# 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

14. **class** RipDemon(threading.Thread):
16. """Creates an instance of a router that implements the RIP routing Daemon.
18. Router can be initialised in the command line with:
20. python rip\_demon.py configX.json
22. Where X is the config file to be used (1-10)
23. """
25. **def** \_\_init\_\_(self, filename, intervalBetweenMessages=1, random=False, timeoutPeriod=15, garbageCollectionPeriod=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
38. **print**(self.routing\_table.getPrettyTable())
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
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))
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)
66. unpickledRIPReceivedPacket = pickle.loads(packet)
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()
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
81. **if** unpickledRIPReceivedPacket.getRouterId() != self.routing\_id **and** identical\_entry\_found == False:
82. self.process\_route\_entry(found\_row, unpickledRIPReceivedPacket.getRouterId(), port\_to\_send)
84. self.timer\_tick()
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
93. time.sleep(tickTime)
94. # update count value
95. self.timer\_value += 1
97. **if** self.timer\_value == self.timer\_interval:
98. self.ready\_for\_periodic\_update = True
99. self.timer\_value = 0
101. **if** self.ready\_for\_periodic\_update:
102. self.periodic\_update()
103. self.ready\_for\_periodic\_update = False
105. self.check\_for\_changed\_routes()
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
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()
127. **if** Route.getTimeoutTime() == self.timeout\_period:
128. self.set\_row\_as\_timed\_out(Route)
130. **def** check\_for\_changed\_routes(self):
131. **for** Row **in** self.routing\_table.getRoutingTable():
132. **if** Row.hasChanged():
133. self.ready\_for\_triggered\_update = True
134. **return**
136. neighbour\_rows = []
137. non\_neighbour\_rows = copy.deepcopy(self.routing\_table.getRoutingTable())
139. **for** Route **in** self.routing\_table.getRoutesWithTimers():
140. **for** Row **in** self.routing\_table.getRoutingTable():
141. **if** int(Route.getDestId()) == int(Row.getDestId()):
142. neighbour\_rows.append(Row)
143. **if** int(Route.getDestId()) == int(Row.getDestId()) **and** **not** Route.hasTimedOut() **and** Row.getLinkCost() == 16:
144. self.routing\_table.removeFromRoutingTable(Route.getDestId())
145. **return**
147. **for** Row **in** neighbour\_rows:
148. non\_neighbour\_rows.remove(Row)
150. **for** Row **in** non\_neighbour\_rows:
151. **if** Row.getLinkCost() == 16:
152. self.routing\_table.removeFromRoutingTable(Row.getDestId())
154. **def** set\_row\_as\_timed\_out(self, route):
155. route.setRouteAsTimedOut()
156. **for** Row **in** self.routing\_table.getRoutingTable():
158. **if** int(Row.getDestId()) == int(route.getDestId()):
159. Row.updateLinkCost(16)
160. Row.setHasBeenChanged()
162. **def** reset\_timers\_of\_dest(self, destId):
163. **for** Route **in** self.routing\_table.getRoutesWithTimers():
164. **if** int(Route.getDestId()) == int(destId):
165. Route.resetTime()
167. **def** process\_route\_entry(self, new\_row, sending\_router\_id, port\_to\_send):
168. # If there's no entry for the senders id
169. flag = False
170. **for** row **in** self.routing\_table.getRoutingTable():
171. **if** int(row.getDestId()) == int(sending\_router\_id):
172. flag = True
173. **break**
174. **if** **not** flag:
175. self.routing\_table.addOneFromConfig()
177. # If next hop port is one of your own, skip this entry
178. **if** new\_row.getNextHopPort() **in** self.routing\_table.getInputPorts():
179. **return**
180. # If row already exists exactly, skip this entry also
181. **if** new\_row **in** self.routing\_table.getRoutingTable():
182. **return**
