COSC 364
Internet Technologies and Engineering
RIP Assignment

Bryson Chen 32687456

Luke Morimoto 33883343

Contributions

For this assignment we worked on it equally 50/50. We did this by working on the same computer. This was done so that we could avoid merge errors, understand the code equally and generally discuss what we were doing at all times.

We worked systematically through our coding process and completed in this order:

- Configuration files and parsing
- Socket binding
- Creating packets
- Updating routing table / Bellman-ford algorithm
- Parsing incoming packets
- Printing routing table
- Main-looping functionality / sending and receiving packets
- Split-horizon with poisoned reverse
- Testing and debugging
- Commenting and writing report

We also took turns coding certain ideas to help with typing workload and allowing for ideas to be further explained.

Testing

Configuration files

This assignment begins with running the program, main.py, using the command line with a configuration file name as an argument. Checks have been implemented to ensure that an argument has indeed been provided in the command line and that the argument is a file that exists.

For the configuration files themselves, a considerable amount of checks during parsing is needed such that the configuration files contain the required parameters in the correct format.

There are three required parameters: router-id, input-ports, and outputs.

router-id needs to be in the format "**router-id X**" where **X** is an integer between 1 and 64,000, inclusive. The value for router-id must also be unique between all the router instances and this will be confirmed manually as opposed to being checked in the code.

input-ports needs to be in the format "**input-ports ports**" where **ports** refers to a group of comma-separated integers between 1024 and 64,000, inclusive. Each one of those integers represents an input port and must be unique.

outputs needs to be in the format "**outputs a-b-c**" where **a** refers to an input port of a peer router, **b** refers to the metric cost of the link, and **c** refers to the router-id of the peer router. The values for **a** and **c** must conform to the requirements as stated for the router-id and input-ports parameters, but **a** must also not be an input port. The metric, **b**, must conform to the RFC specifications [1], meaning that it must be an integer between 1 and 15, inclusive. There can be multiple **a-b-c** triples provided but they must be separated by commas.

In addition to the required parameters, timer is an optional parameter and if provided, must be in the format "**timers time**" where **time** is an integer between 1 and 30 inclusive, representing the timer for periodic updates. The timeout timer will be 6 times this supplied value and the garbage timer will be 4 four times the provided value, such that the timers will be in the ratio of 1:6:4. If this parameter is not provided, default values for timers will be used.

Testing the configuration file parser involved creating multiple versions of configuration files with valid cases, edge cases, values out of range, string and float inputs, as well has not including the required parameters. The results of our testing can be observed in Table 1.

Table 1: Comprehensive testing of the file parsing function

Test Case	**Expected Result**	**Actual Result**
Valid Case router-id 1 input-ports 1111, 1112, 1113 outputs 5000-2-2, 5001-2-3	Success	Success
router-id not provided input-ports 1111, 1112, 1113 outputs 5000-2-2, 5001-2-3	Missing params: ['router-id']	Missing params: ['router-id']
input-ports not provided router-id 1 outputs 5000-2-2, 5001-2-3	Missing params: ['input-ports']	Missing params: ['input-ports']
outputs not provided router-id 1 input-ports 1111, 1112, 1113	Missing params: ['outputs']	Missing params: ['outputs']
Nothing provided	Missing params: ['router-id', 'input-ports', 'outputs']	Missing params: ['router-id', 'input-ports', 'outputs']
Invalid router-id router-id	Error parsing router-id	Error parsing router-id
Invalid router-id router-id 0	router-id in the config needs to be between 1 and 64000	router-id in the config needs to be between 1 and 64000
Edge case for router-id router-id 1 input-ports 1111, 1112, 1113 outputs 5000-2-2, 5001-2-3	Success	Success
Edge case for router-id router-id 64000 input-ports 1111, 1112, 1113 outputs 5000-2-2, 5001-2-3	Success	Success
Invalid router-id router-id 64001	router-id in the config needs to be between 1 and 64000	router-id in the config needs to be between 1 and 64000
Invalid router-id	router-id in the config was not an integer	router-id in the config was not an integer

