Implementation of Peer-to-Peer Network using Distributed Hash Table

COMP3331/9331 Computer Networks and Applications Assignment for Term 1, 2020

Description of System

The application can consist of up to 256 peers. Each peer is both client and server and can send and receive request from others. The main file is p2p.java which contains the main method to create a peer and is responsible for processing the input from user.

Structure

- p2p: The application contains 6 files, and the main method is in*p2p.java* which processes the input arguments, initializes new peer and starts threads.
- Peer: The class of peer nodes which contains information of each node including ID of itself, its successors and predecessors and corresponding ports number. It also contains method of sending TCP request to servers.
- TCPServer: Keep running in an infinite loop, waiting for the request from other peers. Processes request message and send reply to client peer.
- TCPClient: Processes "Quit", "Store" and "Request" command from input when the application is running and sends corresponding request to TCP servers.
- UDPServer: Running in an infinite loop and waiting for the ping message from its two predecessors and store the ID of predecessors to current peer.
- UDPClient: Keep sending ping request to successors' server.

Threads

Here is a piece of code in main method:

```
    // Set up TCP Server and keep listening on the TCP request from other peers
    TCPServer ts = new TCPServer(peer);
    ts.start();
    // Set up TCP Client and keep listening on the input from keyboard
    TCPClient tc = new TCPClient(peer);
    tc.start();
    // Set up UDP connect between peer and its 2 successors
    UDPServer us = new UDPServer(peer);
    us.start();
    UDPClient uc = new UDPClient(peer, pingInterval);
    uc.start();
```

There are 5 threads in the application including the thread of UDP connections, TCP connections and the main thread. - The main thread initializes the peer, starts other threads and processes the command of input. - The second thread is running for TCP server. Each peer set up a TCP server and keep listening on corresponding

port. Once the request from client is received, it will process the sentence and do corresponding actions.
- The third thread will keep listening on the command from keyboard (e.g. "Quit"). Sending different request to specific server based on the command. The method of sending request in Peer will be called in TCPClient. - The next two threads are for UDP server and client respectively. In the fourth thread, peer will initialize a socket to listen on corresponding port and receive pings from its predecessors. - And the last thread, each peer is a UDP client and keeps pinging its successors to check their existence periodically (i.e. 30 seconds as defined in command). Successors and predecessors will be updated during pingings.

Improvement and Extensions

The code structure could be managed more concisely. For example, the TCP request method is implementing in the class of Peer, but it is more reasonable to be moved to the class of TCPClient. And the other point is that the method in `Peer`

```
peer.request(int serverPort, String sentence)
```

is called inside TCPServer in some cases, which means some TCP connections are established inside another connection. This could lead to dead lock between a client and server because both client and server are reading the buffer, and one of them cannot read anything and will keep waiting. To address this issue, it is better to do actions inside server or client uniformly rather than sending request inside servers.

Another thing is the file transmission is not work in this assignment, and I just comment the blocks of code of the attempting code in `TCPServer`. To complete the file transmission, it might need to start anther two threads, one is used to sending file, and the other is used to receiving file. Then set up a new connection between them and sending the file. This could work well but have no time to attempt.

Particular Circumstances

In the beginning seconds after launching the application, the printed out message may not same as the specification. For example, after running the script file and executing seven peers, Peer 19's output could be

```
 Ping requests sent to Peers 2 and 4
Ping response received from Peer 2
Ping response received from Peer 4
Ping request message received from Peer 9
```

Peer 19 does not print out the message of `Ping request message received from Peer 14` because peer 14 may not initialize well at that time. After next period (i.e. 30s), message will print out as expected. So, unexpected message could be generated if test is conducted during the first period.

Reference

Commented code of file transmission in `TCPServer` is borrowed from Augustin Grigorov in StackOverflow.