Decentralized Timeline

Python

Bernardo Ferreira (up201806581@fe.up.pt)

João Rodrigo (up201705110@fe.up.pt)

Telmo Botelho (up201806821@fe.up.pt)

Maria Francisca Almeida (up201806398@fe.up.pt)

Classes

- Peer :
  - ip
  - port
  - loop
  - PeerInfo()
  - myMessages (user posts)
  - timelineMessages (all posts)
- InitiatorPeer :
  - server
  - port
- Messages:
  - id
  - author
  - content
  - ~~date~~
- PeerInfo
  - ip
  - port
  - followers
  - following
  - last_post_id
- Listener
  - ip
  - port
  - peer

kademlia

Kademlia is a distributed hash table for decentralized peer-to-peer computer networks designed by Petar Maymounkov and David Mazières in 2002.

The Kademlia network is made up of a wide range of nodes, which interact with each other through User Datagram Protocol (UDP). Each node on the network is identified by a unique binary number called node ID. The node ID is used to locate values (block of data) in the Kademlia algorithm.

In this project we use this system for the peers to identify other peers and be able to follow and unfollow them. The peer gets the other peer information(ip, port) to send him a message and tell him that he follow him.

Asyncio

- Asyncio is a python library that is used to write concurrent code.
- We used this library to run the necessary threads and make peers send messages to other peers.
- We use the function open_connection from asyncio to send messages to peers.
- Our peer has two threads, the run function that executes the menu and his functionalities and a listener that receives the messages.

Files used for storage

- */users* directory

  In order to keep user’s data we need to save peer’s information locally. When a new peer is created a file with name equal to his username is created to save his information as json. Each time the peer information changes the file is updated.

- *usernames.txt*

  This file identifies which users are online on the network.

Peer to peer communication (TCP)

Message format

“*” means the key is always used

{***operation** : follow, unfollow, get, getResponse; ***content** : __ ; ***ip** : _ ; ***port**: _ ; username: _ }

EXAMPLES (self - Peer()) :

{"**operation**" : "follow", "**content**": PeerInfo() as dictionary, "**ip**" : self.ip, "**port**": self.port, "**username**": self.username}

{"**operation**" : "unfollow", "**content**": self.username, "**ip**" : self.ip, "**port**": self.port}

{"**operation**" : "get", "**content**": message_ID or -1, "**ip**" : self.ip, "**port**": self.port}

{"**operation**" : "getResponse", "**content**": Message() as dictionary, "**ip**" : self.ip, "**port**": self.port}

Operations

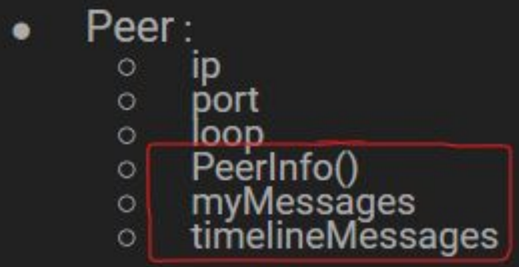
- Online
    - When the node start's its activity his username is added to the usernames.txt file.
- Offline
    - When the user quits his username is deleted from usernames.txt file.
- Follow
    - User A chooses to follow User B from a list of users. User A sends a message with operation = *follow* to user B. User A adds User B to following list, User B adds User A to followers list.
- Unfollow
    - Being that User A follows user B he can choose to unfollow. User A sends a message with operation = *unfollow* to user B. User A removes User B from following list, User B removes User A from followers list.
- Refresh timeline
    - Given that a user follows some people, he needs to refresh his timeline in order to get the latest posts. By choosing that option he sends a message with operation = *get* to his following that answer with a message with operation = *getResponse* and in the content a list of posts which the initial user adds to his timeline.
    - On sending the 'get message' the user checks if he has any post from that someone he follows, and if he does in the message content he sends the id of the most recent message received; if he has no post the id value is -1, which means send all messages

Fault tolerance

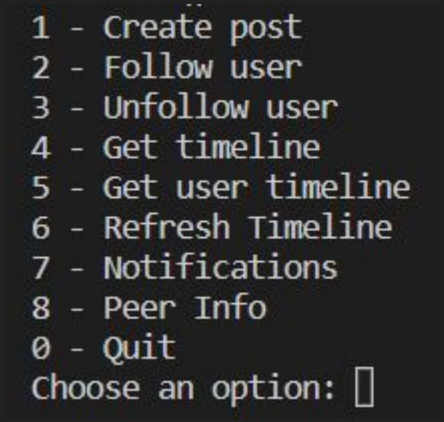
Each time there's a change the peer saves his PeerInfo() plus his messages.

