COSC364 Assignment RIP routing

Liam Bullock 34909160

Danish Jahangir 28134926

Plagiarism Declaration

This form needs to accompany your COSC 364 assignment submission.

I understand that plagiarism means taking someone else's work (text, program code, ideas, concepts) and presenting them as my own, without proper attribution. Taking someone else's work can include verbatim copying of text, figures/images, or program code, or it can refer to the extensive use of someone else's original ideas, algorithms or concepts.

I hereby declare that:

- My assignment is my own original work. I have not adapted, reproduced or modified code, figures/images, or writings of others without proper attribution. I have not used original ideas and concepts of others and presented them as my own.
- I have not allowed others to copy or modify my own code, figures/images, or writings. I have not allowed others to use original ideas and concepts of mine and present them as their own.
- I accept that plagiarism can lead to consequences, which can include total loss of marks, no grade being awarded and other serious consequences, including notification of the University Proctor.

Name: Liam Bullock

Student ID: 34909160

Signature:

Date: 30/04/2022

Plagiarism Declaration

This form needs to accompany your COSC 364 assignment submission.

I understand that plagiarism means taking someone else's work (text, program code, ideas, concepts) and presenting them as my own, without proper attribution. Taking someone else's work can include verbatim copying of text, figures/images, or program code, or it can refer to the extensive use of someone else's original ideas, algorithms or concepts.

I hereby declare that:

- My assignment is my own original work. I have not adapted, reproduced or modified code, figures/images, or writings of others without proper attribution. I have not used original ideas and concepts of others and presented them as my own.
- I have not allowed others to copy or modify my own code, figures/images, or writings. I have not allowed others to use original ideas and concepts of mine and present them as their own.
- I accept that plagiarism can lead to consequences, which can include total loss of marks, no grade being awarded and other serious consequences, including notification of the University Proctor.

Name: Danish Khursheed Jahangir

Student ID: 28134926

Signature:

Date: 30/04/2022

Percentage of each partner's contribution to project:

Danish Jahangir – 50%

Liam Bullock – 50%

Brief list of contributions by each partner:

The whole assignment was done by both of us together in all aspects. Every time we worked on the assignment, we worked on it with each other where one of us was doing the coding and the other was giving advice and cross-checking the others code and vice versa.

Which aspects of our overall program do we consider particularly well done?

We feel like we have done especially well in our file parsing and implementing the usage of python classes to have a better structure and format. This included checking validity across ports, router-ids and packets.

We also feel that we have implemented useful and descriptive comments across all functions which help make clear exactly what a specific part of the program is doing.

Finally, we feel that our implementation of receiving the packets stood out as being well done. Using select to simultaneously read all input ports, then receiving the packet, check it is valid and only add it to the routing table once it passed the validity checks.

Which aspects of our overall program do we think could be improved?

The biggest thing we could've improved on was how we went about the assignment. The approach we took was we got straight into it, following the RIP specification, and found times where we were stuck and unsure how to continue. This could have been avoided by writing up a proper plan to help design our routing demon.

We also felt that in some functions they got messy with nested for loops and a lot of 'if' and 'else' statements, which can make it hard to follow. To improve next time would be to lay these out better. However we have made it followable with comments throughout the code.

How have we ensured atomicity of event processing?

We ensured the atomicity of the event processing using timers. Each router has its own timer, which increases with every update of the routing table, when a certain amount of time passes (5 seconds +/- 5 seconds), then the packet is sent to the router and the timer is reset. Once it goes through all the routers and sends the packets, then the packet is received by the other routers using select and a message is sent. If a packet is not sent by the time the timer reaches 30 seconds (timeout), then it increases the garbage timer. If the garbage timer value reaches 20 seconds (garbage timeout), then the router is deleted from the routing table.

Have we identified any weaknesses of the RIP routing protocol?

RIP has limitations which we found during the creation of the routing demon. Firstly, it is limited to 15 hops. This makes it extremely difficult to use RIP in large network topologies. When testing naturally we changed the timers to be lower than the specification stated because the convergence is exceptionally slow. For larger networks, for RIP to converge, each router can take a few updates for it to realize the need for it to change its table, this can take many minutes for it to happen, which in networking terms is equal to an eternity.

