COSC364 Assignment 1

RIP ROUTING

Matthew Knight - 22442875
Sam Boyles – 14560776

Agreed Contributions:

Sam – 50%
Matthew – 50%

Positive Aspects

We believe that our implementation in an Object Oriented way allowed us to easily organise our code into generic reusable pieces, separating all the main components of the RIPv2 protocol into their own classes. Having the routing table entries as their own RoutingRow class was central to the design as these entries are arguably the most important component of the implementation, and allowed for easy manipulation of the rows upon route changes or updates being received.

Another aspect that we believe we have implemented well is the processing of triggered updates. These updates occur whenever a route has been changed in the routing table, whether it be due to a link going down or a new route being added.

Aspects to be Improved

While we believe our implementation of the RIPv2 protocol is effective given the timeframe, there is always room for improvement. One aspect of the implementation we would try to improve is the complexity of some of our functions, as some of these have multiple nested loops in order to achieve their desired functionality. In particular, in the 'process_route_entry' function we are looping through a routers routing table multiple times, which may end up having scalability issues if there are a large number of routers in the network. However, in the scope of our assignment this wasn't a major issue so was not one that we saw as an urgent change.

Atomicity

In our implementation, we have achieved atomicity by running through a list of jobs that would be triggered due to certain conditions multiple times a second. If a job (e.g. to process an inbound packet or perform a triggered update) were to be called it would complete in its entirety before moving on to another job in a completely sequential manner. This allows every job to have a current and valid view of the flags and routing tables of each RIP instance.

Testing

During the testing of our RIPv2 Implementation, we wanted to ensure that our program was working as initially expected and to ensure that at each stage the routing tables were being correctly populated. At the beginning of the assignment we set out to write a list of Expected Outcomes, and upon completion of our implementation, we went through to ensure the expected outcomes matched the actual outcome of our tests. All of our testing was based on the example network given in Figure 1 on Page 8 of the Assignment Specification.

Initial setup from configuration files

When testing this we wanted to ensure that our process for reading configuration files and assigning these values was working as expected.

Expected Outcome: The routing table will contain entries only for the neighbours specified in the 'outputs' section of the configuration files. There will also be sockets created for each specified 'input-port'.

Actual Outcome: To test this, we initialised each of the routers using 'python3 rip_demon.py configX.json', where X is in the range 1 – 7, and then printed out the resulting routing table after our RoutingTable class' populate table method was called. We then cross referenced this result with what is expected based on the example network. We also printed out confirmation of each input socket being created after these had been created. By manual inspection, all of the entries in the table and the created input sockets were correct.

Discovery of new routes when a new router connects to the network

When testing this we wanted to ensure that upon a new router being switched on, updates would be sent to neighbouring routers informing them of any new routes that are accessible through themselves.

Expected Outcome: Having switched on the routers with ID 1 and 2, and having Router 2's initial table containing routes to router 1 and its neighbours, upon switching on Router 3, Router 2's routing table should be updated to contain a route to router 3 and all of its neighbours.

Actual Outcome: To test this, we initialised Routers 1 and 2 as suggested, and took note of the routing table of Router 2 by cross referencing with the network diagram to ensure correctness. We then switched on Router 3 and after giving the network time to converge, again checked the output of Router 2's routing table. We found that the table of router 2 had been correctly updated to reflect the addition of Router 3 to the network.

Removing routes when they become invalid

The purpose of this test is to ensure that the network is being properly informed of any changes in topology once a link goes down, and that all routing tables are being updated accordingly.

Expected Outcome: After all routers have been initialised, the removal of one of these should trigger updates to inform the rest of the network that this node is unreachable. The routing tables should be updated to no longer include a route to the removed router.

Actual Outcome: After initialising all routers 1 through 7, we switched off Router 5 and allowed the network some time to realise the link was down and trigger updates accordingly. All of the remaining routers in the network correctly updated the routing tables to account for the fact that Router 5 was no longer reachable.

Routes becoming active again after being stopped

The purpose of this test is to ensure that, following a router being switched off and the routing tables being updated to reflect this, upon the router being switched back on the routing tables of all routers will go back to as they were before the router was switched off initially.

