COSC364

RIPv2 Routing Assignment

Team: Drogo Shi (msh217) & Haider Saeed (msa280)

Drogo Shi (50%):

Parts done:

- **Created Router () class which handles the functionality of a router.**
- **Implemented the creation, sending, and testing of packets.**
- **Implemented timers and their related functions.**
- **Implemented split horizon with poisoned reverse.**
- **Completed metric handling including the changing of metric with triggered updates.**
- **Created route handling and dealing with dead routers.**

Haider Saeed (50%):

Parts done:

- **Created Configure () class which reads and processes data from the configuration file.**
- **Created different configuration file test cases.**
- **Created and designed the printing of the routing table.**
- **Did commenting and cleaned code.**
- **Created the report and documentation.**
- **Implemented testing including socket receiving, binding, and sending functions.**

Question 1) Which aspects of your overall program (design or implementation) do you consider particularly well done?

To create the code, we had to treat the configuration file and the router separately. Therefore, the ‘Congifure’ class was made to only deal with the configuration files whereas the ‘Router’ class was made to deal with the routers and its related functions. To ensure code cleanliness and readability, creating these two classes was important. The design of the routing table is also clean, easy to read and informative. The actual functionality of the router works in the way it is supposed to. Any wrong packets are discarded, and any dead routers are deleted after their respective timers run out. The timers work well and seem to stop after they have reached their timer limit. All the fields in the routing table show accurate next hops and metrics along with their correct router ids. In the configure class, a lot of test cases were added to handle incorrect configuration files. These have many test cases like verifying the format and checking for mandatory fields before proceeding to creation and binding of sockets. If anything goes wrong, a detailed message is given to show what went wrong in the program.

Question 2) Which aspects of your overall program (design or implementation) could be improved?

Having a separated timer class could improve the design of the project. It would also improve code readability and separate the timers from the routers. The design of the program could be improved so that if we need to change parts of our program in the future, it can be easily done. This will also allow us to handle any errors separately according to their respective classes.

Question 3) How have you ensured atomicity of event processing?

The atomicity of event processing was ensured in a few ways. For example, the garbage timer only works if the timeout timer runs out. Separation of the two timer functions ensures that if one is being called, the other won’t occur. The sending and receiving of packets were also done separately. This meant that if the router was receiving packets, it wouldn’t interfere with the sending of the packets. In the sending packet functions, a random timer between 0.8 and 1.2 is set while sending packets to ensure atomicity of events.

Question 4) Have you identified any weaknesses of the RIP routing protocol?

The RIP Routing protocol has a couple of weaknesses. One of the major drawbacks of the RIP protocol is that it has maximum hop count of 15. This means that it won’t reach router farther 15 hops and will change to 16 which represents that a destination is unreachable. This makes RIP good for small networks but not so useful for larger networks.
The metrics in RIP cannot be changed and always remain static therefore, if any route metric is required to be changed, the configuration files must be manipulated instead and thus makes the RIP protocol incapable of being used in real time applications where the metrics are constantly changing.

RIP also allows the use of count of infinity where it changes the metric to 16 by incrementing 1 each time. However, this is not very useful as in some cases, the routing loops can carry on for a long time.
Also, if a router goes down in RIP, other routers must wait for a routing update before they know that a router is dead. This can take a while to calculate any alternative routes.

<u>Testing</u>

The testing section is also divided into two parts. The first part of the testing has to do with the configuration files themselves. These are the things being tested in each of the test configuration files. The results will give a detailed explanation on what went wrong with the configuration file during reading. These tests also check if the values of the metric, router ids, input and output port numbers are all accurate and lie within their allowed ranges. The tests also check for any missing mandatory field or values.

id_test1.txt - tests if router id < 1

id_test2.txt - tests if router id > 64000

ip_test1.txt - tests input port >= 1024

ip_test2.txt - tests input port <= 64000

ip_test3.txt - tests if all entries are not in one line

ip_test4.txt - tests if an input port is repeated

op_test1.txt - tests if output port number >= 1024

op_test2.txt - tests if output port number <= 64000

op_test3.txt - tests if output metric number >= 1

op_test4.txt - tests if output metric number <= 15

op_test5.txt - tests if output router id >= 1

op_test6.txt - tests if output router id <= 64000

op_test7.txt - tests to see if an output port number is in input port number

op_test8.txt - tests to see if all output port numbers are on one line

format_test1.txt - tests if missing router id parameter

format_test2.txt - tests if missing input port parameter

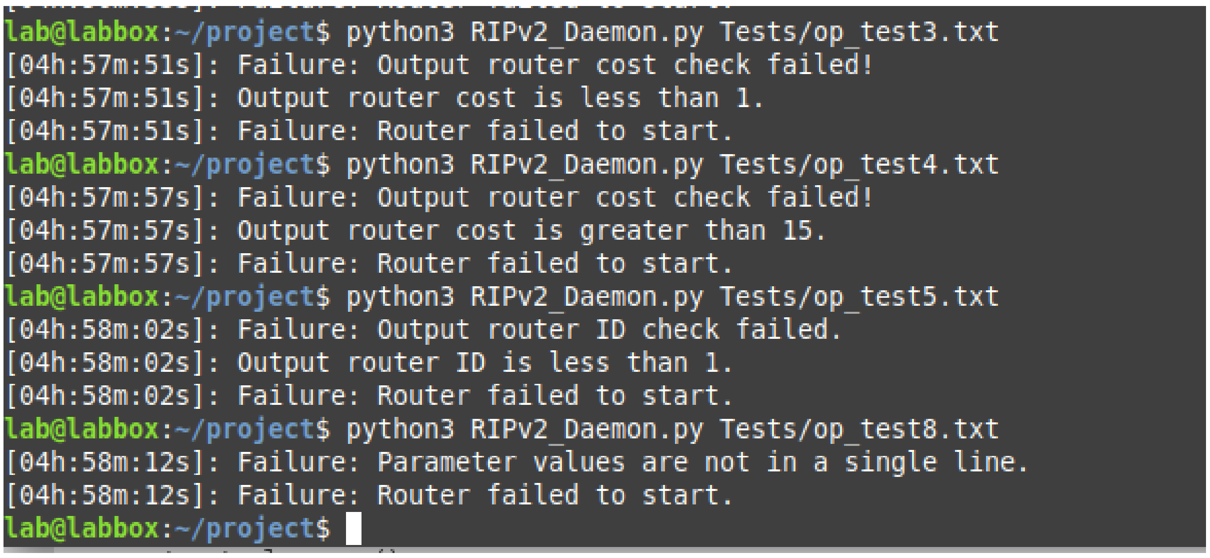
format_test3.txt - tests missing output port parameter

config_test1.txt - tests missing router id value

config_test2.txt - tests missing input ports values

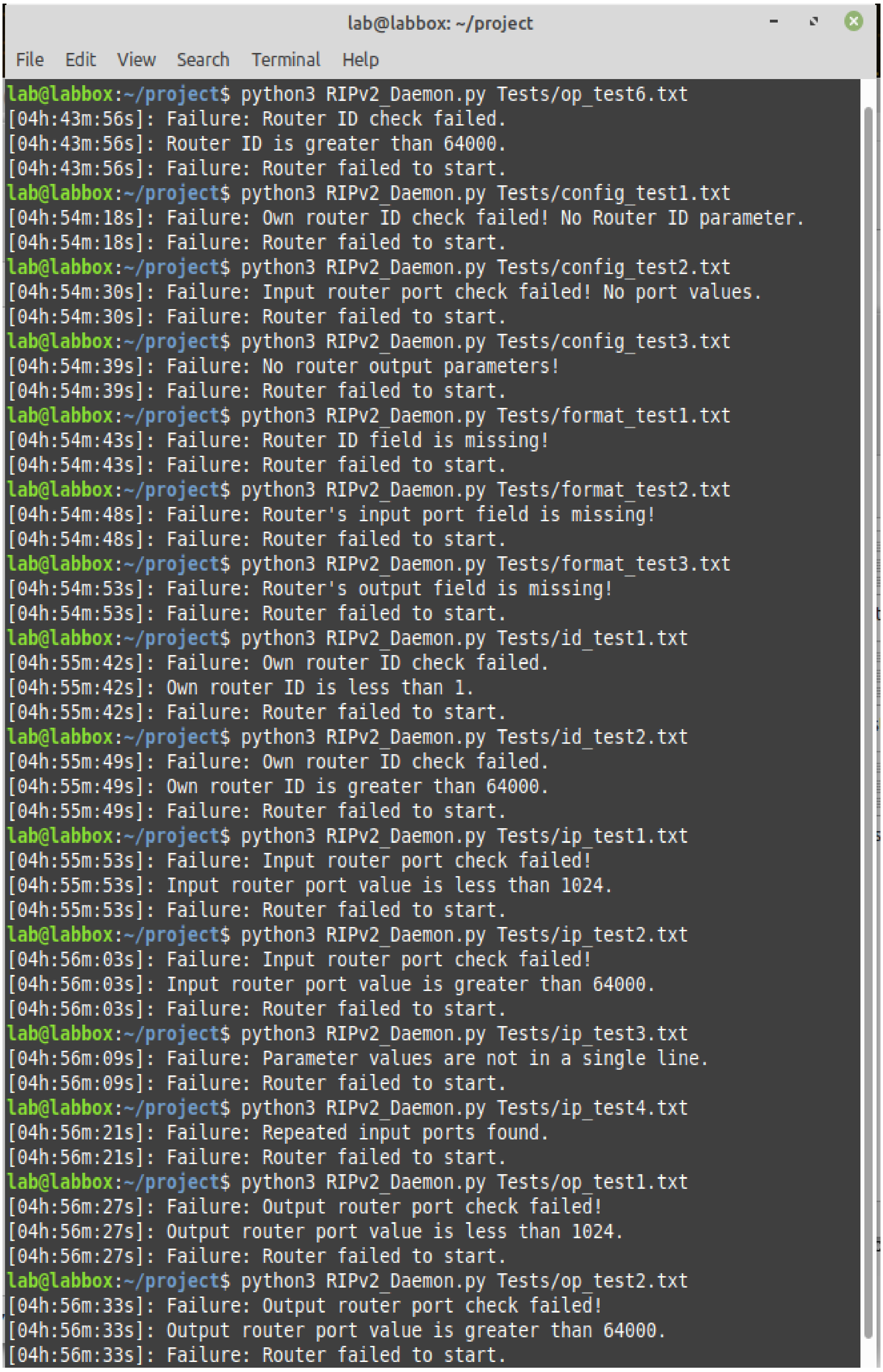
config_test3.txt - tests missing output ports values

