DHT First iteration - Group 2

Compilation and using program

Compiling can be done simply running make and there are no dependencies except for OpenSSL. When the program is started, host and server addresses must be given as arguments in the form `./dhtnode <host-address> <host-port> <server-address> <server-port>`. For example: `./dhtnode 130.200.41.68 2000 130.200.41.69 1234` (if the program is tested on a single computer `localhost` can be used instead of the IP addresses). When the program is running commands can be given through stdin. However, currently the only command is ‘`q`’ which disconnects the node from the network (if the server allows, that is).

By default debug messages (similar to those of the server) are turned on. These can be disabled by compiling with -DNDEBUG flag.

Implementation

Files

`dhtnode`	Main functionality
`dhtpacket`	Packing and unpacking of DHT packets
`socketio`	Sending and receiving packets
`hash`	Hashing things
`typedefs`	Defined types and miscellaneous stuff (i.e. a general include file)

Architecture in a nutshell

- Main function has three parts. First the node connects with the server.
- After connecting the main (running) loop is started
- In main loop packets are received and based on sender and packet type various things are done. This is done with switch and case statements examining packet type, say for example DHT_REGISTER_ACK, and if statements studying whether packet comes from the server or other node.
- Under every case statement, appropriate functions are called and tasks are done. For example after receiving DHT_REGISTER_BEGIN from the server, we will open connection to joining node, do the handshake and (after second iteration) send data.
- When attempting to disconnect the program stays in the main loop and waits for a permission from the server to begin disconnection sequence. After this program comes out of the main loop and to a new loop that only sends the data to the neighbours and terminates when got confirmations from both neighbours.

A lot of things regarding the assignment were unclear (this can be seen in program design) and we initially designed our program to be more clever than what was actually required. This was before we understood that there wasn't any need for keeping several connections or keeping any track of the state of the network because the server did all that.

It should be also mentioned that the program is also (almost completely) stateless. This simplifies program logic and consequently makes development easier.

Testing

The plan was to test continuously (i.e. test after something new is done). However, in reality this turned out to be not possible, since in the beginning we concentrated mostly on architecturally significant decisions and there wasn't really anything working to test before the skeleton of the main function was ready. We carefully finished the main function first and then wrote a few important functions to make it all work. Luckily thanks to careful planning there were very few problems and we were actually able to fix them straight away. Testing was done mainly on Niksula computers, connecting and disconnecting several nodes on a couple of computers. The program was also tested on a virtual machine running Linux Mint. In this test, however, connections couldn't be made using IP addresses (only localhost) but we suspect this might be due to network configuration. Eclipse's debugger was used to find some minor errors in the sending and receiving functions. Valgrind was also used to find memory leaks (there shouldn't be any).

TL;DR: All required features are implemented and testing hasn't shown any problems.

Known bugs and problems

Sometimes server sends a packet which starts with a single '?' character. This causes a slight problems because packet parsing has to be offset by one byte. This is relatively easy to do but breaks if the senders key actually starts with a 0x3F.

Also, in many places the program assumes that inputs are correctly formed and that things just work in general. That is to say, there probably should be some validation in, for example, packet receiving.

Project management

Time management

We didn't have any time management plan because really had no idea how much time this project would actually take. However, we think that our time management has been pretty successful: instead of doing work in a few long sittings we've spread out our work over multiple sessions. We have done things mostly together and to date we have spent around 30 hours with the project.

Project plan

We couldn’t find a common communication method so we have relied heavily on face-to-face meetings. This hasn’t necessarily been bad thing because many things (regarding the assignment) were unclear and discussing them face-to-face was eventually very efficient. We also employed peer programming techniques (although one of us did most commits we actually worked together). This has been for sure effective for the first iteration, as there were many architecturally significant decisions made together. As what it comes to second iteration, we’ll work more individually, because there will be more separate functionalities (GUI, data handling etc.) that can be done independently. All in all, our teamwork has been very successful. Especially peer programming was surprisingly functional and enjoyable.