Command Line's Decentralized Timeline SDLE

...

João Matos, up201703884
Miguel Neves, up201608657
Rui Pinto, up201806441
Tiago Gomes, up201806658

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Problem Description

- Create a decentralized timeline service.
- Similar to services like Facebook and Twitter.
- Users are authenticated, can subscribe to each other's timelines, and are able to publish messages.
- Users shouldn't need to be online for other users to be able to see their messages.

Technologies

- Python
- Console Menu (https://pypi.org/project/console-menu/)
- NTP Library (https://pypi.org/project/ntplib/)
- Kademlia (https://pypi.org/project/kademlia/)
- Async IO (https://docs.python.org/3/library/asyncio.html)

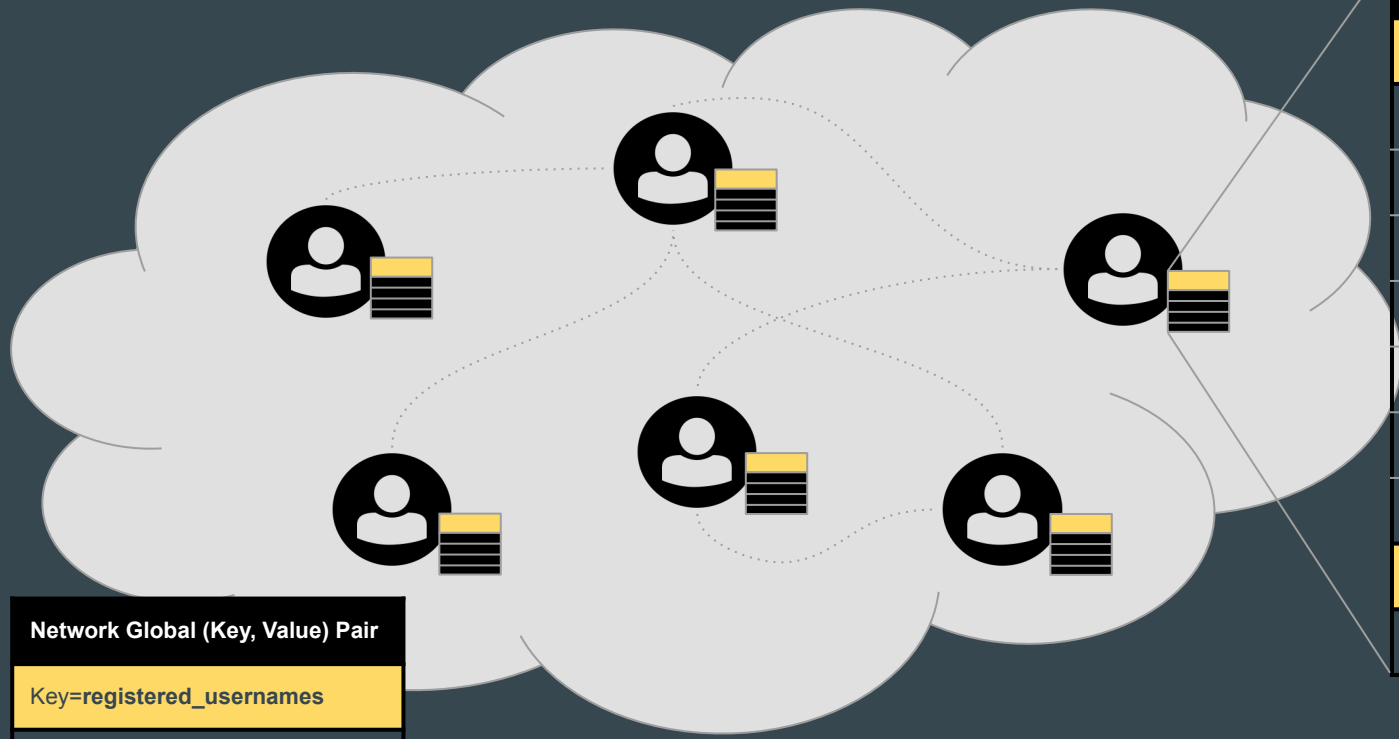
Why Kademlia?

It's just a matter of **evolution**...

P2P File sharing networks:

- **1st Generation (Napster)** - Relied on a central database to coordinate lookups on the network.
- **2nd Generation (Gnutella)** - Used flooding to locate files, searching every node on the network.
- **3rd Generation (Chord, Kademlia, ...)** - Uses Distributed hash tables to store resource locations throughout the network to perform lookups in it.

