Report 2: Routy - a small routy protocol

Abdullah .

September 22, 2021

1 Introduction

This seminar consisted of implementing a link-state routing protocol in Erlang. The goal was to know about the structure of a link-state routing protocol and how a consistent view of the networks is maintained using interfaces.

The implemenation of routy should be able to route messages between different routing processes having logical names such as "berlin", "stockholm" etc. To be able to do this "link state messaages" and "routing table" comes in handy to find a suitable gateway for node to send its message to.

The report shows the different parts implemenation of the routy and the approach to solving some difficulties faced with some handy list operations.

2 The Map

The Map represents the network structure used by the routers to calculate the routing table. It is represented as list = [{Node, [Links]} ,{...}, ... }] where Node is the name of node like "berlin" and links is a list of all the gateways reachable from the Node like "[rome,paris]".

To create a new map, update it, find all reachable nodes and return all nodes some functions needed to be implemented named

```
new/0, update/3, reachable/2, all_nodes/1.
```

To implement the functionality from scratch for each of those fuctions was a difficult part so functions from the listslibrary were used. The functions used were

```
lists:keyfind/3, lists:keyreplace/3, lists:foldl/3.
```

The keyfind and keyreplace functions were used to find and update nodes in the map while foldl was used to reduce the list of nodes and links to a single array containing all nodes. If one is familiar with javascript then he/she will know that foldl in erlang works the same as array.reduce in javascript.

3 Dijkstra

Routing Table is one of the major steps in link-state routing protocol implemenation. The routing table calculates the shortest path to reach a node from its gateway and it is updated each time when nodes are added or removed in the network.

The routing table is constructed using the Dijkstra Algorithm which is used to find the shortest path. The algorithm steps were explained in detail inside the routy document provided. Different methods like

```
entry/2, replace/4, update/4, iterate/3, table/2, route/1
```

were written to implement the algorithm described in the document.

Some of the functions were straight forward to implement because of very good explanation in document. The difficult part to implement was the "iterate/3" function which is also called as the heart of the algorithm. The iterate function is recursively called and it finds reachable nodes for each given node from sorted list and then for those reachable nodes it updates the sorted list with shortest path if any. At the end the entry is added to the table which is called at the end "The Routing Table".

To implement these all functions again list functions were very helpful to avoid some messy and unclean code. The list functions used were

```
lists:keyfind/3, lists:keyreplace/3, lists:keysort/2,
lists:usort/1, lists:map/2, lists:foldl/3.
```

If the document description is followed carefully step by step, the implemenation becomes less streeful.

4 Interfaces, History and Router

4.1 Interfaces

Another major step in implemenation of link-state routing protocol is the maintaining a consistent view of the network. To main this consistent view a router needs to keep track of its interfaces and update it when nodes are added or removed from the network.

To implement the interface for router, function names were given to implement along with description. The functions were

```
new/0, add/4, remove/2, lookup/2, ref/2, name/2, list/1, broadcast/2.
```

The implemenation of these functions was straight forward and some list functions named

```
lists:keyfind/3, lists:keydelete/3, lists:map/2.
```

were used to ease the implemenation logic.

4.2 The History

Along with interfaces router also needs to keep track of what link-state messages it has received over the period of time to avoid cyclic paths and also ensure that the old messages do not interfare to change the view of the network. The logic is that the router has a history data structure implemented which keeps the track of message using a number value and when it receives a link-state message it compares its number with the one saved in history and only forwards the message if the number with message is newer/greater then one in history. The data structure looks like [{Name, CounterValue}]. To ignore the messages from the router itself, an entry with value inf is added when initiating the router. To implement the history for router two functions

```
new/1, update/3.
```

were implemented. The first function creates a history entry for router with initial value set to -1. The second function updates the counter value whenever it receives a new number greater then one stored in the history before.