router-id a		
Invalid router-id router-id 1.2	router-id in the config was not an integer	router-id in the config was not an integer
Invalid input-ports input-ports	Line contains "input-ports" but no ports	Line contains "input-ports" but no ports
Invalid input-ports input-ports 1023	Invalid input port number was provided	Invalid input port number was provided
Edge case input-ports 1024 outputs 5000-2-2, 5001-2-3 router-id 1	Success	Success
Edge case input-ports 64000 outputs 5000-2-2, 5001-2-3 router-id 1	Success	Success
Invalid input-ports input-ports 64001	Invalid input port number was provided	Invalid input port number was provided
Invalid input-ports input-ports 1024, a	Port in the config was not an integer	Port in the config was not an integer
Invalid input-ports input-ports 1024.2 outputs 5000-2-2, 5001-2-3 router-id 1	Port in the config was not an integer	Success This was not the expected result and was due to the code using a ‘try and except’ statement as a catch-all for any errors that occur when converting the input ports to integers. In Python, floats can be converted to integers without producing an error. In order to circumvent this, each of the ports needs to be checked using the isnumeric() method, which was used when checking router-id. This method will also be used for the checking of outputs
Duplicate input port	Invalid input port number	Invalid input port number

input-ports 2000, 2000	was provided	was provided
Invalid output port outputs 1-2-2	Output port out of range	Output port out of range
Edge case output port outputs 1024-2-2	Success	Success
Edge case output port outputs 64000-2-2	Success	Success
Invalid output port outputs 64001-2-2	Output port out of range	Output port out of range
Invalid output port outputs 6400.1-2-2	Provided outputs must be integers	Provided outputs must be integers
Invalid metric outputs 1024-0-2	Metric out of range	Metric out of range
Edge case metric outputs 1024-1-2	Success	Success
Edge case metric outputs 1024-15-2	Success	Success
Invalid metric outputs 1024-16-2	Metric out of range	Metric out of range
Invalid metric outputs 1024-a-2	Provided outputs must be integers	Provided outputs must be integers
Same output port as input port outputs 1024-1-2 input-ports 1024 router-id 1	Output port cannot be in input ports	Output port cannot be in input ports
Invalid output router id outputs 1024-1-0	Output router-id out of range	Output router-id out of range
Edgecase for router id	Success	Success

outputs 1024-1-64000		
Invalid output router id outputs 1024-1-64001	Output router-id out of range	Output router-id out of range
Invalid output router id outputs 1024-1-12.4	Provided outputs must be integers	Provided outputs must be integers
Two outputs to same router-id outputs 1024-15-2, 1026-1-2 input-ports 1025 router-id 1	Success	Success Note: When two outputs are provided to the same router-id, the one with the lower cost will be taken
Valid timer router-id 1 input-ports 1111, 1112, 1113 outputs 5000-2-2, 5001-2-3 timers 1	Success	Success
Invalid timer timers 0	Timer must be positive and less than or equal to 30	Timer must be positive and less than or equal to 30
Edge case timer timers 30	Success	Success
Invalid timer timers 31	Timer must be positive and less than or equal to 30	Timer must be positive and less than or equal to 30
Invalid timer timers 1.111	Provided timer must be an integer	Provided timer must be an integer
Timer parameter without value timers	Line contains "timer" but no value for the timer	Line contains "timer" but no value for the timer

Creating and parsing packets

Functions were needed to create and parse packets. We first started by checking if the creation of packets was correct. This involved manually creating a packet, byte by byte, based on the RIPv1 packet format [1], and translating the process into Python using byte arrays.

The common header consists of 4 bytes in total, with 2 of them being constant for all packets. The first byte represents the command of the packet with a value of 1 representing that a packet is a request packet, and a value of 2 representing that the packet is a response packet [1]. The assignment specification [2] states that the only packets that should be used are response packets, so the first byte can be a fixed value of 2. Similarly, the second byte represents the version and this will also be set to 2. The third and fourth bytes of the header would usually be 0 bytes, however, for the sake of this assignment, they will be used to represent the router-id of the router from which the packet originates. Because a byte can only represent a maximum integer of 255, and router-id can range from 1 to 64000, bitwise operations must be performed on router-id to ensure that it fits in the 2 byte header space.