These tests therefore show that no matter what type of config files is given to the routing protocol, it will never begin execution until all its conditions are met. Instead, an error message will be shown showing what went wrong with the configuration file. This helps verify that all the router details including ids, metric, port numbers and other parameters line up with the requirements before proceeding. These were the results of the testing carried out on various configuration files used for testing:



```
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/op_test3.txt
[04h:57m:51s]: Failure: Output router cost check failed!
[04h:57m:51s]: Output router cost is less than 1.
[04h:57m:51s]: Failure: Router failed to start.
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/op_test4.txt
[04h:57m:57s]: Failure: Output router cost check failed!
[04h:57m:57s]: Output router cost is greater than 15.
[04h:57m:57s]: Failure: Router failed to start.
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/op_test5.txt
[04h:58m:02s]: Failure: Output router ID check failed.
[04h:58m:02s]: Output router ID is less than 1.
[04h:58m:02s]: Failure: Router failed to start.
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/op_test8.txt
[04h:58m:12s]: Failure: Parameter values are not in a single line.
[04h:58m:12s]: Failure: Router failed to start.
lab@labbox:~/project$
```

More tests on next page:

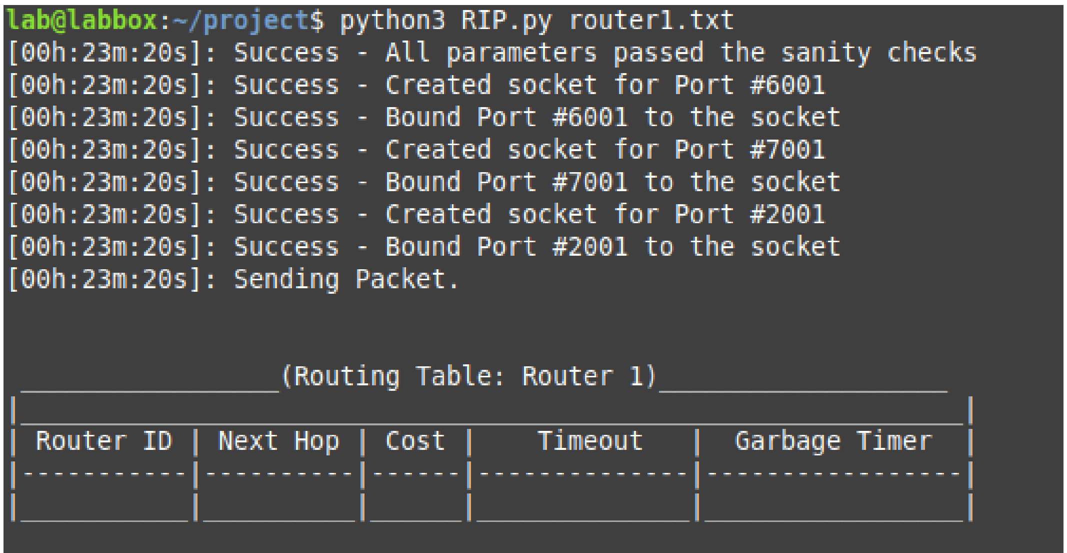


```
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/op_test6.txt
[04h:43m:56s]: Failure: Router ID check failed.
[04h:43m:56s]: Router ID is greater than 64000.
[04h:43m:56s]: Failure: Router failed to start.
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/config_test1.txt
[04h:54m:18s]: Failure: Own router ID check failed! No Router ID parameter.
[04h:54m:18s]: Failure: Router failed to start.
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/config_test2.txt
[04h:54m:30s]: Failure: Input router port check failed! No port values.
[04h:54m:30s]: Failure: Router failed to start.
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/config_test3.txt
[04h:54m:39s]: Failure: No router output parameters!
[04h:54m:39s]: Failure: Router failed to start.
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/format_test1.txt
[04h:54m:43s]: Failure: Router ID field is missing!
[04h:54m:43s]: Failure: Router failed to start.
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/format_test2.txt
[04h:54m:48s]: Failure: Router's input port field is missing!
[04h:54m:48s]: Failure: Router failed to start.
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/format_test3.txt
[04h:54m:53s]: Failure: Router's output field is missing!
[04h:54m:53s]: Failure: Router failed to start.
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/id_test1.txt
[04h:55m:42s]: Failure: Own router ID check failed.
[04h:55m:42s]: Own router ID is less than 1.
[04h:55m:42s]: Failure: Router failed to start.
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/id_test2.txt
[04h:55m:49s]: Failure: Own router ID check failed.
[04h:55m:49s]: Own router ID is greater than 64000.
[04h:55m:49s]: Failure: Router failed to start.
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/ip_test1.txt
[04h:55m:53s]: Failure: Input router port check failed!
[04h:55m:53s]: Input router port value is less than 1024.
[04h:55m:53s]: Failure: Router failed to start.
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/ip_test2.txt
[04h:56m:03s]: Failure: Input router port check failed!
[04h:56m:03s]: Input router port value is greater than 64000.
[04h:56m:03s]: Failure: Router failed to start.
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/ip_test3.txt
[04h:56m:09s]: Failure: Parameter values are not in a single line.
[04h:56m:09s]: Failure: Router failed to start.
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/ip_test4.txt
[04h:56m:21s]: Failure: Repeated input ports found.
[04h:56m:21s]: Failure: Router failed to start.
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/op_test1.txt
[04h:56m:27s]: Failure: Output router port check failed!
[04h:56m:27s]: Output router port value is less than 1024.
[04h:56m:27s]: Failure: Router failed to start.
lab@labbox:~/project$ python3 RIPv2_Daemon.py Tests/op_test2.txt
[04h:56m:33s]: Failure: Output router port check failed!
[04h:56m:33s]: Output router port value is greater than 64000.
[04h:56m:33s]: Failure: Router failed to start.
```

The second part of the testing sections tests the functionality of the routing protocol which includes packet creation, packet correctness, split horizon with poisoned reverse, triggered updates, timers, and routing table correctness. A packet testing function is developed which tests incoming packets for incorrectness and returns an error message with the part of the packet that failed the check. The packet creation functions match the criteria for creating a new RIP packet with all their respective initial fields and values. If a packet is incorrect, it is dropped.

<u>Functionality Tests</u>

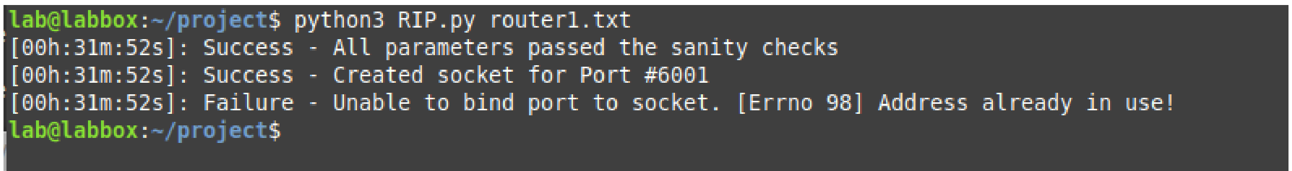
When a router is started, it is supposed to create and bind to all the sockets. Then an empty table should be printed. This is because no other routers are alive yet. The router should try to send packets as well. This is shown below.



```
lab@labbox:~/project$ python3 RIP.py router1.txt
[00h:23m:20s]: Success - All parameters passed the sanity checks
[00h:23m:20s]: Success - Created socket for Port #6001
[00h:23m:20s]: Success - Bound Port #6001 to the socket
[00h:23m:20s]: Success - Created socket for Port #7001
[00h:23m:20s]: Success - Bound Port #7001 to the socket
[00h:23m:20s]: Success - Created socket for Port #2001
[00h:23m:20s]: Success - Bound Port #2001 to the socket
[00h:23m:20s]: Sending Packet.


 _________________(Routing Table: Router 1)___________________
|____________________________________________________________|
| Router ID | Next Hop | Cost |    Timeout    | Garbage Timer |
|-----------|----------|------|---------------|---------------|
|___________|__________|______|_______________|_______________|
```

Normal Router Startup

If a router is already alive, we get an error message that tells us that the port's binding to socket was unsuccessful because the address is already in use. Example:



```
lab@labbox:~/project$ python3 RIP.py router1.txt
[00h:31m:52s]: Success - All parameters passed the sanity checks
[00h:31m:52s]: Success - Created socket for Port #6001
[00h:31m:52s]: Failure - Unable to bind port to socket. [Errno 98] Address already in use!
lab@labbox:~/project$
```

Router Already Alive

Once we start up router 2, we expect router 1 and router 2 to find each other and add each other to their routing table while updating the cost of that path and a respective timeout timer should also be started for both the routers. A link between the two routers is formed like this and the cost of this link should be 1.

