Molater

Peer-to-Peer Social Network akin Twitter

T3G11

[1] – Photo by nickdean

Project description

- Social network similar to **Twitter**;
- Users can **publish** text posts on their timeline;
- Posts can contain **HashTags** to enable topic discovery;
- **Following** users adds their posts to your timeline;
- A user can have his content forwarded by his subscribers (isn't required to be online to have his content read);
- The timelines should always be up-to-date.

Technologies

Technologies

OpenJDK	01	**OpenJDK** Compiler/Programming language.
ØMQ	02	**ZeroMQ** Communication with the key-server
Guava	03	**Guava** Bloom filter & Min-max priority queue
SQLite	04	**SQLite** Key caching & User content saving
APACHE LUCENE	05	**Apache Lucene** Tokenization and searching
POWERED BY APACHE	06	**Apache NTP** Synchronize post times
JAVA SWING	07	**Swing + MIG Layout + FlatLaf** GUI

Core ideas

Design choices

- **No DHT**s, because **content discovery** (keyword matching) is significantly **harder** to achieve;
- Be **decentralized** to the point where any user can run the program by just opening it (**no server setup needed**);
- **Authenticity** of every message is important to provide trust;
- Only store post content in persistent storage (**no temporary files**);
- Be **resource efficient**;
- Everyone should contribute what they can (higher capacity nodes contribute more).

Based on Gia[2]

Gnutella but better.

- Better scalability;
- Dynamic periodical topology adaptation;
- Flow control;
- Query fair-queuing;
- One-hop replication of content index;
- Query (Biased) random walks (instead of flooding);

[2] – Yatin C. Sylvia R., Lee B., Nick L., Scott S.: **Making Gnutella-like P2P Systems Scalable**

Joining the network

1º

Bootstrap Node

At the start of the program, the peer connects to one well-known key-server.

2º

Authentication

Users can register themselves if they do not have an account, or log into their accounts.

3º

Keys

After authentication/registration, the user has access to their public/private key pair.

4º

Connect to Neighbors

Using the key-server's neighbor list, the peer fills its hosts-cache.

Authentication and security

- **Bootstrap** nodes also serve as **key-servers**;
- New users register themselves on a **key-server**, by providing a **username** (**unique**) and **public key**;
- On the network, each post travels encrypted by the **author's private key** (**authenticity** and **integrity**);
- All messages containing posts where the public key can't be found/**doesn't match** are **discarded**;
- This makes it so only registered users can participate in the network.

Topology adaptation

Topology adaptation

1. Nodes fill their **host cache** using the responses to their PING messages – the neighbors of their neighbors.
2. The topology change routine runs in delays based on the node's **satisfaction** (**[0, 1]**) and an aggressiveness factor.
3. If the node isn't satisfied (**1.0**), it tries to find a new neighbor from its **host cache**.

Algorithm 1 Calculate node satisfaction

```
1: procedure SATISFACTION(X)
2:     if num_nbrs_X < min_nbrs then
3:         return 0.0
4:     total ← 0.0
5:     for N ∈ neighbors(X) do
6:         total ← total + C_N / num_nbrs_N
7:     S ← total / C_X
8:     if S > 1.0 or max_nbrs ≤ num_nbrs_X then
9:         return 1.0
10:    return S
```

Algorithm 2 Select node to become our neighbor

```
1: procedure SELECTNEWNEIGHBOR(X)
2:     candidates ← HostCache_X   ▷ Select a small alive subset
3:     Y ← maxCapacity(candidates)   ▷ The highest capacity
4:     if Capacity_Y < Capacity_X then
5:         Y ← Random(candidates)
6:     return Y
```

Topology adaptation – neighbor drop

- I when a node has reached its **maximum neighbor capacity**, it is **satisfied**;
- It **can still adapt its topology** in response to incoming neighbor requests;
- When a neighbor request is received, it **may be desirable to drop a neighbor** (worse than the new one) instead of rejecting the request;
- A node **may reject being dropped** in order to not be alone in the network;
- In reality, nodes reject being dropped when their neighbor count would go below a certain threshold (**minimum neighbor count**).