Likewise, the RIP entries have a set of fixed fields, with the only variable fields being IP address, and metric. For the case of this assignment, the IP address field will instead represent the router-ids of neighbours. Both variable fields are 4 bytes, which is excessive for the ranges that we wish to represent. Since the maximum value router-id can be is 64,000 and two bytes is sufficient to represent that value, the first 2 bytes of the router-id field must always be set to 0. Similarly to the router-id in the header, the router-id in the RIP entries will also be allocated to 2 bytes using bitwise operations. For the case of metric, the maximum value that should be represented is 16, the value for infinity. This means 1 byte is sufficient to represent the metric. So, the metric field will always have 3 zero bytes. However, creating the packet isn't as simple as scouring through the routing table and extracting the router-id and metric fields to place into the packet; split horizon with poisoned reverse had to be implemented as well. This meant that each neighbouring router should receive a separate packet. In order to implement split horizon with poisoned reverse, the create_packet function needed to know the recipient router-id as well. So, for every RIP entry in a router's table, it'd check if the next-hop is the same as the recipient router-id, in which case it'd set the metric to 16.

Testing the create_packet function involved using the parse_packet function, which checks the fields of the packet and extracts the RIP entries. Firstly, the function would check the validity of the header, ensuring that the 2 fixed fields are as expected and that the 2 byte router-id field is within the range of 1 to 64,000. Should the packet fail the test, the packet will be dropped. Next, the function parses each RIP entry separately. Should the entry's fixed fields or router-id field fail the check, instead of being dropped, the function simply continues onto the next entry. The only check for metric is much less stringent as it accepts all values from 1 to 16, with anything above 16 being set to 16 when extracted.

Assessing the functionality of these two functions is tied in with routing table convergence as the tables would only converge to the expected values if both the creation of packets and parsing of packets were working as expected.

Starting up router and updating routing table

Once the functionality for the sending and receiving packets was added, we tested to ensure that they were appropriately being read.

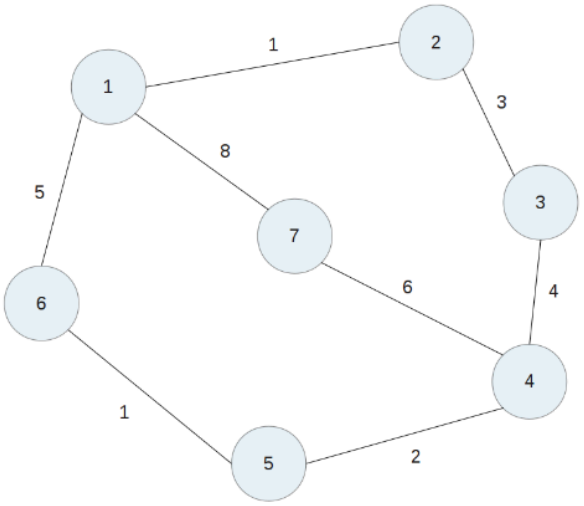


Figure 1: Example network for demonstration [2]

We did this by running the python file through multiple bash terminals and using 7 different config files in order to make the example network for the demonstration (Figure 1). We expected to see the displayed routing tables update with the new routers and converge to the correct values.

With the initial routing table, we had it display the sending router's information. This was done to ensure that it was properly in the routing table and that this information was to be sent to other routers in order to initialise their tables.

```
----------------------------Routing Table 3-------------------------------
  Router ID   |    Next Hop    |     Cost     |    Timeout    |     Garbage Time
     3        |       3        |      0       |     0.00      |       0.00
-------------------------------------------------------------------------------
```

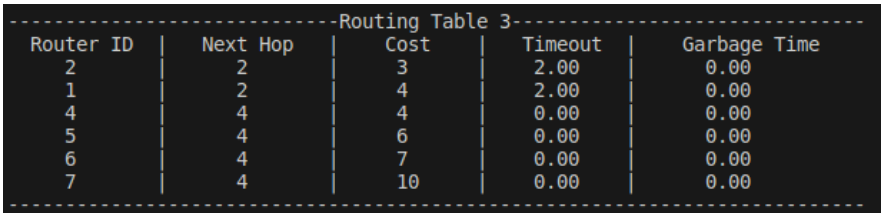
Figure 2: Initial routing table for router 3