Network Architecture



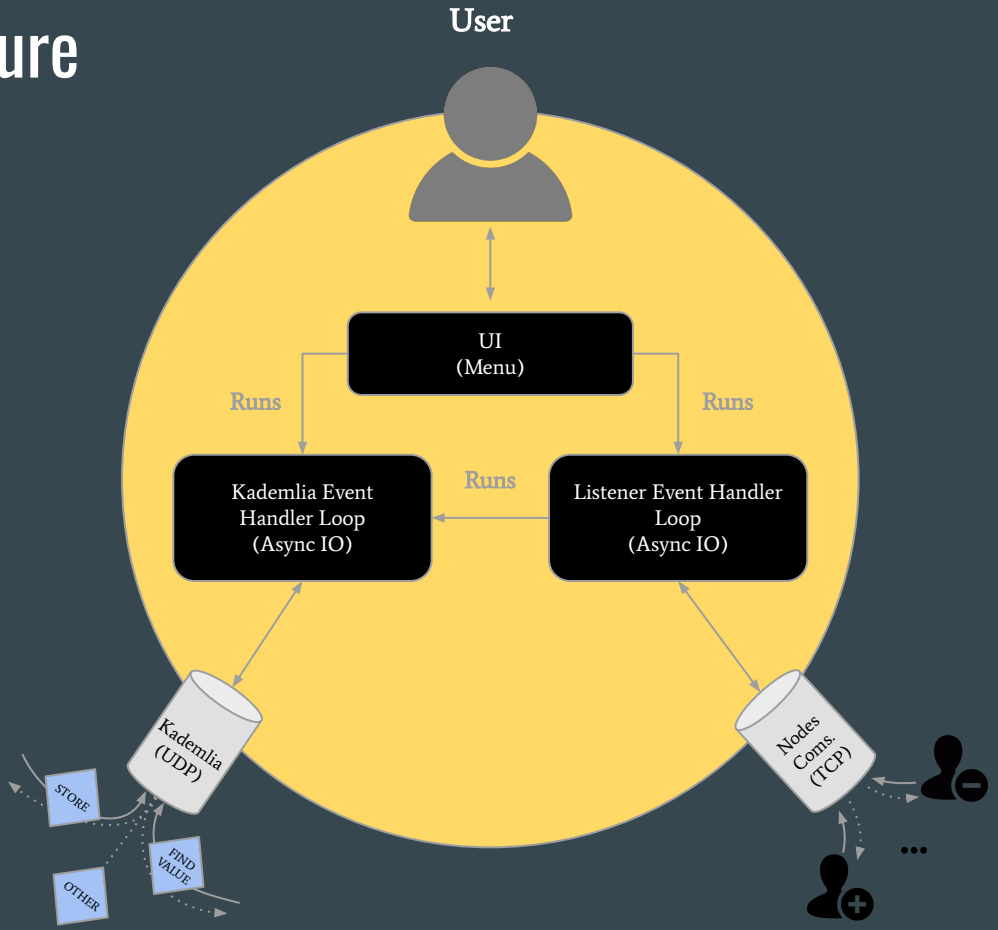
(Key, Value) Pairs
Key={username}
IP
Port
Password
Followers
Following
Messages
Pending Unfollow
Key={username}-pending_followers
Usernames

Network Global (Key, Value) Pair
Key=registered_usernames
Usernames

User State

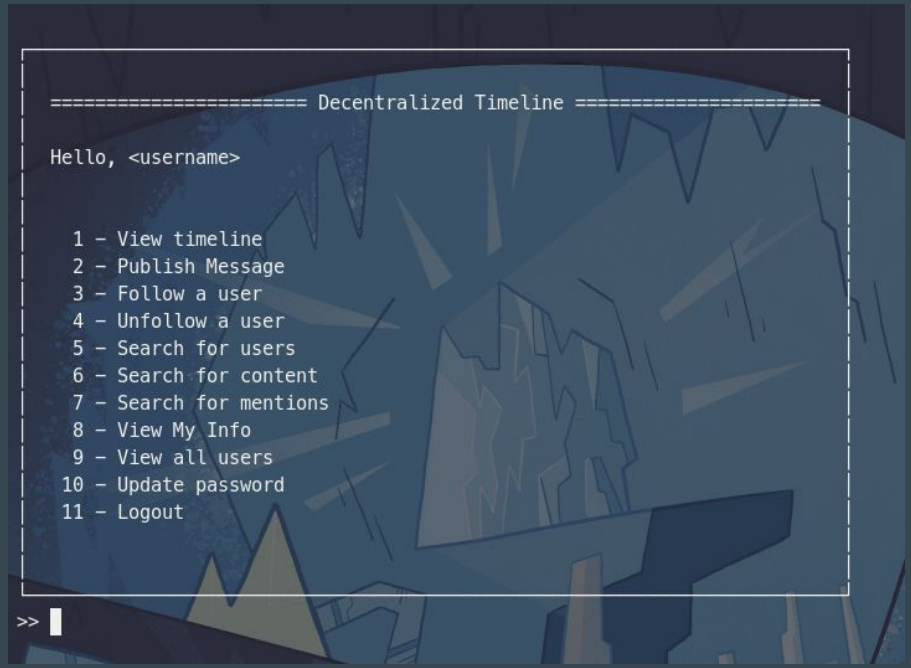
(Key=username, Value) Pairs	
IP	“127.0.0.1”
Port	6000
Password	“07deaa370c0dab6b2bf0f8f25905f94df59295f4504a0ef6c …”
Followers	[“John”, “Mary”, “Laura”]
Following	[{ username: “John”, last_msg_timestamp: “2022-01-08 23:37:52.765412+00:00” }, …]
Messages	[[“Hello World!”, “2022-01-08 23:40:01.813210+00:00”], …]
Pending Unfollow	[“Mary”]
(Key=username-pending_followers, Value) Pairs	
Usernames	[“Bob”]