During the implementation of the routing algorithm, we realized that whenever an unusual situation occurred, we had to depend on counting to infinity to resolve it. This led us to the conclusion that RIP is quite dependent on counting to infinity to solve its issue which can potentially lead to further problems in huge networks.

Testing

The first few tests are based on the actual assignment network topology, followed by edge cases.

Test scenario 1:

The first test that was done was testing if all 7 of the routers converge when they are active. When all 7 routers are up and running it is expected that they converge. This means they will all have information for every router within the network topology, have the metric (cost) to get there, the next hop to get to the destination, correct timer, and garbage timer. Testing for this did not go well the first time around. We had issues with the convergence not working as when we made our distance vector routing algorithm it was too basic and could not handle the network topology changes. This led to more conditions in update_routing_table to make convergence occur correctly. Figure 1 shows the output, when we added the necessary conditions to make it work.

```
==========================================================================
                    Routing table for router 1
Destination      Metric      Next Hop      Timer      Garbage Timer
     2             1             2            17             0
     3             4             2            17             0
     4             8             6            0              0
     5             6             6            0              0
     6             5             6            0              0
     7             8             7            2              0
==========================================================================
```

Figure 1

Test scenario 2:

The second test completed was testing how the network would handle a router being dropped/going down. What was expected to happen was that if for example router 5 goes down is that all routers would converge from neighboring routers of 5 updating the others that the router was no longer available. This would ripple effect to their neighboring routers until router 5 was completely removed from every routing table. The way this is done is the router will reach timeout for neighbor routers, which in term will then go to garbage timer, once this hits timeout it is deleted, and neighbor routers are updated. The test was passed as this occurred.

Test scenario 3:

The next test continues from the last one. This test involves activating a router once the network has converged, to see if it copes with the new addition within the network. The expected output of activating router 5 from the network topology given is that it will converge and any paths which have the same cost will not be updated. In this case to get to destination 4 will be different as it is same cost to go either to 2 or 6 at metric of 8. The demon implemented passes the tests.

Test scenario 4:

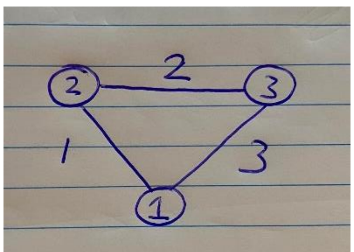

Figure 2

The test from figure 2 is trivial, just to see if convergence works as, it should. The expected result when dropping a router is that the other two routers would converge correctly and only have destination about each other. The actual result was this, proving that convergence has been implemented correctly.

Test scenario 5:

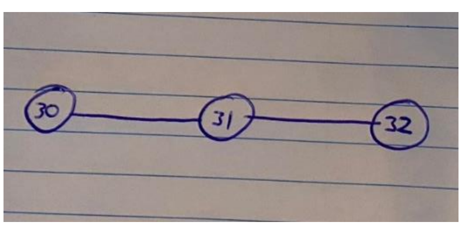

Figure 3

The test shown in figure 3 focuses on split horizon reverse. To test this, we ran all 3 routers, then dropped router 32. What should occur is that routers 30 and 31 converge to remove 32 from its routing table. The actual result was as expected, shown below in figure 4 and figure 5. The convergence used by split horizon meant that information was never sent back in the direction it was received and converged correctly. Upon testing these we found there was an issue with getting the correct router id from the configuration files. It would only get the first digit of the router id, not all digits. To fix this problem we changed it so that to get the router ID, the value is used from the configuration file.

```
                    Routing table for router 30
Destination      Metric      Next Hop      Timer      Garbage Timer
    31             4            31           18            0
================================================================================
```

Figure 4

```
                    Routing table for router 31
Destination      Metric      Next Hop      Timer      Garbage Timer
    30             4            30           3             0
================================================================================
```

Figure 5

Example configuration file

```
router-id, 1
input-ports, 4401, 2201, 3301
output-ports, 5502-1-2, 5506-5-6, 2207-8-7
```