This confirmed to us that the routing table indeed had the correct information about its own router. We next tested for the information that was being received to check if the updates were correct. By turning routers on one at a time, starting with neighbouring routers, we could control the flow of information and check to see what we expected the outcome to be. We turned on routers 1 and 2 next. We expected to see both routers have a next hop of 2 and a cost of 4 and 3 respectively.

```
----------------------------Routing Table 3-------------------------------
  Router ID   |    Next Hop    |     Cost     |    Timeout    |     Garbage Time
     3        |       3        |      0       |     4.00      |       0.00
     2        |       2        |      3       |     0.00      |       0.00
     1        |       2        |      4       |     0.00      |       0.00
-------------------------------------------------------------------------------
```

Figure 3: Updated routing table for router 3

The routing tables successfully updated with the new routers, cost and next hop as shown in Figure 3. These values were in agreement with the values that we calculated for their tables, allowing us to confirm our Bellman-ford algorithm was correct. Upon starting up all routers, we could confirm that the tables converged to their expected values as seen in Figure 4.

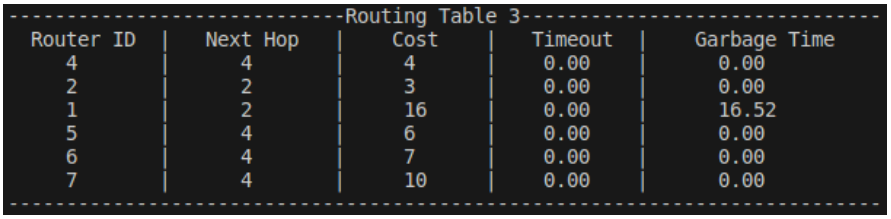


```
----------------------------Routing Table 3-------------------------------
 Router ID  |   Next Hop   |    Cost    |   Timeout   |    Garbage Time
    2       |      2       |     3      |    2.00     |      0.00
    1       |      2       |     4      |    2.00     |      0.00
    4       |      4       |     4      |    0.00     |      0.00
    5       |      4       |     6      |    0.00     |      0.00
    6       |      4       |     7      |    0.00     |      0.00
    7       |      4       |     10     |    0.00     |      0.00
--------------------------------------------------------------------------
```

Figure 4: Fully converged routing table for router 3

Testing Router Crashes

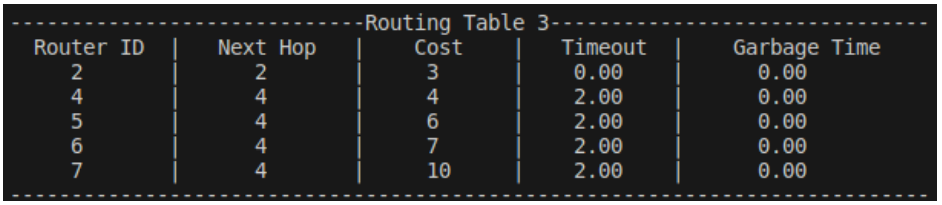
Testing how routers responded to router crashes involved setting up all routers required for the example network, as seen in Figure 1, choosing a router instance to shut off, and then observing the behaviour of the remaining routers. In this case, we chose to shut off Router 1. This was our biggest testing point as from initial testing we ran into some errors. We expected to see Router 2, Router 6, and Router 7 change metric to 16 after timing out, sending triggered updates out to neighbours, and then dropping the router after the garbage timer timed out. Instead the neighbouring routers' entries for Router 1 did not end up timing out, and was instead being set to an incorrect metric. This was not the expected result as seen in Figure 5. We realised this was due to a misunderstanding of split-horizon with poisoned reverse which resulted in a counting-to-infinity problem. This misunderstanding meant that instead of implementing split-horizon with poisoned reverse, we had only set up triggered updates properly. This meant that Router 2 thought that it could still reach Router 1 through Router 3 and back through itself, thereby forming a routing loop.



```
----------------------------Routing Table 3-------------------------------
 Router ID  |   Next Hop   |    Cost    |   Timeout   |    Garbage Time
    4       |      4       |     4      |    0.00     |      0.00
    2       |      2       |     3      |    0.00     |      0.00
    1       |      2       |     16     |    0.00     |      16.52
    5       |      4       |     6      |    0.00     |      0.00
    6       |      4       |     7      |    0.00     |      0.00
    7       |      4       |     10     |    0.00     |      0.00
--------------------------------------------------------------------------
```

