Routy A Small Routing Protocol

Jinjia Zhang

September 14, 2023

1 Introduction

In this assignment, I mainly implemented a small link-state routing protocol. But it has been divided into different parts. First, it is to implement a directional map where a given node can find nodes directly connected to it. The second one is the Dijkstra algorithm that computes a routing table. The table is the most important data structure that gives packet instructions on which interface it should forward to. Moreover, a router should have to keep track of a set of interfaces attached to it, so one process can send messages to other processes, also known as interfaces. Fourth, the history of recording the highest counter value received from each node is implemented to avoid resending messages infinitely. With the help of modules implemented before, the final part is the router process, which handles a few messages, such as adding a new interface. And, erlang provides an easy way to monitor and demonitor a process thus maintaining a consistent view.

On top of it, I also created an automated test that set up a network to test the Routy's functionalities and used Makefile to make compile easier.

2 Main problems and solutions

I have to say the first challenge is to understand the whole assignment. And I have little knowledge about link-state routing protocols. But, the assignment description also gives some key implementations, which helped me a lot. As I started writing each module, I gradually started to understand the entire assignment. The lesson I learned is to try to code and then you can understand something.

For the implementation, I have encountered a few problems. The first is to the all_nodes procedure in the map module, I know how to do it. Basically, take all nodes from the lists and remove duplicates. But, as for the functional language, it took me a while to figure it out, such as finding the proper helper functions in the lists module I needed to use.

The second one is the iteration of the Dijkstra algorithm, even though there is a detailed description on the assignment page. I couldn't understand the process. However after I drew an example, I understood the three cases of the iteration.

The third one is when I want to have a global variable, erlang doesn't allow us to do it. At first, I just copied-pasted. But it suffered when I wanted to change the value. In the end, I found that defining a macro is a good idea to avoid repetitive work.

The last one is how to debug the process. Now I only print the necessary information to debug but sometimes, we need a more practical way to debug, such as a debugger. I haven't used it yet, because printing is enough for me now, but later I definitely need to learn how to use the debugger.

3 Evaluation

First, I tested the router in one region, specifically in Sweden. The following is the topology. The source code is **test.erl**. We also can use this erlang file to test connectivities and get different router process statuses.

```
% +--------------+    +---------------+
% | stockholm(r1)+----+   lund(r2)    |
% +------+-------+    +---------------+
%        |
%        |
% +------+------+
% | malmo(r3)   |
% +------+------+
%        |
%        |
% +------+------+
% | uppsala(r4) |
% +------+------+
%        |
%        |
% +------+------+
% | goteborg(r5)|
% +-------------+
```

As for the bonus, I have three nodes, China, USA, and Sweden. The following is the topology. After these three nodes are started, one file, called **world.erl**, can be used to test the functionality. If I kill the USA node, the other two nodes will receive **DOWN** message. When you try to send messages to the USA Erlang node after it is dead, nothing happens. when we start the USA node again, we have to manually add the interface again, **world:start()**.

```
% +----------+            +--------+    +--------+
% |          |            |        |    |        |
% | losangeles            |shanghai+----+guangzhou
% +----+-----+            +----+---+    +--------+
%      |                       |
% +----+----+             +----+---+
% |         |             |        |
% | newyork +-------------+ beijing|
% +----+----+             +--------+
%      |
%      |
% +----+----+
% |         |
% |stockholm|
% +----+----+
%      |
%      |
% +----+----+
% |         |
% |gothenburg
% +---------+
```

In the current implementation, when I close one node, other nodes that received the **DOWN** message just remove this node from their interfaces. But, one more practical way is to also update the map and table at the same time. The other improvement could be to implement the automatic broadcast and update functionality when new nodes join the network.

4 Conclusions

After completing this assignment, I have some basic knowledge about the link-state routing protocol through changing the link-state message to compute the route table. Besides, I am getting used to functional programming. But the recursion is also a different part for me now, it is not easy to write a correct one at one time. Moreover, message passing during different nodes and handling errors if other nodes die are also familiar to me now. These blocks help me to build a more reliable distributed system in the future.