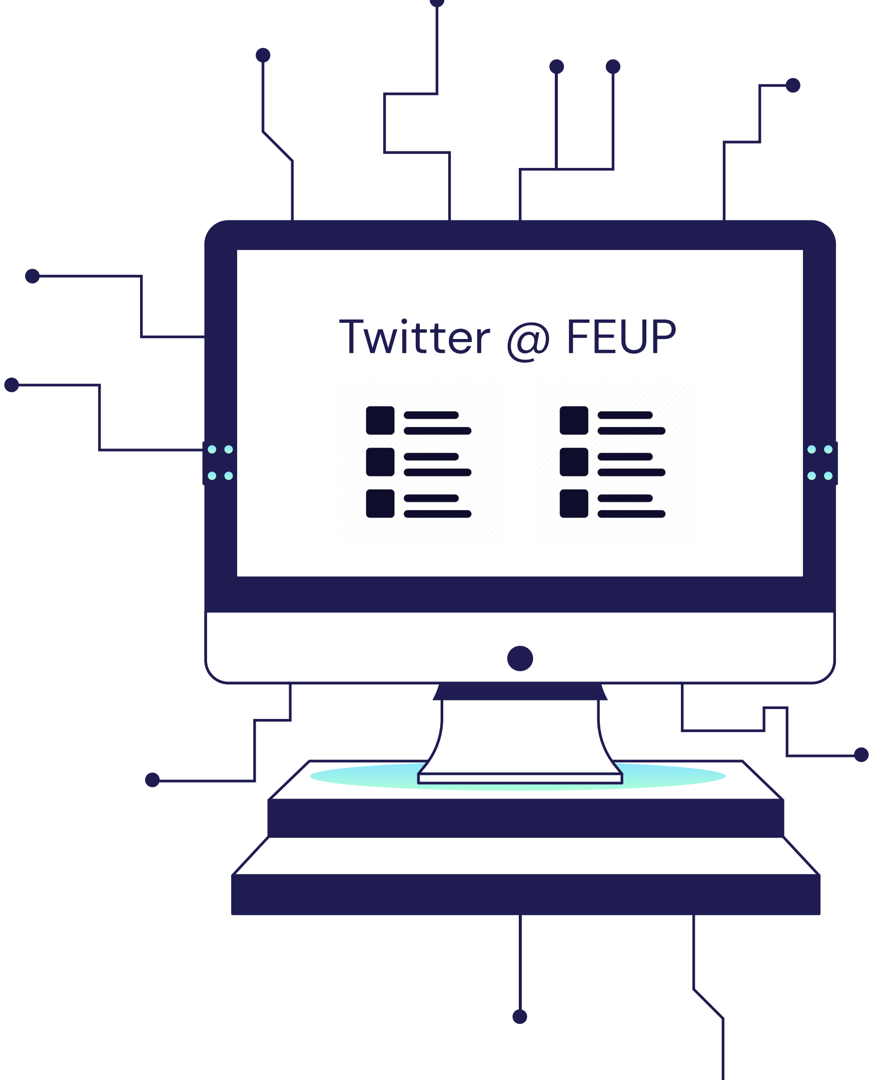


Decentralized Timeline

Large Scale Distributed Systems
FEUP

01 Project Overview

Decentralized Timeline Service

02 Technology

Python, Kademlia and Shell implementation. How to run our project

03 Architecture

Network architecture
Decentralized Hash Table system

04 Implementation

Authentication, Timeline and Post messages

05 Demonstration Video

Short demonstration video of all features implemented

06 Questions

Any Questions?

PROJECT OVERVIEW

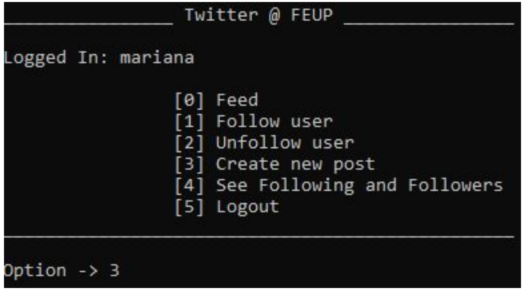
Decentralized Timeline Service

- **Peer-to-peer** and edge devices
- **Users** have an identity, they can **Login** or **Register**
- Users **Post** small text messages to the network
- They can **Follow** and **Unfollow** other users to get their messages
- Remote content is available at all times, as long as there's at least one peer online in the network (bootstrap peer)
- Information from subscribed sources can be ephemeral and only stored and forwarded for a given time period