4.3 The Router

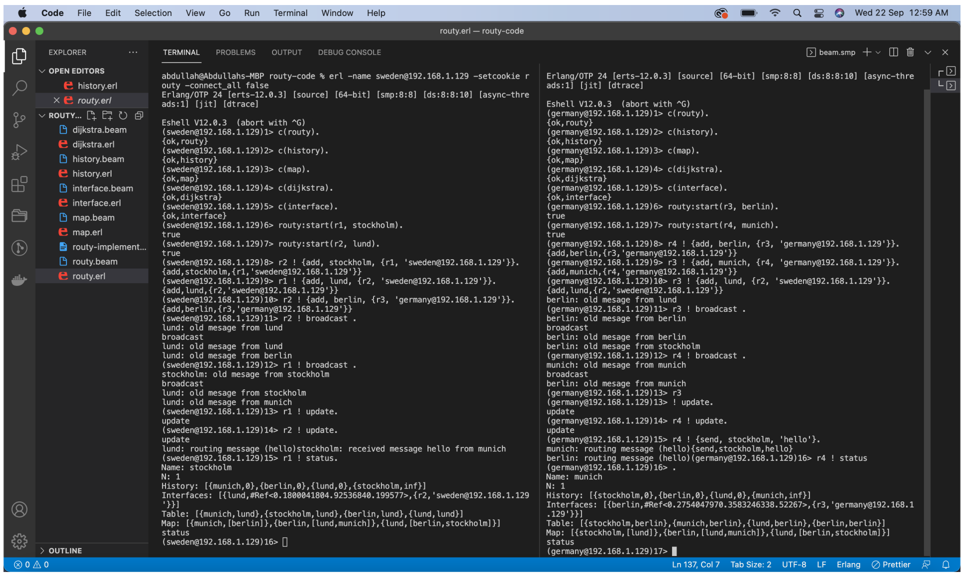
The router is the part where a routing process will not only just route messaages through a network of connected nodes but also, maintain a view of network, construct routing tables. The code for the router was already provided so it was the easy part in the assignment.

5 Testing

I did two types of testing for the routy.

