COSC364 22-S1 Assignment Report

Name: MENG ZHANG
Student ID: 71682325

Nameng: ZHENG CHAO
Student ID: 21671773

Questions:

1. **A percentage contribution of each partner to the overall project**

   MENG ZHANG (71682325): 60%
   ZHENG CHAO (21671773): 40%

2. **For each partner please give a brief list of contributions**

   MENG ZHANG (71682325):
   - Program structure design (coworking)
   - Program test design (coworking)
   - Implementation of following modules:
     - rip_main.py
     - rip_init.py
     - rip_router.py
     - IO_parser.py
     - forwarding_route.py
     - rip_packet.py

   ZHENG CHAO (21671773):
   - Program structure design (coworking)
   - Program test design (coworking)
   - Implementation of following modules:
     - IO_formatter.py
     - network_interface.py

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

   Adequate time and efforts have been devoted to the design of the program. Particularly, the program is well organised in the pattern of object oriented programming, with the Router class as the core of the whole program which binds the data of the routing table with its related methods. Incorporated by an Interface object simulating a network interface, a Router object can advertise, maintain, and manipulate its routing table data and implement the rip protocol. With classes like Route and RipPacket, as well as other supported functional modules (i.e. IO_parser.py, IO_formatter.py), the program is maintainable and extensible.

4. **Which aspects of your overall program (design or implementation) could be Improved?**

   Both the design and the implementation of the program could be improved, but with more time the implementation of the program, especially the incoming message processing part of the Router class could be done in a dryer and more systematic way. In addition, the output of the program could be further improved, i.e. only important new messages are updated and printed out, so that the scrolling messages could be more readable.

5. **How have you ensured the atomicity of event processing?**

   The program is single-threaded and runs in an infinite while loop (refer to the program entry point rip_main.py), in which ROUTER.receive_routes(), ROUTER.advertise_all_routes_periodically(), and ROUTER.check_timeout_entries_periodically() are invoked in order. To implement multiple timer-driven events and ensure their atomicity at the same time, each timer-driven event is assigned a unique timestamp variable, i.e. __regular_advertise_timer (refer to rip_router.py), and an event method is only invoked when its elapsed time is equal to or greater than its preset period. e.g. ROUTER.advertise_all_routes_periodically() is invoked every 3 seconds plus a random offset time; if its elapsed time is less than its preset period, the method skips and continues to the following operation.It is worth to mention that in the ROUTER.receive_routes() method, a select.select() function is called to pick up incoming packets from a given number of input sockets and will block the whole process when there is no incoming messages, in order to save CPU consumption. However, it may also delay other timer-driven events. To minimise this side effect, a timeout (i.e. 0.5 sec) is passed to select.select() so that it will block the process at most 0.5 sec in each loop.

6. **Have you identified any weaknesses of the RIP routing protocol?**

- The RIP protocol is limited to small networks whose longest path is 15 hops, due to its infinity (16) configuration. As a result, it's almost impossible to apply RIP protocol to large networks. If a bigger infinity value is chosen, i.e. 1000, to allow a larger network, the process of convergence could become extremely slow, and also violates backwards compatibility.
- The RIP protocol is designed to only use fixed metrics to compare alternative routes. If real-time costs such as network delay, load, reliability ,etc are used, it could cause highly frequent triggered updates, which would result in massive network load.
- RIP protocol lacks the mechanism to divide large networks into smaller subnetworks like OSPF protocol. The size of the message each node processes and advertises would increase proportional to the size of the whole network, which makes the protocol not as scalable as OSPF.
- The backwards compatibility for RIP protocol version 1 could cause security problems, as version 1 lacks basic authentication mechanisms.

Testing

Test 1: basic init functionality of a single router

- Case 1:
  Description: Check if a router with a valid router config file can be started correctly.
  Steps: start rip_main.py with router1_config.txt
  Expected result:
    - A router is created without error messages and runs in an infinite loop.
    - A routing table containing 1 entry (router 1 itself) is printed out periodically.
    - A regular update message is printed out for each of the three neighbours. (router 2/6/7), indicating matching router id, output port and timestamp.
    - Period is randomised and printed out each time a regular update is advertised.
    - A timeout checking message is printed out periodically with its timestamp.

  Status: Passed

```
 cosc364_assignment git:(main) > python3 rip_main.py router1_config.txt
Starts RIP Daemon...
Set Router regular update period to 3.96
Created Router 1
Sends all routes to Router2 [6020] at 00:46:56.62
Sends all routes to Router6 [6061] at 00:46:56.62
Sends all routes to Router7 [6071] at 00:46:56.62

=========================================================================
|                     Router 01 RIP ROUTING TABLE                       |
=========================================================================
|   Dest   |   Next   |   Metric   |  Timeout  |  Garbage  |   State   |
-------------------------------------------------------------------------
|    1     |    -     |     0      |     -     |     -     |  active   |
-------------------------------------------------------------------------

Set Router regular update period to 3.85
Checking timeout entries at 00:46:59.62
Sends all routes to Router2 [6020] at 00:47:00.62
Sends all routes to Router6 [6061] at 00:47:00.62
Sends all routes to Router7 [6071] at 00:47:00.62
```

- Case 2:
  Description: Check if a router with an invalid route config file can be started or not.
  Steps: start rip_main.py with router1_config_false.txt in which the ratio between period and timeout is not 6 as specified in assignment specs.
  Expected result:
    - The router does not start
    - An error message is printed out, indicating the timer value in the config file is invalid.