Expected Outcome: Upon the router being switched back on, the network shall converge again to incorporate this link being active again

Actual Outcome: After initialising all routers 1 through 7, we switched off Router 5 and allowed the network some time to realise the link was down and trigger updates accordingly. After confirming the routing table were correct as per the previous test, we switched Router 5 back on and observed all of the routers updating their routing table to again incorporate Router 5 in their routes.

Configuration Files – For demonstration network

config1.json

```
{
   "router-id" : "1",
   "input-ports" : [1113, 1114, 1103],
   "outputs" : "1100-1-2, 1101-8-7, 1102-5-6"
}
```

config2.json

```
{
   "router-id" : "2",
   "input-ports" : [1100, 1105],
   "outputs" : "1103-1-1, 1104-3-3"
}
```

config3.json

```
{
   "router-id" : "3",
   "input-ports" : [1104, 1107],
   "outputs" : "1105-3-2, 1106-4-4"
}
```

config4.json

```
{
   "router-id" : "4",
   "input-ports" : [1106, 1115, 1110],
   "outputs" : "1107-4-3, 1108-6-7, 1109-2-5"
}
```

config5.json

```
{
   "router-id" : "5",
   "input-ports" : [1109, 1112],
   "outputs" : "1110-2-4, 1111-1-6"
}
```

config6.json

```
{
   "router-id" : "6",
   "input-ports" : [1102, 1111],
   "outputs" : "1112-1-5, 1113-5-1"
}
```

config7.json

```
{
   "router-id" : "7",
   "input-ports" : [1101, 1108],
   "outputs" : "1114-8-1, 1115-6-4"
```