Figure 5: Expected behaviour for Routing table 3 when router 1 crashes

Once this was fixed by properly implementing split-horizon with poisoned reverse, we got the expected outcome of the router changing the cost to 16 and proceeding to drop it after the garbage collection timer timed out as seen in Figure 6.



```
----------------------------Routing Table 3-------------------------------
 Router ID  |   Next Hop   |    Cost    |   Timeout   |    Garbage Time
    2       |      2       |     3      |    0.00     |      0.00
    4       |      4       |     4      |    2.00     |      0.00
    5       |      4       |     6      |    2.00     |      0.00
    6       |      4       |     7      |    2.00     |      0.00
    7       |      4       |     10     |    2.00     |      0.00
--------------------------------------------------------------------------
```

Figure 6: Updated Routing table 3 after Router 1’s entry is removed

Originally the routers were not propagating the 16 cost because we had the routers configured to only replace their metrics when the incoming metric is less than the value stored in the table. It was fixed by making routers replace their own entries whose next hop is the same as the router-id of the sender of an update packet, whether or not the cost is higher.

To check further we also ‘restarted’ a router to see what would happen. We needed the neighbouring routers to be able to update its cost to 16 after a timeout period but if it was turned back on, update back to the valid cost. This was working but due to triggered updates only being used for router crashes, this update would propagate more slowly through periodic updates only.

Example Config File

```
router-id 1
input-ports 5001, 5002, 5003
outputs 5004-5-6, 5005-8-7, 5006-1-2
```

Code

**attached below*

References

[1] G. Malkin. RIP Version 2. RFC 2453, 1998.
[2] A. Willig. Rip Assignment description, 2024.

