ID2201 HT23 Distributed Systems, Basic Course (50929)

Report Homework 2 - Routy - a routing network

by Eugen Lucchiari Hartz

September 20, 2023

1 Introduction

In this homework a small link-state routing protocol was developed using Erlang. This type of protocol is for example used for OSPF. OSPF, or Open Shortest Path First, is the most used routing protocol for Internet routers. It helps routers figure out the best paths for sending data. Accordingly, it's ensuring that data takes the shortest and most efficient route through the network.
The primary objectives of this assignment include understanding the structure of a link-state routing protocol, maintaining a consistent network view, and addressing network failure challenges. Potential improvements for the implemented router are also discussed in this report.
Accordingly, this assignment is mainly about how router protocols are structured and the functioning of a router in a distributed system. Routers are essential in distributed systems because they serve as traffic directors, directing data between various devices in a network. They help ensure that information reaches its intended destination efficiently, enabling effective communication and data sharing among connected devices.

2 Main problems and solutions

error variable '_' is unbound in: NewMap = lists:delete({Node, _}, Map),
in map.erl
fixed by NewMap = lists:delete({Node, notneededvariable}, Map),

error illegal guard expression in update_entry(Node, N, Gateway, [Entry | Rest])
when N < proplists:get_value(2, Entry)

fixed by assigning the result to a variable: NewLength = proplists:get_value(2, Entry)

problem returned list not sorted

```
316> dijkstra:update(london, 1, stockholm, [{berlin, 2, paris}, {london, 3, paris}]).
[{berlin,2,paris},{london,1,stockholm}]
```

fixed by adding this function

```
% Sorts the list based on the length of the path in ascending order.
sort_list(List) ->
    lists:sort(fun({_, Length1, _}, {_, Length2, _}) -> Length1 < Length2 end, List).
```

and using this function when the sorted list should be accessed or returned

2.1

- Why did we make entry/2 return 0 if the node is not found?

It makes sense to return 0 when the node is not found, because path lengths get compared numerically. Therefore this choice makes it easy to differentiate between reachable and unreachable nodes. Accordingly, 0 in this case represents an infinite length, so an unreachable node.

error you get when you forget to export a function, in this case update (happened several times)

```
75> dijkstra:update(london, 2, amsterdam, []).
** exception error: undefined function dijkstra:update/4
```

2.2

problem it returned that

```
332> dijkstra:iterate([{paris, 0, paris}, {berlin, inf, unknown}],
  .. [{paris, [berlin]}], []).
[{paris,0,paris}]
```

and then returned that

```
364> dijkstra:iterate([{paris, 0, paris}, {berlin, inf, unknown}],
  .. [{paris, [berlin]}], []).
[]
```

issue because it is not correctly updating the Sorted list with new entries for reachable nodes

changing this

```
case get_reachable_nodes(Node, Map) of
    {ok, ReachableNodes} ->
        UpdatedSorted = update_sorted(Node, Length, Gateway, Rest, ReachableNodes),
        iterate(UpdatedSorted, Map, [{Node, Length, Gateway} | Table]);
    _ -> iterate(Rest, Map, Table) % node not found in the map, continue with the rest.
end
```

with the modification when it finds reachable nodes it adds the current node to the Table and then continues with the updated sorted list

dijkstra:iterate([{paris, 0, paris}, {berlin, inf, unknown}],
[{paris, [berlin]}], []).
this returned [{paris, paris}].
Had to change this iterate(Rest, Map, Table) to

```
    _ ->
        iterate(Rest, Map, [{Node, Gateway} | Table])
end
```

 to update the table

problem table returns too many nodes

```
6> dijkstra:table([paris, madrid], [{madrid,[berlin]}, {paris, [rome,madrid]}]).
[{rome,paris},
 {berlin,paris},
 {madrid,paris},
 {paris,paris},
 {berlin,madrid},
 {madrid,madrid}]
```

should not return berlin, paris and madrid, paris, fixed by improving iterate and table function

4.

- Can you create an entry for a node that will make any message look old?

