COSC364 RIP Assignment Report

COSC364-19S1 Computer Networking and Engineering

Isaac Daly - 85231671
Manu Hamblyn - 95140875

Contents

1 Percentage Contribution

Isaac Daly – 50%
Manu Hamblyn – 50%

2 Design and Implementation

2.1 Areas we're happy with

2.1.1 Static Typing

In order to improve productivity, we used Python's relatively new type annotation system to improve productivity and reduce bugs. Although there were still a few quirks since the type checkers are not yet mature, it was hugely beneficial.

2.1.2 Data structures, and Object Orientation

After significant consideration, we chose to use dictionaries (hash tables) to store the routing table. This allowed us to interact with entries inside the database in $O(1)$, since we often need to create, read, update, and delete entries. A custom `RoutingTable` class allowed abstractions such as operator overloading to be implemented.

Also, many times where normal Python tuples could be used, we decided to use subclass `NamedTuple`. They retain the performance advantage of Python tuples in comparison with normal Python classes, but provide easy, named lookup for properties. Another benefit is that they're immutable, preventing accidental mutation and similar bugs creeping in.

2.1.3 Modularity

Due to the modular design of the program, for certain areas we chose to follow a test-driven-development cycle. We could unit test small areas of the program and be confident that they would be stable over time.

2.1.4 System Testing

We came up with a comprehensive range of test scenarios for networks, shutting down, and restarting router instances. Through these scenarios a range of bugs were found and fixed. Often, these test cases made it easier to diagnose problems that we would have encountered in testing the assignment specified sample network. In other cases, edge cases unique to that network were discovered.

2.2 Areas for improvement

2.2.1 Automated Network to Routing Table Calculator

At some point, we discussed implementing automated routing tables. The idea was to write a program that would take existing configuration files and produce the converged tables for each router (rather than laboriously calculating them manually, as we did). Since this was "nice to have" and not a requirement, no further investigation besides an initial concept was carried out. While calculating the tables manually was great for cementing our understanding of the RIP protocol, trying to model larger and more complete networks was time consuming. An automated facility was sorely missed.

2.2.2 Documentation

As always, documentation during the writing of the source code could be lacking. Whilst reviewing code, there were a few cases where something along the lines of "Hang on, why…?" was uttered. When we eventually did figure out the reason for something, it was *then* documented.

2.3 Atomicity

The handling of every received packet (input processing) is atomic. It will result in the changes being immediately applied to the table. All timeout processing and triggered updates are handled on other threads, using the pool of threads defined by `ThreadPoolExecutor` from Python's `concurrent.futures` module. The receiving of packets itself blocks all threads.

Updates are not affected by input processing. This is because the table that the output processing iterates over is a shallow copy - the references to the entries in the table are the same as the entries that other threads have, but a deletion of an entry from the table will not affect output processing as it will retain the reference due to the shallow copy.

3 Testing

Two types of testing were performed:

- Unit or module testing
- System or scenario (often referred to as end-to-end) testing

Unit or module testing

Unit testing involves writing code to check the output of various functions given certain input conditions. When the function output behaves as expected, the functions are believed to be working correctly and can be integrated within the overall program.

System testing

Scenarios are created to check if the overall system behaves as expected. For this assignment, a "system" consists of several routing instances running concurrently with data being sent and received via sockets associated with each instance. The expected system behaviour is that the routing table in each routing instance will converge to known values when: a) all instances are active, b) when one or more instances are shutdown, and c) when one or more instances are re-started. For each test the converged routing tables have been calculated for situation a), b), and c).

Each test represents a known network configuration. The network configurations increase in complexity starting with a single router instance, through to multiple router instances, with the final test representing the example network in the assignment specification.

3.1 Unit Testing

3.1.1 Packet Construction

The following tests ensure that the routing table can be used to correctly construct packets for transmission to other routers.

3.1.1.1 `test_single_entry`

Purpose: Tests a routing table where the single entry does not match the given `router_id`.

Expected result: A single packet is produced, with the single entry from the routing table.

Succeeded: Yes

3.1.1.2 `test_two_entries_router_id_clash`

Purpose: Tests a routing table which has two entries, where one entry was learnt from the router that the packet is going to be sent to.

Expected result: A single packet is generated with a single entry, which contains the entry from the table which wasn't produced by the router the packet is being sent to.

Succeeded: Yes

3.1.1.3 `test_two_entries_infinity`

Purpose: Tests that a routing table with two entries, where one entry has a metric of infinity, produces a packet with the two entries inside.

Expected result: A single packet with the two entries from the routing table inside.

Succeeded: Yes

3.1.1.4 `test_multiple_packets`

Purpose: Tests that multiple packets are returned, when the number of routes is greater than 25.

Expected result: Two packets, where the first packet has 25 entries, and the second packet has 7 entries.

Succeeded: Yes

3.1.2 Packet Reading

The following tests ensure that received packets are read correctly.

3.1.2.1 `test_single_entry`

Purpose: Tests that a packet with a single entry can be read correctly.

Expected result: A single entry, where the address family identifier (AFI) is `AF_INET`, the router id is 1, and the metric is 1.

Succeeded: Yes

3.1.2.2 `test_multiple_entries`

Purpose: Tests that a packet with multiple entries (in this case 25 entries) can be correctly read.

Expected result: The packet has 25 entries, where the address family identifier (AFI) is `AF_INET`, the router id is 1, and the metric is 1.

Succeeded: Yes

3.1.3 Packet Validation

The following tests ensure that received packets contain valid data.

3.1.3.1 `test_valid_packet`

Purpose: Tests that a valid packet is acknowledged to be valid.

Expected result: `True` is returned

Succeeded: Yes

3.1.3.2 `test_sender_is_self`

Purpose: Tests that a packet which is sent from a router whose router id is the same as the receiving router is said to be invalid.

Expected result: Invalid neighbour causes `validate_packet` to return `False`

Succeeded: Yes

3.1.3.3 `test_invalid_command`

Purpose: Tests that a packet which has the `command` set to anything other than 2 is said to be invalid.

Expected result: Invalid `command` causes `validate_packet` to return `False`

Succeeded: Yes

3.1.3.4 `test_invalid_version`

Purpose: Tests that a packet which has the `version` set to anything other than 2 is said to be invalid.

Expected result: Invalid `version` causes `validate_packet` to return `False`

Succeeded: Yes

3.1.4 Entry Validation

The following tests ensure that received entries contain valid data.

3.1.4.1 `test_valid_entry`

Purpose: Tests that a valid entry is acknowledged to be valid.

Expected result: `True` is returned

Succeeded: Yes

3.1.4.2 `test_afi`

Purpose: Tests that an entry which has an AFI which is not `AF_INET` is invalid.

Expected result: Invalid AFI causes `validate_entry` to return `False`

Succeeded: Yes

3.1.5 Configuration file validation

The following tests ensure that the parsed configuration file contains valid data

3.1.5.1 `test_router_id`

Purpose: Tests that an entry which has a router id which is equal to the table's router id is invalid.

Expected result: Invalid `router_id` causes `validate_entry` to return `False`

Succeeded: Yes

3.1.5.2 `test_router_id_low`

Purpose: Tests that an entry which has a router id of less than 1 is invalid.

Expected result: Invalid `router_id` causes `validate_entry` to return `False`

Succeeded: Yes

3.1.5.3 `test_router_id_high`

Purpose: Tests that an entry which has a router id greater than 64000 is invalid.

Expected result: Invalid `router_id` causes `validate_entry` to return `False`

Succeeded: Yes

3.1.5.4 `test_metric_low`

Purpose: Tests that an entry which has a metric of less than 1 is invalid.

Expected result: Invalid `metric` results in `validate_entry` returning `False`

Succeeded: Yes

3.1.5.5 test_metric_high

Purpose: Tests that an entry which has a router id greater than 16 is invalid.

Expected result: Invalid metric results in validate_entry returning False

Succeeded: Yes

3.1.5.6 test_input_ports_low, test_output_ports_low

Purpose: Tests that an entry which has a port number less than 1024 is invalid.

Expected result: Invalid port causes validate_entry to return False

Succeeded: Yes

3.1.5.7 test_input_ports_high, test_output_ports_high

Purpose: Tests that an entry which has a port number greater than 64000 is invalid.

Expected result: Invalid port causes validate_entry to return False

Succeeded: Yes

3.1.5.8 test_input_ports_reuse, test_output_ports_reuse, test_output_ports_reuse_input

Purpose: Tests that port numbers are not re-used across input ports, across output ports, and an input ports is not reused as an output port.

Expected result: Reused port causes validate_entry to return False

Succeeded: Yes

3.1.5.9 test_timers_periodic_1, test_timers_gc

Purpose: Tests that timer ratios are valid.

Expected result: Invalid timer values causes validate_entry to return False

Succeeded: Yes

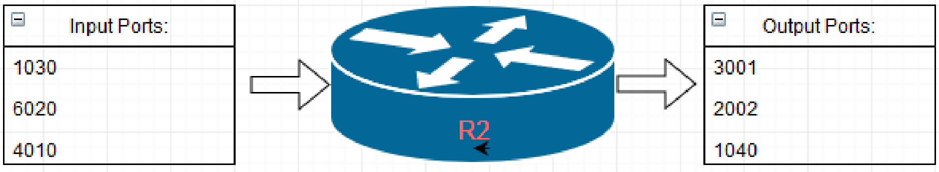
3.2 System Testing

The following sections describe the purpose of each system test, show the network diagram, and note what was learnt from each test. Where expected routing tables are included the format is `Router ID: Destination router ID(next hop router ID, metric).` For corresponding configuration files see Appendix 4.2

3.2.1 Test Scenario 1 – Single router instance(s)

Purpose 1a: Verify that a single router instance will start, read in a configuration file, open ports, opened, start timers, and that a routing table is shown.

Configuration file: R2.cfg



Initially, there was no explicit information showing that the router was sending information to the output ports. A debug option was added which displayed this information.

Purpose 1b: Verify that a single router instance will exit gracefully if it is supplied with a malformed configuration file.

Configuration file: R2a.cfg

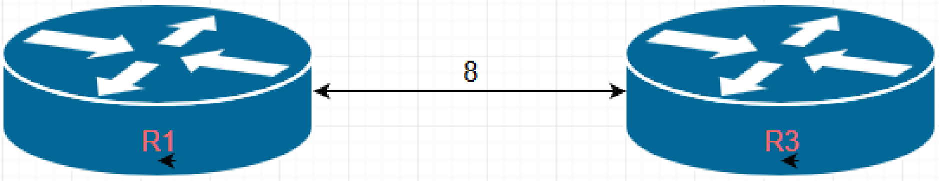
During final testing it was noted there was no system testing of configuration file errors hence this test was added.

3.2.2 Test Scenario 2 - Two peered routers

Purpose:

- Verify that for two router instances, each routing table converges to expected values.
- Verify the neighbour route is removed if the neighbour router instance is shut down.
- Verify that the tables converge again after the shutdown instance is re-started.

Configuration files: R1.cfg, R3.cfg.

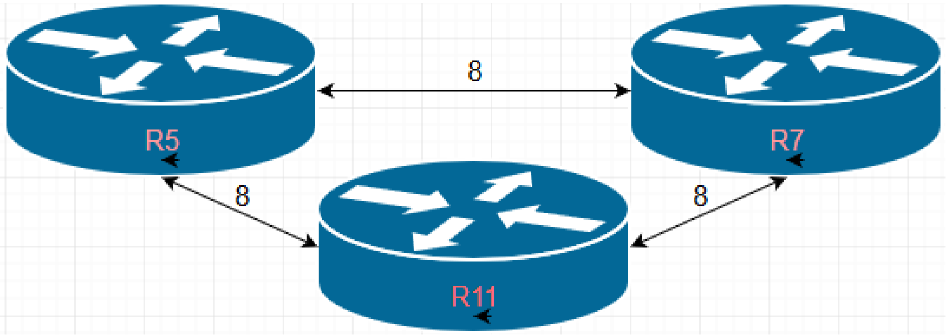


This test behaved as expected. (Each router had a routing table entry to the other with metric 8.)

3.2.3 Test Scenario 3 - Three routers in a ring, equal costs of eight.

Purpose: Verify that each routing table of three router instances in ring configuration will converge to expected values for a) all routers started, b) R7 shutdown, c) R7 re-started.

Configuration files: R5.cfg, R7.cfg, R11.cfg.



During initial testing the system failed to converge. A parser bug which caused R11 to name itself as R1 was fixed.