5.1 Single Machine Testing

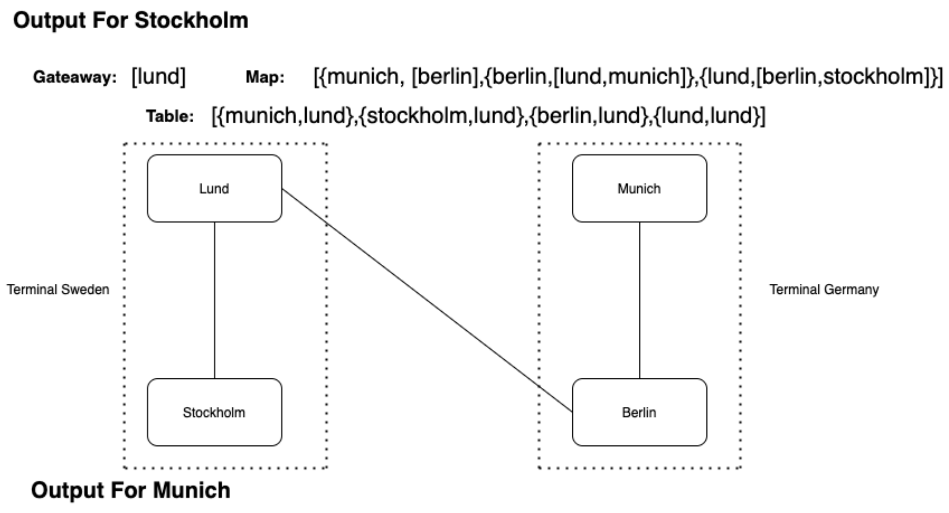
One was simple testing on my own machine mentioning different regions on different shells and sending messaages between different routers to build a simple network of routers I had one terminal for erlang shell named Sweden and another terminal for erlang shell named Germany. The router processes were named with cities of stockholm, lund, berlin and munich. The results from my testing are given in the screeshot from my terminals and the figure diagram



```
abdullah@Abdullahs-MBP routy-code % erl -name sweden@192.168.1.129 -setcookie r
outy -connect_all false
Erlang/OTP 24 [erts-12.0.3] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-thre
ads:1] [jit] [dtrace]

Eshell V12.0.3  (abort with ^G)
(sweden@192.168.1.129)1> c(routy).
{ok,routy}
(sweden@192.168.1.129)2> c(history).
{ok,history}
(sweden@192.168.1.129)3> c(map).
{ok,map}
(sweden@192.168.1.129)4> c(dijkstra).
{ok,dijkstra}
(sweden@192.168.1.129)5> c(interface).
{ok,interface}
(sweden@192.168.1.129)6> routy:start(r1, stockholm).
true
(sweden@192.168.1.129)7> routy:start(r2, lund).
true
(sweden@192.168.1.129)8> r2 ! {add, stockholm, {r1, 'sweden@192.168.1.129'}}.
{add,stockholm,{r1,'sweden@192.168.1.129'}}
(sweden@192.168.1.129)9> r1 ! {add, lund, {r2, 'sweden@192.168.1.129'}}.
{add,lund,{r2,'sweden@192.168.1.129'}}
(sweden@192.168.1.129)10> r2 ! {add, berlin, {r3, 'germany@192.168.1.129'}}.
{add,berlin,{r3,'germany@192.168.1.129'}}
(sweden@192.168.1.129)11> r2 ! broadcast .
lund: old mesage from lund
broadcast
lund: old mesage from lund
lund: old mesage from berlin
(sweden@192.168.1.129)12> r1 ! broadcast .
stockholm: old mesage from stockholm
broadcast
lund: old mesage from stockholm
lund: old mesage from munich
(sweden@192.168.1.129)13> r1 ! update.
update
(sweden@192.168.1.129)14> r2 ! update.
update
lund: routing message (hello)stockholm: received message hello from munich
(sweden@192.168.1.129)15> r1 ! status.
Name: stockholm
N: 1
History: [{munich,0},{berlin,0},{lund,0},{stockholm,inf}]
Interfaces: [{lund,#Ref<0.1800041804.92536840.199577>,{r2,'sweden@192.168.1.129
'}}]
Table: [{munich,lund},{stockholm,lund},{berlin,lund},{lund,lund}]
Map: [{munich,[berlin]},{berlin,[lund,munich]},{lund,[berlin,stockholm]}]
status
(sweden@192.168.1.129)16> 
```

```
Erlang/OTP 24 [erts-12.0.3] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-thre
ads:1] [jit] [dtrace]

Eshell V12.0.3  (abort with ^G)
(germany@192.168.1.129)1> c(routy).
{ok,routy}
(germany@192.168.1.129)2> c(history).
{ok,history}
(germany@192.168.1.129)3> c(map).
{ok,map}
(germany@192.168.1.129)4> c(dijkstra).
{ok,dijkstra}
(germany@192.168.1.129)5> c(interface).
{ok,interface}
(germany@192.168.1.129)6> routy:start(r3, berlin).
true
(germany@192.168.1.129)7> routy:start(r4, munich).
true
(germany@192.168.1.129)8> r4 ! {add, berlin, {r3, 'germany@192.168.1.129'}}.
{add,berlin,{r3,'germany@192.168.1.129'}}
(germany@192.168.1.129)9> r3 ! {add, munich, {r4, 'germany@192.168.1.129'}}.
{add,munich,{r4,'germany@192.168.1.129'}}
(germany@192.168.1.129)10> r3 ! {add, lund, {r2, 'sweden@192.168.1.129'}}.
{add,lund,{r2,'sweden@192.168.1.129'}}
berlin: old mesage from lund
(germany@192.168.1.129)11> r3 ! broadcast .
berlin: old mesage from berlin
broadcast
berlin: old mesage from berlin
berlin: old mesage from stockholm
(germany@192.168.1.129)12> r4 ! broadcast .
munich: old mesage from munich
broadcast
berlin: old mesage from munich
(germany@192.168.1.129)13> r3
(germany@192.168.1.129)13> ! update.
update
(germany@192.168.1.129)14> r4 ! update.
update
(germany@192.168.1.129)15> r4 ! {send, stockholm, 'hello'}.
munich: routing message (hello){send,stockholm,hello}
berlin: routing message (hello)(germany@192.168.1.129)16> r4 ! status
(germany@192.168.1.129)16> .
Name: munich
N: 1
History: [{stockholm,0},{berlin,0},{lund,0},{munich,inf}]
Interfaces: [{berlin,#Ref<0.2754047970.3583246338.52267>,{r3,'germany@192.168.1
.129'}}]
Table: [{stockholm,berlin},{munich,berlin},{lund,berlin},{berlin,berlin}]
Map: [{stockholm,[lund]},{berlin,[lund,munich]},{lund,[berlin,stockholm]}]
status
(germany@192.168.1.129)17> 
```

