CS 550: Advanced Operating Systems

Work realized by

Florentin Bekier
Rémi Blaise

Gnutella P2P File Sharing System: Design Document

Taught by
Dr. Zhiling Lan

Master of Computer Science, Spring 2020

Contents

1 General architecture

The Gnutella P2P File Sharing System is an extension of the previous P2P File Sharing System made for the first programming assignment. Therefore, the general architecture and specifications are the same as before. In this document we will detail the changes between the first version and this one.

1. The **Super-Peer** centralizes an index of all available files in its leaf nodes. Each running instance of the Super-Peer creates a new independent file-sharing network and can communicate with the other super-peers.
2. The **Peer** can download and serve files from/to the network. Each running instance is a node of the network and is a leaf-node of a Super-Peer.

Since the network has to be static for this assignment, we removed the UUID of the peer and instead we identify the leaf nodes by an index value on the super-peer.

To simplify the deployment of a network with several super-peers, we created an installer software that can generate as many super-peers and leaf-nodes as we want according to either the all-to-all or linear topology.

2 Technology specifications

The technologies used for the software are the same as before. However, we removed the SQLite database because it had become unnecessary as the leaf-nodes and super-peers list is static. To store the list of files we chose to simply use an array so we don't have persistence anymore but we make sure that the data are up-to-date after each restart of a Super-Peer.

3 Communication protocol

The communication process is also similar to the previous one. We still use TCP to send JSON-formatted data through a socket. However, there are now 3 communication modes instead of 2: **Super-Peer - Peer**, **Peer - Peer** and **Super-Peer - Super-Peer**. There are also some changes in the **API procedures** to implement the new procedures for the communication between super-peers. Those changes are detailed in the next section.

4 Design of the Super-Peer

The Super-Peer is built with the following software design:

- The `app.js` file is the entry point of the software and instanciates the TCP server.
- The `procedures.js` file defines the actual procedures provided by the Super-Peer API.
- The `repository.js` file handles the persistence layer by handling the arrays that save the data.
- The `interface.js` file implements the client communication protocol.
- The `search.js` file handles the file search (local search and propagation).
- The `config.js` file reads the configuration.

4.1 API

Using the previously defined protocol, the Super-Peer provides the following API:

4.1.1 The `registry` procedure

Enable a peer to register as a *server* in the network, or to update its server information.

A server information is made of:

- its contact address (IP and port),
- its file list,
- the SHA-256 signature of the request certifying the identity of the peer.

Expected request:

```
{
    "name": "registry",
    "parameters": {
        "ip": "string",
        "port": integer,
        "files": [
            {
                "hash": "SHA-1(content)",
                "name": "string",
                "size": integer
            },
            ...
        ],
        "signature": "string"
    }
}
```

Expected response:

```
{
    "status": "success",
    "data": null
}
```

4.1.2 The `search` procedure

Enable a peer (a leaf node or another Super-Peer) to search for a file by name in the network. The response to the request is just an acknowledgment of the request processing, the Super-Peer will then send a `queryhit` to the sender of the request for each hit and propagate the `search` request to its neighbors. When received this request, the Super-Peer checks if it has already handled and if not, performs the following actions:

1. Start a local search of the requested file

2. Log the message received (message ID, IP and port) in order to backpropagate the future hits
3. Propagate the search request to its neighbors (if the TTL is more than 0)
4. Flush old messages (timestamp more than 1 second old)

Expected request:

```
{
    "name": "search",
    "parameters": {
        "messageId": "string",
        "fileName": "string",
        "ttl": integer,
        "ip": "string",
        "port": integer
    }
}
```

Expected response:

```
{
    "status": "success",
    "data": null
}
```

4.1.3 The `queryhit` procedure

If a Super-Peer receives a `queryhit`, it will backpropagate the request to the sender of the original `search` request. To find the sender information, it will use the message ID to look into the message history and find the orginal message data. Note that, this time, `ip` and `port` correspond to the result to backpropagate and not to the sender information.
Expected request:

```
{
    "name": "queryhit",
    "parameters": {
        "messageId": "string",
        "fileId": "string",
        "ip": "string",
        "port": integer
    }
}
```

Expected response:

```
{
    "status": "success",
    "data": null
}
```

4.2 Configuration

The configuration file of the Super-Peers contains the static information of its leaf-nodes and neighbors (`leafNodes` and `neighbors` keys). It also contains a lifetime value to flush the messages logs and other parameters (the same as in the previous assignment).

5 Design of the Peer

The Peer software follows the given design:

- The `app.js` file is the entry point of the software and starts the Peer client and the Peer server.
- The `client.js` file implements the client side: it registers the Peer to the Super-Peer, watches the shared folder and shows the CLI.
- The `server.js` file implements the server side: it instanciates the TCP server and listens for file requests.
- The `interface.js` file implements the communication protocol.
- The `files.js` file handles the file sharing management.
- The `config.js` file reads and initializes the configuration.

5.1 API

Using the previously defined protocol, the Peer provides the following API:

5.1.1 The `retrieve` procedure

Enable another peer to download a specific file. Return the file information and data as a stream.

Expected request:

```
{
    "name": "retrieve",
    "parameters": {
        "fileId": "string"
    }
}
```

Expected response:

```
{
    "status": "success",
    "data": {
        "filename": "string"
    }
}
```

5.1.2 The `queryhit` procedure

After performing a **`search`** request to its Super-Peer, the Peer will listen for incoming **`queryhit`** requests that will deliver the result. The listening time is specified in the configuration file.
Expected request:

```
{
    "name": "queryhit",
    "parameters": {
        "messageId": "string",
        "fileId": "string",
        "ip": "string",
        "port": integer
    }
}
```

Expected response:

```
{
    "status": "success",
    "data": null
}
```

5.2 Command-line interface

The user of the Peer software has access to the following actions:

1. List the current shared files served to the network.
2. Search by name for a file to download. List all corresponding files with ID and size. The user can then select the file he wants to download or return to the menu.
3. Exit the program.

To choose an action, the user enters the number corresponding to that action using its keyboard and press Enter to validate.

6 Security design

From a security point of view, the system could be vulnerable to the following attacks:

1. A malicious tier sends faulty request data to make the server-Peer or the Super-Peer crash.
2. A man-in-the-middle corrupts the file shared by the server-Peer to the client-Peer.
3. A malicious tier registers erroneous server-Peer information to the Super-Peer.

To address these vulnerabilities, we carefully designed the following defenses:

1. Validate any data received from a client so it doesn't make the server crash.

2. Identify the files by their SHA-1 hash so the client-Peer can verify the file is not corrupted after downloading it. If the hash doesn't match, the file is deleted.

3. Provide a public key (statically stored) and cryptographically sign any further call to the `register` procedure.

7 Possible improvements

To go further, the following improvements could be brought to our P2P system:

- A GUI could replace the current CLI of the Peer, making it usable by common users.
- The downloading of big files could use several server-Peers if available. Similarly, the Super-Peer could sort available server-Peers to better distribute the global charge over all servers.
- The Super-Peer could include a web-browsable interface listing available files on the system.
- The Super-Peer could reduce the risk of trying to contact an unavailable file server by periodically checking the availability of hosts or removing servers from the list after a given timeout.
- The protocol doesn't implement any incentive for the peers to become servers. Therefore, all peers have an economic incentive to be only clients, not servers, as serving files costs resources without bringing benefit. Existing peer-to-peer protocols use a credit system enforcing the peer to serve some content to the community before downloading more.