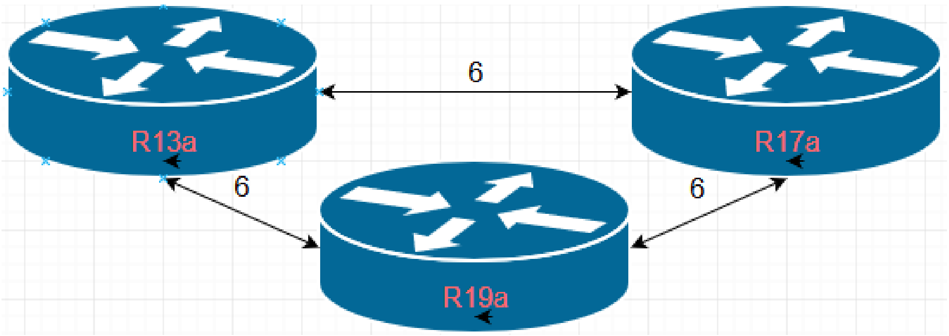
Retesting showed that the shutdown of R11 did not result in the removal of R11 from R7's routing table, although the route was marked as metric 16. The diagnosis was that R7 was still advertising R11 as being reachable and this was being added. Clearly the route should not have been added since the total metric would be 16, thus infinite and unreachable. The exact order of events was unclear, so Test Scenario 4 was created in order to observe routing table updates. Scenario 4 repeats the system configuration but with lower metrics such that if R7 was still advertising R11 as reachable, the total metric for R5 should be 12. Regardless of the route being added it should eventually be removed once R7's timeout timer triggers route deletion process. Repeating Test Scenario 3 many times showed that it was possible for this system to converge correctly, but not reliably.

3.2.4 Test Scenario 4 - Three routers in a ring, equal costs of six.

Purpose: Verify that each routing table of three router instances in ring configuration will converge to expected values for:

a) all routers started
b) R17a shutdown
c) R17a re-started.

Configuration files: R13a.cfg, R17a.cfg, R19a.cfg.



This was a new scenario which was added during final testing, to supplement debugging Test Scenario 3. This test showed that a path to the shutdown router with metric 12 was installed briefly before it was marked as 16, but not removed. Two bug fixes were applied, before the correct behaviour was reliably attained for both Test Scenario 3 and Test Scenario 4.

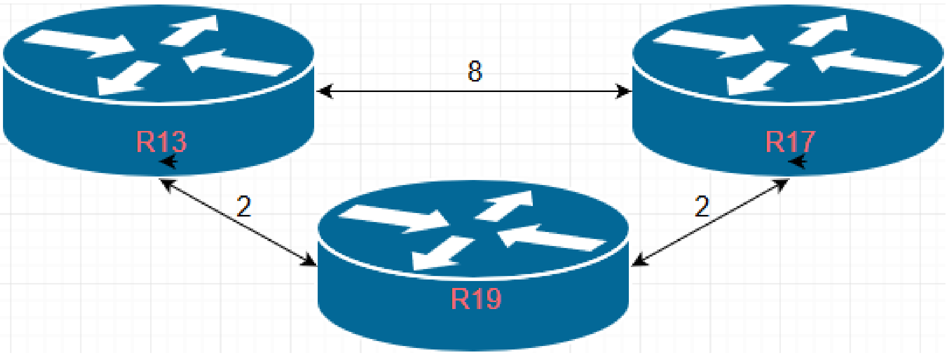
3.2.5 Test Scenario 5 - Three routers in a ring, unequal costs.

Purpose: Verify that a router instance will correctly update a route's metric if the network topology changes due to:

1. An increased route metric – when R19 is shutdown.

2. A decreased route metric – when R19 is restarted.

Configuration files: R13.cfg, R17.cfg, R19.cfg.



Expected routing tables:

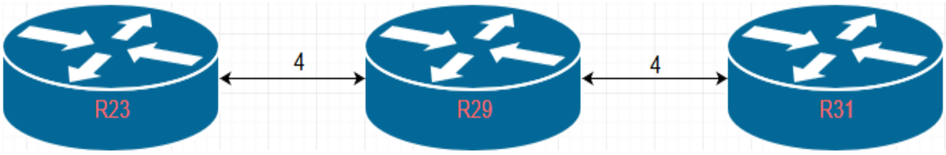
System start up	R19 shutdown	R19 restarted
R13: 17(19,4), 19(19,2)	R13: 17(17,8)	R13: 17(19,4), 19(19,2)
R17: 13(19,4), 19(19,2)	R17: 13(13,8)	R17: 13(19,4), 19(19,2)
R19: 13(13,2), 17(17,2)		R19: 13(13,2), 17(17,2)

During initial testing a "ping pong" effect was observed. Each router would correctly mark the dead router as having metric 16, start the garbage collection timer and delete the entry from the table. However, it would then subsequently re-add the route, learned from the neighbour router. It was apparent that additional debug information was needed.

3.2.6 Test Scenario 6 - Three routers, linear (or hub and spoke), equal costs

Purpose: Verify that a router instance will correctly remove/add routers from/to its routing table if no/new updates are received for non-directly connected routers.

Configuration files: R23.cfg, R29.cfg, R31.cfg.



Expected routing tables:

System start up	R31 shutdown	R31 restarted
R23: 29(29,4), 31(29,8)	R23: 29(29,4)	R23: 29(29,4), 31(29,8)
R29: 23(23,4), 31(31,4)	R29: 23(23,4)	R29: 23(23,4), 31(31,4)
R31: 23(29,8), 29(29,4)		R31: 23(29,8), 29(29,4)

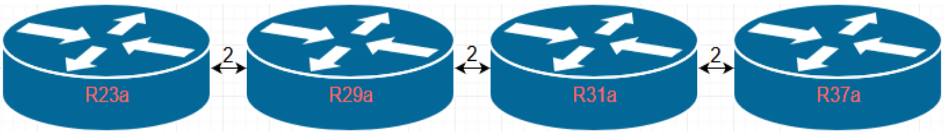
System start up	R29 shutdown	R29 restarted
R23: 29(29,4), 31(29,8)	R23: empty table	R23: 29(29,4), 31(29,8)
R29: 23(23,4), 31(31,4)		R29: 23(23,4), 31(31,4)
R31: 23(29,8), 29(29,4)	R31: empty table	R31: 23(29,8), 29(29,4)

During initial testing, an incorrect configuration file was discovered and fixed. During retesting the sequence of events by which R31 was removed from R23's routing table was not clear, however since convergence was reached in a reasonable timeframe, testing continued.

3.2.7 Test Scenario 7 - Four routers, linear, equal costs of two

Purpose: Verify that a router instance will correctly remove/add routers from/to its routing table if no/new updates are received for non-directly connected routers.

Configuration files: R23a.cfg, R29a.cfg, R31a.cfg, R37a.cfg.



Expected routing tables:

System start up	R37a shutdown	R37a restarted
R23: 29(29,2), 31(29,4), 37(29,6)	R23: 29(29,2), 31(29,4)	R23: 29(29,2), 31(29,4), 37(29,6)
R29: 23(23,2), 31(31,2), 37(31,4)	R29: 23(23,2), 31(31,2)	R29: 23(23,2), 31(31,2), 37(31,4)
R31: 23(29,4), 29(29,4), 37(37,2)	R31: 23(29,4), 29(29,4)	R31: 23(29,4), 29(29,4), 37(37,2)
R37: 23(31,6), 29(31,4), 31(31,2)		R37: 23(31,6), 29(31,4), 31(31,2)

System start up	R31a shutdown	R31a restarted
R23: 29(29,2), 31(29,4), 37(29,6)	R23: 29(29,2)	R23: 29(29,2), 31(29,4), 37(29,6)
R29: 23(23,2), 31(31,2), 37(31,4)	R29: 23(23,2)	R29: 23(23,2), 31(31,2), 37(31,4)
R31: 23(29,4), 29(29,4), 37(37,2)		R31: 23(29,4), 29(29,4), 37(37,2)
R37: 23(31,6), 29(31,4), 31(31,2)	R37: empty table	R37: 23(31,6), 29(31,4), 31(31,2)

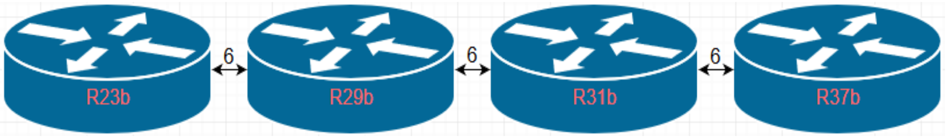
This was a new scenario added for final testing due to inconsistencies in Test Scenario 6. It was observed that R29a and R23a removed their table entries for R37a due to timeout only. Investigation showed triggered updates were sent from R31a (and R29a) meaning triggered updates were not being processed correctly in both R23 and R29a.

3.2.8 Test Scenario 8 - Four routers, linear, equal costs of six

Purpose:

- Verify that a router instance will correctly remove/add routers from/to its routing table if no/new updates are received for non-directly connected routers.
- Verify route metric changes and triggered updates are being processed correctly, and do not affect each other.

Configuration files: R23b.cfg, R29b.cfg, R31b.cfg, R37b.cfg.



Expected routing tables:

System start up	R37b shutdown	R37b restarted
R23: 29(29,6), 31(29,12)	R23: 29(29,6), 31(29,12)	R23: 29(29,6), 31(29,12)
R29: 23(23,6), 31(31,6), 37(31,12)	R29: 23(23,6), 31(31,6)	R29: 23(23,6), 31(31,6), 37(31,12)
R31: 23(29,12), 29(29,6), 37(37,6)	R31: 23(29,12), 29(29,6)	R31: 23(29,12), 29(29,6), 37(37,6)
R37: 29(31,12), 31(31,6)		R37: 29(31,12), 31(31,6)

System start up	R31b shutdown	R31b restarted
R23: 29(29,6), 31(29,12)	R23: 29(29,6)	R23: 29(29,6), 31(29,12)
R29: 23(23,6), 31(31,6), 37(31,12)	R29: 23(23,6)	R29: 23(23,6), 31(31,6), 37(31,12)
R31: 23(29,12), 29(29,6), 37(37,6)		R31: 23(29,12), 29(29,6), 37(37,6)
R37: 29(31,12), 31(31,6)	R37: empty table	R37: 29(31,12), 31(31,6)

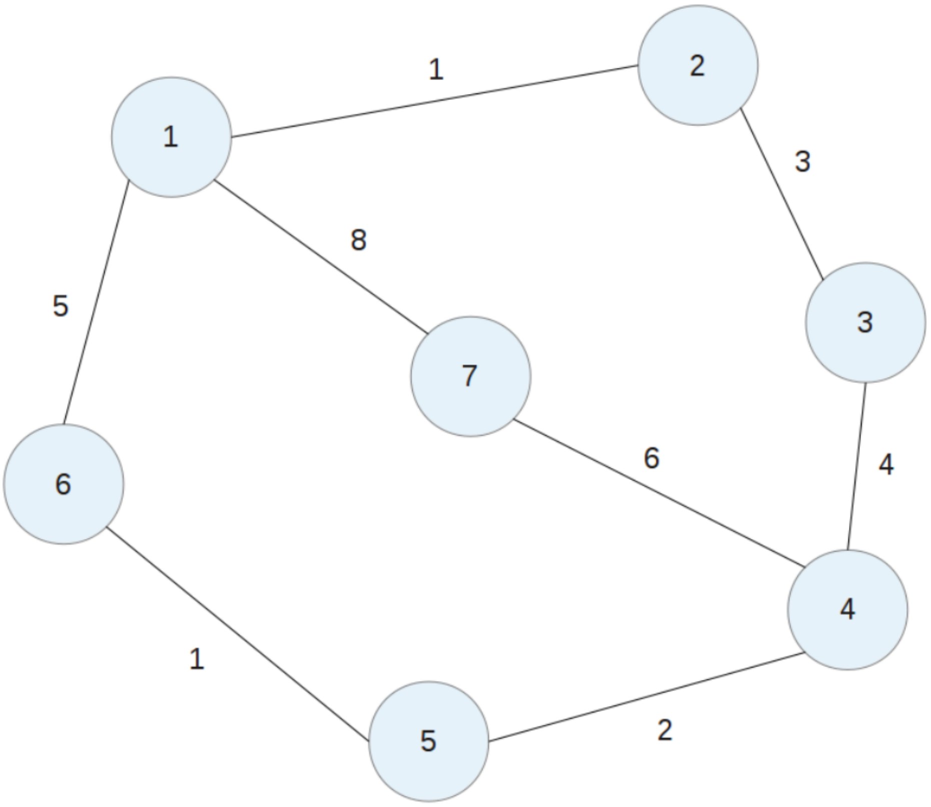
This was a new scenario created for final testing. It behaved as expected. Each end router did not have an entry for the other end router since the total metric for the end routers to reach each other was greater than 16.

