Decentralized Timeline

SDLE • Group 13 • Class 2

Diogo Nunes
up201808546

Mariana Soares
up201605775

José Nuno Silva
up201705591

Rita Nunes Mota
up201703964

Description

01

Description

This project will explore the creation of a decentralized timeline service (e.g. Twitter, Instagram, Facebook) that harvests peer-to-peer and edge devices.

Users:

- Have an identity
- Can publish small text messages in their local machine, forming a local timeline
- Can subscribe to other user's timelines and will help to store and forward their content

Remote content is available when a source or source subscriber is online and can forward the information.

Information from subscriber sources can be ephemeral and only stored and forwarded for a given time period.

Observations from social media

1. Content is mostly reactive
    a. Real world events
    b. Other content posted on the social media
2. Users don’t really mind if a post takes a few seconds to go through
3. New content is favored

Design and Architecture

02

Our approach

A decentralized system can’t be built on trust. All nodes in the network should be able to validate the content integrity and authenticity.

*All the content must be **digitally signed***

NFT

System Design

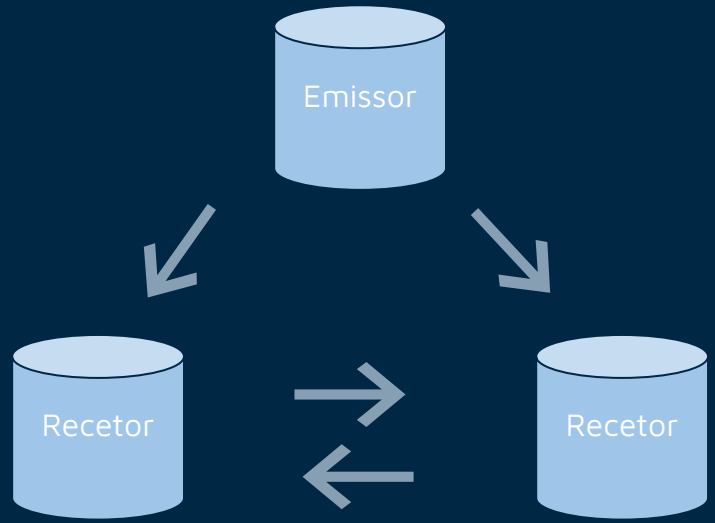
Status Report

- Nodes post their address for content exchange
- Nodes post their current status related to the messages they have
- *All messages are signed and timestamped*

Content Exchange

- Nodes expose an interface to exchange messages
- *All messages are signed and timestamped*

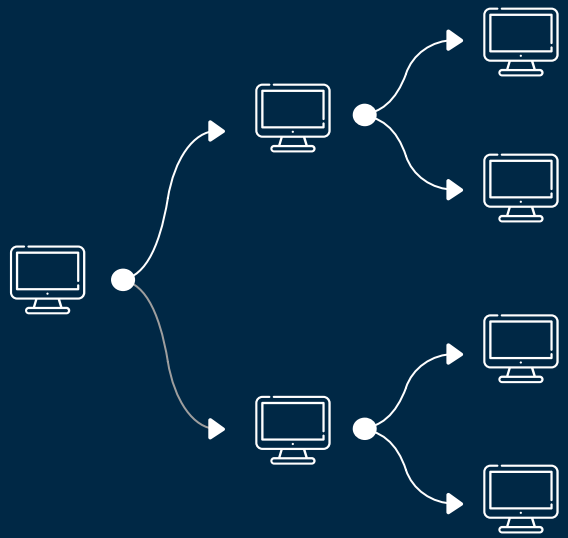
Mutual Effort



Mutual effort to make every message reach the destination(s):

- The message owner sends it to the receptor(s) and they trickle down the message
- The message receptors are always looking for new messages to incorporate

Message publish diffusion



Process of message diffusion:

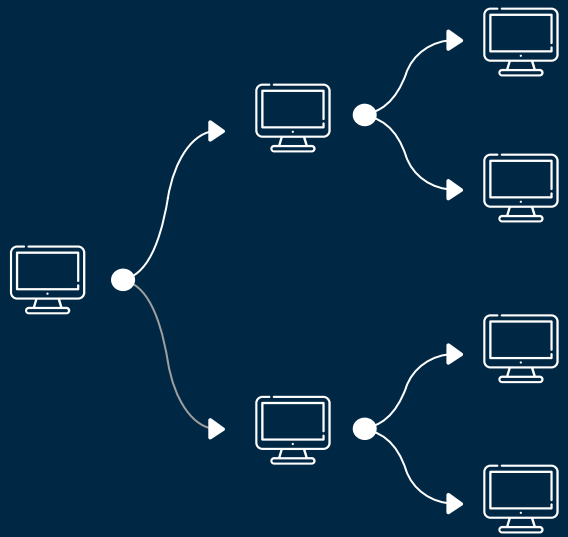
1. Message is published
2. Send it directly to the two nodes with the most up to date messages
3. Receiving nodes wait a randomized amount of time
4. Receiving nodes send the message to the other two most up to date

log2(Followers) = Number of connection levels

log2(60) ~= 6 (Average number of followers of active social media user)

log2(130*10^6) ~= 27 (Most followed person on Twitter)

