Routy: a small routing protocol

Vasigaran Senthilkumar

September 22, 2021

1 Introduction

In this report, the development and analysis of a basic router software using link state routing protocols is discussed. The link state protocol that has been used here is OSPF. The exercise provides an understanding on how link state routing protocols works and how distributed algorithms are written in a functional programming language like Erlang. These topics hold the key to get started with fundamentals of distributed programming.

2 Main problems and solutions

2.1 Full walkthrough and Errors in execution:

The walkthrough of which function is called after each response in the routy module had to be closely understood. Since the code was developed in parts and put together for implementation, the flow of events was not understood for several hours. Also there were lot of errors in execution and sometimes the lists were empty. So for debugging lot of io format to print the value was given at each stage. The Hist module specifically was implemented using list of tuples was returning empty lists due to wrong implementation of List comprehension.

2.2 Difference between Sorted list, routing table, Map:

The difference between the three lists that are used in the map and Dijkstra module was confusing and how each of them are related. The sorted list contains list of tuples with a Node, distance to the node and gateway . The list is always sorted based on the length and it is used for creating a routing table. When sorted list is empty, it means we have added all the required paths to the routing table.The routing table contains information on which gateway must be chosen to send messages to a destination node.The Map consists of node and their links nodes which are directly connected or one hop away.

2.3 List Module:

The functions in the list module was widely used throughout the program. The Erlang documentation had the syntax for the functions. The interesting functions that almost have similar list manipulation operations and have been used here are foldl/3,flatmap/2,foreach/2. foldl/3 allows to call a certain function for every element in a provided list starting with accumulator input and providing final value of accumulator. Flatmap takes a list and calls a function for each element in list and then the results are appended to the output list. Foreach executes a function for each event in the list while also maintaining the order of list input.

```
lists:foldl(fun(X,Y)-> update(X,Length+1,Gateway,Y) end, Tail, Links)
```

```
lists:flatmap((fun({X,Y})-> [X|Y] end),Map)
```

```
lists:foreach(fun({_,_,Pid}) ->
  Pid ! Message end,
  Intf).
```

3 Evaluation

Various test scenarios where experimented to identify if the link state routing protocol is working as expected and messages could be sent from one router to another.

3.1 Test 1:

For the first test, we consider two nodes (sweden and germany) on two shells. sweden has two routers, stockholm and lund. germany has two routers, berlin and frankfurt. stockholm is directly connected to lund. lund is directly connected to berlin. berlin is directly connected to frankfurt.

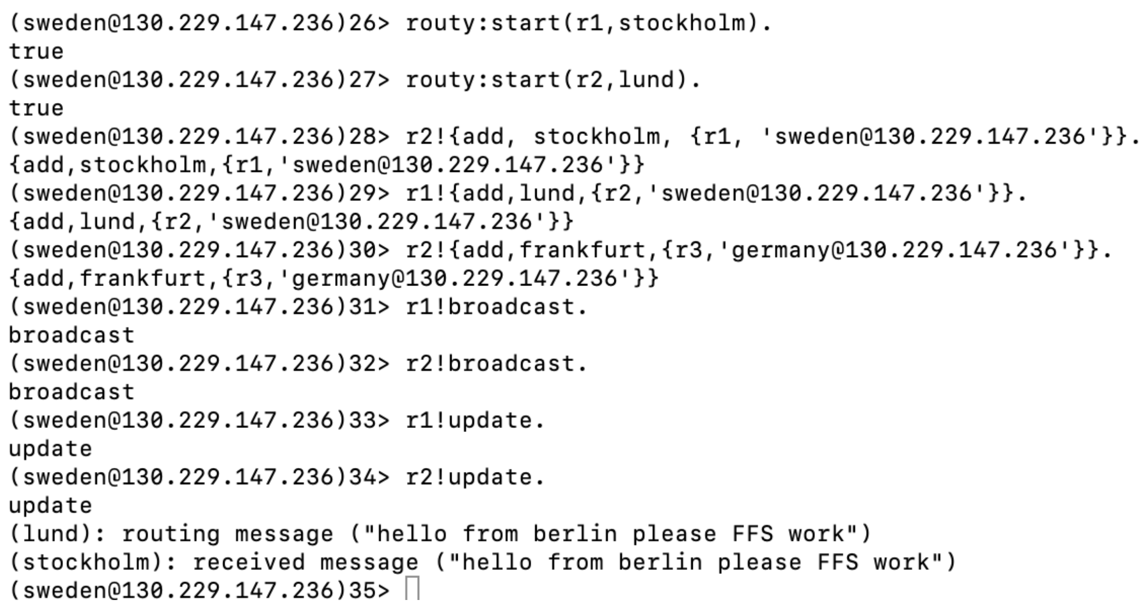
Now a message is sent from berlin to stockholm. So the message travels from berlin to frankfurt, then frankfurt to lund and then reaches stockholm. Below are the screenshots of sending the message and a visual representation of the connections.

