Report 2: Routy

Abyel Tesfay

September 22, 2021

1 Introduction

The following report intends to describe the students implementation of a *link-state* routing protocol in Erlang. The purpose with this homework is to let the student learn the structure of a link-state routing protocol, understand how a consistent view over other routers is maintained and reflect on problems related to network failure. The report covers how the student implemented all necessary modules, the main problems encountered and how they were solved. An example of a running *Routy* is presented along with a graphical description in a later section.

2 Routy implementation

This section describes the initial code of *map, hist, intf* and the main problems encountered in *djikstra*

2.1 Initial code

The map used by the router is represented as a list of entries, where each entry is a tuple consisting

1. The name of the node
2. A list of directly connected nodes

It was easily implemented with the functions *keyfind, keydelete and map* from the *list* module. In the *djikstra* module, the functions *entry, replace* and *update* were also implemented using the lists functions. For the sorting of the Lists though, an additional function *insert* was implemented to insert new entries into the sorted list.

2.2 Main problems

The *iterate* function was difficult to implement in which the student struggled for some time. The initial two statements were solved using pattern matching with functions but the last proved challenging.

```
iterate([], _, Table) ->
    Table;
iterate([{_, 'inf', _}|_], _, Table) ->
    Table;
iterate([{Node, N, Gateway}|Rest], Map, Table) ->
    NewSorted = findNewEntries(Node, N, Gateway, Map, Rest),
    ExtendTable = [{Node, Gateway}|Table],
    iterate(NewSorted, Map, ExtendTable).
```

One key factor was that the function needs to recursively traverse all directly connected nodes from one node and repeat the process. After that the current Node and its Gateway must be to the table before calling the *iterate* function with the new table and sorted list. The function *addNewEntries* helps the iterate function by recursively adding all Nodes and their directly connected Gateways to the sorted list.

```
addNewEntries([], _, _, _, Sorted) ->
    Sorted;
addNewEntries(Node, N, Gateway, Map, Sorted) ->
    case map:reachable(Node, Map) of
        [] ->
            update(Node, N, Gateway, Sorted);
        [First|Rest] ->
            AddedElem = addNewEntries(First, N+1, Gateway, Map, Sorted),
            addNewEntries(Rest, N, Gateway, Map, AddedElem)
    end.
```

Another small problem was that single entries in the sorted list were returned as a tuple, which some pattern matching errors.

2.3 The rest

Implementing the *intf* module was straightforward, the reader is referred to the *intf.erl* file for details. As for the *hist* module, the history can be represented as a list of tuples where each tuple consists the latest message number and the name of the node it came from. The *Routy* module itself was implemented with the given code from the assignment so the reader is referred there or the *routy.erl* file.

3 A routing example

For testing the implemented routing protocol and its modules, the student starts several routing processes. In this case each process represents a city in Sweden. The student then sends *add* messages to each router to connect

them manually. By sending some *broadcast* messages and *update* messages, a routing network was created. Figure 1 shows the initial network. (*unfortunately the student later had problem connecting all routers correctly so only two processes were used, hopefully more can be used during presentation*)

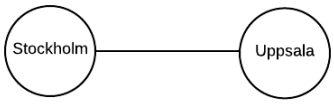

Figure 1: Graphical model of the link-state routing network

When sending a message from *Stockholm* process to *uppsala* the following was received

```
(sweden@83.252.11.133)10> r1 ! {send, uppsala, "hello from stockholm"}.
stockholm: routing message (hello from stockholm)
uppsala: received message from stockholm, hello from stockholm
```

Figure 2: Received message

As for when one of the routers shuts down (or crashes unexpectedly), the following response was given

```
(sweden@83.252.11.133)11> routy:stop(r1).
uppsala: exit received from stockholm
true
```

Figure 3: DOWN message