HW2 Report, Rudy: A Small Routing Protocol

David Fischer

September 16, 2025

1 Introduction

The link-state routing protocol is a key factor enabling the modern internet. OSPF (Open Shortest Path First) is one of the most widely deployed link-state protocols, running in enterprise environment and used by ISPs (Internet Service Providers.)

As part of this assignment *Routy*, a small routing protocol implementing link-state was implemented. The routers, or Erlang processes, are able to build a routing table using the Dijkstra algorithm, send messages and share link-state information with other routers using *signals*, and communicate over different machines.

2 Main problems and solutions

2.1 Module Based Approach

The primary challenge this assignment presented was the module based approach, building up five modules with little correlation made it difficult to see a bigger picture early on.

To overcome this and prevent any tech debt from building up from incorrect module implementations, a set of tests was created based on the assignments given test commands to ensure proper functionality before moving on to the next module. The purpose of each module became immediately clear after building the main router implementation, tying together all of the modules and observing their behavior in the complete system solidified the understanding of the entire stack.

2.2 Debugging Final Implementation

Debugging the final implementation did present challenges primarily with the traceability of calls in a distributed system. With five modules and more than three processes running at a time, reading and correctly understanding the diluted log output was troublesome.

This was solved by ensuring a structured log output, debugging single modules and removing log output from any unrelated components to reduce noise, and automating test cases.

2.3 Duplicate Update and Broadcast

The issue the most time was spent on was each router needing *two* broadcast and update cycles to create a complete topology, it was also tackled last since it didn't make any functional difference for testing.

After the debugging methods from the previous section were applied the issues was traced to the *hist* module. To implement the module, the *Map* data type and `get(Key, Map, Default) -> Value | Default` were used. All routers start out with *0* as their initial message number while the `Default` parameter for the `get/3` function was also set to *0*. This resulted in the first message from any router being disregarded as old, since they weren't registered in the map yet, resulting in the first message defaulting to being old. The second message with the number *1* was then interpreted as new. Additionally, to make routers always interpret their own messages as old, Erlang's type order placing atoms before numbers was applied, setting the nodes own entry to *inf*, thus making any greater than comparison result in `true`.

3 Evaluation

To efficiently evaluate and debug the implementation, a testing module to set up multiple routers, connections, and messaging across different countries, or machines, was created. Figure 1 is an approximate illustration of the testing environment including nine routers in Austria and five routers in Germany with directional and bidirectional connections.

This setup proved crucial in confirming functionality and analyzing the behavior of the distributed system, also helping identify many of the nuances of this link-state implementation. The most significant learnings came from shutting down individual routers and seeing the Dijkstra algorithm compute new optimal paths going around the outage, or correctly identifying that certain routers are now unreachable through the `entry/2` function. This also caused the most interesting edge case when broadcasts and updates aren't propagated through the entire network, leading to infinite message routing loops between two nodes. This occurs when a router doesn't have an up to date view and tries to communicate over another router which does. In the worst case, the second node has identified the message path to go back over the original router, leading them to continuously send the message back and forth.

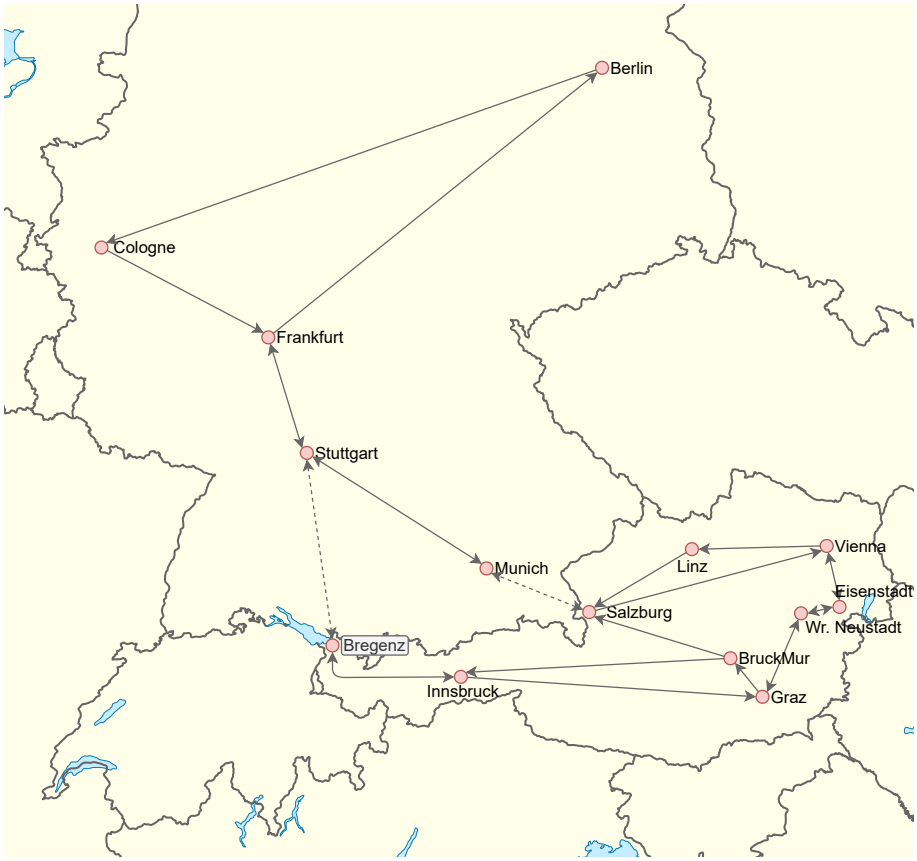

Figure 1: Map of routers used for evaluation.

4 Conclusions

This assignment served as an excellent introduction to the inner workings of link-state protocols. While the sequentially implemented modules did present some initial challenges and lead to some functions being implemented unconventionally for a functional language, showing the authors object oriented background, seeing messages routed over routers written by other classmates in the "The world" bonus task was gratifying. Additionally, the bonus task also reinforced the importance of common interfaces in distributed systems, as any deviation would lead to routers throwing an error or not being able to communicate.

Future development should include automatically and periodically updating and broadcasting each routers links to enable a more seamless operation, a separate logging module to handle log levels and structured output, and reducing the complexity and size of the `router/6` function in the *routy* module.