~/Documents/Cosc364/rip-assignment/rip-assignment/main.py

```
"""
Title: Rip Assignment 2024
Author: Byrson Chen (32687456) | Luke Morimoto (33883343)
Date: 22/04/2024
"""
import sys
import socket
import select
import time
import random


def file_parse(file_name: str):
    """Parses the file provided to extract router_id, input_ports, and outputs"""
    router_id = None
    input_ports = []
    outputs = []
    timers_output = None
    try:
        file = open(file_name, "r")
        line = file.readline()
    except FileNotFoundError:
        print("The file name provided as an argument could not be found")
        exit()
    except:
        print("An error occurred when opening the file")
        exit()

    while line:
        # Do some parsing
        if "router-id" in line:
            # Eg. Parsing: router-id 1
            try:
                router_id = line.split()[1]
            except:
                print("Error parsing router-id")
                exit()

            if (not router_id.isnumeric()):
                print("router-id in the config was not an integer")
                exit()
            else:
                router_id = int(router_id)
                if (router_id > 64000 or router_id < 1):
                    print("router-id in the config needs to be between 1 and
64000")
                    exit()
        elif "input-ports" in line:
            # Holds a string of comma separated input ports
            # Eg. input-ports 6110, 6201, 7345
            try:
                input_ports_string = line.split(" ", 1)[1]
            except:
                print("Line contains \"input-ports\" but no ports")
                exit()
            # Ports all need to be unique and be between 1024 and 64000 inclusive
            input_ports = input_ports_string.split(",")
```

```
57
58              for i in range(len(input_ports)):
59                  if (not input_ports[i].strip().isnumeric()):
60                      print("Port in the config was not an integer")
61                      exit()
62                  else:
63                      input_ports[i] = int(input_ports[i])
64
65              for i in input_ports:
66                  if ((i > 64000 or i < 1024) or input_ports.count(i) > 1):
67                      print("Invalid input port number was provided")
68                      exit()
69
70          elif "outputs" in line:
71              # Holds a string of comma separated outputs
72              # Eg. outputs 5000-1-1, 5002-5-4
73              try:
74                  outputs_string = line.split(" ", 1)[1]
75              except:
76                  print("Line contains \"outputs\" but none provided")
77                  exit()
78              outputs_triple = outputs_string.split(",")
79
80              outputs_ports_list = []
81              outputs_routers_list = []
82
83              for i in outputs_triple:
84                  current_triple = i.strip().split("-")
85                  if len(current_triple) != 3:
86                      print("Error in outputs")
87                      exit()
88                  try:
89                      for j in range(3):
90                          if (not current_triple[j].isnumeric()):
91                              exit()
92                          else:
93                              current_triple[j] = int(current_triple[j])
94                  except:
95                      print("Provided outputs must be integers")
96                      exit()
97
98                  if current_triple[0] > 64000 or current_triple[0] < 1024:
99                      print("Output port out of range")
100                     exit()
101                 if current_triple[1] < 1 or current_triple[1] > 15:
102                     print("Metric out of range")
103                     exit()
104                 if current_triple[2] > 64000 or current_triple[2] < 1:
105                     print("Output router-id out of range")
106                     exit()
107
108                 if current_triple[0] not in outputs_ports_list:
109                     outputs_ports_list.append(current_triple[0])
110                 else:
111                     print("Output ports are not all unique")
112                     exit()
113
114
115
116                 if current_triple[2] not in outputs_routers_list:
```

```
                    outputs_routers_list.append(current_triple[2])
                    outputs.append(current_triple)
                else:
                    # if the current output router-id is in outputs already, replace the current entry if the current triplet's cost is less
                    for i in range(len(outputs)):
                        if outputs[i][2] == current_triple[2] and outputs[i][1] > current_triple[1]:
                            outputs[i] = current_triple

        elif "timers" in line:
            try:
                timers_string = line.split()[1]
            except:
                print("Line contains \"timer\" but no value for the timer")
                exit()

            try:
                if (not timers_string.isnumeric()):
                    exit()
                else:
                    timers_string = int(timers_string)
            except:
                print("Provided timer must be an integer")
                exit()
            if timers_string <= 0 or timers_string > 30:
                print("Timer must be positive and less than or equal to 30")
                exit()


            timers_output = [timers_string, timers_string*6, timers_string*4]



        line = file.readline()


    file.close()

    if router_id is not None and input_ports and outputs:
        for i in outputs:
            if i[0] in input_ports:
                print("Output port cannot be in input ports")
                exit()
            if i[2] == router_id:
                print("Output router-id cannot be the same as router-id")
                exit()
        return (router_id, input_ports, outputs, timers_output)

    else:
        missing_params = []
        if router_id == None:
            missing_params.append("router-id")
        if not input_ports:
            missing_params.append("input-ports")
        if not outputs:
            missing_params.append("outputs")

        print("Missing params: {}".format(missing_params))
        exit()
```

```
175
176 def socket_bind(input_ports):