Total time to get a new message



Time of getting a new message can be defined as:

min(TimeToRetrieve, TimeToTrickleDown)

TImeToRetrieve ~= RetrieveInterval + DHTQueryTime

TimeToTrickleDown ~= log2(1+Order) * (RandomSleep + DHTQueryTime)

Implementation 03

System Design - Implementation

Status Report

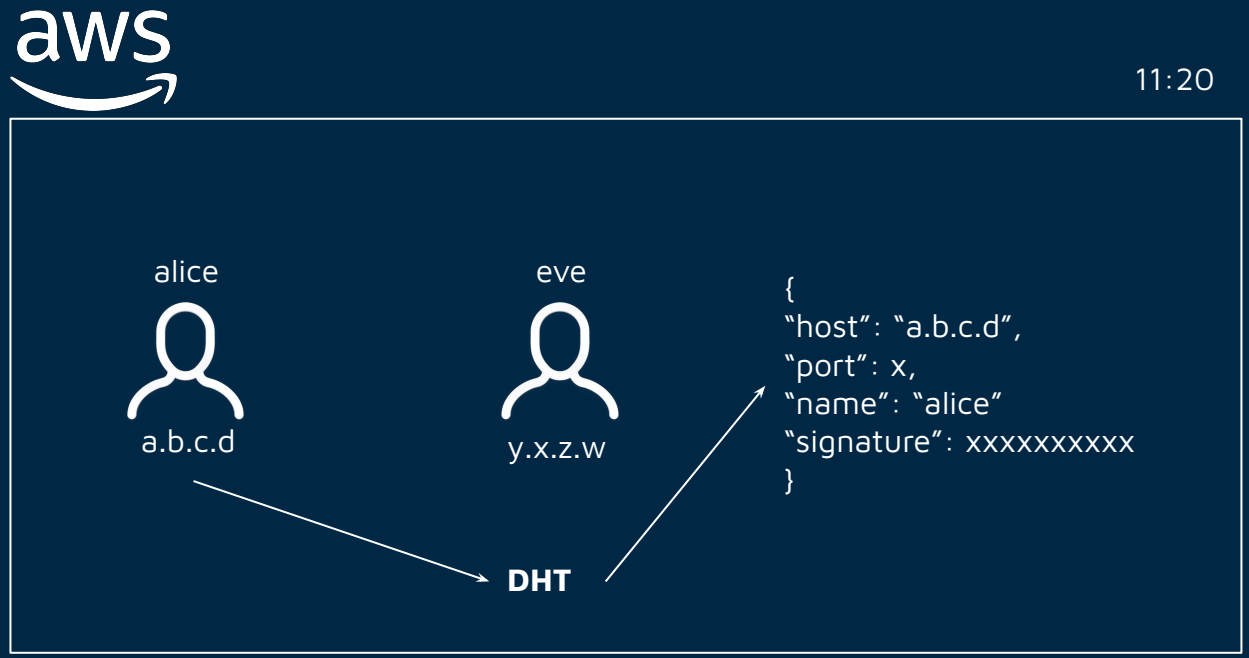
- Nodes post their address for content exchange
- Nodes post their current status related to the messages they have
- *All messages are signed and timestamped*
- DHT
- JSON - Widely supported format

Content Exchange

- Nodes expose an interface to exchange messages
- *All messages are signed and timestamped*
- HTTP - Modern systems that are microservice based use this transport protocol (i.e Twitter)
- JSON - Widely supported format