rip_demon.py

```
1.  import sys
2.  import json
3.  import select
4.  import threading
5.  import time
6.  import copy
7.  import random
8.  import pickle
9.  from socket import *
10. import routing_table
11. import rip_packet
12.
13.
14. class RipDemon(threading.Thread):
15.
16.     """Creates an instance of a router that implements the RIP routing Daemon.
17.
18.        Router can be initialised in the command line with:
19.
20.            python rip_demon.py configX.json
21.
22.        Where X is the config file to be used (1-10)
23.     """
24.
25.     def __init__(self, filename, intervalBetweenMessages=1, random=False, timeoutPeriod=15, garbageCollect
    ionPeriod=15):
26.         threading.Thread.__init__(self)
27.         self.filename = filename
28.         data = json.load(open(self.filename))
29.         self.config_file_check(data)
30.         self.routing_id = data["router-id"]
31.         self.input_ports = data["input-ports"]
32.         self.output_ports = data["outputs"]
33.         self.input_sockets_list = []
34.         self.socket_creator()
35.         self.routing_table = routing_table.RoutingTable(data)
36.         self.alive = False
37.
38.         print(self.routing_table.getPrettyTable())
39.
40.         # Timer settings
41.         self.timer_interval = intervalBetweenMessages
42.         self.timer_value = 0
43.         self.triggered_update_cooldown_timer_value = 0
44.         self.hasRecentlyTriggeredUpdate = False
45.         self.timeout_period = timeoutPeriod
46.         self.garbage_collection_period = garbageCollectionPeriod
47.         self.random = random
48.         self.ready_for_periodic_update = False
49.         self.ready_for_triggered_update = False
50.
51.     def socket_creator(self):
52.         """Initialise the sockets and create a list of socket objects
53.            based on the number of ports specified in config file"""
54.         for port in self.input_ports:
55.             server_socket = socket(AF_INET, SOCK_DGRAM)
56.             server_socket.bind(('', port))
57.             self.input_sockets_list.append(server_socket)
58.             print("Waiting on port " + str(port))
59.
60.     def run(self):
61.         while True:
62.             readable, writable, exceptional = select.select(self.input_sockets_list, [], [], 0.1)
```

```
63.             for s in readable:
64.                 packet, addr = s.recvfrom(2048)
65.
66.                 unpickledRIPReceivedPacket = pickle.loads(packet)
67.
68.                 # Reset timeout timer of destId of received packed
69.                 receivedFromDestId = unpickledRIPReceivedPacket.getRouterId()
70.                 for Route in self.routing_table.getRoutesWithTimers():
71.                     if Route.getDestId() == receivedFromDestId:
72.                         Route.resetTime()
73.
74.                 identical_entry_found = False
75.                 port_to_send = addr[1]
76.                 for found_row in unpickledRIPReceivedPacket.getRIPEntries().getRoutingTable():
77.                     for current_row in self.routing_table.getRoutingTable():
78.                         if current_row.row_as_list() == found_row.row_as_list():
79.                             identical_entry_found = True
80.
81.                     if unpickledRIPReceivedPacket.getRouterId() != self.routing_id and identical_entry_fou
    nd == False:
82.                         self.process_route_entry(found_row, unpickledRIPReceivedPacket.getRouterId(), port
    _to_send)
83.
84.             self.timer_tick()
85.
86.     def timer_tick(self):
87.         # Timer component
88.         if self.random:
89.             tickTime = 0.8 + (random.randint(0, 4)) / 10
90.         else:
91.             tickTime = 1.0
92.
93.         time.sleep(tickTime)
94.         # update count value
95.         self.timer_value += 1
96.
97.         if self.timer_value == self.timer_interval:
98.             self.ready_for_periodic_update = True
99.             self.timer_value = 0
100.
101.                if self.ready_for_periodic_update:
102.                    self.periodic_update()
103.                    self.ready_for_periodic_update = False
104.
105.                self.check_for_changed_routes()
106.
107.                # Check if has recently sent a triggered update
108.                if self.ready_for_triggered_update:
109.                    if not self.hasRecentlyTriggeredUpdate:
110.                        self.triggered_update()
111.                        self.triggered_update_cooldown_timer_value = 0
112.                        self.ready_for_triggered_update = False
113.                        self.hasRecentlyTriggeredUpdate = True
114.                    else:
115.                        self.triggered_update_cooldown_timer_value += 1
116.                        if self.triggered_update_cooldown_timer_value >= (random.randint(1, 5)):
117.                            self.hasRecentlyTriggeredUpdate = False
118.
119.                for Route in self.routing_table.getRoutesWithTimers():
120.                    if Route.hasTimedOut():
121.                        Route.incrementGarbageCollectionTime()
122.                        if Route.getGarbageCollectionTime() == self.garbage_collection_period:
123.                            self.routing_table.removeFromRoutingTable(Route.getDestId())
124.                            self.routing_table.removeFromRoutingTable(Route.getDestId())
125.                    else:
126.                        Route.incrementTimeoutTime()
```

```
                        if Route.getTimeoutTime() == self.timeout_period:
                            self.set_row_as_timed_out(Route)

            def check_for_changed_routes(self):
                for Row in self.routing_table.getRoutingTable():
                    if Row.hasChanged():
                        self.ready_for_triggered_update = True
                        return

                neighbour_rows = []
                non_neighbour_rows = copy.deepcopy(self.routing_table.getRoutingTable())

                for Route in self.routing_table.getRoutesWithTimers():
                    for Row in self.routing_table.getRoutingTable():
                        if int(Route.getDestId()) == int(Row.getDestId()):
                            neighbour_rows.append(Row)
                        if int(Route.getDestId()) == int(Row.getDestId()) and not Route.hasTimedOut() and
    Row.getLinkCost() == 16:
                            self.routing_table.removeFromRoutingTable(Route.getDestId())
                            return

                for Row in neighbour_rows:
                    non_neighbour_rows.remove(Row)

                for Row in non_neighbour_rows:
                    if Row.getLinkCost() == 16:
                        self.routing_table.removeFromRoutingTable(Row.getDestId())

            def set_row_as_timed_out(self, route):
                route.setRouteAsTimedOut()
                for Row in self.routing_table.getRoutingTable():

                    if int(Row.getDestId()) == int(route.getDestId()):
                        Row.updateLinkCost(16)
                        Row.setHasBeenChanged()

            def reset_timers_of_dest(self, destId):
                for Route in self.routing_table.getRoutesWithTimers():
                    if int(Route.getDestId()) == int(destId):
                        Route.resetTime()

            def process_route_entry(self, new_row, sending_router_id, port_to_send):
                # If there's no entry for the senders id
                flag = False
                for row in self.routing_table.getRoutingTable():
                    if int(row.getDestId()) == int(sending_router_id):
                        flag = True
                        break
                if not flag:
                    self.routing_table.addOneFromConfig()

                # If next hop port is one of your own, skip this entry
                if new_row.getNextHopPort() in self.routing_table.getInputPorts():
                    return
                # If row already exists exactly, skip this entry also
                if new_row in self.routing_table.getRoutingTable():
                    return

                destination_router = new_row.getDestId()
                new_distance = new_row.getLinkCost()

                # If router receives a update to a link with metric 16 (i.e that link is down)
                if new_distance == 16:
                    for row in self.routing_table.getRoutingTable():
                        if row.getDestId() == destination_router:
                            row.updateLinkCost(16)
```

```
                            row.setHasBeenChanged()
                            break

                for old_row in self.routing_table.getRoutingTable():
                    # Make the route non modifiable when it hits 16
                    if old_row.getLinkCost() == 16:
                        return

                    cost_to_router_row_received_from = 16
                    costFoundFlag = False
                    for current_row in self.routing_table.getRoutingTable():
                        if int(current_row.getDestId()) == int(sending_router_id):
                            cost_to_router_row_received_from = current_row.getLinkCost()
                            costFoundFlag = True

                    if destination_router == old_row.getDestId() and costFoundFlag:
                        # Process to see if new route is quicker than old, then add
                        prelim_dist = int(cost_to_router_row_received_from) + new_distance

                        if prelim_dist < old_row.getLinkCost():
                            new_row.updateLinkCost(prelim_dist)
                            new_row.updateNextHopId(sending_router_id)
                            new_row.updateLearntFrom(sending_router_id)
                            new_row.updateNextHopPort(port_to_send)
                            new_row.setHasBeenChanged()

                            self.reset_timers_of_dest(new_row.getDestId())

                            if new_row not in self.routing_table.getRoutingTable():
                                self.routing_table.removeToSwap(destination_router)
                                self.routing_table.addToRoutingTable(new_row)
                                print("Added new route -
    > {0}, $ = {1}".format(new_row.getDestId(), new_row.getLinkCost()))
                                print("Removed old entry from the routing table")
                                print(self.routing_table.getPrettyTable())

                    entryExists = False
                    for current_row in self.routing_table.getRoutingTable():
                        if current_row.getDestId() == new_row.getDestId():
                            entryExists = True

                    if not entryExists:
                        if not cost_to_router_row_received_from + new_row.getLinkCost() > 15:
                            new_row.updateLinkCost(cost_to_router_row_received_from + new_row.getLinkCost(
    ))
                            new_row.setHasBeenChanged()
                            new_row.updateLearntFrom(sending_router_id)
                            new_row.updateNextHopId(sending_router_id)
                            new_row.updateNextHopPort(port_to_send)
                            self.routing_table.addToRoutingTable(new_row)
                            print("Adding new neighbour ", new_row.getDestId())
                            print(self.routing_table.getPrettyTable())

            def triggered_update(self):
                print("Calling triggered update...", self.triggered_update_cooldown_timer_value)
                print(self.routing_table.getPrettyTable())

                output_list = self.output_ports.split(", ")
                for entry in output_list:
                    entry = entry.split('-')
                    outbound_router_id = entry[2]
                    entry = entry[0]
                    portToSend = int(entry)

                    # Don't send to self
                    if portToSend != 0:
```

```
256.                         tableToSend = copy.deepcopy(self.routing_table)
257.                         # Remove entries that haven't changed
258.                         for Row in tableToSend.getRoutingTable():
259.                             if not Row.hasChanged():
260.                                 tableToSend.removeToSwap(Row.getDestId())
261.                             if int(outbound_router_id) == int(Row.getLearntFromRouter()):
262.                                 Row.updateLinkCost(16)
263.
264.                         packetToSend = rip_packet.RIPPacket(1, self.routing_id, tableToSend)
265.                         pickledPacketToSend = pickle.dumps(packetToSend)
266.                         self.input_sockets_list[0].sendto(pickledPacketToSend, ("127.0.0.1", portToSend))
267.
268.             # Set the changed flag to false in our current routing table
269.             for ourRow in self.routing_table.getRoutingTable():
270.                 ourRow.resetChanged()
271.
272.         def periodic_update(self):
273.
274.             output_list = self.output_ports.split(", ")
275.             for entry in output_list:
276.                 entry = entry.split('-')
277.                 outbound_router_id = entry[2]
278.                 entry = entry[0]
279.                 portToSend = int(entry)
280.
281.                 # Dont send to self
282.                 if portToSend != 0:
283.                     tableToSend = copy.deepcopy(self.routing_table)
284.
285.                     for row in tableToSend.getRoutingTable():
286.                         if int(outbound_router_id) == int(row.getLearntFromRouter()):
287.                             row.updateLinkCost(16)
288.
289.                     packetToSend = rip_packet.RIPPacket(1, self.routing_id, tableToSend)
290.                     pickledPacketToSend = pickle.dumps(packetToSend)
291.                     self.input_sockets_list[0].sendto(pickledPacketToSend, ("127.0.0.1", portToSend))
292.
293.             print("My Routing Table")
294.             print(self.routing_table.getPrettyTable())
295.
296.         def config_file_check(self, data):
297.             """The most graceful check to see if the config file has all required attributes."""
298.             try:
299.                 self.routing_id = data["router-id"]
300.                 if int(self.routing_id) < 1 or int(self.routing_id) > 64000:
301.                     raise ValueError("Invalid router ID(s). Exiting...")
302.             except KeyError:
303.                 print("Router id not specified in config file. Exiting...")
304.                 exit(0)
305.             try:
306.                 self.input_ports = data["input-ports"]
307.                 for port in self.input_ports:
308.                     if port < 1024 or port > 64000:
309.                         raise ValueError("Input port number(s) out of range. Exiting...")
310.             except KeyError:
311.                 print("Input ports not specified in config file. Exiting...")
312.                 exit(0)
313.             try:
314.                 self.output_ports = data["outputs"]
315.                 output_test_list = self.output_ports.split(', ')
316.                 for entry in output_test_list:
317.                     port = entry.split('-')
318.                     port = int(port[0])
319.                     if port < 1024 or port > 64000:
```

```
320.                             raise ValueError("Output port(s) out of range. Exiting...")
321.                 except KeyError:
322.                     print("Output ports not specified in config file. Exiting...")
323.                     exit(0)
324.
325.
326.         if __name__ == "__main__":
327.             config_file_name = sys.argv[1]
328.             router = RipDemon(config_file_name, 3, True, 7, 15)
329.             router.run()
```