Figure 1: Terminals Sweden and Germany



Output For Stockholm

Gateaway: [lund]   **Map:** [{munich, [berlin]},{berlin,[lund,munich]},{lund,[berlin,stockholm]}]

Table: [{munich,lund},{stockholm,lund},{berlin,lund},{lund,lund}]

Output For Munich

Gateaway: [berlin]   **Map:** [{stockholm,[lund]}, {berlin,[lund,munich]}, {lund,[berlin,stockholm]}]

Table: [{stockholm,berlin},{munich,berlin},{lund,berlin},{berlin,berlin}]

Figure 2: Figure to Explain how routes are connected

5.2 Multi Machines Testing - The World

Another type of testing was done with my classmates for optional task named The World. So we used different regions of the world for our experiement.

Brazil from SouthAmerica
Turkey From Europe (My Machine),
India from Asia,
Australia from Australia,
United States from North America

Our Network Map can be seen in the figure.

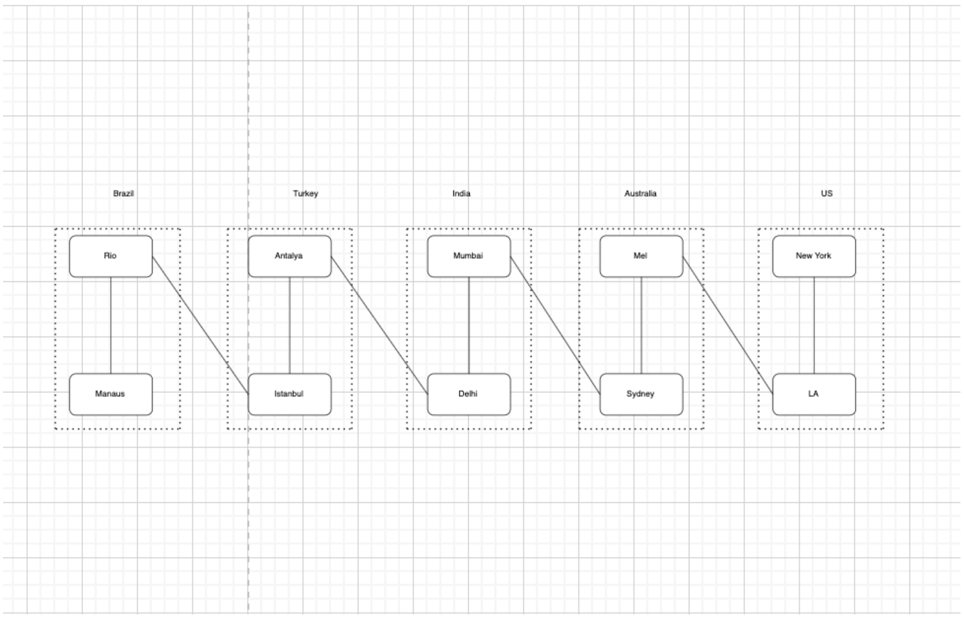


Figure 3: Terminals Sweden and Germany

From the figure it can be seen that we connected different cities from the regions. For all of our city nodes the gateways were in the following manner

```
Nodes -> Gateways

Manuas -> Rio
Rio -> Manuas, Istanbul
Istanbul -> Antalya, Rio
Antalya -> Istanbul, Delhi
Delhi -> Mumbai, Antalya
Mumbai -> Delhi, Sydney
```

```
Sydney -> Melbourne, Mumbai
Melbourne -> Sydney, LA
LA -> Newyork, Melbourne
Newyork -> LA
```