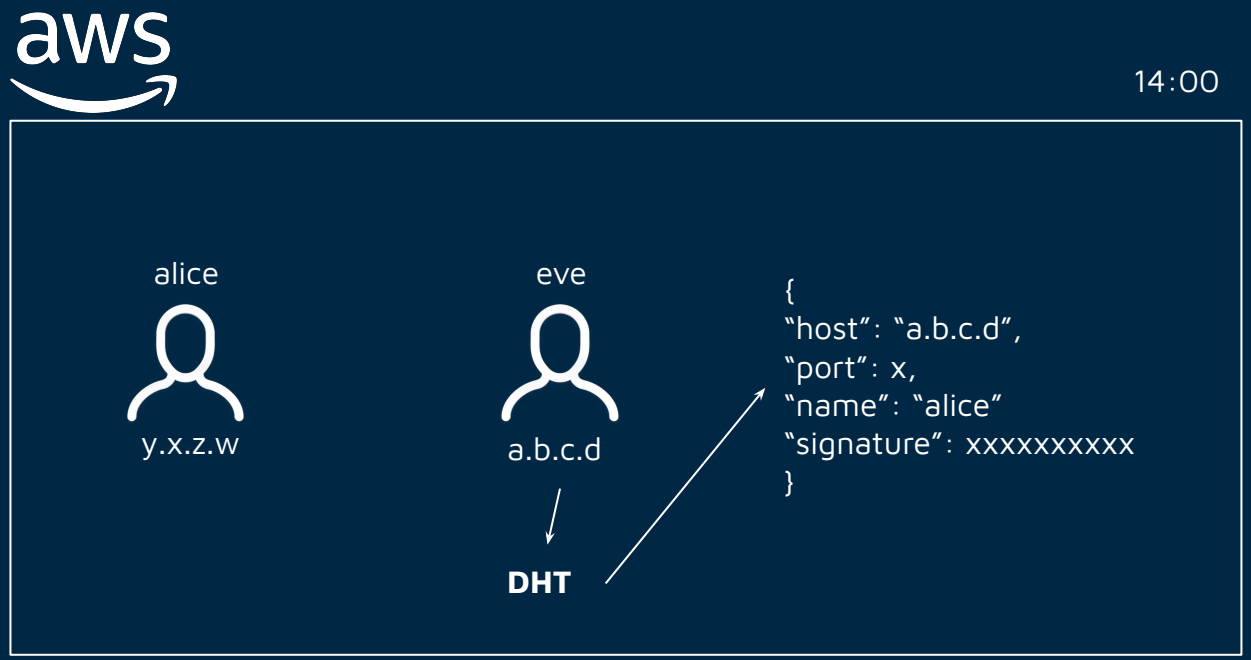
Replay attacks

A malicious actor can reuse old signed messages to perform certain actions.
E.g a node that is using a public cloud provider to host their node might get their IP address reassigned



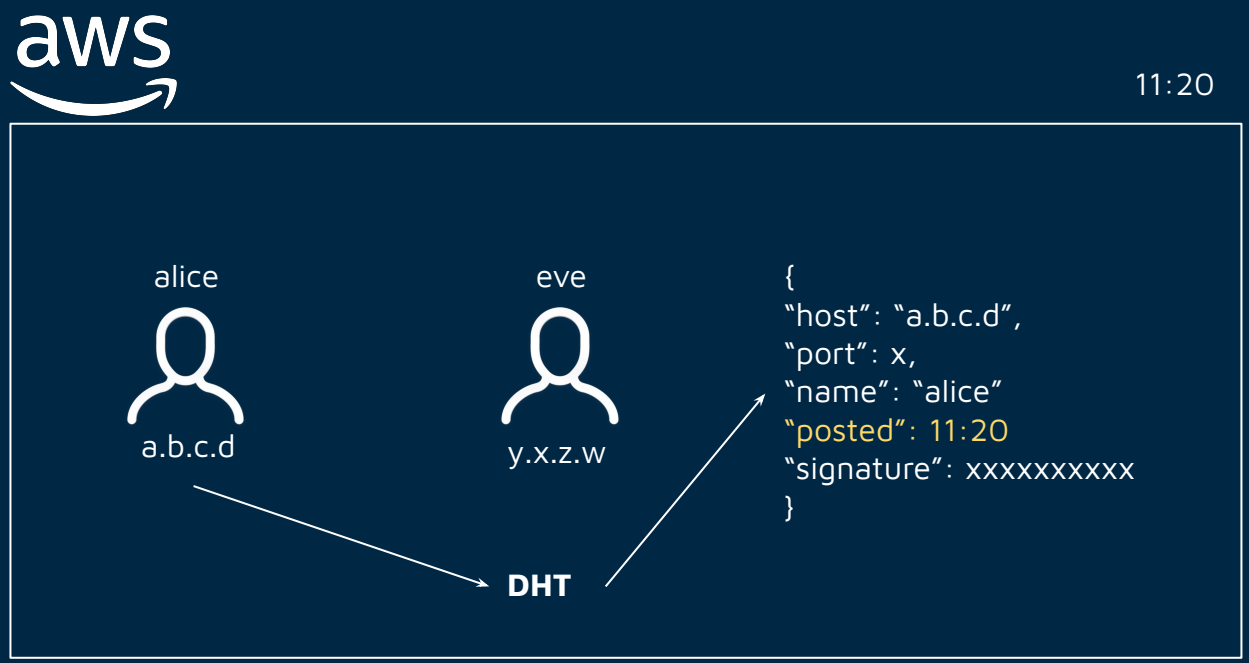
Replay attacks

A malicious actor can reuse old signed messages to perform certain actions.
E.g a node that is using a public cloud provider to host their node might get their IP address reassigned



