**COSC364 RIP Documentation**

**Contribution**

Noel Soong- 56244036 50%

Alexis Sy- 21229382 50%

**Which aspects of your overall program (design or implementation) you consider particularly well done?**

The aspects we consider the overall design/implementation to be particularly well done are:

* Making sure that the files and the program can be easily read with the use of relevant naming of variables, constants, functions and also by using descriptors and comments
* Making sure that we kept each other updated with the use of github, frequently committing and checking the progresses made and also meeting up regularly
* Making sure that the bellman-ford algorithm is accurate
* Making sure that split-horizon with poison reverse works properly

**Which aspects of your overall program (design or implementation) could be improved?**

The aspects we consider the overall design/implementation that could be improved are:

* Making sure that we have the same idea/have a clear goal in mind about the outcome of each function before the implementation of the function.
* We think that we can improve convergence time by removing the timeout timer in the schedule and adding the hold\_down/garbage timer so that it doesn’t have to wait for its timeout to start the garbage collection.
* We also think that our propagation of failed routes was slow because when we learned of a failed route from one of our neighbours we did not flood the failed route via a triggered update to the **other** neighbours. The timeout timer will also be removed and the holddown timer added as we no longer have to wait for the event that the timeout time expire to set the holddown timer.

**How have you ensured atomicity of event processing?**

We have ensured the atomicity of event processing by making the program single threaded using timers so that there is no way race conditions can occur. We have done this by setting up timers for updates, timeouts and garbage collection so that when the timer set expires the program would do the events in the order they have expired, so only one event is processed at a certain time with the concept of a round robin scheduler, therefore the data in the routing tables stay consistent.

**Testing**

We tested the correctness of the program by using the configuration files below and checking that the metric and the next hop in the routing table is correct when all of the routers are up. We also tested that timeout and garbage procedures are carried out correctly by making router 6 go down and seeing how the other routers adjust to this information in their routing table. We also tested this condition again, but in this instance before the garbage timer for router 6 expires for the routers that are alive, we make router 6 alive again which will delete the garbage timer in the routers and add the timeout timer for router 6 respectively in the timer schedule of the routers.

We also tested the scenario in which routers 1, 7, 4, 6, 5 are up and after the convergence time we made router 6 go down. So that in this instance router 1 and router 5 are unreachable to each other and therefore will have a metric of infinity and should be garbage collected respectively in their own routing table, testing that the timeout and garbage timers and their respective functions are implemented and works correctly.

We tested the implementation of split-horizon with poison reverse by setting up the routers 1, 2, 3 and because of poison reverse router 1 advertises to router 2 that router 3 is unreachable as in its own routing table the next hop for router 3 is router 2, as this is the case when router 3 goes down a routing loop would not occur as both router 1 and 2 are sending each other that router 3 is unreachable.

We also added exceptions to catch if we mistyped the configuration file name and the configuration itself, making sure that arguments are correctly given and not missing. We purposely put in wrong information which the exception will catch and stops the program and gives relevant information to debug and which line it is on.

We also tried to isolate a router with its neighbours going down ie. router 1, 7, 4, 5, 6 being alive. Then router 4 and 6 goes down leaving router 5 isolated as it cannot reach the other routers anymore, despite being up. Through this we found that the wait time cannot be 1, instead it has to be a float value, so we changed it to a value close to 1 as for linux 1 second is not exactly 1 in UTC.

