Report 2: Routy - Link-State Routing

Md Sakibul Islam

September 17, 2025

1 Introduction

This report covers the implementation of Routy, a link-state routing protocol developed in Erlang programming. The homework involved creating a functional routing system where nodes maintain network topology maps, exchange link-state messages, compute optimal paths using Dijkstra's algorithm, and efficiently route messages through the network. The main part of this homework covered a deep understanding of how communication is handled in a network that contains several nodes, which is located in a huge region and where the nodes do not have direct access to every other node in the network. Actually, the task demonstrates how independent processes can cooperate to build a consistent global view of a network and ensure reliable message delivery, which is the basis of real routing protocols like Open Shortest Path First.

2 Main problems and solutions

While developing and testing my modules I encountered several issues:

- **Issue 1: Cross-node add**
  First I used bare atoms like `r1`, `r2` in {`add, Node, Pid`}, which caused immediate `'DOWN'` messages. Then I figured out that I need to use the correct remote name tuple:

```
{r1,'sweden@MdSakibuls-MacBook-Pro-M4'} !
    {add, lund, {r2,'denmark@MdSakibuls-MacBook-Pro-M4'}}.
```

- **Issue 2: Order of operations**
  When `broadcast`/`update` operations were performed before both routers were fully connected and running, the routing tables became inconsistent. Sweden's routing table remained `[]` while Denmark's table was properly populated `[lund,stockholm,stockholm,stockholm]`.

So after checking all the code bases and spending lots of time with clearing the terminal testing different ways, I found the actual reason. Then I solved it like: First, ensure both routers are fully started. Establish bidirectional connections using proper remote references. Allow brief synchronization time for process monitoring to stabilize. Then execute `broadcast` operations to exchange link-state messages. Finally, `update` routing tables after confirming complete network visibility.

After implementing this sequence, both routers maintained consistent routing tables and successfully routed messages bidirectionally, as demonstrated by the final successful message exchanged.

- **Issue 3: Message printing**
  Messages appeared as lists of ASCII codes. But when I changed the format specifier in `routy.erl` to `~s` for the message part it seems good to me.

```
io:format("~p: received message ~s from ~p~n",
[Name, Message, From]).
```

3 Evaluation

After fixing those issues, I tested two routers:

- Sweden (`r1`, stockholm)
- Denmark (`r2`, lund)

Message 1: Send Stockholm → Lund

Command:

```
{r1,'sweden@MdSakibuls-MacBook-Pro-M4'} ! {send, lund, "Hej Lund Finally!!"}.
```

Results (logs):

```
stockholm: routing message Hej Lund Finally!! to lund
lund: received message Hej Lund Finally!! from stockholm
```

Message 2: Send Lund → Stockholm

Command:

```
{r2,'denmark@MdSakibuls-MacBook-Pro-M4'} ! {send, stockholm, "Hej Stockholm Finally
```

Results (logs):

```
lund: routing message Hej Stockholm Finally!! to stockholm
stockholm: received message Hej Stockholm Finally!! from lund
```

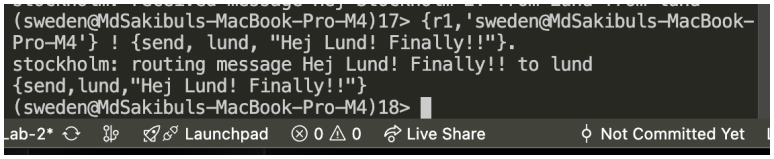

Figure 1: Sending from Sweden (Stockholm) to Denmark (Lund)

Figure 2: Receiving in Denmark (Lund) from Sweden (Stockholm)

Topology

Figure 1 and Figure 2 illustrate the two-node network topology:

```
[stockholm] <> [lund]
```

4 Bonus Task - The World

For the bonus part, I extended the setup to a global network with multiple Erlang nodes:

- Europe: `paris`, `berlin`
- Asia: `tokyo`
- Africa: `cairo`

Routers were connected across nodes, and after running `broadcast/update` on all, messages successfully routed across continents. For example, a message from `Paris` → `Cairo` traveled via `Berlin`, and a message from `Cairo` → `Tokyo` followed the path `Cairo` → `Berlin` → `Paris` → `Tokyo`.

This experiment demonstrated how the link-state protocol scales beyond two nodes, builds a global topology, and routes messages reliably across multiple hops.

5 Conclusions

This assignment really helped me see how routing works beyond the theory. By building Routy in Erlang, I understood how processes talk to each other, share link information, and still deliver messages even when the network changes. It was challenging at first, but in the end it made routing protocols like OSPF feel much more concrete and useful.