COSC364 Assignment 1

George Drummond, Ryan Cox

April 13, 2017

Abstract

1 Main Program

```
#!/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


class RIProuter:
     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
          self.log = init_log(self.routerID)

          print('routerID =',self.routerID)
          print('inport numbers =',self.inPort_numbers)

```

```
32          print('peerInfo =',self.peerInfo)
33          print('timers =',self.timers)
34          print('table=\n',self.routingTable)
35
36
37      def socket_setup(self):
38          self.inPorts = []
39          for portn in self.inPort_numbers:
40              newSocket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
41
42              #newSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR
    , 1)
43
44              newSocket.bind((HOST_ID, portn))
45              self.inPorts += [newSocket]
46
47
48      def close_sockets(self):
49          ''' Close all sockets'''
50          for port in self.inPorts:
51              port.close()
52
53
54      def parse_config(self):
55          ''' Parse the supplied config file'''
56          lines = self.configFile.readlines()
57          for line in lines:
58              entries = line.split(',')
59              #print(entries)
60              lineType = entries[0]
61              tail = entries[1:]
62              if lineType == 'router-id':
63                  self.set_ID(tail)
64
65              elif lineType == 'input-ports':
66                  self.set_InPort_numbers(tail)
67
68              elif lineType == 'outputs':
69                  self.set_peerInfo(tail)
70
71              elif lineType == 'timers':
72                  self.set_timers(tail)
73
74
75      def set_ID(self,tail):
76          ''' Checks and stores routerID'''
77          myID = int(tail[0])
78          if myID in range(1,64001):
79              self.routerID = int(tail[0])
80
81          else:
82              raise(IndexError('Router ID not valid'))
```

```
83
84
85      def set_InPort_numbers(self,tail):
86           ''' Checks and stores all supplied inport numbers'''
87           self.inPort_numbers = []
88           for portstring in tail:
89                port = int(portstring)
90                if (port not in self.inPort_numbers) and (port in range
    (1024,64001)):
91                     self.inPort_numbers += [port]
92                else:
93                     print("invalid inport port {} supplied".format(port))
94
95
96      def set_peerInfo(self,tail):
97           ''' Stores info relevent to immediate neighbours '''
98           self.peerInfo = dict()
99           for triplet in tail:
100               portN,metric,peerID = triplet.split('-')
101               self.peerInfo[int(peerID)] = (int(portN),int(metric))
102
103
104     def set_timers(self,tail):
105          ''' Stores supplied timer info (i.e. periodic, timeout, garbage)
    '''
106          self.timers = []
107          for entry in tail:
108               self.timers += [int(entry)]
109
110
111
112     def send_updates(self):
113          ''' Sends an update message to each neighbour'''
114          i = 0
115          for peerID in self.peerInfo.keys():
116               print("update sent to {}".format(peerID))
117               write_to_log(self.log,
118                            "Sent update to {}".format(peerID))
119               OutSock = self.inPorts[i] # use a different socket to send
    each
120
121               peerPort = self.peerInfo[peerID][0]
122               response = self.response_packet(peerID)
123
124               OutSock.sendto(response.encode('UTF-8'),(HOST_ID,peerPort))
125
126               i += 1
127
128
129     def response_packet(self, peerID):
130          ''' Construct a response packet destined to a neighboring router.
131              Suitable for a periodic or triggered update'''
```

```
132
133         packet = ""
134         packet += rip_header(self.routerID)
135         for Entry in self.routingTable:
136             # Implement split horizon with poisson reverse
137             if (Entry.nextHop == peerID) or (Entry.garbageFlag == 1):
138                 print("split horizon entry sent to {}".format(peerID))
139                 packet += RTE(TableEntry(Entry.dest, INF, Entry.nextHop
    )) # set metric to INF
140             else:
141                 packet += RTE(Entry)
142
143         return packet
144
145     def proccess_rip_packet(self, packet):
146         ''' Processes a RIP response packet'''
147
148         n_RTEs = len(packet[8:])//(8*5)