**Source code**

|  |
| --- |
| **router.py** |
| #Build Ins  from socket import \*  import select  import time  import pickle  import heapq  import sys  #Custom modules  import packet  import my\_timer  from color import bcolors  #Globals assumed  HOST = '127.0.0.1'  INF = 16  class Router():  def \_\_init\_\_(self, router\_id, input\_ports, outputs, update\_time, timeout\_time):  arg\_count = 0 # Counts number of Arguments for  try:  self.router\_id = int(router\_id)  #port numbers of output ports  self.input\_ports = input\_ports  self.output\_ports = []  self.metrics = []  self.neighbour\_routers = [] #List of Neibhour Router IDs  self.sockets\_in = [] # List of input sockets  self.routing\_table = dict() #Routing Table  #TIMERS  self.update\_time = int(update\_time)  self.timeout\_time = int(timeout\_time)  self.hold\_time = int(timeout\_time) + int(update\_time) / 2  for p, m, n in outputs:  self.output\_ports.append(int(p))  self.metrics.append(int(m))  self.neighbour\_routers.append(int(n))  #Set source metric to 0 and neigbour to None  self.routing\_table[self.router\_id] = [0, None]  self.print\_table()  except SyntaxError as msg:  if router\_id:  arg\_count += 1  if input\_ports:  arg\_count += 1  if outputs:  arg\_count += 1  if update\_time:  arg\_count += 1  if timeout\_time:  arg\_count += 1  sys.exit(bcolors.FAIL + tr(msg) + " : Requires 5 arguements has " + str(arg\_count) + bcolors.ENDC)  def print\_table(self):  table = self.routing\_table  c\_time = str(time.ctime())  print(bcolors.OKGREEN + "Router: " + str(self.router\_id) + bcolors.ENDC)  for route in table:  if route == self.router\_id:  continue  elif table[route][0] < INF:  print("{}: {} VIA {} COST OF {}".format(c\_time, route, table[route][1] , table[route][0]))  else:  print(c\_time + bcolors.FAIL + " "+str(route) + " UNREACHABLE" + bcolors.ENDC)  def bellmanford(self, packet):  """Calculates best round and changes the routing table"""  routing\_table = packet.data  router\_id = packet.router\_id  for key in routing\_table:  metric = packet.data.get(key) + self.routing\_table[router\_id][0]  next\_hop = router\_id  if key not in self.routing\_table and metric < INF:  self.routing\_table[key] = [metric, router\_id]  else:  if key in self.routing\_table:  #if the cost of next\_hop changes in routing table  if self.routing\_table[key][1] == next\_hop and metric > self.routing\_table[key][0]:  if metric >= INF:  self.routing\_table[key] = [INF, None]  else:  self.routing\_table[key] = [metric, next\_hop]  #if you find a lower cost  if metric < self.routing\_table[key][0] and metric < INF:  self.routing\_table[key] = [metric, next\_hop]  def set\_to\_inifinity(self, dead\_n):  """Set all routes from a daed neighbour to infinity in the table """  for k in self.routing\_table:  if self.routing\_table[k] == dead\_n:  self.routing\_table[k] == [INF, None]  def garbage(self, dead\_n):  """Deletes old message from table"""  del self.routing\_table[dead\_n]  def send\_update(self, t):  """Sends update to neigbour routers, Performs SPL HOZ with Poison Reverse  when it encounters a route that """  num\_input\_sockets = range(len(self.sockets\_in))  for i in num\_input\_sockets:  send\_routes = {}  for key in self.routing\_table:  """If router entry is not our neighbour but has received the route via  neibhour then send an infinite value for that route to neibhour"""  if key != self.neighbour\_routers[i]:  #Poison Reverse  if self.routing\_table[key][1] == self.neighbour\_routers[i]:  send\_routes[key] = INF  else:  send\_routes[key] = self.routing\_table[key][0]  #PACKET SENT TO NEIGHBOURS  p = pickle.dumps(packet.Packet(self.router\_id, int(time.time()), send\_routes))  self.sockets\_in[i].sendto(p, (HOST, self.output\_ports[i]))  #Update timer resets  t.add\_timer(self.update\_time + int(time.time()), self.router\_id, "update")  def handle\_read(self, s, t):  """Receives packets and reads data in form of routing table and router  id of sender"""  packet, address = s.recvfrom(1024) #Receives Serialized Packet  d = pickle.loads(packet) #Decodes Packet  for i, neighbour\_id in enumerate(self.neighbour\_routers):  if neighbour\_id == d.router\_id:  #If we are still receiving updates from neighbour reset the timeout  self.routing\_table[neighbour\_id] = [self.metrics[i], self.router\_id]  t.remove\_timer(neighbour\_id)  t.add\_timer(self.timeout\_time + int(time.time()), neighbour\_id, "timeout")  #If the metric of a route is still less than INF then reset timeout  for router in d.data:  if d.data.get(router) + self.routing\_table[d.router\_id][0] < INF:  t.remove\_timer(router)  t.add\_timer(self.timeout\_time + int(time.time()), router, "timeout")  self.bellmanford(d) #calculates best route  def handle\_timer(self, exp\_ID, exp\_message, t):  """Timer handler will check the message to determine the type of timer  then it will perform the actions that relate to update"""  if exp\_message == "update":  for (duration, ID, message) in t.timer\_shedule:  # IT SHOULD JUST DELETE THE FIRST OCCURENCE  if exp\_ID == ID and message == "update":  t.remove\_timer(ID)  break  self.send\_update(t)  elif exp\_message == "timeout":  # if timeout expires send tiggered update to neighbours that router unreachable  self.routing\_table[exp\_ID] = [INF, None]  #If route has expired then set metric to INF  for key in self.routing\_table:  if self.routing\_table[key][1] == exp\_ID:  self.routing\_table[key] = [INF, None]  #Timout Expires set Garbage collection Timer  t.add\_timer(self.hold\_time + int(time.time()), exp\_ID, "garbage")  self.send\_update(t) #Triggered Update  elif exp\_message == "garbage":  print(bcolors.FAIL + "Garbage Colection" + bcolors.ENDC)  self.garbage(exp\_ID)  def turn\_on(self):  """Turns on router all routing fucntions are called here.  @Return None"""  # Use input\_port list to create sockets  for \_\_ in range(len(self.input\_ports)):  self.sockets\_in.append(socket(AF\_INET, SOCK\_DGRAM))  # Then Bind Them  for i in range(len(self.sockets\_in)):  self.sockets\_in[i].bind((HOST, self.input\_ports[i]))  t = my\_timer.Timers()  #start a timer to send update first  t.add\_timer(self.update\_time + int(time.time()), self.router\_id, "update")  while True:  wait = t.get\_timeout() #gets the latest timeout  if wait == None or wait == 0:  #If Scheduler is empty set default wait to 1  wait = 0.99797997997  print(bcolors.WARNING + "Scheduler Empty Refeshing" + bcolors.ENDC)  #Waits for input, timeout used for round robin timing    readable, \_, \_ = select.select(self.sockets\_in, [], [], wait)  for input\_socket in readable:  self.handle\_read(input\_socket, t)  for ID, message in t.get\_expired\_timers():  self.handle\_timer(ID, message, t)  self.print\_table()  # print(t.timer\_shedule) |
