# Report 2: Routy Router Assignment

Addi Djikic – addi@kth.se

September 21, 2017

## 1 Introduction

For this assignment, a link-state routing software based program have been made that routs messages between logical names, such as *london* or *stockholm*. First the main methods that the routing uses was implemented, such as where to store the nodes and links and how we find and update them. The algorithm that the routing used was based on Dijkstras shortest path algorithm, we will see in the evaluation the work on how Dijkstra will always route our desired message via the shortest path.

The main topics covered are how the structure of a link-state protocol works, how do we maintain a consistent view of our router-network, and how network failures may occur in the system of nodes.

It is important to understand the basics of routing, for example in how we can optimize the network transmitting speed by taking the shortest distance and how we can trust that are packages arrive or what happens when they do not.

This is strongly related to distributed systems because we use distributed nodes to carry on a message (or a calculation for instance) through the network, process it, and then return it back to the origin, which reflects quite much on the topic.

## 2 Main problems and solutions

The main problems were to build all the necessary modules for the routing, and perhaps the hardest part was to implement the Dijkstra algorithm, first good knowledge on how the shortest path algorithm works is needed, and then carefully understand the test cases and what they should produce. The Erlang `lists` library were used regularly to find, delete and keep track of the necessary functions. Which turned out to be quite helpful, as in the Dijkstra example where we use the `lists` library to find the best solutions. Also there were no test cases for the *history* and the *interface* we built, which made it hard to know if there were fully correct. However, a test suite were built to try out the whole procedure, which is explained in evaluation. It

tests that we can route a message via the shortest path (seen in Fig. 1), and then the links are modified (as seen in Fig. 2) to see if Dijkstra can handle the new path. Also, the consistent view has to be contained, this is also shown in evaluation of the test suite by making every node broadcast the information so that the other nodes know about it. Every node also needs to update their routing-table, in this case the Dijkstra will find the shortests path from the node to the correct gateway.

# 3 Evaluation

To evaluate that our routing procedure works, handles the link-state messages and keeps an consistent view, several test steps were done firstly on the network as in Fig. 1. First the shell was started with an assigned IP as in:

```
>erl -name sweden@130.123.112.23 -setcookie routy -connect_all false
```

Then the nodes were started via routy as:

```
>testMyRouty:startRoutingNodes('sweden@130.123.112.23').
   stockholm ! {add, usa, {washington, ProcessIdentifier}},
   stockholm ! {add, bosnia, {sarajevo, ProcessIdentifier}},
   washington ! {add, bosnia , {sarajevo, ProcessIdentifier}},
   sarajevo ! {add, uk, {london, ProcessIdentifier}},
   london ! {add, sweden, {stockholm, ProcessIdentifier}}.
```

Where we see that this is the connections as in Fig. 1. Now that we have establish how the nodes are connected we need to `broadcast` so that the nodes know of each other as:

```
broadcast()->
    stockholm ! broadcast,
    washington ! broadcast,
    sarajevo ! broadcast,
    london ! broadcast.
```

The link-state message in `routy` will use broadcast with the `intf` and `map` to obtain a consistent view and also notify if if there is a new reachable node, and updates the Map.

Finally we will update the routing table by calling the function:

```
update()->
    stockholm ! update,
    washington ! update,
    sarajevo ! update,
    london ! update.
```

Where we use `Dijkstra` to find shortest gateway path and list them with the help of the `intf` module. When all this is done we can use the send message, for example as:

```
send()->
    stockholm ! {send, bosnia, "Hi Sarajevo! Whats happening!? - Stockholm"},
    sarajevo ! {send, sweden, "Nothing you know, just chillin - Sarajevo"}.
```

to route our message via the shortest path. It will now look for the possible path in the table, but if there is non existing we will simply just drop the package, the routing procedure will not crash.

The output for our message will be as follows:

```
(sweden@130.123.112.23)10> testMyRouty:send().
sweden: routing message: Hi Sarajevo! Whats happening!? - Stockholm
sweden Going Through: bosnia
bosnia: received message: Hi Sarajevo! Whats happening!? - Stockholm
bosnia: routing message: Nothing you know, just chillin - Sarajevo
{send,sweden,"Nothing you know, just chillin - Sarajevo"}
bosnia Going Through: uk
uk: routing message: Nothing you know, just chillin - Sarajevo
uk Going Through: sweden
sweden: received message: Nothing you know, just chillin - Sarajevo
```

Where we see that our message from Stockholm took the shortest path directly by looking at Fig: 1, which indicates that our Dijkstra works. To confirm this we now remove the direct link between Stockholm and Bosnia as in Fig. 2.

```
remove() ->
    stockholm ! {remove, bosnia}.
```

