HW2 : Routy - a small routing protocol

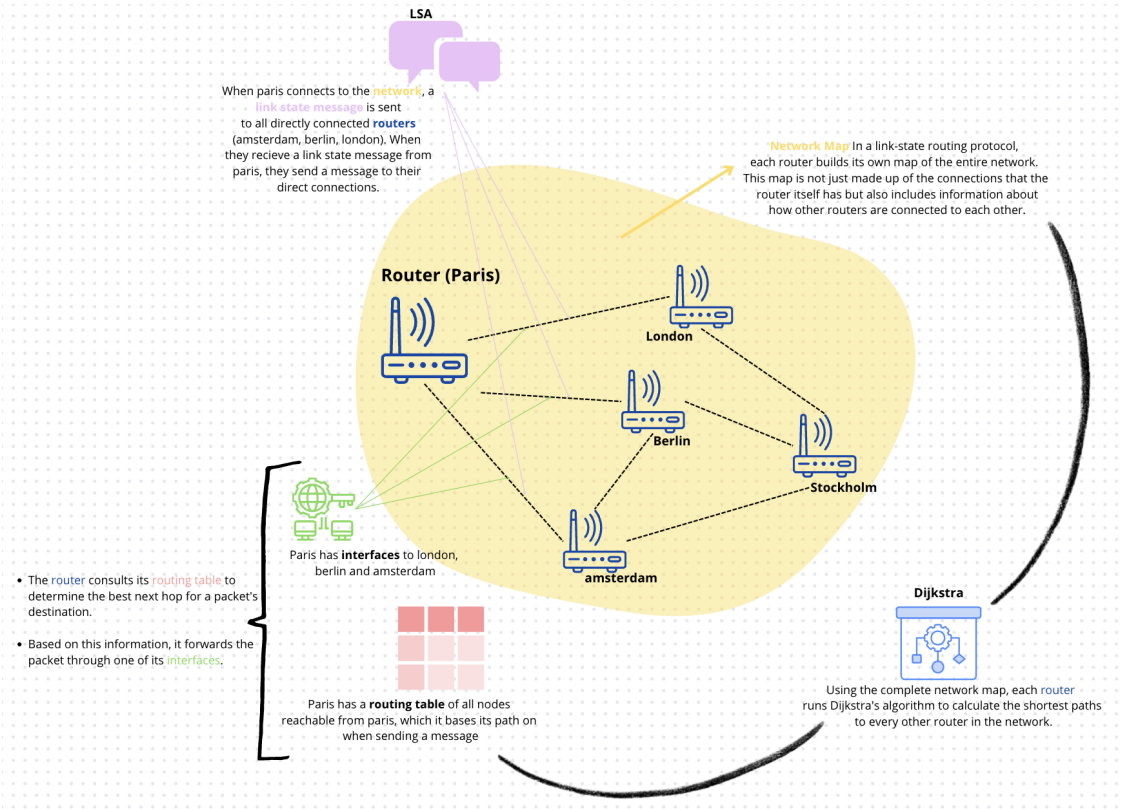
[redacted]

September 17, 2024

1 Introduction

This week we created a link-state routing protocol (Routy) using Erlang. The final goal of the program was to send messages between (simulted) routers which were not directly connected to each other, but via a network of “routers”. The program should calculate the best route to send a message via several connected routers on the network. The routing calculations were handled by a dijkstra algorithm which created routing tables for the connected routers, based on their map of the network, which was built by link state messages as routers were started and connected to each other.

The main learning outcome of this was to understand how link-state routing protocols work, and the benefits of having a link state network as they adapt to change, such as crashing nodes and network failures.



Basic visual of the routing protocol

2 Main problems and solutions

Map: The solution for the map was to create a module to manage the maps for the different nodes, and keep the maps updated for each change in the network of nodes. To manage the update, reachable and all_nodes functions in the map module, I used lists functions such as keydelete, keyfind and usort.

Dijkstra: The dijkstra algorithm used the map for each node to create a routing table, which instructed the node which pathway it should use to send messages to other nodes on the network. To implement this functionality I started by sorting all nodes in the map in a list, either with the path length 0 (for gateways) or inf (∞). This list of nodes with gateways 0 or ∞ will then be iterated through, and the function will update the shortest path and gateway for each node directly reachable from the current node. It iterates over these reachable nodes, extending the path length by one for each, and updates their path in the sorted list if the new path is shorter.

```
   UpdatedSorted = lists:foldl(fun(N, Sorted) -> update(N, Length + 1, Gateway,
Sorted) end, Rest, Reachables),
```

Error : I encountered issues with my dijkstra module at one point. I realized that when I was trying to send a message from ex stockholm -> mora, the correct interface for stockholm was not found. Stockholm had a direct connection to lund, lund had a direct connection to uppsala and uppsala had a direct connection to mora. However, when calling the function interface:lookup/2 with (Gateway, [list of stockholms interfaces]) i realized that the Gateway being inputted, was not lund (the interface), but uppsala. After some debugging, I managed to fix this issue, which had to do with the route function, which was not returning the most suitable gateway.

History: The history uses a counter for each router to track the latest message seen from that router. When a new message arrives, the router checks if the message has a higher counter than the last one it saw from the same source. If it's new (higher counter), the router processes and forwards it; otherwise, it ignores the message to prevent loops.

Routy: Routy would be the manager of the routers in this case. It handles initializing routers with identifiers and locations, and manages connections between these routers. The main functions include starting and stopping routers, updating routing tables based on received link-state information, and routing messages between routers either directly or through multiple hops. (This is where I encountered the error with my route function in dijkstra).

3 Conclusions

I had some confusion when it came to the update function in routy, because why would we not call the dijkstra algorithm to update our routing tables each time some change was made (add/remove). However, I tried to implement this update at each change made to the network map, and it worked fine as well.

To summarize, I feel that this has given me much better understanding of erlang programming in general and although its challenging to understand all aspects of the program, I feel like i understand the main logic and advantages of routing protocols of distributed programming.