Report: Routy, a small routing protocol

Alexandre Tamborrino

September 12, 2012

1 Introduction

During this seminar we have implemented in Erlang a routing protocol based on the well-known link-state routing protocol.

In our implementation, a router can broadcast link-state messages to all their gateways (neighbor routers) in order to tell them what are all its gateways. These neighbor routers process the link-state message (save the information) and broadcast it to all their own gateways. Thus, each link-state message of a router goes through the whole network, therefore all the routers know what are the gateways of the router who has initially send the link-state message. If each router does this, all routers will have a complete map of the network.

Then each router can compute a routing table to know where to route a message that has to go to a specific destination. The processing of the table uses the Dijkstra algorithm to find shortest paths in the network. Thus, messages can be routed effectively. We will see the limitations of this implementation in the conclusion.

2 Main problems and solutions

As I did not have the "very helpful code skeleton file" when I implemented Routy, I encounter some difficulties about how to solve things in a functional programming way. For example, in the Dijkstra algorithm, I was wondering how to iterate over a list updating another list at each iteration. As variables are immutable, I was wondering how to update the list in a way that each iteration uses the updated version of the previous iteration. Then, I look to the lists:foldl/3 function and see what I had to do. Here is the code (part of the Dijkstra algorithm):

```
NewSortedList = lists:foldl(fun(El,SortedList) ->
                    update(El,L+1,Gateway,SortedList)
                    end, T, Reachables);
```

Concerning the history data structure, I have used an Erlang *dict* in order to gain performance. Indeed, this data structure is contained in each

router of the network and has N entry (N is the number of routers in the whole network). In order to scale, the hist:update function has to be fast (*dict* has a better complexity than a classic list of tuples in O(N)).

3 Evaluation

In order to test my implementation, I set up 3 different Erlang nodes (VM), each representing a country. I wrote an Erlang test module that starts several routers in each country, makes the connections between routers, then broadcasts and updates each router. Figure 1 shows the map of the test network.

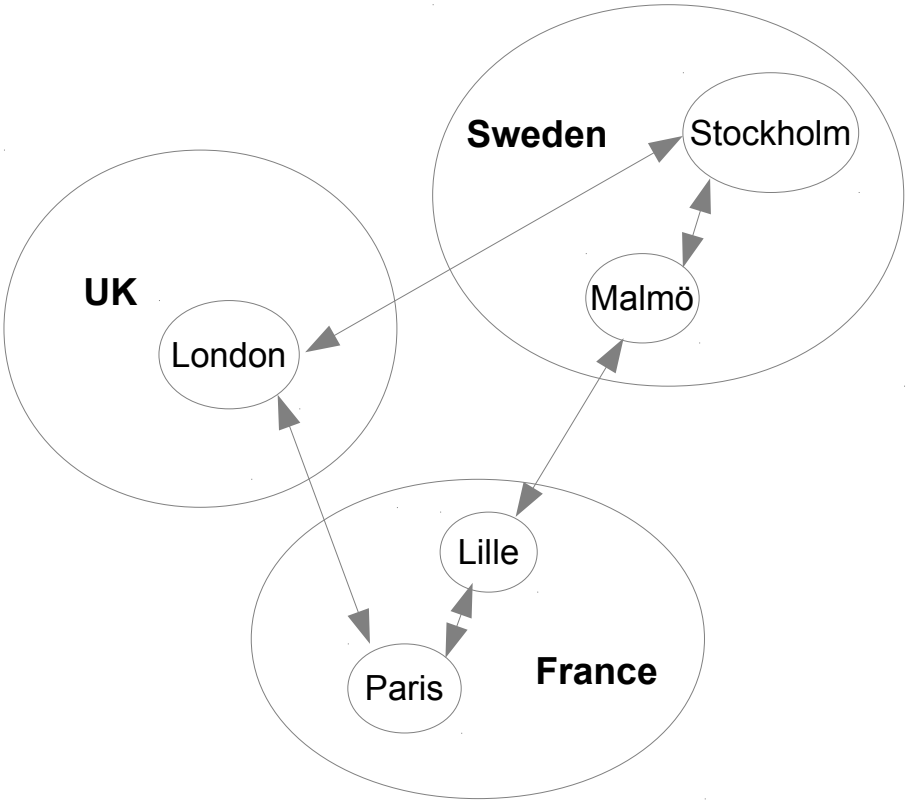

Figure 1: Network map for testing

Then I have tested to route a message from Paris to Stockholm. Here is

the result of io:format on the different shells:

```
On "France Node":
  $> paris ! {send,stockholm,"Hello Stockholm, this is Paris."}.
  paris: routing message (Hello Stockholm, this is Paris.)

On "UK Node":
  london: routing message (Hello Stockholm, this is Paris.)

On "Sweden Node":
  stockholm: received message Hello Stockholm, this is Paris.
```

We see that the message has been correctly routed using the shortest path between Paris and Stockholm.

Now, I shutdown the UK Node. Paris and Stockholm receive the exit message ("DOWN") of London. Then, I broadcast and update Paris and Stockholm. I re-try to route a message from Paris to Stockholm:

```
On "France Node":
  $> paris ! {send,stockholm,"Hello Stockholm, this is Paris."}.
  paris: routing message (Hello Stockholm, this is Paris.)
  lille: routing message (Hello Stockholm, this is Paris.)

On "Sweden Node":
  malmo: routing message (Hello Stockholm, this is Paris.)
  stockholm: received message Hello Stockholm, this is Paris.
```

The message is correctly routed in spite of the fact that UK has crashed, therefore our protocol is fault-tolerant.

4 Conclusions

During this seminar I have understood and implemented a routing protocol based on link-state. Fault-tolerancy is handled (if some routers crash, messages are still successfully routed). Scaling is limited because each router builds a routing table for the entire network. This can be problematic in very large network. It would be interesting to implement *default routes* to overcome this issue.