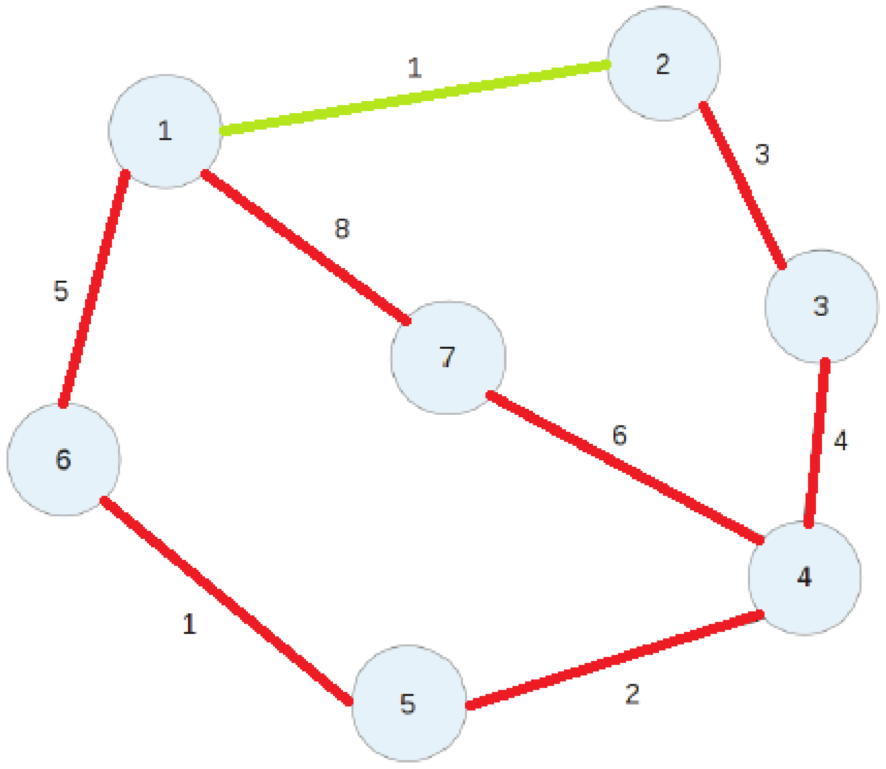
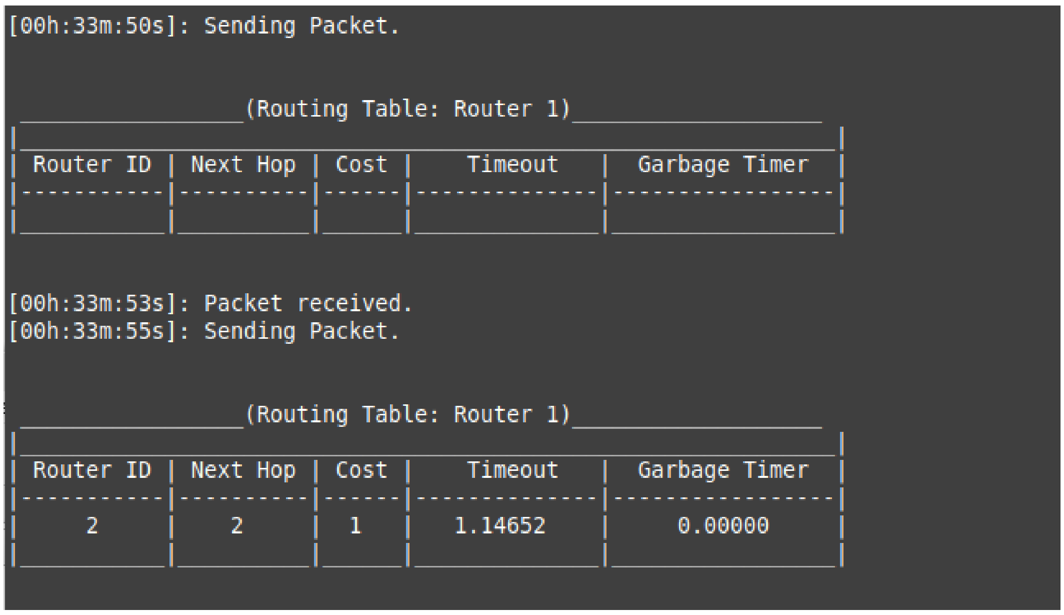
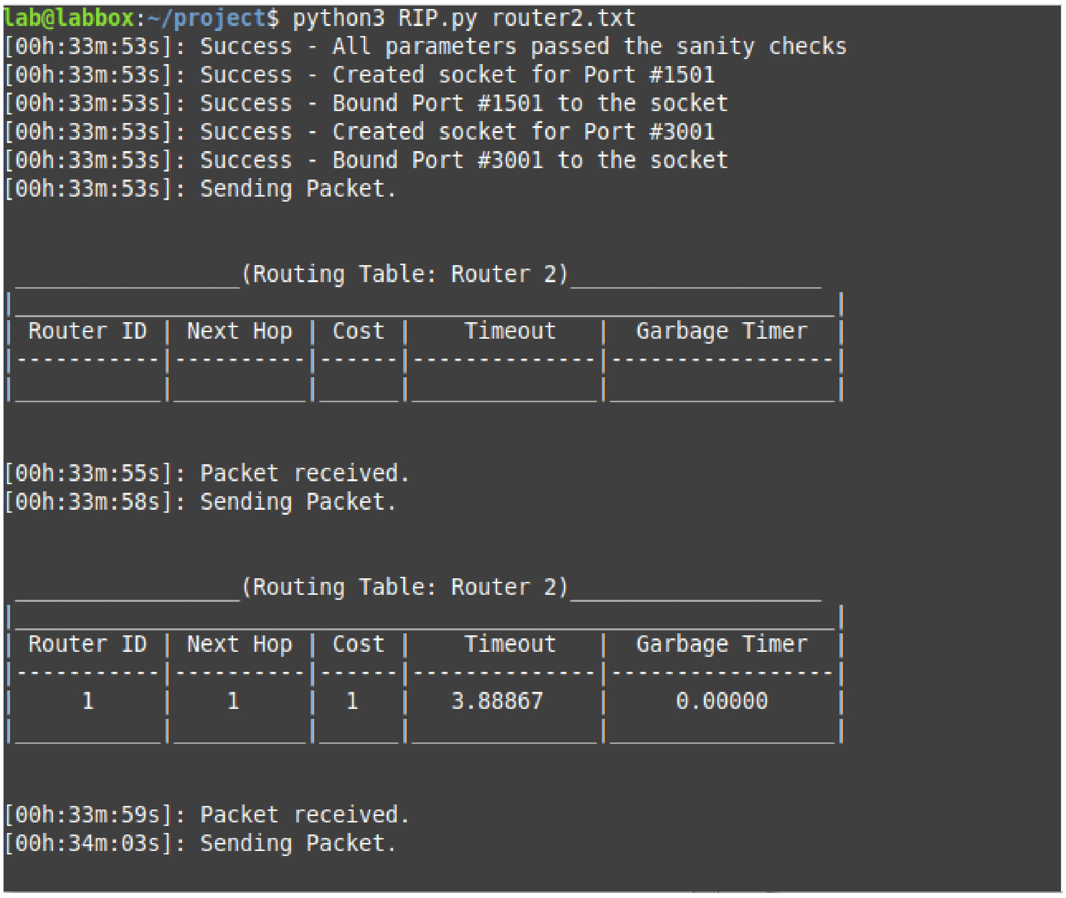


Figure 1: Example network for demonstration

This is exactly what happens when Router 1 finds Router 2. The cost of Router 1 is ‘1’ and the timeout has started. To reach Router 2, the next hop is 2 which is correct. The garbage timer is still 0 as timeout timer hasn’t expired yet. Example:



```
[00h:33m:50s]: Sending Packet.


 _________________(Routing Table: Router 1)___________________
|                                                              |
| Router ID | Next Hop | Cost |     Timeout    |  Garbage Timer |
|-----------|----------|------|----------------|----------------|
|___________|__________|______|________________|________________|


[00h:33m:53s]: Packet received.
[00h:33m:55s]: Sending Packet.


 _________________(Routing Table: Router 1)___________________
|                                                              |
| Router ID | Next Hop | Cost |     Timeout    |  Garbage Timer |
|-----------|----------|------|----------------|----------------|
|     2     |    2     |  1   |   1.14652      |     0.00000    |
|___________|__________|______|________________|________________|
```

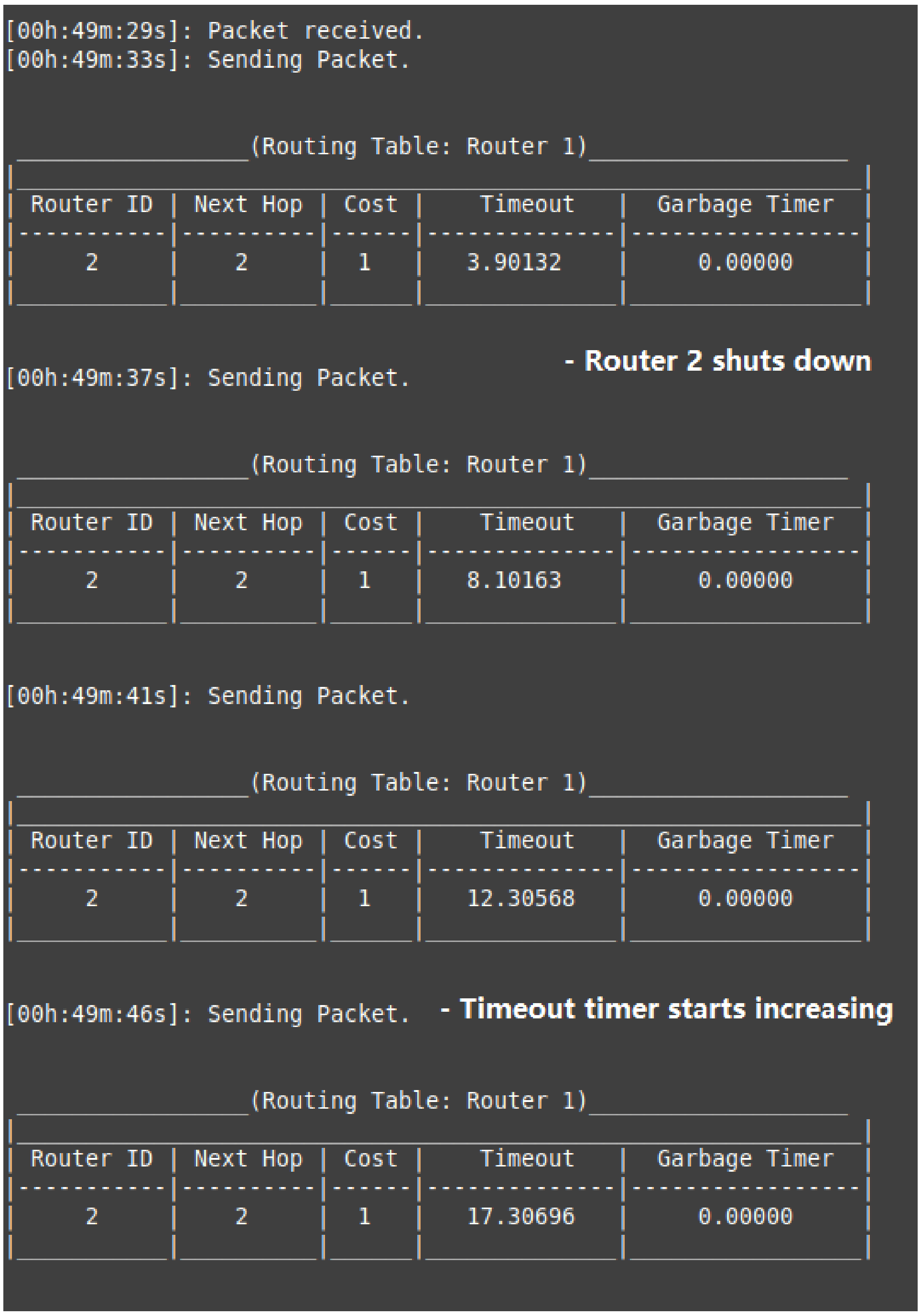


```
lab@labbox:~/project$ python3 RIP.py router2.txt
[00h:33m:53s]: Success - All parameters passed the sanity checks
[00h:33m:53s]: Success - Created socket for Port #1501
[00h:33m:53s]: Success - Bound Port #1501 to the socket
[00h:33m:53s]: Success - Created socket for Port #3001
[00h:33m:53s]: Success - Bound Port #3001 to the socket
[00h:33m:53s]: Sending Packet.


 _________________(Routing Table: Router 2)___________________
|_____________________________________________________________|
| Router ID | Next Hop | Cost |    Timeout    |  Garbage Timer  |
|-----------|----------|------|--------------|-----------------|
|___________|__________|______|______________|_________________|


[00h:33m:55s]: Packet received.
[00h:33m:58s]: Sending Packet.


 _________________(Routing Table: Router 2)___________________
|_____________________________________________________________|
| Router ID | Next Hop | Cost |    Timeout    |  Garbage Timer  |
|-----------|----------|------|--------------|-----------------|
|     1     |    1     |  1   |   3.88867    |     0.00000     |
|___________|__________|______|______________|_________________|


[00h:33m:59s]: Packet received.
[00h:34m:03s]: Sending Packet.
```