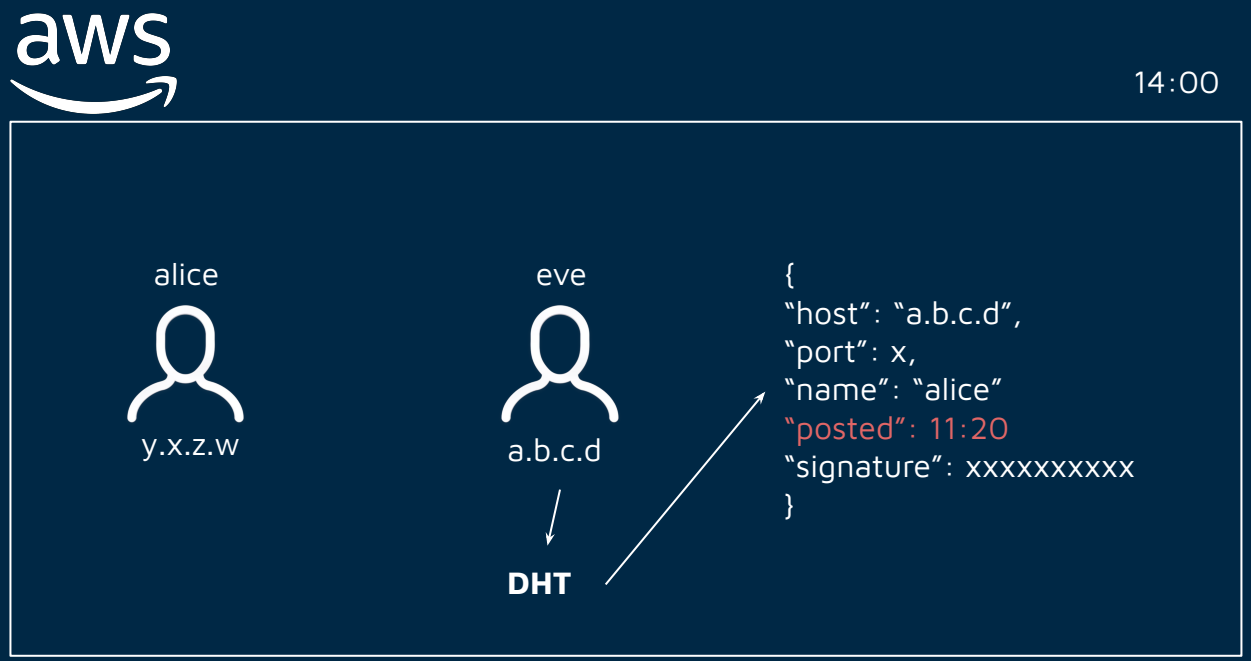
Replay attacks

By adding a timestamp to the message, old messages can be discarded even if the signature is valid.



Replay attacks

By adding a timestamp to the message, old messages can be discarded even if the signature is valid.



Technologies

KADEMLIA

ujson

ntplib

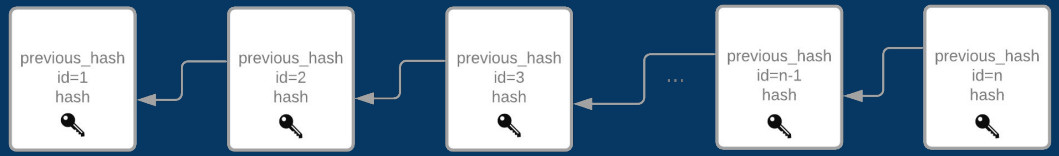
aiosqlite

aiohttp

All the listed technologies can run in an **asynchronous environment**, an important factor in terms of scalability

Message Structure

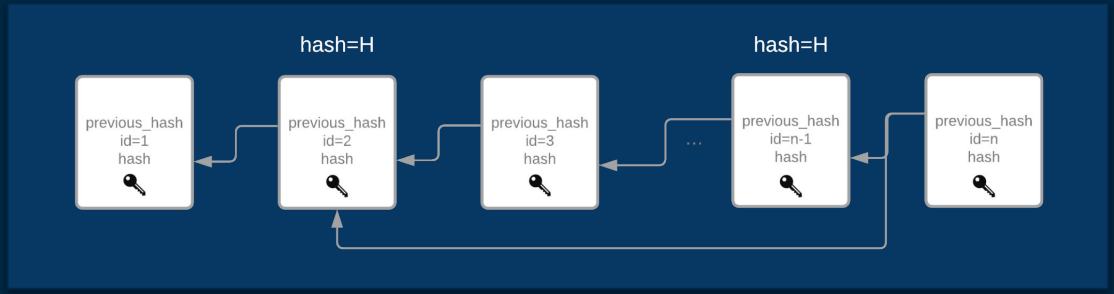
Normal Messages



Ephemeral Messages

Truncation attack

There's a tiny chance of two messages having the same hash, in that case an attacker could truncate the chain of messages and still have a valid chain. Because of this a chain is only valid if the signatures are valid and if and only if the IDs are consecutive.



SQL Schemas

Stored Messages	
message_id	*Id of the message*
publisher	*Who published it*
published	*UTC timestamp of the message*
msg	*Content of message*
until	*Duration of message*
message_hash	*Hash of the message*
previous_hash	*Hash of the previous hash*

Following	
name	*Name of the node we're following*

DHT Content

{node_name}-source:
{
    Host: str - address,
    Port: int - port,
    Posted: int - UTC timestamp,
    Signature: str - signature,
    Latest_normal: int - id of the last normal message node has,
    First_normal: int - id of the first normal message node has,
    Num_ephemeral: int - number of ephemeral message a node has
}

{node_name} publishes this message to announce its presence

DHT Content

{node_name}-replicas:
[
    {
        Host: str - address,
        Port: int - port,
        Posted: int - UTC timestamp,
        Signature: str - signature,
        Latest_normal: int - id of the last normal message node has of {node_name},
        First_normal: int - id of the first normal message node has of {node_name},
        Num_ephemeral: int - number of ephemeral message a node has of {node_name}
    }
]

Replicas publishes these messages their status regarding {node_name} content

DHT Content

Nodes:
[
    {
        Host: str - host address,
        Port: int - port,
        Signature: str - signature
    }
]

