COSC364 Assignment One

Nicholas Coetzee - 71838492 - nco63

Noah Davis - 55570859 - nda87

Noah Davis Contributions

Calculate routes function

Instantiate ports function

Performed tests on a small network of 3 routers.

Created configuration files for figure 1

Performed tests on the figure 1 and 2 networks

Debugged errors found

Report writing

Set up triggered updates

Unit Testing

Nicholas Coetzee Contributions

Created Router class

Created all other methods for router class

Setup configuration files for three node test networks

Performed tests on 3 router network and implemented bug fixes

Implemented timer dictionary logic

Routing loop

Performed tests and made fixes on figure 1 and 2 networks

Report writing

Split: 50/50

1. Testing

1.1 Testing on a three-node network (Noah and Nic)

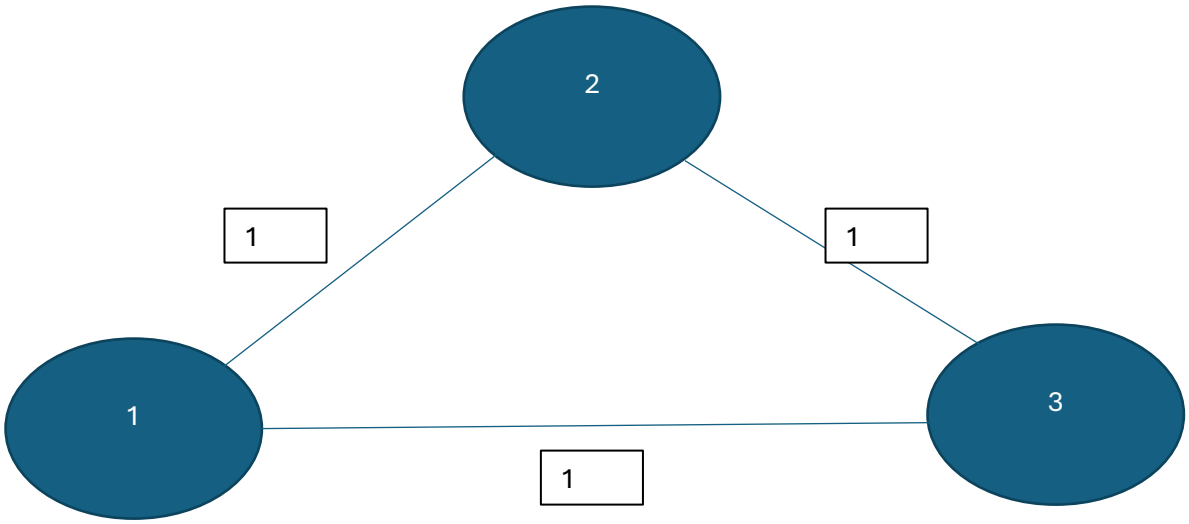

Figure 1: Three-Node Network Used for Initial Tests

After the initial implementation of the calculate_routes() function, several issues arose. Firstly, there was an issue where the routing tables of the three routers appeared to converge initially. However, after running for a while all three routes would become invalid and be garbage collected. This was due to an incorrect implementation of poison reverse. Upon fixing this bug all three routing tables converged correctly.

The second issue encountered was discovered was testing what happens if one of the three routers goes offline. Our procedure for this test is described as follows. All three routers are brought online, and their routing tables converge. Then router two is brought offline. The expected outcome was that routers one and three would timeout and garbage collect their route to router two. However, this resulted in routes 1 -> 3 and routes 3 -> 1 timing out and garbage collecting before their respective routes to 2 time-out and garbage collect. The reason for this was that we were only checking and updating route timers if they were received from a non-neighbor node (using dest_id). We needed to also update routes received from neighboring nodes (using sender_id). Adding this functionality ensured that router 2 would time out and garbage collect before routers one and three. However,

this led to an issue where after garbage collection of route two, routes one and three would subsequently time out. This was solved by updating the route timers dictionary upon successful packet reception in the decode_packets() function. After these updates were made this simple network behaved correctly.