177     """Binds a socket to each input port and returns them in a list"""
178     sockets = []
179
180     for i in input_ports:
181         current_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
182         current_socket.bind(('127.0.0.1', i))
183         sockets.append(current_socket)
184
185     return sockets
186
187
188 def create_packet(router_id, routing_table, recipient_id):
189     """Creates a packet based on the router-id, routing table entries of a router,
    and the recipient router-id"""
190     output_packet = bytearray()
191
192     # Command = 2 for response, Version = RIPv2
193     output_packet.append(2)
194     output_packet.append(2)
195
196     # Bitshift operations on the router-id field
197     output_packet.append(router_id >> 8)
198     output_packet.append(router_id - ((router_id >> 8) << 8))
199
200     # Goes through each router-id in the routing table
201     for i in routing_table.keys():
202
203         # Each RIP entry needs to have 2 for AFI (for IP) followed by 2 zero bytes
204         output_packet.append(0)
205         output_packet.append(2)
206         output_packet.append(0)
207         output_packet.append(0)
208
209         # router-id of the entry
210         output_packet.append(0)
211         output_packet.append(0)
212         output_packet.append(i >> 8)
213         output_packet.append(i - ((i >> 8) << 8))
214
215         # 8 zero bytes
216         for j in range(8):
217             output_packet.append(0)
218
219         # metric of the entry
220         for j in range(3):
221             output_packet.append(0)
222
223         # Poisoned Reverse if the next hop for this entry is through the recipient
    router
224         if routing_table[i][0] == recipient_id:
225             output_packet.append(16)
226         else:
227             output_packet.append(routing_table[i][1])
228
229     return output_packet
230
231 def update_routing_table(sender_router_id, routing_table, rip_entries, outputs,
    sockets, router_id):
232     """Updates the routing table using the RIP entries provided from a packet"""
```

```
    table = routing_table.copy()

    # If the router's table doesn't have an entry for this neighbour
    if (sender_router_id not in table.keys()):
        neighbour_cost = 0
        for i in outputs:
            if i[2] == sender_router_id:
                neighbour_cost = i[1]

        if neighbour_cost == 0:
            #ie. sender is not a neighbour
            return table

        table[sender_router_id] = [sender_router_id, neighbour_cost, time.perf_counter(), time.perf_counter(), False]

    for i in rip_entries:
        # If routing table doesn't have this entry and the cost is less than 16
        if i[0] not in table.keys() and i[1] < 16:
            table[i[0]] = [sender_router_id, i[1] + table[sender_router_id][1], time.perf_counter(), time.perf_counter(), False]

        # If routing table doesn't have this entry but the cost is greater than or equal to 16 or the entry refers to itself
        elif (i[0] not in table.keys() and i[1] >= 16) or (i[0] == router_id):

            continue

        else:
            current_cost = table[i[0]][1]
            metric = None
            for j in outputs:
                if j[2] == sender_router_id:
                    metric = j[1]
            if metric == None:
                return table

            # If the router gets an update from the next hop router, it accepts it no matter what
            if table[i[0]][0] == sender_router_id:

                # If the garbage collection flag is set
                if table[i[0]][4]:
                    table[i[0]][1] = metric + i[1]
                    table[i[0]][2] = time.perf_counter()
                    if table[i[0]][1] >= 16:
                        table[i[0]][1] = 16
                    else:
                        table[i[0]][4] = False

                else:
                    table[i[0]][1] = metric + i[1]
                    table[i[0]][2] = time.perf_counter()

                    # Route is infinity; start garbage collection
                    if table[i[0]][1] >= 16:
                        table[i[0]][1] = 16
                        table[i[0]][4] = True
                        table[i[0]][3] = time.perf_counter()

                        # Create and send triggered update
```

```
                        for i in outputs:
                            packet = create_packet(router_id, table, i[2])
                            send_packet(packet, sockets[0], i[0])

            # If the new cost is better and is less than 16

            elif current_cost > (metric + i[1]) and (metric + i[1]) < 16:
                table[i[0]][0] = sender_router_id
                table[i[0]][1] = metric + i[1]
                table[i[0]][2] = time.perf_counter()
                table[i[0]][4] = False



    return table

def parse_packet(input_packet):
    """Parses the packet provided, ensuring that all its fields are valid, and extracting the RIP entries as [router-id, metric] pairs"""
    rip_entries = []
    try:
        packet_len = len(input_packet)

        # Check command and version fields
        if not (input_packet[0] == 2 and input_packet[1] == 2):
            return None

        # Extract the sender's router-id and checks if it is within range
        sender_router_id = (input_packet[2] << 8) + input_packet[3]

        if (sender_router_id > 64000 or sender_router_id < 1):
            return None


        if (packet_len-4) % 20 != 0:
            return None

        if packet_len > 25 * 20 + 4:
            return None

        for i in range((packet_len - 4) // 20):

            # AFI must be 2 for IP
            if not (input_packet[(20*i)+4] == 0 and input_packet[(20*i)+5] == 2):
                continue

            # Zero bytes
            if not (input_packet[(20*i)+6] == 0 and input_packet[(20*i)+7] == 0):
                continue

            # Extract router-id and check if it is within range
            router_id = (input_packet[(20*i)+8] << 24) + (input_packet[(20*i)+9] << 16) + (input_packet[(20*i)+10] << 8) + input_packet[(20*i)+11]
            if not (router_id <= 64000 or router_id >= 1):
                continue

            # Check if there are 8 zero bytes
            if not (input_packet[(20*i)+12] == 0 and input_packet[(20*i)+13] == 0 and input_packet[(20*i)+14] == 0 and input_packet[(20*i)+15] == 0):
                continue