As seen in the above image, Router 2 also reacts in the same way and starts its own timeout timer. The cost and next hop are ‘1’ and 1 respectively. This is correct and both routers have established an adjacency between them. They are also both sending and receiving packets to and from each other.

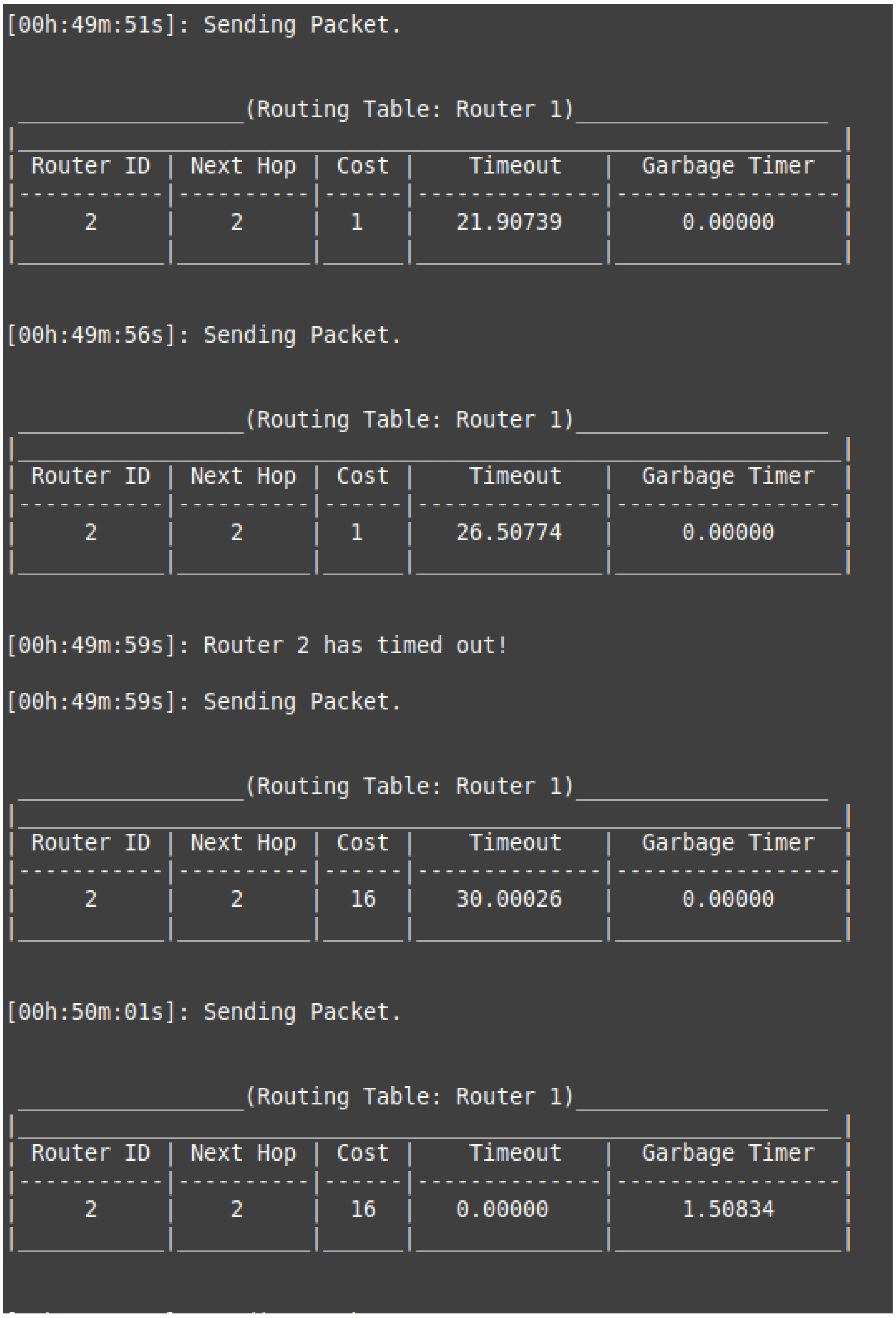
<u>**Split Horizon with Poisoned Reverse**</u>

Next, we shut down Router 2. The expected results are change in the metric from Router 1 to Router 2 which become 16. This is because 16 represents that Router 2 is now unreachable. In the background, the timeout timer is also supposed to run out and a garbage timer is then expected to start where after the completion of the garbage timer, Router 2 is deleted from the routing table of Router 1. This is shown step by step below:



```
[00h:49m:29s]: Packet received.
[00h:49m:33s]: Sending Packet.


 _________________(Routing Table: Router 1)___________________
|_____________________________________________________________|
| Router ID | Next Hop | Cost |    Timeout    |  Garbage Timer |
|-----------|----------|------|---------------|----------------|
|     2     |    2     |  1   |    3.90132    |     0.00000    |
|___________|__________|______|_______________|________________|

                                        - Router 2 shuts down
[00h:49m:37s]: Sending Packet.


 _________________(Routing Table: Router 1)___________________
|_____________________________________________________________|
| Router ID | Next Hop | Cost |    Timeout    |  Garbage Timer |
|-----------|----------|------|---------------|----------------|
|     2     |    2     |  1   |    8.10163    |     0.00000    |
|___________|__________|______|_______________|________________|


[00h:49m:41s]: Sending Packet.


 _________________(Routing Table: Router 1)___________________
|_____________________________________________________________|
| Router ID | Next Hop | Cost |    Timeout    |  Garbage Timer |
|-----------|----------|------|---------------|----------------|
|     2     |    2     |  1   |    12.30568   |     0.00000    |
|___________|__________|______|_______________|________________|

[00h:49m:46s]: Sending Packet.   - Timeout timer starts increasing


 _________________(Routing Table: Router 1)___________________
|_____________________________________________________________|
| Router ID | Next Hop | Cost |    Timeout    |  Garbage Timer |
|-----------|----------|------|---------------|----------------|
|     2     |    2     |  1   |    17.30696   |     0.00000    |
|___________|__________|______|_______________|________________|
```

