Decentralized Timeline

2nd Project Presentation

Large Scale Distributed Systems

FEUP - MEIC 21/22

18th January 2021

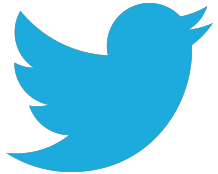
João Pires

João Romão

Rafael Cristino

Xavier Pisco

Problem Approach 🔎



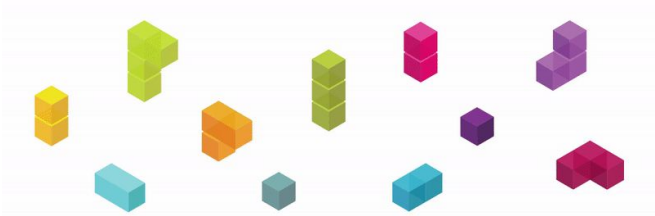
What to do?

- The assignment consisted on a implementation of a distributed timeline
- Users should be able to subscribe and unsubscribe each other, and should be able to retrieve relevant information

How to do?

- What libraries could be relevant?
- Which architectures could be used?
- What assumptions could be made?

Libp2p



Why libp2p?

- Javascript implementation of peer-to-peer networks:
  - Peer routing and discovery already implemented.
  - Transport layer with simple modular protocol based interface.
  - Security on messages sent.
  - Unique identification of peers.
- Implementation of distributed system patterns:
  - PubSub
  - Kademlia
- Documented tutorials

Cons of libp2p?

- Lack of documentation for subjects not covered by the tutorials (in some cases inexistant)
- Some examples and tutorials were not working out of the box

Architectures Taken Into Consideration

Kademlia

✔ **Pros**

- Quickly find any peer in the network
- Saves information about a small subset of the peers, therefore using less memory
- Content constantly and easily available in the network

✖ **Cons**

- When user creates a post, no notification is received
- Peers may store posts from peers they don’t subscribe

PubSub

✔ **Pros**

- Peers receive notifications of peers they subscribe
- Quickly find and connect to other peers
- Only peers subscribed to the peer will receive and store its posts

✖ **Cons**

- Availability of information is dependent on available peers

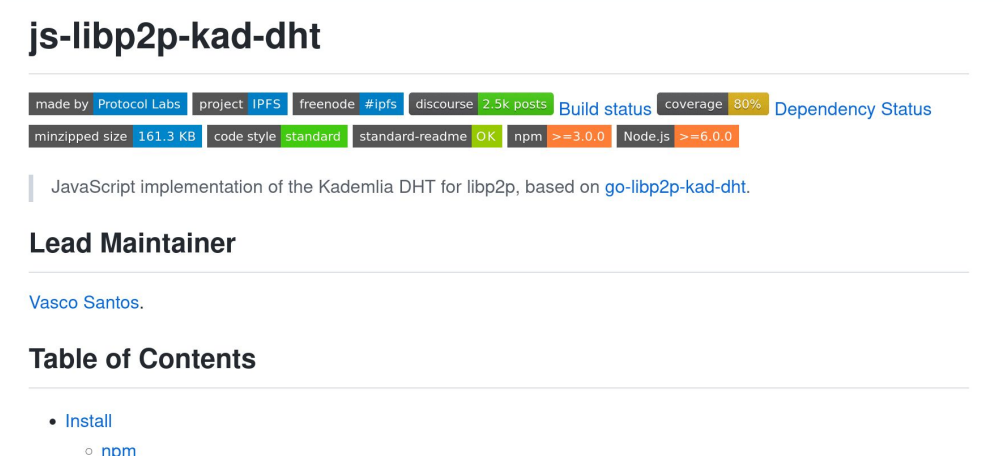
Ideal Scenario 🙌

Combination of both patterns

- Peers connect to each other based on Kademlia
- When a peer creates a post, it publishes its id using PubSub and adds it to the Kademlia network
- This way, user knows about the existence of a post of a subscribed user without saving it
- Peer knows about posts from subscribed peers automatically, posts are easily and reliably accessible in the network

Hard Reality 🤦

- Libp2p documentation is not great.
- We couldn't make Kademlia content routing work.
- We decided to only use PubSub.



js-libp2p-kad-dht

made by Protocol Labs | project IPFS | freenode #ipfs | discourse 2.5k posts | Build status | coverage 80% | Dependency Status
minzipped size 161.3 KB | code style standard | standard-readme OK | npm >=3.0.0 | Node.js >=6.0.0

> JavaScript implementation of the Kademlia DHT for libp2p, based on go-libp2p-kad-dht.

