COSC364: Internet Technologies and Engineering
First Assignment: RIP

Contributors: James Lilley 49778158, Jake Simpson 42402679

Agreed upon percentage:

James = 50%, Jake = 50%

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

First of all, the file setup where we ask for a filename, do the mandatory checks and extract all the necessary information/data, we feel, is done very well. We were able to create single functions for each necessary check, such as a router id check and port number check. Also, the extraction process is simple to understand, yet, efficient. We also feel that the technique used for the set up of routers sending packets/messages works well. We have a make_message function which puts together the message that the router wants to send. This function is then used in the send function where the packet gets sent to the destination it is destined for.

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

The approach we took to taking data from a socket as a string, then creating a new packet and filling it with data to analyze could do with improvement. We had many troubles with the extraction of data along the way. Thus, the way we break down the data as a string could be improved in some way.

How have you ensured atomicity of event processing?

Atomicity has been ensured by starting a process for every incoming port number. Each process is passed a multiprocess queue (Multiprocess.Queue()) that acts as a pipe from the individual processes to the main programme. The main programme checks if a link or an array of links has been sent to be processed, which guarantees that there is not read / write errors, as the processes never access the forwarding table. The processes sole responsibily is to monitor the incoming ports, before appending the data to the queue.

Discussion on testing that has been performed:

Initially, there were many small tests used when writing and building up the code. That is, testing used for debugging and finding faults in the code. For example, adding print statements in parts of code to see where, exactly, the program is breaking down. So, if the print statement was outputted, we knew the code was working up to that point and was breaking down after, or, on the other hand, it would not print and thus we could see the code was failing either at this point or beforehand. Also, to debug, we would comment out some lines and see if the program would still run without that line of code. If it did, then we knew the problem was within that block of code. These small testing procedures were carried out until we felt we had finally developed a fully working code that we deemed ready for final testing.

As stated previously, once we had code that we felt was sufficient for the testing, we began various testing procedures to check that our routing demon program was working and performing correctly. The first test was to start up all the routers and then check they were all communicating with each other. What was expected of this test was for all routers to show that they were sending and receiving packets from one another (this would also allow us to confirm that the configuration files were correct). With this test, we found that our routers were all receiving the right amount of messages from one another and were sending out the correct amount of messages to other routers.

The second test was to start up two routers and check they were communicating with each other. We then would stop one router from running and see how the other router handled it and it if it updated its forwarding table. By running this test we found we had a few issues such as the time to live attribute either kept incrementing or didn't, the router would send the wrong metric information and routers wouldn't add their own information to their forwarding table. We then fixed these issues and found that the routers communicated correctly with each other and when one router was dropped out, the other would update its forwarding table correctly.

Next, we started up all the routers and began to check whether the information in the forwarding tables were correct. That is, checking that all the forwarding tables and expecting to see all the resulting routes as minimum hop. In the first instance, the forwarding tables were not outputting the correct data. This was due to a coding error. Thus, we went back to the code and fixed the error. On the second instance of testing, our forwarding tables were showing the correct data and minimum hop routes. Therefore, we knew our code was working correctly for this particular test.

The next test was to switch off one router. With this test, we were expecting the remaining routers to communicate with each other and establish that a link had been lost. We then expected the remaining routers to converge and create a new minimum hop state. After carrying out this test, we found that the program failed when we got rid of a router. In the forwarding table, the router information would not disappear when it needed too. This showed us that there was a fault in our program. Therefore, we went back to the code, figured out what was wrong/wasn't there and fixed it. We then carried out the test again and found that the program kept running when a router was dropped and the forwarding tables were updated correctly with minimum hop states. We then decided to drop more than one router and expected it to run the same. We found that the program still ran correctly during this test.

Finally, we decided to do the two previous tests and then switch the routers that were turned off, back on again. What we expected to see was the routers communicating with each other and converging their information and see forwarding tables return back into the original state with the same minimum hop routes.

Once we had carried out all these tests and found that our program was working correctly, we deemed it sufficient for submission and demonstration.

