Seminar Report : Routy

Vincen Delaunay

September 25, 2014

1 Introduction

Routy is a simple link-state router in Erlang.

This seminar aims to offer a grasp on link-state routing protocols, such as the famous OSPF. Routing is a typical distributed issue, as many entities have to cooperate and share their truncated knowledge of the network in order to deliver. They also have to be able to adapt to the evolutions of the network's topology.

The instructions helped to isolate the main steps of the implementation. Namely : construction of a map of the network, computation of the routing table using Dijkstra's algorithm, management of the interfaces, of an history, and finally building Routy on top of all this.

2 The map

The map is a representation of all the unidirectional links between nodes, as known by our router.

The main complication here was just to get familiar with the lists module. The function all_nodes was probably the less straight forward, and I went through several implementations until I found one I was satisfied with :

```
all_nodes(Map) ->
    lists:usort(lists:foldl(fun({Node,Links},A)-> [Node|Links++A] end, [], Map)).
```

3 Dijkstra

From the map of nodes and their links, our router needs to construct a routing table. That is, to each possible destination, to know the best gateway. This table is built using the famous Dijkstra algorithm.

The implementation goes in two steps. First, implementing a sorted list, with the apropriate methods to manipulate it. Then, using this structure to implement the protocol.

About the sorted list implementation, an interesting method is replace, used to edit one entry, because we want to keep the list sorted at any time. Rather than appending the entry to the list, and then sorting the entire, we use the function lists:keymerge/3 to simply insert our entry at the right position :

```
replace(Node, D, Gateway, Sorted)->
    lists:keymerge(2,[{Node, D, Gateway}],lists:keydelete(Node, 1, Sorted)).
```

But the most difficult part of this section (and of the entire assignement if you ask me), was the iterate function. It is where Dijkstra comes to play :

```
iterate(Sorted, Map, Table) ->
  case Sorted of
      [] -> Table ;
      [{_,inf,_}|_] -> Table ;
      [{Node,Dist,Gateway}|T] ->
          Sorted_ = lists:foldl(
              fun(N, S)->update(N, Dist+1, Gateway, S) end,
              T,
              map:reachable(Node, Map)
          ),
          iterate(Sorted_,Map,[{Node,Gateway}|Table])

  end.
```

The last part worth mentioning is a trick I used for the table function. It is called with a list of Gateways and a Map, and needs to instanciate a sorted list from it to call iterate. Instead of working heavily on the Map, formatting every entry regarding of wheter or not it is a gateway, and then sorting this big list, I decided to simply remove every gateway from the Map, and then appending the list of Non_gateway_nodes to the list of Gateway_nodes :

```
table(Gateways, Map) ->
    Non_gateway_nodes = lists:map(
        fun(N) -> {N,inf,unknown} end,
        lists:filter(
            fun(N) -> not lists:member(N,Gateways) end,
            map:all_nodes(Map)
        )
    ),
    Gateway_nodes = lists:map(
        fun(N) -> {N,0,N} end,
        Gateways
    ),
    iterate(Gateway_nodes ++ Non_gateway_nodes, Map, []).
```

4 Interfaces and History

The router needs to expose methods to add, delete and address it's interfaces. It also needs some way to avoid having some packets circulating forever between routers, so we implement an history (another common solution is to use a ttl flag).

There isn't really much to say about these implementations, as they are really obvious.

5 The router

The router's commands can be split in three main categories : Managing interfaces, Updating the link-state, and Routing messages, and a less important one, which we will call Control, with status, stop, end, and an extra : print.

There is even less to say about this implementation, since it was given and there were just a few typos to fix.

One not-so-relevant point though, the given implementation of status is prone to be maliciously exploited to crash the router

```
router ! {status, malicious}.
```

Here is a fix :

```
{status, Pid} ->
try
    Pid ! {status, {Name, N, Hist, Intf, Table, Map}}
after
    router(Name, N, Hist, Intf, Table, Map)
end;
```

To go further this road, we can also use a third party router to relay our attack ;

```
thirdParty ! {status, {victimRef,'victimHost'}.
```

And there is no fix for that, except for changing the specifications.

6 Tests

I wrote unit-tests during the process of building Routy (except for the router itslf).Most of the tests were given in the assignment anyway, I just had to automatize them.

Automated compilation of the project :

```
SOURCES = [map, dijkstra, intf, hist],
lists:foreach(fun(X)-> case cover:compile(X) of {error,_} -> error("Compilation
```

The frst two tests :

```
    test("map:new",map:new()==[]),
    test("map:update",map:update(berlin, [london, paris], [])==[{berlin,[london,par
```

These unti-tests proved to be really usefull since Routy puts together several modules with a lot of small methods. It was easy to build the tests and helped make sure every of my implementation actually matched the specification.

7 Evaluation

Everything I tried worked just fine, and with several routers connected I could visualize the update of the routing table when the changes updated thanks to the broadcast. I didn't do any benchmark though, since I don't really know what to measure, how to do it, and it didn't seem to be expected in this assignment.

8 Conclusions

When first looked at, Routy looks really ambitious. But during the entire exercice we are guided so everything goes smoothly and every layer of the implementation ends up being clearly understood.

Also, aside from what we learnt about link-state protocols, I found this assignment to prove really interesting by having to go hands-down with real functionnal programming. I wasn't really familiar with the use of map, filter, foldl and alike, and really enjoyed trying to find a nice way to solve any problem encountered using these new tools.