3.2.9 Test Scenario 9 - Assignment specification - Seven routers, partial mesh, various costs

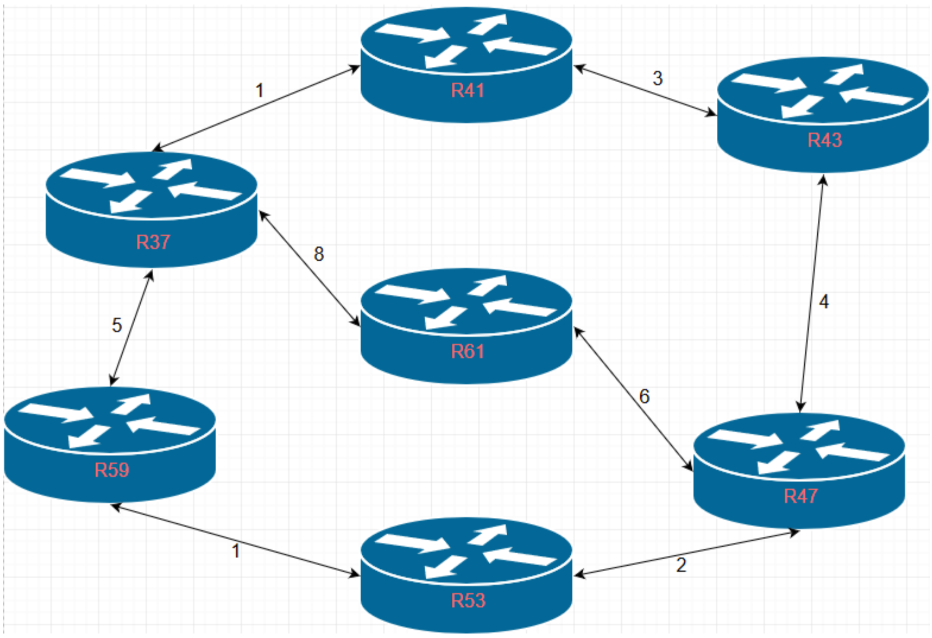
Purpose: Verify that for each router instance its routing table, in a partial mesh network configuration, will converge to the correct next hop and metric values for various router(s) started, shutdown and re-started.

Configuration files: R37.cfg, R41.cfg, R43.cfg, R47.cfg, R53.cfg, R59.cfg, R61.cfg.

System diagram in node form:



System diagram in router instance form:



Expected routing tables:

System start up	**notes**
R37: 41(41,1), 43(41,4), 47(41,8), 53(59,6), 59(59,5), 61(61,8)	R37: … 47(41,8) = 47(59,8) …
R41: 37(37,1), 43(43,3), 47(43,7), 53(37,7), 59(37,6), 61(37,9)	
R43: 37(41,4), 41(41,3), 47(47,4), 53(47,6), 59(47,7), 61(47,10)	
R47: 37(43,8), 41(43,7), 43(43,4), 53(53,2), 59(53,3), 61(61,6)	R47: 37(43,8) = 37(53,8) …
R53: 37(59,6), 41(59,7), 43(47,6), 47(47,2), 59(59,1), 61(47,8)	
R59: 37(37,5), 41(37,6), 43(53,7), 47(53,3), 53(53,1), 61(53,9)	
R61: 37(37,8), 41(37,9), 43(47,10), 47(47,6), 53(47,8), 59(47,8)	
Shutdown R41	
R37: 41(59,2), 47(59,8), 53(59,6), 59(59,5), 61(61,8)	
R43: 37(47,12), 47(47,4), 53(47,6), 59(47,7), 61(47,10)	
R47: 37(53,8), 43(43,4), 53(53,2), 59(53,3), 61(61,6)	
R53: 37(59,6), 43(47,6), 47(47,2), 59(59,1), 61(47,8)	
R59: 37(37,5), 43(53,7), 47(53,3), 53(53,1), 61(53,9)	
R61: 37(37,8), 43(47,10), 47(47,6), 53(47,8), 59(47,9)	
Restart R41	
Tables converge as per system start up	
Shutdown R59	
R37: 41(41,1), 43(41,4), 47(41,8), 53(41,10), 61(61,8)	
R41: 37(37,1), 43(43,3), 47(43,7), 53(43,9), 61(37,9)	
R43: 37(41,4), 41(41,3), 47(47,6), 53(47,6), 61(47,10)	
R47: 37(43,8), 41(43,7), 43(43,4), 53(53,2), 61(61,6)	
R53: 37(47,10), 41(49,9), 43(47,6), 47(47,2), 61(47,8)	
R61: 37(37,8), 41(37,9), 43(47,10), 47(47,6), 53(47,8)	
Restart R41	
Tables converge as per system start up	
Shutdown R41 and R59	Full tables not shown
R37: no route R41, R43, R53, R59	
R43: no routes to R37, R41, R59	

R53: no routes to R37, R41, R59	
Restart R41 and R59	
Tables converge as per system start up	

Initial test:

With all the routers started the routing tables did not converge. Both flags and metrics continuously changed. Each router was sending updates to every other router it was aware of, via the port the router learned from. However, each router only needed to send one update with all the routes it was aware of to each neighbour, whilst considering split horizon and poison reverse. This should have consisted of one update clockwise, one update anticlockwise, and one update to middle mesh if attached. The central node (middle mesh) should have one update in each direction.

Final test:

At system start-up both next hop and metric changes could be observed in routers until they reached their converged value. On shutdown of an instance or instances, once the timeout timer had expired triggered updates could be observed propagating through the routing instances and across the network. On restart of an instance or instances, metric updates could be observed propagating through the routers and across the network. Numerous combinations of routers were shut down and restarted before the test was considered a success. Note that for brevity not all test routing tables have been shown.

4 Appendices

4.1 Source Code

Modules are listed in alphabetical order. The entry point for the program is Error! Reference source not found.. To start the program, go to the root of the folder which contains the following modules, and type:

```
$ python3 router.py <PATH TO CONFIG FILE> [DEBUG]
```

DEBUG should only be used if running the program in debug mode.

4.1.1 input_processing.py

```
from datetime import datetime, timedelta
from select import select
from socket import AF_INET, socket
from typing import List, Tuple

from output_processing import deletion_process, pool
from packet import ResponseEntry, ResponsePacket, read_packet, validate_packet
from routeentry import RouteEntry
from routerbase import logger
from routingtable import RoutingTable
from validate_data import INFINITY, MAX_ID, MAX_METRIC, MIN_ID, MIN_METRIC


def validate_entry(table: RoutingTable, packet_entry: ResponseEntry) -> bool:
    """Validates an individual router entry."""

    if packet_entry.router_id == table.router_id:
        return False

    # Checks the entry's AFI value
    if packet_entry.afi != AF_INET:
        logger(
            f"The value {packet_entry.afi} for AFI does not match the "
            f"expected value of AF_INET = {AF_INET}.",
            is_debug=True,
        )
        return False

    # Checks for the router_id
    if packet_entry.router_id < MIN_ID or packet_entry.router_id > MAX_ID:
        logger(
            f"The entry's router id of {packet_entry.router_id} should be an "
            f"integer between {MIN_ID} and {MAX_ID}, inclusive.",
            is_debug=True,
        )
        return False

    # Checks that the entry's metric is between the expected minimum and
    # maximum metric.
    if packet_entry.metric < MIN_METRIC or packet_entry.metric > MAX_METRIC:
        logger(
            f"The entry's metric of {packet_entry.metric} was not between the "
            f"expected range of {MIN_METRIC} and {MAX_METRIC}, inclusive.",
            is_debug=True,
        )
        return False
    # If all the checks pass, then return `True`
    return True


def add_route(
    table: RoutingTable,
    packet_entry: ResponseEntry,
    new_metric: int,
    next_hop: int,
    sock: socket,
):
    """Adds a newly learned route to the routing table."""
    if new_metric == INFINITY:
        # Don't add routes with costs of infinity to the database - there's
        # no point adding routes that you cannot reach.
        return

    logger(f"Adding route with cost of {new_metric}", is_debug=True)

    actual_port = table.config_table[next_hop].port
    entry = RouteEntry(actual_port, new_metric, table.timeout_delta, next_hop)
    entry.gc_time = None
    entry.flag = True
    table.add_route(packet_entry.router_id, entry)

    # NOTE: As per the assignment spec, "implement triggered updates when
    # routes become invalid  (i.e. when a router sets the routes metric to
    # 16 <INFINITY> for whatever reason, compare end of page 24 and
    # beginning of 25 in [1]), not for other metric updates or new routes".
    # Thus, the following line is commented out, and the success lines added.
    # send_responses(table, sock)


def adopt_route(
    table: RoutingTable,
    table_entry: RouteEntry,
    new_metric: int,
    next_hop: int,
    sock: socket,
    router_id: int,
):
    """
    Adopts the newly received route, and updates the existing routing table
    entry.
    """
    logger(
        f"Adopting route with metric of {new_metric}, router_id {router_id}",
        is_debug=True,
    )

    table_entry.metric = new_metric
    table_entry.next_hop = next_hop
    table_entry.flag = True
    table_entry.next_hop = next_hop
    table_entry.triggered_update_time = None

    # NOTE: As per the assignment spec, "implement triggered updates when
    # routes become invalid  (i.e. when a router sets the routes metric to
    # 16 <INFINITY> for whatever reason, compare end of page 24 and
    # beginning of 25 in [1]), not for other metric updates or new routes".
    # Thus, the following line is commented out.
```

```python
    # send_responses(table, sock)

    if new_metric == INFINITY:
        # The following will eventually cause a triggered update.
        logger("About to go into deletion process", is_debug=True)
        pool.submit(deletion_process, table, sock, router_id)
    else:
        table_entry.update_timeout_time(table.timeout_delta)


def update_table(
    table: RoutingTable,
    packet_entry: ResponseEntry,
    new_metric: int,
    next_hop: int,
    sock: socket,
):
    """
    Goes through the process of updating the routing table with the new route,
    if applicable.
    """
    logger(
        f"Updating the table with router_id {packet_entry.router_id}",
        is_debug=True,
    )
    table_entry: RouteEntry = table[packet_entry.router_id]

    if next_hop == table_entry.next_hop and new_metric != INFINITY:
        table_entry.update_timeout_time(table.timeout_delta)

    if (
        next_hop == table_entry.next_hop and new_metric != table_entry.metric
    ) or new_metric < table_entry.metric:
        adopt_route(
            table,
            table_entry,
            new_metric,
            next_hop,
            sock,
            packet_entry.router_id,
        )
    elif new_metric == INFINITY:
        # nothing happens if the entry's existing metric is `INFINITY`
        if table_entry.metric != INFINITY:
            pool.submit(deletion_process, table, packet_entry.router_id)
    elif new_metric == table_entry.metric and next_hop != table_entry.next_hop:
        # Adding a check for `next_hop` means that the entry will not be
        # updated if its the same as the old entry.

        # If the timeout for the existing route is at least halfway to the
        # expiration point, switch to the new route.
        time_diff: timedelta = table_entry.timeout_time - datetime.now()
        half_time = table.timeout_delta / 2
        if time_diff.seconds >= half_time:
            adopt_route(
                table,
                table_entry,
                new_metric,
                next_hop,
                sock,
                packet_entry.router_id,
            )
    else:
        # The packet_entry is no better than the current route
        pass


def process_entry(
    table: RoutingTable,
    packet_entry: ResponseEntry,
    packet: ResponsePacket,
    port: int,
    sock: socket,
):
    """Processes a single entry from a received packet."""
    logger("Processing an entry", is_debug=True)
    if not validate_entry(table, packet_entry):
        # Ignores invalid entries
        return

    # Update the metric
    new_metric = packet_entry.metric + table[packet.sender_router_id].metric
    if new_metric > INFINITY:
        new_metric = INFINITY

    if packet_entry.router_id in table:
        update_table(
            table, packet_entry, new_metric, packet.sender_router_id, sock
        )
    else:
        add_route(
            table, packet_entry, new_metric, packet.sender_router_id, sock
        )


def get_packets(
    sockets: List[socket]
) -> List[Tuple[ResponsePacket, int, socket]]:
    """
    Gets a tuple of the received packets from the input sockets, and their
    associated port numbers and sockets.

    Returns a list of tuples, where each tuple is (ResponsePacket, port, sock).
    """
    read: List[socket]
    read, _, _ = select(sockets, [], [], 1)

    packets = []

    for sock in read:
        _, port = sock.getsockname()
        # The buffer size is bigger than the maximum packet size.
        raw_packet, client_address = sock.recvfrom(1024)
        packet = read_packet(raw_packet)
        packets.append((packet, port, sock))
        logger(
            f"Received packet from router_id: {packet.sender_router_id} | "
            f"input port {port}",
            is_debug=True,
        )

    return packets
```

```python
def add_discovered(table: RoutingTable, packet: ResponsePacket, sock: socket):
    """
    Adds entries discovered implicitly from the packet itself into the routing
    table.
    """
    if packet.sender_router_id not in table.config_table:
        logger(
            f"router_id {packet.sender_router_id} is not in "
            "the config file.",
            is_debug=True,
        )
        return

    metric = table.config_table[packet.sender_router_id].cost
    # The idea is generally the same as adding a new route into the table, even
    # if updating an entry, because we'll want to completely wipe the entry and
    # timers.
    fake_packet_entry = ResponseEntry(AF_INET, packet.sender_router_id, metric)
    add_route(table, fake_packet_entry, metric, packet.sender_router_id, sock)


def input_processing(table: RoutingTable, sockets: List[socket]):
    """
    The processing is the same, no matter why the Response was generated.
    """
    for packet, port, sock in get_packets(sockets):
        router_id = packet.sender_router_id
        if validate_packet(table, packet):
            for entry in packet.entries:
                process_entry(table, entry, packet, port, sock)

        # The following adds entries if the packet sender's `router_id` is
        # inside the config file.
        if router_id in table.config_table:
            if router_id not in table:
                # If the sender's `router_id` isn't in the routing table, add
                # it.
                add_discovered(table, packet, sock)
            elif table.config_table[router_id].cost <= table[router_id].metric:
                # Actually updates the entry. The underlying idea is the same
                # as adding a newly discovered route.
                add_discovered(table, packet, sock)
            elif (
                table[router_id].next_hop not in table
                or table[table[router_id].next_hop].metric == INFINITY
            ):
                # Let the next_hop for the router be R. If R isn't in the
                # routing table, or if R has a metric of infinity, update
                # the entry with the information inferred from the packet.
                add_discovered(table, packet, sock)
            else:
                # There's no changes here, thus update the timeout timer.
                table[router_id].update_timeout_time(table.timeout_delta)

    logger(str(table))
```

