ID 2201 Report

Routy: A Small Routing Protocol

Yilai Chen

Project Overview

This report outlines the development and structure of a distributed routing system implemented in Erlang. The system is designed to simulate network routing where multiple nodes communicate by sending messages through a dynamically updated routing table. The nodes represent different geographical locations, and the system handles the addition, removal, and message routing between these nodes. The project includes several modules, such as `map`, `intf`, `dijkstra`, `hist`, and `routy`, each responsible for specific functionalities.

Modules Overview

1. `map.erl`: This module manages the network topology, storing the connections between nodes in a structure called `Map`. It supports adding new nodes, updating node connections, retrieving all nodes, and removing nodes.
    - `update(Node, Links, Map)`: Adds or updates a node and its connections.
    - `remove(Node, Map)`: Deletes a node from the network and removes all links to this node from other nodes.
2. `intf.erl`: The `intf` module handles the interfaces between nodes, maintaining a list of connected nodes and their associated processes.
    - `add(Name, Ref, Pid, Intf)`: Adds a new node and its process reference to the interface.
    - `remove(Name, Intf)`: Removes a node from the interface list.
    - `lookup(Name, Intf)`: Finds the process identifier for a given node name.
    - `broadcast(Message, Intf)`: Sends a message to all nodes in the network.
3. `dijkstra.erl`: This module implements Dijkstra's algorithm to calculate the shortest paths between nodes and build the routing table.
    - `table(Gateways, Map)`: Constructs the routing table for a node by calculating the shortest paths to other nodes based on the current network topology.
4. `hist.erl`: The history module keeps track of message updates between nodes, ensuring that the system avoids routing loops or duplicate transmissions.
    - `update(Node, N, History)`: Updates the history of a node, ensuring that only newer updates are processed.
5. `routy.erl`: The core module of the system, responsible for node communication, message routing, and managing the state of each node. It spawns processes for each node and manages their lifecycle, including adding new nodes, handling node removals, and updating routing tables.

- `router/6`: The main loop that handles incoming messages, processes routing information, and sends messages between nodes.
- `broadcast`: Sends the routing table and status updates to all connected nodes.

Process Lifecycle

The system starts by spawning processes for each node using `routy:start(NodeName)`. Each node maintains its state, including the network topology (`Map`), routing table (`Table`), and connected interfaces (`Intf`). Nodes can send messages to each other, and the routing table ensures that messages follow the shortest path.

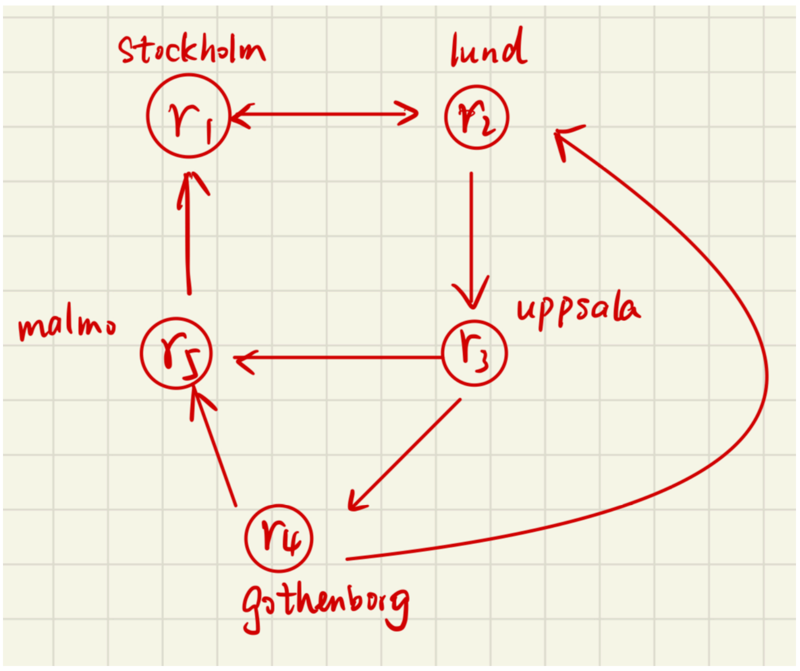
When a node is removed (e.g., `r5`), the system updates the network topology by removing the node from the map and recalculating the routing tables for all remaining nodes. This is done using the `{'DOWN', Ref, process, _, _}` message in `routy`, which triggers a broadcast to notify other nodes about the removal and initiates an update of their routing tables.

Example Scenario

In a scenario with nodes `stockholm`, `lund`, `uppsala`, `gothenburg`, and `malmo`, the following actions demonstrate the system's behavior:

- **Node Addition**: New nodes are added using `routy:start(NodeName)`. They are connected through their interfaces, and each node maintains its routing table, allowing messages to be routed efficiently.
- **Node Removal**: When `malmo` is removed (`r5`), the system broadcasts the removal to all nodes, and each node updates its routing table, ensuring the network remains functional.
- **Message Routing**: After node removal, a message from `uppsala` to `stockholm` is routed through the updated shortest path, bypassing the removed node `malmo`.

Diagrams and Output



```
7> test:run().

Ask lund for status from stockholm
Ask stockholm for status from uppsala

Send message from stockholm to lund
stockholm: routing message ([72,101,108,108,111])
lund: received message [72,101,108,108,111]

Send message from uppsala to stockholm
uppsala: routing message ([72,101,108,108,111])
malmo: routing message ([72,101,108,108,111])
stockholm: received message [72,101,108,108,111]

Send message from gothenburg to lund
gothenburg: routing message ([72,101,108,108,111])
{route,lund,gothenburg,"Hello"}
lund: received message [72,101,108,108,111]
```

Conclusion

This distributed routing system simulates dynamic message routing between nodes in a network. It handles node addition and removal, dynamically updates routing tables using Dijkstra's algorithm, and ensures that messages are routed through the shortest available path. The system's ability to update itself in response to network changes ensures robustness in handling failures such as node removals.