Nodes publish here to announce to other nodes they exist

Running jobs

- **DHT announce presence** - periodically the node posts on the ‘nodes’ DHT entry and updates it ‘source’ entry
- **DHT share stats** - periodically shares the node status on the data it is holding of other nodes
- **Get messages** - periodically checks if there are new messages and if there are any it attempts to get them (first from the source then the replicas)
- **Missing fixer** - periodically detects messages that are invalid, and thus break the chain and attempts to retrieve them (first from the source then the replicas)
- **Trim and save** - periodically the message list is persisted to the database and trimmed in memory a size set by the user
- **Persist following** - periodically persists the nodes that we’re following
- **Delete invalid entries** - periodically deletes persisted messages that are corrupted in disk

Parameterization

name: name attributed to the node - required

host_dht: IP address of the distributed hash table - default ‘127.0.0.1’
port_dht: endpoint where the dht is running - first free port assigned

host_comm: IP address of the http communication - default ‘127.0.0.1’
port_comm: endpoint where the http communication is running - first free port assigned

host_boots: IP address of the dht node to which the service connects - default ‘127.0.0.1’
port_boots: endpoint where the dht node to which the service connects is running - default None

max_saved_msgs: maximum number of messages from that node to be kept in memory - default 5
request_timeout: number of seconds before an http request timeout - default 5
replicate_min_interval: minimum number of seconds for a node to replicate a message - default 5
replicate_max_interval: maximum number of seconds for a node to replicate a message - default 10
trim_and_save_interval: time interval in which the messages kept in memory are trimmed and persisted to the database - default 5
delete_invalid_entries_interval: time interval in which messages with an invalid signature are deleted -default 10
dht_share_stats_interval: time interval in which the node updates the dht on what information he has on other nodes - default 10
dht_announce_interval: time interval in which the node must acknowledge its presence - default 10
missing_fixer_interval: time interval in which missing messages are detected and found - default 5
get_msgs_interval: time interval in which a message request is done - default 5
persist_following_interval: time interval in which the nodes being followed are persisted to the database - default 10
remove_entry_seconds: how many seconds after being posted a message can still be accepted - default 40
self_trim: whether a node should trim its own messages - default False

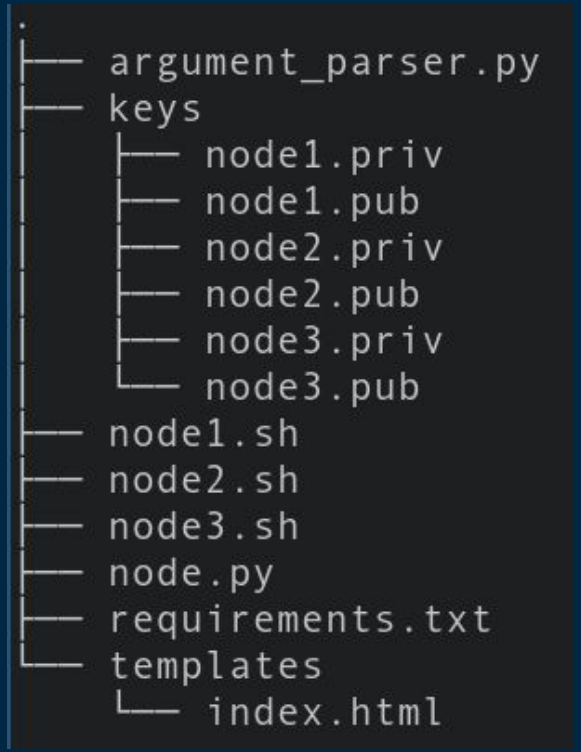
Endpoints

- **/** : User interface
- **/post_msg**: POST method to send a message to another node timeline
- **/get_ephemeral/<node_name>**: GET method to collect all ephemeral messages of “node_name”
- **/get_msgs/<node_name>**: GET method to collect all normal messages of “node_name”
- **/submit_message**: POST method to publish a message in a node timeline
- **/follow/<name>**: GET method to follow a node called “name”
- **/unfollow/<name>**: GET method to unfollow a node called “name”