```

```
348             if not (input_packet[(20*i)+16] == 0 and input_packet[(20*i)+17] == 0
    and input_packet[(20*i)+18] == 0 and input_packet[(20*i)+19] == 0):
349                 continue
350 
351             # Extract metric
352             metric = (input_packet[(20*i)+20] << 24) + (input_packet[(20*i)+21] <<
    16) + (input_packet[(20*i)+22] << 8) + input_packet[(20*i)+23]
353             if metric > 16:
354                 metric = 16
355             rip_entries.append([router_id, metric])
356 
357 
358         return (sender_router_id, rip_entries)
359 
360 
361     except:
362         return None
363 
364 
365 def send_packet(packet, sending_socket, port):
366     """Sends a packet to the specifed port"""
367     sending_socket.sendto(packet, ("127.0.0.1", port))
368 
369 
370 
371 def print_routing_table(routing_table, router_id):
372     """Displays the routing table in a human-readable format"""
373     print(f"-----------------------------Routing Table {router_id}----------------
    ---------------")
374     print("  Router ID  |   Next Hop   |    Cost    |   Timeout  |    Garbage
    Time")
375     for i in routing_table.keys():
376         current_router_id = i
377         next_hop = routing_table[i][0]
378         cost = routing_table[i][1]
379         timeout = time.perf_counter() - routing_table[i][2]
380         garbage_time = time.perf_counter() - routing_table[i][3]
381         flag = routing_table[i][4]
382 
383         # If the flag for garbage collection is True, display the garbage-timer
    and not the timeout-timer
384         if i != router_id:
385             if flag:
386                 print("{:^12} | {:^12} | {:^10} | {:^10.2f} | {:^15.2f}"
    .format(current_router_id, next_hop, cost, 0, garbage_time))
387             else:
388                 print("{:^12} | {:^12} | {:^10} | {:^10.2f} | {:^15.2f}"
    .format(current_router_id, next_hop, cost, timeout, 0))
389 
390     print("-------------------------------------------------------------------------
    ----")
391 
392 
393 def main_loop(sockets, routing_table, router_id, outputs, timers):
394     """The main loop which runs after the router configs have been set up."""
395     table = routing_table.copy()
396     print_routing_table(table, router_id)
397     while True:
398 
399         readable, writable, exceptional = select.select(sockets, [], sockets, 0.5)
400 
401         # When a packet is received on a socket
```

```
402         for current_socket in readable:
403             current_packet = current_socket.recvfrom(1024)[0]
404             result = parse_packet(current_packet)
405             if result is not None:
406                 sender_router_id = result[0]
407                 rip_entries = result[1]
408                 table = update_routing_table(sender_router_id, table, rip_entries,
    outputs, sockets, router_id)
409                 print_routing_table(table, router_id)
410 
411 
412         # If it is time for a periodic update
413         if time.perf_counter() - table[router_id][2] >= (random.uniform(0.8, 1.2)
    * timers[0]):
414             for i in outputs:
415                 packet = create_packet(router_id, table, i[2])
416                 send_packet(packet, sockets[0], i[0])
417             table[router_id][2] = time.perf_counter()
418 
419         garbage_values = []
420 
421         for entry in table.keys():
422             # If the entry is not itself
423             if entry != router_id:
424 
425                 # If the timeout timer has been exceeded and the garbage
    collection flag is not on
426                 if (time.perf_counter() - table[entry][2] >= timers[1]) and not
    table[entry][4]:
427                     table[entry][4] = True
428                     table[entry][3] = time.perf_counter()
429                     table[entry][1] = 16
430                     for i in outputs:
431                         packet = create_packet(router_id, table, i[2])
432                         send_packet(packet, sockets[0], i[0]) #Triggered update
433 
434                 # If the garbage collection flag is on and the timer has been
    exceeded
435                 if time.perf_counter() - table[entry][3] >= timers[2] and
    table[entry][4]:
436                     garbage_values.append(entry)
437 
438         for garbage in garbage_values:
439             table.pop(garbage)
440 
441         # If an entry has been deleted, print the resulting routing table
442         if len(garbage_values) > 0:
443             print_routing_table(table, router_id)
444 
445 def main():
446     if len(sys.argv) > 2:
447         print("This program only accepts 1 argument: The file name")
448 
449     elif len(sys.argv) == 1:
450         print("This program requires the file name as an argument")
451 
452     else:
453         file_name = sys.argv[1]
454         router_id, input_ports, outputs, timers = file_parse(file_name)
455         if timers == None:
456             timers = [5, 30, 20]
```

```
457         sockets = socket_bind(input_ports)
458 
459         routing_table = dict()
460         routing_table[router_id] = [router_id, 0, time.perf_counter(),
    time.perf_counter(), False]
461 
462         main_loop(sockets, routing_table, router_id, outputs, timers)
463 
464 
465 
466 main()
```