Report Routy: a small routing protocol

Anestos Nikolaos Ektoras

September 24, 2014

1 Introduction

In this seminar, we implement a link-state routing protocol in Erlang. We use Dijkstra algorithm to compute a routing table.

2 Main problems and solutions

Since we didn't have anything other than the function declarations, this exercise was very challenging, (especially map, and dijkstra modules).

2.1 Module: map

Map module is needed in order to update efficiently a given map, by replacing an entry with a new entry. It took me notable amount of time and effort to get this part running, considering that we didn't have any skeleton code, or some extended examples of what this module should do. When i got hold of the idea, it was actually quite easy to do this, using some erlang lists functions such as lists:keyfind/3 , lists:keydelete/3, lists:usort/1.

2.2 Module: dijkstra

The second part of the exercise, was the most challenging, since we should understand and implement the Dijkstra algorithm. The functions entry/2, replace/3 and update/4 were straight forward. Functions iterate/3 and table/2 required a lot of work to get them working. I managed to solve this puzzle after a while with the help of lists:foldl/3 and the use of a function i created (initSortedList/3 which constructs a list with dummy entries for all nodes with the length set to infinity, inf, and the gateway to unknown and entries of the gateways which have length zero and gateway set to itself.) respectively.

```
initSortedList(List,Gateways,NewList)->
   case List of
      []->
```

```
      lists:keysort(2, NewList);
   [H|T] ->
      case lists:member(H,Gateways) of
         true ->
            initSortedList(T,Gateways,[{H,0,H}]++NewList);
         false->
            initSortedList(T,Gateways,[{H,inf,unknown}]++NewList)
   end
 end.
```

2.3 Module: intf

The intf module is mostly tuplelist manipulation and was relatively easy to implement.

2.4 Module: hist

The hist module was a bit of a puzzle at first. Since we didn't have a guide on how to create the data structure. After a small debate with a few colleagues, and a peek at routy module, we found out that it should be a list of tuples [Name,Number] with the name of the node and the number of messages we have seen from that node.

We could fool the history and let it think that all messages are old by creating an entry with a string as the number of messages.(N>Counter will always fail and the update function will return old).

```
update(Node, N, History)->
   case lists:keyfind(Node, 1, History) of
      {Name, Counter} ->
         if
            N > Counter ->
               {new, lists:keyreplace(Name, 1, History, {Name, N})};
            true ->
               old
         end;
      false ->
         {new, new(Node)  ++ History}
   end.
```

2.5 Module: routy

The routy module was very good documented in the exercise, thus easy to understand.

Key part here was to understand the link-state messages. The idea is that each time a new node is connected to the network, it has to broadcast (tell it's neighbors about himself). The nodes receiving this message, either do nothing (if they have received the same message earlier), or broadcast as well to let eventually everyone in the network know about the new node.

3 Evaluation

In order to test the code and behavior of the system i ran a test the course assistants provide me in the lab, which starts 2 routers, adds one another so they both have the process identifier of each other, broadcast and update. At first i was not 100% sure of what was happening, so i added a printout to router/6 function which shows in the console the current state of each node. This gave me a new level of perspective and also helped me find a few syntactic errors.

Then i opened an erlang shell using a different name and started a router with a new city. This way i was able to test that even the not directly connected city, could send the new node messages and visa versa.

4 Conclusions

The system we implemented handle node crashes seamlessly, since each time such thing happens it creates a new map, making it fault-tolerant. On the other hand, each node builds a routing table for the entire network, which causes bad scalability.