

"""

COSC364 2022-S1 Assignment: RIP routing
Authors: MENG ZHANG (71682325), ZHENG CHAO (21671773)
File: rip_router.py

"""

Import Modules

import time

import random

from datetime import datetime

from network_interface import Interface

from forwarding_route import Route

from rip_packet import RipPacket, RipEntry

from IO_formatter import routing_table_formatter

Router Class

class Router:

"""

An object that simulates a router with rip protocol

"""

Class attributes

INFINITY = 16

REGULAR_TIMER_OFFSET = 1.0

def __init__(self, router_id,
 inputs, outputs,
 period, timeout):

"""

the __ attributes are private attributes which can only be
accessed by getter outside of class.*

Parameters:

router_id: an integer, i.e. 1, 2, 3, etc

inputs: a list of integers, i.e. [5001, 5002, 5003]

outputs: a dictionary of dictionaries, i.e.

{6010(port): {'metric': 1, 'router_id': 1},

6030(port): {'metric': 2, 'router_id': 3},

... : {...}

}

period: an integer

timeout: an integer

"""

Instance attributes

self.__router_id = router_id

self.__split_horizon_poison_reverse = True

self.__input_ports = inputs

self.__output_ports = outputs

self.__regular_advertise_timer = time.time()

self.__default_period = period

self.__period = period

self.__trigger_advertise_timer = time.time()

self.__default_triggered_updates_period = period / 6

self.__triggered_updates_period = period / 6

self.__timeout_check_timer = time.time()

self.__timeout = timeout

self.__garbage_collection_time = period * 4

self.__interface = None

self.__routing_table = {}

Initialisation

self.init_interface(inputs)

self.init_routing_table()

self.random_offset_period()

```
64 def get_router_id(self):
65     """
66     router_id getter
67     """
68     return self.__router_id
69
70
71 def set_router_id(self, new_id):
72     """
73     router_id setter
74     """
75     self.__router_id = new_id
76
77
78 def get_input_ports(self):
79     """
80     router input_ports getter
81     """
82     return self.__input_ports
83
84
85 def set_input_ports(self, new_inputs):
86     """
87     router input_ports setter
88     """
89     self.__input_ports = new_inputs
90     self.init_interface(new_inputs)
91
92
93 def get_output_ports(self):
94     """
95     router output_ports getter
96     """
97     return self.__output_ports
98
99
100 def set_output_ports(self, new_outputs):
101     """
102     router output_ports setter
103     """
104     self.__output_ports = new_outputs
105
106
107 def get_period(self):
108     """
109     router period getter
110     """
111     return self.__period
112
113
114 def set_period(self, new_period):
115     """
116     router period setter
117     """
118     self.__period = new_period
119
120
121 def get_timeout(self):
122     """
123     router timeout getter
124     """
125     return self.__timeout
126
127
128 def set_timeout(self, new_timeout):
129     """
130     router timeout setter
131
```

```

132     """
133     self.__timeout = new_timeout
134
135
136 def get_interface(self):
137     """
138     router interface getter
139     """
140     return self.__interface
141
142
143 def get_routing_table(self):
144     """
145     router routing_table getter
146     """
147     return self.__routing_table
148
149
150 def print_routing_table(self):
151     """
152     Print the current self.__routing_table
153     """
154     print(routing_table_formatter(self.__router_id,
155                                   self.__routing_table))
156
157
158 def random_offset_period(self):
159     """
160     randomize self.__period +- TIMER_OFFSET
161     """
162     self.__period = self.__default_period + \
163         random.uniform(-self.REGULAR_TIMER_OFFSET, \
164                       +self.REGULAR_TIMER_OFFSET)
165     print("Set Router regular update period to " + \
166           f"{self.__period:.2f}")
167
168
169 def random_triggered_updates_period(self):
170     """
171     randomize self.__triggered_updates_period
172     """
173     self.__triggered_updates_period = \
174         self.__default_triggered_updates_period - \
175         random.uniform(0, 0.4)
176     print("Set Router triggered update period to " + \
177           f"{self.__triggered_updates_period:.2f}")
178
179
180 def init_interface(self, ports):
181     """
182     Create a new Interface object and set it as the default
183     interface for the current Router object
184     """
185     self.__interface = Interface(ports)
186
187
188 def init_routing_table(self):
189     """
190     Initialise the __routing_table attribute
191
192     Route object format:
193     route.next_hop: 2,
194     route.metric: 1,
195     route.timeout: 1234,
196     route.garbage_collect_time: None(default)
197     state: 'active'(default)
198     """
199