Source Code:

packets.py

```
"""
a class for the packet header, complete with a field for 'payload'
which is the router information to be used in the router table
"""
class Packet:
    # command = command (1) (1 = request, 2 = response)
    # version = version (1) (version of the rip protocol used - always 2)
    # generating_router_id = Router that generates this RIP packet
    # payload = RIP entries (20) (between 1 - 25 inc, otherwise multiple packets)


    def __init__(self, command=None, version=None, generating_routerID=None,
p_payload=None):

        # Command
        if len(str(command)) == 1:
            self.command = command

        # Version
        if len(str(version)) == 1:
            self.version = version

        # Generating Router ID
        if len(str(generating_routerID)) == 2:
            self.generating_routerID = str(generating_routerID)
        elif len(str(generating_routerID)) == 1:
            self.generating_routerID = "0" + str(generating_routerID)

        # Payload
        self.p_payload = p_payload



    def __str__(self):
        string = str(self.command) + str(self.version) + str(self.generating_routerID)
        string += str(self.p_payload)
        return string



"""
a class for the packet payload, which gets included in the
class packet as payload. Between 1 and 25 payload objects to one packet object
```

```
"""
class Payload:
    # addr_fam_id = address family identifier (2)
    # ipv4_addr = IPv4 address (4)
    # routerID = the router that the Generating Router is describing
    # metric = metric (4) (cost - must be between 1 - 15 inc, 16 = inf / unreachable)

    def __init__(self, addr_fam_id=None, ipv4_addr=None, routerID=None, metric=None):

        # Address Family Identifer
        if len(str(addr_fam_id)) == 2:
            self.addr_fam_id = str(addr_fam_id)
        elif len(str(addr_fam_id)) == 1:
            self.addr_fam_id = "0" + str(addr_fam_id)

        # 00
        self.must_be_0_2 = '00'

        # IPv4 Address
        if len(str(ipv4_addr)) == 4:
            self.ipv4_addr = str(ipv4_addr)
        elif len(str(ipv4_addr)) == 3:
            self.ipv4_addr = "0" + str(ipv4_addr)
        elif len(str(ipv4_addr)) == 2:
            self.ipv4_addr = "00" + str(ipv4_addr)
        elif len(str(ipv4_addr)) == 1:
            self.ipv4_addr = "000" + str(ipv4_addr)

        # Router ID
        if len(str(routerID)) == 4:
            self.routerID = str(routerID)
        elif len(str(routerID)) == 3:
            self.routerID = "0" + str(routerID)
        elif len(str(routerID)) == 2:
            self.routerID = "00" + str(routerID)
        elif len(str(routerID)) == 1:
            self.routerID = "000" + str(routerID)

        # 0000
        self.must_be_0_4 = '0000'

        # Metric
        if len(str(metric)) == 4:
            self.metric = str(metric)
        elif len(str(metric)) == 3:
            self.metric = "0" + str(metric)
        elif len(str(metric)) == 2:
```

```
            self.metric = "00" + str(metric)
        elif len(str(metric)) == 1:
            self.metric = "000" + str(metric)


    def __str__(self):
        return str(self.addr_fam_id) + str(self.must_be_0_2) + str(self.ipv4_addr) +
str(self.routerID) + str(self.must_be_0_4) + str(self.metric)
```

