Rettiwt

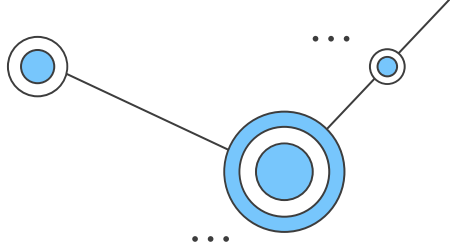
Large Scale Distributed Systems

FEUP 2022/2023

MEIC03 G15

Carlos Gomes - up201906622@edu.fe.up.pt
José Costa - up201907216@edu.fe.up.pt
Pedro Silva - up201907523@edu.fe.up.pt
Sérgio Estevão - up201905680@edu.fe.up.pt

Problem description



Build a social network (similar to Twitter or Facebook):

- Users can **post content**
- Users can **follow**/**unfollow** other users to see their content
- Users can see a **timeline** of their content and of others

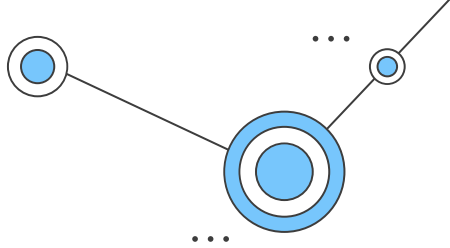
The social network should be decentralized:

- Shouldn't be controlled by an authority/organization
- Peers should be equipotent and equally privileged

Raises a lot challenges and questions:

- How to do user and data authentication?
- How to do access control?
- How is content stored and distributed?
- How to do moderation?
- How do users know each other?

Architecture



P2P architecture:

- Users are **peers**
- **Bootstrap nodes** to provide **initial configuration** to new peers on the network
  - But “normal” peers should also be able to provide bootstrapping
- Bootstrap nodes know each other and peers **know them initially**
- Use a **Distributed Hash Table (DHT)** as infrastructure for the system
  - Structured approach to efficiently search the overlay network for a given content

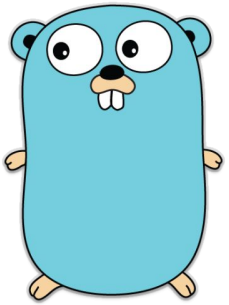
Each peer:

- Has **local storage** and **cache**
- **Never leaves** the system
- May go offline
- One account

Technologies

Go:

- Performant - compiled language
- Fast compile/build times
- Goroutines and Channels
- Runtime and compile time race condition detections
- Simple language

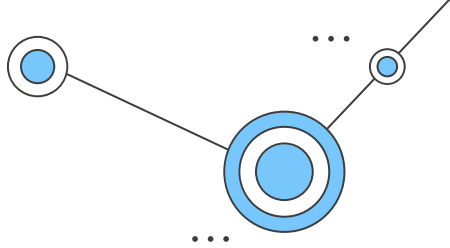


Libp2p:

- Networking stack made by the IPFS community
- Modular and extensible system of protocols
- Enables building P2P systems
- Fast develop time
- But documentation leaves a lot to be desired

Design

Kademlia DHT



DHTs are P2P staples:

- Efficient lookups
- Can nullify DoS attacks

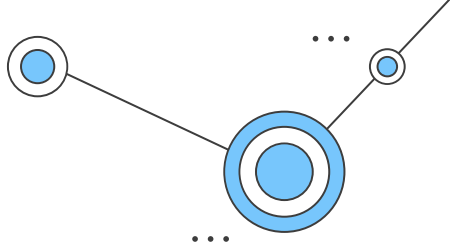
We chose **Kademlia**:

- In a search, at most **O(log(n)) contacts** are performed
- Efficiency with **XOR metric** for distance between points in key space
- Exploits the fact that node failures are inversely related to uptime.

LibP2P has a Kademlia implementation:

- Seems to use quorums and mechanisms to avoid concurrency issues
- One namespace for each type of record (ex: pk, ipns)
- Each namespace requires a validator
- We can pass the bootstrap nodes to the DHT and the library takes care of everything

Peer discovery



LibP2P offers utilities for **peer discovery**:

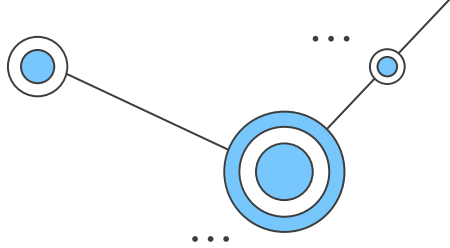
- NewRoutingDiscovery() : We used the **DHT** as discovery method
- Advertise() : **Persistently advertise** a service
- FindPeers() - **Collects peers** from discoverer (synchronous)
- Connectedness() and DialPeer() - **check** and **add connectivity**
- Bootstrap mechanism