Results

03

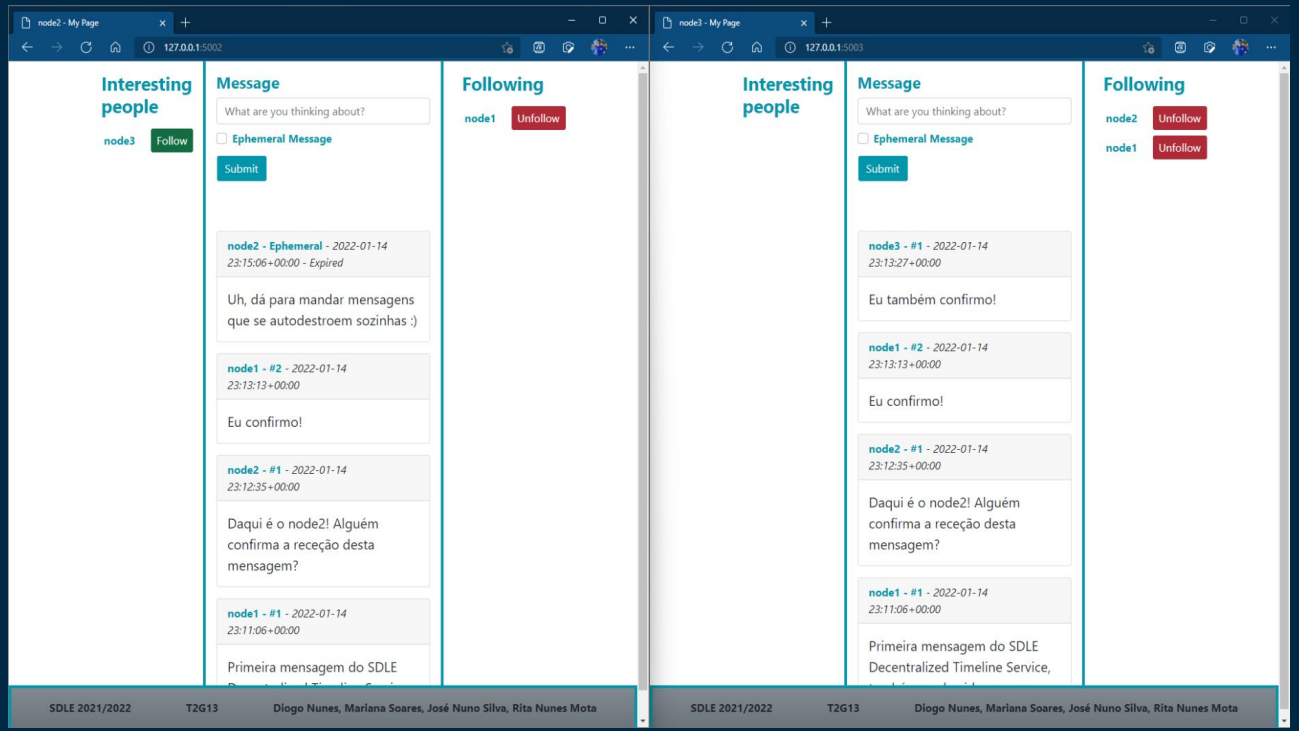
Source code structure



- argument_parser.py - generic and flexible argument parser
- keys/ - contains public and private keys
- *.sh - scripts that initialize a network with three nodes
- node.py - contains the code related to the node
- templates/index.html - main page
- static/styles.css - our css
- requirements.txt - necessary dependencies to install

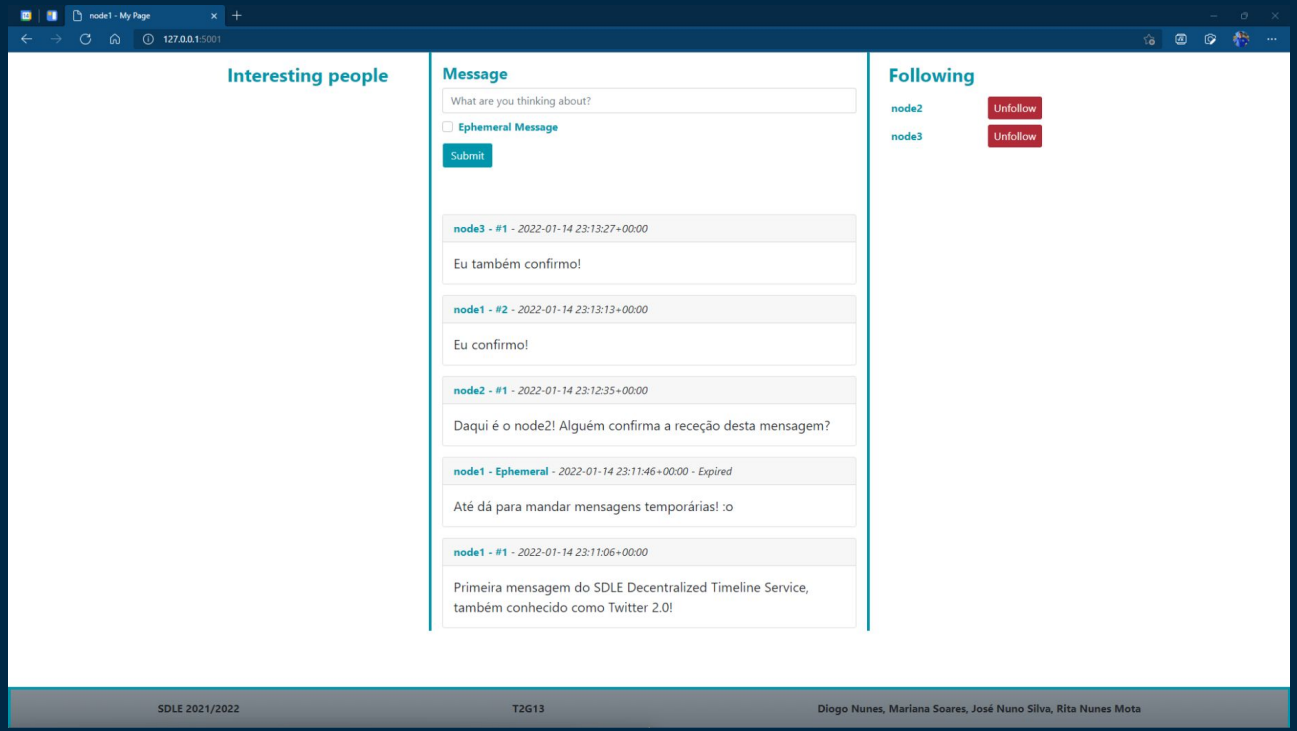
User Interface

Since the nodes contain a HTTP server, it can also **serve HTML** allowing for a user interface in the browser.



