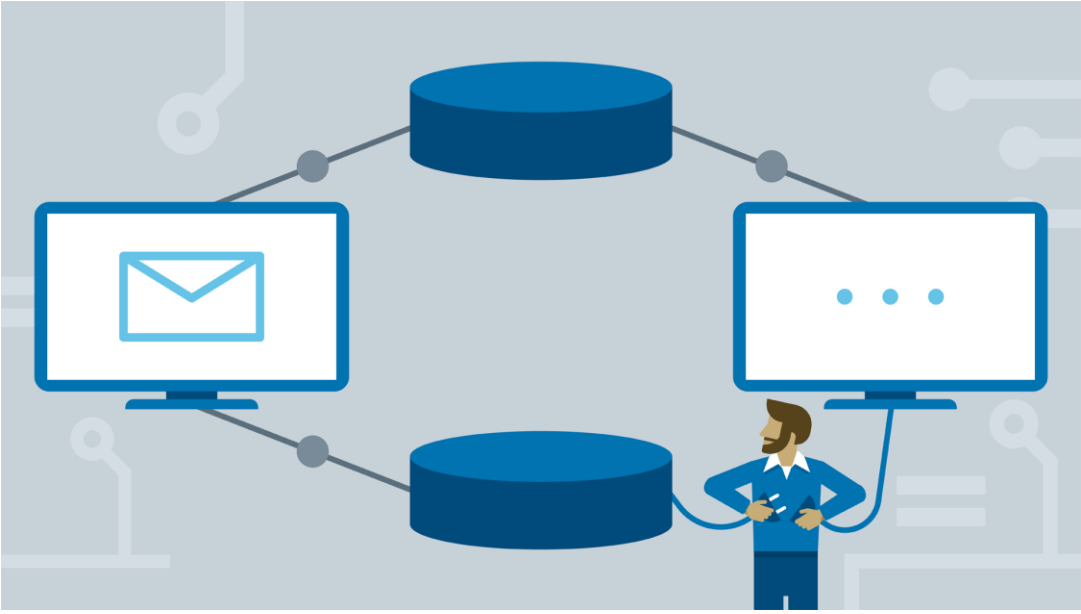
COSC364 ASSIGNMENT 1

Routing Demon RIPv2

(Routing Information Protocol)



Ahmad Alsaleh

Student ID: 14749959

Chadol Han

Student ID: 79364948

The following report contains the source code for the RIPv2 routing protocol written in Python 3, the report also discusses several issues regarding the protocol.

Percentage of contribution: We feel that the contribution was equal from both parties (50% percent from each person).

Well done aspects:- We feel that the structure for the routing protocol was done particularly well. Through many well defined functions, it was easier to maintain and catch errors as each function had a specific purpose. Thus testing and debugging these functions were made easier and simpler. Although this made the program contain many different functions, we felt this was the best way to proceed. - Another specific issue we dealt with in a correct way, was that the routing table was made using dictionaries in python, with all the information required in it (router id, cost, garbage flags and timers). This enabled us to pass this table throughout many functions and use it continuously with easy access to it.

Aspects that can be improved:- We felt that error detection for changes in the routing table would of helped us. Example: A cost from the configuration file of one router to another not matching the cost in another configuration file should raise a flag in the code and show an error. - Another aspect which we felt we could have improved was properly closing the ports that were being assigned to sockets and binded to ports. More than a few occasions we would get some errors regarding ports/addresses being already in use, this forced us to change the ports multiple times and reconfiguring the whole structure.

Atomicity of event processing: We ensured atomicity in our programming, by using the select module. The available updates are read and processed sequentially but won't always arrive in order. If there is an error in the update it must be discarded, so by using select() there will be no interruptions during the operations, and waits for the packet to be read. This ensures that all operations are completed before the processing of update occurs.

Testing:

- Basic testing:

The most basic test was running the program on different configuration settings (different ports, costs) and seeing if the program was calculating the correct costs and correct hope. The outcome was as expected, the calculations were done by hand to confirm the metric costs and next hop destination. The routers did indeed choose the least cost path and the correct destinations.

- Disconnecting a router:

This test was conducted to see the result what would happen if a router was disconnected (exited) from our 'network'. The expected result would that the program would run until it reached the timeout limit of 7 seconds, then it would update the hops, thus the routing table accordingly to not include the removed router. The outcome was inconsistent, some routers would drop the disconnected router once the router was dropped, some others would hold on to the router, assign them a cost of 16 (this error arose from our cost checking however

could not get it to function at 100%). We concluded that the routers do indeed know that the disconnected router does not exist and assigned them a cost.

