SDLE

T5G12

Allan Sousa - up201800149

Breno Pimentel - up201800170

Diogo Gomes - up201806572

Margarida Cosme - up201709304

Introduction

Objectives

- Build a decentralized timeline application
    - Users can subscribe to other users
        - Users can store messages from subscribed users.
        - Users can forward and provide these stored messages to interested parties.

- Exploration of peer-to-peer challenges
    - Connectivity
    - Peer discovery
    - Data Replication
    - ...

Tools

- C++
- Python
- Boost.Asio (Asynchronous communication)
- Pyside (Interface)

Peer-to-Peer Layer

Kademlia

The application's underlying peer-to-peer layer was built using an implementation of the Kademlia DHT (Distributed Hash-Table) developed by us. The implementation supports all of the main procedures of original Kademlia. To name a few:

- FIND_NODE : Searches for a node in the network
- FIND_VALUE: Searches for a value associated with a given key in the network.
- STORE: Stores a key-value pair in the network.

The decision of implementing our own version of Kademlia was mainly for curiosity purposes, and as such, the optimizations mentioned in the original paper and the optimizations that were invented over the years were not implemented : the focus was a version of kademlia that worked well enough for our purposes.

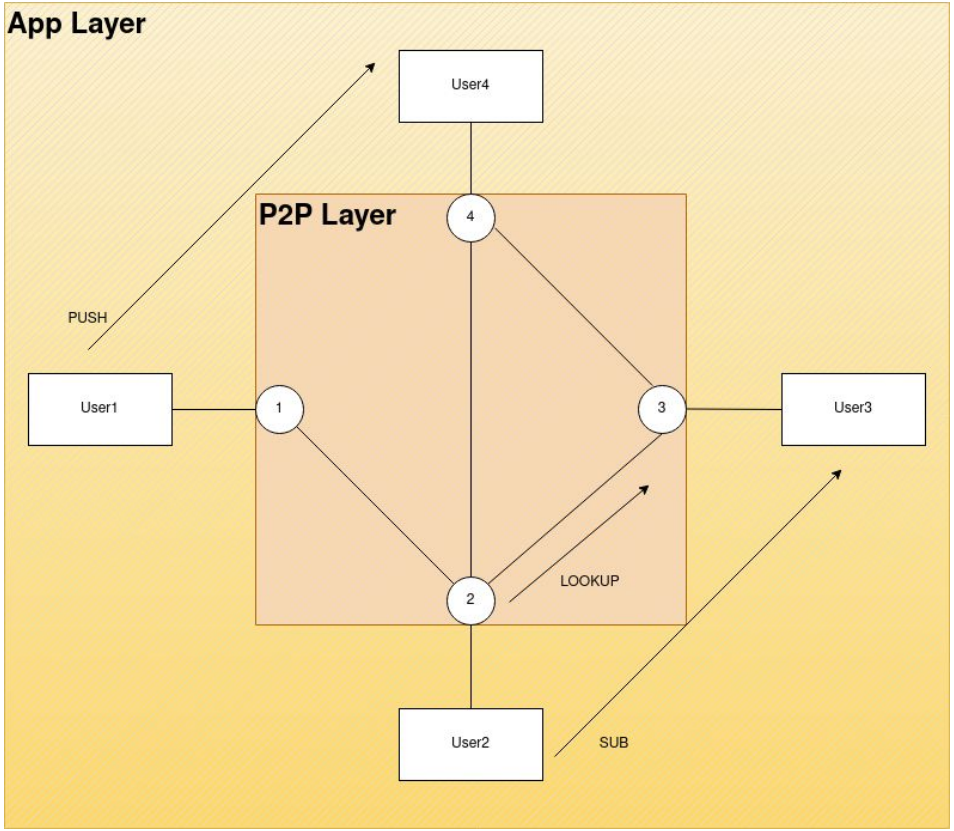
Usage of the Kademlia network by the application:

- Peer discovery

- Storage of followers(ID-ADDRESS pairs) of a peer in the DHT.
  - The peer ID is the key.