1.2 Testing on a seven-node network (Nic)

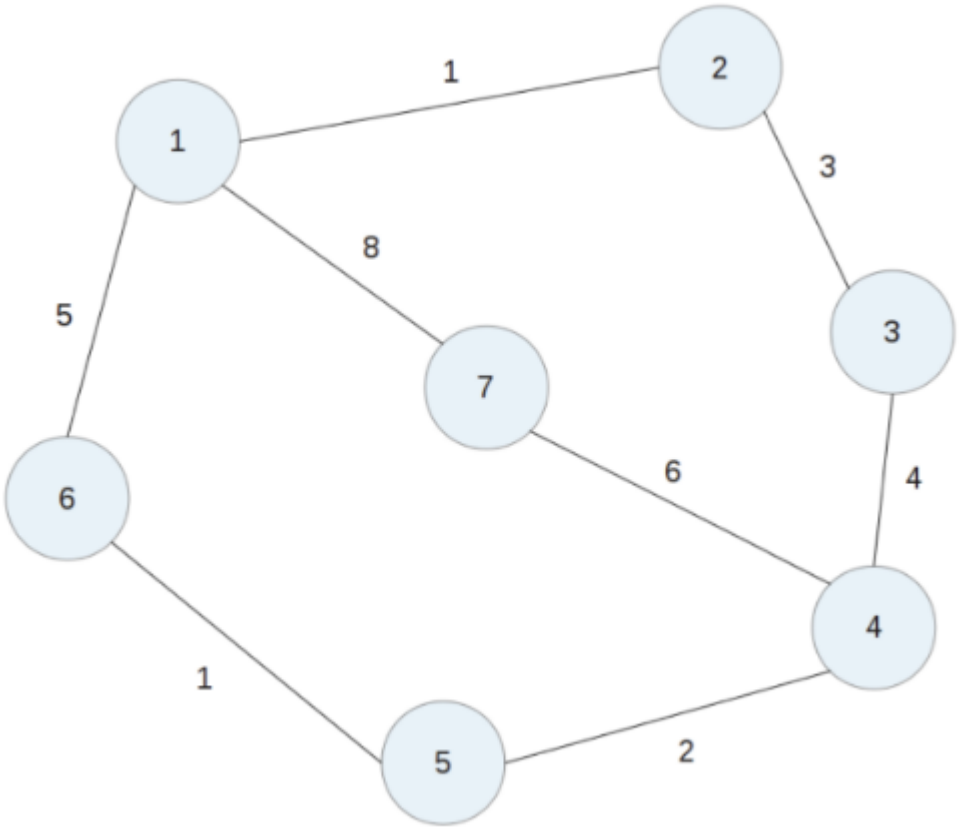

Figure 2: Seven-Node Network (figure 1 from specification [1])

Upon implementation of the configuration files for figure 2, a bash script that automatically starts each router in a new terminal was used for testing.

Upon starting up this network it converged to the right costs quickly, successfully demonstrating calculate_routes() had the correct logic. However, upon further testing some other issues became evident. After killing router two, the routers began successfully timing out and garbage collecting as the information that it was dead propagated throughout the network. However, due to the way we were handling invalid routes our triggered updates were being ignored as the cost was set to 16. This resulted in router one garbage collecting the route to router two correctly, but then router one received an advertisement from a further part of the network and created a new route. This meant router one calculated a route to router two using a different path even though it is its direct neighbor. To solve this problem, we needed to correctly handle the triggered updates. Currently, any incoming route with a cost of 16 is immediately ignored. Instead, when a route is received that has a cost of 16, its corresponding entry in the routing table is set with a cost 16 and as invalid. The garbage collection timer is then

started. Now when a router goes down the knowledge is spread quickly, and once the route is garbage collected it is never brought back by late route updates; only router reactivation.

The next bug occurred after router two was brought back online. The costs of its connections somehow had flipped. Ie. From figure 2, router one set its cost to router two as three instead of one, and router three set it to one instead of three. However, router two still had its costs in its routing table correctly. The network then converged correctly with these 'incorrect' costs, meaning no route calculation logic was incorrect. This exact behavior was replicated when killing and reviving router six. To try and diagnose the issue, router four was offline as it has three neighbors instead of two; unlike routers two and six. Reactivating router four after network convergence revealed that the link between routers two and four had a cost of two, the link between routers five and four had a cost of four, and between routers seven and four was four. The link cost of 6 originally between routers four and seven had not been shuffled around (as expected) but instead had entirely disappeared. To solve this, two changes were made. Firstly, the router's list of neighbors to a dictionary that also included costs for each neighbor learned from the configuration file. Then, a check in calculate_routes() was added before the distance vector calculation that checked if the sender was not in the routing table and if it was in the neighbors list. If those conditions are met, it means a neighbor has come back online and can be immediately injected back into the routing table with its associated neighbor cost. This solved the issue, and the network correctly converged back to the correct state.