lund Map: [{berlin,[frankfurt]},{frankfurt,[berlin,lund]},{stockholm,[lund]}]
lund Routing Table[{berlin,frankfurt},{lund,frankfurt},{stockholm,stockholm},{frankfurt,frankfurt}]

frankfurt Map: [{berlin,[frankfurt]},{lund,[stockholm,frankfurt]},{stockholm,[lund]}]}
frankfurt Routing Table: [frankfurt,berlin,stockholm,lund,lund,lund,berlin,berlin}]

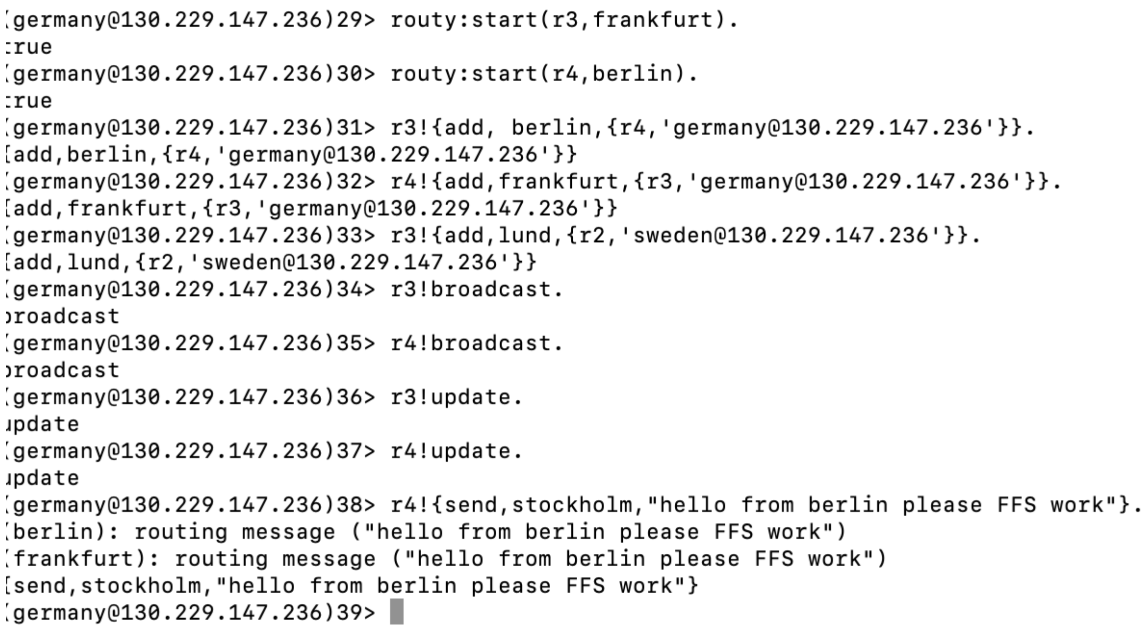
berlin Map: [{frankfurt,[berlin,lund]},{lund,[stockholm,frankfurt]},{stockholm,[lund]}]}
berlin Routing Table: [{stockholm,frankfurt},{berlin,frankfurt},{lund,frankfurt},{frankfurt,frankfurt}]

