COSC364 Assignment 1

George Drummond(53243258),
Ryan Cox(64656394)

May 4, 2017

Abstract

The following discusses an implementation of the RIP routing protocol. This was a joint work by George Drummond and Ryan Cox in accordance with the course requirements of COSC364-17S1 and is a result of **equal contribution (**50%/50%**).**

Contents

1 Favourite aspects

One of the initial successes of our design was the decent degree of modulation with which we were able to develop our program. Through the *RIP_packet.py* and *writelog.py* files, we were able to achieve a reasonable level of abstraction around the tasks of packet processing and logging errors. This led to far easier reading and debugging.

Within these modulated programs, the methods for constructing and processing the RIP response packet formed a strong point in our design. The RIP response packet was represented by a python string of hex values. In accordance with this, Python's type conversion abilities were heavily utilised to allow for swift extraction of the relevant information. Again, this was able to be done "under the hood" by the helper file *RIP_packet.py*, avoiding much unnecessary indexing and such by the main program.

These good practices led to a great degree of functionality being achieved by our program. Via the tiered testing method detailed below, we were able to quickly debug and achieve a working (and pretty!) result.

2 Aspects to be improved

One aspect of the program which could have been improved was the way in which we implemented our timing procedures. Though operationally sound, these are not as aesthetically pleasing as they could be, with the precise nature of their operation obscured to the casual observer. Instead of the implicit structure of our states, it would perhaps have been better to implement an explicit finite automata by defining state variables and transition functions. This would not only have been nicer to look at but also perhaps easier to develop as we could have considered the protocol on a "state by state" basis rather than a "conditional" one.

3 Atomicity of event processing

Our program is atomic by nature in that each line is executed sequentially with nothing interrupting the program as it runs through the main "round robin" style loop. By not necessarily reacting to timers the moment they "tick over", we introduce a further small element of randomness to the timing procedures. This is beneficial to the RIP routing protocol as it helps to avoid synchronisation of updates. Every "timer event" is addressed in the loop and occurs depending on its respective timer value at that instant. Crucially, no event can interrupt another event that is running.

The blocking and waiting state constitutes the majority of the time, with each call leading to a 0.5 second wait. All other processing amounts to around 0.0025 seconds per 'loop' so the chance we miss a packet is low. That is, we are not blocking for %0.5 of the time on average.

4 Discussion

Our primary source for testing during development was our "example network" of 7 routers configured as follows.

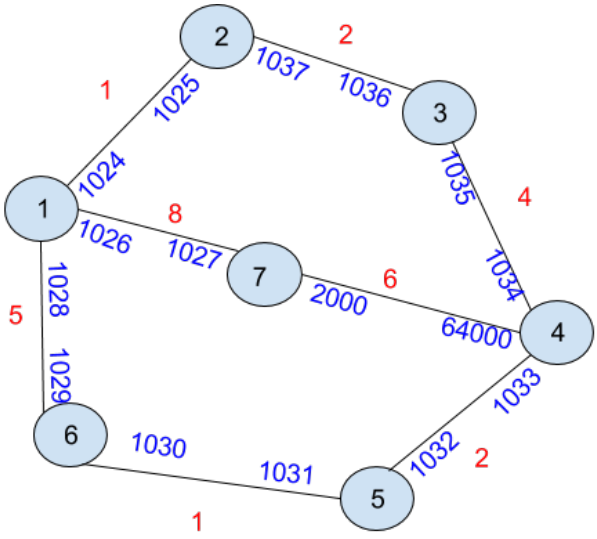

Figure 1: Example Network

Where metrics are shown in red and port numbers in blue. This network was implemented by the seven config files of the form *routerx*, $x \in \{1, 2, 3, 4, 5, 6, 7\}$ and, in its entirety, is large enough to allow for thorough testing of the RIP routing protocol. This example network configuration is also of particular developmental use as it tests multiple configuration criteria, such as involving both the smallest and largest possible accepted port numbers, checking that the range worked correctly and wasn't accidentally off by one.

During early development, we routinely ran only routing demons 1 and 2 above. This enabled us to quickly check elements of basic functionality (sending/receiving packets, updating routing tables, timeout and garbage collection etc) but obviously did not give us much insight into the correctness of our protocol implementation.

Our second, deeper stage of testing (and the one used for the majority of development) involved a greater number of routing demons and multiple topological aspects. These were namely, the routers 1,2,3 and 6 above. This gave us a far more interesting, though

still easily observable network of two "transit" routers 1 and 2 as well as two "stub" routers 3 and 6. By taking down either 3 or 6 we were able to test for the correctness of the timeout and garbage collection mechanisms as well as the propagation of link failure information in the form of triggered updates. Likewise, by taking down either 1 or 2, we were able to observe the isolation of a "stub" router (6 or 3 respectively) from the rest of the network. This stage of testing however, did stop short of the whole picture as it did not test for non-trivial path updates.

