Report 2: Routy, a small web server

Nils Blomgren
September 20, 2023

1. Introduction

This week's homework covered a link-state routing protocol and more specifically sending messages over the internet. To represent the network a map is used, the map is a list of [{ Nodes ,[Links]}, T]. In order to find where to send a message to reach its destination, we had to implement the dijkstra algorithm, an algorithm where the goal is to find the shortest path to a node. This is done by visiting all nodes and keeping track of which nodes are the fastest way to a specific destination node. The dijkstra algorithm is being executed on our routing tables that consist of [{ Node, Gateways }, T]. Where Gateway is either a link to the destination or a link to another node. The program also has functionality to keep track of message history, so that if a lot of messages are sent during the same time the program has a way to identify old messages.

The heart of the program is the routy module, from there the user can start the program and a process. More importantly we have the router (router/6) function which is running the core functions of the program. That is; adding a node as a gateway to another node, removing a node, broadcasting and updating a nodes table, sending messages and routing messages to other nodes in the network.

2. Main problems encountered

During the implementation of the code I had a few problems occur:

I found the dijkstra algorithm tricky and time consuming to create. It was my first time building a dijkstra algorithm on my own but I have worked with them before. But I was able to make it work after a few hours of struggling.

I also thought it was a bit tricky to understand how to run the program. The actual commands to start running the program took a while to grasp, but made a lot of sense when I figured it out.

Another issue that occurred for me when testing out the program during the bonus task was the fact that there aren't any automated updates and broadcasts. This isn't a big deal in this phase but a future improvement to my code would have been to automate this process.

Below is my test and validation of the bonus assignment:
In this case we have Sweden on my computer and Germany on Leonards computer.
Sweden has two nodes in its network, which are linked both ways to each other, these are;

Sthlm and Gbg.
Germany has also two nodes linked both ways;
Muc and Tolz.
Then the two countries are connected to each other by doing the following;
Muc adds links to sthlm.
Sthlm adds links to Muc.

We were then able to send messages from Gbg to Muc but also from Tolz to Gbg.

And when a node was disconnected the other country was notified, but still had to broadcast and update its nodes.

```
(sweden@130.229.142.200)4> routy:start(sthlm, sthlm).
true
(sweden@130.229.142.200)5> sthlm ! {add, gbg, {gbg, 'sweden@130.229.142.200'}}.
sthlm: gbg added
{add,gbg,{gbg,'sweden@130.229.142.200'}}
(sweden@130.229.142.200)6> gbg ! {add, sthlm, {sthlm, 'sweden@130.229.142.200'}}.
gbg: sthlm added
{add,sthlm,{sthlm,'sweden@130.229.142.200'}}
(sweden@130.229.142.200)7> sthlm ! {add, muc, {muc, 'germany@193.10.37.247'}}.
sthlm: muc added
{add,muc,{muc,'germany@193.10.37.247'}}
(sweden@130.229.142.200)8> sthlm ! broadcast.
sthlm: broadcast {links,sthlm,0,[muc,gbg]}
broadcast
(sweden@130.229.142.200)9> sthlm ! update.
sthlm: update []
update
(sweden@130.229.142.200)10> gbg ! update.
update
gbg: update [{gbg,sthlm},{muc,sthlm},{sthlm,sthlm}]
(sweden@130.229.142.200)11> gbg ! {send, muc, 'Hello Germany!!'}.
gbg: routing message to muc
{send,muc,'Hello Germany!!'}
GW: sthlm
sthlm: routing message to muc
GW to muc not found
(sweden@130.229.142.200)12> gbg ! update.
gbg: update [{tolz,sthlm},{gbg,sthlm},{muc,sthlm},{sthlm,sthlm}]
update
(sweden@130.229.142.200)13> gbg ! {send, muc, 'Hello Germany!'}.
gbg: routing message to muc
{send,muc,'Hello Germany!'}
GW: sthlm
sthlm: routing message to muc
GW to muc not found
(sweden@130.229.142.200)14> sthlm ! update.
sthlm: update [{tolz,muc},{sthlm,muc},{muc,muc}]
update
(sweden@130.229.142.200)15> gbg ! {send, muc, 'Hello Germany!'}.
gbg: routing message to muc
{send,muc,'Hello Germany!'}
GW: sthlm
sthlm: routing message to muc
GW: muc
sthlm: routing message to gbg
GW to gbg not found
(sweden@130.229.142.200)16> gbg ! broadcast.
gbg: broadcast {links,gbg,0,[sthlm]}
broadcast
(sweden@130.229.142.200)17> gbg ! update.
gbg: update [{tolz,sthlm},{gbg,sthlm},{muc,sthlm},{sthlm,sthlm}]
update
(sweden@130.229.142.200)18> sthlm ! update.
sthlm: update [{tolz,muc},{sthlm,muc},{gbg,gbg},{muc,muc}]
update
sthlm: routing message to gbg
GW: gbg
gbg: received message 'Hej, hur gar det till?' from tolz
(sweden@130.229.142.200)19>
```

As seen in the terminals above the trickiest part was remembering to broadcast and update the nodes.

3. Conclusion

The implementation of the code was harder on this assignment but it was also fun but more importantly a good lesson on how routing networks actually works. A good complement to the lectures. As mentioned earlier a lot of the code is handled manually, the user needs to add links, broadcast messages to all interfaces and update their own table. This could be automated so that the risks of faulty inputs becomes a smaller issue. One of the main lessons I take from this homework is the importance of having the correct information at each node. Always remembering to broadcast and update the routing tables. This has given me an understanding of how hard this must be in big networks.