Normal peers and bootstrap nodes do peer discovery

No **hole punching** (NAT, firewalls, ...):

- But the library allows to do it

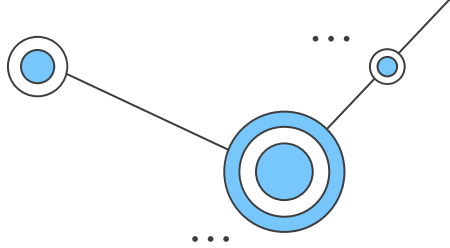
User authentication



User authentication implemented with the DHT's key-value store:

- New namespace where the key is a **username** and the value is the **hashed password**
- Hashes done with **Bcrypt** and a **cost of 10**
- **Uniqueness** of a **username** (during Register) ensured by checking if a record already exists with that username
- Login is done by comparing the hash of the provided password with the hash stored on the DHT

Content Routing – Followers



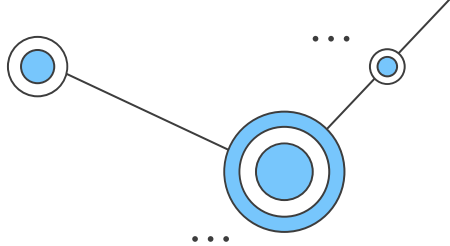
We used a **pub/sub** pattern:

- A topic is a given user
- Subscribing a topic is **following** that user
- Unsubscribing a topic is **unfollowing** that user

LibP2P has a **GossipSub** implementation, which we used

- ListPeers() : returns the subscribers of a given topic

Content Routing – Timeline



The DHT has built-in content routing functionalities:

- Provide() : to "announce" that the peer can provide a given content identified by a **CID** (content ID)
- FindProviders() : to return the peers that can provide a given content
- A Provide record is **ephemeral** (the default is 12 hours)

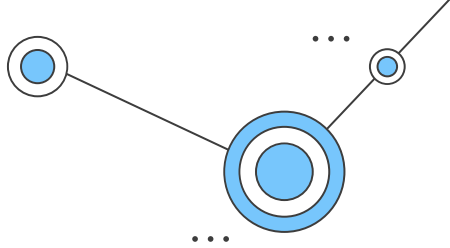
When a **peer** wants to **post something**, he:

- Creates a CID for the post
- Provides the content
- Saves it on his CIDs cache (inside the DHT)
- Publishes the CID with pub/sub

When a **follower** of that peer wants to **get his post**, he:

- Reads the published CID
- Finds the providers of that content
- Asks for the content from the providers (not only the owner of the post)
- If he missed the CID because he was offline, he can check the cache for it
- He provides that content

Posts Cache and storage



A user stores his posts **indefinitely** in storage.

A user keeps a **cache** of posts from the people he **currently follows**:

- Keeps a preset number of posts per person
- Keeps a post for at most 24 hours
- Posts are garbage collected with task scheduling
- On node startup, we checks for outdated posts

Conclusions, Limitations and “Future Work”

Conclusions

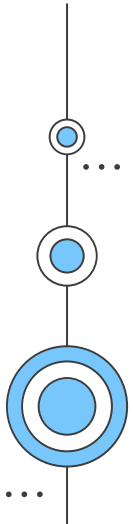
- We developed a basic decentralized social network
- Go + LibP2P proved to be powerful tools to develop distributed systems. LibP2P lacks good documentation.
- P2P systems are hard to build :(