routing_table.py

```
330.         import routing_row, rip_route
331.
332.
333.         class RoutingTable(object):
334.
335.             def __init__(self, data):
336.                 self.routing_id = data["router-id"]
337.                 self.input_ports = data["input-ports"]
338.                 self.output_ports = data["outputs"]
339.                 self.table = []
340.                 self.neighbourTimers = []
341.                 self.populateTable()
342.                 self.neighbours = []
343.                 self.populateNeighbours()
344.
345.             def addOneFromConfig(self):
346.                 output_list = self.output_ports.split(', ')
347.                 for i in output_list:
348.                     values = i.split('-')
349.                     nextHopPort = values[0]
350.                     linkCost = values[1]
351.                     destId = values[2]
352.                     learnedFrom = 0  # As it was learned from ConfigFile
353.                     row = routing_row.RoutingRow(nextHopPort, destId, linkCost, destId, learnedFrom)
354.                     if row not in self.table:
355.                         self.addToRoutingTable(row)
356.
357.             def populateTable(self):
358.                 """Populate the routing table with the information learned from the Configuration files"""
359.
360.                 output_list = self.output_ports.split(', ')
361.
362.                 for i in output_list:
363.                     values = i.split('-')
364.                     nextHopPort = values[0]
365.                     linkCost = values[1]
366.                     destId = values[2]
367.                     learnedFrom = 0  # As it was learned from ConfigFile
368.                     row = routing_row.RoutingRow(nextHopPort, destId, linkCost, destId, learnedFrom)
369.                     self.addToRoutingTable(row)
370.
371.             def populateNeighbours(self):
372.                 output_list = self.output_ports.split(', ')
373.                 for i in output_list:
374.                     values = i.split('-')
375.                     destId = values[2]
376.                     self.neighbours.append(destId)
377.                     self.neighbourTimers.append(rip_route.Route(destId))
378.
```

```
379.            def getNeighbours(self):
380.                return self.neighbours
381.
382.            def getPrettyTable(self):
383.                for foundRow in self.table:
384.                    if foundRow is not None:
385.                        print("-
    > {0.destId}, $ = {0.linkCost}, Next hop ID: {0.nextHopId}, Learned from: {0.learnedFrom}".format(foundRow
    ))
386.
387.            def getRoutingTable(self):
388.                return self.table
389.
390.            def getRoutesWithTimers(self):
391.                return self.neighbourTimers
392.
393.            def getRouterId(self):
394.                return self.routing_id
395.
396.            def getInputPorts(self):
397.                return self.input_ports
398.
399.            def getOutputPorts(self):
400.                return self.output_ports
401.
402.            def addToRoutingTable(self, new_row):
403.                if not isinstance(new_row, routing_row.RoutingRow):
404.                    raise TypeError("Non routing-row provided as arg")
405.
406.                self.table.append(new_row)
407.
408.            def removeToSwap(self, destId):
409.                index = 0
410.                for row in self.table:
411.                    if row.row_as_list()[2] == int(destId):
412.                        del self.table[index]
413.                    index += 1
414.
415.            def removeFromRoutingTable(self, destId):
416.                for Row in self.table:
417.                    if int(Row.getDestId()) == int(destId) or int(Row.getNextHopId()) == int(destId) or \

418.                            (Row.getLearntFromRouter()) == int(destId):
419.                        self.table.remove(Row)
```