Yes, by initializing the highest message number for that node to a very high value. This way, any incoming message with a message number less than this high value will be considered old. This is the case because in this implementation, messages are considered old if their message number is less than or equal to the highest message number recorded in the history for that specific node.

```
% create a new history with an initial message number for the specified node.
new(Node) when is_atom(Node) -> % ensure that argument 'Node' is an atom
    #history{name = Node, highest_msg_number = 999999}. % set a very high initial value.
```

5
Error

```
(sweden@130.123.112.23)8> routy:start(router1, "Router1").
true
=ERROR REPORT==== 18-Sep-2023::21:33:37.322890 ===
Error in process <0.122.0> on node 'sweden@130.123.112.23' with exit value:
{undef,[{intf,new,[],[]},{routy,init,1,[{file,"routy.erl"},{line,15}]}]}
```

```
14    init(Name) ->
15        Intf = intf:new(),
```

fixed by renaming interfaces.erl to intf.erl

error

```
(sweden@130.123.112.23)23> lund ! {add, stockholm, {r1, 'sweden@130.123.112.23'}}.
** exception error: bad argument
     in operator  !/2
        called as lund ! {add,stockholm,{r1,'sweden@130.123.112.23'}}
```

it's an error in the assignment, because of course not process called Lund has been registered, the process is called r2, so instead do

```
(sweden@130.123.112.23)7> r2 ! {add, stockholm, {r1, 'sweden@130.123.112.23'}}.
{add,stockholm,{r1,'sweden@130.123.112.23'}}
```

5.4

- Could we also have a dummy entry for the node itself so that we would not need to have a special message entry to handle messages directed to the router itself?

Yes that is possible and would simplify the handling of messages to the router itself
It could be achieved by modifying the init function in routy.erl from this

```
init(Name) ->
    Intf = intf:new(),
    Map = map:new(),
    Table = dijkstra:table(Intf, Map),
    Hist = hist:new(Name),
    router(Name, 0, Hist, Intf, Table, Map).
```

to this

```
init(Name) ->
    Intf = intf:new(),
    Map = map:new(),
    Table = dijkstra:table(Intf, Map),
    % dummy entry for the node itself
    TableWithSelf = [{Name, 0, Name} | Table],
    Hist = hist:new(Name),
    router(Name, 0, Hist, Intf, TableWithSelf, Map).
```

With this inserted code line [{Name, 0, Name} gets added to the routing table. Accordingly it represents a routing table entry for the router itself. The number which represents the distance from the current node to the destination node is set to 0 to indicate that the router can reach itself directly, so it acts as its own gateway.

Discuss if any potential improvements to Routy could be made:

- Performance improvements would be necessary to be able to handle larger networks, for example by utilize Erlang's concurrency features to process messages in parallel. Accordingly routers can handle multiple messages simultaneously, improving overall throughput.
- Error Handling could also be improved, for example by using Erlangs integrated error mechanism try…catch
- You could also have auto-scaling, so that the number of routers get automatically adjusted based on the system load.
- You could also optimize data structures of for example routing tables by using hash tables and binary search trees which would increase the router performance

Overview of tags or atoms to indicate the type of messages being sent between router processes:

- {add, Node, Pid}: used to instruct a router to add a connection/interface to another router
- {remove, Node}: used to instruct a router to remove a connection/interface to another router
- {'DOWN', Ref, process, _, _}: used to indicate that a monitored process (specified by its reference) has terminated or exited. It's used for error handling when a connected node goes down.
- {status, From}: used to request or provide status information about a router process. It's used for debugging and monitoring purposes.
- {links, Node, R, Links}: used to exchange information about network links or connections between routers. It's used to update the router's view of the network topology.
- {route, To, From, Message}: used to send messages between routers, indicating the intended source, destination, and message content. It's the main tag used for routing messages within the network.
- update: used to trigger an update of the routing table based on the network's current state.
- broadcast: used to broadcast information or messages to other routers in the network

