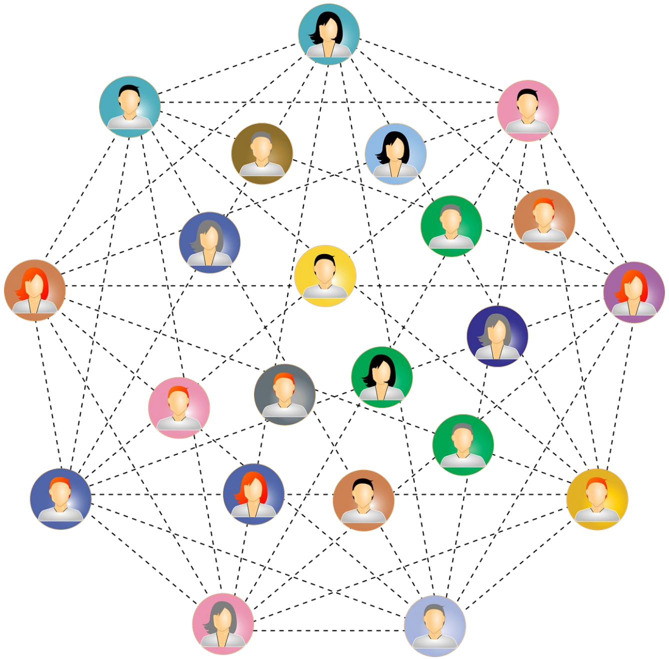
Kinder Bueno

Ana Barros - up201806593
Diogo Rosário - up201806582
João Martins - up201806436
Ivo Saavedra - up201707093

Problem Specification

- Decentralized timeline system
    - Timeline is a collection of posts
- User's identity is their nickname
    - A user can be logged in across different devices at the same time
- Clients may add/update/delete posts of their timeline
- Search timelines/posts by username or content
- Guarantee fair workload across peers
    - Nodes of the network store other timelines/posts ephemerally
- Clients can subscribe other clients
    - Some nodes redirect subscription content