routing_row.py

```
1.  class RoutingRow(object):
2.
3.      def __init__(self, nextHop, nextHopId, linkCost, destId, learnedFrom):
4.          self.nextHopPort = int(nextHop)
5.          self.linkCost = int(linkCost)
6.          self.destId = int(destId)
7.          self.nextHopId = int(nextHopId)
8.          self.changed = False
9.          self.learnedFrom = int(learnedFrom)
10.
11.     def __repr__(self):
12.         return "{0.nextHopPort} {0.linkCost} {0.destId} {0.nextHopId} {0.learnedFrom}".format(self)
13.
14.     def __eq__(self, other):
15.         return self.__dict__ == other.__dict__
16.
```

```
    def hasChanged(self):
        return self.changed

    def resetChanged(self):
        self.changed = False

    def setHasBeenChanged(self):
        self.changed = True

    def getNextHopPort(self):
        return self.nextHopPort

    def getLinkCost(self):
        return self.linkCost

    def getDestId(self):
        return self.destId

    def getNextHopId(self):
        return self.nextHopId

    def getLearntFromRouter(self):
        return self.learnedFrom

    def updateLinkCost(self, cost):
        self.linkCost = cost

    def updateNextHopId(self, hop):
        self.nextHopId = hop

    def updateNextHopPort(self, port):
        self.nextHopPort = port

    def updateLearntFrom(self, id):
        self.learnedFrom = id
    def row_as_list(self):
        return [self.nextHopPort, self.linkCost, self.destId, self.nextHopId, self.learnedFrom]
```