Application

System Diagram



System Diagram

Functionality

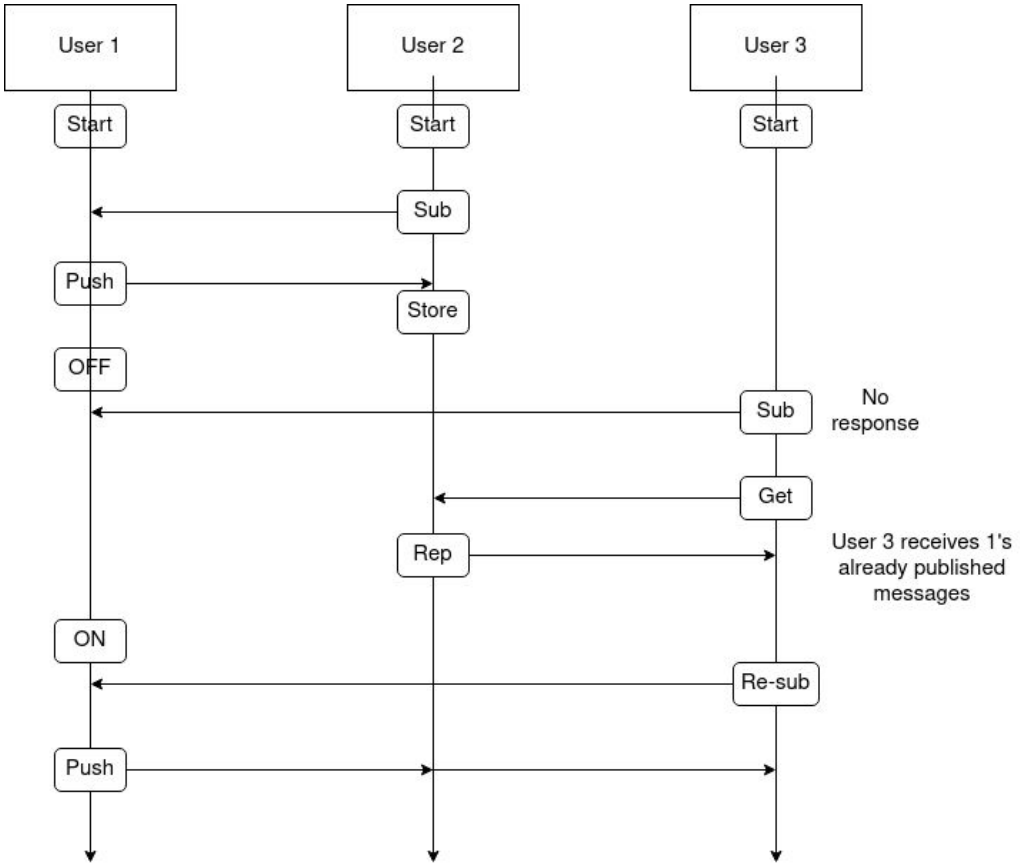
A user has access to the application API. This API provides the following functionality:

- **Subscribe** to a user.
  - Users can only subscribe to online peers. If a peer is offline, but the follower list (partial or total) is available in the DHT, the user will send GET messages to the followers to receive that user's already published messages.
- **Unsubscribe** from a user
  - Users can unsubscribe from users that they previously subscribed too. The issuing peer updates the DHT after receiving the unsubscribe and will no longer send messages to the issuing user. The stored messages from the unsubscribed peer will be dropped..
- **Publish** a message in the network.
  - A user can publish messages. The message is sent directly to all the current subscribers of the publishing peer. The publishing user does not expect a response from the receiving users : A user becomes aware of an outdated timeline if it receives a message which counter is not consecutive to the last message 's counter.

Design choices

- Only the peer whose ID matches an entry in the DHT can update that entry :
  - This is a choice to avoid “race conditions” : e.g if any peer can update any entry, a popular user will have its entry in the DHT updated frequently, possibly at the same time, and information would be lost.
- The DHT only stores the followers of the user identified by the ID (key of the key-value pair).
  - This is a decision which aims to keep the activity in the peer-to-peer layer to a minimum : all other information is exchanged “directly” between application clients.
- Users do not respond PUSH messages (messages containing a message that was published by another user) :
  - This was proposed with the intention to increase scalability of popular users : with no reply expected to a PUSH message, popular peers do not need to track information about all the peers that did not receive a message (different list of peers that did not received a message, per message!) and do not need to worry how to handle such situations : synchronization occurs periodically.

Working example



Communication between peers

The peers communicate exchanging messages. These messages are encapsulated in UDP packets.

Fig 3 - Message structure

After receiving a message, its type is verified, and it will be either redirected (internally) to the application layer or the peer-to-peer layer.

Ephemeral messages

When a user subscribes to another user, the subscribing user will store up to N (configurable) messages from that user. Once the limit is reached, older messages will be dropped to accommodate new ones.

A user also stores all of its own messages. Once a user unsubscribes from another, all storage messages of the unsubscribed peer will be dropped locally.

Each [published] message has an 8-bytes counter (per user), so the order is preserved when storing/forwarding these messages.

Future work (based on identified problems)

- Security:
  - “Central Server/Peer” for distribution of keys (prevention of attacks in kademlia networks e.g sybil attack) and/or authentication.
- Discoverability:
  - No way around the “double NAT” problem in our system. Peers must not be behind a firewall :
    - Possible solution : Intermediate peer that supply the connecting peer with an IP:PORT pair of the wanted peer in order to perform “hole-punching”
- Message counters:
  - Last message ‘s counter could be included in a peer’s DHT associated value : Subscribed users would have an easy access to the last message of a user, simplifying synchronization.