  Status: passed

```
 cosc364_assignment git:(main) > python3 rip_main.py router1_config_false.txt
Starts RIP Daemon...
The ratio timeout vs period should be 6
Some value of the config file is invalid
Traceback (most recent call last):
  File "/home/mz/Documents/repositories/cosc364/cosc364_assignment/rip_main.py", line 28, in <module>
    ROUTER = rip_router_init(config_file_name)
  File "/home/mz/Documents/repositories/cosc364/cosc364_assignment/rip_init.py", line 23, in rip_router_init
    router = Router(config['router_id'],
TypeError: 'NoneType' object is not subscriptable
```

Conclusion of test 1:

- The tested program can correctly parse a valid config file, start a router daemon and run into an infinite loop, printing out the expected routing table and update messages on time.
- The tested program won't start a router with an invalid config file and can print out an error message relevant to the invalid parts of the false config file.

Test 2: basic routing functionality of a 3-router network

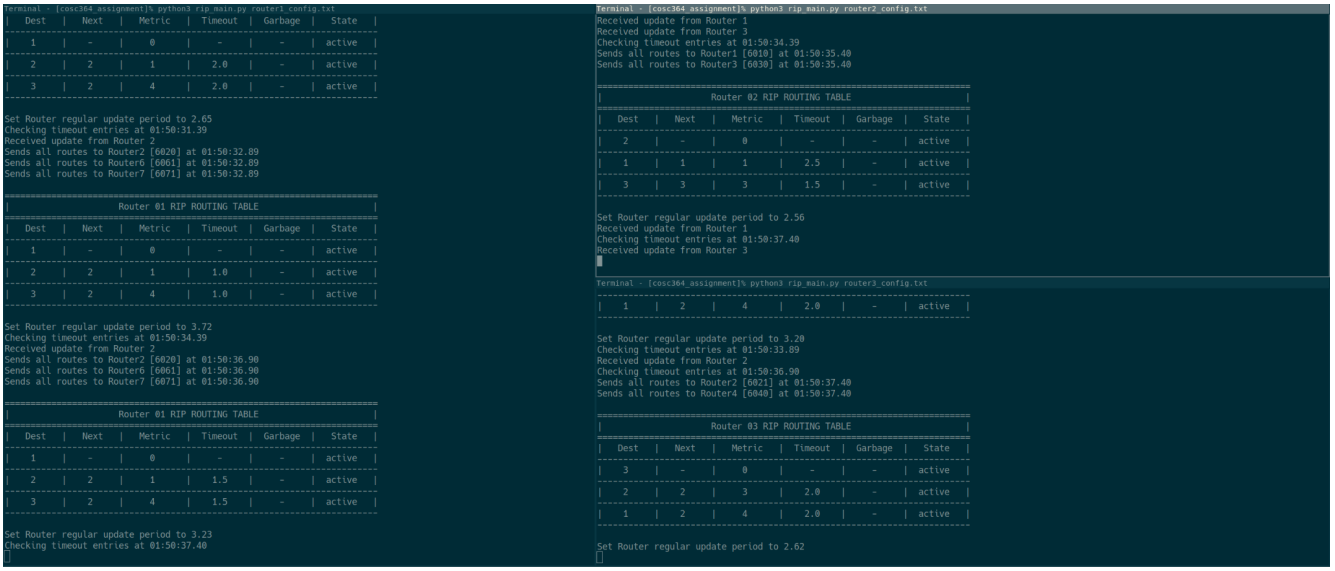
- Case 1: Discover neighbours and add new routes

  Description: Check if 3 routers can be started, discover their neighbours, and have their routing tables converge

  Steps: start router1, router2 and router3

  Expected result:
  - All 3 routers can be started without error messages.
  - All 3 routers can discover their neighbours after their first loop
  - Router 1 gets a route to router 3 through router 2
  - Router 3 gets a route to router 1 through router 2
  - All 3 routers have their routing tables converge in a short time.

  Status: Passed



- Case 2: Invalid route

  Description: Check if an invalid route can be timed out and removed after garbage collection time.

  Steps:
  1. Start router1, router2 and router3
  2. After the routing tables of the 3 routers converge, stop router 3

  Expected result:

  After the route to router 3 is invalid:
  - The timeout timer in the routing tables increments correctly
  - The garbage collection timer of the route to router 3 starts after the timeout timer expires, the route's state flag is labelled "dying", and the metric to router 3 is changed to infinity (16).

