COMP3331: Assignment Report

Due on Friday, May 29, 2015

Mahbub Hassan

Ruan de Menezes Costa z5050761

Design

The program has the following classes /interfaces:

- Peer: Represents a peer in the CDHT. A peer has it's own id, the sucessor's and the predecessor's. A peer also has 4 functionalities: To send and receive messages via UDP and TCP. It has an object to deal with each of these functions.
- TcpClient: Allows the peer to send FileRequest, FileResponse and DepartureMessage messages through a TCP connection. It works in a separate thread. It will monitor the command line for requests from the user.
- TcpServer: Allows the peer to receive FileRequest, FileResponse and DepartureMessage messages. Works in a different thread.
- UdpClient: Allows the peer to send PingRequest and PingResponse messages. Works in a separate thread.
- UdpServer: Allows the peer to receive PingRequest and PingResponse messages. Works in a separate thread.
- Message: Defines a common interface for all kinds of messages. A message contains the sender and receiver id. Some of the methods in this interface aren't meaningful for all kinds of messages, but this design led to an elegant solution, avoiding switches and if's. The most important thing here is the method executeAction(Peer receivingPeer). Each type of Message will have the code that should be executed by the receiving peer when this peer receives the message. When the server receives the message (TcpServer or UdpServer), it will have the message action to be executed by doing message.executeAction(peer), passing it's own peer as parameter.
- There are 5 types of message: PingRequest, PingResponse, FileRequest, FileResponse and DepartureMessage. It's not really instructive to describe them here. They do exactly what their names suggest.

After the peer is initialized, the method run() can be executed. This method will start the four threads. The ping request interval is 5 seconds. From my tests, this value is good enough and doesn't cause overhead, even if ten peers are being run.

Sucessor and Predecessor updates

To keep the predecessors updated, in the body of the executeAction(Peer receivingPeer) method of the PingRequest message, the receiving peer will update itself, using the sender's first sucessor in order to know if the sender is the first predecessor or the second. Similarly, when a peer quits, sending a DepartureMessage to it's predecessors, the receiving server will execute the executeAction(Peer receivingPeer) method of the message, passing it's own peer as parameter. The code inside executeAction(Peer receivingPeer) will update the sucessors.

Possible improvements

I can think of two improvements. The first would be to implement the Part 5 of the assignment using timeout and sequence numbers. I didn't do this because of time constraints. The second one would be to transfer real files. I believe this wouldn't be much harder.