TECHNOLOGY

Python
Kademlia
Asyncio

Interface



```
_______________ Twitter @ FEUP _______________

Logged In: mariana

               [0] Feed
               [1] Follow user
               [2] Unfollow user
               [3] Create new post
               [4] See Following and Followers
               [5] Logout

_____________________________________________

Option -> 3
```

Run our project

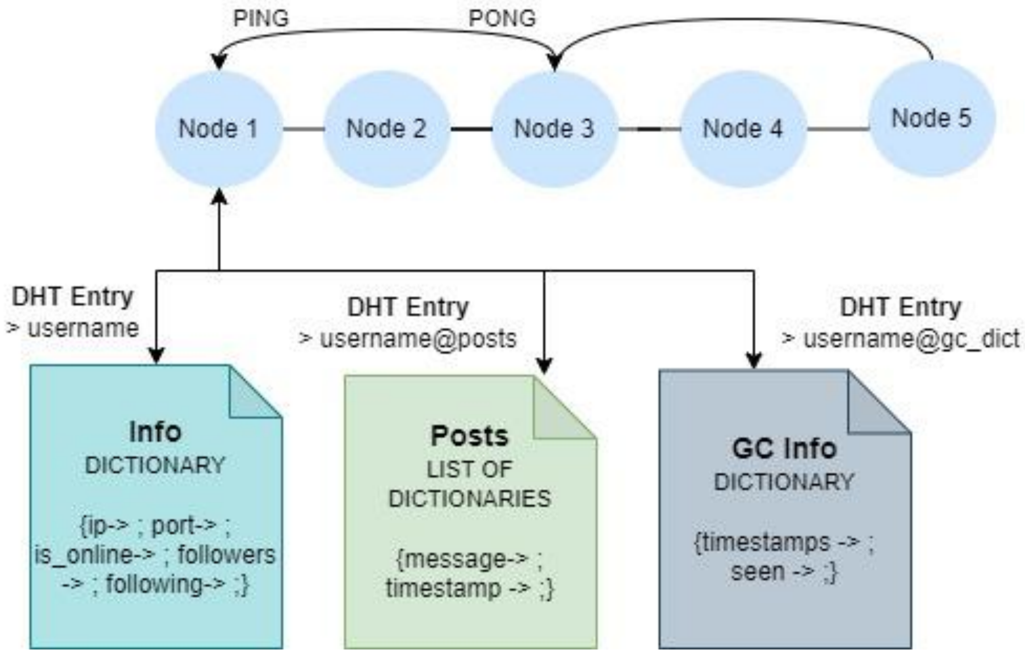
Four main files:

bootpeer.py
mainmenu.py
peer.py
server.py

> python bootpeer.py
> python main.py [port]