4.1.2 output_processing.py

```python
from datetime import datetime
from socket import socket
from typing import Optional

from packet import construct_packets
from routeentry import RouteEntry
from routerbase import logger, pool
from routingtable import RoutingTable
from validate_data import INFINITY


def send_response(sock: socket, port: int, packet: bytearray):
    """
    Sends a `Response` packet to the given `router_id`.
    """
    # Ignore the pyright error - it fails to resolve bytearray and bytes, when
    # it really should.
    sock.sendto(packet, ("localhost", port))


def _send_responses(table: RoutingTable, sock: socket, clear_flags=False):
    logger(str(table))
    for router_id in table:
        if router_id in table.config_table:
            packets = construct_packets(table, router_id)
            port = table.config_table[router_id].port
            for packet in packets:
                logger(
                    f"Sending to router_id {router_id} port {port} ",
                    is_debug=True,
                )
                send_response(sock, port, packet)

    if clear_flags:
        for router_id in table:
            entry: RouteEntry = table[router_id]
            entry.flag = False


def send_responses(table: RoutingTable, sock: socket, clear_flags=False):
    """
    Sends unsolicited `Response` messages containing the entire routing
    table to every neighbouring router.
    """
    pool.submit(_send_responses, table, sock, clear_flags)


def timeout_processing(table: RoutingTable, entry: RouteEntry, sock: socket):
    """Starts processing for the timeout timer."""
    logger("About to set gc time", is_debug=True)
    entry.set_garbage_collection_time(table.gc_delta)
    entry.metric = INFINITY
    entry.flag = True

    now = datetime.now()
    can_update = table.set_triggered_update_time(now)

    # Suppresses the update if another triggered update has been sent
    if can_update:
        # Send triggered updates
        send_responses(table, sock, True)
```

```
def gc_processing(
    table: RoutingTable, router_id: int, entry: RouteEntry, now: datetime
):
    """Starts processing for the garbage collection timer."""
    if (
        router_id in table
        and entry.gc_time is not None
        and entry.gc_time <= now
    ):
        logger(f"Deleting router id {router_id}", is_debug=True)
        del table[router_id]


def deletion_process(
    table: RoutingTable, sock: socket, new_infinite_id: Optional[int] = None
):
    """
    Handles the timeout and garbage collection timer processing for the
    routing table.
    """
    logger("In deletion process", is_debug=True)
    logger(f"New infinite id is {new_infinite_id}", is_debug=True)
    now = datetime.now()  # Only calling it once minimises system time
    for router_id in table:
        logger(f"Current router id is {router_id}", is_debug=True)
        entry: RouteEntry = table[router_id]
        if entry.gc_time is not None:
            logger(
                f"About to go into GC processing for router {router_id}",
                is_debug=True,
            )
            gc_processing(table, router_id, entry, now)
        elif entry.timeout_time <= now or router_id == new_infinite_id:
            logger(
                f"About to go into timeout processing for router {router_id}",
                is_debug=True,
            )
            pool.submit(timeout_processing, table, entry, sock)
```

4.1.3 packet.py

```
from socket import AF_INET
from typing import List, NamedTuple, Union

from routeentry import RouteEntry
from routerbase import logger
from routingtable import RoutingTable
from validate_data import INFINITY

MAX_ENTRIES = 25
ENTRY_LEN = 20
HEADER_LEN = 4
RIP_PACKET_COMMAND = 2
RIP_VERSION_NUMBER = 2


class ResponseEntry(NamedTuple):
    afi: int
    router_id: int
    metric: int
```

```
class ResponsePacket(NamedTuple):
    command: int
    version: int
    sender_router_id: int
    entries: List[ResponseEntry]


def get_next_packet_entries(table: RoutingTable, dest_router_id: int):
    """
    Gets the entries from the routing table, which can be sent to the given
    `router_id`. Entries which have a metric of less than infinity, or have a
    flag, and are not learned from the router to whom the packets are going to
    be sent to are added to the yielded list of entries.

    Keyword arguments:
    table -- The routing table, containing all of the entries.
    router_id -- The router the packet is being sent to.
    """
    entries = []
    for current_router_id in table:
        route: RouteEntry = table[current_router_id]
        if route.next_hop == dest_router_id:
            route = route.shallow_copy()
            route.metric = INFINITY

        entries.append((current_router_id, route))
        if len(entries) == MAX_ENTRIES:
            yield entries
            entries = []
    yield entries


def _construct_packet_header(packet: bytearray, table) -> None:
    """
    Modifies the packet's header.

    Keyword arguments:
    packet -- The packet who will have its header populated.
    table -- The table from whom the packet is going to be sent from.
    """
    # the following are implicitly converted to bytes
    packet[0] = RIP_PACKET_COMMAND
    packet[1] = RIP_VERSION_NUMBER
    packet[2:4] = table.router_id.to_bytes(2, "big")


def _construct_packet(table: RoutingTable, entries) -> bytearray:
    """
    Constructs an individual packet, with up to 25 entries inside, with the
    given table entries.
    """
    packet = bytearray(HEADER_LEN + len(entries) * ENTRY_LEN)
    _construct_packet_header(packet, table)
    current_index = 4

    for (destination_router_id, entry) in entries:
        packet[current_index : current_index + 2] = AF_INET.to_bytes(2, "big")
        packet[
            current_index + 4 : current_index + 8
        ] = destination_router_id.to_bytes(4, "big")
        packet[
            current_index + 16 : current_index + ENTRY_LEN
        ] = entry.metric.to_bytes(4, "big")
        current_index += ENTRY_LEN

    return packet


def construct_packets(table: RoutingTable, router_id: int) -> List[bytearray]:
    """Constructs packets to send to a `router_id`, from the routing table."""
    packets: List[bytearray] = []

    for entries in get_next_packet_entries(table, router_id):
        packets.append(_construct_packet(table, entries))

    return packets


def _read_packet_entry(packet: Union[bytearray, bytes], start_index: int) -> ResponseEntry:
    """
    Returns the properties of a single RIP entry inside a RIP response packet.
    """
    afi = int.from_bytes(packet[start_index : start_index + 2], byteorder="big")
    router_id = int.from_bytes(
        packet[start_index + 4 : start_index + 8], byteorder="big"
    )
    metric = int.from_bytes(
        packet[start_index + 16 : start_index + 20], byteorder="big"
    )
    return ResponseEntry(afi, router_id, metric)


def read_packet(packet: Union[bytearray, bytes]) -> ResponsePacket:
    """Returns the properties of the received RIP response packet."""
    command: int = packet[0]
    version: int = packet[1]
    sender_router_id: int = int.from_bytes(packet[2:4], byteorder="big")

    entries = []
    start_index = HEADER_LEN

    while start_index < len(packet):
        end_index = start_index + ENTRY_LEN
        if end_index <= len(packet) and len(entries) < MAX_ENTRIES:
            entries.append(_read_packet_entry(packet, start_index))
        else:
            return ResponsePacket(command, version, sender_router_id, entries)
        start_index = end_index
    return ResponsePacket(command, version, sender_router_id, entries)


def validate_packet(table: RoutingTable, packet: ResponsePacket):
    """
    Returns a Boolean indicating whether the given packet is valid.
    """
    # Checks whether the router_id belongs to the router itself
    if table.router_id == packet.sender_router_id:
        logger(
            f"The packet's router_id of {packet.sender_router_id} illegally "
            + "matches the router_id of this router.",
            is_debug=True,
        )
        return False

    # Checks whether the router_id is from a valid neighbour
    if packet.sender_router_id not in table.neighbours():
        logger(
            f"Packet received from router_id {packet.sender_router_id}, which "
            "is not a neighbour of this router.",
            is_debug=True,
        )
        logger(
            f"Current neighbours of this router {table.router_id} are "
            f"{[i for i in table.neighbours()]}.",
            is_debug=True,
        )
        return False

    # Checks that the command is valid
    if packet.command != 2:
        logger(
            f"The packet has a command value of {packet.command}, instead of 2.",
            is_debug=True,
        )
        return False

    if packet.version != 2:
        logger(
            f"The packet has a version value of {packet.version}, instead of 2.",
            is_debug=True,
        )
        return False

    return True
```

4.1.4 poc_parser_v03.py

```
################################################################################
# 2019_03_06
# Manu Hamblyn
# 95140875
# my_starter.py
# code to parse a text configuration file and put contents into a Dict
#
# v0_1
# Very simple test program to get to grips with Python text parsing
# Initial test code derived from:
# https://stackabuse.com/read-a-file-line-by-line-in-python/
# and
# https://stackoverflow.com/questions/3277503/how-to-read-a-file-line-by-line-into-a-list
# and
# https://stackoverflow.com/questions/1706198/python-how-to-ignore-comment-lines-when-reading-
in-a-file
#
# v0_2
# getting required data need to remove newline characters and put data into
#  variables (ID and timers,
#  list for input, table/dict for output)
# https://stackoverflow.com/questions/6213063/python-read-next
#
# v0_3
# poc_parser.py
# Guidance, support and sample code provided by jps111.
# Reads complete config file since size is small.
# Uses join and split operations on lines.
#
# v04: 18 April 2019
# Fixed some linting errors
#
# v05: 19 April 2019
# Bug fix: Changed id = int(cost) to id = int(id)
#
# v06: 20 April 2019
# Made pyright happy by changing variable names, thus fixing typing inference
################################################################################

from routerbase import logger


def read_config(filename):
    f = open(filename)
    router_id = None
    input_ports = None
    output_ports = None
    timers = None

    # read whole line
    for line in f.readlines():  # iterate through the lines in file
        option = line.split(" ")[
            0
        ].strip()  # if line starts with whitespce strip line
        values = " ".join(
            line.split(" ")[1:]
        )  # join lines togther to just have text
        if option == "router-id":
            if router_id is not None:
                logger("multiple router-id lines")
                break

            router_id = int(values.strip())

        elif option == "input-ports":
            # check if we have already set input-ports
            if input_ports is not None:
                logger("multiple input-ports lines")
                break

            input_ports = []
            parts = values.split(
                ","
            )  # seperate the line into parts, using comma
            for port_str in parts:  # iterate through the parts
                port = int(port_str.strip())  # remove whitespace
                input_ports.append(port)  # append to list

        elif option == "output-ports":
            if output_ports is not None:
                logger("multiple output-ports lines")
                break

            output_ports = []
            parts = values.split(",")
            for part in parts:
                part = part.strip()
                (port_str, cost, curr_id) = part.split(
                    "-"
                )  # further split by hyphen
                port_str = int(port_str)
                cost = int(cost)
                curr_id = int(curr_id)  # fixed error here
                output_ports.append((port_str, cost, curr_id))
        elif option == "timers":
            if timers is not None:
                logger("multiple timers lines")
                break

            timers = []
            parts = values.split(",")
            for port_str in parts:
                port = int(port_str.strip())
                timers.append(port)
    f.close()
    return (router_id, input_ports, output_ports, timers)


if __name__ == "__main__":

    try:
        (router_id, input_ports, output_ports, timers) = read_config("R2.cfg")
        logger("router-id", router_id)
        logger("input-ports", input_ports)
        logger("output-ports", output_ports)
        logger("timers", timers)
    except (Exception):
        logger("An error occurred")
```