Lead Maintainer

Vasco Santos.

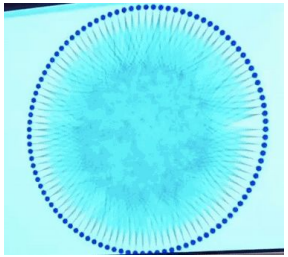
Table of Contents

- Install
  - npm

FloodSub vs GossipSub

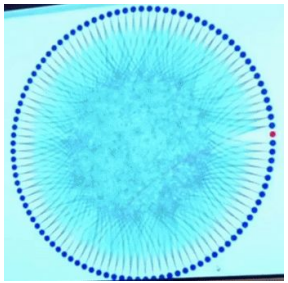
FloodSub

- Send message to all the peers it knows that are following the topic.
- Minimizes latency since messages are sent by all the available paths.
- Highly robust, as peer will receive multiple times the same message.
- Creates a lot of redundant traffic in the network.



GossipSub

- Send message to a random subset of peers that follows the topic.
- Uses control messages to avoid replication of messages.
- Increases latency as a tradeoff with bandwidth efficiency.



Implementation

Peer Discovery

Peer discovery based on Bootstrap and GossipSub

- When a peer want to join the network, it needs the id of any other peer already in the network.
- After the first peer discovery, the other peers will be discovered based on GossipSub.
- Libp2p creates unique IDs for each peer with 58 bytes.
  - QmZz5QRpw4D1kg4Fn666dMt33ukd6qjy4AC6bPB7MrLKHk
- To simplify their names, when two peers discover each other, they share the usernames they chose.

Post Distribution

- Our post distribution is completely based on GossipSub.
- To each peer corresponds a topic, which is equal to its username.
- To publish a post, the peer publishes a **POST** message to its topic.
- The new peer’s posts are delivered to the topic subscribers.

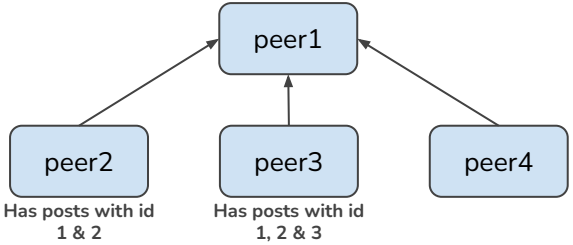
Post Distribution

- This topic also supports two control messages types that enable control mechanisms that involving all of the peer's subscribers: **FIND** and **SEND.**
- **FIND** - protocol initiated by a peer that needs to obtain posts published after a specified timestamp.
- **SENDING** - control message sent by peers that are sending posts to the peer who requested them.
    - includes the unique identifier of each of the posts
    - if any peer has posts that have not yet been sent by the other peers, they will send those too

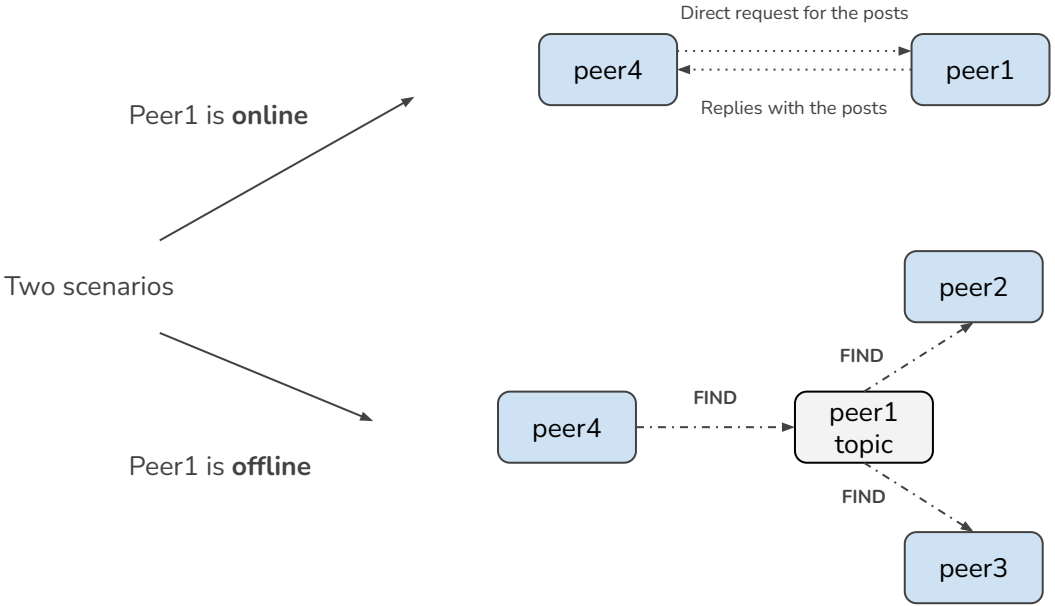
FIND Protocol: Why?