Node Architecture



Functionalities

- Login/logout
- Register
- Publish messages
- Check mentions across the network
- View timeline
- Follow/unfollow other users
- Search:
  - Search for messages
  - Search for users
- View all users on the network
- Update password



Implementation Details

- **Authentication** is done using a **username** and **password**. The credentials are compared with the information in the network.
- All data is on the network and can be accessed from any node in the network.
- Use of asynchronous code for Kademlia **STORE** and **FIND_VALUE** operations.
- In **Clock synchronization**, we opted for using **external clock synchronization** with a server providing time using the **NTP**.
- **Causal Ordering** is obtained by comparing and sorting messages using timestamps.

Implementation Details (cont.)

- Concurrent changes in network states. Here, we based our solution on a **CSMA/CD** approach where we wait a random amount of time every time a concurrent change happens. Therefore the system is considered **Eventually consistent**.
- Each node has 2 additional threads, one for message reception and another to keep trying to set a value while using the aforementioned **CSMA/CD** approach.
- When viewing the timeline, a node fetches the messages of the following nodes' timelines in the network. These operations are done in **parallel**.

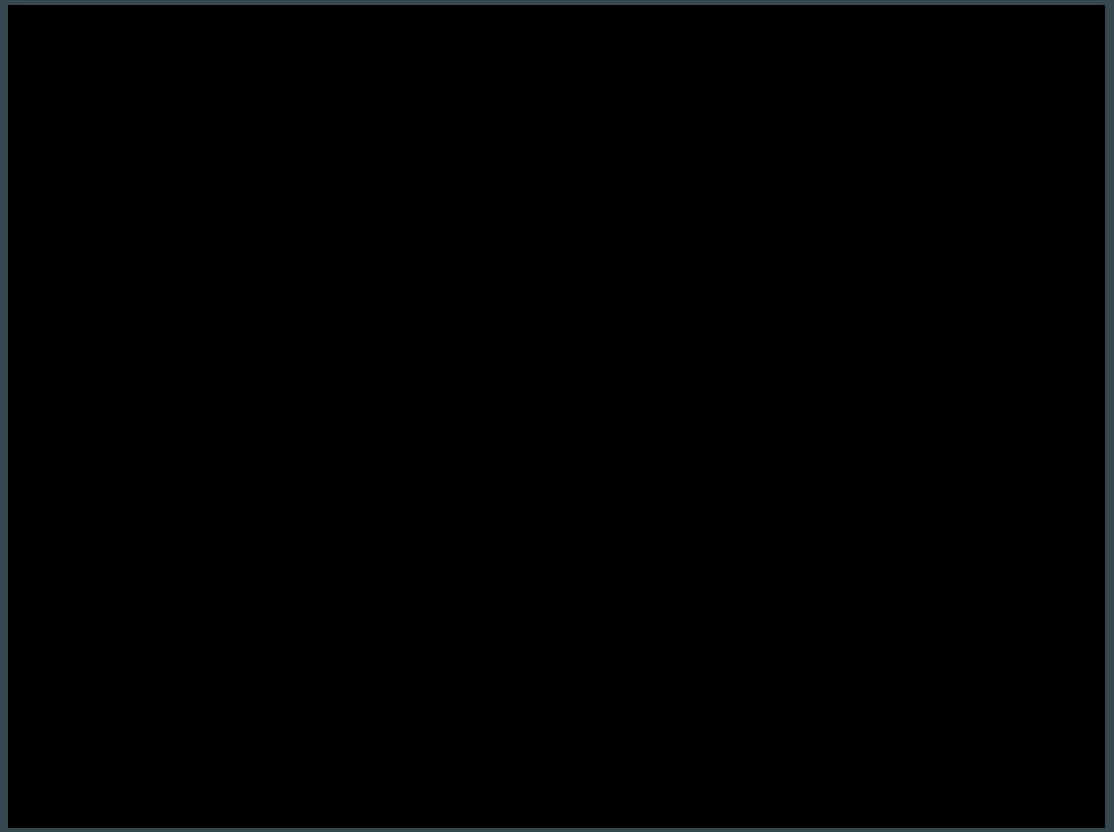
Implementation Details (cont.)

- Defining a protocol for direct communication between nodes, using *asyncio* sockets through the TCP protocol. This communication is **SSL encrypted** providing **confidentiality** to the messages.
- Messages are never discarded but they are not shown to other users if they have already read that message and a configurable timeout of 1 minute has passed.
- Node failures are handled using the network (key, value) pairs.

Problems found during development

- Time taken by a Kademlia **STORE** operation is approximately 10s.
- More actions should be performed concurrently inside a node to make the project more scalable.
- From our testing at least 2-3 nodes need to be available for the data in the network to be consistent and accessible.

Demonstration



Questions?