```

```

200 # Create a new Route object to router itself
201 self_route = Route('-', 0, None)
202 self.__routing_table[self.__router_id] = self_route
203
204
205 #-----
206 # Above is the init implementation
207 #-----
208
209
210 def advertise_all_routes_periodically(self):
211     """
212     Call advertise_all_routes() periodcally by self.__period
213
214     Use random.random() to calculate offset for self.__period
215     in order to avoid synchronized update messages which can lead
216     to unnecessary collisions on broadcast networks.
217     """
218     now = time.time()
219     if now - self.__regular_advertise_timer >= self.__period:
220         self.advertise_routes('all')
221         self.print_routing_table()
222         self.__regular_advertise_timer = now
223         self.random_offset_period()
224
225
226 def advertise_updated_routes(self):
227     """
228     advertise the updated routes to all neighbours
229     """
230     now = time.time()
231     if now - self.__trigger_advertise_timer >= \
232         self.__triggered_updates_period:
233         self.advertise_routes('update')
234         self.print_routing_table()
235         self.__trigger_advertise_timer = now
236         self.random_triggered_updates_period()
237
238
239 def advertise_routes(self, mode):
240     """
241     parameter:
242     mode: a string 'all' / 'update'
243     get the latest advertising rip packet from
244     update_packet() & triggered_packet() methods and
245     advertise the packet to all the neighbours (ouput ports)
246
247     need to add a parameter for updata_packet/triggered_packet
248     """
249     try:
250         ports_num = len(self.__output_ports)
251         if ports_num < 1:
252             raise ValueError("No output port/socket available")
253         for dest_port, metric_id in self.__output_ports.items():
254             packet = self.update_packet(metric_id['router_id'], mode)
255             if packet is None:
256                 print("A packet without entry. Stop Sending")
257                 return
258             self.__interface.send(packet, dest_port)
259             current_time = datetime.now().strftime("%H:%M:%S.%f")[:-4]
260             if mode == 'all':
261                 message = "Sends all routes to Router"
262             else:
263                 message = 'Sends triggred update to Router'
264             print(message +
265                   f"{{metric_id['router_id']}} " +
266                   f"{{[{dest_port}] at {current_time}}}")

```

```

268     # clear flags of "update"
269     for route in self.__routing_table.values():
270         if mode == 'update' and route.state == 'updated':
271             route.state = 'active'
272     except ValueError as error:
273         print(error)
274
275
276 def update_packet(self, receiver_id, mode):
277     """
278     parameter:
279     receiver_port
280
281     Process the current routing table data and convert it into
282     a rip format packet for advertise_all_routes() method
283     """
284     # Create RipEntries for all the routes
285     entries = []
286     for dest, route in self.__routing_table.items():
287
288         if mode == "update" and route.state == "active":
289             continue
290         metric = route.metric
291         # split_horizon_poison_reverse
292         if self.__split_horizon_poison_reverse and \
293             route.next_hop == receiver_id:
294             metric = self.INFINITY
295         entry = RipEntry(dest, metric)
296         entries.append(entry)
297
298     # Create RipPacket
299     packet = RipPacket(entries, self.__router_id)
300     packet_bytes = packet.packet_bytes()
301     return packet_bytes
302
303
304 #-----
305 # Above is sender implementation
306 #-----
307
308
309 def receive_routes(self):
310     """
311     Receive the routes update from neighbours (input ports)
312
313     The implementation is in a while loop and should be called with
314     a separate thread from the main thread
315     """
316     # The __interface only listen to the input ports
317     # print(f"Listening to ports at {time.ctime()}")
318     packets_list = self.__interface.receive()
319     for raw_packet in packets_list:
320         self.process_received_packet(raw_packet)
321
322
323 def process_received_packet(self, raw_packet):
324     """
325     Process the received packet and call update_routing_table()
326     if necessary
327
328     Parameter: packet
329     an array of bytes
330     """
331     # Check if raw_packet valid in RipPacket and RipEntry classes
332     # Process the raw_packet if valid,
333     # and return (True, RipPacket object)
334     # otherwise, return (False, router_id)
335

```

```

336 is_valid, rip_packet = RipPacket.decode_packet(raw_packet)
337 if is_valid:
338     # update routing_table if incoming packet is valid
339     print(f'Received update from Router {rip_packet.router_id}')
340     self.update_routing_table(rip_packet)
341 else:
342     # drop the packet if incoming packet is invalid
343     print(f'Drop invalid packet from Router {rip_packet}')
344
345
346 def update_routing_table(self, rip_packet):
347     """
348     check all the entries in rip_packet object, and update current
349     routing table if necessary
350
351     Parameter:
352     rip_packet: a valid rip_packet object
353
354     Return: boolean
355     return True if new route added, otherwise False
356     """
357     # get metric from sender
358     sender_id = rip_packet.router_id
359     metric_to_sender = None
360     for neighbour in self.__output_ports.values():
361         if neighbour['router_id'] == sender_id:
362             metric_to_sender = neighbour['metric']
363     for entry in rip_packet.entries:
364         # update the metric for each entry
365         # by adding the metric to sender
366         # metric = min(metric + metric_to_sender, 16(infinity))
367         updated_metric = min(entry.metric + metric_to_sender,
368                             self.INFINITY)
369         # if route to dest is unavailable in __routing_table
370         if updated_metric != self.INFINITY and \
371             not entry.dest in self.__routing_table.keys():
372             self.__routing_table[entry.dest] = \
373                 Route(sender_id, updated_metric, time.time())
374             # Triggered update for new route
375             # self.__routing_table[entry.dest].state = 'updated'
376             # print("Triggerd update for new route")
377             # self.advertise_updated_routes()
378         elif entry.dest in self.__routing_table:
379             self.update_availabe_route(entry,
380                                       updated_metric,
381                                       sender_id)
382
383
384
385 def update_availabe_route(self, entry, updated_metric, sender_id):
386     """
387     Parameters:
388     entry: a RipEntry object
389     sender_id: the router id from which the entry is sent
390     """
391     # if route to dest is available in __routing_table
392
393     # 1. if packet is from the same router as
394     # existing router, reinitialize the timeout anyway
395     from_same_router = sender_id == \
396         self.__routing_table[entry.dest].next_hop
397     is_timeout = not \
398         self.__routing_table[entry.dest].garbage_collect_time is \
399         None
400     if from_same_router:
401         self.__routing_table[entry.dest].timeout = time.time()
402
403