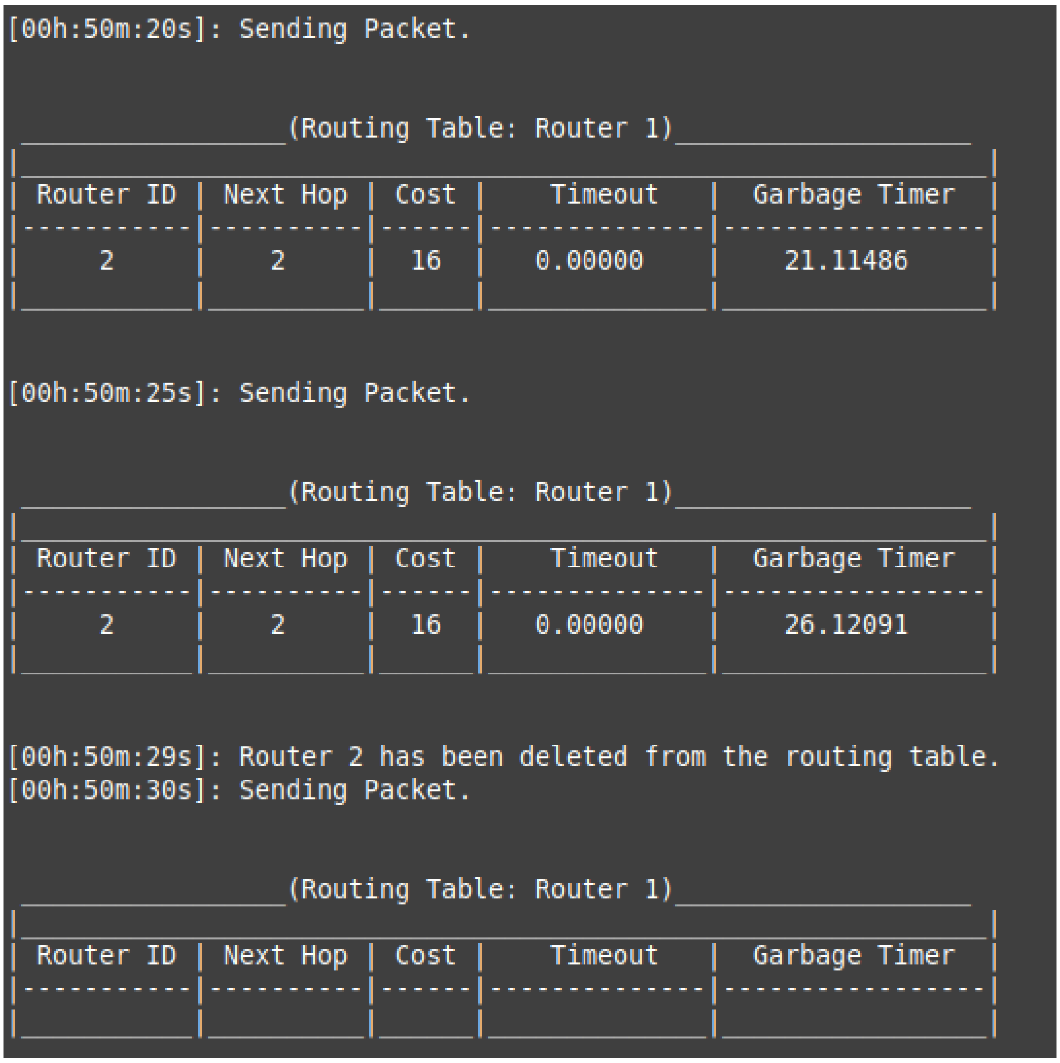
After Router 2 shuts down, Router 2’s timeout timer starts increasing. Router 1 now again keeps sending a packet but doesn’t receive any from Router 2.



```
[00h:49m:51s]: Sending Packet.


 _________________(Routing Table: Router 1)___________________
|______________________________________________________________|
| Router ID | Next Hop | Cost |    Timeout    |  Garbage Timer  |
|-----------|----------|------|--------------|-----------------|
|     2     |    2     |  1   |   21.90739    |     0.00000     |
|___________|__________|______|______________|_________________|


[00h:49m:56s]: Sending Packet.


 _________________(Routing Table: Router 1)___________________
|______________________________________________________________|
| Router ID | Next Hop | Cost |    Timeout    |  Garbage Timer  |
|-----------|----------|------|--------------|-----------------|
|     2     |    2     |  1   |   26.50774    |     0.00000     |
|___________|__________|______|______________|_________________|


[00h:49m:59s]: Router 2 has timed out!

[00h:49m:59s]: Sending Packet.


 _________________(Routing Table: Router 1)___________________
|______________________________________________________________|
| Router ID | Next Hop | Cost |    Timeout    |  Garbage Timer  |
|-----------|----------|------|--------------|-----------------|
|     2     |    2     |  16  |   30.00026    |     0.00000     |
|___________|__________|______|______________|_________________|


[00h:50m:01s]: Sending Packet.


 _________________(Routing Table: Router 1)___________________
|______________________________________________________________|
| Router ID | Next Hop | Cost |    Timeout    |  Garbage Timer  |
|-----------|----------|------|--------------|-----------------|
|     2     |    2     |  16  |   0.00000     |     1.50834     |
|___________|__________|______|______________|_________________|
```

After not receiving any response from Router 2 for the set amount of timeout time, Router 2 will be timed out and its timeout timer will reset to 0. The cost to Router 2 will change to 16 making it unreachable and the garbage timer for the router will start.

After the garbage timer runs out, it is expected that Router 1 removes Router 2 from its routing table. Router 1 table becomes empty again as it knows that Router 2 is dead. This is precisely what goes down as seen in this example snippet:



```
[00h:50m:20s]: Sending Packet.


 _________________(Routing Table: Router 1)___________________
|____________________________________________________________|
| Router ID | Next Hop | Cost |    Timeout    |  Garbage Timer |
|-----------|----------|------|---------------|----------------|
|     2     |    2     |  16  |   0.00000     |    21.11486    |
|___________|__________|______|_______________|________________|


[00h:50m:25s]: Sending Packet.


 _________________(Routing Table: Router 1)___________________
|____________________________________________________________|
| Router ID | Next Hop | Cost |    Timeout    |  Garbage Timer |
|-----------|----------|------|---------------|----------------|
|     2     |    2     |  16  |   0.00000     |    26.12091    |
|___________|__________|______|_______________|________________|


[00h:50m:29s]: Router 2 has been deleted from the routing table.
[00h:50m:30s]: Sending Packet.


 _________________(Routing Table: Router 1)___________________
|____________________________________________________________|
| Router ID | Next Hop | Cost |    Timeout    |  Garbage Timer |
|-----------|----------|------|---------------|----------------|
|___________|__________|______|_______________|________________|
```

<u>Convergence test for all the routers.</u>

Now, we start up all the routers to see if the routes converge as expected. The routing table will then show the shortest path to reach each router. We will take Router 1 as an example for this scenario.

The expected shortest path from Router 1 to:

Router 2 – (Next Hop = 2, Cost = 1)

Router 3 – (Next Hop = 2, Cost = 4)

Router 4 – (Next Hop = 2, Cost = 8)

Router 5 – (Next Hop = 2, Cost = 10)

Router 6 – (Next Hop = 6, Cost = 5)

Router 7 – (Next Hope = 7, Cost = 8)

Let’s start up Router 1, 2, 3, 4 and 7. This is what the connected map should look like.

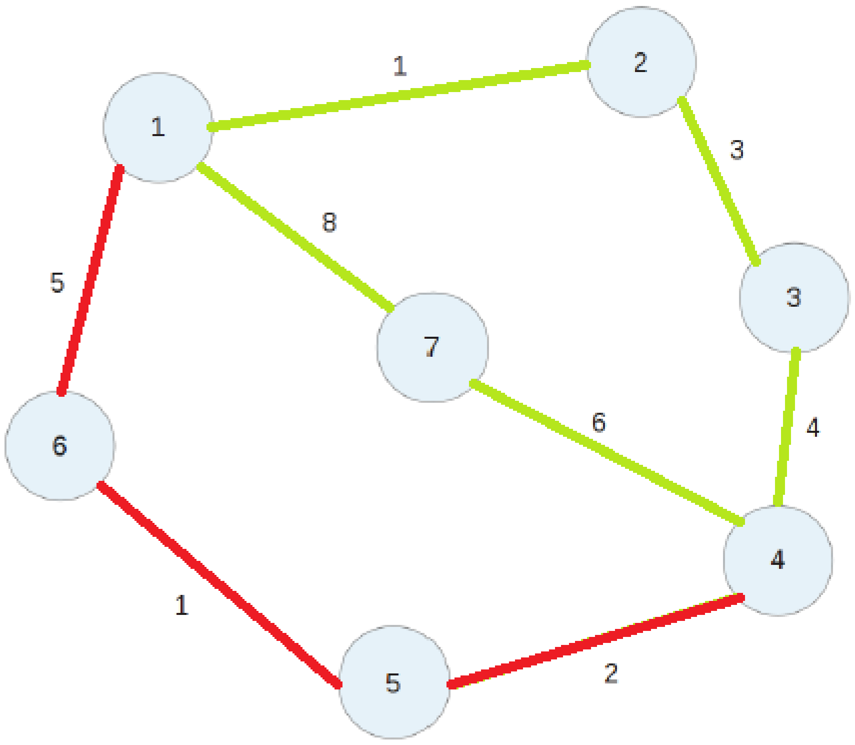
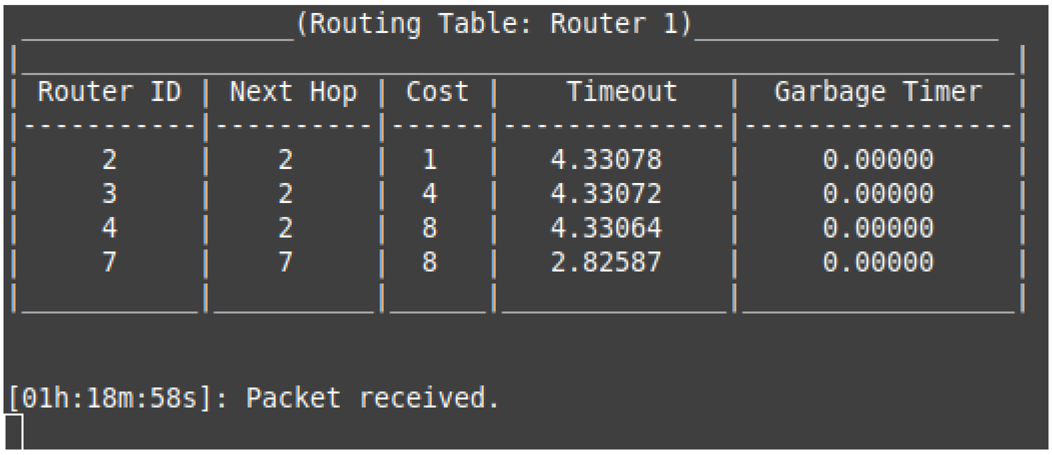


Figure 1: Example network for demonstration

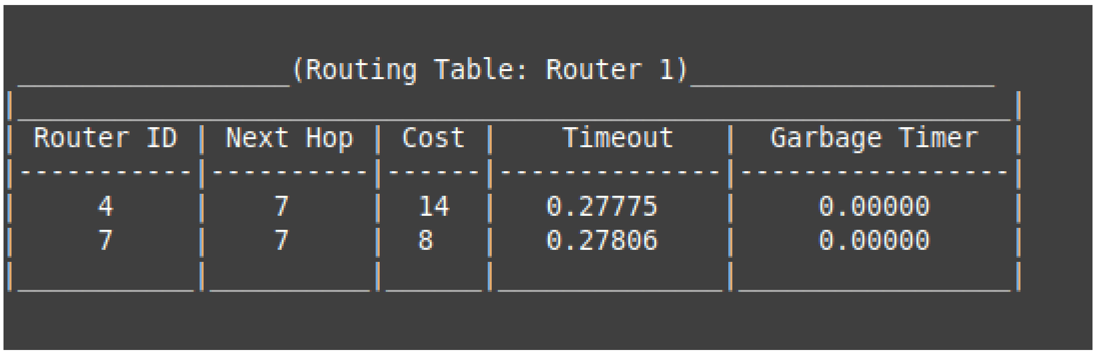
,

Once the following routes are developed, we see the following routing table for Router 1.