149         """Check packet feilds are correct here"""
150         peerID = int(packet[4:8],16)
151
152         if peerID < 1 or peerID > 64000:
153             print("[Error] peerID {} out of range".format(peerID))
154             write_to_log(self.log, "[Error] peerID {} out of range".
    format(peerID))
155             #need to do something here
156
157         print("Proccessing packet from {}".format(peerID))
158         cost = self.peerInfo[peerID][1]
159
160         # Consider direct link to peer Router
161         incomingEntry = self.routingTable.get_entry(peerID)
162         if incomingEntry is None:
163             print("added directlink entry to router {}".format(peerID))
164             self.routingTable.add_entry(peerID, cost, peerID)
165         else:
166             incomingEntry.timeout = 0 # Reinitialise timeout for this
    link
167             incomingEntry.garbageFlag = 0
168             incomingEntry.garbage = 0
169
170
171         i = 8 # Start of first RTE
172         while i < len(packet):
173
174             dest = int(packet[i+8:i+16],16) # Read dest from RTE
175             metric = int(packet[i+32:i+40],16) # Read metric from RTE
176
177             new_metric = min(metric + cost, INF) # update metric
178
179             """check metric here?"""
180             currentEntry = self.routingTable.get_entry(dest)
```

```
181
182                 currentEntry = self.routingTable.get_entry(dest)
183
184
185                 if new_metric >= INF:
186                     print("Metric greater than {} and so is unreachable".
    format(INF))
187                     write_to_log(self.log,
188                                  "[Error] path ({},{}) from {} unreachable"
    .format(dest, metric,peerID))
189                     #do something here
190
191                 if (currentEntry is None):
192                     print("current entry is NONE!")
193                     if (new_metric < INF): # Add a new entry
194                         NewEntry = TableEntry(dest, new_metric, peerID)
195                         print('new Entry {}'.format(NewEntry))
196                         write_to_log(self.log,
197                                      "New route added from {} to {} with
    Metric {}"
198                                      .format(self.routerID, NewEntry,
    new_metric))
199                         self.routingTable.add_entry(dest, new_metric,
    peerID)
200
201
202                         self.routingTable.addEntry(dest, new_metric,
    peerID)
203
204                 else: # Compare to existing entry
205
206                     print("Existing entry for {}".format(dest))
207                     if (currentEntry.nextHop == peerID): # Same router as
    existing route
208
209                         currentEntry.timeout = 0 # Reinitialise timeout
210                         currentEntry.garbageFlag = 0
211                         currentEntry.garbage = 0
212
213                         if (new_metric != currentEntry.metric):
214                             self.existing_route_update(currentEntry,
    new_metric, peerID)
215
216
217
218                     elif (new_metric < currentEntry.metric):
219                         print("update route to {}".format(dest))
220                         write_to_log(self.log,
221                                      "Route from {} to {} updated with new
     Metric {}"
222                                      .format(self.routerID, NewEntry,
    new_metric))
```

```
223                         self.existing_route_update(currentEntry,
    new_metric, peerID)
224
225
226
227
228
229             i += (8*5) # Proceed to next RTE
230
231
232     def existing_route_update(self, currentEntry, new_metric, peerID):
233         currentEntry.metric = new_metric
234         print("route to {} updated to metric = {}".format(currentEntry.
    dest,new_metric))
235         currentEntry.nextHop = peerID
236
237         if (new_metric >= INF):
238             print("Triggered update flag set")
239             self.updateFlag = 1 #Set some update flag
240             currentEntry.garbageFlag = 1
241
242
243
244
245 class RoutingTable:
246     def __init__(self, timeoutMax, garbageMax):
247         self.table = []
248         self.timeoutMax = timeoutMax
249         self.garbageMax = garbageMax
250
251     def __iter__(self):
252         i = 0
253         while i < len(self.table):
254             yield(self.table[i])
255             i += 1
256
257     def __repr__(self):
258         blank = "-" * 54
