Reflection report

Dorien Meijer Cluwen

April 19, 2017

1 Development process

As apposed to last time, I have taken the first few days (read: the weekend) to only think about the problem and not do any coding. This helped me to setup classes like `myUDPHeader` and `UDPPacket` and corresponding tests. I chose to have seperate client and server classes. This leads to more commands (eg. both sides need an downloadCommand), but makes it easier to tailer commands. With the use of the command structure of last project I could quickly setup a basic structure in which both client and server could react to certain keywords (like upload, download, pause and resume).

Implementing (reliable) sending and receiving was a lot harder however. I struggled a lot with both client and server throwing `java.net.BindException: Address already in use` (eg. the address/port) exceptions. In the end this appeared to happend due to one side not shutting down correctly, and hence leaving them open. Using the `service` command in combination with `tail -f /var/log/syslog` and using fixed ports seemed to help.

Another struggle happened in the multicast dns algorithm and it not being able to connect to a multicast address. None of the addressses suggested online worked and in the end it turned out that the NedapUniversity network had no multicast address (as administrators normally set those up).

This accumulation of seemingly minor bugs and having a lot of stubbed code meant that it took about a week to just do a handshake between the client and the pi. Which lead to a bit of stress and wondering if I would be able to send anything at all before that deadline.

I decided to do away with the acking of messages for a while and focus on getting a 'list files' request over to the pi and back. When that worked I started implementing a Stop-and-Wait protocol, which only has a window size of 1, but is very usefull for debugging the actual commando's/requests. I chose to let each request start and end at the client side, in this way the client always knows if a request has ended succesfully. For the ending of a request I use a `LAST` flag, a flag I intended to use to tell when a file fragement was the last of the sequence of fragemented packets.

With the Stop-and-Wait protocol in place the upload and download requests could be implemented. And after that a checksum both per packet and per file downloaded/uploaded. The checksum gave some strange errors, even though the tests worked perfectly, the packets containing file data never had a correct checksum. After a lot of debugging it appeared that fragments of size $MAX\_BUFFER + headerSize$ were send, but fragments of size MAX_BUFFER were extraced from the socket. Eg. all data packets, which used up all the available space in the packet, where missing $headerSize$ bytes. After that I have never encountered an invalid packet (or incorrect md5 for a file) anymore.

At this point (read: Tuesday) I could finally start on a Sliding Window protocol. Luckely we already did something like that in the challenges. Implementing it was quick, but not without fault. The acknowledgement of the md5 file checksum could not reach the client. Its packet(s sequence number) was outside of the clients receiver window and only those packets acknowledgment numbers were stored. Aka. we already received the packets data but not the acknowledge number it also carried. This was easily fixed.

The last day I spend on adding some statistics and testing larger files. The program also has the possibility to pause and resume active commands, resuming canceled/aborted commands is due to time constraints not possible.

2 Code review

I am most satisfied about the structure of the commands, it makes it easy to add new commands. Its also easy to add another protocol to some commands, eg. only the upload and download commands use sliding window. The other commands use so few packets that stop-and-wait is better suitable. The use of helper classes make it less error prone to send data for mDNS or meta data for file transfer.

Improvements could be made in the way the classes interoperate, there are a lot of links between classes. For example, the `Sender` asks all registered commands in a Round Robin fashion if the have another packet to send. But to register a command to the `Sender` class it has to go via the `Handler` and through the `ReliableUDPChannel` class.

Other improvements that could be made is the ability to resume aborted requests, either canceled by the user or because either host has gone down. And being more hufter proof, eg. if you do nicely what is asked it will work, otherwise it might not.

3 Design

This section shows the workings of the applications through some sequence diagrams.

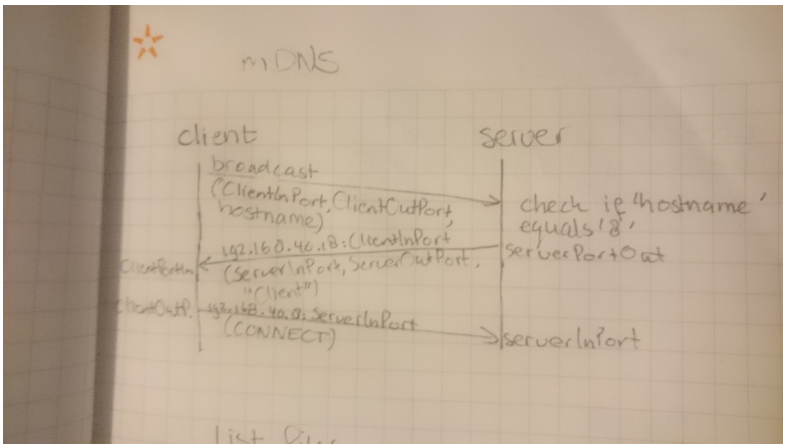

Figure 1: mDNS

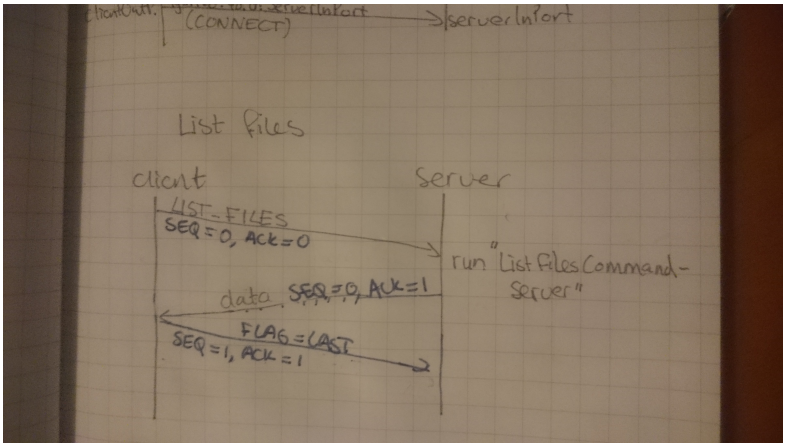

Figure 2: Listing files from the pi.

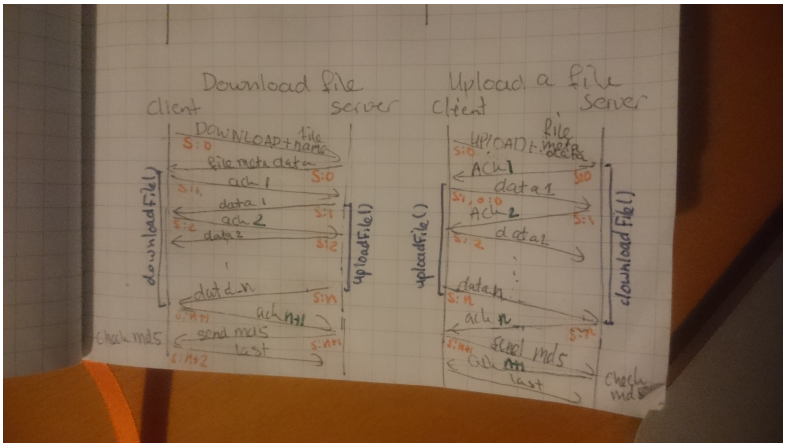

Figure 3: Uploading and downloading