stockholm Map : [{berlin,[frankfurt]},{frankfurt,[berlin,lund]},{lund,[stockholm,frankfurt]}]
stockholm Routing Table: [{berlin,lund},{stockholm,lund},{frankfurt,lund},{lund,lund}]



```
(sweden@130.229.147.236)26> routy:start(r1,stockholm).
true
(sweden@130.229.147.236)27> routy:start(r2,lund).
true
(sweden@130.229.147.236)28> r2!{add, stockholm, {r1, 'sweden@130.229.147.236'}}.
{add,stockholm,{r1,'sweden@130.229.147.236'}}
(sweden@130.229.147.236)29> r1!{add,lund,{r2,'sweden@130.229.147.236'}}.
{add,lund,{r2,'sweden@130.229.147.236'}}
(sweden@130.229.147.236)30> r2!{add,frankfurt,{r3,'germany@130.229.147.236'}}.
{add,frankfurt,{r3,'germany@130.229.147.236'}}
(sweden@130.229.147.236)31> r1!broadcast.
broadcast
(sweden@130.229.147.236)32> r2!broadcast.
broadcast
(sweden@130.229.147.236)33> r1!update.
update
(sweden@130.229.147.236)34> r2!update.
update
(lund): routing message ("hello from berlin please FFS work")
(stockholm): received message ("hello from berlin please FFS work")
(sweden@130.229.147.236)35> 
```

Figure 1: Shell 1



```
(germany@130.229.147.236)29> routy:start(r3,frankfurt).
rue
(germany@130.229.147.236)30> routy:start(r4,berlin).
rue
(germany@130.229.147.236)31> r3!{add, berlin,{r4,'germany@130.229.147.236'}}.
{add,berlin,{r4,'germany@130.229.147.236'}}
(germany@130.229.147.236)32> r4!{add,frankfurt,{r3,'germany@130.229.147.236'}}.
{add,frankfurt,{r3,'germany@130.229.147.236'}}
(germany@130.229.147.236)33> r3!{add,lund,{r2,'sweden@130.229.147.236'}}.
{add,lund,{r2,'sweden@130.229.147.236'}}
(germany@130.229.147.236)34> r3!broadcast.
roadcast
(germany@130.229.147.236)35> r4!broadcast.
roadcast
(germany@130.229.147.236)36> r3!update.
pdate
(germany@130.229.147.236)37> r4!update.
pdate
(germany@130.229.147.236)38> r4!{send,stockholm,"hello from berlin please FFS work"}.
(berlin): routing message ("hello from berlin please FFS work")
(frankfurt): routing message ("hello from berlin please FFS work")
{send,stockholm,"hello from berlin please FFS work"}
(germany@130.229.147.236)39> 
```

