Routy - a small routing protocol

Mattias Cederlund, mcede@kth.se

September 24, 2014

1 Introduction

The task was to implement a link-state routing protocol in Erlang.

The objective of the assignment was to gain knowledge in the structure of link-state protocols and how a consistent view is maintained among nodes in a network. Also problems related to network failures will be reflected upon.

The implementation should be able to route messages between routing processes with logical names such as 'stockholm' and 'london'. To do this, the router needs to consult its routing table to find which gateway it should send the message to.

2 Map and Dijkstra

A Map is a representation of the network structure used by the routers to calculate the routing table. In this implementation the Map is a list with the format [{Node, [Links]}, ...], where Node is the node name and Links is a list of all nodes reachable from Node.

For convenience, methods for modifying and querying the Map (new/0, update/3, reachable/2, all_nodes/1) were implemented. Here, the functions from the lists library (keysearch/3, keyreplace/3, foldl/3) and the sets library (from_list/1, to_list/1) came in handy.

The routing table used in this implementation is on the format [{Node, Gateway}, ...]. If we want to send a message to Node we should send it

to Gateway (for direct access or routing). To build the routing table dijkstras algorithm was used and the implementation followed the description provided. I think that the detailed description of the algorithm made the implementation quite straight forward, as I was able to make use of the lists library functions map/2 and foldl/3 to avoid a lot of messy recursive functions. Nevertheless you needed to watch your step when translating the description into code.

3 History, interfaces and Routy

One of the things the router needs to keep track of is its current interfaces. Therefore some convenience functions were implemented to ease the handling of interfaces. I think the implementation of the interfaces was very straight forward.

The router also needed to keep track of what link-state messages it received from other routers, to make sure no old links make it into the network Map. A history list was implemented where the router keeps a list of routers it received link-state messages from together with the message number (included in every message). When the router receives a link-state message, it compares the message number with the one kept in the history. Only if the link-state message number is higher than the one in the history it will update the Map and forward the message to all routers connected through its interfaces. To make sure all link-state messages originating from the router itself was ignored, an entry with the routers name and inf as message number was added when initiating the router.

The router code was already provided and it was simply put together and some tests were performed. When testing Routy I managed to send and route messages through a network of routers. Although, with the current implementation there was a hassle when setting up the network since I needed to add interfaces to every router manually. Then I also needed to broadcast the routers interfaces and update the routing table manually for every router. With multiple routers in a network this quickly became annoying.

4 Conclusions

I think this router protocol implementation is a good example of problems with making sure all nodes have both consistent and correct information.

Having to broadcast and update the routing table manually shows how large and complex such a task might be, especially in a large network.

One problem I found with this implementation is that when a node goes down its connected nodes gets the 'DOWN' message and removes the interface, but this information is not propagated to the other routers in the network since we dont broadcast it automatically. Also the routing table would need to be updated to keep a correct view of the network.