DECENTRALIZED TIMELINE

Large Scale Distributed Systems
FEUP

José Rocha - up201806371@edu.fe.up.pt
Miguel Silva - up201806388@edu.fe.up.pt
Pedro Ferreira - up201806506@edu.fe.up.pt
Pedro Ponte - up201809694@edu.fe.up.pt

TABLE OF CONTENTS

01
INTRODUCTION

INTRODUCTION

- **Peer-to-peer**
- Each node in the network is represented by a **User**
- A User can:
  - **Register / Login / Logout**
  - **Follow / Unfollow** other Users
  - **Publish** Messages
  - View Timeline
  - Check who follows and who is following
- Nodes help in **storing** and **forwarding** content within the network
- Content is available at all time, as long as the Bootstrap Peers are available

02

TECHNOLOGY

Pedro F.

TECHNOLOGY

- **Python**
  - Kademlia
  - Asyncio

initial_peers.py

> python3 initial_peers.py

peer.py

> python3 peer.py <port>

```
============================== Authentication ==============================
| 1: Login                                                                  |
| 2: Register                                                               |
| 3: Close the Application                                                  |
========================================================================
```

```
============================== Welcome teste ===============================
| 1: Search for an User                                                     |
| 2: Show Timeline                                                          |
| 3: Post a Message                                                         |
| 4: Check My Followers                                                     |
| 5: Check Who I'm Following                                                |
| 6: Logout                                                                 |
========================================================================
>
```

03

ARCHITECTURE

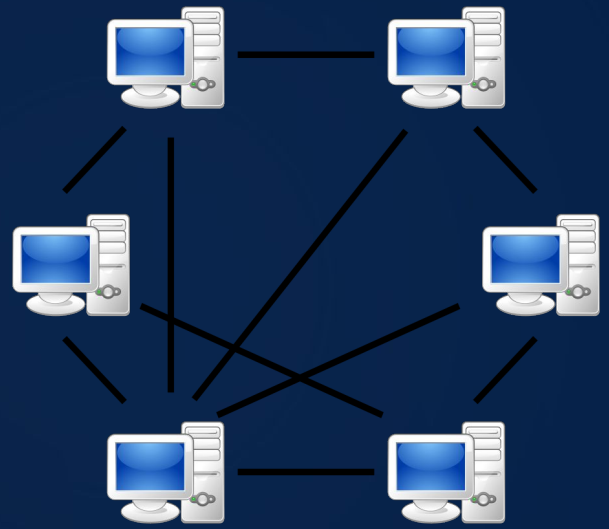
ARCHITECTURE

- **Decentralized**
  - Kademlia's DHT
- Network only stores data concerning User's credentials
- Messages are stored locally in each Peer
- Forwarding of messages has a time complexity of O(log n)

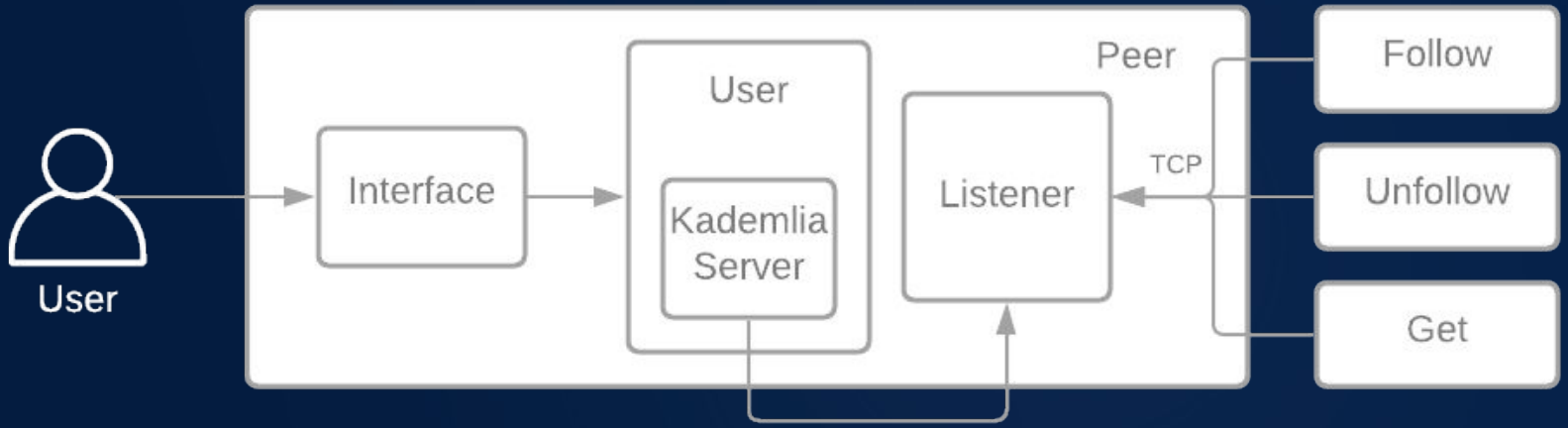
Key	Value
followers	Username's list of followers
following	Username's list of following
port	User's port for establishing connections
notifications	Username's list for follow/ unfollow notifications
online	Boolean true stating whether User is online (or not)

ARCHITECTURE

- **3 Bootstrap Nodes** - used for introducing other nodes into the network
- **Connections**
  - UDP - used by Kademlia for establishing the network
  - TCP - used for follow/unfollow and retrieval of messages between Peers