Figure 2: Shell 2

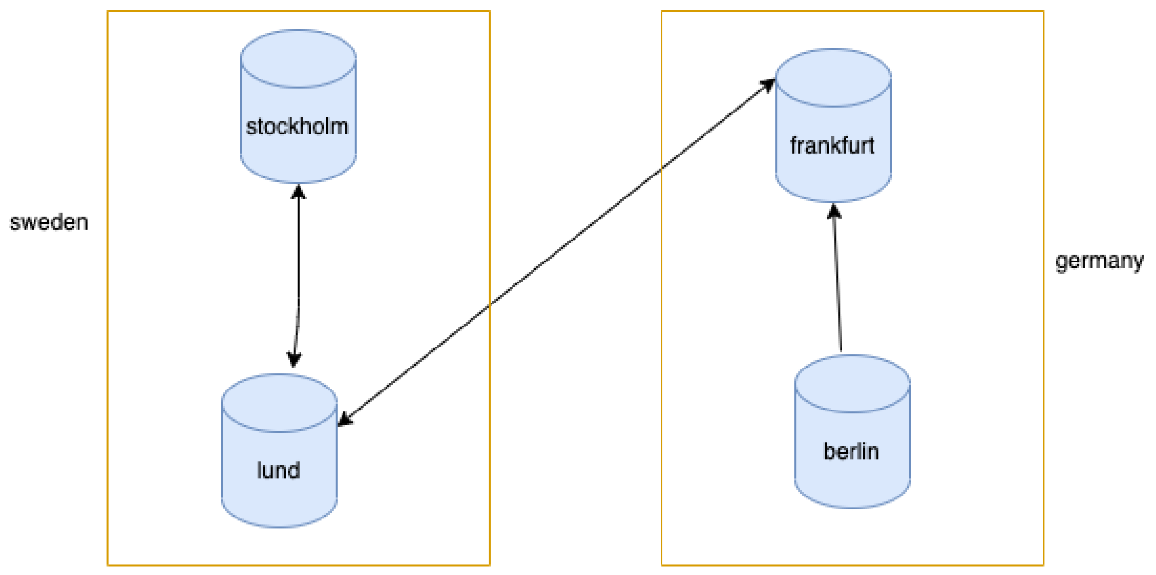


Figure 3: Test with 4 routers

3.2 Test 2

The optional task to test the protocol on different machines was performed with my classmates. Each machine was representing different regions of the world. There are 5 nodes which renamed to 5 countries. Each country has two routers of cities in the country. I started the India node with routers at delhi and mumbai. Similiarly other countries Turkey, US, Brazil, austalia was also started. After the routers were started, between two routers in every node, connection was made. Then connections were made between different nodes to form a routing network. My router at delhi was connected to antalya in Turkey and router at mumbai was connected to router at sydney. Then a message was sent from router at Manaus to the router at Newyork. Both mumbai and delhi was involved in routing the messages to Newyork. When the Node at India was made to go down with routy:stop(Node), both delhi and mumbai routers exited. But the rest of the routing network was split into two without a gateway between. And the delivery of a 'DOWN' message to the other nodes is almost instantaneous. Below is the routing tables for delhi and mumbai routers and the entire routing diagram of network tested.

mumbai Map: [{newyork,[la]},{manaus,[rio]},{rio,[istanbul,manaus]}, {la,[newyork,melbourne]},{antalya,[delhi,istanbul]},{delhi,[mumbai,antalya]}, {melbourne,[la,sydney]},{istanbul,[rio,antalya]},{sydney,[mumbai,melbourne]}]

Routing table: [{manaus,delhi},{rio,delhi},{newyork,sydney},{la,sydney}, {istanbul,delhi},{mumbai,delhi},{antalya,delhi},{melbourne,sydney}, {sydney,sydney},{delhi,delhi}]