Flow control

Flow Control

- A **Start-Time Fair Queuing** based algorithm is used to schedule the processing of incoming queries;
- Each query to be processed is assigned a **start and finish tag**. These tags depend on the capacity of the neighbor, size of the query, and the "**virtual time**" of the node;

Algorithm 1 Handle new query

1: **procedure** HANDLEQUERY(Q, T, N)
2: **if** $node\_busy$ **then**
3: $X \leftarrow queryInService(T)$
4: $virtual\_time \leftarrow start\_tag_X$
5: **else**
6: $virtual\_time \leftarrow max\_finish\_tag$ ▷ Maximum finish tag of a processed query
7: $start\_tag \leftarrow max(virtual\_time, prev\_finish\_tag)$
8: $finish\_tag \leftarrow start\_tag + \frac{size_Q}{capacity_N}$

- **Virtual time** – time based on the current query being served by the node or the maximum finish tag server.
- Incoming queries are inserted into the **priority queue** (ordered by **start tag**) corresponding to that neighbor (**flow**). Queries from the various flows are handled based on which has the **earliest start time**.
- Priority is given to idle flows, and flows from neighbors with higher capacity.

One-hop replication

One Hop Replication

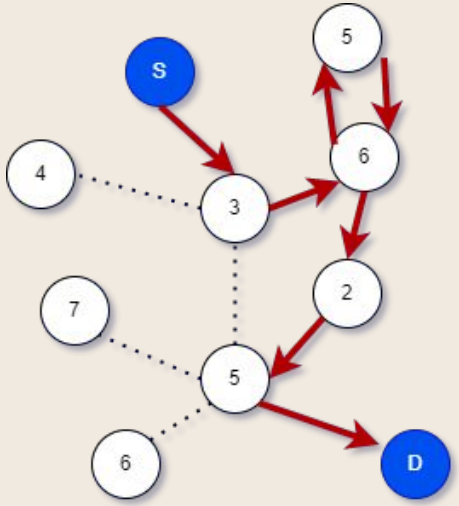
- Each node indexes information about their neighbors' contents;
- This information is shared in **bloom filters** containing: user **content**, user **subscriptions**, and **tags** that the user has posted on;
- Bloom filters are kept with around 1% error chance[3], so any hit is likely to be true.
- This information is synced by PING/PONG messages;
- When a neighbor is **lost**, information about it is **discarded**;

[3] – September 30, 2010 - https://thenextweb.com/news/twitter-statistics-82-of-twitter-users-have-less-than-350-followers

Search protocol

Biased Random Walk

- Queries are identified by a **GUID**, and expire on a given **TTL** or when the **number of required hits** is reached;
- Since the topology adaptation guarantees a high connectivity between high capacity and high degree nodes, it is likely that these nodes can provide answers to a higher number of **QUERY** messages;
- Queries are **redirected** to nodes with **likely matching content**, prioritizing **higher capacity** nodes. If no peer matches, queries are redirected to the highest capacity neighbor;
- If a node is propagating a query that it has already propagated, it will **send it to a different node**. In case the query has been forwarded to all neighbors, this information is reset.



Query types

- **USER –** A query that **fetches the posts of a specific user**. Used to update our timeline with new content from our subscriptions:
  - Contains the date of the latest saved post from that user;
  - All posts fetched will be newer than the date specified (**prevents receiving duplicate posts**).
- **TAG –** A query that **searches for a tag** in the posts of all users:
  - Enables **content discovery**;
  - A tag is represented by a word that follows a **#** (for example, posts with *#dog*);
  - Searching for ***dog*** would yield posts containing any ***#dog*.**

Thank you for your attention!

Now onto the video demonstration

Davide Castro – up201806512
Henrique Ribeiro – up201806529
João Costa – up201806560
Tiago Silva – up201806516