Report 2: Routy

Nikita Gureev

September 14, 2016

1 Introduction

In this homework a simple router programmethat uses a link-state routing protocol is implemented in Erlang. The actual program is then used as a singular router that has an arbitrary name. A network of such processes is built and examined, both in the state when all of the routers are working and when some of the routers fail. The effects of such failure are also examined and described.

2 Main problems and solutions

The main problems that were encountered during this task were the usage of lists library of Erlang and testing the network in case of router failure. The first problem was mainly due to the lack of experience with the language, as some of the functions used required a degree of understanding of functional paradigm. The second problem was due to the fact that all of the processes were running on a singular computer and some of the required events happen much faster than they should, e.g. the router failure. Also the disconnected network card can not influence the router behaviour as they do not require network connection in order to function.

3 Evaluation

Overall, the implementation of link-state protocol ises Dijkstra algorithm to be aware of which gates are the to be used in order to route the message to intended destination. The links are directional, which means that sometimes messages can only be sent in one direction. That is the case, when we do not add some routers to each other and only add one to another. The implementation includes four different modules, each of which has a distinct purpose and can be tested accordingly.

First is the map module, which includes an implementation of a map that stores a node and all the links that it has. This module can also check if a node is reachable and provide the links, by which it can be accessed.

The second module is the dijkstra module, which includes the implementation of the algorithm to find the closest gateway to a node. It constructs a routing table that stores a destination and a gateway, to which a message should be sent in order to route it there. As input it takes gateways and a map, which has all of the connections between gateways and other nodes. It then iterates through them until there are only dummy nodes in the list, adding the destination/node pairs to the table as it goes through gateway links that are provided by the map.

The third module is the history, which simply stores pairs of node name and the highest message counter received from that node. It does so in order to discard older messages that can circulate through the router network.

The fourth module is the interfaces, which keeps track of the current set of routers that are available for message passing. It stores their name, reference and pid and can also broadcast messages to all of the stored routers.

The fifth and last module is the routy, which handles all of the message receiving, routing and other possible commands that it can receive. It can receive a new router and add it to its interface, while monitoring its state. On the event that some router goes down, the process excludes the downed router out of its interfaces and updates the routing table accordingly.

After the implementation was finished a network of routers was built and examined. As a base case a network of four nodes was examined.

```
routy:start(r1, swe).
routy:start(r2, nor).
routy:start(r3, fra).
routy:start(r4, spa).
%Connect swe and nor, bidirectional
r1 ! {add, nor, r2}.
r2 ! {add, swe, r1}.

%Connect fra and nor, bidirectional
r2 ! {add, fra, r3}.
r3 ! {add, nor, r2}.

%Connect fra and spa, bidirectional
r4 ! {add, fra, r3}.
r3 ! {add, spa, r4}.

r1 ! broadcast.
r2 ! broadcast.
r3 ! broadcast.
r4 ! broadcast.

r1 ! update.
```

```
r2 ! update.
r3 ! update.
r4 ! update.
```

Such a network could send messages in both directions, e.g. from swe to spa and from spa to swe. However, the items of most interest were the directional links, e.g. one can send a message from one router to another without problem, but a message going in the opposite direction may not be delivered, as all of the nodse that reroute this message must have bidirectional links. The case was tested with two hops in between, and it was possible to send a message from fra to swe, but not the other way around.

```
routy:start(r1, swe).
routy:start(r2, nor).
routy:start(r3, fra).
%Connect swe and nor, bidirectional
r1 ! {add, nor, r2}.
r2 ! {add, swe, r1}.

%Connect fra and nor, only from nor to fra
r2 ! {add, fra, r3}.

r1 ! broadcast.
r2 ! broadcast.
r3 ! broadcast.

r1 ! update.
r2 ! update.
r3 ! update.
```

Another facsinating case is a link that makes it impossible to send messages further than one hop into another part of the network. Consider such a network:

```
routy:start(r1, swe).
routy:start(r2, nor).
routy:start(r3, fra).
routy:start(r4, spa).
%Connect swe and nor, bidirectional
r1 ! {add, nor, r2}.
r2 ! {add, swe, r1}.

%Connect fra and nor, only from nor to fra
r2 ! {add, fra, r3}.
r3 ! {add, nor, r2}.
```

```
%Connect fra and spa, bidirectional
r4 ! {add, fra, r3}.
r3 ! {add, spa, r4}.

r1 ! broadcast.
r2 ! broadcast.
r3 ! broadcast.
r4 ! broadcast.

r1 ! update.
r2 ! update.
r3 ! update.
r4 ! update.
```

It seems that it should be possible to send a message from swe to spa, however, as the protocol needs the nodes to be able to broadcast their existance, the link from fra to spa is unknown for nor, and as such, can not be used.

4 Conclusions

In conclusion, the problems presented in this assignment are relevant to the real world in many ways. The link-state protocol helps to clarify the finer problems of working in distributed environment, where different nodes interact with each other over network. Another matter of importance is the algorithm that is used to route messages through the network, as it is extremely useful in understanding how message passing through a number of nodes works in distributed systems. The examination of the effects on the network in case of node failure or in case of passing messages only in one direction is also of much interest, as it helps to better understand how a distributed system behaves, when a number of nodes go down. In addition, it helps to understand how bottlenecks can occure, in case a number of crucial nodes are down and most of the communication has to occur through a single node. Overall, this assignment is very useful and it helps to understand many common occurences and problems in distributed systems.