- Disconnecting a router, then relaunching it:

A router was disconnected, then reconnected to the 'network'. The expected result was that the router would re-enter the network and update accordingly to the routing table. The outcome was that the new router would be recognised, however not updated due to what we believe might be incorrect implementation of the costs and the status of the router back into the routing table.

RIPv2_build_100.py

```
1.  ######################################
2.  ###  Ahmad Alsaleh (ID: 14749959) ###
3.  ###  Chadol Han (ID: 79364948    ###
4.  ###  COSC364 Assignment 1        ###
5.  ###  Routing Demon RIPv2         ###
6.  ######################################
7.
8.  #Importing modules
9.  import os
10. import socket
11. import select
12. import sys
13. import queue
14. import time
15. import random
16.
17. my_id = -1     #Global variable for router ID
18. output_p = {}  #Dictionary for output ports, k = port number, v = peer router
19. input_p = []   #Input ports list
20.
21. def show_table(table):
22.     '''Function to print the table in the terminal in a visible format'''
23.
24.     router_ids = sorted(table.keys())
25.     table_length = len(table)
26.
27.     ############################   TABLE FORMATTING   ############################
28.     print("\n\nRouter ID: {0}".format(my_id))
29.     print("| ID | Next Hop | Cost |Time Out |")
30.
31.     for i in range(0, table_length):
32.         info = table[router_ids[i]]
33.         print("| {:>2} |".format(router_ids[i]), end='')
34.         print(" {:>6}   | {:>2}   | {:<8.4f}|"\
35.             .format(info[0], info[1], float(str(info[3][0])[:8])))
36.
37.     print("\n")
38.     #Don't need to show input ports, too messy and dont need to see
39.     #print("Input ports currently active for router: {0}".format(my_id))
40.     #print(input_p)
41.     print("\n")
42.     print("Output ports | Peer router")
43.     for k, v in output_p.items():
44.         print("   ", k, "    |     ", v)
45.     print("\n")
46.     ############################   TABLE FORMATTING   ############################
47.
48. def state_firsthop(f_router, table):
49.     '''Function that returns the routers in which the routes in the routing
50.     table use the input f_router as first hop'''
51.
52.     routers = []
```

```
    router_ids = sorted(table.keys())
    for router in router_ids:
        if table[router][0] == f_router:
            routers.append(router)
    #print(routers)
    return routers

def rtable_build(filename):
    '''Function that reads the config text files (which contain the ports,
    metrics) and creates the routing table from the information inside'''

    global my_id
    total = []
    table = {}

    ############################  READING CONFIG FILE  ############################
    config = open(filename, 'r')
    for i in config.readlines():
        i = i.split(', ')
        total.append(i)
    ############################  READING CONFIG FILE  ############################

    my_id = int(total[0][1])

    for i in range(1, len(total[1])):
        input_p.append(int(total[1][i]))

    for i in range(1,len(total[2])):
        temp = total[2][i]
        temp = temp.split('-')
        router_id = int(temp[-1])
        port_num = int(temp[0])
        output_p[port_num] = router_id
        f_router = router_id
        cost = int(temp[-2])
        flag = False
        timers = [0, 0]
        table[router_id] = [f_router, cost, flag, timers]
    return table

def check_key_routerid(table):
    '''Returns the routerid's in a list from the routing table by reading the
    keys from the dictionary of the routing table'''

    routerid_keys = []
    t = table
    empty = None

     if t != empty:

        for y in t.keys():
            routerid_keys.append(y)
        #print(routerid_keys)
        return routerid_keys
```

```
107.
108.    elif t == empty:
109.        #print(routerid_keys)
110.        return routerid_keys
111.
112.
113. def listenlist():
114.    '''Function that binds the ports from the config file to sockets to which
115.    we need to listen'''
116.    llist = []
117.    input_port_length = len(input_p)
118.    ############################    BINDING PORTS    ############################
119.
120.    for i in range(0, input_port_length):
121.        #Create the socket
122.        sock = socket.socket(socket.AF_INET, #Internet
123.                             socket.SOCK_DGRAM) #UDP
124.        #sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
125.        #Bind to port
126.        sock.bind(('localhost', int(input_p[i]) ))
127.        llist.append(sock)
128.    #print(llist)
129.    return llist
130.
131.    ############################    BINDING PORTS    ############################
132.
133.
134. def id_checker(routerid_check, table):
135.    '''Function that takes the routerid as an input and checks whether it
136.    matches/exists in the routing table'''
137.    key = check_key_routerid(table)
138.
139.    if routerid_check in key:
140.        return True
141.    else:
142.        return False
143.
144.
145. def receiver(r_table, timeout):
146.    '''Function that checks if messages have been received on the sockets'''
147.
148.    socket_list = listenlist()
149.    table_key = []
150.    table = r_table
151.    a,b,c = select.select(socket_list,[],[], timeout)
152.
153.    if a != []:
154.        s = a[0]
155.        s.settimeout(7)
156.        #Receive the data
157.        data, addr = s.recvfrom(1024) #Buffer size is 1024 bytes
158.        data = str(data)
159.        data = data.replace("b","").replace("'","")
160.        data = data.split(',')
```

```
161.        src = int(data[2])
162.        print("Data received from router ID: {}".format(src))
163.        start = 3
164.
165.        while start < len(data):
166.            routerid_check = int(data[start])
167.            router_id = int(data[start])
168.
169.            try:
170.                if src in table:
171.                    cost = min(int(data[start + 1]) + table[src][1], 16)
172.                else:
173.                    print("Incorrect costs")
174.                    cost = 16
175.            except:
176.                print("COST ERROR")
177.
178.            if cost not in range(0,17):
179.                print("Packet is invalid: Cost not in range 0-16")
180.                while 1:
181.                    continue
182.
183.
184.            if router_id not in output_p.values() and router_id != my_id:
185.                if not id_checker(routerid_check, table):
186.                    table[router_id] = [src, cost, False, [0,0]]
187.
188.                if (cost < table[router_id][1]):
189.                    table[router_id][1] = cost
190.                    table[router_id][0] = src
191.
192.                if (src == table[router_id][0]):
193.                    table[router_id][1] = cost
194.                    table[router_id][0] = src
195.                    table[router_id][-1][0] = 0
196.                    table[router_id][-1][1] = 0
197.                    table[router_id][2] = False
198.
199.            start += 2
200.
201.
202.        if src in table:
203.            table[src][-1][0] = 0
204.            table[src][-1][1] = 0
205.            table[src][2] = False
206.
207.    return table
208.
209.def create_message(table, port):
210.    '''Function that creates a message that contains a dictionary representing
211.    the routing table and where it is being sent (port number)'''
212.
213.    t = table
214.    command = "2"
```

```
    version = "2"
    source = my_id
    key = check_key_routerid(t)
    head = command + ',' + version + ','+ str(source)
    final = head

    for v in key:
        value = t[v]
        final += ',' + str(v) + ','
        cost = str(value[1]) if v not in t.values() else 16
        final += cost
    #print(final)
    return final

def send_msg(table):
    '''Function to send message with updates to all neighbours'''

    #Create socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #UDP

    for port in output_p:
        msg = create_message(table, port)
        msg = msg.encode('utf-8')
        #Sending the data
        sock.sendto(msg, ("localhost", port))
        #print(msg)


def timer_update(table, time):
    '''Function to add time to routing table entry timers'''

    timeout_error = 7 #Timeout duration till it cuts off
    garbage_collection_timeout = 4
    for key in sorted(table.keys()):
        if table[key][2]:
            table[key][-1][1] += time
            if table[key][-1][1] > garbage_collection_timeout:
                del table[key]
        else:
            table[key][-1][0] += time
            if table[key][-1][0] > timeout_error:
                table[key][1] = 16
                table[key][2] = True
                for router in state_firsthop(key, table):
                    table[router][1] = 16
        #print(key)
        #print(table)
        #print(time) ERRORRRRR