Finally a "complete run" of the network was used as our last stage of testing. Bringing up all routers allowed us to observe convergence times and also to check the validity of the shortest path information of each router. We could also, at this point, test the network's adaptation to extreme topological change. A favourite such change was to bring down routers 4 and 6. This had the joint effect of "isolating 5" (i.e making router 5 unreachable from all other routers) and also changing a plethora of shortest path information (for example D(7,3) changes from a value of 10 to 11). It was with this particular test, that a great deal of valuable debugging was achieved.

Lastly the runlog_x $x \in \{1, 2, 3, 4, 5, 6, 7\}$files generated by each routing demon instance also constituted a particularly useful tool for debugging. After being initialised by the routing demon, we could record various runtime information (such as the receiving and sending of RIP response packets) and any errors that have occurred with a timestamp. Each routing demon makes its own separate log using its ID and since the log file object was part of the routing class it was easy to write to the log from the code without having to pass around the file object.

5 Main Program

```
"""
#RIP_routing_daemon.py
#Authors: George Drummond - gmd44
#         Ryan Cox - rlc96
#Last Edit: 5/4/2017
#
#Routing demon instance participates in the version 2 RIP routing protocol.
#Demon emulates a router in a given network from the supplied config file.
#
"""

#!/usr/bin/python
import sys
import select
import socket
import random
from RIP_packet import *
from writelog import *
import time

MAX_BUFF = 600
MAX_DATA = 512
INF = 16
HOST_ID = '127.0.0.1'

# Router STATES: 0 -> Waiting for input with periodic updates
#                1 -> Needs to send a triggered update

def valid_portn(portn):
     return int(portn) in range(1024,64001)

def valid_ID(routerID):
     return int(routerID) in range(1,64001)

def valid_metric(metric):
     return int(metric) in range(0,INF+1)



class RIProuter:
     '''RIP router class'''
     def __init__(self,configFile):
          self.periodic = 0
          self.updateFlag = 0
          self.configFile = configFile
          self.parse_config()
          self.socket_setup()
          self.routingTable = RoutingTable(self.timers[1],self.timers[2]) #
    timeout and garbage considered
```

```
49          self.log = init_log(self.routerID)
50
51          print('routerID =',self.routerID)
52          print('inport numbers =',self.inPort_numbers)
53
54          print('peerInfo =',self.peerInfo)
55          print('timers =',self.timers)
56          print('table=\n',self.routingTable)
57
58
59      def socket_setup(self):
60          '''Sets up a socket with each of the given port numbers'''
61          self.inPorts = []
62          for portn in self.inPort_numbers:
63              newSocket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
64              newSocket.bind((HOST_ID, portn))
65              self.inPorts += [newSocket]
66
67
68      def close_sockets(self):
69          ''' Close all sockets'''
70          for port in self.inPorts:
71              port.close()
72
73
74      def parse_config(self):
75          ''' Parse the supplied config file'''
76          lines = self.configFile.readlines()
77          for line in lines:
78              entries = line.split(',')
79              #print(entries)
80              lineType = entries[0]
81              tail = entries[1:]
82              if lineType == 'router-id':
83                  self.set_ID(tail)
84
85              elif lineType == 'input-ports':
86                  self.set_InPort_numbers(tail)
87
88              elif lineType == 'outputs':
89                  self.set_peerInfo(tail)
90
91              elif lineType == 'timers':
92                  self.set_timers(tail)
93
94
95      def set_ID(self,tail):
96          ''' Checks and stores routerID'''
97          myID = int(tail[0])
98          if valid_ID(myID):
99              self.routerID = int(tail[0])
100
```

```
101             else:
102                 raise(IndexError('Router ID not valid'))
103
104
105     def set_InPort_numbers(self,tail):
106         ''' Checks and stores all supplied inport numbers'''
107         self.inPort_numbers = []
108         for portstring in tail:
109             port = int(portstring)
110             if (port not in self.inPort_numbers) and valid_portn(port):
111                 self.inPort_numbers += [port]
112             else:
113                 print("invalid inport port {} supplied".format(port))
114
115
116     def set_peerInfo(self,tail):
117         ''' Stores info relevent to immediate neighbours '''
118         self.peerInfo = dict()
119         for triplet in tail:
120             portN,metric,peerID = triplet.split('-')
121             if valid_portn(portN) and valid_metric(metric) and valid_ID(
    peerID):
122                 self.peerInfo[int(peerID)] = (int(portN),int(metric))
123             else:
124                 print("invalid peer info for peer {}".format(peerID))
125
126
127     def set_timers(self,tail):
128         ''' Stores supplied timer info (i.e. periodic, timeout, garbage)
    '''
129         self.timers = []
130         for entry in tail:
131             self.timers += [int(entry)]
132
133
134
135     def send_updates(self):
136         ''' Sends an update message to each neighbour'''
137         i = 0
138         for peerID in self.peerInfo.keys():
139             print("update sent to {}".format(peerID))
140             write_to_log(self.log,
141                          "Sent update to {}".format(peerID))
142             OutSock = self.inPorts[i] # use a different socket to send
    each
143
144             peerPort = self.peerInfo[peerID][0]
145             response = self.response_packet(peerID)
146
147             OutSock.sendto(response.encode('UTF-8'),(HOST_ID,peerPort))
148
149             i += 1
```

```
150
151
152     def response_packet(self, peerID):
153         ''' Construct a response packet destined to a neighboring router.
154             Suitable for a periodic or triggered update'''
155
156         packet = ""
157         packet += rip_header(self.routerID)
158         for Entry in self.routingTable:
159             # Implement split horizon with poisson reverse
160             if (Entry.nextHop == peerID) or (Entry.garbageFlag == 1):
161                 print("split horizon entry sent to {}".format(peerID))
162                 packet += RTE(TableEntry(Entry.dest, INF, Entry.nextHop
    )) # set metric to INF
163             else:
164                 packet += RTE(Entry)
165
166         return packet
167
168     def process_rip_packet(self, packet):
169         ''' Processes a RIP response packet'''
170
171         (peerID,RTEs) = rip_packet_info(packet)
172
173
174         if not valid_ID(peerID):
175             print("[Error] peerID {} out of range".format(peerID))
176             write_to_log(self.log, "[Error] peerID {} out of range".
    format(peerID))
177
178
179         print("processing packet from {}".format(peerID))
180         cost = self.peerInfo[peerID][1]
181
182         # Consider direct link to peer Router
183         incomingEntry = self.routingTable.get_entry(peerID)
184         if incomingEntry is None:
185             print("added directlink entry to router {}".format(peerID))
186             self.routingTable.add_entry(peerID, cost, peerID)
187         else:
188             incomingEntry.metric = cost
189             incomingEntry.timeout = 0 # Reinitialise timeout for this
    link
190             incomingEntry.garbageFlag = 0
191             incomingEntry.garbage = 0
192
193
194
195         for RTE in RTEs:
196             self.processRTE(RTE, peerID, cost)
197
198
```

