Routy: a small routing protocol

Antonios Kouzoupis <antkou@kth.se>

September 25, 2014

1 Introduction

In this seminar we have implemented *Routy*, a small yet reliable routing protocol. The topics we have covered are how a link–state protocol works and how to achieve consistency of a network when nodes stop working. As a consequence of this seminar we had an even deeper look in Erlang.

Routy is a link–state protocol, which mean that every node in the network should construct a connectivity map to other nodes. Afterwards, using Dijkstra's algorithm, it finds the shortest path to a node and finally it builds the routing table. Routy also uses a monitoring system in order to detect and emend network topology changes such as failures.

2 Main problems and solutions

Personally, I believe that the most difficult part of the seminar was to obtain a solid understanding of how the protocol works. How to use the *Map*, the *Sorted List* and how to combine all the different components together in order for the router to work. So reading the seminar paper and the book was of crucial importance to continue with the implementation.

Also, implementing Dijkstra's algorithm was not trivial. It consisted of many operations that you had to be completely sure they worked correctly before you proceed further. So testing was the only solution. Testing every functionality separately and then the whole algorithm. It was very important for the algorithm to produce the correct result since it is the heart of our routing protocol. In Figure 1 is illustrated how the algorithm is implemented. Edge cases of function are when the *Sorted List* is empty or the first element's distance is "inf". Then for every node that is reachable from the node in question I update the entry in the Sorted List with new weight and gateway. Finally I fill the routing table with the current node and gateway.

History module was also of great importance since, an error in keeping track of old messages led in infinite loop while broadcasting. Finally the *broadcast* and *update* procedure was not very clear to me so I had to run

```
update_all(Reachable, N, Gateway, List) ->
    case Reachable of
        [] -> List;
        [H | T] ->
            TmpList = update(H, N, Gateway, List),
            update_all(T, N, Gateway, TmpList)
    end.

iterate(Slist, Map, Table) ->
    case Slist of
        [] -> Table;
        [Head | Tail] ->
            case [Head] of
                [{}] -> Table;
                [{_, inf, _}] -> Table;
                [{LstNode, LstN, LstGW}] ->
                    Reachable = map:reachable(LstNode, Map),
                    Slist0 = update_all(Reachable, LstN + 1,
                        LstGW, Tail),
                    Table0 = lists:append(Table, [{LstNode,
                        LstGW}]),
                    iterate(sort(Slist0), Map, Table0)
            end
    end.
```

Figure 1: Dijkstra's algorithm

many tests and gather the output from multiple Routy instances to figure out how routers are advertised and update their map and routing table.

3 Evaluation

For the evaluation procedure, I wrote a test case which started five routers, *Stockholm, Borås, Luleå, Umeå* and *Lund*. They were connected to each other according to the topology illustrated in Figure 2.

The first scenario was to send a message from *Stockholm* to *Lund*. As shown in the topology, there are two available paths, one which uses *Borås* as a gateway and another which uses *Luleå* as a gateway and pass from *Umeå* to reach *Lund*. Routy chooses to use the first path since Dijkstra's algorithm indicated it as the shortest one.

The second scenario was to send a message from *Stockholm* to *Umeå* but with *Luleå* node down. So after I started all the nodes, broadcast and updated the interfaces, map and routing table, I closed the *Luleå* node and update the map and routing table of the remaining nodes. Normally the message would pass through *Luleå* since it is the shortest path but because it is down, necessarily it will pass through *Borås* and *Lund*. The "update" function is not executed automatically, so we must explicitly call

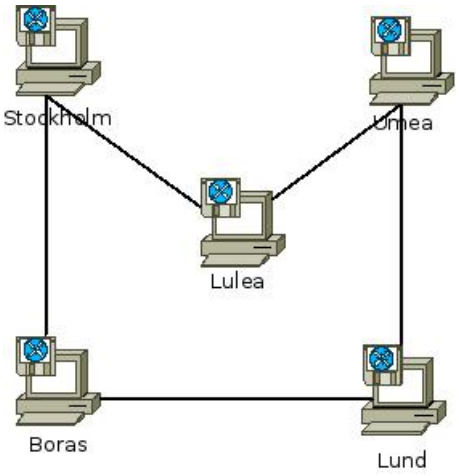

Figure 2: Routy network topology

it, otherwise our message will not be able to route. This one proves that indeed Routy adapts to network topology changes, if a node is down it constructs another path.

4 Conclusions

In this seminar we have learnt how link–state protocols, like OSPF work. We implemented a router based on link–state and Dijkstra's algorithm. It was really interesting to see messages routed through various nodes and construct new paths to deal with failures. Finally, through the seminar homeworks I learn more and more about Erlang.