rip_packet.py

```
class RIPPacket(object):
    def __init__(self, command, router_id, rip_entries):
        self.command = command
        self.version = 2
        self.router_id = router_id
        self.rip_entries = rip_entries

    def getCommand(self):
        return self.command

    def getVersion(self):
        return self.version

    def getRouterId(self):
        return self.router_id

    def getRIPEntries(self):
        return self.rip_entries
```

rip_route.py

```
class Route(object):

    def __init__(self, destId):
        self.destId = destId
        self.timeoutTime = 0
        self.hasTimeout = False
        self.garbageCollectionTime = 0

    def __repr__(self):
        return "-> {0.destId} {0.hasTimeout} Time: {0.timeoutTime}\n".format(self)

    def setRouteAsTimedOut(self):
        self.hasTimeout = True

    def hasTimedOut(self):
        return self.hasTimeout

    def incrementTimeoutTime(self):
        self.timeoutTime += 1

    def incrementGarbageCollectionTime(self):
        self.garbageCollectionTime += 1

    def getTimeoutTime(self):
        return self.timeoutTime

    def getGarbageCollectionTime(self):
        return self.garbageCollectionTime

    def resetTime(self):
        self.timeoutTime = 0
        self.garbageCollectionTime = 0

    def getDestId(self):
        return self.destId
```