4.1.5 port_closer.py

```
#########################################################
#
# A function to close ports / sockets
#
# Version 01: 17 April 2019
#   First pass
#
# Version 02:
#   Return codes:
#   1 => error closing input ports
#   2 => error closing output ports
#   3 => error closing input and output ports
#
# Version 03: 18 April 2019
#   Only needs to work with list of input sockets
#
# Version 03a: 19 April 2019
#   Removed unnecessary imports
#########################################################

from routerbase import logger


def port_closer(socket_list):

    for a_socket in socket_list:
        try:
            name = a_socket.getsockname()
            a_socket.close()

            logger("Closed socket / port, ", a_socket, "/", name)

        except (Exception):
            return 1

    return 0


if __name__ == "__main__":
    from port_opener import port_opener

    # open input ports
    input_ports = (3001, 4001, 5001)
    socket_list = port_opener(input_ports)
    logger(socket_list)

    # input ports
    port_closer(socket_list)
```

4.1.6 port_opener.py

```
#########################################################
#
# A function to open / bind ports to sockets
#
# Version 01: March 2019
#   First pass
#
# Version 02: 17 April 2019
#   change to match output port with destination router
#   ID
#
# Version 03:
#   updated to take in the complete "output_list" tuple
#   check if id is null:
#   YES => return tuple of sockets (input ports)
#   NO => return tuple of sockets (output ports) and
#   destination router ID
#
# Version 04:
#   need a try catch exception handler, in case a port
#   did not open/bind which returns an error code so
#   main can close everything and exit
#
# Version 05: 18 April 2019
#   Removed the output ports - as these are not actually
#   needed, as we send routing info to the input port of
#   the destination router
#
#########################################################

import socket
from port_closer import port_closer

from routerbase import logger


def port_opener(input_ports):
    socket_list = []

    a_port = None

    try:
        for a_port in input_ports:

            new_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            # new_socket.setblocking(0)
            new_socket.bind(("localhost", a_port))

            logger("Opened socket / port, ", new_socket, "/", a_port)

            socket_list.append(new_socket)

        return socket_list

    except OSError:
        logger(f"The port {a_port} is already in use. Please pick another one.")
        port_closer(socket_list)
        return None


if __name__ == "__main__":

    # input ports
    input_ports = (3001, 4001, 5001)
    socket_list = port_opener(input_ports)
    logger(socket_list)
```

4.1.7 routeentry.py

```
from datetime import datetime, timedelta
from typing import Optional, Union


class RouteEntry:
    """
    Entry for a route inside the RIP routing table.

    Instance variables:

    flag -- Set to `True` to indicate that the entry has changed.

    port -- The port that this `RouteEntry` is going to be sent to.

    metric -- The cost in total.

    next_hop -- The router-id of the router from whom this route was
    learned from. `-1` if the router wasn't learned from anyone (i.e. learned
    from the config file on startup).

    timeout_time -- The delta for the time at which the timeout occurs, and the
    deletion process for this `RouteEntry` starts. This is not stored.

    gc_time --- The time after which this `RouteEntry` should be deleted from
    `RoutingTable`.
    """

    flag = False
    port: int
    metric: int
    next_hop: int

    timeout_time: datetime
    gc_time: Optional[datetime] = None

    def __init__(
        self,
        port: int,
        metric: int,
        timeout_time: Union[int, datetime],
        next_hop=-1,
    ):
        self.port = port
        self.metric = metric
        if isinstance(timeout_time, int):
            self.timeout_time = datetime.now() + timedelta(seconds=timeout_time)
        else:
            self.timeout_time = timeout_time
        self.next_hop = next_hop

    def shallow_copy(self):
        copy = RouteEntry(
            self.port, self.metric, self.timeout_time, self.next_hop
        )
        copy.flag = self.flag
        copy.gc_time = self.gc_time
        return copy

    def update_timeout_time(
        self, timeout_time: int, initial_time_arg: Optional[datetime] = None
    ) -> datetime:
        """
        Updates the timeout time, at which point this `RouteEntry` enter the
        deletion process.

        Returns the `initial_time`, which is the what `timeout_time` is added
        to.

        Keyword arguments:

        timeout_time -- The delta for between the `initial_time` and the new
        `timeout_time`.

        initial_time -- The initial time, defaults to `datetime.now()`
        """
        initial_time = (
            initial_time_arg if initial_time_arg is not None else datetime.now()
        )
        self.timeout_time = initial_time + timedelta(seconds=timeout_time)
        self.gc_time = None
        return initial_time

    def set_garbage_collection_time(
        self, gc_delta: int, initial_time_arg: Optional[datetime] = None
    ) -> datetime:
        """
        Updates the garbage collection time, at which point this `RouteEntry`
        will be removed from the table.

        Returns the `initial_time`, which is the what `gc_delta` is added to.
        Keyword arguments:
        gc_time -- The delta for between the `initial_time` and the new
        `gc_time`.

        initial_time -- The initial time, as specified. Defaults to
        `datetime.now()`
        """
        initial_time = (
            initial_time_arg if initial_time_arg is not None else datetime.now()
        )
        self.gc_time = initial_time + timedelta(seconds=gc_delta)
        return initial_time
```

4.1.8 router.py

```
import sys
from datetime import datetime
from socket import socket
from typing import List, Tuple

import routerbase
from input_processing import input_processing
from output_processing import deletion_process, send_response, send_responses
from packet import construct_packets
from poc_parser_v03 import read_config
from port_closer import port_closer
from port_opener import port_opener
from routingtable import RoutingTable
from validate_data import validate_data


def daemon(table: RoutingTable, sockets: List[socket], output_sock: socket):
    """Main body of the router."""
    while True:
        while table.sched_update_time > datetime.now():
            input_processing(table, sockets)
            deletion_process(table, output_sock)

        send_responses(table, output_sock)
        table.update_sched_update_time()


def create_table(
    router_id: int,
    sockets: List[socket],
    output_ports: List[Tuple[int, int, int]],
    timers: List[int],
):
    """
    Creates the routing table, and saves the information from the config file,
    so that it can be easily be accessed later from a single place.
    """
    update_time, timeout_time, gc_time, *extra = timers
    table = RoutingTable(router_id, update_time, timeout_time, gc_time)

    for port, cost, neighbour_router_id in output_ports:
        table.add_config_data(neighbour_router_id, port, cost)

    return table
```

```
def startup(table: RoutingTable, output_sock: socket):
    """Upon startup, it immediately sends out packets to neighbours."""
    routerbase.logger("Starting up...")
    for router_id in table.config_table:
        packets = construct_packets(table, router_id)
        for packet in packets:
            port = table.config_table[router_id].port
            send_response(output_sock, port, packet)


def get_params():
    """Gets the filename and logging level from the command line arguments."""
    if len(sys.argv) < 2:
        raise IndexError
    filename = sys.argv[1]
    is_debug = False
    if len(sys.argv) >= 3 and sys.argv[2].lower() == "debug":
        is_debug = True
    return filename, is_debug


def main():
    sockets: List[socket] = []
    try:
        filename, is_debug = get_params()
        routerbase.DEBUG_MODE = is_debug

        (router_id, input_ports, output_ports, timers) = read_config(filename)
        if not validate_data(router_id, input_ports, output_ports, timers):
            return

        result = port_opener(input_ports)

        if result is None:
            return
        else:
            sockets = result

        if len(sockets) == 0:
            routerbase.logger("No sockets were opened.")
            return

        # first open socket is chosen to be the output socket
        output_sock: socket = sockets[0]

        table = create_table(router_id, sockets, output_ports, timers)
        startup(table, output_sock)
        daemon(table, sockets, output_sock)

    except IndexError:
        routerbase.logger("Please give a filename. Correct")
    except FileNotFoundError:
        routerbase.logger("Please give a valid filename")
    except KeyboardInterrupt:
        routerbase.logger("\nKeyboard interrupt detected.")
    except ValueError:
        routerbase.logger("Invalid configuration file.")
    except Exception as ex:
        routerbase.logger("Something bad happened.")
        routerbase.logger(ex, is_debug=True)
    finally:
        routerbase.logger("Router shutting down.")
        port_closer(sockets)
        routerbase.logger("Bye!")


if __name__ == "__main__":
    main()
```

4.1.9 routerbase.py

```
from concurrent.futures import ThreadPoolExecutor
from datetime import datetime
from typing import Any

DEBUG_MODE = False

pool = ThreadPoolExecutor()


def logger(*output_args: Any, is_debug=False):
    """Custom logging solution."""
    # `*output_args` collects all the arguments to this function
    # which are not keyword arguments.
    output = ""
    for arg in output_args:
        output += str(arg)
    if is_debug and DEBUG_MODE:
        print(f"{datetime.now()} [DEBUG] {output}")
    elif is_debug is False:
        print(output)
```

