Design Document

1. Introduction

This program is about a Gnutella style file sharing system, which is totally a peer to peer system. This system composed by a certain number of peers, each of which plays a role of both a client and a server.

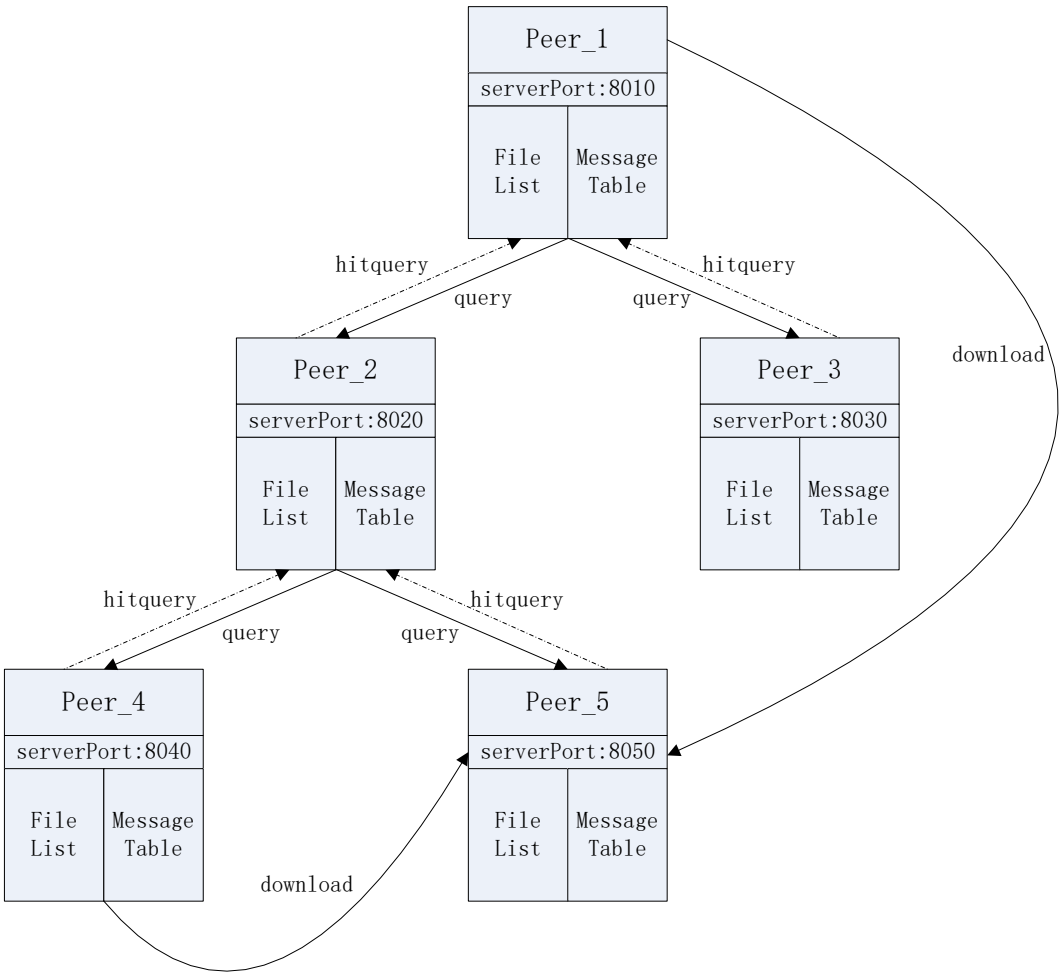
In this program, java programming language and socket programming and multithreads methods were used to implement this simple P2P system.

2. Program Design

This program is composed by peers and the index server in the previous program is removed. The peer can act as both a client and a server. TCP socket and multithread are mainly used in this program. The detail design principles and implemented functions are showed as follows.

2.1 Overall Program

This program implements 4 main functions, such as static initialization, query, hitquery and download.



Each peer works independently and uses configured ports to connect with other peers. All peers play the same role in this program. The above picture shows a simple example to illustrate the working principle.