RIP.py

```
import os
import sys
import re
import socket
import time


from multiprocessing import *
from packets import *

ROUTER_ID = None
INPUT_PORTS = []
OUTPUTS = []
USED_ROUTER_IDS = []
INCOMING_SOCKETS = []

TIME_TO_SLEEP = 3
LOW_ROUTER_ID_NUMBER = 0
HIGH_ROUTER_ID_NUMBER = 64000
LOW_PORT_NUMBER = 1024
HIGH_PORT_NUMBER = 64000

IP = "127.0.0.1"

# a tuple with the layout (Router ID, Metric Value, Router Learnt From, Time Loop)
FORWARDING_TABLE = []


########################## FILE SETUP ##########################

""" ask for filename """
def getInputFile():
    print("Enter a valid configuration file: ")
    configFile = input()
    return configFile



""" check the port number provided is within the allowed parameters """
def routerIdCheck(routerID):
    if int(routerID) > LOW_ROUTER_ID_NUMBER and int(routerID) <
HIGH_ROUTER_ID_NUMBER:
        return True
```

```
""" check that the port numbers provided from the config file are between
   the allowed parameters"""
def portNumberCheck(portno):
   if int(portno) > LOW_PORT_NUMBER and int(portno) < HIGH_PORT_NUMBER:
      return True



""" extract the router ID from the line parsed in the config file.
   If two router IDs are given in the config file, the second one will be
   deemed a correction / will override the first or previous Router IDs """
def extractRouterID(line):
   global ROUTER_ID
   routerID = re.findall(r'\b\d+\b', line)
   if routerIdCheck(int(routerID[-1])) == True:
      ROUTER_ID = (int(routerID[-1]))
      USED_ROUTER_IDS.append(ROUTER_ID)



""" extract the inport ports to set up UDP sockets. Returns a list of all valid
   port numbers between 1024 and 64000 """
def extractValidInputPorts(line):
   inputPorts = re.findall('[0-9]+', line)
   for ports in inputPorts:
      if portNumberCheck(ports) == True:
         INPUT_PORTS.append(int(ports))



""" extract the output ports to set up UDP sockets. returns a list of all valid
   port numbers between 1024 and 64000 """
def extractValidOutputPorts(line):
   global FORWARDING_TABLE
   global OUTPUTS
   global USED_ROUTER_IDS
   age = 0
   splitline = line.split(" ")
   for lines in splitline:
      outputPorts = re.findall('[0-9]+', lines)

      if outputPorts != []:
         if len(outputPorts) == 3:
            portnum = outputPorts[0]
            metric = outputPorts[1]
            router_id = outputPorts[2]
```

```python
            OUTPUTS.append([int(portnum), int(metric), int(router_id)])
            USED_ROUTER_IDS.append(int(router_id))



""" read and extract the router-id, input-ports and outputs """
def readInputFile(configFile):
    try:
        infile = open(configFile)
        lines = infile.readlines()

        for line in lines:
            if "router-id" in line:
                extractRouterID(line)

            if "input-ports" in line:
                extractValidInputPorts(line)

            if "outputs" in line:
                extractValidOutputPorts(line)

        infile.close()

    except FileNotFoundError:
        print("Sorry, the entered file was not found.")
        configFile = getInputFile()
        setupData = readInputFile(configFile)


########################### UDP SOCKETS ###########################

""" Set up a UDP port for all input ports. Acting as server side """
def incomingSocketSetUp():
    for inputSock in INPUT_PORTS:
        sockID = "IncomingSocket" + str(inputSock)
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.setblocking(True)
        try:
            sock.bind((IP, inputSock))
            INCOMING_SOCKETS.append((sockID,sock))
        except OSError:
            print("Socket already bound \n\n")
            sys.exit(1)



""" close all open sockets """
```

```
def closeSockets():
   try:
      for sockID, sock in INCOMING_SOCKETS:
         sock.shutdown(socket.SHUT_RDWR)
         sock.close()
      print(INCOMING_SOCKETS)

   except OSError:
      print("Error closing Sockets")
      sys.exit(1)

   finally:
      sys.exit(0)


######################### PROCESS FUNCTIONS #########################

""" a function that takes the data from a socket as a string, then creates
   a new packet, and fills it with data to analyse """
def openData(packet, queue):
   global USED_ROUTER_IDS
   global FORWARDING_TABLE

   i = 4
   pay = ''
   # turn Packet object into string and strip off byte indicators
   data_unstripped = str(packet)
   data = data_unstripped[2:-1]

   packet_info = []

   # get packet header information
   command = data[:1]
   version = data[1:2]
   generating_routerID = data[2:4]
   while i < len(data):
      # get packet payload information
      p_addr_fam_id = data[i:i+2]
      p_must_be_0_2 = data[i+2:i+4]
      p_ipv4_addr = data[i+4:i+8]
      p_routerID = data[i+8:i+12]
      p_must_be_0_4 = data[i+12:i+16]
      p_metric = data[i+16:i+20]
      i += 20

      # create a new object to insert into our routing table, or update if already inserted
      graph_data = [int(p_routerID), int(p_metric), int(generating_routerID), 0]
```

```
        packet_info.append(graph_data)

    # put object into queue to be sorted by main programme
    queue.put(packet_info)



""" a function called for the receive thread instead of run(). an infinite
loop that checks incoming sockets, and forwards accordingly or drops """
def receive(socket, queue):

    while True:
        data, addr = socket[1].recvfrom(1024) # buffer size is 1024 bytes
        openData(data, queue)


########################## FORWARDING  FUNCTIONS ##########################

""" a function that makes a packet to send to the output links """
def make_message(output):
    command = 2
    version = 2
    payld = ''

    # make a specialised packet for each output
    for router in FORWARDING_TABLE:
        #if the router we learnt a link from is the router we are making a packet for
