Distributed Systems - Routy

Francis Gniady

September 19, 2022

1 Introduction

In this assignment we will be creating a router with a link-state routing protocol, naturally in erlang. We will then connect multiple routers together and try to send messages between then, seeing if they reach their destination even when we start to kill of some routers in the network.

2 Main problems and solutions

To start of we follow the instructions in the assignment pdf, which gives us a working albeit simple router.

We have to implement some of the modules ourselves, but these are rather generic coding problems rather than anything specific to our implementation of routers. None of which proved particularly challenging.

There were some discrepancies in the instructions such as sending the message to add a router to the pretty name of said router, and not the registered rX process name, but nothing major.
Specifically this example given in the assignment pdf.

```
(sweden@130.123.112.23)>routy:start(r1, stockholm).
(sweden@130.123.112.23)>routy:start(r2, lund).
(sweden@130.123.112.23)>lund ! {add, stockholm, {r1, 'sweden@130.123.112.23'}}.
true
```

Following the instructions in the assignment it gives us the code

```
start(Reg, Name) ->
  register(Reg, spawn(fun() -> init(Name) end)).
```

Meaning that the first argument to start is what the process will be registered as, which in the example is "r2", not "lund", thus we cannot send a message to a process with the name "lund" because it does not exist.

Other than the basics, I also took the liberty to implement flooding of routers for easily populating router tables. This is just a message telling a router to first broadcast a link-state message, just like when we send the "broadcast" message to a router, but then also make the router broadcast a flood message telling other routers to do the same. To prevent infinite loops, each router adds its name to a list in the flood message, if the router receives a flood message that contains its name, the message is ignored. This is by far not the most efficient way of doing this, but for smaller networks it works, and was easy to implement.
On top of this, whenever a router receives a new link-state message, they will update their table. These two features together let us send a single flood message to any router in the network and have the whole network populate their tables.

Whenever a router receives a down message from one of its monitors, it will update its interfaces, map, and table, and also broadcast a link-state message. This will occasionally let the network recover depending on the structure however in some cases we will have to manually tell routers update to fix the network.

3 Extra assignment

As for the extra assignment, I opted to perform the tests with a multi-node setup on my own machine. For the assignment this should not matter, as whether or not a node is local or on a friends laptop is up to erlang to solve, and would not change our local implementation in any way.

Nevertheless, I set up the following network, with each country being its own node.
After flooding the network we can send messages from stockholm to malmo routing through Sweden only, and if we stop the lund router, stockholm will automatically update its tables and route messages to malmo through cologne and around the world.
There are some limitations to this self-healing behavior without our intervention. Say for example we create a link from hamburg to lund, now to send a message from cologne to malmo, the quickest way is to route to hamburg, and then directly to Sweden. If we now stop lund, hamburg will update its tables and send any messages that are going to malmo to cologne instead, however hamburg is the only router that updated its tables, so if we try to send a message again from cologne to malmo we will send it to hamburg, and hamburg will send it back to cologne creating an infinite loop. We can of course fix this by making berlin broadcast and update its links, or by flooding the network again. There are of course ways to solve this issue, we

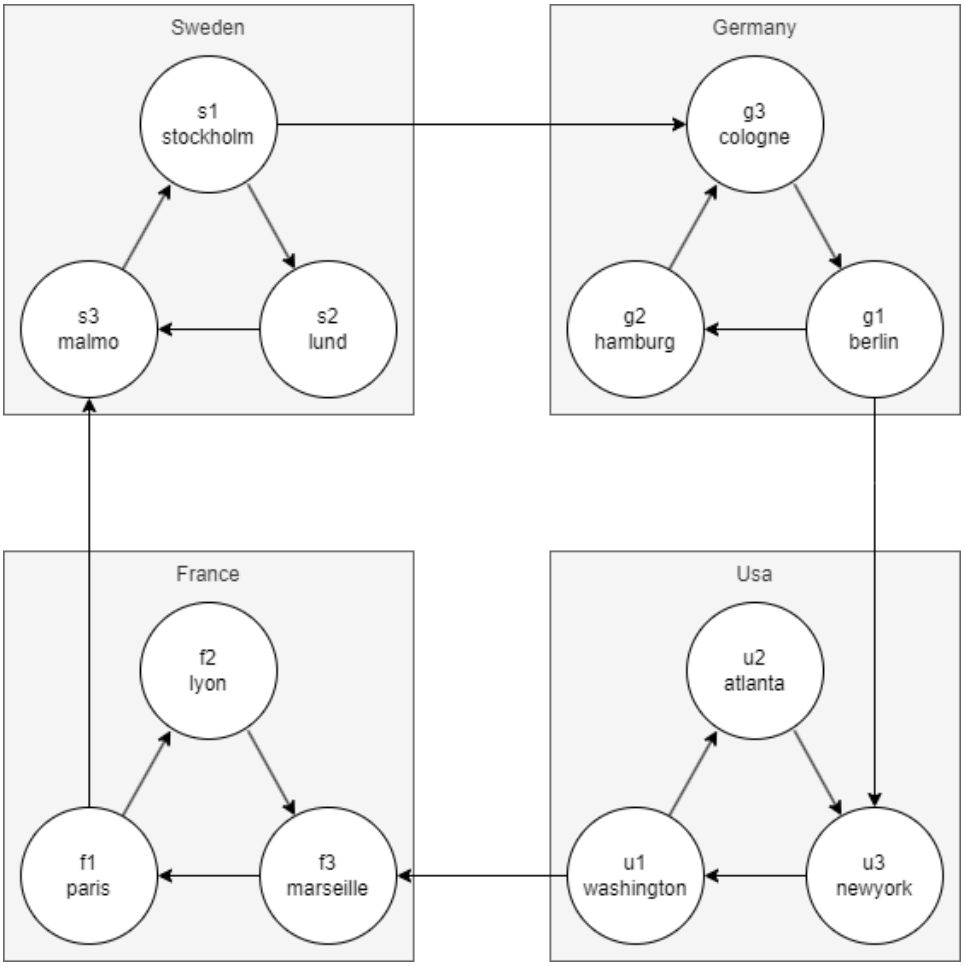

Figure 1: Network

could for example flood the network every time a connection changes or a router dies, or making a smarter algorithm to propagate changes across a network instead of "resetting" all connections by flooding. However I will not be attempting any of these solutions in this assignment.

4 Conclusions

In the end, this assignment was a nice way to learn about link-state routing. Of course our implementation was rather simple, and could not withstand certain changes to the network without our intervention. It would have been fun to have a way of visualizing this network and maybe seeing it change and propagate messages in real-time, however that is a project for another day.