Report 2: Routy Seminar Report

Alberto Lorente Leal

14 September 2011

1 Introduction

In this seminar, we cover in this case how the routing of messages in IP is handled. We see in detail how a routing algorithm works, in this case we look in detail the OSPF and also we learn how to practical implement the dijkstra algorithm. It is an important subject to see in detail, because it helps us see how messages that are sended from one node gets to the other node following a path are routed by the different nodes. Thanks to dijkstra, we also manage to reduce the amount of time to send a message as this algorithm searches for the shortest route available in a graph of nodes. In a summary we handle the following:

- We get the structure of an OSPF packet which is based on a link stated packet which contains information about it neighbors and also how much it costs to reach them. Thanks to this we can know the structure of the whole network.
- We see how to handle lists and tuples in more detail, we see how to keep a record of the information and how to search and modify them in Erlang.

2 Main problems and solutions

2.1 Problems during the development of the solution:

There where not many mayor problems during the development of the solution but i would remark the following issues:

- There where moments when try to solv the problem in which it was impossible to know where was the error in the code. This made me move in circles and try different implementations and examples to see the error. This really became a huge nightmare until I managed to learn how to run the incorporated debugger in Erlang. This valuable tool helped me to identify the error in the code (it was in the end a typo in my implementation). Due to this small error, i did lose a lot

of time which i could have saved if i knew earlier how to launch the debugger.

2.2 Adding missing pieces of Routy:

In order to complete the missing bits and pieces, I searched on the documentation of Erlang for information about keysearch/3, keydelete/3, map/2 and fold/1.

For keysearch we needed to give a key, the position of the tuple in order to compare with the key and the list where we are going to search in. Similarly the keydelete works the same, only that in this case; it returns a new modified list without the entry we wanted to delete. In the case of fold i didn't manage to understand its functionality. Here I will list some code examples

Map Module:

In the *reachable(Node, Map)* we had to build a new list taking only the nodes which are reachable to this node. The missing piece looked like the following:

```
reachable(Node, Map) ->
        case lists:keysearch(Node, 1, Map) of
        :
        end.
```

In the *update(Node, Links, Map)* we update the map given a set of links reachable from a node. So the key in this case is Node, which will be in the first position of the tuple:

```
update(Node, Links, Map) ->
    case lists:keysearch(Node, 1, Map) of
        {value, {_, _}} ->
                [{Node,Links}|lists:keydelete(Node,1,Map)];
        :
        end.
```

Dijkstra Module:

In the dijsktra module we finished the implementation of the *iterate()* , *table() and update()*. In this case we need to add 2 cases for the iteration, one when we receive an empty list in which we reply with the table. The other case when we receive a node with infinity weight in the first place of the list, in which we return the table. I will list some code samples from the important parts which are mainly related with OSPF.

```
table(Gws, Map) ->
        Nodes = map:all_nodes(Map),
        Rest = lists:filter(fun (X) ->
```

```
                        not lists:member(X, Gws) end, Nodes),
        Direct = lists:map(fun (Nd) -> {Nd,0,Nd} end, Gws),
        Indirect = lists:map(fun (Nd) -> {Nd,inf,na} end, Rest),
        Sorted = lists:append(Direct, Indirect),
        iterate(Sorted, Map, []).

iterate([], _, Table) ->
        Table;
iterate([{_,inf,_}|_], _, Table) ->
        Table;
iterate([{Node, N, Gw}|Nodes], Map, Table) ->
        Reachable = map:reachable(Node, Map),
        :
        iterate(Updated, Map, [{Node,Gw}|Table]).

update(Node, N, Gw, Nodes) ->
        M =  entry(Node, Nodes),
        if      N < M ->
                replace(Node, N, Gw,Nodes);
        :
        end.
```

History module:

```
update(Name, X, History)->
        case lists:keysearch(Name, 1, History) of
        {value, {Name, Y}} ->
             if
                X > Y ->
                        {new, [{Name, X}|
                                lists:keydelete(Name, 1, History)]};
                true ->
                        old
                end;
                false ->
                        {new, [{Name, 0}|
                                lists:keydelete(Name, 1, History)]}
                end.
```

2.3 Testing of Routy:

I did some testing with the code provided in the assignment information to implement the routy module. Before that I checked that the interface module and History module were working as they should. In this case I spawned

an erlang shell named sweden with 2 routy processes named lund and stockholm. I created then another node named upsala and started adding the interfaces for the connections. I organized a network where lund was connected to stockholm and upsala, then stockholm connected with upsala and upsala with lund. After that we sended broadcast messages so the nodes could identify their neighbors and later I updated the routing tables with the update command.

3 Conclusions

Thanks to this problem, i've learnt how a routing protocol works in detail. I think this a quite important thing to get to know because Internet works thanks to routing protocols. Routing is an important aspect in distributed systems because we want that the messages sent to arrive correctly to the other side and using the smallest amount of time. We learn how the structure of an OSPF message is and how it works implementing the protocol with Erlang. Also at the same time we get to handle lists in Erlang with the methods keysearch, keydelete and map which are useful methods when managing and doing operations with lists.