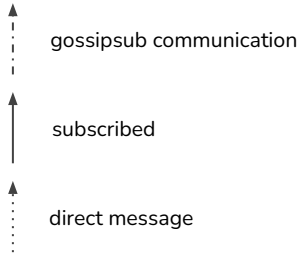
- This protocol is executed with the goal of obtaining older posts of a peer.
- The find protocol is initiated when a peer subscribes to another peer. This happens in more than one occasion:
    - When the peer is subscribing to another peer for the first time - **obtains the posts that were published when he was not a subscriber**.
    - When the peer is re-subscribing to another peer, for example after being offline and turning back on - **obtains the posts that were published while it was offline**.

FIND Protocol: How?

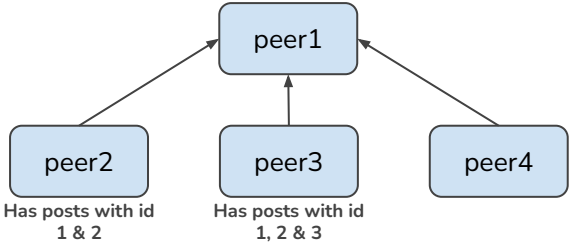


- Let there be 4 peers: **peer1**, **peer2**, **peer3 and peer4**.
- **Peer2**, **peer3** and **peer4** are subscribed to **peer1**.
- Let's suppose that **peer4** starts a FIND protocol to obtain **peer1**'s older posts.





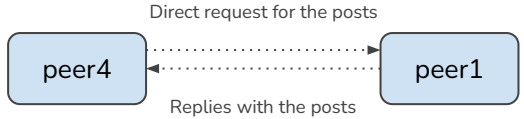
FIND Protocol: How?



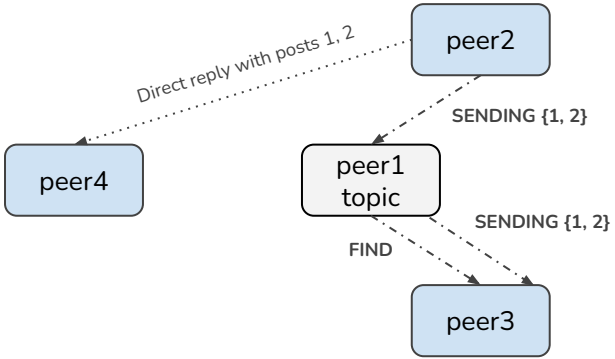
- Let there be 4 peers: **peer1**, **peer2**, **peer3 and peer4**.
- **Peer2**, **peer3** and **peer4** are subscribed to **peer1**.
- Let's suppose that **peer4** starts a FIND protocol to obtain **peer1**'s older posts.

