Report 2: Routy, a small Routing Protocol

Lorenzo Deflorian

September 15, 2025

1 Introduction

The main goal of this assignment was to implement a link-state routing protocol using the Dijkstra algorithm. The protocol should be able to handle dynamic changes in the network topology, such a link failures and recoveries, and update the routing tables accordingly.

2 Main problems and solutions

The main challenge was to implement the Dijkstra algorithm correctly, making sure that the routing tables are updated correctly, reflecting the current state of the network.

2.1 Network Topology

We start defining the network topology as shown in figure 1. We divide the whole network in two areas, running on two different Erlang nodes. We call the first area "Italy", which contains routers r1 (Rome), r2 (Milan), r3 (Turin), while the second area "Spain" contains routers r4 (Barcelona), r5 (Madrid).

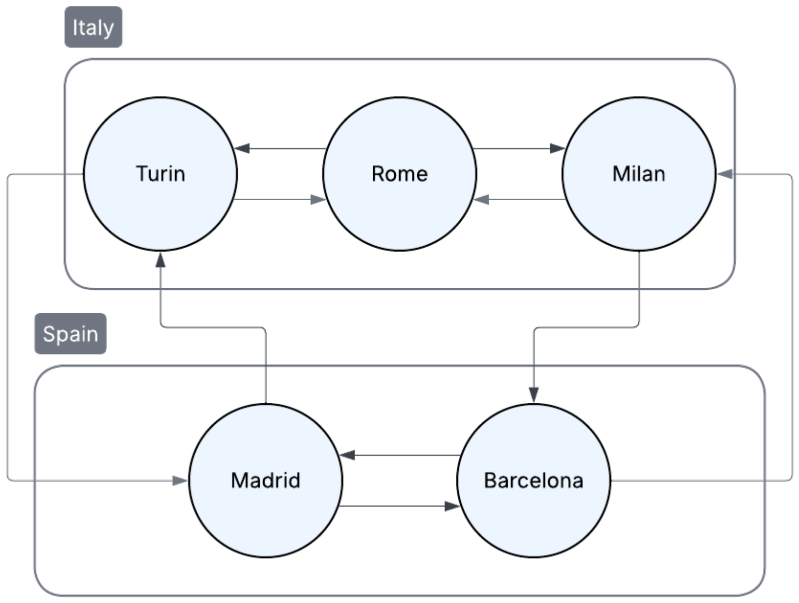

Figure 1: Network Topology

The connections between the routers are also shown in figure 1. We set links in both directions, where all links have the same cost.

3 Evaluation

3.1 Testing the implementation

After setting up the network topology as described above, we call the the `broadcast` and `update` functions on each router to propagate the link-state information and computer the initial routing tables.

3.1.1 Initial message routing

Let's send a message from Rome (r1) to Madrid (r5) and see how it routed through the network.

```
(italy@DORORO)1> r1 ! {send, madrid, "Hello there"}.
rome: routing message "Hello there"
rome: gateway resolved to turin
{send,madrid,"Hello there"}
turin: routing message "Hello there"
turin: gateway resolved to madrid
```

```
(spain@DORORO)1> madrid: routing message "Hello there"
madrid: received message "Hello there"
```

As we can see the message is correctly routed through the shortest path, which is **Rome** → **Turin** → **Madrid**.

3.1.2 Simulating a link failure

Let's simulate a link failure between Turin (r3) and Madrid (r5) by removing the connection between them. We also need to update the routing tables on both areas and propagate the changes through the network.

```
(italy@DORORO)2> r1 ! {remove, turin}.
{remove,turin}
(italy@DORORO)3> test:update_italy().
ok
```

```
(spain@DORORO)2> test:update_spain().
ok
```

Now, let's try to send the same message from Rome (r1) to Madrid (r5) again:

```
(italy@DORORO)4> r1 ! {send, madrid, "Hello there"}.
rome: routing message "Hello there"
rome: gateway resolved to milan
{send,madrid,"Hello there"}
milan: routing message "Hello there"
milan: gateway resolved to barcelona
```

```
(spain@DORORO)3> barcelona: routing message "Hello there"
barcelona: gateway resolved to madrid
madrid: routing message "Hello there"
madrid: received message "Hello there"
```

As we can see the message is still correctly routed through the network, but this time the path is **Rome** → **Milan** → **Barcelona** → **Madrid**, which is the new shortest path after the link failure.

4 Conclusions

In this report, we have described the implementation and testing of a link-state routing protocol using the Dijkstra algorithm. We have shown how the protocol can handle dynamic changes in the network topology, such as link failures and recoveries, and update the routing tables accordingly. The testing results demonstrate that the protocol is able to find the shortest path between any two routers in the network, even in the presence of link failures.

For better improvement, we could consider implementing link cost variations, where different links have different costs, and the Dijkstra algorithm would need to take these costs into account when computing the shortest path.

Another possible improvement could be to implement an automatic way for the routers to detect link failures and changes in the network topology, rather than relying on manual updates.

This could be achieved by implementing a heartbeat mechanism, where routers periodically send "hello" messages to their neighbors to check if they are still reachable. If a router does not receive a "hello" message from a neighbor within a certain time frame, it can assume that the link to that neighbor has failed and update its routing table accordingly.