3 Evaluation

```
7> map:new().
[]
8> map:update(berlin, [london, paris], []).
[{berlin,[london,paris]}]
9> map:reachable(berlin, [{berlin,[london,paris]}]).
[london,paris]
10> map:reachable(london, [{berlin,[london,paris]}]).
[]
11> map:all_nodes([{berlin,[london,paris]}]).
[berlin,london,paris]
12> map:update(berlin, [madrid], [{berlin,[london,paris]}]).
[{berlin,[madrid]}]
```

```
13> dijkstra:update(london, 2, amsterdam, []).
[]
14> dijkstra:update(london, 2, amsterdam, [{london, 2, paris}]).
[{london,2,paris}]
15> dijkstra:update(london, 1, stockholm,
 .. [{berlin, 2, paris}, {london, 3, paris}]).
[{london,1,stockholm},{berlin,2,paris}]
```

```
16> dijkstra:iterate([{paris, 0, paris}, {berlin, inf, unknown}],
 .. [{paris, [berlin]}], []).
[{berlin,paris},{paris,paris}]
17> dijkstra:table([paris, madrid], [{madrid,[berlin]}, {paris, [rome,madrid]}]).
[{rome,paris},{berlin,madrid},{paris,paris},{madrid,madrid}]
```

erl -name sweden@130.123.112.23 -setcookie routy -connect_all false
c(map).
c(intf).
c(dijkstra).
c(hist).
c(routy).
c(routy_test).

routy:start(r1, stockholm).
routy:start(r2, lund).
r2 ! {add, stockholm, {r1, 'sweden@130.123.112.23'}}.

```
(sweden@130.123.112.23)7> routy:start(r1, stockholm).
true
(sweden@130.123.112.23)8> routy:start(r2, lund).
true
(sweden@130.123.112.23)9> r2 ! {add, stockholm, {r1, 'sweden@130.123.112.23'}}.
{add,stockholm,{r1,'sweden@130.123.112.23'}}
(sweden@130.123.112.23)10>
```

To test the functionalities of Routy, I developed a test program. It first initializes and starts five routers: "stockholm," "lund," "uppsala," "helsingborg," and "kiruna." It also establishes links between these routers, allowing them to communicate with each other. After that with the statelink function the test program simulates the routers broadcasting state-link messages to their neighbours. There's also an added pause timer:sleep(2000) which allows routers to update their routing tables. The next function called update updates the routers routing tables based on received state-link messages. The status function checks the status of each router and collects information about their state, interfaces, routing tables, and maps. The message function is used to test the messaging functionality. There's also a function which demonstrates sending a message from one router to another. It gets the references to the sender and destination routers, defines a message, and sends it. There's also a stop function to stop all the routers and their processes.

```
(sweden@130.123.112.23)23> routy_test:start().
{add,stockholm,{stockholm,'sweden@130.123.112.23'}}
```

```
(sweden@130.123.112.23)22> routy_test:statelink().
lund new links received [lund,uppsala,helsingborg,kiruna]
uppsala new links received [lund,uppsala,helsingborg,kiruna]
helsingborg new links received [lund,uppsala,helsingborg,kiruna]
kiruna new links received [lund,uppsala,helsingborg,kiruna]
stockholm new links received [lund,uppsala,helsingborg,kiruna]
helsingborg new links received [lund,uppsala,helsingborg,kiruna]
stockholm new links received [lund,uppsala,helsingborg,kiruna]
helsingborg new links received [lund,uppsala,helsingborg,kiruna]
Updated Table: []
Updated Table: [{kiruna,stockholm},
                {helsingborg,stockholm},
                {uppsala,stockholm},
                {lund,stockholm},
                {stockholm,stockholm}]
Updated Table: [{helsingborg,helsingborg}]
Updated Table: [{kiruna,stockholm},
                {helsingborg,stockholm},
                {uppsala,stockholm},
                {lund,stockholm},
                {stockholm,stockholm}]
Updated Table: [{helsingborg,helsingborg}]
stockholm new links received [stockholm]
lund new links received [stockholm]
uppsala new links received [stockholm]
helsingborg new links received [stockholm]
kiruna new links received [stockholm]
stockholm new links received [stockholm]
helsingborg new links received [stockholm]
stockholm new links received [stockholm]
helsingborg new links received [stockholm]
Updated Table: [{stockholm,lund},{lund,lund}]
Updated Table: [{kiruna,stockholm},
                {helsingborg,stockholm},
                {uppsala,stockholm},
                {lund,stockholm},
                {stockholm,stockholm}]
Updated Table: [{helsingborg,helsingborg}]
Updated Table: [{kiruna,stockholm},
                {helsingborg,stockholm},
                {uppsala,stockholm},
                {lund,stockholm},
                {stockholm,stockholm}]
Updated Table: [{helsingborg,helsingborg}]
helsingborg new links received [helsingborg]
stockholm new links received [helsingborg]
lund new links received [helsingborg]
uppsala new links received [helsingborg]
```

```
(sweden@130.123.112.23)23> routy_test:update().
Updated Table: [{stockholm,helsingborg},
                {uppsala,uppsala},
                {lund,lund},
                {kiruna,kiruna},
                {helsingborg,helsingborg}]
Updated Table: [{kiruna,stockholm},
                {helsingborg,stockholm},
                {uppsala,stockholm},
                {lund,stockholm},
                {stockholm,stockholm}]
Updated Table: [{kiruna,helsingborg},
                {uppsala,helsingborg},
                {lund,helsingborg},
                {stockholm,helsingborg},
                {helsingborg,helsingborg}]
Updated Table: [{kiruna,stockholm},
                {helsingborg,stockholm},
                {uppsala,stockholm},
                {lund,stockholm},
                {stockholm,stockholm}]
Updated Table: [{kiruna,helsingborg},
                {uppsala,helsingborg},
                {lund,helsingborg},
                {stockholm,helsingborg},
                {helsingborg,helsingborg}]
update
```

```
(sweden@130.123.112.23)28> routy_test:message(<0.163.0>, lund, stockholm, 'Hi from Stockholm!').
stockholm: routing message ('Hi from Stockholm!')
{route,lund,stockholm,'Hi from Stockholm!'}
GW: lund
lund: received message 'Hi from Stockholm!' from stockholm
```

```
(sweden@130.123.112.23)20> routy_test:send_example_message().
{route,lund,<0.144.0>,{greeting,"Hello, Lund!"}}
```

```
(sweden@130.123.112.23)22> routy_test:stop().
stop
```