Two scenarios

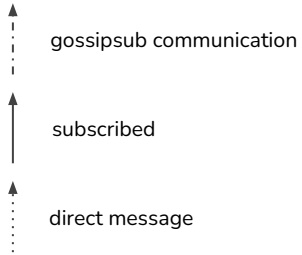
Peer1 is **online**



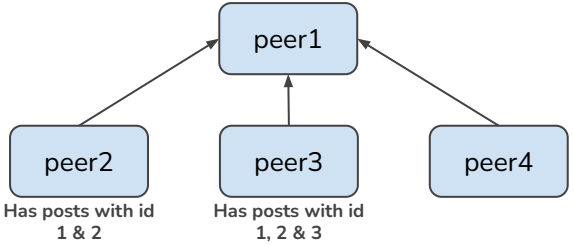
Peer1 is **offline**



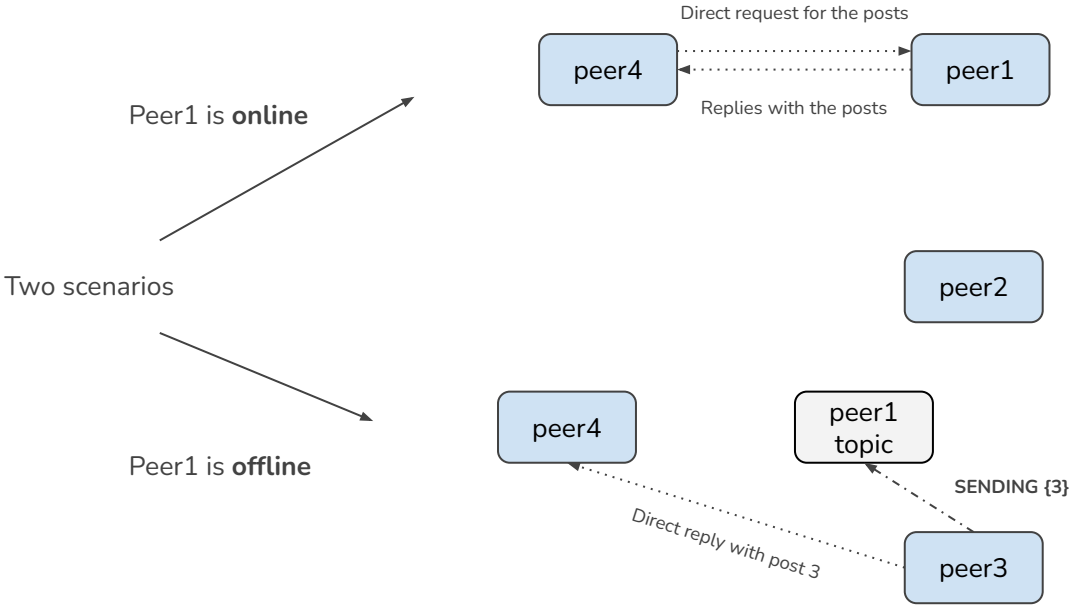
Note: peer2 sends first due to random delay after FIND message reception



FIND Protocol: How?



- Let there be 4 peers: **peer1**, **peer2**, **peer3 and peer4**.
- **Peer2**, **peer3** and **peer4** are subscribed to **peer1**.
- Let's suppose that **peer4** starts a FIND protocol to obtain **peer1**'s older posts.



Note: if by chance more than one peer replies with the same posts, the user who initiated the protocol detects that the posts are the same (same id) and only adds one to the timeline

Peer Storage

- Each peer has information about both himself and other peers, and stores it in a file every second.
- Saved information:
  - Username
  - Timeline
  - Own posts
  - Users it subscribes to
- When a peer is restarted with the same username, it will read this information and continue from the point where it was stopped.
  - For example, will subscribe to the users to whom he was subscribed before going offline.

Problems Faced

and our solutions

Subscribing a peer

Problem: obtain the older peer posts when subscribing

- Send a message to the peer it wants to subscribe asking for previous posts.
- If the peer is not online, request the information from the peer's subscribers by publishing a control message to the peer's topic (FIND protocol).
    - In this case, in order to avoid a flood of messages from all of the subscribers replying at the same time they publish a SENDING message to the topic
- Either the original peer replies with all of the posts, or its subscribers reply with all of the posts they have.

Restart a peer

Problem: when a peer shuts down and restarts it should:

- continue subscribing the peers it subscribed
- receive posts that were sent while it was offline

- Peer re-subscribes to all the topics it subscribed before went down.
- When subscribing it will ask for posts that were published after the last post it has received, in a similar way to the one described in the last slide (FIND protocol).

Test App

35425 Load peer Close peer Reload

Hi, one

Publish

Publish

Timeline

Number of posts to keep Clear

Hi, I am two!
by two Tue, 18 Jan 2022 00:02:54 GMT

Own Posts

OIIIII
by one Tue, 18 Jan 2022 00:02:43 GMT

Hi, I am one!
by one Tue, 18 Jan 2022 00:02:32 GMT

Users

two Unsubscribe
three Subscribe
four Unsubscribe
five Subscribe
six Subscribe

Conclusions

Conclusions

- We confirmed that good documentation is essential for other users to understand and make use of our code.
- All designs will have its own flaws. No design is fail proof.
- Mechanisms to overcome these flaws will always have to be studied.
- It was possible to understand the several advantages that a distributed design has when compared to a centralized one.

Demo