```

RIPV2.py

```
 1.  ######################################
2.  ###  Ahmad Alsaleh (ID: 14749959) ###
3.  ###  Chadol Han (ID: 79364948)    ###
4.  ###  COSC364 Assignment 1         ###
5.  ###  Routing Demon RIPv2          ###
6.  ######################################
7.
8.  from RIPv2_build_100 import *
9.  def main():
10.    try:
11.        counter = 0
12.        r_table = rtable_build(sys.argv[1])
13.        #List index out of range. Still works
14.        #Attempt to fix by limiting arguments
15.        while True:
16.            max_time = 2 + random.randint(-2,2)
17.            timeout = max_time
18.            rec = time.time()
19.            elapsed = rec - time.time()
20.            timer_incr = 0
21.
22.            while elapsed < max_time:
23.                r_table = receiver(r_table, timeout)
24.                timer_incr = time.time() - rec
25.                rec = time.time()
26.                timer_update(r_table, timer_incr)
27.                elapsed += timer_incr
28.                timeout = max(max_time - elapsed, 0)
29.
30.            show_table(r_table)
31.            send_msg(r_table)
32.            counter += 1
33.    #If control-c pressed, stop the loop for 100 seconds
34.    except KeyboardInterrupt:
35.        time.sleep(100)
36.
37. if __name__ == "__main__":
38.    main()
39.
40. '''Final checks'''
41. #Check routing table 3, 5, 4, 7 Seems like 4 and 7 arent connecting
42. #Fixed connection between 4 and 7
43. #Address being used, socket error. Reset sockets?
44. #Changed ports > workaround to error
45.
```