```
199     def processRTE(self, RTE, peerID, cost):
200         '''processes an RTE of a RIP responce packet from a peer router
    '''
201         (dest, metric) = RTE
202 
203         new_metric = min(metric + cost, INF) # update metric
204 
205         """check metric here?"""
206         currentEntry = self.routingTable.get_entry(dest)
207 
208 
209 
210         if new_metric >= INF:
211             print("Path ({},{}) from {} not processed as unreachable".
    format(dest, metric,peerID))
212             write_to_log(self.log,
213                 "Path ({},{}) from {} not processed as unreachable".
    format(dest, metric,peerID))
214                 #do something here
215 
216         if (currentEntry is None):
217             print("current route not in table")
218             if (new_metric < INF): # Add a new entry
219                 NewEntry = TableEntry(dest, new_metric, peerID)
220                 print('new Entry {}'.format(NewEntry))
221                 write_to_log(self.log,
222                     "New route added from {} to {} with Metric {}"
223                     .format(self.routerID, NewEntry, new_metric))
224 
225 
226                 self.routingTable.add_entry(dest, new_metric, peerID)
227 
228         else: # Compare to existing entry
229 
230             print("Existing entry for {}".format(dest))
231             if (currentEntry.nextHop == peerID): # Same router as
    existing route
232 
233                 currentEntry.timeout = 0 # Reinitialise timeout
234                 currentEntry.garbageFlag = 0
235                 currentEntry.garbage = 0
236 
237                 if (new_metric != currentEntry.metric):
238                     self.existing_route_update(currentEntry,
    new_metric, peerID)
239 
240 
241 
242             elif (new_metric < currentEntry.metric):
243                 print("update route to {}".format(dest))
244                 write_to_log(self.log,
245                         "Route from {} to {} updated with new Metric
```

```
    {}"
246                                .format(self.routerID, NewEntry, new_metric))
247                     self.existing_route_update(currentEntry, new_metric,
    peerID)
248
249
250
251
252
253
254
255
256     def existing_route_update(self, currentEntry, new_metric, peerID):
257         ''' updates an existing routing table entry with a new metric'''
258         currentEntry.metric = new_metric
259         print("route to {} updated to metric = {}".format(currentEntry.
    dest,new_metric))
260         currentEntry.nextHop = peerID
261
262         if (new_metric >= INF):
263             print("Triggered update flag set")
264             self.updateFlag = 1 #Set some update flag
265             currentEntry.garbageFlag = 1
266
267
268
269
270 class RoutingTable:
271     def __init__(self, timeoutMax, garbageMax):
272         self.table = []
273         self.timeoutMax = timeoutMax
274         self.garbageMax = garbageMax
275
276     def __iter__(self):
277         i = 0
278         while i < len(self.table):
279             yield(self.table[i])
280             i += 1
281
282     def __repr__(self):
283         blank = "-" * 54
284         print(blank + "\n| dest | metric | nextHop | flag | timeout |
    garbage |")
285         for Entry in self.table:
286             print("|{:>5} |{:>7} |{:>8} |{:>5} |{:>8.3f} |{:>8.3f} |".
    format(
287                     Entry.dest, Entry.metric, Entry.nextHop, Entry.
    garbageFlag,
288                     Entry.timeout, Entry.garbage))
289
290         return blank
291
```

```
292     def add_entry(self,dest, metric, nextHop):
293         self.table += [TableEntry(dest, metric, nextHop)]
294 
295     def remove_entry(self, Entry):
296         print("Entry {} removed".format(Entry))
297         self.table.remove(Entry)
298 
299     def get_entry(self, dest):
300         ''' returns required table entry if already present'''
301         i = 0
302         while i < len(self.table):
303             Entry = self.table[i]
304             if Entry.dest == dest:
305                 return Entry
306 
307             i += 1
308 
309         return None
310 
311 
312 
313 class TableEntry:
314     def __init__(self,dest, metric, nextHop):
315         self.dest = dest
316         self.metric = metric
317         self.nextHop = nextHop
318         self.garbageFlag = 0
319         self.timeout = 0
320         self.garbage = 0
321 
322     def __repr__(self):
323         return str((self.dest, self.metric,
324                     self.nextHop, self.garbageFlag, self.timeout, self.
    garbage))
325 
326 
327 
328 def main():
329     configFile = open(sys.argv[1])
330     #configFile = open("router1.conf") # Just for developement
331     router = RIProuter(configFile)
332     selecttimeout = 0.5
333     periodicWaitTime = router.timers[0]
334 
335     starttime = time.time() #Gets the start time before processing
336 
337     while(1):
338         try:
339             # Wait for at least one of the sockets to be ready for
    processing
340             print("table reads\n",router.routingTable)
341             readable, writable, exceptional = select.select(router.
```

```
    inPorts, [], router.inPorts, selecttimeout) #block for incoming packets
    for half a second
342
343             # Send triggered updates at this stage
344             if (router.updateFlag == 1):
345                 router.send_updates()
346                 router.updateFlag = 0
347
348             for sock in readable:
349
350                 packet = sock.recv(MAX_BUFF).decode('UTF-8')
351                 router.process_rip_packet(packet)
352
353             timeInc = (time.time() - starttime) #finds the time taken on
     processing
354             #print("proc time = {}".format(timeInc))
355             starttime = time.time()
356             router.periodic += timeInc
357
358             if (router.periodic >= periodicWaitTime): # Periodic update
359                 router.send_updates()
360                 #Recalculate new random wait time in [0.8*periodic,
    1.2*periodic]
361                 periodicWaitTime = random.uniform(0.8*router.timers
    [0],1.2*router.timers[0])
362
363                 router.periodic = 0 # Reset periodic timer
364                 print("Periodic update")
365
366             for Entry in router.routingTable:
367
368                 if (Entry.garbageFlag == 1):
369                     Entry.garbage += timeInc
370                     if (Entry.garbage >= router.timers[2]): # Garbage
    collection
371                         print('Removed {}'.format(Entry))
372                         write_to_log(router.log,
373                                      "[Warning] Route from {} to {}
    has been removed"
374                                      .format(router.routerID, Entry.
    dest))
375                         router.routingTable.remove_entry(Entry)
376
377                 else:
378                     Entry.timeout += timeInc
379                     if (Entry.timeout >= router.timers[1]): # timeout/
    delete event
380                         print('Timeout')
381                         write_to_log(router.log,
382                                      "[Warning] Route from {} to {}
    has timed out"
383                                      .format(router.routerID, Entry.
```

```
    dest))
384                                 Entry.metric = INF
385                                 router.updateFlag = 1 # require triggered
    update
386                                 Entry.garbageFlag = 1 # Set garbage flag
387
388             except KeyboardInterrupt: # 'Taking down' router
389                 print("Exiting program")
390                 close_log(router.log)
391                 router.close_sockets()
392                 break
393
394
395 main()
```