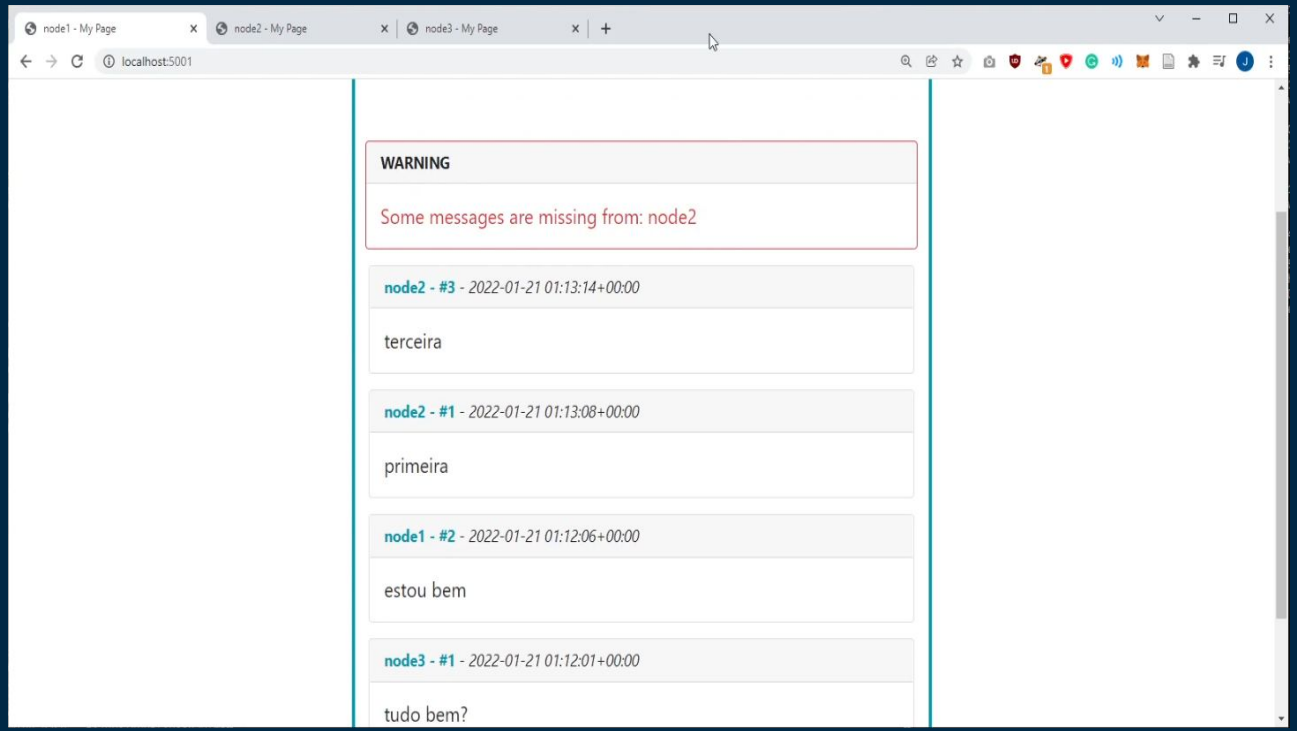
User Interface

Since the nodes contain a HTTP server, it can also **serve HTML** allowing for a user interface in the browser.



User Interface

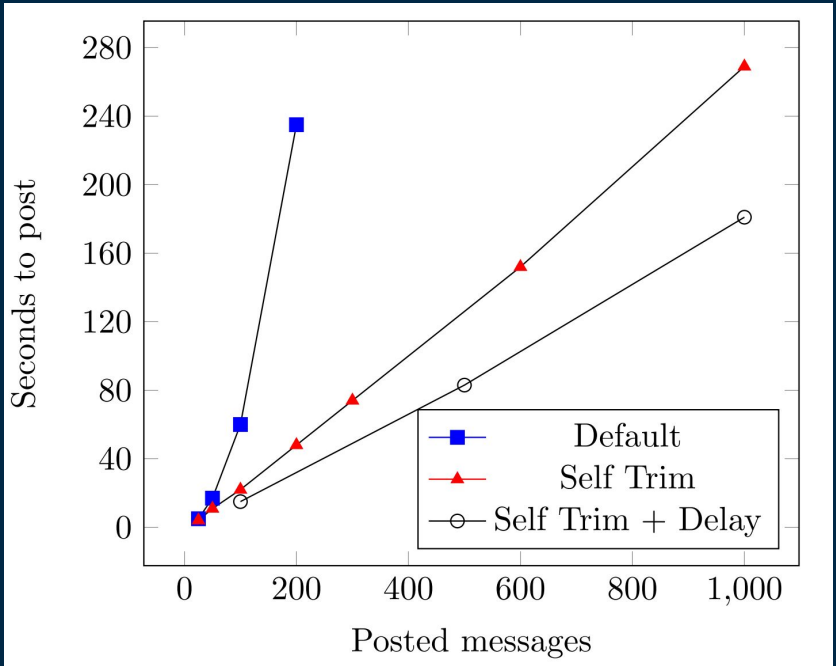
Since the nodes contain a HTTP server, it can also **serve HTML** allowing for a user interface in the browser.