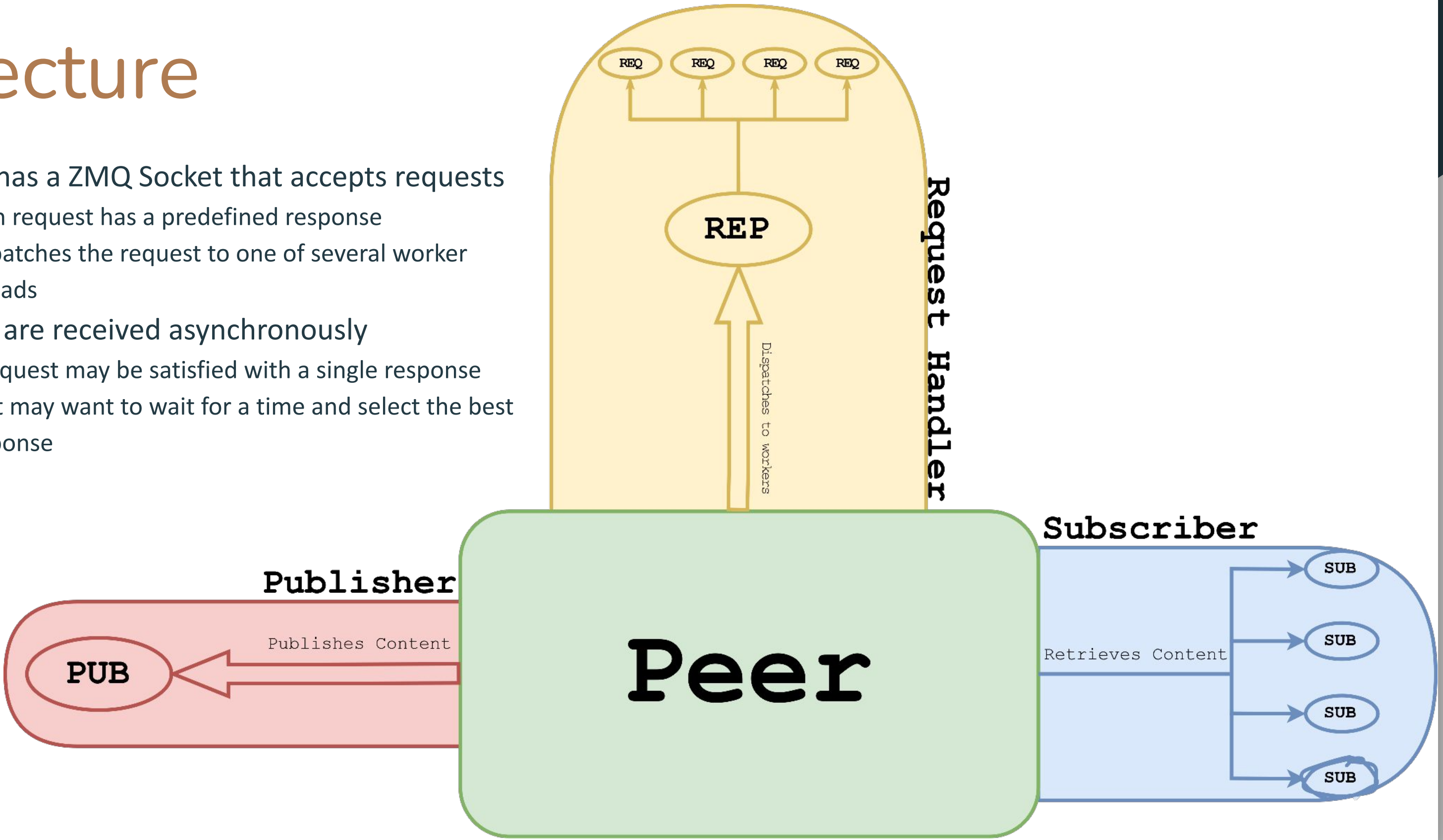


Architecture

- Java and ZeroMQ
- Based on Gnutella[1] and Gia[2]
- Nodes have capacity that represents the number of requests a node can handle.
    - Determined as a function of a client's bandwidth, processing power, disk speed, etc…
    - Higher capacity nodes tend to have more neighbours ⇨ superpeers
- Each node has a satisfaction level that represents how satisfied a node is with its current set of neighbours
    - As long as a node is not fully satisfied, the topology adaptation continues to search for appropriate neighbours to improve the satisfaction level.

Architecture

- Each peer has a ZMQ Socket that accepts requests
    - Each request has a predefined response
    - Dispatches the request to one of several worker threads
- Responses are received asynchronously
    - A request may be satisfied with a single response
    - Or it may want to wait for a time and select the best response



Visualization

- Makes use of GraphStream[3] package for java.
- Useful to represent operations in the network, such as pings and other messages.
- Uses observer pattern, so it can show all changes in real time as they happen.