6 RIPpacket

```
#!/usr/bin/python

# REMEMBER bytes.hex() and bytes.fromhex()

def bytes_to_int(byte_string):
    return int.from_bytes(byte_string, byteorder='big')

#def int_to_bytes(myint,size):
    #''' converts integer to 'size' number of bytes'''
    #return (myint).to_bytes(size, byteorder='big').hex()


def int_to_bytes(myint, size):
    suffix = hex(myint)[2:]
    prefix = '0'*(2*size-len(suffix))
    return (prefix + suffix)

def rip_header(routerID):
    header = '0201' + int_to_bytes(routerID,2)
    return header

def RTE(Entry):
    zero_row =  int_to_bytes(0,4)
    s = zero_row
    s+= int_to_bytes(Entry.dest,4)
    s+= zero_row
    s+= zero_row
    s+= int_to_bytes(Entry.metric,4)
    return(s)


def rip_packet_info(packet):
    ''' Extracts relevent info from a RIP response packet'''

    RTEs = []

    peerID = int(packet[4:8],16)

    i = 8 # Start of first RTE
    while i < len(packet):

        dest = int(packet[i+8:i+16],16) # Read dest from RTE
        metric = int(packet[i+32:i+40],16) # Read metric from RTE
        RTEs += [(dest, metric)]

        i += (8*5) # Proceed to next RTE

    return(peerID,RTEs)
```

