Report 1: Rudy - A Web Server

Pethrus Gärdborn

September 14, 2017

1 Introduction

In this seminar, we have worked closely with the network aspect of distributed systems. As a distributed systems programmer, it is important to have a good understanding of how the network below the application functions and how to optimize the different network options for an application's specific needs to be able to develop applications that are as good as possible. Using Erlang, we programmed a very basic server by making use of sockets. We learned about the difference between a listening socket and the socket later used for communicating with the client, and their different usages, which of course is also important to know when you are developing distributed applications. By creating a server and using it to communicate with a client, deeper insight into the important client-server model has also been gained. This is one of among several different architectural models that one may use in distributed systems programming. Through hands-on experience with this model may help to later on choose which model should be used when designing another kind of distributed application. Once the server was programmed and ready, we went on to do some benchmarking of how fast the communication between client and server was. The benchmarking process highlights the fact specific to distributed systems programming, that the latency in the network has no upper limit. One may have an average value from most trials but then a certain connection takes much much longer.

2 Main problems and solutions

The fact that much of the code was already given, did a lot to facilitate the programming work. However, before starting to programming, I had to refresh my knowledge about the HTTP protocol and also read some about sockets to get a more secure understanding of the concept. For the most part, it went well to fill in the missing pieces in the "rudy" module. However when trying to contact "rudy" from my browser with the given example, my browser did not display the desired "/foo" on the screen. This problem took

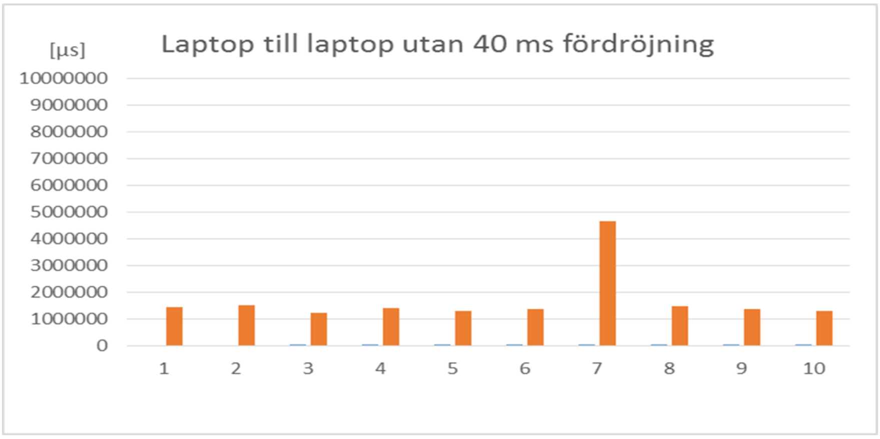

Figure 1: Laptop to laptop without any artifical delay

me a long time to debug. It turned out that I had changed one of the given ”don’t cares” to the variable ”Body” which I was using as an argument to the function ”http:ok/1”. The function was given like this leaving it up to the student to fill in the argument to ”http:ok/1”.

```
reply({{get, URI, _}, _, _}) ->
     http:ok(...).
```

I had changed the code to this:

```
reply({{get, _URI, _Ver}, _Headers, Body}) ->
    http:ok(Body).
```

It was wrong to have ”Body” as an argument and by changing back to how the code was given from start, realized that ”URI” had to be the right variable to use and everything worked fine after that.

To be able to use the ”gen_tcp” functions correctly, I read some of the Erlang documentation. I also read a well-written document of how sockets work from Oracle’s website on Java. I also encountered problems in setting up the benchmarking program. I tried to use one of the computers in the computer lab along with my laptop but I could never get Erlang to work properly on the computers in the computer lab. It did not recognize certain built in functions such as ”erlang:system_time/1”. I drew the conclusion that it was because of an old Erlang installation and since I did not have the permission to download a newer version, I found an old laptop at home where I was able to install the latest version of Erlang and the benchmarking went fine after that.

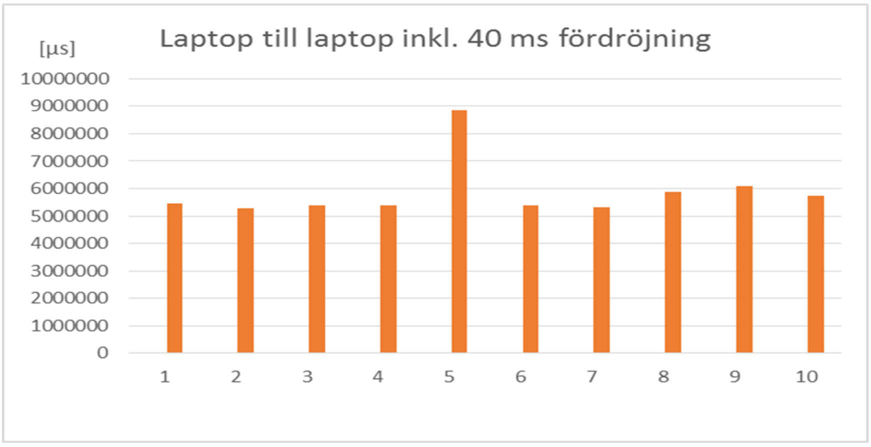

Figure 2: Laptop to laptop with 40 ms delay

3 Evaluation

Once the coding was completed and the benchmark program set up properly, I ran three different series of tests with ten trials each, contacting the server over a wireless LAN. The server was set up on one laptop and another laptop as well as an iPhone was used to contact the server. The three different tests were as follows:

1. Laptop to laptop without any artifical delay
2. Laptop to laptop with 40 ms delay
3. Laptop and iPhone connecting at the same time with 40 ms delay

4 Conclusions

When doing the benchmarking, it was interesting to see in practice how in two of the different series - fig.1 and fig.3 - there was an average contact time that stayed fairly in the same range, but there were one trial in each series that really stood out and was about twice as long as the other calls. That highlighted to me the problem of a network that cannot give guarantees about latency in distributed programming. Furhtermore, as I mentioned in the beginning, the concept of sockets have now become much clearer to me when programming and that once a connection has been established in a listening socket, it should be handed over to another socket continuing the communication with that client while the listening socket can be continued to be used to listening to other clients.

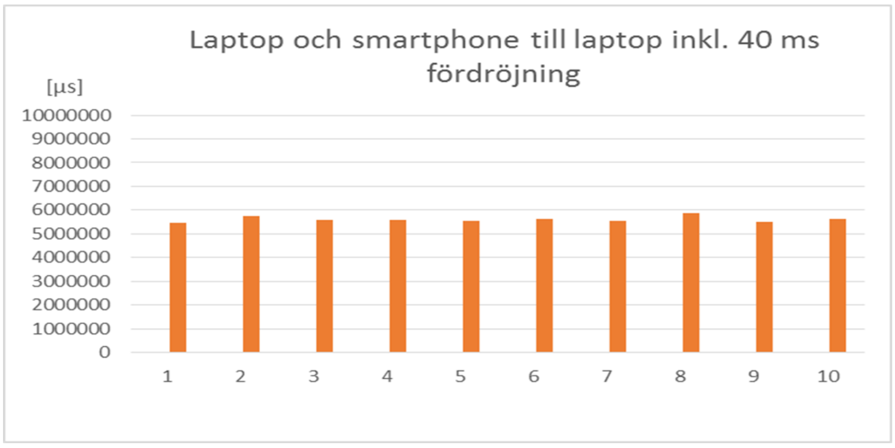

Figure 3: Laptop and iPhone connecting at the same time with 40 ms delay