Bonus

```
eugenius@Eugens-MacBook-Pro Code % erl -name sweden -setcookie routy -connect_all false
Erlang/OTP 26 [erts-14.0.2] [source] [64-bit] [smp:10:10] [ds:10:10:10] [async-threads:1] [jit] [dtrace]

Eshell V14.0.2 (press Ctrl+G to abort, type help(). for help)
(sweden@Eugens-MacBook-Pro.local)1> c(routy).
{ok,routy}
(sweden@Eugens-MacBook-Pro.local)2> routy:start(stockholm, stockholm).
true
(sweden@Eugens-MacBook-Pro.local)3> whereis(stockholm).
<0.98.0>
(sweden@Eugens-MacBook-Pro.local)4> whereis(stockholm).
undefined
(sweden@Eugens-MacBook-Pro.local)5>
```

```
eugenius@Eugens-MacBook-Pro code % erl -name germany -setcookie routy -connect_all false
Erlang/OTP 26 [erts-14.0.2] [source] [64-bit] [smp:10:10] [ds:10:10:10] [async-threads:1] [jit] [dtra
ce]

Eshell V14.0.2 (press Ctrl+G to abort, type help(). for help)
(germany@Eugens-MacBook-Pro.local)1> {stockholm,'sweden@Eugens-MacBook-Pro.local'} ! stop.
stop
(germany@Eugens-MacBook-Pro.local)2>
```

These two screenshots here above prove that it worked to establish a connection between the routers of the two countries. As proof, two different Erlang shells were started for this. One shell represents Sweden, the other one Germany. In Sweden a router "stockholm" was started. That the router was started was proved with the command whereris(stockholm). Subsequently, the router "stockholm" is stopped by the other shell, so to speak from Germany. If one executes now again the command whereis(Stockholm) in Sweden, it returns undefined. The router "stockholm" was thus successfully stopped from the other shell, so Germany.

```
(sweden@Eugens-MacBook-Pro.local)1> routy:start(stockholm, stockholm).
true
(sweden@Eugens-MacBook-Pro.local)2> {berlin, 'germany@Eugens-MacBook-Pro.local'} ! "Hello from Stockholm!".
"Hello from Stockholm!"
```

{berlin, 'germany@Eugens-MacBook-Pro.local'} ! "Hello from Stockholm!". Sending a message from Stockholm to Berlin.

4 Conclusion

In summary, the assignment was very useful to better understand of which parts a routing protocol consists of and how they function. It helped me a lot to get a better understanding of how data packets are sent between different connected devices of a network. It was also very interesting to test the own implementation and to discuss how the implementation of the router can be improved.