| **scanner.py** |
| #Python Ver. 2.7.12  import re import sys import router  class Scanner():  def \_\_init\_\_(self, filename):  """Constructor for scanner  @var str filename,  int current\_token\_index = 0,  int line\_index = 0,  int router\_id,  list router\_input\_ports,  list router\_output\_ports,  int update\_time = 30,  int timeout\_time = 180  @Return None"""  self.filename = filename  self.current\_token\_index = 0  self.line\_index = 0   self.router\_id = None  self.router\_input\_ports = []  self.router\_output\_ports = []  self.update\_time = 30  self.timeout\_time = 180  def scan(self):  """Recieves file path in the form of a strings  then executes interpret\_line  @var f,  @funct isinstance, readlines, interpret\_line  @Return None"""  if isinstance(self.filename, basestring):  with open(self.filename, "rw") as f:  for line in f.readlines():  self.line\_index += 1  self.interpret\_line(line.splitlines())  else:  sys.exit("FILENAME MUST BE A STRING")  def get\_token(self):  """Previews the next token  @Return token\_index"""  token\_index = self.current\_token\_index + 1  return token\_index  def interpret\_line(self, full\_line):  """ Recieves a line at a time splits the lines by space and calls switch  to check the current token  @Return None"""  if full\_line != None:  self.current\_token\_index = 0 #resets once loops to new line  line = full\_line[0].rsplit()  for \_\_ in line:  if self.current\_token\_index >= len(line):  break  self.switch(line[self.current\_token\_index], line)  self.current\_token\_index += 1  def switch(self,token,line):  """Checks for all the statically typed cases in if statements.  @Return None"""  if re.match(r"router-id", token):  self.new\_router(line)  elif re.match('input-ports', token):  self.new\_input\_ports(line)  elif re.match('output-ports', token):  self.new\_output\_ports(line)  elif re.match("#"+".\*?", token):  self.current\_token\_index = len(line)  elif re.match('set-update', token):  self.set\_timer\_update(line)  elif re.match('set-timeout', token):  self.set\_timer\_timeout(line)  else:  sys.exit("LINE " + str(self.line\_index) +  " : Does not follow proper Syntax at... " + line[0])  def new\_router(self, line):  """Creates New subclass Router() also checks if there is an integer after  router-id otherwise exception thrown  @Return None"""  if re.match(r"\d", line[self.current\_token\_index + 1]):  self.router\_id = line[self.current\_token\_index + 1]  self.current\_token\_index += 1  else:  sys.exit("LINE " + str(self.line\_index) +  " : Integer must follow router-id")  def new\_input\_ports(self, line):  """Helper function for finding input-port numbers  @Return None"""  try:  while self.get\_token() < len(line):  token = line[self.get\_token()]  if int(token) in range(1,64001):  self.router\_input\_ports.append(int(token))  self.current\_token\_index += 1  else:  raise ValueError  except ValueError as msg:  sys.exit(  "Socket input must be followed by an integer range 1 to 64000" +  str(msg) + " at " + str(self.line\_index))    def new\_output\_ports(self, line):  """Helper function for finding peer output-port numbers,metric,router-id  @Return None"""  try:  while self.get\_token() < len(line):  token = line[self.get\_token()]  output\_router = token.split("-")   soc = int(output\_router[0])  metric = int(output\_router[1])  next\_hop = int(output\_router[2])   if soc in range(1, 64001) and metric < 16 and next\_hop < 1000:  self.router\_output\_ports.append([soc, metric, next\_hop])   self.current\_token\_index += 1  except ValueError as msg:  sys.exit(str(msg) + " Output Ports must be a value 1 to 64000")  def set\_timer\_update(self, line):  """Sets Update Time in Seconds"""  try:  token = line[self.get\_token()]  if re.match(r"[0-9]", token):  if int(token) < self.timeout\_time:  self.update\_time = int(token)  else:  raise ValueError#raise value error if token does not fullfil requirements  self.current\_token\_index += 1  except ValueError as msg:  sys.exit(str(msg) + " Time must be less than timeout. Set update time first.")  def set\_timer\_timeout(self, line):  """Sets Timeout Time in Seconds"""  try:  token = line[self.get\_token()]  if re.match(r"[0-9]", token):  if int(token) > self.update\_time:  self.timeout\_time = int(token)  else:  raise ValueError#raise value error if token does not fullfil requirements  self.current\_token\_index += 1  except ValueError as msg:  sys.exit(str(msg) + " Time must be greater than update time. Set update time first.")  def main():  """Main executions here"""  read = None  file\_input = None  while True:  file\_input = raw\_input("Input File Name (q to quit): ")  if file\_input == "q":  sys.exit("Quiting Program")  elif re.match(r".", file\_input):  read = Scanner(file\_input)  read.scan()  break   r = router.Router(read.router\_id, read.router\_input\_ports, \  read.router\_output\_ports, read.update\_time, read.timeout\_time)  r.turn\_on()   if \_\_name\_\_ == "\_\_main\_\_":  main() |
| **my\_timer.py** |
| import time from heapq import \*  class Timers(object):  """Timer Class. An Instance shuold be created in router"""  def \_\_init\_\_(self):  self.timer\_shedule = []  def add\_timer(self, duration, ID, message):  """ Add a timer of the given duration and (unique) ID.  It should also have some information stored with it.  """  heappush(self.timer\_shedule, (duration, ID, message))    def remove\_timer(self, removeID):  """The timer of the given ID will be removed"""  for i, (duration, ID, message) in enumerate(self.timer\_shedule):  if ID == removeID:  self.timer\_shedule.pop(i)  heapify(self.timer\_shedule)    def get\_timeout(self):  """ Caclulate the ammout of time left until the next timer will go off  @return Timer time  """  if self.timer\_shedule:  return self.timer\_shedule[0][0] - int(time.time())    def get\_expired\_timers(self):  """ Retrieves expired timers  @return List expired  """  expired = []  for (duration, ID, message) in self.timer\_shedule:  if duration <= int(time.time()):  expired.append((ID, message))  self.remove\_timer(ID)  return expired |
| **packet.py** |
| class Packet():   def \_\_init\_\_(self, router\_id, time, routing\_table=None):  self.version = 2  self.type = 2  self.router\_id = int(router\_id)  self.data = routing\_table  self.time = time |
| **color.py** |
| class bcolors:  HEADER = '\033[95m'  OKBLUE = '\033[94m'  OKGREEN = '\033[92m'  WARNING = '\033[93m'  FAIL = '\033[91m'  ENDC = '\033[0m'  BOLD = '\033[1m'  UNDERLINE = '\033[4m' |