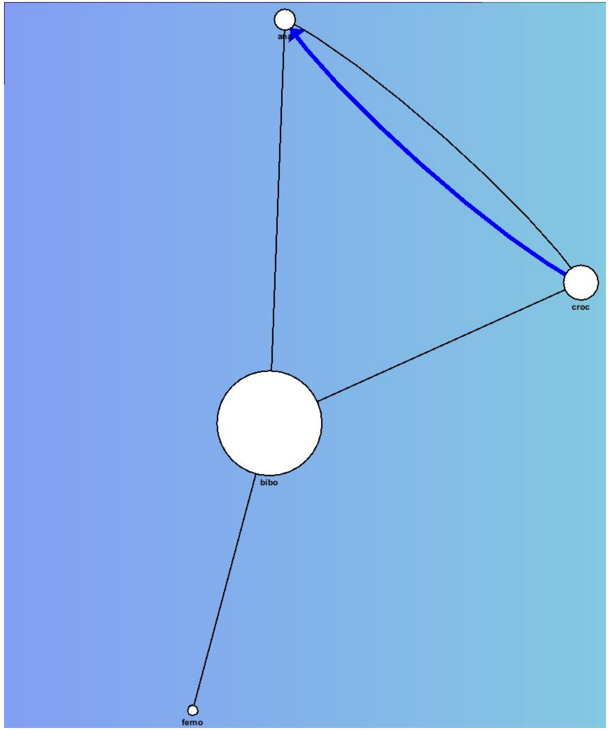


Figure 3 - Graph visualization example

Topology adaptation

- Bootstrapping mechanisms similar to those in Gnutella to locate other nodes.
- Each client maintains a **host cache** consisting of other nodes.
  - Populated throughout the lifetime of the client by exchanging host information with neighbours through **PING-PONG** messages.
- A node periodically pings only its neighbours.
  - To update its knowledge about the network:
    - Find dead/offline neighbours.
    - Find new online peers
    - Share timelines index.

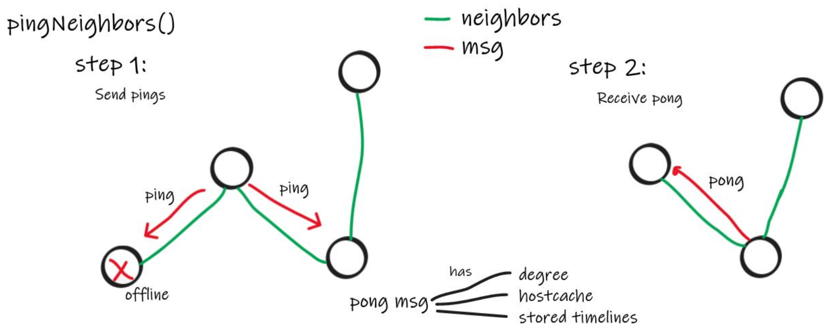


Figure 4- Ping neighbours implementation

Adding new neighbours

- A node periodically tries to improve its neighbour list
  - Uses two-way handshake to verify neighbour status
  - Whilst trying to not bully any low capacity nodes

procedure *addNeighbor*
    $p \leftarrow hostCache.getPeerNotNeighbor()$
    if $neighbours.size + 1 <= MAX_N GBRS$ **then**
        $neighbors.add(p)$ ▷ accept p
        return
    $subset \leftarrow peer.Neighbours()\ where\ C_n < capacity$
    if *subset is empty* **then return** ▷ reject p
    $z \leftarrow subset.max(degree)$ ▷ get node with less capacity and highest degree
    if $p > neighbors.maxCapacity \vee y.degree + hystheresis < z.degree$ **then**
        $neighbors.remove(z)$ ▷ replace z for p
        $neighbors.add(p)$ ▷ replace z for p
        return
    elsereturn

Search Protocol

- Initiator peer sends a QUERY message with the desired content
- Peers respond with a QUERY_HIT if they have the requested content
- The initiator assembles the responses from all peers
- QUERY messages are propagated across the network
  - A message expires when its TTL is reached
  - Simple peers use a random walk: restricted flooding.
- A direct connection is established to reply with an answer
- Based on the topology adaptation
  - High capacity nodes typically provide useful responses for a large number of queries.

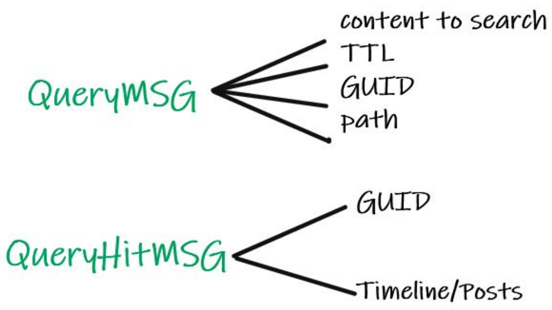


Figure 6- Query/QueryHit message content

Query Types

Timeline lookup

- Initiator asks for a timeline of a user
- Peers that are logged in as the user or have stored the request timeline respond
- Initiator picks the most recent timeline of all received responses
- Initiator stores the most recent timeline temporarily

