Report 2: Routy

Nagasudeep Vemula

September 18, 2019

1 Introduction

In this assignment we implement a link state routing protocol using erlang. We go in detail into the Dijkstra/Open shortest path first(OSPF) algorithm, the most used routing protocol for routers. The Dijkstra algorithm helps us to find the shortest route it takes for a message to travel among the graph of nodes. We also learn about:-

- The structure of a link-state routing protocol.
- Maintaining of a consistent view.
- Observe network failure related issues.
- Greater handling of lists and tuples, how to record, search and modify information in erlang.

2 Main Problems and Solutions

I faced difficulty in learning about the higher order functions approach in erlang as this was a relatively new concept for me. Due to this methods such as foldl which could simplify the map reduction were difficult for me to implement, however with the help of the teaching assistants and the erlang documentation I was able to get more clarification of the same and also gained insights into the functions keysearch,keydelete and map.

Map Module: The map is used to give the directions to the nodes connected to a given node. One of the key components of this module is the function reachable() which returns a list of notes directly reachable from a particular node. This was enabled by using the following snippet mainly:-

```
case lists:keysearch(Node, 1, Map) of
  {value, {Node, Nodes}} ->
     Nodes;
  false ->
     []
 end.
```

We also update the map given a set of links that are reachable from a particular node. The key in this case will be the node that is in the first position of the tuple and this is done by calling the update function update(Node, Links, Map).

Dijsktra Module: This module is for implementing the dijsktra algorithm. Its inputs are a map and a list of gateways and it uses them to generate the routing table. There are 2 cases for the iteration function in this module, one where we receive an empty list and respond with a table and the other case where a node with infinity weight is in the first place of the list and respond with the table.

The table is computed using the following code:

```
table(Gateway,Map)->
        Nodes = map:all_nodes(Map),
    Rest = lists:filter(fun (X) -> not lists:member(X, Gateway) end, Nodes),
    Direct = lists:map(fun (Nd) -> {Nd,0,Nd} end, Gateway),
    Indirect = lists:map(fun (Nd) -> {Nd,inf,unknown} end, Rest),
    Sorted = lists:append(Direct, Indirect),
    iterate(Sorted, Map, []).
```

Interface Module: A router needs to keep track of its set of interfaces in order to know the address of the nodes. These are described using a symbolic name, process reference and a process identifier. The usage of the ref and name functions is interesting as ref is used to return a reference given the name and name is used to find the name of an entry given a reference. They are implemented as follows:-