259         print(blank + "\n| dest | metric | nextHop | flag | timeout |
    garbage |")
260         for Entry in self.table:
261             print("|{:>5} |{:>7} |{:>8} |{:>5} |{:>8.3f} |{:>8.3f} |".
    format(
262                 Entry.dest, Entry.metric, Entry.nextHop, Entry.
    garbageFlag,
263                 Entry.timeout, Entry.garbage))
264
265         return blank
266
267     def add_entry(self,dest, metric, nextHop):
268         self.table += [TableEntry(dest, metric, nextHop)]
269
```

```
270     def remove_entry(self, Entry):
271         print("Entry {} removed".format(Entry))
272         self.table.remove(Entry)
273 
274     def get_entry(self, dest):
275         ''' returns required table entry if already present'''
276         i = 0
277         while i < len(self.table):
278             Entry = self.table[i]
279             if Entry.dest == dest:
280                 return Entry
281 
282             i += 1
283 
284         return None
285 
286 
287 
288 class TableEntry:
289     def __init__(self,dest, metric, nextHop):
290         self.dest = dest
291         self.metric = metric
292         self.nextHop = nextHop
293         self.garbageFlag = 0
294         self.timeout = 0
295         self.garbage = 0
296 
297     def __repr__(self):
298         return str((self.dest, self.metric,
299                     self.nextHop, self.garbageFlag, self.timeout, self.
    garbage))
300 
301 
302 
303 def main():
304     configFile = open(sys.argv[1])
305     #configFile = open("router1.conf") # Just for developement
306     router = RIProuter(configFile)
307     selecttimeout = 0.5
308     periodicWaitTime = router.timers[0]
309 
310     starttime = time.time() #Gets the start time before processing
311 
312     while(1):
313         ## Wait for at least one of the sockets to be ready for
    processing
314         print("table reads\n",router.routingTable)
315 
316         readable, writable, exceptional = select.select(router.inPorts,
    [], router.inPorts, selecttimeout) #block for incoming packets for half
    a second
317 
```

```
318         # Send triggered updates at this stage
319         if (router.updateFlag == 1):
320              router.send_updates()
321              router.updateFlag = 0
322 
323         for sock in readable:
324 
325              packet = sock.recv(MAX_BUFF).decode('UTF-8')
326              router.proccess_rip_packet(packet)
327 
328         timeInc = (time.time() - starttime) #finds the time taken on
    processing
329         #print("proc time = {}".format(timeInc))
330         starttime = time.time()
331         router.periodic += timeInc
332 
333         if (router.periodic >= periodicWaitTime): # Periodic update
334              router.send_updates()
335             #Recalculate new random wait time in [0.8*periodic, 1.2*
    periodic]
336              periodicWaitTime = random.uniform(0.8*router.timers[0],1.2*
    router.timers[0])
337 
338              router.periodic = 0 # Reset periodic timer
339              print("Periodic update")
340 
341         for Entry in router.routingTable:
342 
343              if (Entry.garbageFlag == 1):
344                   Entry.garbage += timeInc
345                   if (Entry.garbage >= router.timers[2]): # Garbage
    collection
346                        print('Removed {}'.format(Entry))
347                        write_to_log(router.log,
348                                     "[Warning] Route from {} to {} has
    been removed"
349                                     .format(router.routerID, Entry.dest))
350                        router.routingTable.remove_entry(Entry)
351 
352              else:
353                   Entry.timeout += timeInc
354                   if (Entry.timeout >= router.timers[1]): # timeout/
    delete event
355                        print('Timeout')
356                        write_to_log(router.log,
357                                     "[Warning] Route from {} to {} has
    timed out"
358                                     .format(router.routerID, Entry.dest))
359                        Entry.metric = INF
360                        router.updateFlag = 1 # require triggered update
361                        Entry.garbageFlag = 1 # Set garbage flag
362 
```

```
363
364
365
366
367
368 # Small developement test case
369 t = RoutingTable(18,12)
370 t.add_entry(2, 4, 3)
371 t.add_entry(1, 5, 6)
372 t.add_entry(5, 3, 3)
373
374 for Entry in t:
375     pass
376
377 main()
```

2 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)
```

3 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)'''
```