Post lookup

- Initiator asks posts that match that query
- Peers respond with posts that match the query
- Initiator receives all responses and orders them by time

Bloom Filters

Using Guava's bloom filter implementation[5]

- Each superpeer has its own cluster
    - Super peers mantain a bloom filter for its cluster.
    - The filter indexes usernames
- Upon receiving a query message:
    - A super peer knows if its cluster might have the requested timeline
    - A super peer knows where it should propagate its query:
        - Biased random walks

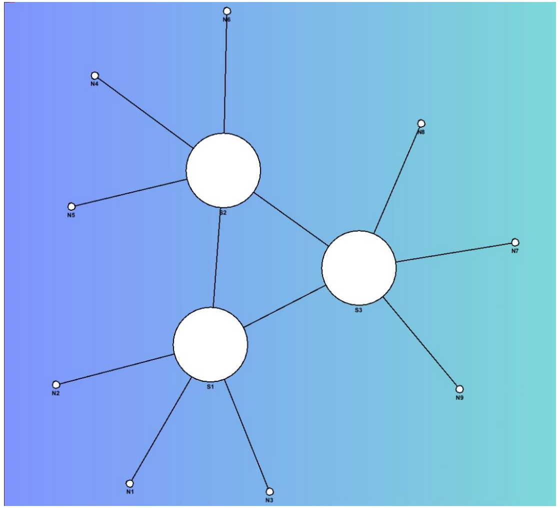


Figure 6- Bloom filter cluster representation

NTP

- Makes use of apache.commons.net package[4] for java.
- Each peer makes a request to a NTP server in order to get its local clock offset.
  - Each peer connects to one of several NTP servers
- Important because it makes sure all peers and their timelines are synchronized with each other.
- All timestamps of posts and timelines are calculated using this offset
  - Timestamps are used to choose the most recent content accurately

```
Local clock offset in ms: -6000
```

Figure 7 - Local clock offset example

Authentication

- Peers have a username-password pair, so they can register/login.
- DSA (Digital Signature Algorithm) signs each post/timeline if the user is authenticated using the private key.
- Peers have a private key for them and a public key for all other peers.
- Requires an authentication service, which unfortunately adds a point of centralization to the project.
  - But multiple authentication services can be used simultaneously between peers
- Any Peer can verify the authenticity of any post/timeline as long as it can get connect to an authentication server

```
Posts:
    {1=     ID: 1:
    user: bibo
        Timestamp: 14:29:11.310394531
        Content: 'hello world'
Verified:
    true
Sign:
    [48, 61, 2, 28, 35, -76, -18, -126, -59, 56, 111, 75, -58, 29, -71, -98, 78, 122, -106, -71, 22, 1, 109, 98, 10, -78, 36, -32, -21, -70, -10, 11, 2, 29, 0, -84, 0, -59, -44, 111, -86, 73, -15, 10, -44, 34,
-37, 24, -22, 28, -81, 9, 47, -120, -27, -104, 115, 111, 119, -39, -19, -73, 99]}
```

Figure 8 - Signed post example

Subscription Service

- An initiator peer sends a **query** message with the username of the desired subscription
- The target or one of its subscribers receives the message:
    - If it has capacity to handle subscriber then sends a **sub hit** message with its **PUB** socket information
    - If it doesn’t have capacity it ignores the message
        - Another available peer should respond with a valid subscription
- The initiator receives a sub hit, opens a **SUB** socket that connects to the publisher
- Any node is able to redirect posts received by its subscriptions
    - It may respond to Sub requests to users that it is subscribed to, if it has enough capacity
- Nodes heartbeat their subscription
    - And request new subscriptions when necessary