184. destination\_router = new\_row.getDestId()
185. new\_distance = new\_row.getLinkCost()
187. # If router receives a update to a link with metric 16 (i.e that link is down)
188. **if** new\_distance == 16:
189. **for** row **in** self.routing\_table.getRoutingTable():
190. **if** row.getDestId() == destination\_router:
191. row.updateLinkCost(16)
192. row.setHasBeenChanged()
193. **break**
195. **for** old\_row **in** self.routing\_table.getRoutingTable():
196. # Make the route non modifiable when it hits 16
197. **if** old\_row.getLinkCost() == 16:
198. **return**
200. cost\_to\_router\_row\_received\_from = 16
201. costFoundFlag = False
202. **for** current\_row **in** self.routing\_table.getRoutingTable():
203. **if** int(current\_row.getDestId()) == int(sending\_router\_id):
204. cost\_to\_router\_row\_received\_from = current\_row.getLinkCost()
205. costFoundFlag = True
207. **if** destination\_router == old\_row.getDestId() **and** costFoundFlag:
208. # Process to see if new route is quicker than old, then add
209. prelim\_dist = int(cost\_to\_router\_row\_received\_from) + new\_distance
211. **if** prelim\_dist < old\_row.getLinkCost():
212. new\_row.updateLinkCost(prelim\_dist)
213. new\_row.updateNextHopId(sending\_router\_id)
214. new\_row.updateLearntFrom(sending\_router\_id)
215. new\_row.updateNextHopPort(port\_to\_send)
216. new\_row.setHasBeenChanged()
218. self.reset\_timers\_of\_dest(new\_row.getDestId())
220. **if** new\_row **not** **in** self.routing\_table.getRoutingTable():
221. self.routing\_table.removeToSwap(destination\_router)
222. self.routing\_table.addToRoutingTable(new\_row)
223. **print**("Added new route -> {0}, $ = {1}".format(new\_row.getDestId(), new\_row.getLinkCost()))
224. **print**("Removed old entry from the routing table")
225. **print**(self.routing\_table.getPrettyTable())
227. entryExists = False
228. **for** current\_row **in** self.routing\_table.getRoutingTable():
229. **if** current\_row.getDestId() == new\_row.getDestId():
230. entryExists = True
232. **if** **not** entryExists:
233. **if** **not** cost\_to\_router\_row\_received\_from + new\_row.getLinkCost() > 15:
234. new\_row.updateLinkCost(cost\_to\_router\_row\_received\_from + new\_row.getLinkCost())
235. new\_row.setHasBeenChanged()
236. new\_row.updateLearntFrom(sending\_router\_id)
237. new\_row.updateNextHopId(sending\_router\_id)
238. new\_row.updateNextHopPort(port\_to\_send)
239. self.routing\_table.addToRoutingTable(new\_row)
240. **print**("Adding new neighbour ", new\_row.getDestId())
241. **print**(self.routing\_table.getPrettyTable())
243. **def** triggered\_update(self):
244. **print**("Calling triggered update...", self.triggered\_update\_cooldown\_timer\_value)
245. **print**(self.routing\_table.getPrettyTable())
247. output\_list = self.output\_ports.split(", ")
248. **for** entry **in** output\_list:
249. entry = entry.split('-')
250. outbound\_router\_id = entry[2]
251. entry = entry[0]
252. portToSend = int(entry)
254. # Don't send to self
255. **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)
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))
268. # Set the changed flag to false in our current routing table
269. **for** ourRow **in** self.routing\_table.getRoutingTable():
270. ourRow.resetChanged()
272. **def** periodic\_update(self):
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)
281. # Dont send to self
282. **if** portToSend != 0:
283. tableToSend = copy.deepcopy(self.routing\_table)
285. **for** row **in** tableToSend.getRoutingTable():
286. **if** int(outbound\_router\_id) == int(row.getLearntFromRouter()):
287. row.updateLinkCost(16)
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))
293. **print**("My Routing Table")
294. **print**(self.routing\_table.getPrettyTable())
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)