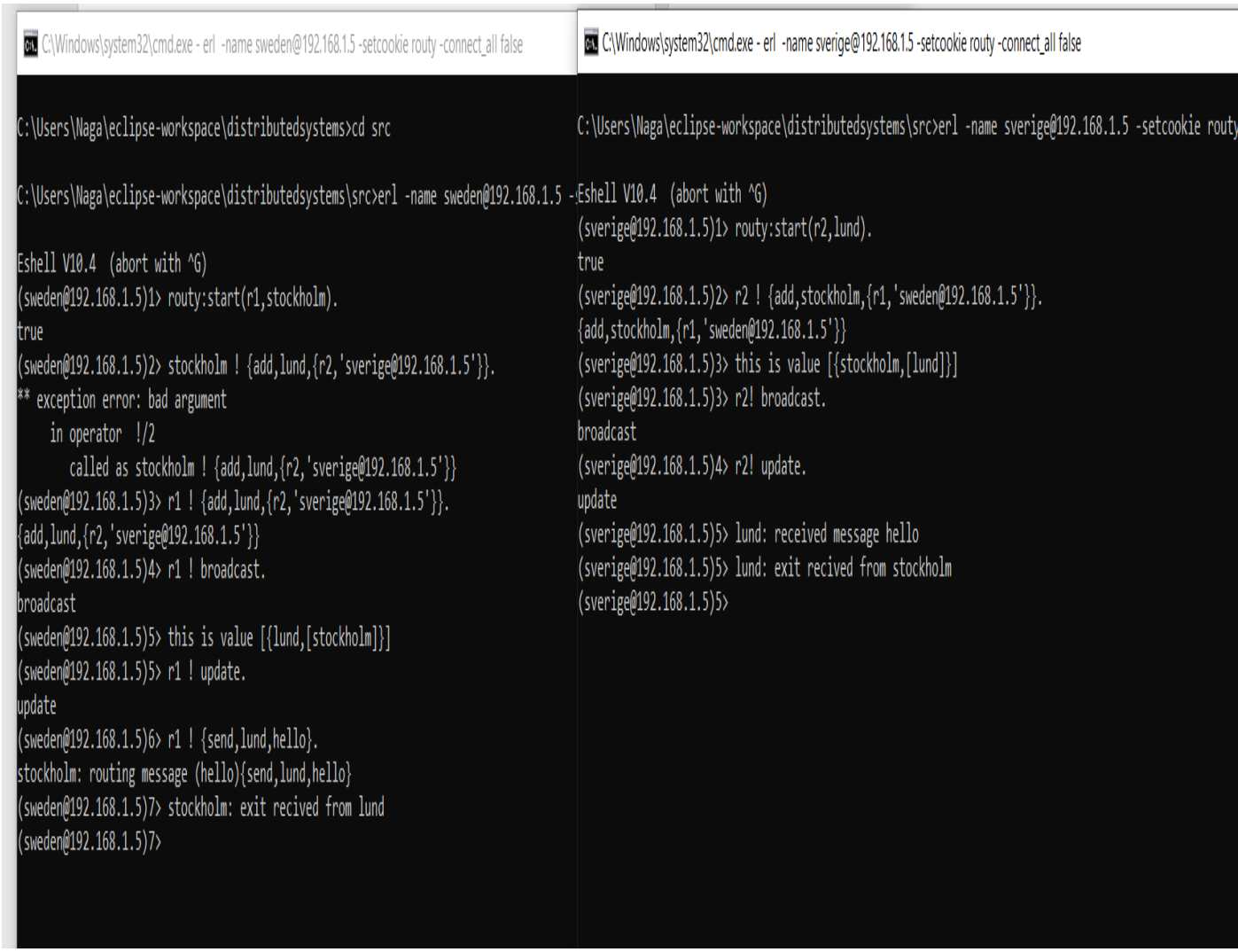
```
        ref(Name,Intfs)->
        case lists:keysearch(Name, 1, Intfs) of
        {value, {_, Ref, _}} ->
            {ok, Ref};

       name(Ref,Intfs)->
       case lists:keysearch(Ref, 2, Intfs) of
        {value, {Name, _, _}} ->
            {ok, Name};
        false ->
            unknown
            end.
```

History Module: This module is used for avoiding cyclic paths and in turn preventing the resending of messages. The update functionality of this module is used to check if the message number from a node is new or old.

```
update(Name,X,History)->
        case lists:keysearch(Name, 1, History) of
        {value, {Name, Y}} ->
            if
                X > Y ->
                    {new, [{Name, X}|lists:keydelete(Name, 1, History)]};
                true ->
                    old
            end;
        false ->
            {new, [{Name, 0}|lists:keydelete(Name, 1, History)]}
    end.
```

3 Evaluation

I carried out the tests as described in the assignment documentation for the routy module. Tests were carried out on the map, interface, Dijkstra and history modules to ensure that the code is ready for the routy section. The tests run for routy were in accordance with the document wherein I spawned an erlang shell named Sweden with 2 routy processes named Stockholm and Lund. I organized a connection between these 2 and broadcast the messages so that in theory the nodes can identify their neighbours. Finally the tables were updated using the update command and messages were sent between the two. Picture of the same is shown below :-



4 Conclusion

This problem has given me insights into how a routing protocol is to be designed and developed and what are the considerations and constraints needed in order to formulate a reliable one that prevents loss of messages and is very efficient at the same time.Routing is an important concept in distributed systems,this also gave me a chance to learn about the

Dijsktra algorithm and also implement something of that complexity in Erlang.We learn how to implement the OSPF protocol with erlang.
Lastly but probably one of the most important takeaways was the introduction to list handling with the methods keysearch,keydelete and map which are useful for the list operations and will be critical in solving many problems using erlang.