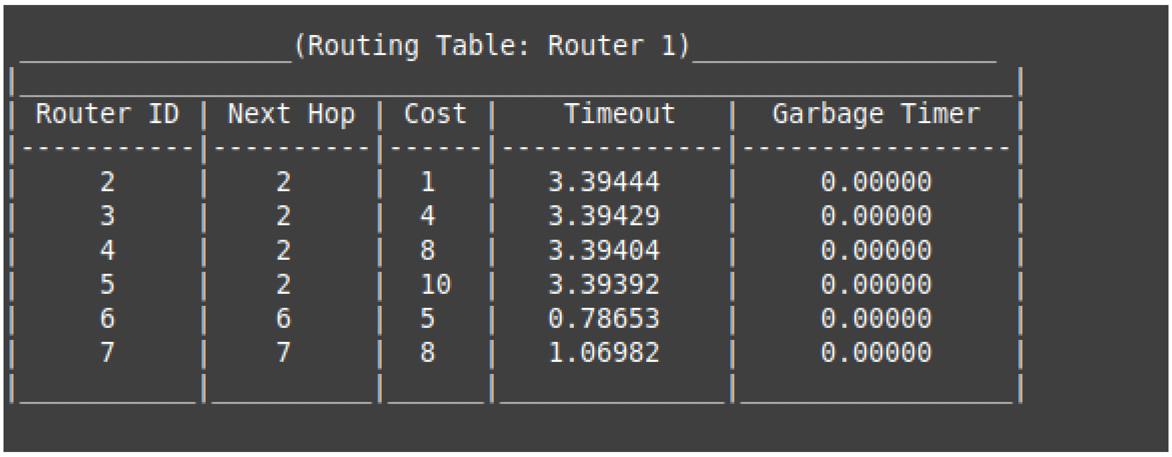
```
 _________________(Routing Table: Router 1)___________________
|                                                             |
| Router ID | Next Hop | Cost |    Timeout    |  Garbage Timer |
|-----------|----------|------|---------------|----------------|
|     2     |    2     |  1   |   4.33078     |    0.00000     |
|     3     |    2     |  4   |   4.33072     |    0.00000     |
|     4     |    2     |  8   |   4.33064     |    0.00000     |
|     7     |    7     |  8   |   2.82587     |    0.00000     |
|___________|__________|______|_______________|________________|


[01h:18m:58s]: Packet received.
```

Here we see another very important thing. Router 1 chooses to go to Router 4 through Router 2 and 3 rather than Router 7. This is because through Router 2 and 3, a shorter path is offered. This means that the route calculation is working well and only the correct metric and next hops are being identified. If we turn off Router 2, the link from Router 1 to Router 4 through Router 2 will be broken. The link from Router 1 to Router 3 will also be broken This will force Router 1 to find a new shortest path to Router 4. This means it will then go through Router 7. Our results show this once Router 2 goes down:



```
 _________________(Routing Table: Router 1)___________________
|                                                              |
| Router ID | Next Hop | Cost |    Timeout    |  Garbage Timer  |
|-----------|----------|------|---------------|-----------------|
|     4     |    7     |  14  |   0.27775     |     0.00000     |
|     7     |    7     |  8   |   0.27806     |     0.00000     |
|___________|__________|______|_______________|_________________|
```

Here we see that Router 1 no longer has a route to Router 2 or Router 3. Router 2 got deleted so of course Router 1 can’t find that router. As for Router 3, there is another path available which is through Router 7 and Router 4. However, the total cost of this path is 18 and RIP V2 only allows a maximum of 15 hops. Therefore, Router 3 also gets deleted. As for a path to Router 4, Router 1 now finds a new path which is through Router 7. Therefore, it updates its next hop to 7 and changes its metric to 14 which is correct. This is a good example to show both triggered updates and poisoned reverse. The following is the routing table for Router 1 after all other Routers are on and the table has converged.



```
 _________________(Routing Table: Router 1)___________________
|                                                              |
| Router ID | Next Hop | Cost |    Timeout    |  Garbage Timer  |
|-----------|----------|------|---------------|-----------------|
|     2     |    2     |  1   |   3.39444     |     0.00000     |
|     3     |    2     |  4   |   3.39429     |     0.00000     |
|     4     |    2     |  8   |   3.39404     |     0.00000     |
|     5     |    2     |  10  |   3.39392     |     0.00000     |
|     6     |    6     |  5   |   0.78653     |     0.00000     |
|     7     |    7     |  8   |   1.06982     |     0.00000     |
|___________|__________|______|_______________|_________________|
```

If we look back, it matches perfectly with our predicted outcome which was:

“The expected shortest path from Router 1 to:

Router 2 – (Next Hop = 2, Cost = 1)

Router 3 – (Next Hop = 2, Cost = 4)

Router 4 – (Next Hop = 2, Cost = 8)

Router 5 – (Next Hop = 2, Cost = 10)

Router 6 – (Next Hop = 6, Cost = 5)

Router 7 – (Next Hope = 7, Cost = 8)"

This is what the routing map looks like after all the Routers are up.

5 Deliverables

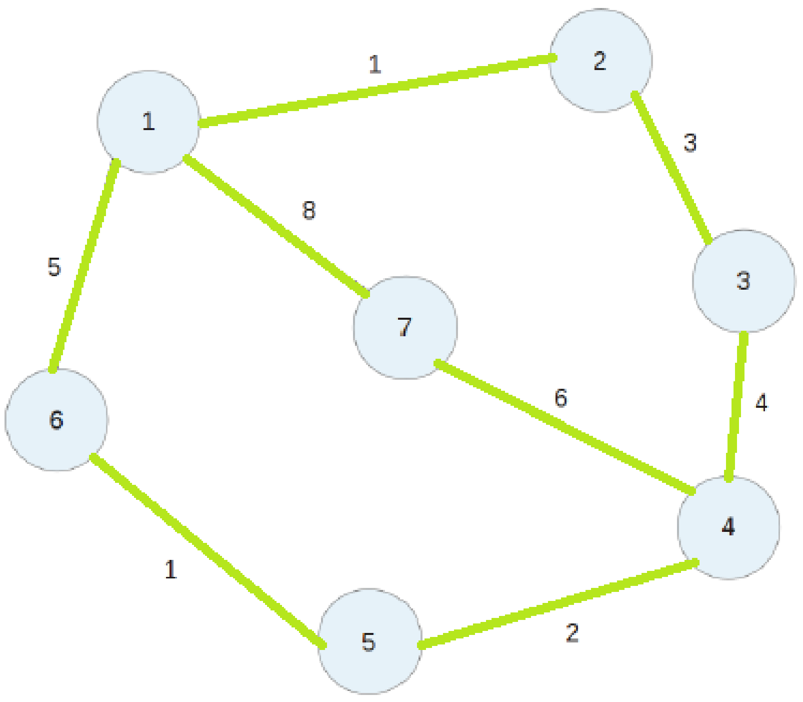
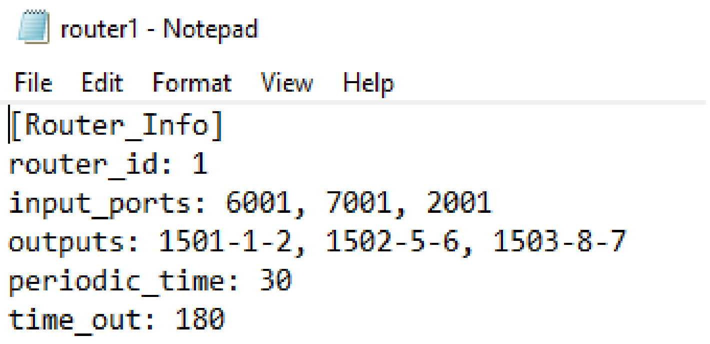


Figure 1: Example network for demonstration

Here is an example of the configuration file of Router 1 used in the project.



```
[Router_Info]
router_id: 1
input_ports: 6001, 7001, 2001
outputs: 1501-1-2, 1502-5-6, 1503-8-7
periodic_time: 30
time_out: 180
```

This shows that the RIP Routing Protocol is working well. The design and implementation of the Routing Protocol was successful. All the test cases are passing. Poisoned reverse is working by changing the metric to 16. New routes are also being calculated. Routes with a metric of 16 or more are being ignored. Timeout and garbage timers are working correctly when it comes to timing out and deleting a router. The routing table converges correctly for all Routers. Configuration file tests are handling any errors inside the configuration file before processing whereas the routing tests have all passed.