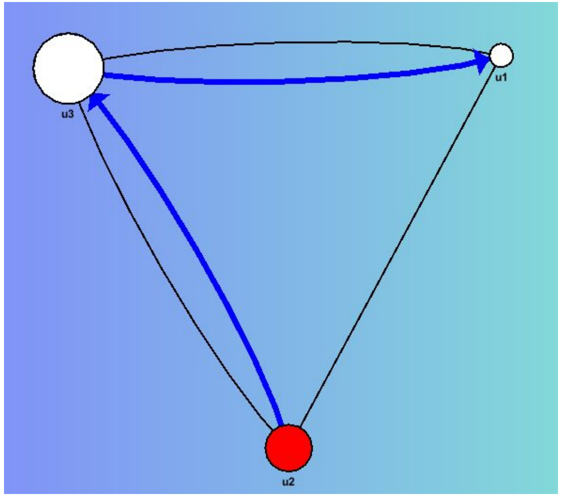


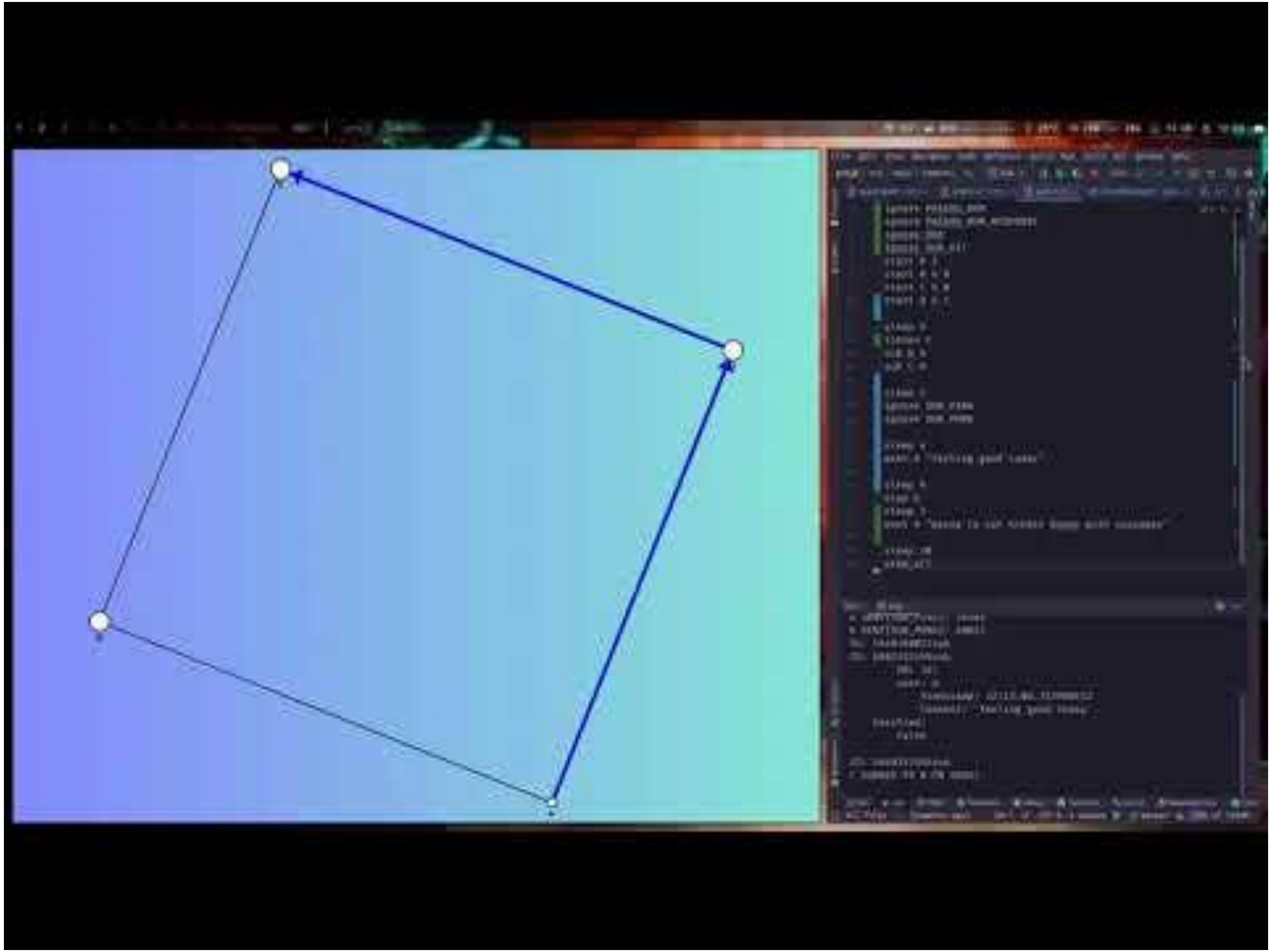
Figure 9 - Redirect posts example

Test Application

- A Test Application tests all the functionalities of the project more extensively.
    - Executes a set of peers
    - Displays the graph visualization of peers and their communication
    - Can display posts/timelines received by peers in stdout
- Makes use of one config file.
- Configuration file is read line by line, each line representing a command.
- Has a live test mode which allows the execution of manually inputted commands.

```
START <username> <capacity> [<join_peer_id>]
START_MULT <n>
POST <username> "<content>"
UPDATE <username> <post_id> "<content>"
DELETE <username> <post_id>
TIMELINE <username> <req_timeline>
SUB <username> <target_username>
IGNORE <message>
MSG_DELAY <value>
LISTEN <username>
PRINT <username>
AUTH
LOGIN <username> <password>
REGISTER <username> <password>
LOGOUT <username>
PRINT_PEERS
STOP <username>
STOP_ALL
BREAK
SLEEP <seconds>
```

Snippet 1 - Available operations



TTL in messages

