Report 2: Routy - A routing network

Emil Ståhl

September 17, 2019

1 Introduction

This report covers the implementation of a routing network which is able to route messages between nodes in a network using link state protocol. In order to create a routing table, Dijkstra's algorithm was used. The main topic covered in this work is to get a deeper understanding of how communication is handled in a network that contains several nodes / servers which are located around a huge region and where the nodes doesn't have direct access to every other node in the network.

This requires the system to keep track of information about the network and make frequent changes to the system. The system should be fault-tolerant and be able to handle the situation that certain nodes goes offline without affecting the performance of the system.

2 Main problems and solutions

The system consists of a number of files described below:

map.erl

Map is used to assist in the work of finding out which nodes are directly connected to each other. However, since nodes can come and go we need to update the map accordingly.

interfaces.erl

All of the references and process identifiers of the different routers is kept track of using interfaces. An interface is described by the symbolic name (city), a process reference and a process identifier. This file also consists of a bunch of functions used

to get name, reference or process identifier which are needed for different purposes throughout the system.

hist.erl

A problem faced with the implementation of link state protocol is resending of messages that has already been sent. This is taken care of by hist.erl by keeping track of the messages that are already sent as old messages and if the message is new it will update the history list thus keeping track of the messages sent to respective nodes.

dijkstra.erl

The real challenge in this work was to get the routing table up and working correctly. To get the gateways that should be used to route a message to an arbitrary node, a sorted list is used to keep track of all the entries. In order to update the shortest path a couple of functions is used, mainly entry, update, iterate, table, replace and route.

routy.erl

This file is the one that gets it all together and routes all the message using the files above. A shell is started and named with the following command:

```
erl -name country@ip-adress -setcookie routy -connect_all false
```

After the shell is up and running it's time to start some routing nodes in the country, often named after cities. This is simply done by typing routy:start(Reg, Name).

However, the different nodes will also need to be connected to some other nodes, which is done by `Reg1 ! {add, city, {Reg2, 'country@ip-adress'}}`. Now it's all set for message passing between nodes. Each of the router will maintain its own set of history, interfaces, map and a routing table.

Other problems not concerning the implementation was to compile the program, I spent some time trying to figure out why interface:new(). couldn't be executed from the routy module. It turned out it was because of that all the files hadn't been compiled, which is a quite basic problem but easy to forget when coming from a language like Java where you just press a button in the IDE and it's done.

Furthermore, testing out that the implementation had been done correctly wasn't easy. It required a lot of typing in the terminal but seems to be working fine. For further projects like this is would be preferable to set up a unit test taking care of this.

3 Evaluation

To evaluate that the features of the system works properly four command line shells were open on the same machine. Furthermore, a number of different router processes were then initiated in each shell/country. The countries and cities are given in the table below.

Sweden	Norway	USA	Canada
Stockholm	Oslo	Los Angeles	Montreal
Kiruna	Bergen	San Francisco	Toronto
Gothenburg	Stavanger	New York	Vancouver
Visby	Trondheim	Miami	Calgary

Each of the cities on the first row in the table were connected in the following manner; Stockholm - Oslo - Los Angeles - Montreal and thus a network of nodes is created. Messages passing between different countries worked smoothly.

3.1 Bonus task

To finish the bonus task the Stockholm node was closed which resulted in an exit message was sent out to all the connected nodes in micro seconds after the closing command had been issued.

4 Conclusions

This assignment was a bit more challenging than the last one. It contained a lot more files and code to keep track. Also, it felt like this assignment required a bit more functional programming concepts than the previous one.

The main topics learned was implementing a link state routing protocol and using Dijkstra's algorithm. In addition to this, it was a good learning experience to implement some of the features required in a message passing system, like history and acknowledgements. More challenging features can be implemented in this router in which delivery receipt message from the receiver will be sent out in acknowledgement. Using the built in libraries for list handling was also a new experience for me since we in previous courses wrote those functions by ourselves.