For Each configuration file we followed the format of having router-id, input-ports, and output-ports. Each component is separated by a comma.

```python
"""
 Author: Liam Bullock - 34909160
 Author: Danish Jahangir - 28134926

 Implementing the RIP routing protocol
"""

import random
import socket
import json
import select
import sys
from copy import deepcopy

inputPorts = []
outputPorts = []
INFINITY = 16
TIMEOUT = 30
GARBAGE_TIMER = 1
GARBAGE_TIMEOUT = 20


class Socket:
    """ Class that handles the functions of the sockets"""

    def __init__(self, input_ports, socket_table):
        self.socket_table = socket_table
        self.input_ports = input_ports

    def open_socket(self):
        """Opens a socket for every input port"""
        self.socket_table = []
        for inputSocket in self.input_ports:
            sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            try:
                sock.bind(("127.0.0.1", int(inputSocket)))
                self.socket_table.append(sock)
            except OSError as e:
                print(e)
                print("Socket bind unsuccessful\n")
                exit()
        return self.socket_table

    def close_socket(self):
        """Closes a socket for every port"""
        try:
            for entry in self.socket_table:
                entry.close()
        except OSError:
            print("Socket close failed\n")
            exit()


class Packet:
    """Class that handles the functions of the packets"""

    def __init__(self, table):
        self.table = table

    def create_packet(self):
        """The header and entry of the RIP packet"""
        command = 2
        version = 2
        zeroField = router_id
        entry = []
        header = [command, version, zeroField]
```

```python
        for i in self.table.keys():  # Iterates through the routers and adds the costs and destination values to each in a tuple
            cost = self.table[i][0]
            entry.append((i, cost))
        packet = {"Header": header, "Entry": entry}  # Creates a dictionary called packet with header and entry as keys
        return packet

    def send_packet(self, packet):
        """Sending the UDP datagrams to neighbours"""
        for outputPort in outputPorts:  # iterates through the output ports, opens a socket for each and sends a new packet
            table_copy = deepcopy(self.table)
            routing = Routing(table_copy, packet)
            poison_table = routing.split_horizon(
                outputPort)  # creates a table using the split horizon function from the router class
            self.table = poison_table
            sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            sock.sendto(json.dumps(self.create_packet()).encode('ascii'), ("127.0.0.1", int(outputPort)))
        print("Packet sent successfully!")


class Routing:
    """Class that handles the functions for routing"""

    def __init__(self, table, packet):
        self.table = table
        self.packet = packet

    def split_horizon(self, neighbour_id):
        """Implement split horizon poison reverse to prevent the occurrence of routing loops"""
        for destination, info in self.table.items():
            if destination == neighbour_id:
                info[0] = INFINITY
        return self.table

    def update_entry_cost(self, current_router):
        """Updates the cost of the entry of the packet"""
        for entry in range(len(self.packet['Entry'])):
            if self.packet['Entry'][entry][0] == router_id:
                if self.packet['Entry'][entry][1] < 16:  # Verifies that the cost is not 16
                    # Used new metric even if it is larger than the old one
                    self.table[current_router] = [self.packet['Entry'][entry][1], current_router, False, 0, 0]

    def update_routing_table(self):
        """
        Implement a loop which create a packet for each router and sends its information to the neighbouring routers send
        routing information (destination and distance to the destination) to neighbour

        Check information from neighbour 'G'
        Add cost associated with G, call result 'D'
        Compare result distance with current entry in table
        If distance D is smaller than current distance update table entry to have new metric 'D'
        If G' is the router from which the existing router came, then use new metric even if it is larger than the old one
        """
        current = self.packet['Header'][2]
        neighbours = []
        for entry in self.packet['Entry']:  # Adds the destinations of the routers into a list 'neighbours'
            neighbours.append(entry[0])

        if current not in self.table.keys():  # Checks if the router is present in the table and if not adds it
            self.update_entry_cost(current)