Capturing data

Capturing useful insights on a distributed system is a really hard task because under normal circumstances there is a lot of variability regarding. Due to this characteristic we decided to load test a single node.

Comparison of different settings on load



Messages of 128 characters were generated and posted sequentially to a node under perfect conditions and compared to scenarios of tweaked flags.

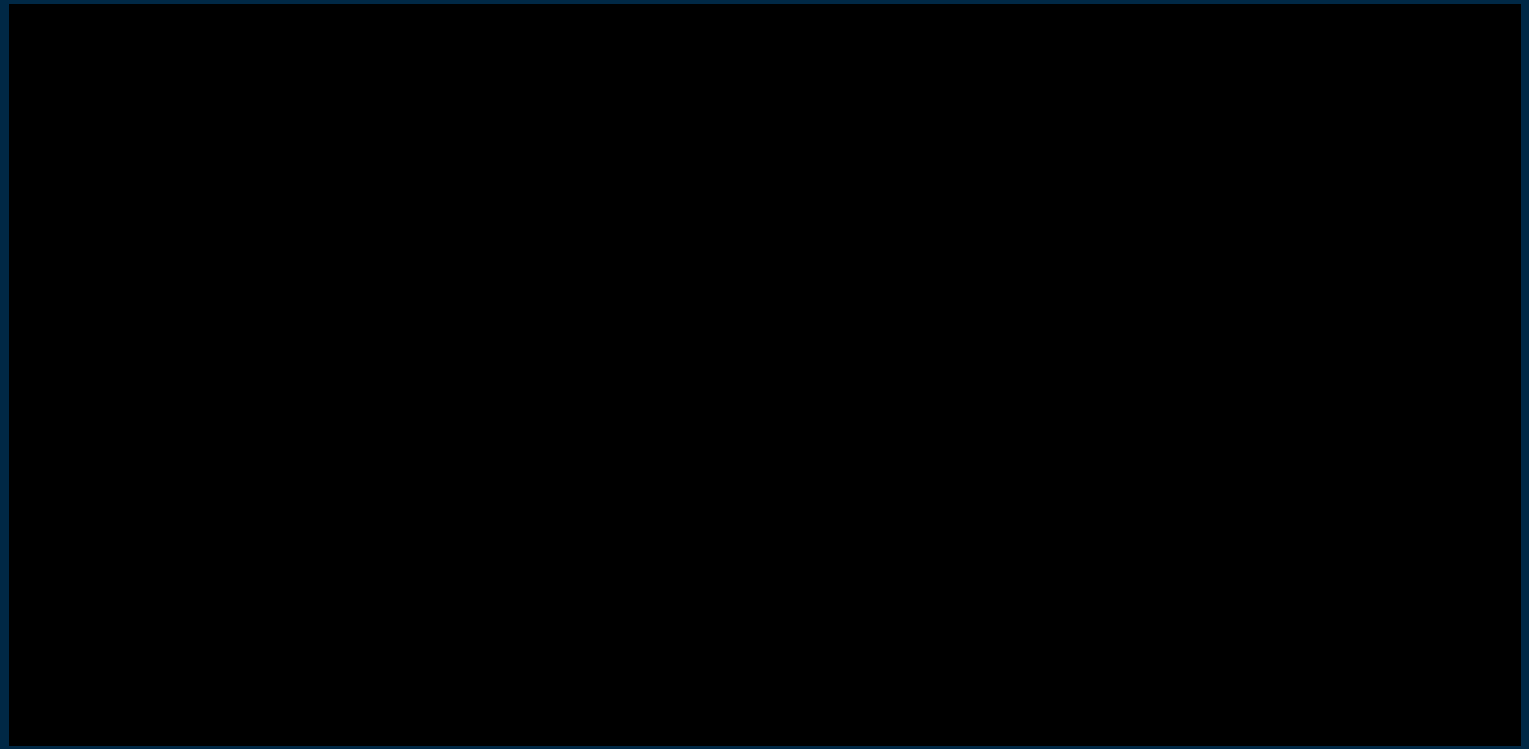
- Self trim - message list from the same node are not trimmed to
- Delay - trim_and_save_interval set to 1 second

Sybil Attacks

Even though most attack vectors have been mitigated, Sybil attacks are still a possibility but its effects are reduced. The worst they can do is **block the communication between the nodes**.

A possible solution for this type of attacks could be applying a seniority factor to decide which nodes to listen to first.

Video



Conclusion

In this project we can conclude that through existing technologies and concepts, we can create an agnostic (programming language wise) decentralized timeline service such as social networks with emphasis in the authenticity, integrity and verifiability.

Do you have any questions?

SDLE • Group 13 • Class 2

Diogo Nunes
up201808546

Mariana Soares
up201605775

José Nuno Silva
up201705591

Rita Nunes Mota
up201703964