ARCHITECTURE



04

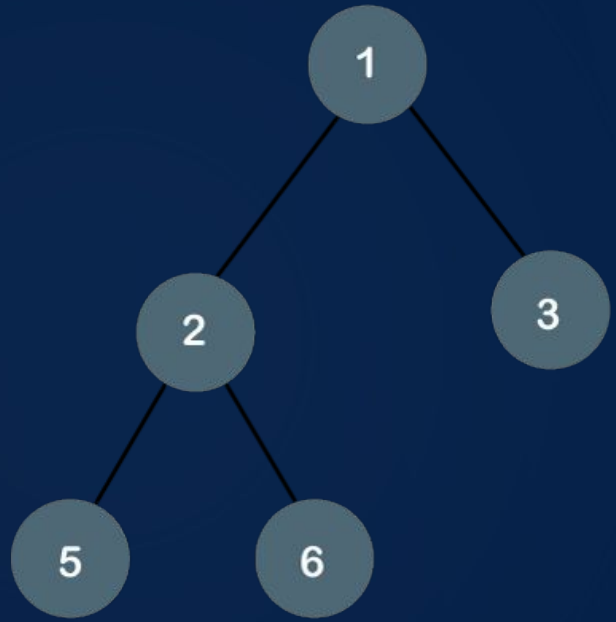
IMPLEMENTATION

MESSAGE DISSEMINATION

- Each protocol requires a connection between Peers
- Follow / Unfollow only involve two Peers
- Message dissemination involve an undefined number of Peers
    - Problems concerning scalability
    - For example, a Peer with 1000 followers needs to establish 1000 connections in order to send messages
    - Inefficient
    - How do we solve this?

MESSAGE DISSEMINATION (cont.)

- **Binary Tree Approach**
  - Limit the amount of connections
- Dissemination is done in a hierarchical way
- Root Node decides the hierarchy
- Forwarding of messages has a time complexity of O(log n) instead of O(n)
- Possibility of overloading one Node is removed

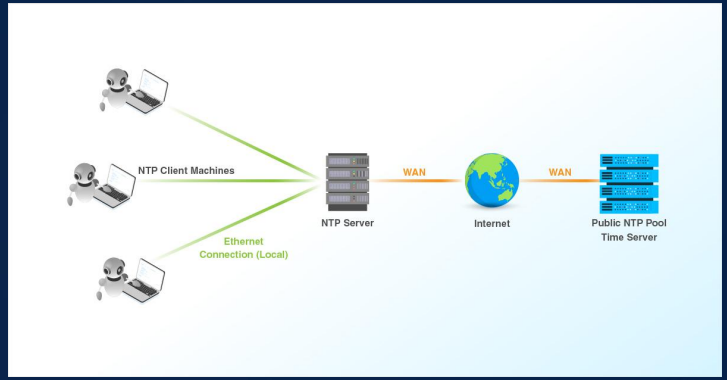


MESSAGE DISSEMINATION (cont.)

- Received Messages can be of three different types:
  - **Follow** - informs the User there is a new follower
  - **Unfollow** - informs the User that someone unfollowed them
  - **Get** - informs the User he needs to reply with his local timeline
- Posted messages are stored locally
  - Sent to Users following them whenever necessary

CLOCK SYNCHRONIZATION

- As we are working on a Timeline App, we need to ensure that posts follow an order
- Initially, we are using time() from Python time library
- The main problem is if peers clocks are not synchronized
- To solve that, we use Network Time Protocol (NTP) to maintain a more accurate order of posts



GARBAGE COLLECTOR

In order to remove data that is no longer useful / relevant for user experience, we have implemented a garbage collector mechanism.

- Runs whenever a user logs in to his account and then periodically
- Applied for all messages: own and stored belonging to other users
- The time the message was published is compared to the current time
- Remove older posts when their lifespan surpasses 5 minutes

WHAT IF SOMETHING GOES WRONG?

FAULT TOLERANCE

- Let's imagine this scenario: **Peer A** tries to establish a connection with **Peer B**
- **Peer B** crashed / is offline
- What happens?
  - **Peer A** retries to establish the connection up to a total of 3 tries
  - Follow / Unfollow - **Peer A** updates **Peer B**'s registry on the network
  - **Peer B** is flagged as being offline
  - The *notifications* field in **Peer B**'s registry is updated
  - When **Peer B** reconnects he becomes aware of what happened during their absence
- **But what about message dissemination?**

FAULT TOLERANCE (cont.)

- Peers store both own published messages as well as other Peer's received message
- When disseminating messages both sets of messages are sent
  - Duplicate messages are discarded
- Hierarchy is done based on online Peers
  - If a Peer hadn't been flagged as offline mid-way through the procedure, their registry is updated and the process of deciding the hierarchy is redone entirely

SERIALIZATION

Ensures the system was capable of dealing with all kinds of problems and crashes throughout its usage with no information loss.

- Runs periodically while the user is online
- If the user has posted messages, they are saved in a *.dat* file
- When the user logs in again, reads this file so he can access his messages (deserialization)

05

CONCLUSION AND FUTURE WORK

Conclusion and Future Work

During the development of this project, we learned how to create a reliable decentralized timeline service.

The main functionalities of the system were implemented. However, due to time constraints, we couldn't add more features we would like to have:

- Recommendation system, to suggest new users to follow
- Password and public / private key system for each user to increment the security

06

REFERENCES

REFERENCES

- https://kademlia.readthedocs.io/en/latest/
- https://github.com/bmuller/kademlia
- https://docs.python.org/3/library/asyncio.html
- https://docs.python.org/3/library/asyncio-stream.html
- https://pypi.org/project/ntplib/
- https://docs.python.org/3/library/datetime.html