4.1.10 routing_table.py

```
from datetime import datetime, timedelta
from random import randint
from time import sleep
from typing import Dict, Iterator, NamedTuple, Optional

import routerbase
from routeentry import RouteEntry


class ConfigData(NamedTuple):
    port: int
    cost: int


class RoutingTable:
    """
    Contains all of the entries for the routing table, for this router.

    Instance variables:

    table -- Contains the routing table, in the form of
    `{[key: router_id]: RouteEntry}`.

    config_table - Contains information from the config file.

    router_id -- The router of this router.

    sched_update_time -- The time at which a normally scheduled Response will
    be sent to other routers.

    update_delta -- The delta to increment the `sched_update_time` by.

    timeout_delta -- The delta for the time after which a route is no longer
    valid.

    gc_delta -- The delta for the time after which invalid routes are removed
    from the table.
    """

    table: Dict[int, RouteEntry]
    config_table: Dict[int, ConfigData]

    sched_update_time: datetime
    triggered_update_time: Optional[datetime] = None

    update_delta: int
    timeout_delta: int
    gc_delta: int

    def __init__(
        self,
        router_id: int,
        update_delta: int,
        timeout_delta: int,
        gc_delta: int,
    ):
        self.table = {}
        self.router_id = router_id
        self.update_delta = update_delta
        self.update_sched_update_time()
        self.timeout_delta = timeout_delta
        self.gc_delta = gc_delta
        self.config_table = {}

    def __len__(self):
        """Returns the number of items inside the routing table"""
        return len(self.table)

    def __iter__(self) -> Iterator[int]:
        """Iterates over the `RouteEntry` items in the routing table"""
        return iter(self.table.keys())

    def __getitem__(self, index: int) -> RouteEntry:
        """
        Returns a specific `RouteEntry` in the routing table, given its
        `router_id`.
        """
        return self.table[index]

    def __contains__(self, router_id: int) -> bool:
        """
        Checks to see if the given `router_id` is inside the routing table.
        """
        return router_id in self.table

    def __delitem__(self, router_id: int):
        """
        Removes the `router_id` and associated `RouteEntry` from the table.
        """
        self.remove_route(router_id)
    def _str_headers(self, router_id: int) -> str:
        output = "| self_id | router_id | port | next_hop | metric"
        if routerbase.DEBUG_MODE:
            output += " | flag"

        output += (
            " | "
            + "timeout_time".ljust(26)
            + " | "
            + "gc_time".ljust(26)
            + " |\n"
        )
        delim = output.split("|")
        for d in delim[1:-1]:
            output += "|" + "=" * len(d)
        output += "|\n"
        return output

    def _str_entry(self, router_id: int) -> str:
        """
        Returns the string representation of a `RouteEntry` inside the table.
        """
        e = self.table[router_id]
        output = (
            "| "
            + str(self.router_id).ljust(7)
            + " | "
            + str(router_id).ljust(9)
            + " | "
            + str(e.port).ljust(4)
            + " | "
            + str(e.next_hop).ljust(8)
            + " | "
            + str(e.metric).ljust(6)
            + " | "
        )

        if routerbase.DEBUG_MODE:
            output += str(e.flag).ljust(5) + "| "

        try:
            output += str(e.timeout_time).ljust(26)
        except AttributeError:
            output += str(None).ljust(26)
        output += " | "

        try:
            output += str(e.gc_time).ljust(26)
        except AttributeError:
            output += str(None).ljust(26)
        output += " |\n"

        return output

    def __str__(self):
        """Returns a string representation of the routing table."""

        output = ""

        router_id = None
        for router_id in self.table:
            output += self._str_entry(router_id)

        if output and router_id is not None:
            output = self._str_headers(router_id) + output
        else:
            output = "Empty table"

        return output

    def neighbours(self):
        """
        Returns the `router_id`s of the neighbouring routers.
        """
        return self.table.keys()

    def add_route(self, router_id: int, route: RouteEntry) -> None:
        """
        Adds the `RouteEntry`  to the table, and associates it with the given
        `router_id`."""
        self.table[router_id] = route

    def add_config_data(self, router_id: int, port: int, cost: int):
        self.config_table[router_id] = ConfigData(port, cost)

    def remove_route(self, router_id: int) -> None:
        """
        Removes the `router_id` and associated `RouteEntry` from the table.
        """
        # Creates a shallow copy of the dictionary. This prevents Python
        # complaining if there's a deletion while something is iterating over
        # the table.
        self.table = dict(self.table)
        del self.table[router_id]

    def update_sched_update_time(
        self, initial_time_arg: Optional[datetime] = None
    ) -> datetime:
        """
        Updates the scheduled time at which an update will be sent out for this
        `RouteEntry`. Returns the `initial_time`, which is the what
        `self.update_delta` is added to.

        Keyword arguments:

        initial_time -- The initial time, as specified. Defaults to
        `datetime.now()`
        """
        initial_time = (
            initial_time_arg if initial_time_arg is not None else datetime.now()
        )
        self.sched_update_time = initial_time + timedelta(
            seconds=self.update_delta + randint(-5, 5)
        )
        return initial_time

    def set_triggered_update_time(
        self, initial_time_arg: Optional[datetime] = None
    ) -> int:
        """
        Updates the triggered time at which an update will be sent out for
        this `RouteEntry`.
        Returns a Boolean indicating whether another triggered update has been
        requested. This detects if this method has been called after
        `self.triggered_update_time` has been set. If it has been set again,
        then it will return `False`. Otherwise, it will return `True`.

        Returns `False` if the `triggered_update_time` is after the next
        scheduled `sched_update_time`.

        Keyword arguments:

        initial_time -- The initial time, as specified. Defaults to
        `datetime.now()`.
        """
        initial_time = (
            initial_time_arg if initial_time_arg is not None else datetime.now()
        )
        diff = randint(1, 5)
        diff_delta = timedelta(seconds=diff)
        self.triggered_update_time = initial_time + diff_delta
        if self.triggered_update_time >= self.sched_update_time:
            return False

        sleep(diff)
        return initial_time + diff_delta == self.triggered_update_time
```

4.1.11 validate_data.py

```
##########################################################
#
# A function to validate that the data from the config
#   file is valid. Will print an error message,
#   and return an error code to calling program
#
# Version 01; 16 April 2019
#   First pass
#   Help from ID for the checking a set length idea
#   Needs checking output ports, metric
#   Design decision to seperate the config file parser
#    and validation of the read data
#
# Version 02;
#   reference material https://www.programiz.com/
# python-programming/methods/set/isdisjoint
#   Fixed a bunch of for and if statement format errors
#
# Version 03: 17 April 2019
#   more tidy up
#
# Version 04: 18 April 2019
#   work on addtional validation
#   changed error constants to variables - they are not
#   constants, just variables and initialisation, and
#   moved into module defintition
#
#  Version 05:
#   Adding doc tests:
#   Router ID = sort of OK
#   Moved the output-ports checking (port, metric, id)
#   into signle for loop
#
# Version 06: 19 April 2019
#   Removed the metric set (since we can have the same
#   metric more than once) and checkng of it. All we
#   need to check is the metric range is 1-15.
#   Fixed issue with timer checking.
#   Move the port reuse error code check to after the
#   metric and id error code checks. (When break out of
#   the for loop, the sets do not match)
#   Need to check for empty lists being passed in
#
# Version 07: 19 April 2019:
#   Switched to returning Boolean values
#
# Version 08: 20 April 2019:
#   Moved doctests to their own unit test module
#########################################################


from routerbase import logger

# Router ID limits
MIN_ID = 1
MAX_ID = 64000

# Input / Output port limits
MIN_PORT = 1024
MAX_PORT = 64000

# Metric limits
INFINITY = 16
MIN_METRIC = 1
MAX_METRIC = INFINITY

# Timer ratios
PERIODIC_DEAD_RATIO = 6
PERIODIC_GARBAGE_RATIO = 4


def validate_data(router_id, input_ports, output_ports, timers):
    id_error = False
    input_port_error = False
    output_port_error = False
    metric_error = False
    timers_error = False

    # Check Router ID
    if router_id is None:
        id_error = True
    else:
        if (router_id < MIN_ID) or (router_id > MAX_ID):
            id_error = True

    if id_error:
        logger("Router ID Configuration Error")
        return False

    # Check input ports
    temp_input_set = set()
    if input_ports is None:
        input_port_error = True
    else:
        for a_port in input_ports:
```

```
        if (a_port < MIN_PORT) or (a_port > MAX_PORT):
            input_port_error = True
            break
        temp_input_set.add(a_port)  # add port to a temporary list

# set error error if the length of temporary list
#   is not the same as length of original port list
if input_ports is None or len(temp_input_set) != len(input_ports):
    input_port_error = True

if input_port_error:
    logger("Input Ports Configuration Error")
    return False

# Check output ports
temp_output_port_set = set()
temp_id_set = set()
# temp_metric_set = set()

if output_ports is None:
    logger("Output Ports Configuration Error: No output ports were given")
    return False
else:
    for an_item in output_ports:
        # check port range
        if (an_item[0] < MIN_PORT) or (an_item[0] > MAX_PORT):
            logger("Output Ports Configuration Error: Port out of range")
            return False
            break
        temp_output_port_set.add(an_item[0])

        # check metric
        if (an_item[1] < MIN_METRIC) or (an_item[1] > MAX_METRIC):
            metric_error = True
            break
        # temp_metric_set.add(an_item[1])

        # check id
        if (an_item[2] < MIN_ID) or (an_item[2] > MAX_ID):
            id_error = True
            break
        temp_id_set.add(an_item[2])

# if length of set of ports not same as the length of output-ports, hv error
if len(temp_output_port_set) != len(output_ports):
    output_port_error = True

# check the set of output ports with set of input ports, if they
# are not disjoint (i.e. element(s) in common) we have error
if (temp_output_port_set.isdisjoint(temp_input_set)) is False:
    output_port_error = True

if metric_error:
    logger("Output Ports Configuration Error: Cost / Metric")
    return False

# we might need this i.e. check if a cost / metric is missing?
# if len(temp_metric_set) != len(output_ports(2)):
# id_error = True

# do we need a check for missing ID?
if id_error:
    logger("Output Ports Configuration Error: ID")
    return False

if output_port_error:
    logger("Output Ports Configuration Error: Port number re-use")
    return False

# Check Timers
if len(timers) != 3:
    timers_error = True
else:
    if (timers[1] / timers[0]) != PERIODIC_DEAD_RATIO:
        timers_error = True
    if (timers[2] / timers[0]) != PERIODIC_GARBAGE_RATIO:
        timers_error = True

if timers_error:
    logger("Timers Configuration Error")
    return False

# All good, yay! return a zero
return True
```

4.2 Configuration Files - System Testing

4.2.1 Test Scenario 1 - Single router instances

R2.cfg

```
router-id 2
input-ports 1030, 6020, 4010
output-ports 3001-6-3, 2002-4-6, 1040-2-4
timers 11, 66, 44
```

R2a.cfg

```
# Router ID = integer between 0 and 64000
router-id 2
garbage here
input-ports 1030, 6020, 4010 59029702329706
output-ports 3001-6-3, 2002-4-6, 1040-2-4
timers 11, 66, 44
```

4.2.2 Test Scenario 2 - Two peered routers

R1.cfg

```
router-id 1
input-ports 1030
output-ports 3001-8-3
timers 11, 66, 44
```

R3.cfg

```
router-id 3
input-ports 3001
output-ports 1030-8-1
timers 11, 66, 44
```

4.2.3 Test Scenario 3 - Three routers in a ring, equal costs of eight.

R5.cfg

```
router-id 5
input-ports 5007, 5011
output-ports 7005-8-7, 1105-8-11
timers 11, 66, 44
```

R7.cfg

```
router-id 7
input-ports 7005, 7011
output-ports 5007-8-5, 1107-8-11
timers 11, 66, 44
```

R11.cfg

```
router-id 11
input-ports 1107, 1105
output-ports 5011-8-5, 7011-8-7
timers 11, 66, 44
```

4.2.4 Test Scenario 4 - Three routers in a ring, equal costs of six.

R13a.cfg

```
router-id 13
input-ports 1317, 1319
output-ports 1713-8-17, 1913-6-19
timers 11, 66, 44
```

R17a.cfg

```
router-id 17
input-ports 1713, 1719
output-ports 1317-6-13, 1917-6-19
timers 11, 66, 44
```

R19a.cfg

```
router-id 19
input-ports 1913, 1917
output-ports 1319-6-13, 1719-6-17
timers 11, 66, 44
```

4.2.5 Test Scenario 5 - Three routers in a ring, unequal costs.

R13.cfg

```
router-id 13
input-ports 1317, 1319
output-ports 1713-8-17, 1913-2-19
timers 11, 66, 44
```

R17.cfg

```
router-id 17
input-ports 1713, 1719
output-ports 1317-8-13, 1917-2-19
timers 11, 66, 44
```

R19.cfg

```
router-id 19
input-ports 1913, 1917
output-ports 1319-2-13, 1719-2-17
timers 11, 66, 44
```

4.2.6 Test Scenario 6 - Three routers, linear (or hub and spoke), equal costs

R23.cfg

```
router-id 23
input-ports 2329, 2330, 2332, 2334
output-ports 2923-4-29, 3023-2-30, 3223-2-32, 3423-2-34
timers 11, 66, 44
```

R29.cfg

```
router-id 29
input-ports 2923, 2931, 2930, 2932
output-ports 2329-4-23, 3129-4-31, 3029-2-30, 3229-2-32
timers 11, 66, 44
```

R31.cfg.

```
router-id 31
input-ports 3129, 3130, 3132, 3134
output-ports 2931-4-29, 3031-2-30, 3231-2-32, 3431-2-34
timers 11, 66, 44
```

4.2.7 Test Scenario 7 - Four routers, linear, equal costs of two

R23a.cfg

```
router-id 23
input-ports 2329, 2330
output-ports 2923-2-29, 3023-2-30
timers 11, 66, 44
```

R29a.cfg

```
router-id 29
input-ports 2923, 2931, 2930
output-ports 2329-2-23, 3129-2-31, 3029-2-30
timers 11, 66, 44
```

R31a.cfg

```
router-id 31
input-ports 3129, 3137, 3138
output-ports 2931-2-29, 3731-2-37, 3831-2-38
timers 11, 66, 44
```

R37a.cfg

```
router-id 37
input-ports 3731, 3732
output-ports 3137-2-31, 3237-2-32
timers 11, 66, 44
```

4.2.8 Test Scenario 8 - Four routers, linear, equal costs of six

R23b.cfg

```
router-id 23
input-ports 2329, 2330
output-ports 2923-6-29, 3023-6-30
timers 11, 66, 44
```

R29b.cfg

```
router-id 29
input-ports 2923, 2931, 2930
output-ports 2329-6-23, 3129-6-31, 3029-6-30
timers 11, 66, 44
```

R31b.cfg

```
router-id 31
input-ports 3129, 3137, 3138
output-ports 2931-6-29, 3731-6-37, 3831-6-38
timers 11, 66, 44
```

R37b.cfg

```
router-id 37
input-ports 3731, 3732
output-ports 3137-6-31, 3237-6-32
timers 11, 66, 44
```

4.2.9 Test Scenario 9 - Assignment specification - Seven routers, partial mesh, various costs

R37.cfg

```
router-id 37
input-ports 3759, 3741, 3761
output-ports 5937-5-59, 4137-1-41, 6137-8-61
timers 11, 66, 44
```

R41.cfg

```
router-id 41
input-ports 4137, 4143
output-ports 4341-3-43, 3741-1-37
timers 11, 66, 44
```

R43.cfg

```
router-id 43
input-ports 4341, 4347
output-ports 4143-3-41, 4743-4-47
timers 11, 66, 44
```

R47.cfg

```
router-id 47
input-ports 4743, 4753, 4761
output-ports 4347-4-43, 5347-2-53, 6147-6-61
timers 11, 66, 44
```

R53.cfg

```
router-id 53
input-ports 5347, 5359
output-ports 4753-2-47, 5953-1-59
timers 11, 66, 44
```

R59.cfg

```
router-id 59
input-ports 5953, 5937
output-ports 5359-1-53, 3759-5-37
timers 11, 66, 44
```

R61.cfg.

```
router-id 61
input-ports 6147, 6137
output-ports 4761-6-47, 3761-8-37
timers 11, 66, 44
```

4.3 Source Code - Unit Tests

The following unit tests use Python’s `unittest` module.

