Report 2: Homework Report routy

Marcus Samuelsson

September 21, 2023

1 Introduction

This task revolves around getting a deeper understanding on how communication is managed in a network with several nodes/servers. These nodes/servers could be located far from each other in different region, where each node does not have direct access to the others.

More specifically it is to create a routing network using the link state protocol and the Dijkstra algorithm. This network should be able to send messages between the nodes in the network in the end. The network need to be flexible to handle frequent changes to the system with nodes being able to go offline and create new connections during runtime.

2 Main problems and solutions

2.1 General issues

The biggest issue throughout this task and the previous task was again the somewhat lacking information around *Erlang* and unclear instructions on how to run some parts of the code. It took a while to run the final program as the errors given was not easy to find solutions to on the web or in the course. Eventually the cause was understood and the solution was reached through trial and error. Furthermore the *lists:foldl/3* library that was given was given a warning that it should not be used if it could not be understood. This is used in the task, but using it was somewhat easy to understand but the description of it was still difficult to grasp completely.

2.2 Faulty instructions

The second issue was the file *routy.erl* which was given completely in the instructions. This file contained numerous faulty code. Bellow are some examples of faulty code.

The given line was:

```
    router(Name, 0, Msgs, Intf, Table, Map).
```

Should be:

```
router(Name, 0, Hist, Intf, Table, Map).
```

The parts using the file *intf.erl* was different for different parts of the code and took a while to figure out. These both used the same method from the same file but use different names for the same method.

Which eventually was discovered(as seen bellow):

```
Intf1 = intf:remove(Node, Intf),
.
.
.
interfaces:broadcast({links, Node, R, Links}, Intf),
```

Something else was the fact that the following command line was given:

```
lund ! {add, stockholm, {r1, 'sweden@130.123.112.23'}}.
```

That in actuality should have been in this format:

```
r2 ! {add, stockholm, {r1, 'sweden@130.123.112.23'}}.
```

Where the structure is:

```
reg1 ! {add, city, {reg2, 'counry@ip-address'}}.
```

This was another factor that just made the process very confusing as instructions that was thought to be accurate led to looking for solutions in the wrong part of the task.

2.3 The code

The last issue which also caused a lot of extra time was the problems with formatting when mixing files. These problems became a problem because of following the tests given in the tasks which resulted in the correct results, but when connecting everything these parts started seeming like it worked incorrectly. Specifically in the module *dijkstra.erl* where the sent in table for the *table/2* function was found out to be faulty, where the first object of the list was another object. Eventually it was found that the issue was not in the module *dijkstra.erl* and instead in the issue was in *intf.erl* with the function *list/1*. Because *table/2* in *dijkstra* was using so many other functions a lot of things where tested unnecessary around those before finding the exact problem. This is one example of similar problems that also occured.