**Configuration Files Needed for Figure 1**

**Router 1 Configuration File**

router-id 1

#input ports

input-ports 6000 6001 6002

#output ports PORTNUM,METRIC,NEXTHOP

output-ports 5000-5-6 5001-1-2 5002-8-7

set-update 6

set-timeout 36

**Router 2 Configuration File**

router-id 2

#input ports

input-ports 5001 6003

#output ports PORTNUM,METRIC,NEXTHOP

output-ports 5003-3-3 6001-1-1

set-update 6

set-timeout 36

**Router 3 Configuration File**

router-id 3

#input ports

input-ports 5003 6004

#output ports PORTNUM,METRIC,NEXTHOP

output-ports 5004-4-4 6003-3-2

set-update 6

set-timeout 36

**Router 4 Configuration File**

router-id 4

#input ports

input-ports 5004 6005 6007

#output ports PORTNUM,METRIC,NEXTHOP

output-ports 5005-2-5 5007-6-7 6004-4-3

set-update 6

set-timeout 36

**Router 5 Configuration File**

router-id 5

#input ports

input-ports 5005 6006

#output ports PORTNUM,METRIC,NEXTHOP

output-ports 5006-1-6 6005-2-4

set-update 6

set-timeout 36

**Router 6 Configuration File**

router-id 6

#input ports

input-ports 5000 5006

#output ports PORTNUM,METRIC,NEXTHOP

output-ports 6000-5-1 6006-1-5

set-update 6

set-timeout 36

**Router 7 Configuration File**

router-id 7

#input ports

input-ports 5002 5007

#output ports PORTNUM,METRIC,NEXTHOP

output-ports 6002-8-1 6007-6-4

set-update 6

set-timeout 36