4.3.1 `input_processing_test.py`

```python
from socket import AF_INET
from unittest import TestCase, main
from unittest.mock import Mock, patch

from input_processing import validate_entry
from packet import ResponseEntry
from routingtable import RoutingTable
from validate_data import MAX_ID, MAX_METRIC, MIN_ID, MIN_METRIC


class TestValidateEntry(TestCase):
    table: RoutingTable

    def setUp(self):
        self.table = RoutingTable(2, 1, 1, 1)

    def assertOutputEqual(self, expected: str, logger: Mock):
        actual = ""
        for item in logger.call_args:
            if isinstance(item, tuple):
                for i in item:
                    actual += i
        self.assertEqual(expected, actual)

    def test_valid_entry(self):
        entry = ResponseEntry(afi=AF_INET, router_id=1, metric=1)
        self.assertEqual(validate_entry(self.table, entry), True)

    @patch("input_processing.logger")
    def test_afi(self, logger):
        entry = ResponseEntry(afi=1, router_id=1, metric=1)
        self.assertEqual(validate_entry(self.table, entry), False)
        self.assertOutputEqual(
            "The value 1 for AFI does not match the expected "
            "value of AF_INET = 2.",
            logger,
        )

    @patch("input_processing.logger")
    def test_router_id(self, logger):
        self.table = RoutingTable(1, 1, 1, 1)
        entry = ResponseEntry(afi=AF_INET, router_id=1, metric=17)
        self.assertEqual(validate_entry(self.table, entry), False)

    @patch("input_processing.logger")
    def test_router_id_low(self, logger):
        entry = ResponseEntry(afi=AF_INET, router_id=0, metric=1)
        self.assertEqual(validate_entry(self.table, entry), False)
        self.assertOutputEqual(
            "The entry's router id of 0 should be an integer between "
            f"{MIN_ID} and {MAX_ID}, inclusive.",
            logger,
        )

    @patch("input_processing.logger")
    def test_router_id_high(self, logger):
        entry = ResponseEntry(afi=AF_INET, router_id=64001, metric=1)
        self.assertEqual(validate_entry(self.table, entry), False)
        self.assertOutputEqual(
            "The entry's router id of 64001 should be an integer between "
            f"{MIN_ID} and {MAX_ID}, inclusive.",
            logger,
        )

    @patch("input_processing.logger")
    def test_metric_low(self, logger):
        entry = ResponseEntry(afi=AF_INET, router_id=1, metric=0)
        self.assertEqual(validate_entry(self.table, entry), False)
        self.assertOutputEqual(
            "The entry's metric of 0 was not between the "
            f"expected range of {MIN_METRIC} and {MAX_METRIC}, inclusive.",
            logger,
        )

    @patch("input_processing.logger")
    def test_metric_high(self, logger):
        entry = ResponseEntry(afi=AF_INET, router_id=1, metric=17)
        self.assertEqual(validate_entry(self.table, entry), False)
        self.assertOutputEqual(
            "The entry's metric of 17 was not between the "
            f"expected range of {MIN_METRIC} and {MAX_METRIC}, inclusive.",
            logger,
        )


if __name__ == "__main__":
    main()
```

4.3.2 packet_test.py

```
from socket import AF_INET
from unittest import TestCase, main
from unittest.mock import Mock, patch

import routerbase
from packet import (
    ResponsePacket,
    construct_packets,
    read_packet,
    validate_packet,
)
from routeentry import RouteEntry
from routingtable import RoutingTable


def get_single_packet():
    packet = bytearray(24)
    packet[0] = 2
    packet[1] = 2
    packet[2:4] = (0).to_bytes(2, "big")
    packet[4:6] = AF_INET.to_bytes(2, "big")
    packet[8:12] = (1).to_bytes(4, "big")
    packet[20:24] = (1).to_bytes(4, "big")
    return packet


def get_two_packets():
    packet1 = bytearray(504)
    packet1[0] = 2
    packet1[1] = 2
    packet1[2:4] = (0).to_bytes(2, "big")
    diff = 1
    for i in range(25):
        # if i + diff == 2:
        #     diff += 1
        packet1[i * 20 + 4 : i * 20 + 6] = AF_INET.to_bytes(2, "big")
        packet1[i * 20 + 8 : i * 20 + 12] = (i + diff).to_bytes(4, "big")
        packet1[i * 20 + 20 : i * 20 + 24] = (1).to_bytes(4, "big")

    packet2 = bytearray(144)
    packet2[0] = 2
    packet2[1] = 2
    packet2[2:4] = (0).to_bytes(2, "big")
    for i in range(7):
        packet2[i * 20 + 4 : i * 20 + 6] = AF_INET.to_bytes(2, "big")
        packet2[i * 20 + 8 : i * 20 + 12] = (i + 26).to_bytes(4, "big")
        packet2[i * 20 + 20 : i * 20 + 24] = (1).to_bytes(4, "big")

    return packet1, packet2


class TestPacketConstruction(TestCase):
    def setUp(self):
        routerbase.DEBUG_MODE = True

    def _test_single_packet(self, packet, expected_packet, iteration=0):
        for i, val in enumerate(expected_packet):
            self.assertEqual(
                packet[i],
                val,
                f"Iteration {iteration}. Failed on byte {i}. Expected: {val}. "
                f"Received: {packet[i]}",
            )

    def test_single_entry(self):
        """
        Tests a routing table where the single entry does not match the
        given router_id.
        """
        table = RoutingTable(0, 0, 0, 0)
        # metric: 1, next_hop: 2
        table.add_route(1, RouteEntry(0, 1, 0, 2))

        packets = construct_packets(table, 3)

        expected_packet = get_single_packet()

        self.assertEqual(len(packets), 1)
        packet = packets[0]
        self.assertEqual(len(packet), len(expected_packet))
        self._test_single_packet(packet, expected_packet)

    def test_two_entries_router_id_clash(self):
        """
        The routing table has two entries, where one entry was learnt from the
        router that the packet is going to be sent to. The packet being
        produced should contain two entries, however the entry with the id
        clash should have a metric of 16.
        """
        table_router_id = 1
        table = RoutingTable(table_router_id, 0, 0, 0)
        table.add_route(1, RouteEntry(0, 1, 0, 0))
        table.add_route(2, RouteEntry(0, 2, 0, 3))

        packets = construct_packets(table, 3)

        expected_packet = bytearray(44)
        expected_packet[0] = 2
        expected_packet[1] = 2
        expected_packet[2:4] = (1).to_bytes(2, "big")
        expected_packet[4:6] = AF_INET.to_bytes(2, "big")
        expected_packet[8:12] = (1).to_bytes(4, "big")
        expected_packet[20:24] = (1).to_bytes(4, "big")

        expected_packet[20 + 4 : 20 + 6] = AF_INET.to_bytes(2, "big")
        expected_packet[20 + 8 : 20 + 12] = (2).to_bytes(4, "big")
        expected_packet[20 + 20 : 20 + 24] = (16).to_bytes(4, "big")

        self.assertEqual(len(packets), 1)
        packet = packets[0]
        self.assertEqual(len(packet), len(expected_packet))
        self._test_single_packet(packet, expected_packet)

    def test_two_entries_infinity(self):
        """
        Tests that a routing table with two entries, where one entry has a
        metric of infinity, [OLD: but is flagged], produces a packet with
        the two entries inside.
        """
        table = RoutingTable(0, 0, 0, 0)
        table.add_route(1, RouteEntry(0, 1, 0, 0))
        table.add_route(2, RouteEntry(0, 16, 0, 0))
        # table[2].flag = True

        packets = construct_packets(table, 1)

        expected_packet = bytearray(44)
        expected_packet[0] = 2
        expected_packet[1] = 2
        expected_packet[2:4] = (0).to_bytes(2, "big")
        expected_packet[4:6] = AF_INET.to_bytes(2, "big")
        expected_packet[8:12] = (1).to_bytes(4, "big")
        expected_packet[20:24] = (1).to_bytes(4, "big")
        expected_packet[20 + 4 : 20 + 6] = AF_INET.to_bytes(2, "big")
        expected_packet[20 + 8 : 20 + 12] = (2).to_bytes(4, "big")
        expected_packet[20 + 20 : 20 + 24] = (16).to_bytes(4, "big")

        self.assertEqual(len(packets), 1)
        packet = packets[0]
        self.assertEqual(len(packet), len(expected_packet))
        self._test_single_packet(packet, expected_packet)

    # def test_two_entries_flag(self):
    #     """
    #     Tests that a routing table with two entries, where one entry has a
    #     metric of infinity, and is not flagged, produces a packet with the
    #     entry which isn't infinity inside.
    #     """
    #     table = RoutingTable(0, 0, 0, 0)
    #     table.add_route(1, RouteEntry(0, 1, 0, 0))
    #     table.add_route(2, RouteEntry(0, 16, 0, 0))
    #     table[2].flag = False

    #     packets = construct_packets(table, 3)

    #     expected_packet = bytearray(24)
    #     expected_packet[0] = 2
    #     expected_packet[1] = 2
    #     expected_packet[2:4] = (0).to_bytes(2, "big")
    #     expected_packet[4:6] = AF_INET.to_bytes(2, "big")
    #     expected_packet[8:12] = (1).to_bytes(4, "big")
    #     expected_packet[20:24] = (1).to_bytes(4, "big")

    #     self.assertEqual(len(packets), 1)
    #     packet = packets[0]
    #     self.assertEqual(len(packet), len(expected_packet))
    #     self._test_single_packet(packet, expected_packet)

    def test_multiple_packets(self):
        """
        Tests that multiple packets are returned, when the number of routes is
        greater than 25.
        """
        # Number of entries inside the table: 32
        # Number of entries being sent out: 31
        table = RoutingTable(0, 0, 0, 0)
        table.add_route(1, RouteEntry(0, 1, 0, 0))
        table.add_route(2, RouteEntry(0, 1, 0, 0))
        # table.add_route(2, RouteEntry(0, 16, 3, 0, 0))
        # table[2].flag = False
        for i in range(30):
            table.add_route(i + 3, RouteEntry(0, 1, 0, 0))

        packets = construct_packets(table, 3)
        expected_packets = get_two_packets()

        self.assertEqual(len(packets), 2)
        for i, packet in enumerate(packets):
            expected_packet = expected_packets[i]
            self.assertEqual(len(packet), len(expected_packet))
            self._test_single_packet(packet, expected_packet, i)


class TestPacketReading(TestCase):
    def setUp(self):
        routerbase.DEBUG_MODE = True

    def test_single_entry(self):
        """Tests that a packet with a single entry can be read correctly."""
        packet = get_single_packet()
        command, version, sender_router_id, entries = read_packet(packet)

        self.assertEqual(command, 2)
        self.assertEqual(version, 2)
        self.assertEqual(sender_router_id, 0)
        self.assertEqual(len(entries), 1)
        for afi, router_id, metric in entries:
            self.assertEqual(afi, AF_INET)
            self.assertEqual(router_id, 1)
            self.assertEqual(metric, 1)
    def test_multiple_entries(self):
        """
        Tests that a packet with multiple entries (in this case 25
        entries) can be correctly read.
        """
        packet, _ = get_two_packets()
        command, version, sender_router_id, entries = read_packet(packet)

        self.assertEqual(command, 2)
        self.assertEqual(version, 2)
        self.assertEqual(sender_router_id, 0)
        self.assertEqual(len(entries), 25)

        for i, (afi, router_id, metric) in enumerate(entries):
            expected_router_id = i + 1
            self.assertEqual(afi, AF_INET)
            self.assertEqual(router_id, expected_router_id)
            self.assertEqual(metric, 1)


class TestValidatePacket(TestCase):
    table: RoutingTable

    def setUp(self):
        self.table = RoutingTable(
            router_id=1, update_delta=10, timeout_delta=60, gc_delta=40
        )
        self.table.add_route(
            router_id=2,
            route=RouteEntry(port=4000, metric=2, timeout_time=60, next_hop=3),
        )

    def assertOutputEqual(self, expected: str, logger: Mock):
        actual = ""
        for item in logger.call_args:
            if isinstance(item, tuple):
                for i in item:
                    actual += i
        self.assertEqual(expected, actual)

    @patch("packet.logger")
    def test_valid_packet(self, logger):
        packet = ResponsePacket(
            command=2, version=2, sender_router_id=2, entries=[]
        )
        self.assertEqual(validate_packet(self.table, packet), True)

    @patch("packet.logger")
    def test_sender_is_self(self, logger):
        packet = ResponsePacket(
            command=2, version=2, sender_router_id=1, entries=[]
        )
        self.assertEqual(validate_packet(self.table, packet), False)
        self.assertOutputEqual(
            "The packet's router_id of 1 illegally matches the router_id of "
            "this router.",
            logger,
        )

    @patch("packet.logger")
    def test_invalid_command(self, logger):
```

```python
        packet = ResponsePacket(
            command=1, version=2, sender_router_id=2, entries=[]
        )
        self.assertEqual(validate_packet(self.table, packet), False)
        self.assertOutputEqual(
            "The packet has a command value of 1, instead of 2.", logger
        )

    @patch("packet.logger")
    def test_invalid_version(self, logger):
        packet = ResponsePacket(
            command=2, version=1, sender_router_id=2, entries=[]
        )
        self.assertEqual(validate_packet(self.table, packet), False)
        self.assertOutputEqual(
            "The packet has a version value of 1, instead of 2.", logger
        )


if __name__ == "__main__":
    main()
```