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**

1. **import** routing\_row, rip\_route

4. **class** RoutingTable(object):
6. **def** \_\_init\_\_(self, data):
7. self.routing\_id = data["router-id"]
8. self.input\_ports = data["input-ports"]
9. self.output\_ports = data["outputs"]
10. self.table = []
11. self.neighbourTimers = []
12. self.populateTable()
13. self.neighbours = []
14. self.populateNeighbours()
16. **def** addOneFromConfig(self):
17. output\_list = self.output\_ports.split(', ')
18. **for** i **in** output\_list:
19. values = i.split('-')
20. nextHopPort = values[0]
21. linkCost = values[1]
22. destId = values[2]
23. learnedFrom = 0  # As it was learned from ConfigFile
24. row = routing\_row.RoutingRow(nextHopPort, destId, linkCost, destId, learnedFrom)
25. **if** row **not** **in** self.table:
26. self.addToRoutingTable(row)
28. **def** populateTable(self):
29. """Populate the routing table with the information learned from the Configuration files"""
31. output\_list = self.output\_ports.split(', ')
33. **for** i **in** output\_list:
34. values = i.split('-')
35. nextHopPort = values[0]
36. linkCost = values[1]
37. destId = values[2]
38. learnedFrom = 0  # As it was learned from ConfigFile
39. row = routing\_row.RoutingRow(nextHopPort, destId, linkCost, destId, learnedFrom)
40. self.addToRoutingTable(row)
42. **def** populateNeighbours(self):
43. output\_list = self.output\_ports.split(', ')
44. **for** i **in** output\_list:
45. values = i.split('-')
46. destId = values[2]
47. self.neighbours.append(destId)
48. self.neighbourTimers.append(rip\_route.Route(destId))
50. **def** getNeighbours(self):
51. **return** self.neighbours
53. **def** getPrettyTable(self):
54. **for** foundRow **in** self.table:
55. **if** foundRow **is** **not** None:
56. **print**("-> {0.destId}, $ = {0.linkCost}, Next hop ID: {0.nextHopId}, Learned from: {0.learnedFrom}".format(foundRow))
58. **def** getRoutingTable(self):
59. **return** self.table
61. **def** getRoutesWithTimers(self):
62. **return** self.neighbourTimers
64. **def** getRouterId(self):
65. **return** self.routing\_id
67. **def** getInputPorts(self):
68. **return** self.input\_ports
70. **def** getOutputPorts(self):
71. **return** self.output\_ports
73. **def** addToRoutingTable(self, new\_row):
74. **if** **not** isinstance(new\_row, routing\_row.RoutingRow):
75. **raise** TypeError("Non routing-row provided as arg")
77. self.table.append(new\_row)
79. **def** removeToSwap(self, destId):
80. index = 0
81. **for** row **in** self.table:
82. **if** row.row\_as\_list()[2] == int(destId):
83. **del** self.table[index]
84. index += 1
86. **def** removeFromRoutingTable(self, destId):
87. **for** Row **in** self.table:
88. **if** int(Row.getDestId()) == int(destId) **or** int(Row.getNextHopId()) == int(destId) **or** \
89. (Row.getLearntFromRouter()) == int(destId):
90. self.table.remove(Row)

**routing\_row.py**

1. **class** RoutingRow(object):
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)
11. **def** \_\_repr\_\_(self):
12. **return** "{0.nextHopPort} {0.linkCost} {0.destId} {0.nextHopId} {0.learnedFrom}".format(self)
14. **def** \_\_eq\_\_(self, other):
15. **return** self.\_\_dict\_\_ == other.\_\_dict\_\_
17. **def** hasChanged(self):
18. **return** self.changed
20. **def** resetChanged(self):
21. self.changed = False
23. **def** setHasBeenChanged(self):
24. self.changed = True
26. **def** getNextHopPort(self):
27. **return** self.nextHopPort
29. **def** getLinkCost(self):
30. **return** self.linkCost
32. **def** getDestId(self):
33. **return** self.destId
35. **def** getNextHopId(self):
36. **return** self.nextHopId
38. **def** getLearntFromRouter(self):
39. **return** self.learnedFrom
41. **def** updateLinkCost(self, cost):
42. self.linkCost = cost
44. **def** updateNextHopId(self, hop):
45. self.nextHopId = hop
47. **def** updateNextHopPort(self, port):
48. self.nextHopPort = port
50. **def** updateLearntFrom(self, id):
51. self.learnedFrom = id
52. **def** row\_as\_list(self):
53. **return** [self.nextHopPort, self.linkCost, self.destId, self.nextHopId, self.learnedFrom]

**rip\_packet.py**

1. **class** RIPPacket(object):
2. **def** \_\_init\_\_(self, command, router\_id, rip\_entries):
3. self.command = command
4. self.version = 2
5. self.router\_id = router\_id
6. self.rip\_entries = rip\_entries
8. **def** getCommand(self):
9. **return** self.command
11. **def** getVersion(self):
12. **return** self.version
14. **def** getRouterId(self):
15. **return** self.router\_id
17. **def** getRIPEntries(self):
18. **return** self.rip\_entries

**rip\_route.py**

1. **class** Route(object):
3. **def** \_\_init\_\_(self, destId):
4. self.destId = destId
5. self.timeoutTime = 0
6. self.hasTimeout = False
7. self.garbageCollectionTime = 0
9. **def** \_\_repr\_\_(self):
10. **return** "-> {0.destId} {0.hasTimeout} Time: {0.timeoutTime}\n".format(self)
12. **def** setRouteAsTimedOut(self):
13. self.hasTimeout = True
15. **def** hasTimedOut(self):
16. **return** self.hasTimeout
18. **def** incrementTimeoutTime(self):
19. self.timeoutTime += 1
21. **def** incrementGarbageCollectionTime(self):
22. self.garbageCollectionTime += 1
24. **def** getTimeoutTime(self):
25. **return** self.timeoutTime
27. **def** getGarbageCollectionTime(self):
28. **return** self.garbageCollectionTime
30. **def** resetTime(self):
31. self.timeoutTime = 0
32. self.garbageCollectionTime = 0
34. **def** getDestId(self):
35. **return** self.destId