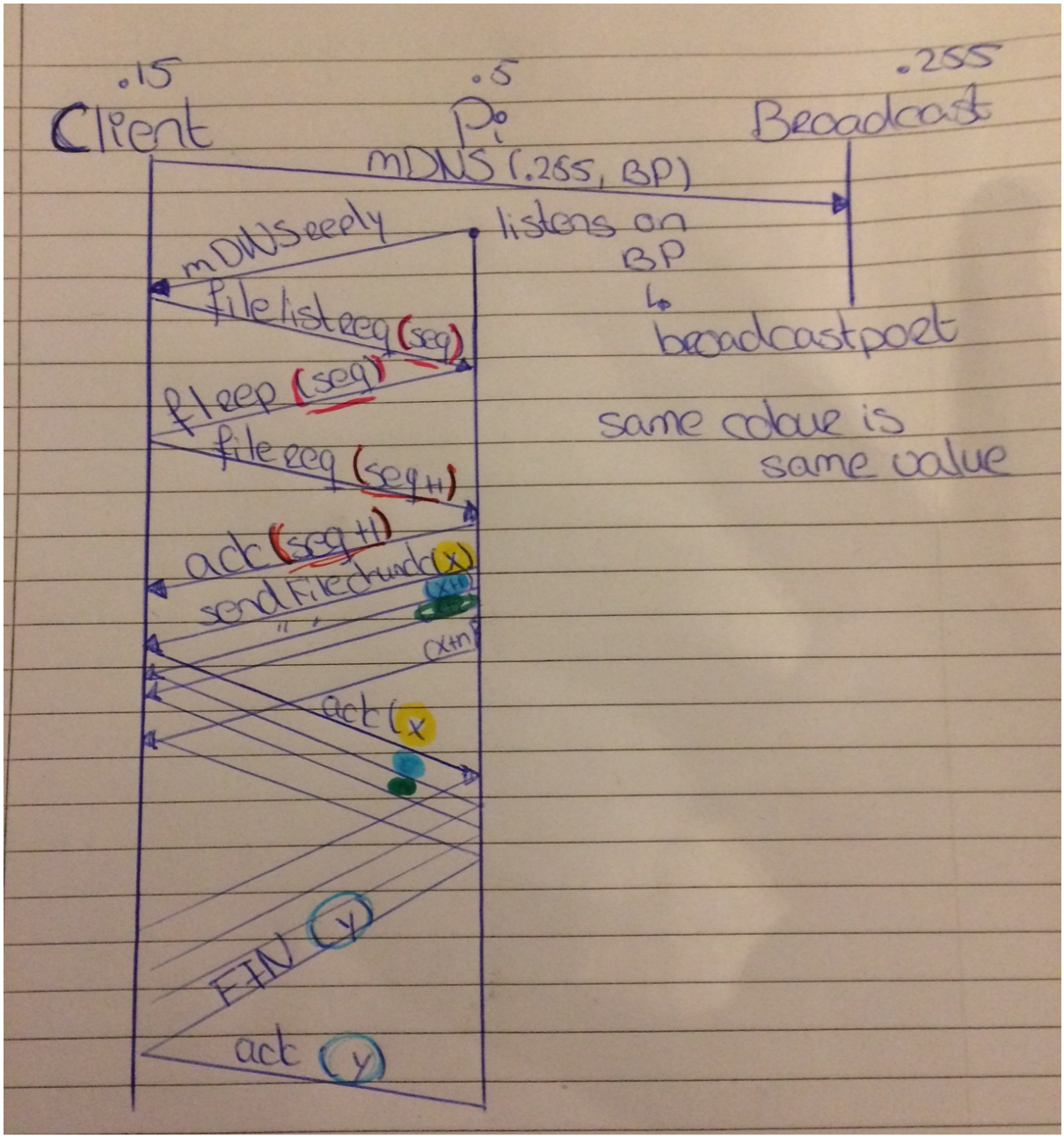
Reflection report Network Systems

by Anne-Greeth van Herwijnen

Introduction

This report describes my implementation and experience during the final assignment for module 2 “Network Systems”. The goal was to achieve reliable data transfer over an unreliable medium. We used a raspberry Pi 3b to upload files to and download files from. The structure of the report is as follows: Sequence diagram, What I learned from Java, Code review, Improvements & ToDo’s, personal feedback and Conclusion.