Limitations

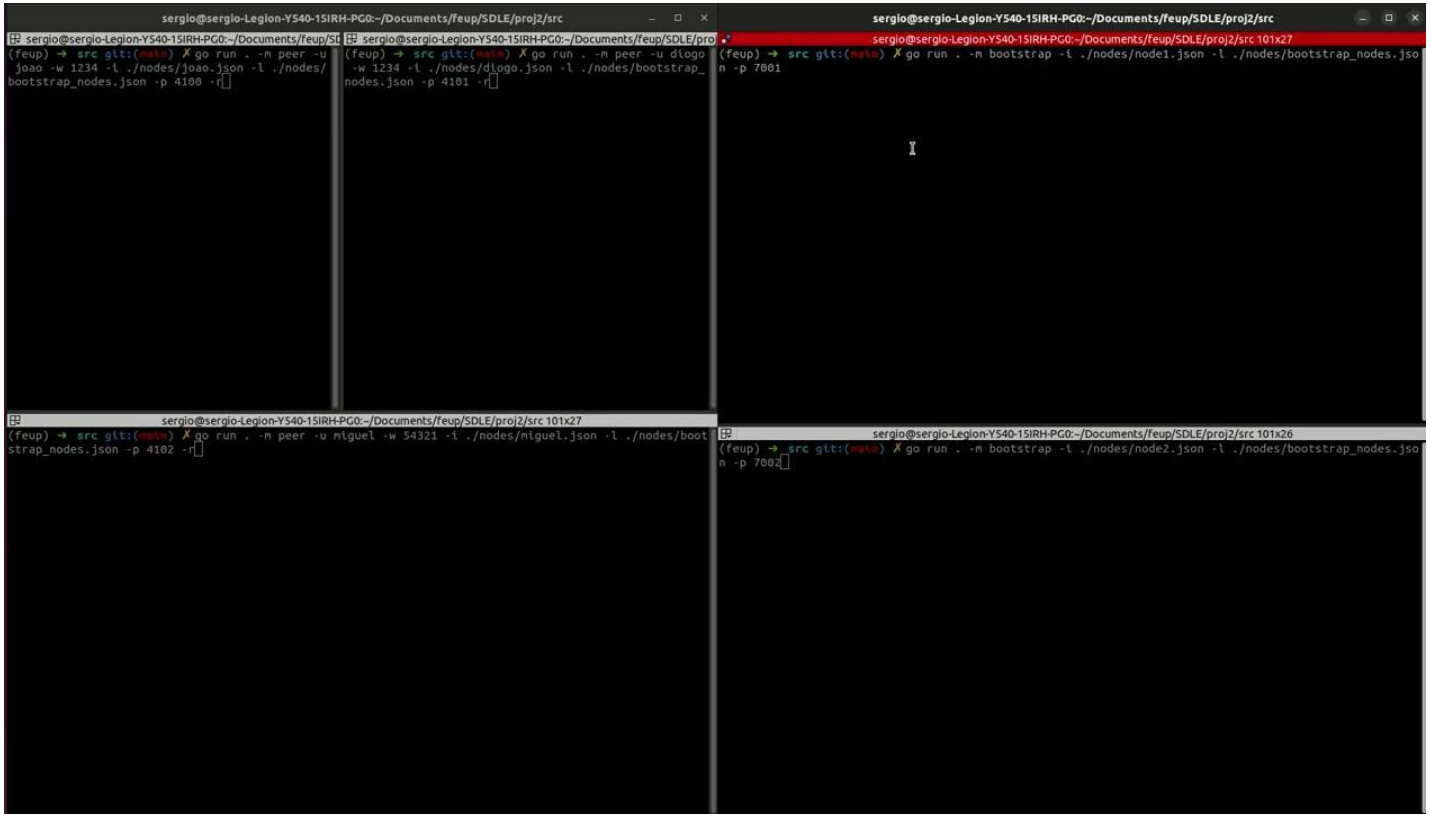
- DHT records have no access control => anyone on the network can change them (hashes, CIDs, etc..)
- Limited to one account per node
- Followers might miss a post completely
- Post garbage collection is heavy

“Future Work”

- Implement content control (abuse and spam control):
  - Reward well-behaved users and punish badly-behaved ones?
  - How do we avoid inadvertently creating biases which lose the trust of our users?
- Authentic data:
  - Signatures
- Configurable caching
- Using my account on another node
- Better content routing
- Better garbage collection
- Implement a more user friendly interface (e.g front end ui)



Demo


sergio@sergio-Legion-Y540-15IRH-PG0:~/Documents/feup/SDLE/proj2/src
(feup) ➜ src git:(main) ✗ go run . -m peer -u joao -w 1234 -l ./nodes/joao.json -l ./nodes/bootstrap_nodes.json -p 4100 -r
(feup) ➜ src git:(main) ✗ go run . -m peer -u diogo -w 1234 -l ./nodes/diogo.json -l ./nodes/bootstrap_nodes.json -p 4101 -r
sergio@sergio-Legion-Y540-15IRH-PG0:~/Documents/feup/SDLE/proj2/src 101x27
(feup) ➜ src git:(main) ✗ go run . -m bootstrap -l ./nodes/node1.json -l ./nodes/bootstrap_nodes.json -p 7001
sergio@sergio-Legion-Y540-15IRH-PG0:~/Documents/feup/SDLE/proj2/src 101x27
(feup) ➜ src git:(main) ✗ go run . -m peer -u miguel -w 54321 -l ./nodes/miguel.json -l ./nodes/bootstrap_nodes.json -p 4102 -r
sergio@sergio-Legion-Y540-15IRH-PG0:~/Documents/feup/SDLE/proj2/src 101x26
(feup) ➜ src git:(main) ✗ go run . -m bootstrap -l ./nodes/node2.json -l ./nodes/bootstrap_nodes.json -p 7002

Questions?