct communication is needed in the following situations:

- When a user follows another, the followed user must send its posts to the new follower.
- When the followed user is offline, the new follower must try to reach his followers in order to fetch its posts.
- When a new post is made, it is sent to all the post author's followers.
- After rejoining the network, a user must send all its posts to their followers.

This communication was implemented using ZeroMQ sockets, in a PUSH-PULL pattern, so that every time a message is published, the follower receives it without having to perform any request.
ZeroMQ Tornado was used to provide non-blocking I/O.

Send and Receive Posts

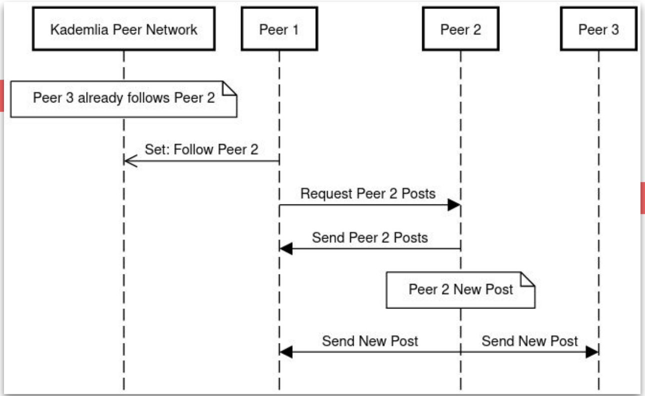


Fig. 2 - Normal flow off a follow event

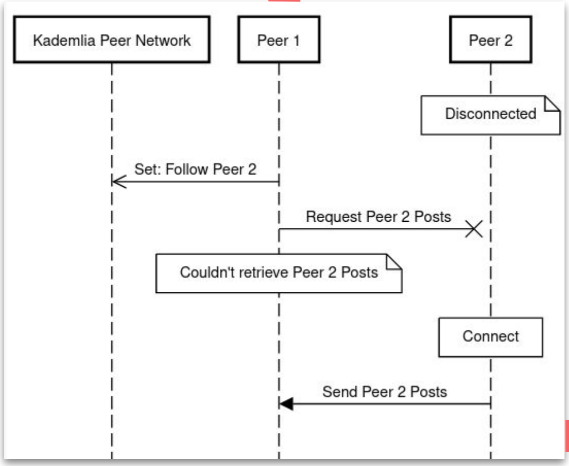


Fig. 3 - Peers send updated timeline to its followers on connection

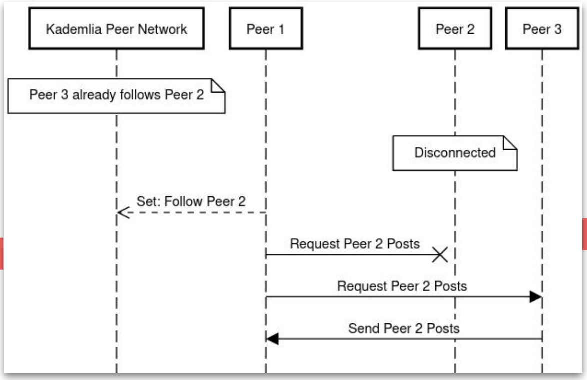


Fig. 4 - Request is resent to a follower of the request's original target

Handling disconnected users

When a user rejoins the network, a request of the posts from the followed users is sent:

- First, it tries to reach the followed user. If they are online, the posts are sent successfully through a direct connection.
- If the followed user is offline, the request starts being sent to each of the other followers of the followed user, until one of them retrieves the required posts.

The same strategy is applied when a user follows another and immediately requests their posts to update their timeline. After rejoining, they also send their posts to each of their followers:

- This way, we guarantee that the followers are always updated, even if the previous strategy failed (for example, when the followed user is offline and has no other followers that we could reach).

Garbage Collector

In order to remove old messages that have lost interest and are no longer relevant to the user experience of the timeline, a garbage collector was implemented.

- The garbage collector allows the removal of posts that have been published over a certain period of time (post lifespan)
- The default value used for the user post lifespan is 3 minutes (value used for testing purposes).
- The lifespan of a post can be changed by each user in the config.ini file or even disabled, located at the root of the project.
- The garbage collector is called whenever a user logs into its account, and every minute afterwards.
- Own posts are always kept, the garbage collector will only remove posts from other users, this is done as we need to keep at least one copy of each post in case of a request to view old posts.

SECURITY

In order to guarantee a secure network, some safety measures were implemented:

- Passwords are hashed using a random salt. Although these fields are public because they are stored in Kademlia, using salt is a powerful tool since passwords like “123456” or “password” aren’t immediately revealed when a hashed password is identified.
- Upon account creation, every user is given a private and public key. The public key will be stored on the Kademlia network as seen before, and the private key will be stored inside a folder, on the user machine. The user will use the private key to sign every message sent through a direct connection, ensuring the integrity of the message.

Conclusion and Future Work

During the development of this peer to peer network, we learnt how to build a reliable large scale network timeline.

Due to time constraints, we couldn't add more features to this network. However, for future work, we plan to add more restrictions to the namespaces. This is, each namespace could have a different scope. The namespace *private* could be encrypted and then it will be only accessed by that user. Furthermore, only the user should have write access on the *connections* table, but everyone should have read access to it.

The recommendation system can also be improved by having two types of recommendation, in which the first one to be described is the one implemented:

- Users that you might know (recommend followers of the users you follow). Appropriate for suggesting users inside the same social circle.
- Users that might interest you (recommend users whose followers share some followed users with you). Appropriate for suggesting interests rather than social connections.

Additionally, an improvement can also be done to provide recommendations even for users that do not follow anyone yet, for example, when those users just created an account. This could be done by recommending the most popular users or implementing recommendations using geolocation.

References

- Accessed December 2021. https://github.com/bmuller/kademlia
- Accessed December 2021. https://github.com/cf-natali/ntplib/
- Accessed December 2021. https://zeromq.org/
- Accessed December 2021. https://en.wikipedia.org/wiki/Snowflake_ID
- Accessed January 2021. https://pycryptodome.readthedocs.io/
- Petar Maymounkov and David Mazières 2002. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. https://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf