Decentralized Timeline

LARGE SCALE DISTRIBUTED SYSTEMS

Catarina Pires
up201907925@edu.fe.up.pt

Diogo Costa
up201906731@edu.fe.up.pt

Francisco Colino
up201905405@edu.fe.up.pt

Pedro Gonçalo Correia
up201905348@edu.fe.up.pt

Group T3G14

Introduction

Goal — Implement a decentralized timeline service

Technologies

> Language — **Python**
> **Asynchronous implementation** (Python's asyncio package)
> **DHT** — **kademlia** (PyPI package)
> **Pretty print** results in tabular form — tabulate (PyPI package)

Bootstrapping

> **All nodes are equal** implementation-wise
> A node can specify the **addresses of other nodes** it knows of when starting
> There **may** be nodes with **widely known addresses** and **high uptime** guarantees

Clock Synchronization

> Relevant for **merging timelines of different users**
> Posts of different users **not causally related**
> Timestamps in timelines and timeline caches handled by the **owner of the timeline**
> Nodes' **local time** assumed to be **reasonably synchronized** with an NTP server

Operations

Optional flags in all operations

```
-h – show help message
-d – show debug logs
-l – local port
```

The **userid** is a pair **`ip:port`** with the node's ip and port used to receive requests from other nodes. The port can be omitted, with the default being 8000. A **one-to-one association** between **users** and **nodes** is assumed.

Start — starts running the user's node

```
python run.py start <userid>

Optional flags:
-k – port used for Kademlia DHT
-b – list of addresses to bootstrap the DHT network
-f – caching frequency
-t – cache time to live
-c – max cached posts per subscription
```

View — view your feed (posts made by you or by the users you subscribed to)

```
python run.py view [max-posts]
```

Get — find a user's timeline

```
python run.py get <userid> [max-posts]
```

Post — make a new post

```
python run.py post <filepath>
```

Remove — delete a post you made

```
python run.py remove <postid>
```

Sub — subscribe to a user

```
python run.py sub <userid>
```

Unsub — unsubscribe from a user

```
python run.py unsub <userid>
```

People I may know — 2nd degree connections

```
python run.py people-i-may-know [max-users]
```

Communication Protocol

DISTRIBUTED HASH TABLE

- Used to keep track of **user subscriptions**
- Implemented with **Kademlia DHT**
- By default communication happens on port 8468

Subscribed to users

`key` – **“<userid>-subscribed”**
`value` – **[<userid>, ...]**

Lists the users that a given user is subscribed to

Only that user will write to this key

Subscribers of user

`key` – **“<userid>-subscribers”**
`value` – **[<userid>, ...]**

Lists the users subscribed to a given user

Any user can write to this key → **race conditions** minimized with exponential backoff and recovered from with periodic checks

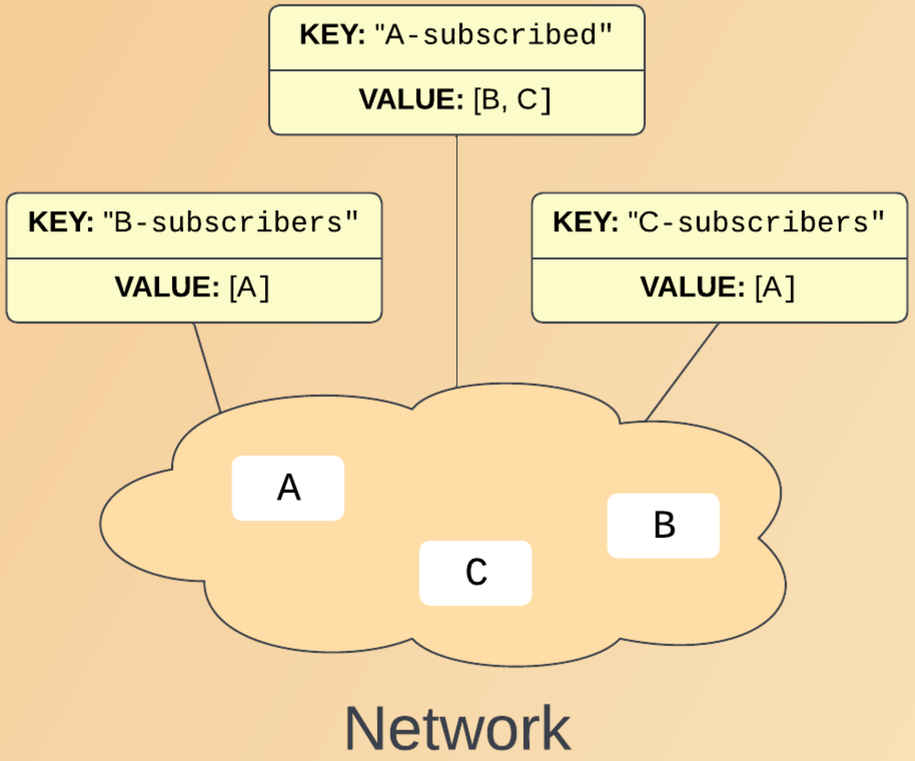

Fig. 1: Subscription information stored in the DHT when user *A* is subscribed to *B* and *C*.

Group T3G14

Communication Protocol

REQUESTS TO PEERS

- Used to obtain Timelines of other users
- Requests and responses sent in **JSON**
- Communication over **TCP** socket
- By default communication happens on port 8000

Request

> command – **“get-timeline”**
> userid – **user who’s timeline we are looking for**
> max-posts – **maximum number of posts** (optional)

Response

> status – **“ok”** or **“error”**
> error – **error message** (if status == “error”)
> timeline – **timeline cache** (if status == “ok”)

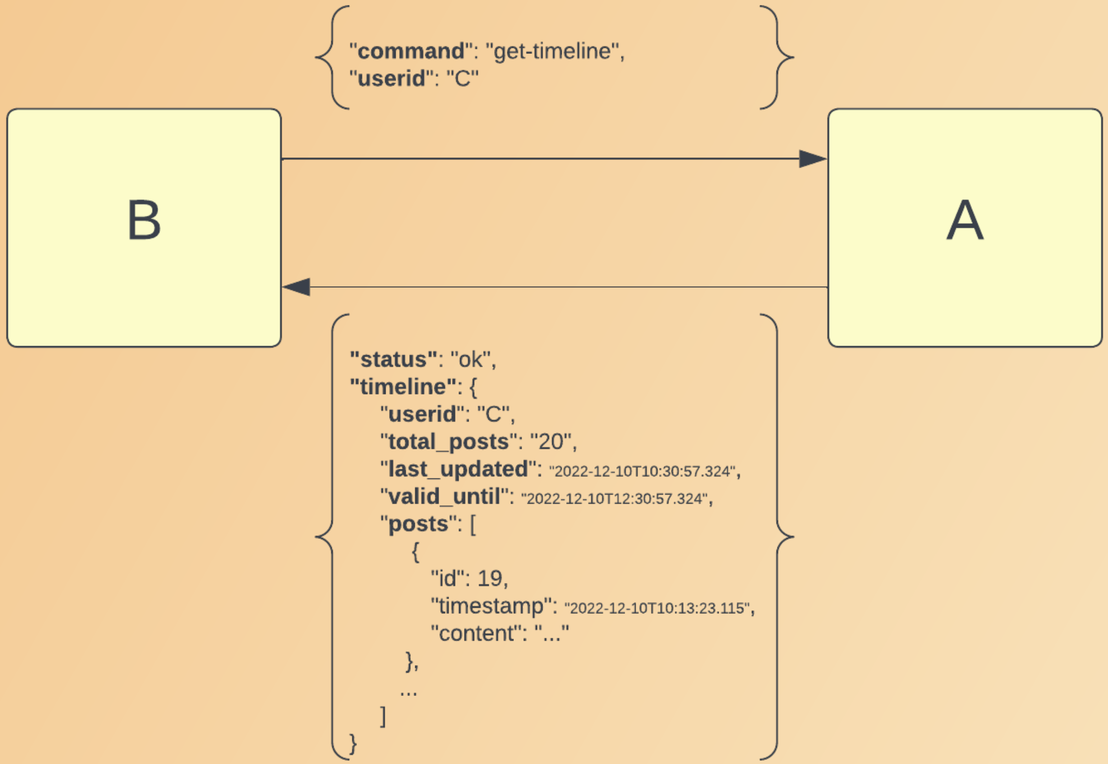

Fig. 2: Example of a request from user B to user *A* and the respective response.

Communication Protocol

L O C A L C O N N E C T I O N

- Used to execute operations in the user's running node
- Requests and responses sent in **JSON**
- Communication over **TCP** socket
- By default communication happens on port 8600

Request

command – one of **"view" "get" "post" "remove" "sub" "unsub" "people-i-may-know"**
userid, max-posts, content, postid, max-users
– **arguments of the command depend on the command**

Response

status – **"ok"** or **"error"**
error – **error message** (if status == "error")
timeline, users, warnings – **depends on the command**

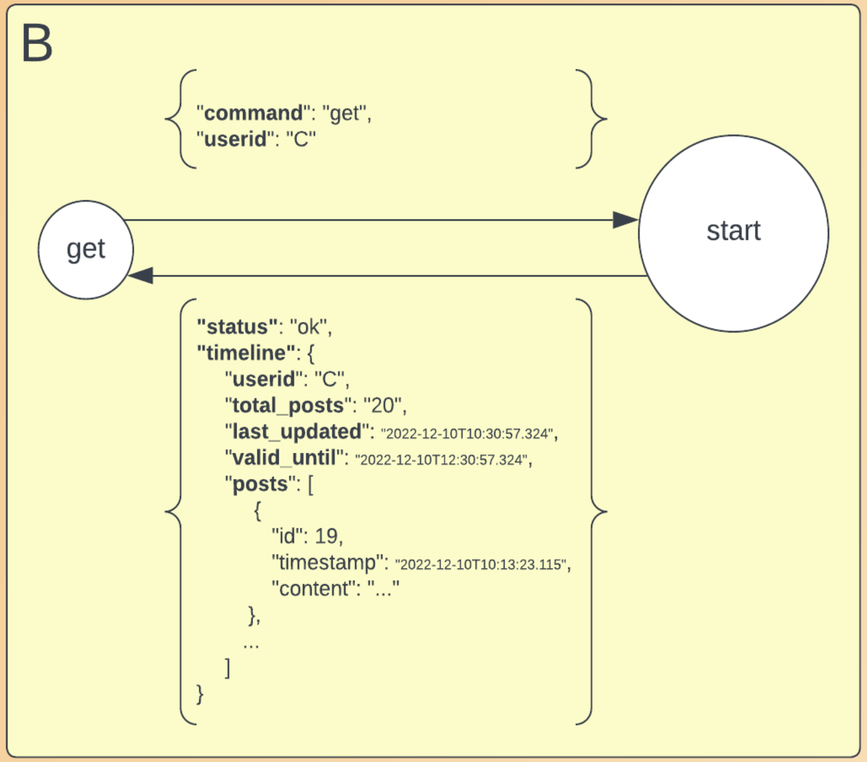

Fig. 3: Example of the communication between the processes "run.py get" and "run.py start" in the computer of the user B in order to execute the **get** command.

Persistent Storage

- Keeps **data stored in disk** to withstand crashes
- Stores **own timeline**, subscribed **timeline caches**, list of **users it is subscribed to** and **counter** for unique post ids

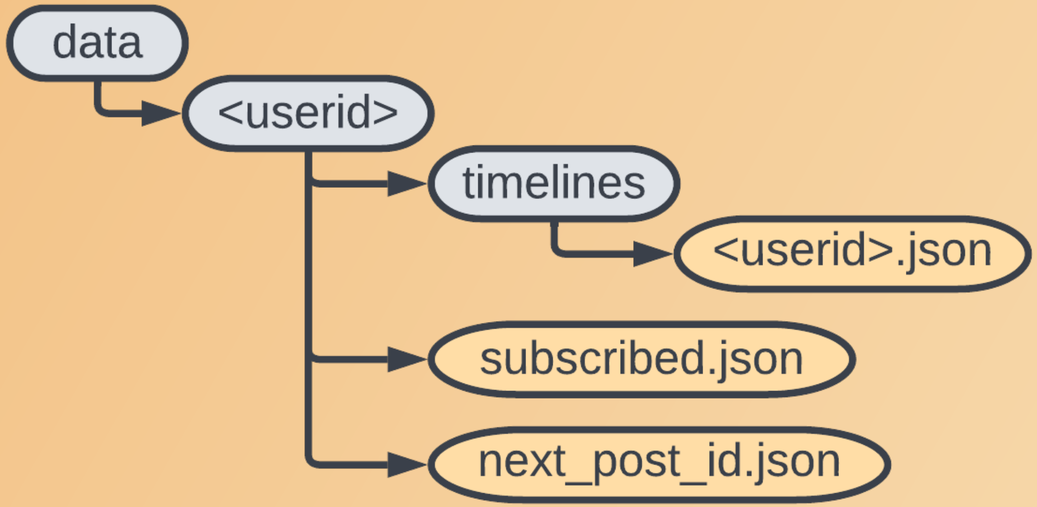

Fig. 4: File structure used to persistently store relevant data.

Periodic Caching

- Every 120 seconds (this period can be configured), the node will try to **update the cache** of subscribed timelines
- A cache may become **invalid after a period of time** (configurable)
- A cache only contains the 20 **most recent posts** (amount of posts is configurable)

For each subscription:

Ensure the user appears as subscribed in the DHT → Tries to retrieve the timeline from the owner **IF FAILS** → Tries to retrieve cache from other subscribers

this recovers from race conditions that may have happened

only accepts caches that are more recent than its own

Retrieve Timelines

1. Locally:

- Local **get** command
- **get-timeline** command from other node (Note: if a non-subscribed timeline is requested, checks DHT to recover from race conditions that may have happened after unsubscribing)

if our timeline was requested

Make cache of owned timeline

OR

if other user's timeline was requested

Return cached timeline from local storage (only if it exists and has not expired)

2. From peers:

- Periodic **caching**
- Local **get** command (if failed to get locally)

Request timeline directly to owner

IF FAILS →

Request timeline from another subscriber

IF FETCHED MOST RECENT TIMELINE
OR WITH PROBABILITY *p* $(p \leftarrow p/2)$

Group T3G14

Conclusions and Future Work

- The **goals** of the assignment were **accomplished**
- Our implementation is **robust** and we **correctly implemented all intended features** given the assumptions specified
- The communication between processes using the local TCP socket permits an **easy extension** with a **graphical user interface**, which is left as a **future work**
- Another feature to implement in the **future** could be **protection** from **malicious writes** to the DHT and **changes to a timeline cache**