Report 2: Routy

Corentin LEROY

September 19, 2018

1 Introduction

The goal of this assignment was to develop a link state routing protocol in Erlang. The objectives were to understand the structure of a router process, how real network topology is represented in a map and how a routing table is built, how routers are connected and how they handle network failures.

2 Main problems and solutions

Main problems and solutions The main difficulty here was to understand the handling of lists and tuple objects in Erlang, and all the operations done by a router process to treat these objects. I created five Erlang modules:

- `map`

  This module aims at giving a virtual representation of the topology of connected routers. It was the first program I wrote so here I had to read the Erlang documentation and understand how lists and tuples work in this language. I also had to read about all the built-in functions to handle them. The only method that caused me problems was the `all_nodes()` function. I had to apply a function to each element of the map, return a list of all results and remove duplicates. I learnt how to write anonymous functions in Erlang and read the lists documentation to find out about the `flatmap()` and `usort()` functions that did what I wanted. The difficulty was essentially to find the right BIFs.

- `dijkstra`

  The goal of this module is to compute a routing table based on a map and a list of gateways. It was by far the hardest to code for me. The iteration of the algorithm took me a lot of re-readings of the subject and thinking to understand how it works and how it generates a proper routing table. In the end it is not that complicated, but the problem was to juggle the three lists (a sorted list of nodes, a map of the nodes, and the routing table to construct), through BIFs and lists comprehensions, etc.

- `interface`

  This module gives a representation of a router process. Each router has a set of interfaces, each one corresponding to another router process it knows about. This program was quite straightforward to code.

- `history`

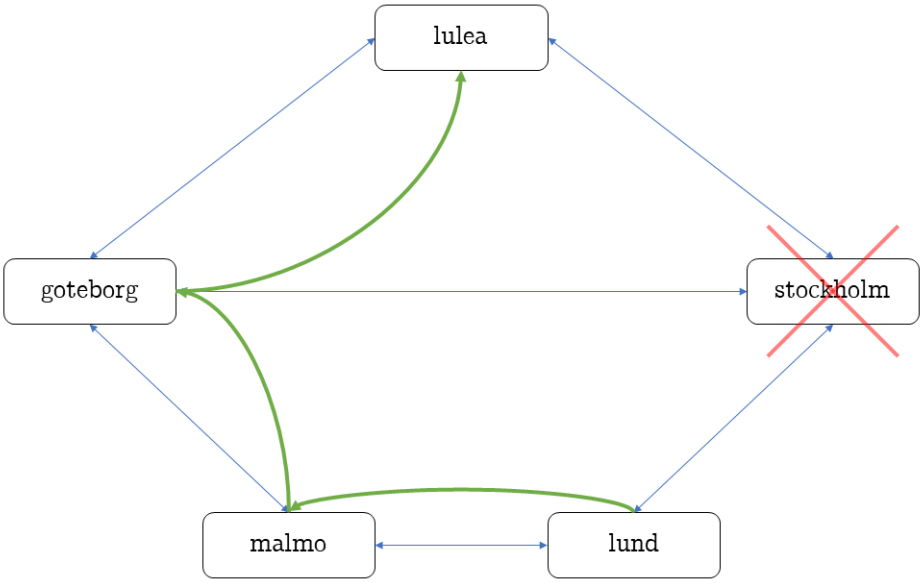
  The goal of this module is to keep track of the messages received from other nodes and determine if they are old or new, to avoid delays of old messages and circular loops. This one did not cause many difficulties either.

- `routy`

  Finally, the router process with which the user communicates, that checks for incoming messages from other router processes and replies to them. All the code except for the `displayStatus()` function was given. The core of this module is a recursive function that handles messages received either from the user or from other routers and does the corresponding operations.

3 Evaluation

I created different routers and connected them to see that the protocol works well. For example, I ran five routers to represent this scenario.



I checked that all connected routers could communicate. And I also checked that the routing of messages still works if one of the routers goes down. To do that, I stopped `stockholm`. Before that, when `lund` tried to send a message to `lulea`, it passed through `stockholm`, but after this latter is shut down, the message passes through `malmo` and `goteborg` before finally reaching `lulea`. So the protocol handles well network failures.

4 Conclusions

Through this seminar I have learnt how a routing process is structured and how it builds a routing table. I have seen the deep operation of a link-state protocol and that it is robust to routers going down. As a side effect I have improved my understanding of Erlang language and especially lists handling. The router process could be improved by being more user-friendly for the control messages. And building a network is quite tedious; it could be a bit more automated.