Routy - a routing network

Xin Ren

September 12, 2017

1 Introduction

In this homework, a link-state routing protocol is implemented in Erlang. Routing processes with logical names such as london, berlin, paris etc are implemented. Routers can be connected to each other with directional (one-way) links and they can only communicate with the routers that they are directly connected to.

The routing processes is able to receive a message of the form {route, london, berlin, "Hello"} and determine that it is a message from berlin that should be routed to london. A routing process consults its routing table and determines which gateway (a routing process that it has direct connection to) is best suited to deliver the message. If a message arrives at its destination (the router called london) it is printed on the screen. Messages for which paths are not found are simply thrown away, no control messages are sent back to the sender.

2 Detailed Implementation

Following modules are implemented in this homework.

1. map.erl: The map is represented as a list of entries where each entry consist of a city with a list of directly connected cities. This module gives us a very quick way of updating the map, simply replace an entry with a new entry and get the directly connected cities of a given city.

2. dijkstra.erl: This module applies Dijkstra algorithm to compute a routing table. The table is represented by a list with one entry per node where the entry describes which gateway, city, should be used to reach the node. The input to the algorithm is:
    - a map
    - a list of gateways to which we have direct access

3. intf.erl: A interface is described by the symbolic name (london), a process reference and a process identifier. A routing processor keeps track on a set of interfaces which stands for the nodes that it has direct access to. This module provides functions to add, remove and look up interfaces.

4. hist.erl: Each routing process maintains history of link-state messages it has received from other nodes to avoid cyclic paths. This module provides function to update the history.

5. routy.erl: This module can start multiple concurrent routing processes. Each routing process has a state which is set to initial empty value when the process is created and can be printed out at any time. The state contains following info:
    - a symbolic name such as london

- a counter
- a history of received messages
- a set of interfaces
- a routing table
- a map of the network

The user can add directly connected nodes to a routing process by adding interface and update routing table. The user can also initiate link-state message broadcast to all the directly connected nodes. The nodes which have not received this message previously will also broadcast this message after receiving it and update the map which can be further used to update the routing table.

The routing process routes messages to other nodes and print messages that have it as destination.

Erlang monitors are used by routing process to detect if other nodes are reachable. When a node becomes unreachable, it is removed from other nodes' interface lists.

3 Problems and solutions

1. After a routing process broadcasts link-state messages, the nodes that receive it may broadcast it again, which mean the routing process may receive its own link-state messages which causes cyclic paths. To avoid this, an entry with counter "inf" attached to its own node name is added into the routing process's history when it is created.

   hist: new(Name) -> [{Name, inf}].

4 Evaluation

Test was done by starting several erlang nodes on one machine and then starting several routing processes on each erlang node.

Test for routing messages among a network with several machines was done in the homework presentation in a group.

5 Conclusions

From the work, I have learnt the structure of a link-state routing protocol, the Dijkstra algorithm, how to maintain a consistent view, and how to handle network failure in distributed systems. I have also learnt how to send messages to an erlang process and receive messages with an erlang process through this homework and I am more familiar with erlang lists library.