We ran different tests and sent messages to see the sending of message, routing of message and receiving of message in action.

5.2.1 First Test - The Long Message

The first test was the Long Route message "hello from manuas" which was from

```
Manuas -> Newyork
```

So in order to reach this message the following route was created

```
Manuas -> Rio -> Istanbul -> Antalya -> Delhi -> Mumbai -> Sydney
-> Melbourne -> LA -> Newyork
```

As on my machine the Turkey region was enabled and had two router processes named Istanbul and Antalya so I received the message at Istanbul from Rio and Istanbul forwarded it to Antalya and Antalya forwarded message to Delhi.

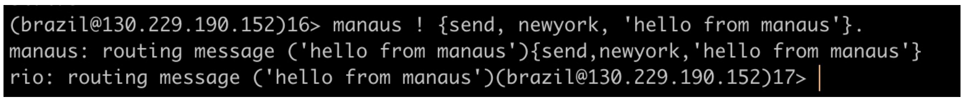
5.2.2 Second Test - Mumbai Outage

Second test we performed was to DOWN the link or router of Mumbai and I instantly received message that exit is received for mumbai. But after that we sent a message from

```
Rio -> Istanbul
```

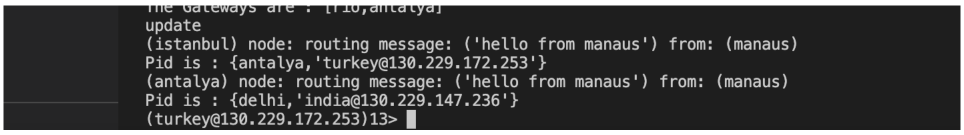
and the message was received at Istanbul which shows that the routing works for Europe and South America and messaages can be received but messages will not be sent across Australia and USA as India Link is down.

The output of both tests is shown in screenshots attached.



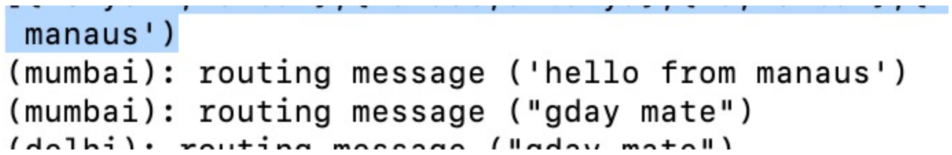
```
(brazil@130.229.190.152)16> manaus ! {send, newyork, 'hello from manaus'}.
manaus: routing message ('hello from manaus'){send,newyork,'hello from manaus'}
rio: routing message ('hello from manaus')(brazil@130.229.190.152)17> 
```

Figure 4: Routing Long Message - The World - 1



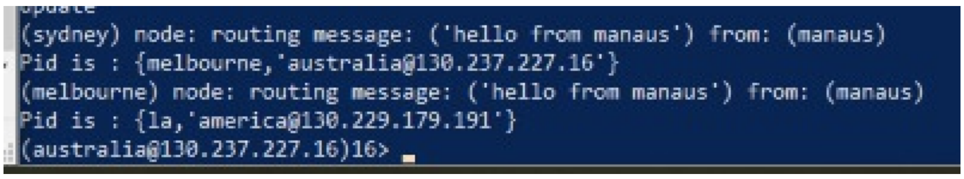
```
The Gateways are : [rio,antalya]
update
(istanbul) node: routing message: ('hello from manaus') from: (manaus)
Pid is : {antalya,'turkey@130.229.172.253'}
(antalya) node: routing message: ('hello from manaus') from: (manaus)
Pid is : {delhi,'india@130.229.147.236'}
(turkey@130.229.172.253)13>
```