Words: 2207

```python
'''
                COSC364 (RIPv2 Routing Protocol)
           Authors: Haider Saeed (msa280), Drogo Shi (msh217)
                    Date: 07/03/2022
                 Filename: RIPv2_Daemon.py

Program Definition: Configures RIP routing protocol based on the specifications
                outlined in RIP Version 2 (RFC2453). (Section 4 not included)
'''


import sys
import configparser     # ConfigParser class which implements a basic configuration language
import time
import socket
import select
import threading    # use timer here so sys load will not affect the time
import random
from RIPv2_Router import*
from RIPv2_ConfigureFile import*



LOCAL_HOST = '127.0.0.1'



def start_daemon():
    """Starts up the router."""
    # Gets the name of file from the command line. Example (python3 RIPV2_Daemon.py router1.txt)
    filename = sys.argv[1]

    file = RIPv2_ConfigureFile(filename)
    # Configures and creates and binds to sockets
    file.read_and_process_file()

    router_id_self = file.router_info['router_id']

    #let first one input socket to send out packet
    #send_socket_port = list(file.router_info['inputs'].keys())[0]
    #file.router_info['inputs'].get(send_socket_port)

    sending_socket = list(file.router_info['inputs'].values())[0]

    # Create a new router
    router = RIPv2_Router(router_id_self, file.neighbor, sending_socket)

    # Send packet to neighbour
    router.periodically_send_packets()

    while True:

        all_input_sockets = list(file.router_info['inputs'].values())

        socket_list, w_list, e_list = select.select(all_input_sockets, [], [], 5)

        for socket in socket_list:
            packet = socket.recvfrom(1024)[0]
            router.receive_packet(packet)




def main():
    """ This is the main function which runs the routing protocol."""
    start_daemon()
```

```
main()
```

```
'''
                    COSC364 (RIPv2 Routing Protocol)
            Authors: Haider Saeed (msa280), Drogo Shi (msh217)
                           Date: 07/03/2022
                   Filename: RIPv2_ConfigureFile.py

Program Definition: Reads the configuration files of and router and prints out
                   any error messages. Otherwise, it creates and binds to
                   sockets.
'''

import sys
import configparser  # ConfigParser class which implements a basic configuration language
import time
import socket
import select
import threading  # Used timer here so system load will not affect the time.
import random
from RIPv2_Router import*



LOCAL_HOST = '127.0.0.1'


class RIPv2_ConfigureFile():
    ''' Configure class which reads and processes the configuration file of a router
    using filename. It then tries to create and bind to the sockets. '''


    def __init__(self, config_file):
        """ Initiates the configuration file. """

        self.config_file = config_file
        self.router_info = {}
        self.neighbor = {}



    def router_id_check(self, router_id, port_type):
        """ Checks if the router_id is within the supported parameter range. """

        if (router_id == ''):
            exit_msg(f'Failure: {port_type} router ID check failed! No Router ID parameter.')
        elif (1 <= int(router_id) <= 64000):
            return True
        else:
            print_msg(f'Failure: {port_type} router ID check failed.')
            if (int(router_id) < 1):
                exit_msg(f'{port_type} router ID is less than 1.')
            else:
                exit_msg(f'{port_type} router ID is greater than 64000.')





    def cost_check(self, cost):
        """ Checks if the cost is within the supported parameter range. """

        if (cost == ''):
            exit_msg(f'Failure: Output router cost check failed! No cost values.')
        elif (1 <= int(cost) <= 15):
            return True
        else:
            print_msg('Failure: Output router cost check failed! ')
        if (int(cost) < 1):
            exit_msg('Output router cost is less than 1.')
        else:
            exit_msg('Output router cost is greater than 15.')




def port_check(self, port, port_type):
    """ Checks if the port is within the supported parameter range. """

    if (port == ''):
        exit_msg(f'Failure: {port_type} router port check failed! No port values.')
    elif (1024 <= int(port) <= 64000):
        return True
    else:
        print_msg(f'Failure: {port_type} router port check failed!')
        if (int(port) < 1024):
            exit_msg(f'{port_type} router port value is less than 1024.')
        else:
            exit_msg(f'{port_type} router port value is greater than 64000.')




def get_router_id(self, config):
    """ Gets the router id from the config file of the router after
    performing sanity checks. """

    try:
        router_id = config['Router_Info']['router_id']
    except:
        exit_msg('Failure: Router ID field is missing!')

    if (self.router_id_check(router_id, 'Own')):
        self.router_info['router_id'] = int(router_id)




def get_inputs(self, config):
    """ Gets the router's input ports then check if any of the input ports
    exist in output_ports."""

    input_ports = []

    try:
        router_inputs = config['Router_Info']['input_ports'].split(', ')
    except KeyError:
        exit_msg("Failure: Router's input port field is missing!")

    for input_port in router_inputs:
        self.port_check(input_port, 'Input')

        if (int(input_port) not in input_ports):
            input_ports.append(int(input_port))
        else:
            exit_msg("Failure: Repeated input ports found.")

    return input_ports
```

```python
def get_outputs(self, config):
    """ Gets the router's output values."""
    try:
        router_outputs = config['Router_Info']['outputs'].split(', ')
    except:
        exit_msg("Failure: Router's output field is missing!")


    if (router_outputs == ['']):
        exit_msg('Failure: No router output parameters!')

    outputs = []

    # Converting (5001-2-3) to (5001, 2, 3)
    for output in router_outputs:
        params = output.split('-')
        outputs.append(params)

    return outputs




def router_output_format_check(self, output):
    """ Checks if the router outputs exist and return the
    parameter values. """

    port = None
    cost = None
    router_id = None

    try:
        port = output[0]
    except:
        exit_msg('Failure: Router output has no port value.')

    try:
        cost = output[1]
    except:
        exit_msg('Failure: Router output has no cost value.')

    try:
        router_id = output[2]
    except:
        exit_msg('Failure: Router output has no router id value.')

    return (port, cost, router_id)




def read_outputs(self, config):
    """ Gets the output ports of the router. """

    outputs = self.get_outputs(config)
    output_ports = []
    self.router_info['outputs'] = []

    for output in outputs:

        (port, cost, router_id) = self.router_output_format_check(output)

        id_check_passed = self.router_id_check(router_id, 'Output')
        cost_check_passed = self.cost_check(cost)
        port_check_passed = self.port_check(port, 'Output')
        port_not_repeated = int(port) not in output_ports
```

```
            if (id_check_passed and cost_check_passed and port_check_passed and port_not_repeated):
                output_ports.append(port)
                output_format = {'router_id': int(router_id), 'port': int(port), 'cost': int(cost)}
                self.router_info['outputs'].append(output_format)
                self.neighbor[int(router_id)] = [int(cost), int(port)]
            else:
                exit_msg("Failure: Router output check failed! Wrong output values.")


        input_ports = self.get_inputs(config)

        # Checks if any port number from input port exists in output ports
        for port in input_ports:
            if (int(port) in output_ports):
                exit_msg('Failure: No output port numbers can be in input ports.')

        print_msg('Success: Router output parameter checks have passed.')




    def create_and_bind_socket(self, port):
        """Creates socket for the given port and attempts to bind to it. """

        # Trying to create a UDP socket for each port.
        try:
            self.router_info['inputs'][port] = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            print_msg('Success - Created socket for Port #' + str(port))
        except socket.error as message:
            exit_msg('Failure - Unable to create socket. ' + str(message))


        # Trying to bind each port to the socket. #try to move this to router class
        try:
            self.router_info['inputs'][port].bind((LOCAL_HOST, port))
            print_msg('Success - Bound Port #' + str(port) + ' to the socket')
        except socket.error as msg:
            exit_msg('Failure - Unable to bind port to socket. ' + str(msg))

        print_msg("Success: Router is bound to all sockets.")




    def read_inputs(self, config):
        """ Gets the input ports for the router and for each port opens a UDP
        socket and attempts to bind to it. """

        input_ports = self.get_inputs(config)

        self.router_info['inputs'] = {}
        for port in input_ports:
            self.create_and_bind_socket(port)




    def read_and_process_file(self):
        """ Starts processing and reading the configuration file of the router
        while performing sanity checks. """

        try:
            config = configparser.ConfigParser()
            config.read(self.config_file)
        except configparser.ParsingError:
            exit_msg('Failure: Parameter values are not in a single line.')

        self.get_router_id(config)
        self.read_outputs(config)
        self.read_inputs(config)

        print_msg('Success - All parameters passed the sanity checks')




def print_msg(message):
    """ Prints the message and the time at which it was sent. """
    current_time = time.strftime("%Hh:%Mm:%Ss")
    print("[" + current_time + "]: " + message)


def exit_msg(message):
    """ Prints the message if something goes wrong with starting the router. """
    print_msg(message)
    print_msg('Failure: Router failed to start.')
    sys.exit()
```

```
'''
                COSC364 (RIPv2 Routing Protocol)
            Authors: Haider Saeed (msa280), Drogo Shi (msh217)
                        Date: 07/03/2022
                    Filename: RIPv2_Router.py

Program Definition: Contains the functionality of a router which includes
                the creation and checking of packets. It also includes
                route and metric calculation, timers, split hoirzon with
                poisoned reverse. This class handles the functions of a
                RIP Version 2 Router.
'''


import sys
import configparser    # ConfigParser class which implements a basic configuration language
import time
import socket
import select
import threading    # use timer here so sys load will not affect the time
import random
from threading import Timer



LOCAL_HOST = '127.0.0.1'



class RIPv2_Router():
    """ This is the Router class for the RIPv2 Protocol."""


    def __init__(self, router_id, neighbours, sending_socket):
        """ Initialises the router. """
        self.table = {}
        self.self_id = router_id
        self.neighbours = neighbours
        self.sending_socket = sending_socket
        self.timeout = 30
        self.garbage_time = 30
        self.timeout_timer_dict = {} #diction for record every entry's start time of timeout timer
        self.garbage_timer_dict = {} #diction for record every entry's start time of garbage timer




    def init_timer(self, dst_id):
        """ Initializes the timer. """
        (flag, timeout_timer, garbage_timer) = (self.table[dst_id][2], self.table[dst_id][3], self.table[dst_id][4])
        self.table[dst_id][2] = False
        self.table[dst_id][3].cancel()
        self.table[dst_id][4].cancel()
        self.table[dst_id][3] = self.start_timeout(dst_id)
        self.table[dst_id][4] = threading.Timer(self.garbage_time, self.delete_router, (dst_id,))