        if int(output[2]) == int(router[2]):
            # set metric to 16
            route_payload = Payload(2, 2, str(router[0]), 16)

        else:
            k = 0
            while  k < len(OUTPUTS):
                # if the router we are sending the data to is a router we learnt the link for
                if router[0] == OUTPUTS[k][2]:
                    # set metric to the link cost
                    router[1] = OUTPUTS[k][1]
                    break
                k+=1

            # turn the variables into an Payload object
            route_payload = Payload(2, 2, str(router[0]), str(router[1]))
        # turn the payload to a string and add to a string
        payld += str(route_payload)
    #create a Packet object to send
    pac = Packet(command, version, ROUTER_ID, payld)
```

```
    return pac



""" a function called by the receive function, to forward a packet to the
next destination """
def send():
    # create UDP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    # for each output - make a packet and send using a UDP port
    for output in OUTPUTS:
        message = make_message(output)
        sock.sendto(str(message).encode('utf-8'), ("127.0.0.1", int(output[0])))



""" prints the forwarding table """
def print_table(table):
    print("Forwarding table:")
    print("ID, Metric, Learnt From, TTL(x30sec)")
    for line in FORWARDING_TABLE:
        print(line)
    print()



""" a function to take an individual link and process it, by adding it to the
    FORWARDING_TABLE if not present, otherwise reset counter to 0 """
def process_link(link):
    global FORWARDING_TABLE
    j = 0
    while j < len(FORWARDING_TABLE):
        #if the recieved link routerID equals an entry in the forwarding table
        if link[0] == FORWARDING_TABLE[j][0]:
            # reset the time to live to 0
            FORWARDING_TABLE[j][3] = 0

            # if the recieved link has a lesser cost to the router
            '''
            if (rcvd_link[1] < FORWARDING_TABLE[j][1]):
                # if the received routerID is in the list of directly connected routers
                if rcvd_link[0] in [item[3] for item in OUTPUTS]:
                    # update the link with the new cost
                    FORWARDING_TABLE[j][1] = rcvd_link[1]
                    FORWARDING_TABLE[j][2] = rcvd_link[2]
            '''
            break
```

```
        j+=1
    # if the link has not yet been discovered
    if j >= len(FORWARDING_TABLE):
        got_from = link[0]
        link_cost = 0

        FORWARDING_TABLE.append(link)



""" a function that monitors the queues that act as pipes, from the individual
   processes. to the main programme. When a link or an array of links has been
   discovered, we split them up to individual links and pass to the process link
   function """
def update(queue):
    global FORWARDING_TABLE
    global USED_ROUTER_IDS
    while True:
        try:
            # recieve a list of links from a process
            while True:
                rcvd_link = queue.get(False)
                # if only one link has been posted
                for link in rcvd_link:
                    # see if we recieved multiple links in one packet
                    try:
                        for li in link:
                            # process multiple links
                            process_link(li)
                    except:
                        # process the one link
                        process_link(link)
        # no data waiting for us from the queue - move on
        except:
            #print("no new data")
            i = 1

        # go through all lists in the forwarding table
        for route in FORWARDING_TABLE:
            # dont increment TTL feild in FORWARDING TABLE if we are looking at ourself
            if route[0] == ROUTER_ID:
                continue

            # increment time to live counter
            route[3] = route[3] + 1
            # if the link is 6 rotations old without an update, set cost to 16
            if route[3] >= 6:
```

```
            index = FORWARDING_TABLE.index(route)
            route[1] = 16
            FORWARDING_TABLE[index][1] = 16
        # or if the link has expired, delete link
        if route[3] >= 10:
            FORWARDING_TABLE = delete_link(route, FORWARDING_TABLE)
    # print table
    print_table(FORWARDING_TABLE)
    # create new packets to send
    send()
    # wait
    time.sleep(TIME_TO_SLEEP)



""" delete a dead link out of the FORWARDING TABLE """
def delete_link(route, table):
    USED_ROUTER_IDS.remove(route[2])
    table.remove(route)
    return table


########################### MAIN ###########################

def main():
    configFile = getInputFile()
    setupData = readInputFile(configFile)
    incomingSocketSetUp()

    print("\nRouter ID =", ROUTER_ID)

    # add ourself to the forwarding table
    FORWARDING_TABLE.append([ROUTER_ID, 0, ROUTER_ID,0])
    # start a queue for the processes to pass links back to the main programme
    queue = Queue()

    for socket in INCOMING_SOCKETS:
        process = Process(target=receive, args=(socket, queue, ))
        process.start()
    update(queue)

    closeSockets()


if __name__ == "__main__":
    main()
```

Configuration Files:

Config1.txt
router-id 1
input-ports 2001, 6001, 7001
outputs 1102-1-2, 1106-5-6, 1107-8-7

Config2.txt
router-id 2
input-ports 1102, 3002
outputs 2001-1-1, 2003-3-3

Config3.txt
router-id 3
input-ports 2003, 4003
outputs 3002-3-2, 3004-4-4

Config4.txt
router-id 4
input-ports 3004, 5004, 7004
outputs 4003-4-3, 4005-2-5, 4007-6-7

Config5.txt
router-id 5
input-ports 4005, 6005
outputs 5004-2-4, 5006-1-6

Config6.txt
router-id 6
input-ports 5006, 1106
outputs 6005-1-5, 6001-5-1

Config7.txt
router-id 7
input-ports 1107, 4007
outputs 7001-8-1, 7004-6-4