Report 2: Routy - A Small Routing Protocol

Michail Roussos, michailr@kth.se

September 20, 2023

1 Introduction

In this exercise we created a link-state routing protocol in Erlang. Routers in our network can be connected to each other with directional (one-way) links and can only communicate with the routers they are directly connected to. To enable this we had to implement different modules, that will enable this functionality.

2 Main problems and solutions

We had to implement the following modules:

1. The first module we had to implement was the map module, called *map*. In this map we can represent the one-directional links between the nodes.

2. The Dijkstra routing algorithm, called *dijkstra*.

3. A module for Interfaces, called *intf*, which will be used to hold the name, process reference and process identifier for the nodes.

4. A module for the History, called *hist*, which will make sure that each node knows which is the last received message from other nodes to make sure that the nodes will keep track of the messages and we will not resend messages for no reason.

5. A module for the router, called *routy*, which will initiate the process and which will store the necessary info for each node and that will define the behaviour for the potential messages.

3 Evaluation

In order to check the functionality of the implementation I created the following module that sets up a network

test.erl

```erl
-module(test).
-export([start/1,broadcast/0, update/0]).

Used 0 times
start(Name) ->
    routy:start(r1, stockholm),
    routy:start(r2, uppsala),
    routy:start(r3, gothenburg),
    routy:start(r4, nybro),
    routy:start(r5, kristianstad),

    r1 ! {add, uppsala, {r2,Name}},
    r1 ! {add, gothenburg, {r3,Name}},
    r1 ! {add, nybro, {r4,Name}},
    r2 ! {add, kristianstad, {r5,Name}},
    r2 ! {add, gothenburg, {r3,Name}},
    r4 ! {add, stockholm, {r1,Name}},
    r5 ! {add, gothenburg, {r3,Name}},
    r5 ! {add, gothenburg, {r3,Name}},
    timer:sleep(4000),
    broadcast(),
    timer:sleep(4000),
    update(),
    routy:print_status(r1),
    routy:print_status(r2),
    routy:print_status(r3),
    ok.

Used 1 times
broadcast() ->
    r1 ! broadcast,
    r2 ! broadcast,
    r3 ! broadcast,
    r4 ! broadcast,
    r5 ! broadcast.

Used 1 times
update() ->
    r1 ! update,
    r2 ! update,
    r3 ! update,
    r4 ! update,
    r5 ! update.
```

For the optional part of the assignment, we connected two different computers one being Sweden and the other one was Norway. The module for Sweden was similar to the test one:

routy.erl | test.erl | optional.erl

optional.erl

```
1   -module(optional).
2   -export([start/0]).
3   % 192.168.220.33
4   % 192.168.220.207
5   start() ->
6       routy:start(stockholm, stockholm),
7       routy:start(uppsala, uppsala),
8       routy:start(gothenburg, gothenburg),
9       routy:start(kristianstad, kristianstad),
10
11      stockholm ! {add, uppsala, {uppsala,'sweden@192.168.220.33'}},
12      stockholm ! {add, gothenburg, {gothenburg,'sweden@192.168.220.33'}},
13      uppsala ! {add, gothenburg, {gothenburg,'sweden@192.168.220.33'}},
14      gothenburg ! {add, kristianstad, {kristianstad,'sweden@192.168.220.33'}},
15      kristianstad ! {add, stockholm, {stockholm,'sweden@192.168.220.33'}},
16      %stockholm ! {add, bergen, {bergen,'norway@192.168.220.207'}}.
17      % stockholm ! update.
18      % stockholm!{send,bergen,"hello"}.
19      % stockholm!{send,paris,"hello1"}.
20      timer:sleep(1000),
21      broadcast(),
22      timer:sleep(1000),
23      update(),
24      ok.
25
26      broadcast() ->
27          stockholm ! broadcast,
28          uppsala ! broadcast,
29          gothenburg ! broadcast,
30          kristianstad ! broadcast.
31
32      update() ->
33          stockholm ! update,
34          uppsala ! update,
35          gothenburg ! update,
36          kristianstad ! update.
```

The Erlang shell for Sweden was initiated with this command

```
erl -name sweden@192.168.220.33 -setcookie routy -connect_all false
```

and then the optional:start/0 function was executed.

```
History: [{stockholm,inf},{kristianstad,0},{gothenburg,0},{uppsala,0}]
ok
(sweden@192.168.220.33)9> stockholm!{send,bergen,"hello"}.
stockholm: routing message ([104,101,108,108,111]){send,bergen,"hello"}
stockholm: exit recived from bergen
(sweden@192.168.220.33)10> stockholm ! {add, bergen, {bergen,'norway@192.168.220.207'}}.
{add,bergen,{bergen,'norway@192.168.220.207'}}
(sweden@192.168.220.33)11> routy:print_status(stockholm).
Name: stockholm
Interface: [{uppsala,#Ref<0.1004719367.4097835010.186652>,
                     {uppsala,'sweden@192.168.220.33'}},
            {gothenburg,#Ref<0.1004719367.4097835010.186653>,
                        {gothenburg,'sweden@192.168.220.33'}},
            {bergen,#Ref<0.1004719367.4097835010.186810>,
                    {bergen,'norway@192.168.220.207'}}]
Map: [{kristianstad,[stockholm]},
      {gothenburg,[kristianstad]},
      {uppsala,[gothenburg]}]
Table: [{bergen,bergen},
        {gothenburg,gothenburg},
        {uppsala,uppsala},
        {stockholm,kristianstad},
        {kristianstad,gothenburg}]
History: [{stockholm,inf},{kristianstad,0},{gothenburg,0},{uppsala,0}]
ok
(sweden@192.168.220.33)12> stockholm!update.
update
(sweden@192.168.220.33)13> routy:print_status(stockholm).
Name: stockholm
Interface: [{uppsala,#Ref<0.1004719367.4097835010.186652>,
                     {uppsala,'sweden@192.168.220.33'}},
            {gothenburg,#Ref<0.1004719367.4097835010.186653>,
                        {gothenburg,'sweden@192.168.220.33'}},
            {bergen,#Ref<0.1004719367.4097835010.186810>,
                    {bergen,'norway@192.168.220.207'}}]
Map: [{kristianstad,[stockholm]},
      {gothenburg,[kristianstad]},
      {uppsala,[gothenburg]}]
Table: [{bergen,bergen},
        {gothenburg,gothenburg},
        {uppsala,uppsala},
        {stockholm,kristianstad},
        {kristianstad,gothenburg}]
History: [{stockholm,inf},{kristianstad,0},{gothenburg,0},{uppsala,0}]
ok
(sweden@192.168.220.33)14> stockholm!update.
update
(sweden@192.168.220.33)15> stockholm!{send,bergen,"hello"}.
stockholm: routing message ([104,101,108,108,111]){send,bergen,"hello"}
stockholm: received message (heyhey) from bergen
(sweden@192.168.220.33)16>
```

We sent a message from Stockholm and received a message from Bergen:

```
update
(sweden@192.168.220.33)15> stockholm!{send,bergen,"hello"}.
stockholm: routing message ([104,101,108,108,111]){send,bergen,"hello"}
stockholm: received message (heyhey) from bergen
(sweden@192.168.220.33)16>
```

Bergen received the message and sent one back:

```
bergen: received message (hello) from stockholm
(norway@192.168.220.207)6> bergen ! {send, stockholm,"heyhey"}.
bergen: routing message (heyhey)
{send,stockholm,"heyhey"}
```

4 Conclusions

In this assignment we had the opportunity to further our Erlang skills, we saw how the Dijkstra algorithm can be implemented in Erlang and how to create a functional router.