        if self.garbage_timer_dict.get(dst_id):
            self.garbage_timer_dict.pop(dst_id) #so no garbage timer started




    def start_timeout(self, router_id):
        """Starts a timeout timer for an entry in the routing table"""
        # Remember when deleting from the table, cancel this timer first
        threading_time = threading.Timer(self.timeout, self.end_timeout,(router_id,)) #for every 30 will call not_reciving func
        threading_time.start()
        self.timeout_timer_dict[router_id] = time.time()
        return threading_time




    def end_timeout(self, router_id):
        """ Updates the routing table after a router has timed out. """
        # After timeout, the router changes the metric of that entry and triggers updates

        self.give_update("Router {} has timed out!\n".format(router_id))
        entry = self.table.get(router_id)
        if entry[2] == False:
            entry[0] = 16  # Change metric to 16
            entry[2] = True # Change Flag
            self.send_packet() # Triggers updates when metric first becomes 16
        self.start_garbage_timer(router_id) # Start a garbage timer
        entry[3].cancel() # Closes the timeout timer




    def start_garbage_timer(self, router_id):
        '''Starts a garbage timer. '''
        self.table.get(router_id)[4].start()
        self.garbage_timer_dict[router_id] = time.time()




    def delete_router(self, router_id):
        '''Upon completion of garbage timer, it pops the router from the table.'''
        timeout_timer, garbage_timer = (self.table[router_id][3], self.table[router_id][4])
        #cancel timer for the timeout and garbage
        timeout_timer.cancel()
        garbage_timer.cancel()
        popped_router = self.table.pop(router_id, 0)
        self.give_update("Router {} has been deleted from the routing table.".format(router_id))




    def create_packet(self, send_to_neighbour_id):
        """ Creates a RIP Packet from the routing table to send to one of the neighbours. """

        packet = bytearray()

        # RIP Common Header
        packet.append(2) # Command
        packet.append(2) # Version
        packet.append(self.self_id &0xFF) # Router id
        packet.append((self.self_id >> 8) &0xFF) #R outeprint_routing_table ID

        # RIP Entries
        if (len(self.table) == 0): #If no entry in table, only header will be sent to the table
            return packet

        for dst_router_id, values in self.table.items():
            if dst_router_id == send_to_neighbour_id:
                continue

            # Address family identifier
            packet.append(0)
            packet.append(0)

            # Route tag
            packet.append(0)
            packet.append(0)

            # Router ID/IP
            packet.append(0)
            packet.append(0)
            packet.append(dst_router_id &0xFF)
            packet.append((dst_router_id >> 8) &0xFF)
            # Subnet mask
            packet.append(0)
            packet.append(0)
            packet.append(0)
            packet.append(0)
            # Next Hop
            packet.append(0)
            packet.append(0)
            packet.append(0)
            packet.append(0)


            metric = values[0]
            next_hop = values[1]

            # Implementing split horzion with posioned reverse. Updating metric accordingly.
            # Metric
            if ((send_to_neighbour_id != dst_router_id) and (next_hop == send_to_neighbour_id)):
                packet.append(0)
                packet.append(0)
                packet.append(0)
                packet.append(16 &0xFF)
            else:
                packet.append(0)
                packet.append(0)
                packet.append(0)
                packet.append(metric &0xFF)

        return packet




    def check_rip_header(self, header):
        """ Checks if the RIP header is correct. If it is correct, it returns
        the id of the router it received from."""

        command = int(header[0])
        version = int(header[1])
        received_from_router_id = (int(header[2] & 0xFF)) + int((header[3] << 8))

        if ((command != 2 or version != 2) or (received_from_router_id < 1 or received_from_router_id > 6400)) :
            print_msg("Wrong Packet Received!\n")
            return False
        else:
            return received_from_router_id




    def check_rip_entry(self, entry):
        """  Checks the incoming packet for correctness and returns True if
        check passed. """

        address_family_id = (int(entry[0]), int(entry[1]))
        route_tag = (int(entry[2]), int(entry[3]))
        router_id = (int(entry[4]), int(entry[5]))
        subnet_mask = (int(entry[8]), int(entry[9]), int(entry[10]), int(entry[11]))
```

```python
        next_hop = (int(entry[12]), int(entry[13]), int(entry[14]), int(entry[15]))
        metric_zero_part = (int(entry[16]), int(entry[17]), int(entry[18]))
        metric = (int(entry[19]))

        entry_check_passed = False

        # Check address family identifier
        if (address_family_id != (0,0)):
            print_msg("\nIncorrect Packet. Wrong address family identifier value.")
        # Check route tag
        elif (route_tag != (0,0)):
            print_msg("\nIncorrect Packet. Wrong route tag value.")
        # Check router id
        elif (router_id != (0,0)):
            print_msg("\nIncorrect Packet. Wrong router id value.")
        # Check subnet mask
        elif (subnet_mask != (0,0,0,0)):
            print_msg("\nIncorrect Packet. Wrong subnet mask value.")
        # Check next_hop
        elif (next_hop != (0,0,0,0)):
            print_msg("\nIncorrect Packet. Wrong next hop value.")
        # Check metric
        elif (metric_zero_part != (0,0,0)):
            print_msg("\nIncorrect Packet. Wrong metric value.")
        elif (metric > 16 or metric < 0):
            print_msg("\nIncorrect Packet. Metric value out of range.")
        else:
            entry_check_passed = True

        return entry_check_passed




    def receive_packet(self, packet):
        """ Process a received packet. """
        self.give_update("Packet received.")

        # Check header and entries
        len_packet = len(packet)
        neb_id = self.check_rip_header(packet)

        # If header check fails, then the packet is dropped.
        if (neb_id == False):
            print_msg("Incorrect packet header. Packet dropped!")
            return

        # Packet checking
        for entry_start_index in range(4, len_packet, 20): #every entry has 20byts
            if not (self.check_rip_entry(packet[entry_start_index:entry_start_index+20])):
                index_entry = (len_packet - 4) // 20
                print_msg(f"Wrong entry for index_entry: {index_entry}")
                print_msg("Dropped the packet!")
                return

        # If table doesn't have neighbour
        if (self.table.get(neb_id) == None):
            cost,_ = self.neighbours.get(neb_id)
            self.table[neb_id] = [cost, neb_id, False, self.start_timeout(neb_id), threading.Timer(self.garbage_time,
self.delete_router, (neb_id,))]
        # Else, reinitialize the timer and cost
        else:
            cost,_ = self.neighbours.get(neb_id)
            self.init_timer(neb_id)
            self.table[neb_id][0] = cost
        # Stops furthur processing if only the header is received.
        if (len_packet == 4):
            return
        # End of neighbour processing

        # Start for non-neighbour processing
        for entry_start_index in range(4,len_packet,20):
            index_entry = (len_packet - 4) // 20
            self.process_entry(packet[entry_start_index:entry_start_index+20],neb_id)

        # Prints routing table after receiving and processing packet.
        #self.print_routing_table()





    def process_entry(self, entry, neb_id):
        """Processes one entry so the table might be changed."""
        router_id = (int(entry[6] & 0xFF)) + int((entry[7] << 8))
        entry_metric = int(entry[19])
        total_cost = min(16, self.neighbours.get(neb_id)[0] + entry_metric)

        # Change metric of the table
        if self.table.get(router_id):

            # Next hop
            metric, next_hop = self.table.get(router_id)[0], self.table.get(router_id)[1]

            #next hop is packet from
            if (next_hop == neb_id):

                # If cost is different, reinitialze the timer and change cost
                if (total_cost != metric):
                    self.init_timer(router_id)
                    self.table.get(router_id)[0] = total_cost

                    # Trigger event if metric changes to 16
                    if (total_cost == 16):
                        # Flag changes to True
                        self.table.get(router_id)[2] = True
                        # Timeout timer gets cancelled
                        self.table[router_id][3].cancel()
                        # Packet is then sent to neighbour and garbage timer is started
                        self.periodically_send_packets()
                        self.start_garbage_timer(router_id)
                else:
                    # If cost doesn't change and is not 16, reset router timers
                    if (total_cost != 16):
                        self.init_timer(router_id)

            # If next hop != neb_id  and it cost less
            else:
                if (total_cost < metric):
                    # Cancel both the timers
                    self.table[router_id][3].cancel()
                    self.table[router_id][4].cancel()
                    self.table[router_id] = [total_cost, neb_id, False, self.start_timeout(router_id), threading.Timer(self.garbage_time,
self.delete_router,(router_id,))]

        elif self.table.get(router_id) == None and total_cost != 16:
            self.table[router_id] = [total_cost, neb_id, False, self.start_timeout(router_id), threading.Timer(self.garbage_time,
self.delete_router, (router_id,))]

        else:
            return
```

```
def print_routing_table(self):
    """ Prints the routing table's of the router."""

    #self.wait(0.4)
    print("\n")
    print(" _________________________[Router {}]____________________________".format(self.self_id))
    print("|______________________________________________________________|")
    print("| Router ID | Next Hop | Cost |    Timeout   |  Garbage Timer  |")
    print("|===========|==========|======|==============|=================|")

    router_id_list = list(self.table.keys())
    router_id_list.sort()
    for router_id in router_id_list:
        metric, next_hop, flag, timeout, garbage = self.table.get(router_id)
        timeout_time = time.time() - self.timeout_timer_dict[router_id]

        if (self.garbage_timer_dict.get(router_id)):
            garbage_timer_time = time.time() - self.garbage_timer_dict[router_id]
            timeout_time = 0
        else:
            garbage_timer_time = 0

        print("|{:^11}|{:^10}|{:^6}|{:^14.3f}|{:^17.3f}|".format(router_id, next_hop, metric, timeout_time, garbage_timer_time))

    print("|___________|__________|______|______________|_________________|")
    print("\n")




def send_packet(self):
    """ Creates a packet for each neigbour and sends it to the neighbour. """
    self.give_update("Sending Packet.")
    #self.wait(0.4)
    for neighbor_id, values in self.neighbours.items():
        packet = self.create_packet(neighbor_id)
        neb_port_num = values[1]
        self.sending_socket.sendto(packet, (LOCAL_HOST, neb_port_num))




def periodically_send_packets(self):
    """ Sends packets to neigbours periodically. Done when a certain
    random amount of time has passed. """
    self.send_packet()
    t = threading.Timer(5 + 0.2 * (random.randrange(0, 6) * random.randrange(-1, 1)), self.periodically_send_packets)
    t.start()
    self.print_routing_table()




def give_update(self, message):
    """ Gives the update message and the time at which it was sent. """
    current_time = time.strftime("%Hh:%Mm:%Ss")
    print("[" + current_time + "]: " + message)
```