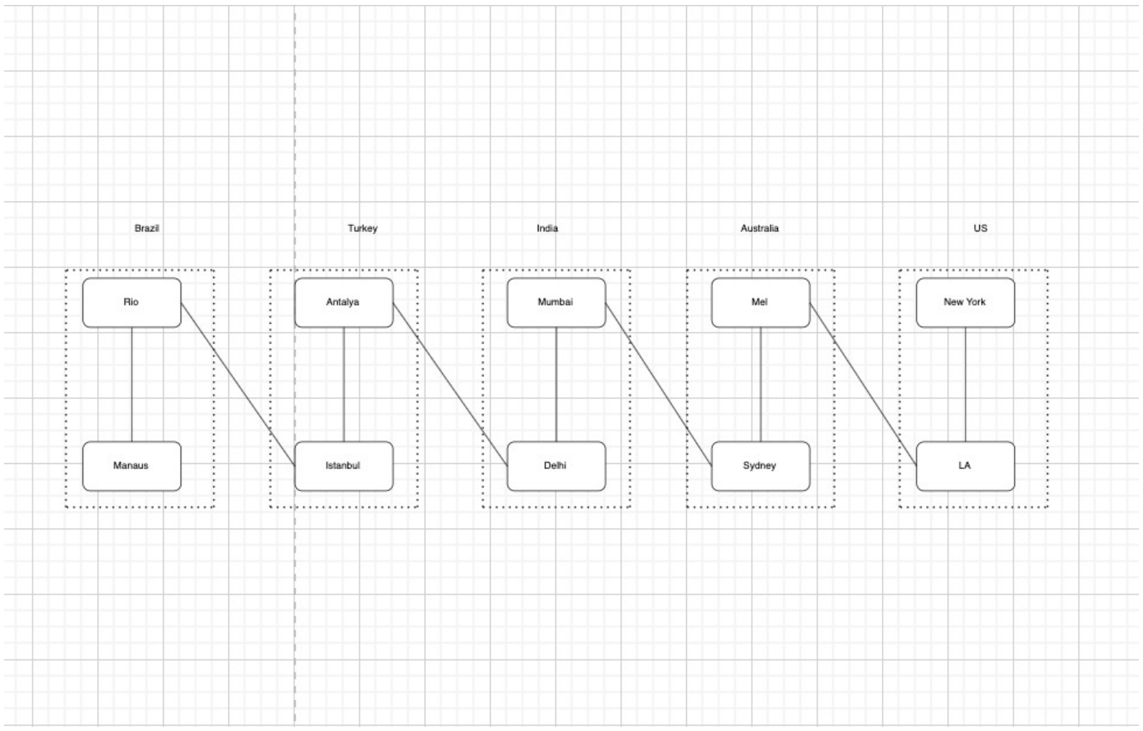
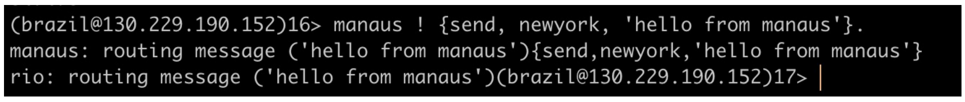


Figure 4: Multiple Machines testing



```
(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> 
```

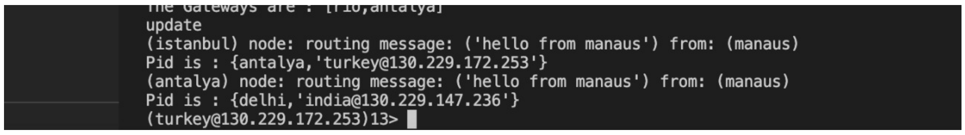
Figure 5: Send Message initiated at Manaus

delhi Map: [{newyork,[la]},{manaus,[rio]},{rio,[istanbul,manaus]}, {la,[newyork,melbourne]},{antalya,[delhi,istanbul]},{melbourne,[la,sydney]}, {istanbul,[rio,antalya]},{sydney,[mumbai,melbourne]},{mumbai,[delhi,sydney]}]

delhi Routing table: [{newyork,mumbai},{manaus,antalya},{la,mumbai}, {melbourne,mumbai},{rio,antalya},{delhi,antalya},{istanbul,antalya}, {sydney,mumbai},{mumbai,mumbai},{antalya,antalya}]

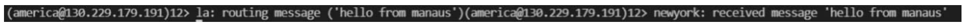
3.2.1 Setup commands for Test 2

```
erl -name india@130.229.147.236 -setcookie routy -connect_all false
routy:start(delhi,delhi).
routy:start(mumbai,mumbai).
mumbai!{add,delhi,{delhi,'india@130.229.147.236'}}.
delhi!{add,mumbai,{mumbai,'india@130.229.147.236'}}.
mumbai!{add,sydney,{sydney,'australia@130.237.227.16'}}.
```



```
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 6: Message routing between intermediate nodes



```
(america@130.229.179.191)12> la: routing message ('hello from manaus')(america@130.229.179.191)12> newyork: received message 'hello from manaus'
```

Figure 7: Message received at destination router newyork

```
delhi!{add,antalya,{antalya,'turkey@130.229.172.253'}}.
 mumbai!broadcast. delhi!broadcast. mumbai!update.  delhi!update.
```

4 Conclusions

The exercise was very difficult but the learning experience and bringing the code to work perfectly was a satisfactory feeling. It made me explore the documentation pages of Erlang and also made me understand the implementation of Dijkstra algorithm in depth. The activity to send messages from routers at one machine to another along with classmates in the same network was interesting.