On the event that a peer exits unexpectedly, upon relaunching the peer reads the file were information was saved and loads his previous state.

Information is saved to a .txt file (in /users folder) named after user's username and information is saved using Json.



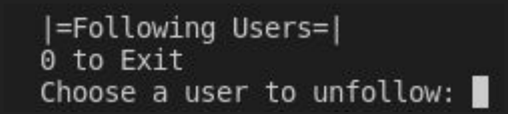
- Peer :
  - ip
  - port
  - loop
  - PeerInfo()
  - myMessages
  - timelineMessages



```
1 - Create post
2 - Follow user
3 - Unfollow user
4 - Get timeline
5 - Get user timeline
6 - Refresh Timeline
7 - Notifications
8 - Peer Info
0 - Quit
Choose an option: 
```

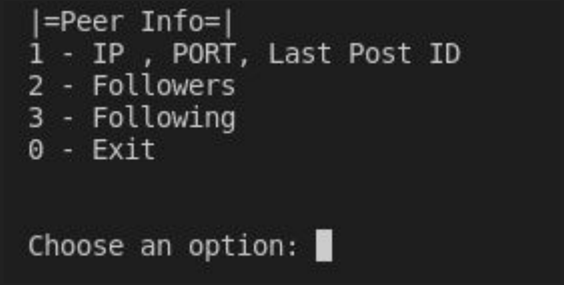
Interface

Main menu that allows the user to use the main system' features.



```
|=Following Users=|
0 to Exit
Choose a user to unfollow: 
```

This page has all the users the current user follows. Here the user can unfollow those users.

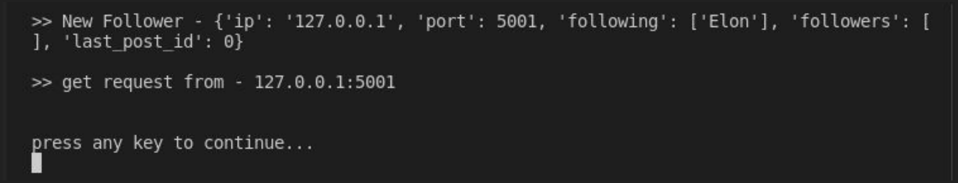


```
|=Peer Info=|
1 - IP , PORT, Last Post ID
2 - Followers
3 - Following
0 - Exit


Choose an option: 
```

Options where a user can see his own information.

Interface

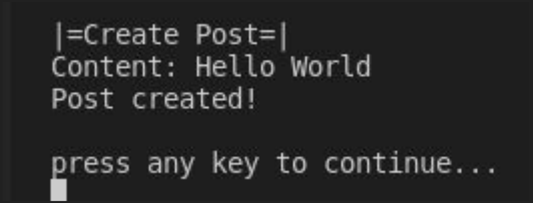


```
>> New Follower - {'ip': '127.0.0.1', 'port': 5001, 'following': ['Elon'], 'followers': [
], 'last_post_id': 0}

>> get request from - 127.0.0.1:5001


press any key to continue...
```

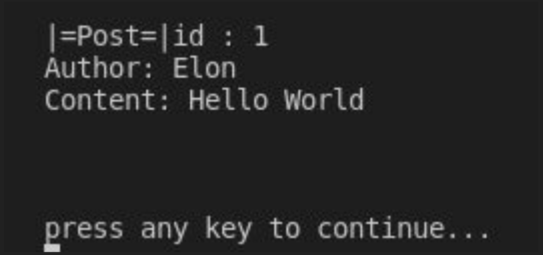
This is the notifications page. Here the user can see all of his notifications. For example when someone follows him, he receives a notification that is save in this page.



```
|=Create Post=|
Content: Hello World
Post created!

press any key to continue...
```

In this page the user can create a post.



```
|=Post=|id : 1
Author: Elon
Content: Hello World



press any key to continue...
```

Here the user can see the timeline.

Workflow (video)

Conclusion

- We believe that we achieve the purpose of this project in creating a decentralized peer to peer system.
- For future work we could do a more appealing interface and add more functionalities. We thought of automatic post receiving, a way to repost old posts, and a login and register.

References

- System Design For Large Scale - powerpoint from the theoretical classes
- https://pypi.org/project/kademlia/
- https://pythonhosted.org/kademlia/intro.html
- https://kademlia.readthedocs.io/en/latest/source/modules.html
- https://docs.python.org/3/library/asyncio.html