        for neighbour in range(len(neighbours)):  # iterates through the list of destinations
            n = neighbours[neighbour]
            if n == router_id:  # checks if the destination is equal to the current router and if yes adds the router to the table
                self.update_entry_cost(current)
            else:
                cost = min(self.packet['Entry'][neighbour][1] + self.table[current][0],
```

```python
                    INFINITY)  # Adding the cost associated with neighbour

            if n not in self.table.keys():  # check if the destination router is not in the table and if not then adds it
                if cost >= 16:
                    continue
                else:
                    self.table[n] = [cost, current, False, 0, 0]

            elif current == self.table[n][1]:  # checks if current router is equal to the next hop and if yes then changes the cost
                if self.table[n][0] == 16:
                    continue
                else:
                    self.table[n] = [cost, current, False, 0, 0]

            elif cost < self.table[n][0]:  # Compare result distance with current entry in the table
                self.table[n][
                    0] = cost  # Since distance is smaller than current distance, new metric is the distance
                self.table[n][1] = current
        return self.table

    def response_messages(self, sockets, timeout):
        """Processes the response messages for the packet received
          Reasons for response to be received:
        - Response to specific query
        - Regular update
        - Triggered update caused by a route change

        Validity checks - Datagram
        Response is ignored if it is not from RIP port
        Check datagram source address is from valid neighbour, source of datagram must be on a directly-connected network
        Check response is from one of the routers own addresses
        Ignore if a router processes its own output as a new input

        Validity checks - RTEs (entry)
        Check if destination address valid - unicast, not net 0 or 127
        Check if metric valid - Must be between 1 and 16 (inclusive)

        Check if explicit route for destination address

        First run loop to wait for packets to be received
        """

        packet_table = self.table
        read, write, err = select.select(sockets, [], [], timeout)  # Listens to all input ports simultaneously
        if len(read) > 0:
            for i in read:
                rec_packet_raw = i.recvfrom(1023)  # Receives a packet with buffer size 1023
                message_packet = rec_packet_raw[0].decode('ascii')  # Decodes the received packet
                message_packet_dict = json.loads(message_packet)  # Convert string to dictionary
                self.packet = message_packet_dict
                valid_header = check_packet_header(message_packet_dict)
                valid_entry = check_packet_entry(message_packet_dict)
                print('Packet Received')
                if valid_header and valid_entry:  # If passes the validity checks then the routing table updates
                    packet_table = self.update_routing_table()
                else:
                    print('Dropped invalid packet')
        return packet_table

    def print_routing_table(self):
        """Prints routing table in pretty format"""
        print("==" * 40)
        print("                    Routing table for router {}".format(router_id))
        print("Destination     Metric      Next Hop      Timer      Garbage Timer")
        for key, data in sorted(self.table.items()):
            print(
```

```python
            "{:^12d} {:^14d} {:^12d} {:^12d} {:^14d}".format(key, data[0], data[1], data[3], data[4]))
    print("==" * 40)


""" HELPER FUNCTIONS"""


def open_file():
    """Reads the content of the file and formats it into a table"""
    global router_id
    arguments = sys.argv
    filename = arguments[1]
    if len(arguments) != 2:
        print("Invalid number of arguments.\n Please enter in format: python3 routing.py config_file_(number)")
        exit()
    with open(filename) as f:  # Opens config file and formats it
        contents = f.readlines()
        routerIdRaw = contents[0]
        inputPortsRaw = contents[1]
        outputPortsRaw = contents[2]

        routerIdList = routerIdRaw.strip().split(", ")
        inputPortsList = inputPortsRaw.strip().split(", ")
        outputPortsList = outputPortsRaw.strip().split(", ")

        router_id = int(routerIdList[1])
        table = {}  # Creates a table as a dictionary

        if 1 > int(routerIdList[1]) or int(
                routerIdList[1]) > 64000:  # Verifies that the router id is within the specified range
            print("ERROR: Router-id must be between 1 and 64000")
            exit()
        for i in range(1, len(inputPortsList)):