Sequence Diagram



What I learned from Java

This part describes the things that are most out there that I learned to use in Java this module.

Datagram Socket

The first thing is the Datagram Socket, since we only used TCP during the university challenges and the previous module the datagram socket, used to send UDP packets in Java, was new for me. The fact that is was new also made it complex, since things work differently. For example you don't wait for a socket to connect to yours, you just open a port. Also the use of the multicast Socket was new, but interesting. Especially the fact that I needed to think about how IP-addresses really work, resulting in the use of the 192.168.40.255 as broadcast-IP.

Volatile

One of the strange bugs I encountered this assignment was fixed using the keyword volatile.

> *"…volatile has semantics for memory visibility. Basically,the value of a volatile field becomes visible to all readers (other threads in particular) after a write operation completes on it. Without volatile, readers could see some non-updated value." – StackOverflow*

The problem was that I used a boolean to notify the client/Pi of the fact that a packet had arrived and should be picked from the queue. However, even when the boolean changed the if-statement was not triggered. Luckily, with the help from Joël the bug was found and he introduced me to the concept of volatile booleans.

Concurrent lists etc.

Of course, using threads leads to concurrent exceptions. I did not had those with my Go-assignment, so this was my first time. But with some googleing I found the different implementations of concurrent lists and queues. I used the queue implementation when it was really a queue. Something I would like to learn more about, which I did not seem to find one, two, three on Google/StackOverflow was what you should use: synchronizedList or concurrentLinkedQueue etc.

Code review

I think the general structure of the code is solid. I'm still confident that it was a good choice to keep a separate class for the Pi and the Client since the behaviour (especially concerning terminal input) differs a lot. However there is still a lot of duplicate code that does not really should be in the Utils package, so that could be improved. There is also not a lot of documentation with the functions, partly because I did not have the time to do that, but also because I feel that the function don't really do complex things, and the names are pretty self-explainatory.

Improvements & ToDo's

This section is called improvements and ToDo's because unfortunately I did not manage to meet all the requirements. My application is able to upload and download a file of 100MB (however, it's not that quick). You can also list and choose a specific file and the multi-cast DNS is implemented. The integrity of the downloaded file can also be checked. But the

ToDo's are: Pause and resume a download, multiple files at the same time and some more statistics (there are now only basic statistics). One of the things I would consider somewhere between a ToDo and improvement is the reply with an ACK on the FIN message. In the current situation, the sender just replies with an ACK when the FIN is received. However, there should be a check and possible re-request when the checksum results in false. Since I just realized this today (19-4) I was not able to come up with a beautiful solution yet.

For some reason the download from the pi only works correctly after doing an upload or request from the client. Probably because this is used to sync the sequence numbers for the pi. I wanted to fix this, but it's 23:27 so that is not going to happen before the deadline.

Improvements

The application could benefit from a code review to find some performance improvements. Also, as discussed above, there should be even less duplicate code. But still, the focus should be on improving the performance and of course finishing the ToDo's.

Personal feedback

Overall I think the biggest improvement from the first project (GO) to this one is that I tried to work more structured. I also tried to keep the code organized: keeping the InteliJ warnings to a minimum and creating functions even if it was for something small. I started the project with one packet and I used this until Easter Monday, when I decided that it was probably better to move things to packet to improve the scope of the functions. Another thing that helped a lot with keeping me focussed was the use of the SCRUM method, which we do with our team. This "forced" me to snip the assignment into smaller pieces and to tell others about what I was doing and was going to do.

Conclusion

I really liked this challenge, and I liked the fact that it showed how much we learned during this module, also in writing Java even though that was not the focus of this module. I think the tricky part with this assignment was that it seemed as if we could reuse a lot of the things we programmed during the challenges. However, since the challenges used TCP a lot of concepts were still new (see: What I learned from Java) which made it more challenging. I would love to spend some more hours on this to complete all the requirements and polish it a bit more, but for now I can at least send a 100MB file.