ARCHITECTURE

NETWORK ARCHITECTURE

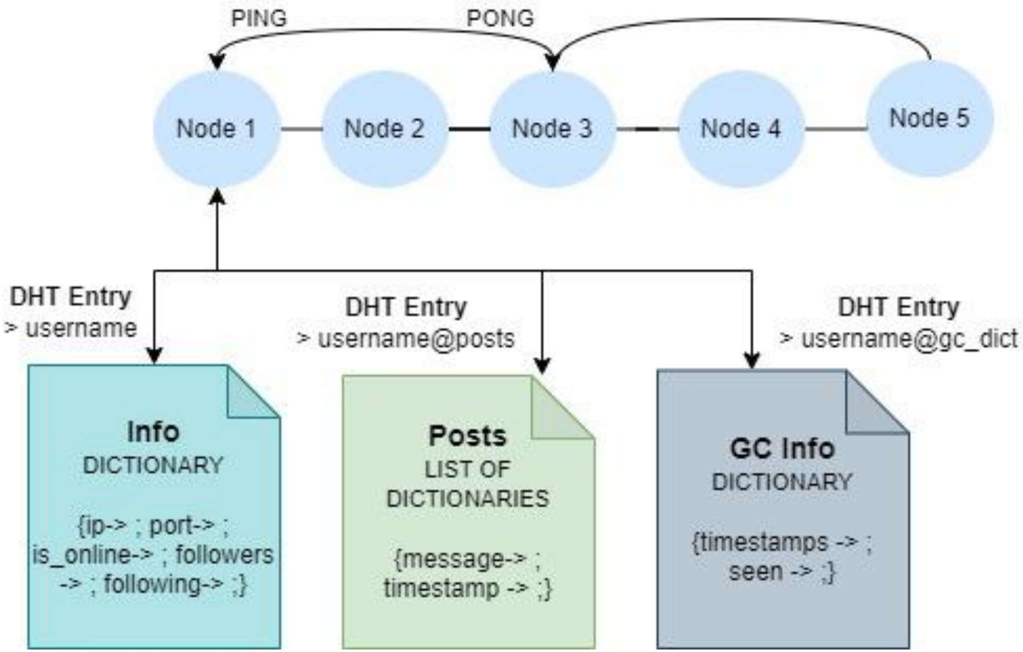


DISTRIBUTED HASH TABLE

ip	Peer address
port	Port listening for TCP connections
is_online	Online status
followers	Username's list of followers
following	Username's list of users being followed
message	Post content
timestamp	Timestamp of creation of post
timestamps	List of timestamps
seen	List of users that have seen posts of a respective timestamp

ARCHITECTURE

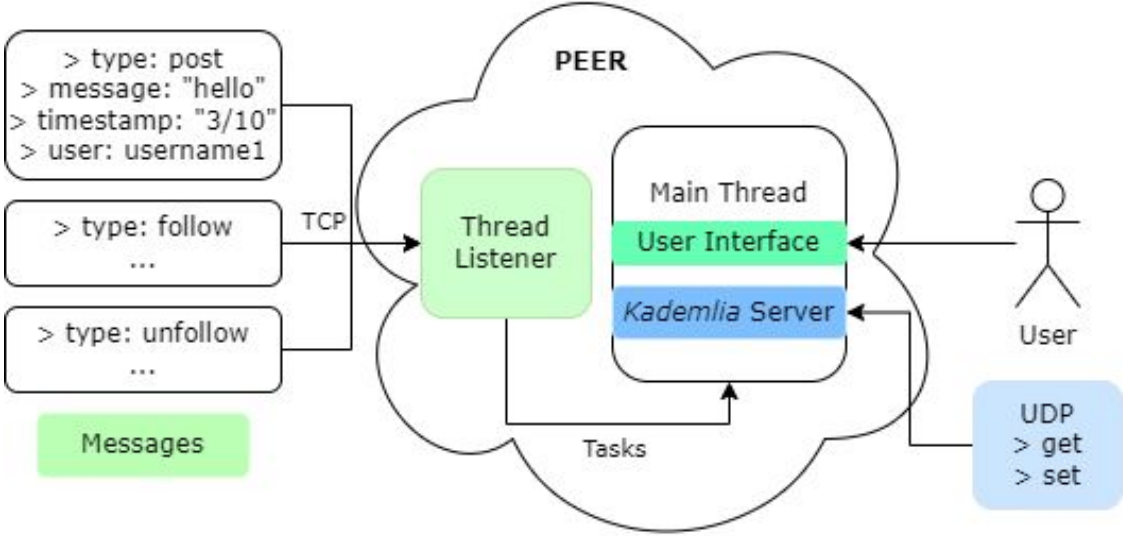
NETWORK ARCHITECTURE



- **Kademlia node connections** – XOR based distances
- 3 **bootstrap nodes** interact with periodic pings for entity recognition
- Most information is retrieved through Kademlia's crawling algorithms (using **get** and **set**)
- If both peers are online to **post and receive new messages**:
  Python Asyncio Sockets
  > `asyncio.open_connection`
- **Special DHT Entry** (created by a bootpeer on startup) > `user_list`

IMPLEMENTATION

FEATURES IMPLEMENTED



```
> type: post
> message: "hello"
> timestamp: "3/10"
> user: username1
```