```

```

404 # 2. compare metrics
405 new_metric = updated_metric
406 old_metric = self.__routing_table[entry.dest].metric
407 have_differnt_metrics = new_metric != old_metric
408 is_lower_new_metric = new_metric < old_metric
409 is_almost_timeout = \
410     not self.__routing_table[entry.dest].timeout is None and \
411     not is_timeout and \
412     (time.time() - self.__routing_table[entry.dest].timeout) \
413     >= self.__timeout / 2
414
415 if from_same_router and have_differnt_metrics:
416     self.__routing_table[entry.dest].metric = new_metric
417     if not is_timeout and new_metric == self.INFINITY:
418         self.__routing_table[entry.dest].garbage_collect_time \
419             = time.time()
420         # Triggered update for invalid route
421         self.__routing_table[entry.dest].state = 'dying'
422         print("triggered update for invalid route")
423         self.advertise_updated_routes()
424     elif is_timeout:
425         self.__routing_table[entry.dest].garbage_collect_time \
426             = None
427         self.__routing_table[entry.dest].state = 'active'
428
429 elif is_lower_new_metric:
430     self.__routing_table[entry.dest].metric = new_metric
431     self.__routing_table[entry.dest].next_hop = sender_id
432     self.__routing_table[entry.dest].timeout = time.time()
433     if is_timeout:
434         self.__routing_table[entry.dest].garbage_collect_time \
435             = None
436         self.__routing_table[entry.dest].state = 'active'
437         # Triggered update
438         # self.__routing_table[entry.dest].state = 'updated'
439         # print("triggered updated route from different router with lower metric")
440         # self.advertise_updated_routes()
441 elif not from_same_router and \
442     not have_differnt_metrics and \
443     not is_timeout and is_almost_timeout:
444     self.__routing_table[entry.dest].next_hop = sender_id
445     self.__routing_table[entry.dest].timeout = time.time()
446
447
448
449 #-----
450 # Above is receiver implementation
451 #-----
452
453 def check_timeout_entries_periodically(self):
454     """
455     call check_timeout_entries() every default_period
456     """
457     now = time.time()
458     if now - self.__timeout_check_timer >= self.__default_period:
459         self.check_timeout_entries()
460         self.__timeout_check_timer = now
461
462
463 def check_timeout_entries(self):
464     """
465     Check the timeout of each entry in __routing_table
466
467     if an entry is timeout, start its garbage_collect_time
468     """
469     current_time = datetime.now().strftime('%H:%M:%S.%f')[:-4]
470     print(f"Checking timeout entries at {current_time}")
471

```

```

472 entries_to_remove = []
473 for dest_id, entry in self.__routing_table.items():
474     if not entry.timeout is None and \
475         entry.garbage_collect_time is None and \
476         time.time() - entry.timeout >= self.__timeout:
477         entry.garbage_collect_time = time.time()
478         entry.metric = self.INFINITY
479         entry.state = 'dying'
480         # Triggered update
481         print("Triggered update for invalid route")
482         self.advertise_updated_routes()
483
484     if not entry.garbage_collect_time is None and \
485         (time.time() - entry.garbage_collect_time) \
486         >= self.__garbage_collection_time:
487         entries_to_remove.append(dest_id)
488
489 for dest_id in entries_to_remove:
490     self.__routing_table.pop(dest_id)
491     print(f"Removed timeout route to {dest_id}")
492     self.print_routing_table()
493
494
495 #-----
496 # Above is timeout and garbage_collection implementation
497 #-----
498
499 def __str__(self):
500     return ("Router: {0}\n"
501            "Input Ports: {1}\n"
502            "Output Ports: {2}\n"
503            "Period: {3}\n"
504            "Timeout: {4}").format(self.__router_id,
505                                self.__input_ports,
506                                self.__output_ports,
507                                self.__period,
508                                self.__timeout)

```