Homework 2 Report
Riccardo Fragale

September 17, 2025

1 Introduction

The task was to implement a link-state routing protocol in Erlang. The link-state protocol is used in, for example OSPF, the most used routing protocol for Internet routers. We were to implement routing processes with logical names such as London, Berlin, Paris etc. Routers are connected to each other with directional(one-way) links and can only communicate with the routers they are directly connected to. The routing processes should be able to receive a message of the form **(route, london, berlin, "Hello")** and determine that it is a message from Berlin that should be routed to London. A routing process should consult its routing table (where we have computed the shortest path to reach every node in the network) and determine which gateway is best suited to deliver the message. If a message arrives at its destination it is printed on the screen. Messages for which paths are not found are thrown away, and no control messages are sent back to the sender.

2 Main problems and solutions

The main challenge was to correctly implement the Dijkstra algorithm in Erlang, making sure that the routing table are updated and handled correctly reflecting the Map of the network. It is also important to notice that the system I was implementing was supposed to be resilient to changes due to the removal of nodes (or better, routers) in the network topology.

I found a lot of issues related to the actual implementation of the Dijkstra algorithm (that can be found in the module **dijkstra.erl**. In particular, since I hadn't developed the function **map:allNodes** in a proper way, the iteration through the network needed to create and update the routing table was leading into major errors. Somehow, in the routing table of some nodes, the system was suggesting that a single node should have forwarded messages to gateways that are not connected to it. As a result, the whole system was not able to correctly send messages. After reshaping the whole module map,

I was able to correctly deliver all the nodes to the *iterate* function and thus all of sudden everything worked correctly. To test the whole system I created a file called **routy_demo** where I set up a network on a node called Sweden. The picture below shows the network topology.

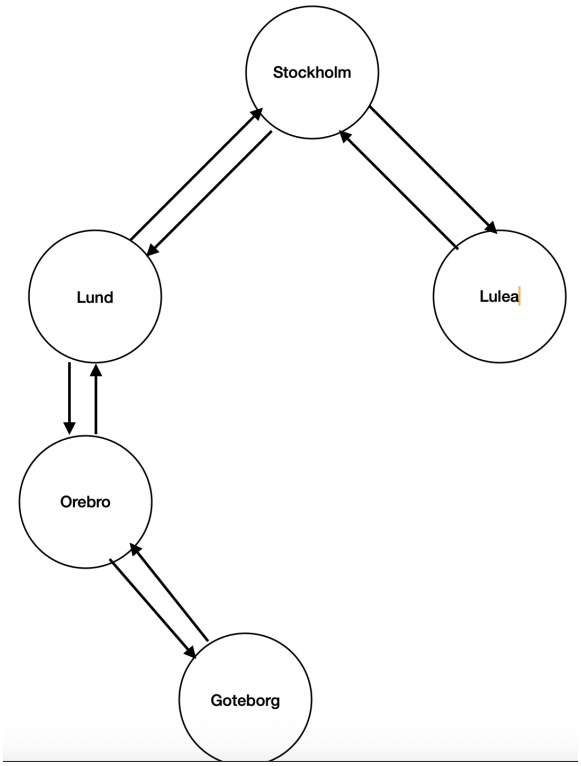

Figure 1: Picture of my network

In my demo, after creating the network and verifying the routing table, I sent a message from Stockholm to Goteborg.

```
r1 ! {route, goteborg, r1, "Hello from Stockholm to Goteborg!"},
```

My application is printing on screen:

```
stockholm: routing message ("Hello from Stockholm to Goteborg!")
lund: routing message ("Hello from Stockholm to Goteborg!")
orebro: routing message ("Hello from Stockholm to Goteborg!")
goteborg: received message "Hello from Stockholm to Goteborg!"
```

This is correct as the messages should pass through all those nodes to reach the final router.

Then I tried to kill one router (Orebro) on the network and see what is happening.

```
r1 ! {remove,lund},
```

Then, sending a message again from Stockholm to Goteborg, shouldn't lead neither to an error (if the system is correctly implemented) nor to a message received by Goteborg. Actually, the output is

```
stockholm: gateway lund not found in interfaces
```

This shows that Stockholm is not able to find anymore the gateway Lund, thus the message is not correctly arriving to destination.

I decided to implement in the module called **intf.erl** the "support" for communications among routers that are not in the same node. Inside the function called *broadcast* I inserted the possibility to send the broadcast message not only to processes whose Pid is known(and so they are in the node) but also to processes whose Pid is unknown but the node is correctly registered. This was useful especially for the optional task

3 Optional Task

I decided to implement the optional part with a colleague of the course called Lorenzo. We used my network (as defined above) connected to his network (see the picture below). We took away the connection between Turin and Madrid in his network. We also decide to connect my router r3 (Lund) to both Torino and Madrid.

