Report 2: Routy, a small Routing Protocol

Lorenzo Deflorian

September 17, 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 as 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, ensuring that the routing tables are updated to reflect the current state of the network. Once the algorithm was implemented, I tested it on a small network topology, consisting of 5 routers, divided into two areas, running on two different Erlang nodes.

2.1 Network Topology

We start by defining the network topology as shown in Figure 1. We divide the whole network into 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).

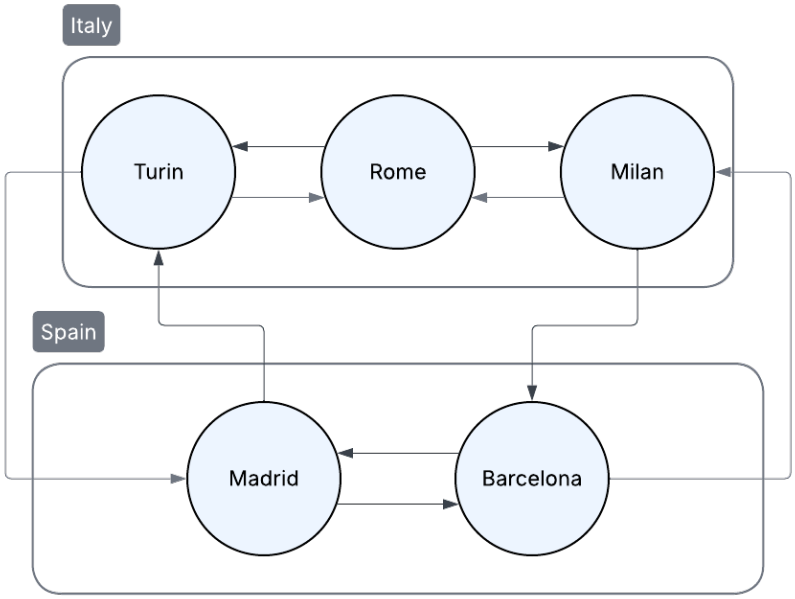


Figure 1: Network Topology

The connections between the routers are also shown in Figure 1. We set bidirectional links, all with the same cost.

3 Evaluation

3.1 Testing the implementation

After setting up the network topology as described above, we call the `broadcast` and `update` functions on each router to propagate the link-state information and compute 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 is 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 in 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 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.

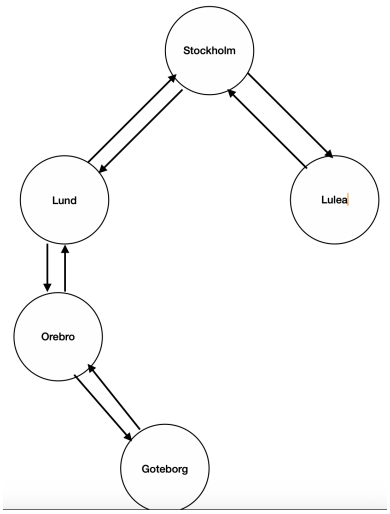
To further improve the protocol, we could consider implementing link cost variations, where different links have different costs, and the Dijkstra algorithm would need to take these costs into account when computing the shortest path.

Another possible improvement would be to implement an automatic mechanism for 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, in which 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.

5 The world

Let's try to see if we can route a message from our machine to another node running on a different machine. To do this i have asked the help of a friend, Riccardo Fragale, who has set up his topology as follows:



From my topology I will remove the connection between **Turin** and **Madrid**. Then we will connect both **Turin** and **Madrid** to **Lulea** the new router that is running in Sweden (Riccardo's network).

Let's set up the network and see if the routing still works. This is the result of calling msg:status(r3, 'italy@DORORO', 1000).

```
Italy setup complete.
msg:status({r3, 'italy@130.229.168.103'}, 1000).
Status received from {r3,'italy@130.229.168.103'}:
  Name: turin
  N: 1
  Hist: [{milan,0},
         {rome,0},
         {madrid,0},
         {barcelona,0},
         {goteborg,0},
         {orebro,0},
         {lund,0},
         {stockholm,0},
         {lulea,0},
         {turin,inf}]
  Intf: [{lulea,#Ref<0.338350561.3291742212.72605>,
                {r3,'sweden@n128-p22.eduroam.kth.se'}},
         {rome,#Ref<0.338350561.3291742212.72566>,
               {r1,'italy@130.229.168.103'}}]
  Table: [{lulea,lulea},
          {rome,rome},
          {turin,turin},
          {milan,rome},
          {stockholm,lulea},
          {madrid,lulea},
          {lund,lulea},
          {barcelona,rome},
          {orebro,lulea},
          {goteborg,lulea}]
  Map: [{milan,[barcelona,rome]},
        {rome,[turin,milan]},
        {madrid,[lulea,barcelona]},
        {barcelona,[milan,madrid]},
        {goteborg,[orebro]},
        {orebro,[goteborg,lund]},
        {lund,[orebro,stockholm]},
        {stockholm,[lulea,lund]},
        {lulea,[madrid,turin,stockholm]}]
ok
(italy@130.229.168.103)2>
```

Perfect, we can see that the routing tables are correclty set up. Now let's try to send a message from Turin to Madrid. This, time there is a new shortest path that goes through Lulea.

```
(italy@130.229.168.103)2> r3 ! {send, madrid, "Hello there"}.
turin: routing message "Hello there"
turin: gateway resolved to lulea
{send,madrid,"Hello there"}
(italy@130.229.168.103)3>
```

Turin sends a message to lulea, which then forwards it to Madrid. The message is correctly routed through the new path: **Turin** $\rightarrow$ **Lulea** $\rightarrow$ **Madrid**.

```
lulea: routing message ("Hello there")
(sweden@n128-p22.eduroam.kth.se)2>
```

```
madrid: routing message "Hello there"
madrid: received message "Hello there"
```

Let's now see what happens if we kill the erlang process running on Riccardo's machine. This simulates a complete failure of the Sweden area.

```
itsrich@n128-p22 HW2 % ./kill.sh
Erlang processes killed at: 1758108121
```

After sending a kill message to Sweden, on my machine I get:

```
turin: exit received from lulea at timestamp 1758108122792
madrid: exit received from lulea at timestamp 1758108122791
```

Looking at the timestamp from Riccardo's machine (1758108121) we can see that after 1 second both Turin and Madrid have detected the failure of Lulea.

Now, let's try to send a message from Turin to Madrid again:

```
r3 ! {send, madrid, "Hello there"}.
turin: routing message "Hello there"
turin: gateway resolved to lulea
{send,madrid,"Hello there"}
turin: gateway lulea not found in interfaces
(italy@130.229.168.103)4>
```

The message routing fails, as expected, because there is no available path between Turin and Madrid after the failure of Lulea.

To wrap it up let's update the routing tables in both areas and send a message again from Turin.

```
(italy@130.229.168.103)4> test_world:update_italy().
ok
(italy@130.229.168.103)5> r3 ! {send, madrid, "Hello there"}.
turin: routing message "Hello there"
turin: gateway resolved to rome
{send,madrid,"Hello there"}
rome: routing message "Hello there"
rome: gateway resolved to milan
(italy@130.229.168.103)6> milan: routing message "Hello there"
milan: gateway resolved to barcelona
```

```
test_world:update_spain().
ok
(spain@130.229.168.103)2> 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 correctly routed through the new path: **Turin** → **Rome** → **Milan** → **Barcelona** → **Madrid**.