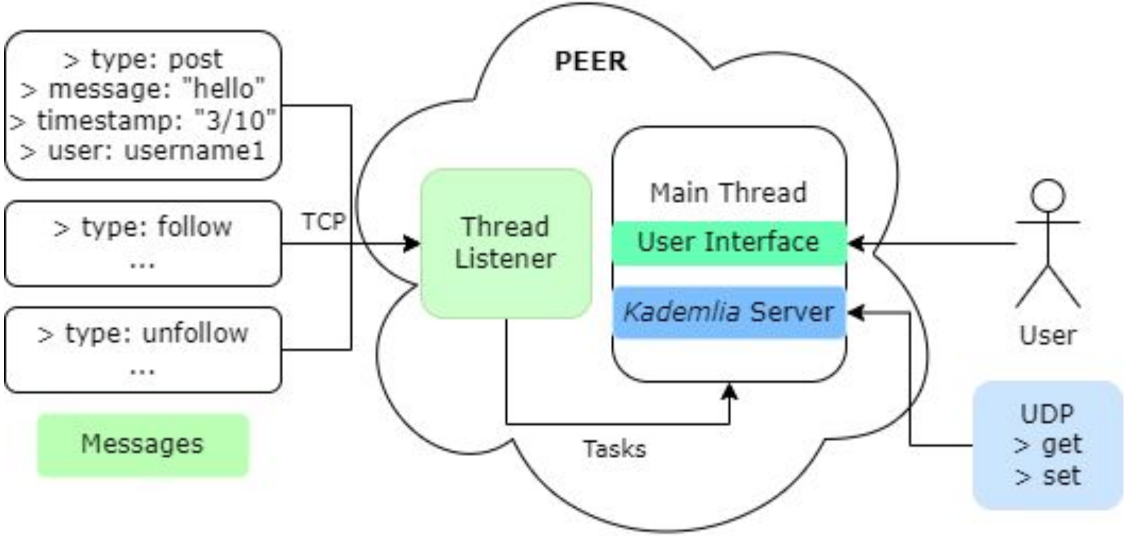
```
> type: follow
...
```

```
> type: unfollow
...
```

- **Follow and Unfollow an user**
  - Update DHT Info Entries;
  - Both online - asyncio TCP Socket;
- **See list of users being followed and following**
- **See Timeline Feed**
  - Sorted by timestamps
  - Update DHT GC_Info -> Seen entries

IMPLEMENTATION

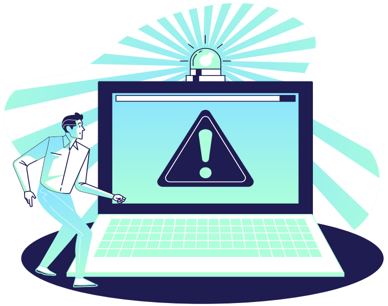
FEATURES IMPLEMENTED



- **Create new post**
  - Add post to DHT Posts entry
  - Sent directly if both peers are online
- **Authentication**
  - Register, Login and Logout
  - Bootstrap peers – always online

IMPLEMENTATION DETAILS

STORING OF INFORMATION

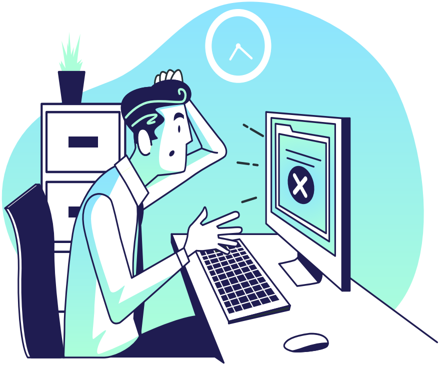


- **Scheduled asyncio tasks** to check if peers are still online:
  - Bootpeer : > check_online_status > check_connection
- **Get relevant timeline posts after reconnecting**
  - Get followers list from the network and get all relevant posts from respective users
- **Network Time Protocol for clock synchronization**
  - Periodical requests to an NTP server; the system clock is adjusted with a system call. When a new post is made, the system time is used.

IMPLEMENTATION DETAILS

EPHEMERAL STORAGE OF MESSAGES

- **Get previous posts to the following operation that are still in the network**
    - After a `FOLLOW`, the feed is automatically updated
- **Garbage collection** – Scheduled asyncio task by the bootpeer
    - Delete every post that has reached all followers
    - If a user has more than 10 posts, delete:
        - Posts that have reached the highest % of followers
        - Tie case: posts with older timestamps



Conclusions and References

The decentralized timeline service was implemented with success.

We implemented a consistent distributed system, where the state among peers is the same, even after reconnections.

Kademlia's DHT allowed most information/entries to be spread out throughout the nodes, which facilitates fault tolerance regarding posts (almost no posts can be lost, and the ones lost can be controlled - garbage collection) and disconnections (periodic ping-pongs and crawler)

However, this implementation may cause some issues with concurrency and slowness because of the high amount of tasks and data being transmitted in the network.

- ❏ Kademlia Documentation — Kademlia 2.2.1 documentation
- ❏ asyncio — Asynchronous I/O — Python 3.10.2 documentation
- ❏ JSON
- ❏ ntplib - PyPI
- ❏ datetime — Basic date and time types — Python 3.10.2

DEMONSTRATION VIDEO

THANKS!

Do you have any questions?

Carlos Lousada – up201806302@up.pt
José Maçães – up201806622@up.pt
Mariana Ramos – up201806869@up.pt
Tomás Mendes – up201806522@up.pt

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, infographics & images by **Freepik** and illustrations by Stories