4.3.3 validate_data_test.py

```python
import sys
from io import StringIO
from unittest import TestCase, main

from validate_data import validate_data


class ValidateDataTest(TestCase):
    captured_output = StringIO()

    def setUp(self):
        self.captured_output = StringIO()
        sys.stdout = self.captured_output

    def assertOutputEqual(self, expected: str, endline="\n"):
        sys.stdout = sys.__stdout__
        self.assertEqual(expected + endline, self.captured_output.getvalue())

    def test_good_data(self):
        """Succeeds for valid data."""
        self.assertEqual(
            validate_data(
                1,
                [3001, 4001, 5001],
                [(5003, 3, 5), (9003, 3, 9), (1303, 3, 13)],
                [10, 60, 40],
            ),
            True,
        )

    def test_router_id_low(self):
        """Error: router id is too low"""
        self.assertEqual(
            validate_data(
                (0),
                [3001, 4001, 5001],
                [(5003, 3, 5), (9003, 3, 9), (1303, 3, 13)],
                [10, 60, 40],
            ),
            False,
        )
        self.assertOutputEqual("Router ID Configuration Error")

    def test_router_id_high(self):
        """Error: router id is too high"""
        self.assertEqual(
            validate_data(
                (64001),
                [3001, 4001, 5001],
                [(5003, 3, 5), (9003, 3, 9), (1303, 3, 13)],
                [10, 60, 40],
            ),
            False,
        )
        self.assertOutputEqual("Router ID Configuration Error")

    def test_input_ports_low(self):
        """Error: input port number is too low"""
        self.assertEqual(
            validate_data(
                (1),
                [1023, 4001, 5001],
                [(5003, 3, 5), (9003, 3, 9), (1303, 3, 13)],
                [10, 60, 40],
            ),
            False,
        )
        self.assertOutputEqual("Input Ports Configuration Error")

    def test_input_ports_high(self):
        """Error: input ports number is too high"""
        self.assertEqual(
            validate_data(
                (1),
                [3001, 4001, 64001],
                [(5003, 3, 5), (9003, 3, 9), (1303, 3, 13)],
                [10, 60, 40],
            ),
            False,
        )
        self.assertOutputEqual("Input Ports Configuration Error")

    def test_input_ports_reuse(self):
        """Error: input port reused."""
        self.assertEqual(
            validate_data(
                (1),
                [3001, 4001, 3001],
                [(5003, 3, 5), (9003, 3, 9), (1303, 3, 13)],
                [10, 60, 40],
            ),
            False,
        )
        self.assertOutputEqual("Input Ports Configuration Error")

    def test_output_ports_range_low(self):
        """Error: output port too low."""
        self.assertEqual(
            validate_data(
                1,
                [3001, 4001, 5001],
                [(1022, 3, 5), (9003, 3, 9), (1303, 3, 13)],
                [10, 60, 40],
            ),
            False,
        )
        self.assertOutputEqual(
            "Output Ports Configuration Error: Port out of range"
        )

    def test_output_ports_range_high(self):
        """Error: output port too high."""
        self.assertEqual(
            validate_data(
                1,
                [3001, 4001, 5001],
                [(5003, 3, 5), (64001, 3, 9), (1303, 3, 13)],
                [10, 60, 40],
            ),
            False,
        )
        self.assertOutputEqual(
            "Output Ports Configuration Error: Port out of range"
        )

    def test_output_ports_reuse(self):
        """Error: output ports reused"""
        self.assertEqual(
            validate_data(
                1,
                [3001, 4001, 5001],
                [(5003, 3, 5), (1303, 3, 9), (1303, 3, 13)],
                [10, 60, 40],
            ),
            False,
        )
        self.assertOutputEqual(
            "Output Ports Configuration Error: Port number re-use"
        )

    def test_output_ports_reuse_input(self):
        """Error: output port reused in the input ports"""
        self.assertEqual(
            validate_data(
                1,
                [3001, 4001, 5001],
                [(5003, 3, 5), (9003, 3, 9), (3001, 3, 13)],
                [10, 60, 40],
            ),
            False,
        )
        self.assertOutputEqual(
            "Output Ports Configuration Error: Port number re-use"
        )

    def test_output_cost_low(self):
        """Error: cost too low"""
        self.assertEqual(
            validate_data(
                1,
                [3001, 4001, 5001],
                [(5003, 0, 5), (9003, 15, 9), (1303, 3, 13)],
                [10, 60, 40],
            ),
            False,
        )
        self.assertOutputEqual(
            "Output Ports Configuration Error: Cost / Metric"
        )

    def test_output_cost_infinity(self):
        """Valid input: Cost is infinity"""
        self.assertEqual(
            validate_data(
                1,
                [3001, 4001, 5001],
                [(5003, 1, 5), (9003, 16, 9), (1303, 3, 13)],
                [10, 60, 40],
            ),
            True,
        )

    def test_output_cost_high(self):
        """Error: cost too high"""
        self.assertEqual(
            validate_data(
                1,
                [3001, 4001, 5001],
                [(5003, 1, 5), (9003, 17, 9), (1303, 3, 13)],
                [10, 60, 40],
            ),
            False,
        )
        self.assertOutputEqual(
            "Output Ports Configuration Error: Cost / Metric"
        )

    def test_output_cost_valid(self):
        """Valid input: different costs"""
        self.assertEqual(
            validate_data(
                1,
                [3001, 4001, 5001],
                [(5003, 1, 5), (9003, 15, 9), (1303, 3, 13)],
                [10, 60, 40],
            ),
            True,
        )

    def test_output_id_low(self):
        """Error: router id is too low"""
        self.assertEqual(
            validate_data(
                1,
                [3001, 4001, 5001],
                [(5003, 3, 0), (9003, 3, 9), (1303, 3, 13)],
                [10, 60, 40],
            ),
            False,
        )
        self.assertOutputEqual("Output Ports Configuration Error: ID")

    def test_output_id_high(self):
        """Error: router id is too high"""
        self.assertEqual(
            validate_data(
                1,
                [3001, 4001, 5001],
                [(5003, 3, 1), (9003, 3, 64001), (1303, 3, 13)],
                [10, 60, 40],
            ),
            False,
        )
        self.assertOutputEqual("Output Ports Configuration Error: ID")

    def test_output_id_valid(self):
        """Valid input: router ids"""
        self.assertEqual(
            validate_data(
                1,
                [3001, 4001, 5001],
                [(5003, 3, 1), (9003, 3, 64000), (1303, 3, 13)],
                [10, 60, 40],
            ),
            True,
        )

    def test_timers_periodic_1(self):
        """Error: incorrect periodic timer ratio"""
        self.assertEqual(
            validate_data(
                1,
                [3001, 4001, 5001],
                [(5003, 3, 5), (9003, 3, 9), (1303, 3, 13)],
                [10, 50, 40],
            ),
            False,
        )
        self.assertOutputEqual("Timers Configuration Error")

    def test_timers_gc(self):
        """Error: incorrect gc timer ratio"""
        self.assertEqual(
            validate_data(
                1,
                [3001, 4001, 5001],
                [(5003, 3, 5), (9003, 3, 9), (1303, 3, 13)],
                [10, 60, 30],
            ),
            False,
        )
        self.assertOutputEqual("Timers Configuration Error")

    def test_timers_periodic_2(self):
        """Error: incorrect periodic timer ratio"""
        self.assertEqual(
            validate_data(
                1,
                [3001, 4001, 5001],
                [(5003, 3, 5), (9003, 3, 9), (1303, 3, 13)],
                [10, 35, 40],
            ),
            False,
        )
        self.assertOutputEqual("Timers Configuration Error")
    def test_timers_wrong_self(self):
        """Error: generally incorrect timer ratios"""
        self.assertEqual(
            validate_data(
                1,
                [3001, 4001, 5001],
                [(5003, 3, 5), (9003, 3, 9), (1303, 3, 13)],
                [20, 10, 42],
            ),
            False,
        )
        self.assertOutputEqual("Timers Configuration Error")


if __name__ == "__main__":
    main()
```

4.4 Configuration file generation

For a limited set of number combinations using these rules should make it easier to write and debug configuration files.

Use a prime number as the router ID.
For input ports, the first two digits are this router ID. The last two digits are the source router ID. (I.e. the neighbour router connecting into this router.)
For output ports, the last two digits are this router ID. The first two digits as the destination router ID. (I.e. the neighbour router being connected to.)
The filename is R<router ID>.cfg.
Where additional tests have been inserted and the router ID has been reused, the filename becomes R<router ID><a>.cfg, where <a> is a lower-case character.

Example.
Router ID = **17**, the connected router = *19*, metric or link cost = 4.

R17.cfg is:
Router-id **17**
input-ports **17**_19_
output-ports 19**17**-4-19

4.5 Plagiarism Declarations

Plagiarism Declaration

This form needs to accompany your COSC 364 assignment submission.

I understand that plagiarism means taking someone else's work (text, program code, ideas, concepts) and presenting them as my own, without proper attribution. Taking someone else's work can include verbatim copying of text, figures/images, or program code, or it can refer to the extensive use of someone else's original ideas, algorithms or concepts.

I hereby declare that:

- My assignment is my own original work. I have not reproduced or modified code, figures/images, or writings of others without proper attribution. I have not used original ideas and concepts of others and presented them as my own.
- I have not allowed others to copy or modify my own code, figures/images, or writings. I have not allowed others to use original ideas and concepts of mine and present them as their own.
- I accept that plagiarism can lead to consequences, which can include partial or total loss of marks, no grade being awarded and other serious consequences, including notification of the University Proctor.

Name: Isaac Daly

Student ID: 85231671

Signature:

Date: 08/05/2014

Plagiarism Declaration

This form needs to accompany your COSC 364 assignment submission.

I understand that plagiarism means taking someone else's work (text, program code, ideas, concepts) and presenting them as my own, without proper attribution. Taking someone else's work can include verbatim copying of text, figures/images, or program code, or it can refer to the extensive use of someone else's original ideas, algorithms or concepts.

I hereby declare that:

- My assignment is my own original work. I have not reproduced or modified code, figures/images, or writings of others without proper attribution. I have not used original ideas and concepts of others and presented them as my own.
- I have not allowed others to copy or modify my own code, figures/images, or writings. I have not allowed others to use original ideas and concepts of mine and present them as their own.
- I accept that plagiarism can lead to consequences, which can include partial or total loss of marks, no grade being awarded and other serious consequences, including notification of the University Proctor.

Name: MANU HAMBLYN

Student ID: 95140875

Signature:

Date: 08/05/2019