- What is the optimal TTL value for query messages?
  - Tradeoff between network congestion and result accuracy
  - For this example the optimal value is 2-3, but depends on topology

Hop Count	Precision	Recall	Sent messages
1	25%	11%	8
2	33%	50%	27
3	38%	83%	39
4	41%	100%	44

Table 1 - Message hop count evaluation results

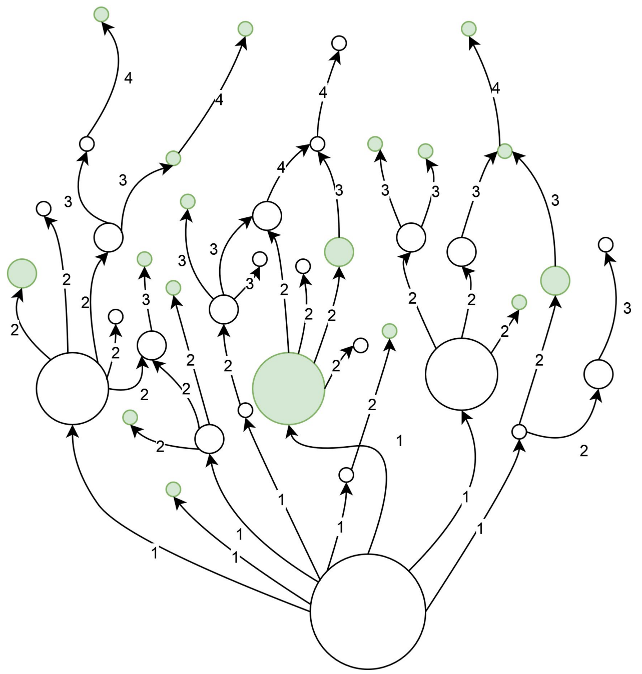


Figure 10 - The tested system

Future Work

- Improve cluster generation with superpeers
  - By avoiding bloom filter overlap
- Decentralize post content indexing across peers
- Evaluate system’s performance, scalability …
- Avoid overloading nodes with query messages through the use of tokens[2]
- Blocking users
- Decentralize authentication service
  - Implement protocol to communicate between servers
  - Decentralize authentication completely[6]

References

1. Clip2, The Gnutella Protocol Specification v0.4
2. Making Gnutella-like P2P Systems Scalable (Gia)
3. GraphStream
4. Apache Foundation, NTP Documentation
5. Google, Guava's BloomFilter implementation
6. A Blockchain-Based PKI Management Framework