In the above figure, the Gnutella style file sharing system is illustrated according to the simple binary tree structure, using 5 peers. The detail design document is described below.

2.2 Peer Design

A peer is both a client and a server.

As a client:

The first, the peer reads a configure file and register a local file on its own file list for other peers to search and download.

The second, it can send query messages to other peers and transmit hitquery messages.

The third, if it chooses to download this file, it will set up a connection to the peer, where this file is located in and then download it.

As a server:

It can accept connection requests from other peers for downloading request and provide downloading for other peers.

2.2.1 Static configuration

The configuration file will include peer name, peer IP, server port and its neighbors. When start this program, the peer will set itself up by reading the configure file automatically.

Configure file:

```
1p1 127.0.0.1 8010 p3
2p2 127.0.0.1 8020 p3 p4
3p3 127.0.0.1 8030 p1 p2
4p4 127.0.0.1 8040 p2
```

Peer structure:

```
public class Node implements Serializable{
    private static final long serialVersionUID = 1L;

    public String peerName;
    public String IP;
    public int port;
```

2.2.2 Registry

When one peer sets up, it can provide users option to choose to register files on its local file list, which stores the basic information of the file. In this part, we just use array list to store files' information instead of using other complicated data structures, such as data base for not too much files. And the file information contains file name

and peer ID, so that we know which peer this file came from and we will not confuse about the same file.

File list:

```
public static ArrayList<String> fileList = new ArrayList<String>();
```

2.2.3 Automatic update

This part is similar with that in program one. One thread is used to monitor the status of the folder. A Timer class is used to create a schedule that executes every 100ms. In each execution, it scans the specified folder to watch the change of files in it. According to each scan and compare with the register list in the local peer, if one file registered on the server is removed, it will call the unregister function to unregister this file from the index server automatically.

2.2.4 Query

Peer sends query messages to the neighbor peers according to the configure file, including query command, file name, message id, TTL and the information of the node who send the query message. Apart from this, we use judge statement to see if the upstream peer is in the neighbor list of current peer. The current peer will not send query message back.

Message ID structure:

```
public class MessageID implements Serializable{
    private static final long serialVersionUID = 1L;
    private int sequenceNumber;
    private Node peerID;
```

```
// Query message
public Message(String command, MessageID messageID, int TTL, String fileName)
```

In order to trace the message information and send hitquery message back, a message table is used to store query massage in each peer. The message table is designed by using concurrent Hash Map. The Hash Map key is the message ID and the value is the query message from the upstream node.

```
peerInfo.local.messageTable = new ConcurrentHashMap<Integer,MessageID>();
```

2.2.5 Hitquery

Each node when receive a query message will send back a hitquery message, which will contain the same message ID with query message the peer received. When a peer receives a hitquery message, it will compare this message in the local message table and find out the next hop to continue to transfer the hitquery message.

```
// HitQuery message
public Message(String command, MessageID messageID,
               int TTL, String fileName, String peerIP, int port)
```

2.2.6 Obtain

When a peer want to down load a file from other peers, it will send out a down load message first and then call down load thread to down load that file.

```
// Down load message
public Message(String command, String fileName, String peerIP, int port)
```

In order to implement multiple peers to download file from the same user at the same time, multithread is used to deal with this problem. When peer_1 want to download file from peer_2, it send a message to peer_2, telling him that it will setup a server socket using another port for downloading. Peer_2 receive this message and then set up connection by using this specified port. Similarly, it has the same procedure to make connect with other peers.

2.2.7 Accept multiple client requests at the same time

All the functions are created to deal with multiple client requests at the same time. When one peer send request message to the index server, the server side creates one thread to deal with the request message and guarantee several peers call the same function at the same time.

3. Improvements and Extensions

In the first, we implement a download function to obtain a file from one peer. To be more efficient, we can divide one file into several parts, and obtain each part from one peer. In another word, implement the download function to get file from several peers simultaneously.

In the second, another way to implement a multithread mechanism is using thread pool for efficient scheduling of multithread, which can improve the efficiency to allow more users to connect to the index server at the same time.