The last bug featured in this initial testing stage is when a router goes offline and gets timed out by other Routers but comes back online before it gets garbage collected. Specifically, the routers never revalidate the revived Router's routes, but also don't garbage collect because its timers are being reset by because it keeps receiving valid routes. A simple fix was to modify the check described in the preceding paragraph. Specifically, checking if it's in the set of garbage timers to reinject it back into the routing table. It is then advertised by its neighbors correctly and the network converges.

The last test performed was to kill routers one and four to see if the network converges correctly. This was correct on the first try. Router seven had no information in its routing table, and routers six and five and two and three only had each other as pairs. Bringing router one back online resulted in the network converging correctly with routers four and seven having high costs to reach routers five. After reviving router four the network converged correctly again back to its initial state after bootup (barring the order of entries in the routing).

2. Unit Testing (Noah)

This section outlines all the unit testing performed on the RIP routing daemon. Several testing scripts were developed, each testing various aspects of the program, from basic functionality to specific routing scenarios that may occur in practice. Each script has its own section, and each test case is presented in a separate subsection.

2.1 Test Calculate Routes

2.1.1 Adding New Routes

This test checks that new routes are correctly added to the routing table. Initially, the router has no routes to routers four or five. Calling calculate_routes() with information about routes to these nodes should result in the routing table being correctly updated. Specifically, routers four and five should become reachable via routers two and three, respectively. In this case, all link costs are set to one, so the cost of each route should be two. This test passed, verifying that calculate_routes() correctly updates the routing table with new routes.

2.1.2 Update Existing Routes

This test case had two iterations.

Firstly, a router was initialized with a route to destination four via router two, with a high cost of six. Router two then advertised a route to destination four with a lower cost of three. This test checks that the router correctly updates its routing table with the new cost and that the route remains valid. The test passed, implying that the router can correctly update its routing table when a lower-cost route is received.

However, this did not test the case where a more expensive route is received. In such a case, the router should ignore the higher-cost route. Adding this scenario to the test resulted in a failure.

The issue was fixed by modifying calculate_routes() to check if the destination exists, if the next hop is the sender, and if the new cost is better than the old route before updating. After applying this fix, the test passed successfully.

2.2 Test Downed Routers

2.2.1 Test Router Down and Recovery Case

This test case creates a scenario where an adjacent router goes offline and later comes back online.

In particular, router four is initialized in a routes array and marked as reachable via router two. Similarly, router five is marked as reachable via router three. Calculate routes are called with these two routes as parameters. The expected outcome is that these routes will be correctly added into the routing table, a timeout scenario will then be induced for router two meaning the route to four should now be invalid. Finally, router two is brought back online and routes are recalculated. If router four appears in the routing table and is valid then the test passes.

Initially, this test failed because the route remained marked as invalid even after router two came back online and sent a recovery advertisement. Investigation revealed that, when calculating the cost for new routes, the router incorrectly used stale (invalid) cost information from the dead neighbor. This caused the cost of the recovery route to be inflated and ignored.

The bug was fixed by adding a check in calculate_routes() to ensure that the sender is valid before recalculating costs.

After applying the fix, the recovery route (to router four via router two) was correctly calculated and marked as valid in the routing table, and more expensive routes are ignored.

2.2.2 Split Horizon with Poison Reverse

This case tested the implementation of Split Horizon with Poison Reverse by creating a scenario where the router learned a route to a destination via router two and then attempted to construct an update packet to send back to router two. According to the Split Horizon with Poison Reverse rule, the route should be advertised with a cost of 16 to prevent routing loops. The test confirmed that the route was correctly poisoned, setting the cost to 16 in the update packet. This result verifies that the Split Horizon with Poison Reverse mechanism has been implemented correctly.

2.2.3 Garbage Collection

This case tests if invalid routes are removed from the routing table after garbage collection timers expire. By inducing a garbage collection timer for router four via router two and waiting for the garbage collection timer to expire, we would expect that the route would be removed from the routing table. This is exactly what happened, which verifies that garbage collection works as expected.

References

[1] Assignment 1 Specification, COSC364, University of Canterbury, 2025.

Appendices

Appendix A: Configuration File for router 4 in Figure 2

Output ports follow the structure of ‘port-cost-router_id’.

router-id 4
input-ports 11008 11009 11010
output-ports 11007-4-3 11011-2-5 11012-6-7