            # Verifies that the input ports are within the specified range
            if 1024 >= int(inputPortsList[i]) or int(inputPortsList[i]) >= 64000:
                print("ERROR: Port number {0} must be between 1024 and 64000".format(inputPortsList[i]))
                exit()
            else:
                inputPorts.append(inputPortsList[i])

        if len(inputPorts) > len(set(inputPorts)):  # Verifies that the input ports are all unique
            print("ERROR: Every input port number must be unique")
            exit()

        for j in range(1, len(outputPortsList)):
            output = outputPortsList[j].split('-')
            # Verifies that the output ports are within the specified range
            if 1024 >= int(output[0]) or int(output[0]) >= 64000:
                print("ERROR: Port number {0} must be between 1024 and 64000".format(output[0]))
                exit()
            else:
                outputPorts.append(output[0])
            flag = False
            timer = 0
            garbageTime = 0
            table[int(output[2])] = [int(output[1]), int(output[2]), flag, timer, garbageTime]
            # output[1] = metric/cost
            # output[2] = destination id

        if len(outputPorts) > len(set(outputPorts)):  # Verifies that the output ports are unique
            print("ERROR: Every output port number must be unique")
            exit()

        for i in range(0, len(inputPorts)):
            for j in range(0, len(outputPorts)):
                if inputPorts[i] == outputPorts[j]:
```

```python
                print("Port {0} in input and output ports must be unique".format(inputPorts[i]))
                exit()

    return table


def check_packet_header(packet):
    """Verifies that the packet header format is as it should be"""
    checkHeader = True
    if int(packet['Header'][0]) != 2 or int(packet['Header'][1]) != 2:  # Check command and version are 2
        checkHeader = False
    elif int(packet['Header'][2]) < 1 or int(packet['Header'][2]) > 64000:  # Check header port is within range
        checkHeader = False
    return checkHeader


def check_packet_entry(packet):
    """Verifies that the packet entry format is as it should be"""
    checkEntry = True
    for entry in packet['Entry']:
        if int(entry[1]) > 16:  # checks if cost is greater than 16
            checkEntry = False
        elif int(entry[0]) < 1 or int(entry[0]) > 64000:  # Checks if the destination ports are within the specified range
            checkEntry = False
    return checkEntry


def update_timers(main_table):
    """Function which deals with checking the timer values in accordance to what should occur on the routing table"""
    for id in sorted(main_table.keys()):  # Iterates through the routers in the table
        if main_table[id][3] >= TIMEOUT:  # Checks if the timer exceeds the timeout value
            main_table[id][0] = INFINITY  # Sets metric to infinity
            main_table[id][2] = True  # Sets flag to true
        else:
            main_table[id][3] += 1  # Timer goes up by 1

        if main_table[id][2]:  # Checks if the flag is true, and if yes increments the garbage timer
            main_table[id][4] += GARBAGE_TIMER
            if main_table[id][4] >= GARBAGE_TIMEOUT:  # If garbage timer value reaches max, then removes the route from the
table
                del main_table[id]


def main(main_table, new_socket):
    """The main function of the file which runs the program"""
    timer = 0
    random_timer = random.randint(1, 10)  # Offsetting the 30-second timer by a random interval (30 +- 5)

    sockets = new_socket.open_socket()

    while True:
        timer = timer + 1
        # Initialises the packet class and creates a new packet
        newPacket = Packet(main_table)
        packet = newPacket.create_packet()

        # Initialises the routing class and prints the routing table every timer tick
        routing = Routing(main_table, packet)
        routing.print_routing_table()

        # Main part of the function which sends packets and increments the timer values according to the conditions specified
        if timer == random_timer:  # Waits for a random interval of 5+/-5 seconds and then sends a packet while resetting the timer
            newPacket.send_packet(packet)
            timer = 0

        update_timers(main_table)
```

```python
        main_table = routing.response_messages(sockets, timeout=1)


if __name__ == "__main__":
    # Parses through the config file and formats the data into a table
    file = open_file()

    # Open the sockets for input ports and gets them ready for receiving packets
    sockets = Socket(inputPorts, file)
    try:
        main(file, sockets)
    except (KeyboardInterrupt, SystemExit):
        sockets.close_socket() # Closes all the sockets of the router
        print("\nRouter Shut Down\n")
        exit()
```