Routy: A small routing protocol

Tobias Johansson

September 23, 2015

1 Introduction

This report summarises the work of implementing a link-state routing protocol in Erlang. The server part was was given but the modules need to implement the server was not. The needed module API was given and the main work has been to implement and test these modules. After implementation of the the need modules, the router has has been tested. Through different setups of server nodes where the propagation of information in the network has been observed after broadcast.

2 Main problems and solutions

An Erlang routing server module named **routy** was given and a description of the API for additional modules need to implement the server. The main work have been to implement, test and analyse these modules. These modules was as following.

- **map**: for handle and and extract basic data from representation of a directional map.
- **dijkstra**: computes the routing table, and handles a sorted list of the routing table.
- **intf**: handles a set of interfaces for directly connected routers and can send messages to all directly connected routers (broadcast).
- **hist**: handles a history set for incoming link-state messages containing the router that started the propagation together with a time stamp from the message.

To implement **map** Erlang's built in module **lists** was used to manipulate lists. This module was studied thoroughly before production code was written and tested. Used functions from this module that was used was; `keystore/4`, `keyfind/3` and `foldl/3`. the latest mentioned function took tome to understand and was later used very frequently in the other modules. The module was tested with given tests.

2.1 Implemented modules

The heart of the **dijkstra** module was its function iterate that executes Dijkstra's algorithm that finds the shortest path between nodes in a graph, or in this case routers in a network. The algorithm can be described as:

1. Identify the root (the node itself)
2. Attach all neighbour node temporarily
3. Make link and node with least cumulative cost permanent
4. Choose this node
5. Repeat 2 and 3 until all nodes are permanent

The **dijkstra** moudle is used for each router in the network to calculate its routing table with the function `table` that takes a list of its directly connect routers (Gateways) and a its map over the network as input and generates a routing table as output.

Furthermore the interface module **intf** that handles a set of interfaces for directly connected routers was implemented. Each interface contains of the information a symbolic name, a process reference and a process identifier (PID). Its function `broadcast/2` takes a message and a set of interfaces as input, extract the PID for each interface and send a message to the PID:s corresponding router using the following code.

```
broadcast(Message, Intf) ->
    lists:foreach(fun({_, _, Pid}) ->
        Pid ! Message end, Intf).
```

Later in the report it will be showed how this function is used in the routing module **routy**. Last of the implemented modules is **hist** that handles the history for incoming link-state messages. This history contains the symbolic name for each router's link-state messages that have been received and a corresponding time stamp (counter) sent from the router. It mainly have a function `update/3` that takes a symbolic name, a counter value and the history as input. The output is an update history or the atom old `old` if the incoming counter was not higher (newer) than the eventual stored counter value for the symbolic name, so the symbolic name for each router needs to be unique.

2.2 Main module

The main module **routy** was given, it contains a routing server that can receive messages for modifying its interface set, handle link-state messages and routing messages.

While analysing the code it was seen that it could receive `{status, From}` and answer to the PID given in `From` the status of the router with its symbolic name, counter value, history, interface set, routing table and its map over the network. Therefore an extra function was written, `status/1` that takes a routers PID and sends a status request to it, receives the answer and prints it in the terminal.

3 Evaluation

Two thats tests was done to evaluate the functionality of the router was as following. Firstly four routers connected bidirectional in a circle of 4 routers, all router was connected directionally to its neighbour. Secondly four routers connected with directed connections in a circle with all directions in the same so we get a directional loop.

Firstly the first test is reviewed. When a `broadcast` was sent from a router, when all routers had empty maps it could be seen that its information of route 1's directly connected routers (interface) propagated over the network and the maps in all other router got update with this information. How was this achieved? Router 1 sent a list of the symbolic names in its interface using the `broadcast/2` function in the **intf** module, the message it sends with it is `{links, Name, N, intf:list(Intf)}` where which gives the sending routers name, its counter value that get incremented by 1 after the broadcast, and the list of directly connected routers

The receivers of the broadcast message updates its history and if it was a new message, from a new router or with a counter value higher than what is stored, then it sends the same message to all its directly connected routers. In this way the broadcast message information propagates over the network. On startup of a server its counter is set to 0. With help of the counter value cyclic paths is avoided because we can decide the age of the incoming message relatively to the last broadcast receive from originated router.

Also ordinary messages was sent over the network and it was observed that they got correctly routed. To be able to route an `update` request was needed to be send to the server, this calculates the routing table using the `table/2` function in the **dijkstra** module. We observed in the first test that the shortest possible path between two routers was used through observe the routings. It was possible to send messages to all nodes in the first test. But in the second test it was only possible to send messages one direction, because the connections is not bidirectional.

4 Conclusions

The process of implementing the routers was very time consuming but very informative for the author that learned a lot about the link-state protocol. The implementation was successful, routing works fine with **routy**, but firstly bugs in the written extra modules stopped a setup with multiple routers and tests of the routing. But this debugging forced the author to really understand the **routy** module and the link-state protocol. In the seminar it will be an interesting follow up of what happening in case of network failure.