Figure 5: Routing Long Message - The World - 2



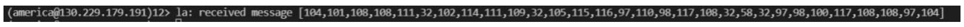
```
manaus')
(mumbai): routing message ('hello from manaus')
(mumbai): routing message ("gday mate")
(delhi): routing message ("gday mate")
```

Figure 6: Routing Long Message - The World - 3



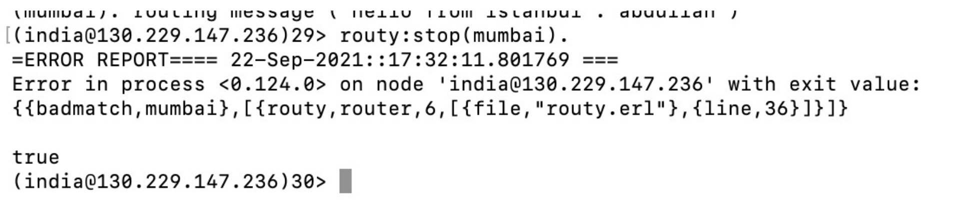
```
update
(sydney) node: routing message: ('hello from manaus') from: (manaus)
Pid is : {melbourne,'australia@130.237.227.16'}
(melbourne) node: routing message: ('hello from manaus') from: (manaus)
Pid is : {la,'america@130.229.179.191'}
(australia@130.237.227.16)16>
```

Figure 7: Routing Long Message - The World - 4



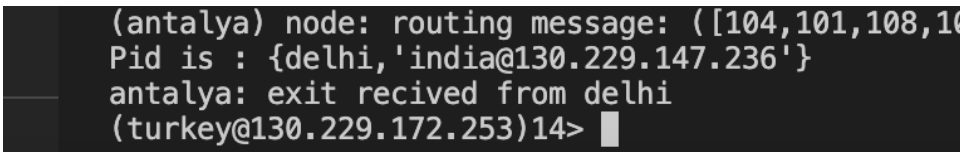
```
(america@130.229.179.191)12> la: received message [104,101,108,108,111,32,102,114,111,109,32,105,115,116,97,110,98,117,108,32,58,32,97,98,100,117,108,108,97,104]
```

Figure 8: Routing Long Message - The World - 5



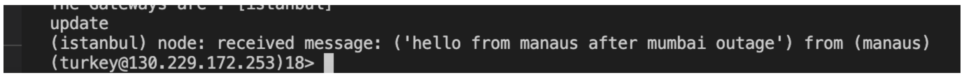
```
(mumbai): routing message ('hello from istanbul : abdullah')
(india@130.229.147.236)29> routy:stop(mumbai).
=ERROR REPORT==== 22-Sep-2021::17:32:11.801769 ===
Error in process <0.124.0> on node 'india@130.229.147.236' with exit value:
{{badmatch,mumbai},[{routy,router,6,[{file,"routy.erl"},{line,36}]}]}

true
(india@130.229.147.236)30>
```

Figure 9: Mumbai Outage - 1



```
(antalya) node: routing message: ([104,101,108,10
Pid is : {delhi,'india@130.229.147.236'}
antalya: exit recived from delhi
(turkey@130.229.172.253)14>
```

Figure 10: Mumbai Outage - 2



```
The Gateways are : [istanbul]
update
(istanbul) node: received message: ('hello from manaus after mumbai outage') from (manaus)
(turkey@130.229.172.253)18>
```

Figure 11: Mumbai Outage - 3

6 Conclusions

The seminar introduced me to understand the structure and algorithm of link-state routing protocol. I learned about the implemenation of dijkstra algorithm and the use of some good list functions.

Routy implemenation was good practice to get the idea about the complexity level of how a routing table is updated in large networks and making sure that all the nodes in a network map have consistent connectivity and information.