We then broadcast again and update our routing table once again, and after the send function was ran the output became:

```
(sweden@130.123.112.23)14> testMyRouty:send().
sweden: routing message: Hi Sarajevo! Whats happening!? - Stockholm
sweden Going Through: usa
usa: routing message: Hi Sarajevo! Whats happening!? - Stockholm
usa Going Through: bosnia
bosnia: received message: Hi Sarajevo! Whats happening!? - Stockholm
bosnia: routing message: Nothing you know, just chillin - Sarajevo
{send,sweden,"Nothing you know, just chillin - Sarajevo"}
bosnia Going Through: uk
uk: routing message: Nothing you know, just chillin - Sarajevo
```

```
uk Going Through: sweden
sweden: received message: Nothing you know, just chillin - Sarajevo
(sweden@130.123.112.23)15>
```

Where we notice that our message takes the longest path through Washington, because the link does not exists anymore between Stockholm and Sarajevo. But our message still came through.

# 4 Conclusions

This assignment gave quite good knowledge on how a routing link-state protocol works and how we can send messaged through different nodes in an efficient way, depending on how they are connected. In the evaluation of our test suite we can see that we managed to obtain a consistent view, continuously updating the routing table, and sending new messages via the shortest paths we could find. This was not an easy task to implement, some modules took a really long time and were complex. However, it gave a good perception on how many processes that needs to be implemented before one could simply send a message to a different router, along with all the things that you need to think of and keep track of, such as old messages and if a node exists in the network or not.

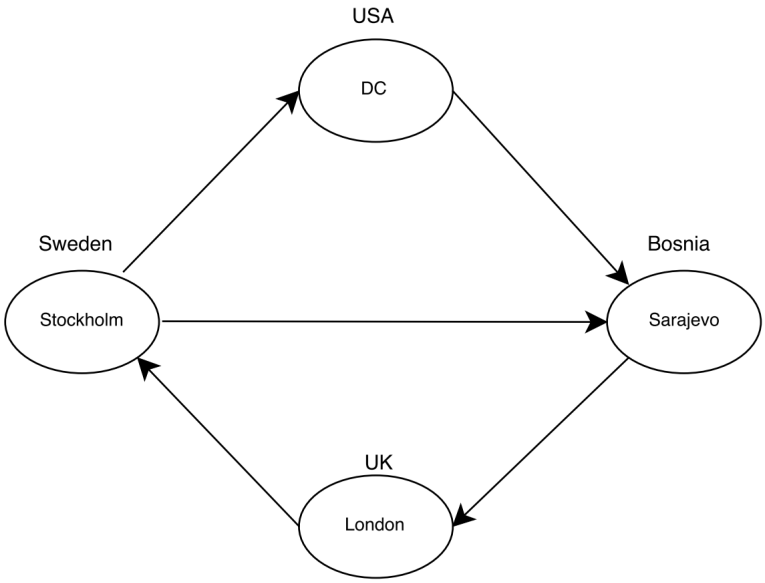


Figure 1: Showing all connected links for the test suite network where Stockholm are connected to both Sarajevo and DC.

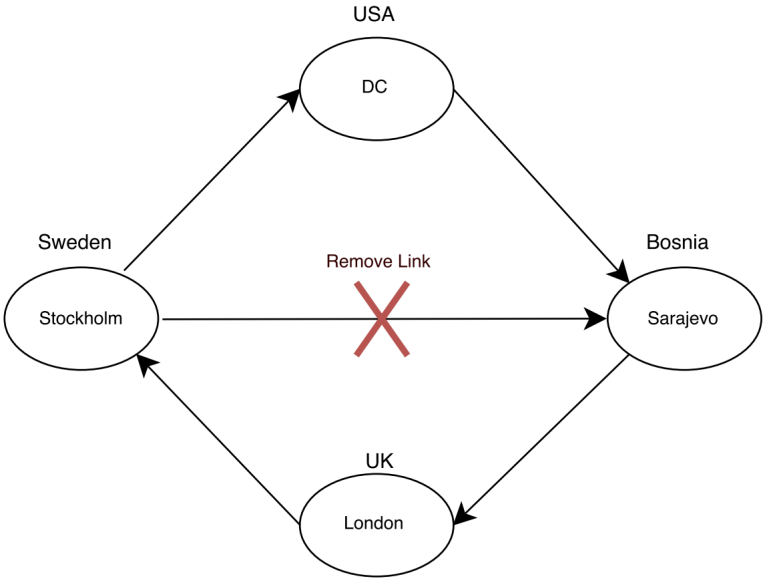


Figure 2: Showing the same network as in Fig. 1 but with the broken connection between Stockholm and Sarajevo, which forces Stockholm to send messages to UK via DC instead (a longer path).