3 Evaluation

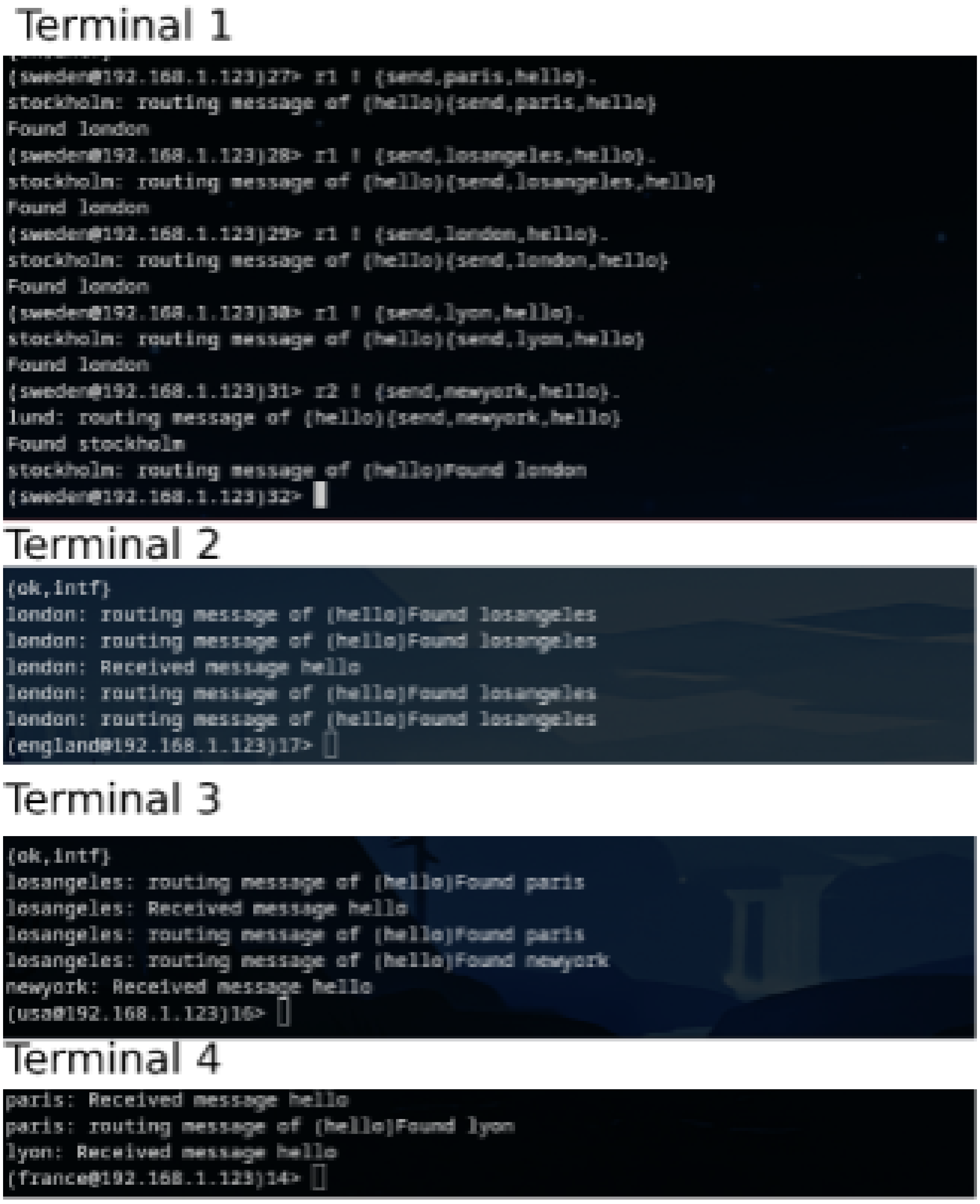
To evaluate this task four different shells where run at once. Each shells represents a country where all the cities are connected as routers. Bellow is the list of countries and cities.

Sweden	England	USA	France
Stockholm	London	Los Angeles	Paris
Lund	Birmingham	New York	Marseilles
Göteborg	Nottingham	Texas	Lyon
Norrtälje	Cambridge	Ohio	Toulouse
Malmö	Lancaster	Hawaii	Nice
Jämtland	Westminster	Washington DC	Lille

The different countries where then connected through the capital in this manner bellow:

```
Stockhom - London - Los Angeles - Paris
```

By connecting the capitals a network of nodes is created. This allowed for message passing between the networks/countries(see Figure 1 bellow).


Terminal 1

```
(sweden@192.168.1.123)27> r1 ! {send,paris,hello}.
stockholm: routing message of (hello){send,paris,hello}
Found london
(sweden@192.168.1.123)28> r1 ! {send,losangeles,hello}.
stockholm: routing message of (hello){send,losangeles,hello}
Found london
(sweden@192.168.1.123)29> r1 ! {send,london,hello}.
stockholm: routing message of (hello){send,london,hello}
Found london
(sweden@192.168.1.123)30> r1 ! {send,lyon,hello}.
stockholm: routing message of (hello){send,lyon,hello}
Found london
(sweden@192.168.1.123)31> r2 ! {send,newyork,hello}.
lund: routing message of (hello){send,newyork,hello}
Found stockholm
stockholm: routing message of (hello)Found london
(sweden@192.168.1.123)32>
```

Terminal 2

```
{ok,intf}
london: routing message of (hello)Found losangeles
london: routing message of (hello)Found losangeles
london: Received message hello
london: routing message of (hello)Found losangeles
london: routing message of (hello)Found losangeles
(england@192.168.1.123)17>
```

Terminal 3

```
{ok,intf}
losangeles: routing message of (hello)Found paris
losangeles: Received message hello
losangeles: routing message of (hello)Found paris
losangeles: routing message of (hello)Found newyork
newyork: Received message hello
(usa@192.168.1.123)16>
```

Terminal 4

```
paris: Received message hello
paris: routing message of (hello)Found lyon
lyon: Received message hello
(france@192.168.1.123)14>
```

Figure 1: Test run of Stockholm sending messages to other cities

4 Conclusions

This task was a lot more difficult then the previous task. It required a lot more manual implementation of code rather then pre-made code skeletons to use. This task overall gave a good understanding on how to create a link state routing protocol and how it works. The biggest challenges was the confusion around the faulty instructions causing confusion, having to figure out all the new libraries and how to test the program properly. Overall it went well in the end with everything working.