7 writelog

```
from time import *
import inspect

def init_log(ID):
    """Initialises the log with an ID for the file"""
    filename = "runlog_" + str(ID) + ".log"
    program_log = open(filename, 'w')
    program_log.write("Log File for {} in program {}\n{}\n"
                      .format(ID, inspect.stack()[1][1], "/" * 200))
    write_to_log(program_log, "Log Started")
    return program_log

def write_to_log(log, string):
    """takes a log object and writes the given string and timestamps it"""
    logtime = strftime("[%H:%M:%S %d/%m/%Y] ", gmtime())
    log.write(logtime + string + '\n')

def close_log(log):
    """closes the file"""
    write_to_log(log, "Log Ended")
    log.close()

'''#Test case
log = init_log(0)
write_to_log(log, "Error 1")
write_to_log(log, "Error 2")
write_to_log(log, "Error 3")
write_to_log(log, "Error 4")
close_log(log)'''
```

8 Configuration files for Example Network

8.1 router1

```
router-id, 1
input-ports, 1024, 1026, 1028
#format: portn-metric-router-id
outputs, 1025-1-2, 1027-8-7, 1029-5-6
#format: periodic, timout, garbage
timers, 3, 18, 12
```

8.2 router2

```
router-id, 2
input-ports, 1025, 1037
#format: portn-metric-router-id
outputs, 1024-1-1, 1036-3-3
#format: periodic, timout, garbage
timers, 3, 18, 12
```

8.3 router3

```
router-id, 3
input-ports, 1036, 1035
#format: portn-metric-router-id
outputs, 1037-3-2, 1034-4-4
#format: periodic, timout, garbage
timers, 3, 18, 12
```

8.4 router4

```
router-id, 4
input-ports, 1034, 64000, 1033
#format: portn-metric-router-id
outputs, 1035-4-3, 2000-6-7, 1032-2-5
#format: periodic, timout, garbage
timers, 3, 18, 12
```

8.5 router5

```
router-id, 5
input-ports, 1031, 1032
#format: portn-metric-router-id
outputs, 1033-2-4, 1030-1-6
#format: periodic, timout, garbage
timers, 3, 18, 12
```

8.6 router6

```
router-id, 6
input-ports, 1029, 1030
#format: portn-metric-router-id
outputs, 1031-1-5, 1028-5-1
#format: periodic, timout
timers, 3, 18, 12
```

8.7 router7

```
router-id, 7
input-ports, 1027, 2000
#format: portn-metric-router-id
outputs, 1026-8-1, 64000-6-4
#format: periodic, timout, garbage
timers, 3, 18, 12
```