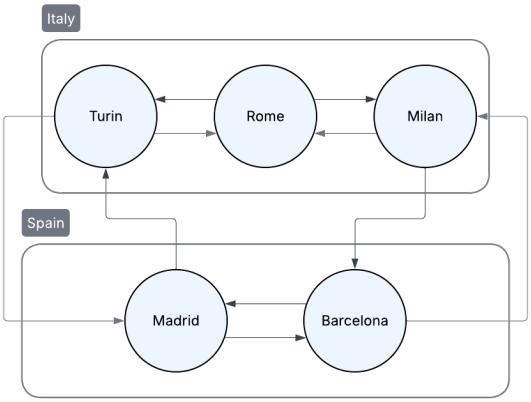

Figure 2: Picture of Lorenzo's network

First, we set up the whole network and checked whether the algorithm to produce the routing table was working properly. In addition, we needed also to verify that the map was correct and that the interfaces were correctly added. To test it we inserted a status message to node Turin. As it can be seen from the photo below everything seems as it should be.

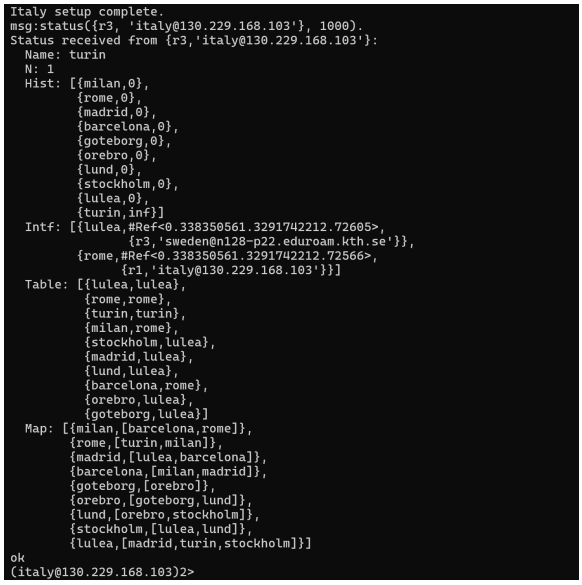

Figure 3: Picture of the routing table of Turin

Then we decided to send a message from Turin (called r3 in Lorenzo's node) to Madrid.

```
r3  ! {send,Madrid,"Hello there"}.
```

The faster way is obviously to pass through Lulea and everything worked as the three screenshots below show.

Figure 4: Starting of message routing

Figure 5: Message pass through Lulea

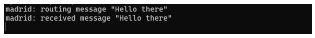

Figure 6: Message arrives in Madrid

Figure 7: Routing from Turin to Madrid through Lulea

Then, we decided to kill completely my node (called Sweden) and verify how the network is reacting. We used a script to kill the processes and take into account the timestamp both for my shell and for the shell of Lorenzo(which is receiving a message thanks to the Erlang monitor).

```
#!/bin/bash
pkill -u "$USER" beam.smp
echo "Erlang processes killed at: $(date +%s)"
```

The following are the timestamps we observed in our terminals:

```
Exit timestamp 1758108121 (on node Sweden)
turin: exit received from lulea at timestamp 1758108122792
madrid: exit received from lulea at timestamp 1758108122791
```

It is really a small amount of time. As a final test, we decided to send again the message from Turin to Madrid. Initially, the network is not able to find the gateway Lulea so the message is lost.

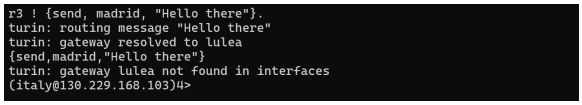

Figure 8: Failed try of routing from Turin to Madrid

Then, simply by sending an update message to his nodes (called Italy and Spain), the routing tables are corrected and the routing is successful (as shown in the photo below). This shows the resilience of the network to nodes killed.

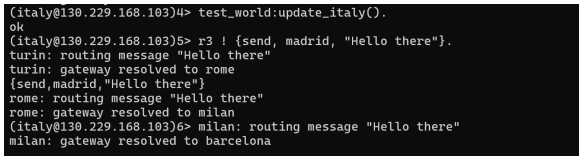

Figure 9: Correct routing through Barcellona

4 Conclusions

This problem was very useful both to improve my Erlang skills and to understand how the routing process works in a distributed architecture. I can see at least some major improvements that could be made. As an example, we may attach a cost to each single connection, rather than considering all of them as equal. In this way, dijkstra algorithm would also have to take this costs into account when looking for the best path between two routers. Adding this feature would make our router network more similar to real systems in my opinion. Another minor improvement might be to create an internal system to detect dynamic changes in the network topology rather than forcing this two aspects via a manual signal sent to the processes. In our system we are forced to communicate to a single node the removal of neighbouring gateways. This not only slows down the entire system but also exposes the network to possible errors due to the user's mistakes. We might opt for a simple set of messages that routers are sending between them to verify their liveliness every certain timestamp.