- When the garbage collection timer starts, an update for the invalid route is triggered, and a message of sending a triggered update is printed out with its timestamp.
- After the garbage collection timer expires, the route to router 3 is removed from the routing table of router 1 and router 2.
- No routing loops between router 1 and router 2

Status: Passed

```
Set Router regular update period to 2.98
Received update from Router 1
Checking timeout entries at 02:21:34.14
Triggered update for invalid route
Sends triggred update to Router1 [6010] at 02:21:34.14
Sends triggred update to Router3 [6030] at 02:21:34.14

=================================================================================
|                        Router 02 RIP ROUTING TABLE                            |
=================================================================================
|   Dest   |   Next   |   Metric   |  Timeout  |  Garbage  |   State   |
---------------------------------------------------------------------------------
|    2     |    -     |     0      |     -     |     -     |  active   |
---------------------------------------------------------------------------------
|    1     |    1     |     1      |    1.0    |     -     |  active   |
---------------------------------------------------------------------------------
|    3     |    3     |     16     |   18.5    |     0     |   dying   |
---------------------------------------------------------------------------------
```

- Case 3: Recover route
  Description: Check if a route can be recovered before/after being removed.
  Steps:
  1. Start router1, router2 and router3
  2. After the routing tables of the 3 routers converge, stop router 3
  3. Reconnect router 3 after the garbage-collection timer starts and before the route is removed from the routing table of router 2 and router 1
  4. After the routing tables of the 3 routers converge, stop router 3 again
  5. Reconnect router 3 after the route to router 3 is removed from the routing tables of router 2 and router 1

  Expected result:
  - Router 3 can be restarted
  - The route to router 3 can be added back to the routing tables of router 1 and router 2, and the stage flag is changed to “active” after router 3 is reconnected to the network
  - The routing tables of router 1/2/3 converge in a short time at last.

  Status: Passed

Conclusion of test 2:

- The tested routers can discover all their neighbours immediately after they are started.
- The tested routers can exchange and process routing table data from their neighbours and add new routes to their routing table.

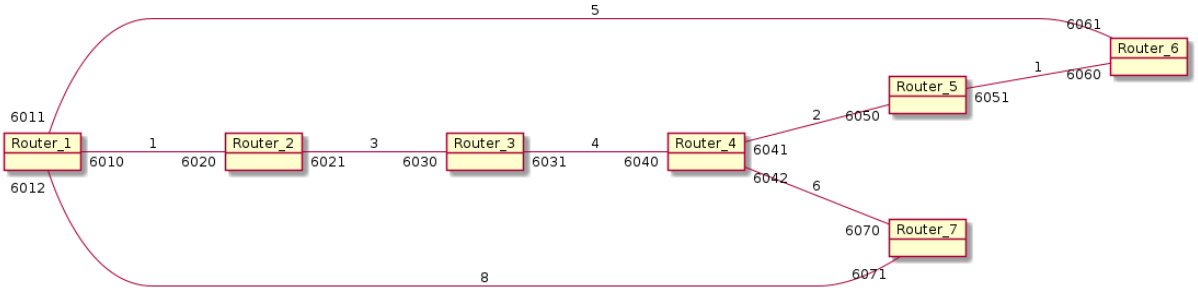
- When a route invalid, its directly-connected neighbour can start a garbage-collection process for the route after its timeout expires and send a triggered update to the neighbours immediately. When the garbage-collection timer expires, the invalid route can be removed from the routing table.
- No routing loop happens when an invalid route occurs at one end of a 3-router network, which means the implementation of split-horizon with poisoned reverse works in the program.
- A removed router can be connected back to the network during the garbage-collection process or after the route is removed.
- The routing tables can converge after new neighbours/routes are discovered, invalid routes are removed, and removed routers are reconnected.

Test 3: Routing functionality of a 7-router network

- Case 1: the shortest route

Description: Check if a router would choose the shortest route from multiple paths to the same destination

Steps: start router 1-7



Expected result:

- In a short time, all the routing tables can converge
- Each router can always choose the shortest route to other routers

Status: Passed

```
Checking timeout entries at 03:34:56.82
Received update from Router 7
Sends all routes to Router2 [6020] at 03:34:58.83
Sends all routes to Router6 [6061] at 03:34:58.83
Sends all routes to Router7 [6071] at 03:34:58.83

=================================================================
|                  Router 01 RIP ROUTING TABLE                  |
=================================================================
|   Dest   |   Next   |   Metric   |  Timeout  |  Garbage  |   State   |
-----------------------------------------------------------------
|    1     |    -     |     0      |     -     |     -     |  active   |
-----------------------------------------------------------------
|    2     |    2     |     1      |    2.5    |     -     |  active   |
-----------------------------------------------------------------
|    3     |    2     |     4      |    2.5    |     -     |  active   |
-----------------------------------------------------------------
|    4     |    2     |     8      |    2.5    |     -     |  active   |
-----------------------------------------------------------------
|    5     |    6     |     6      |    3.5    |     -     |  active   |
-----------------------------------------------------------------
|    6     |    6     |     5      |    3.5    |     -     |  active   |
-----------------------------------------------------------------
|    7     |    7     |     8      |    1.0    |     -     |  active   |
-----------------------------------------------------------------

Set Router regular update period to 2.38
```

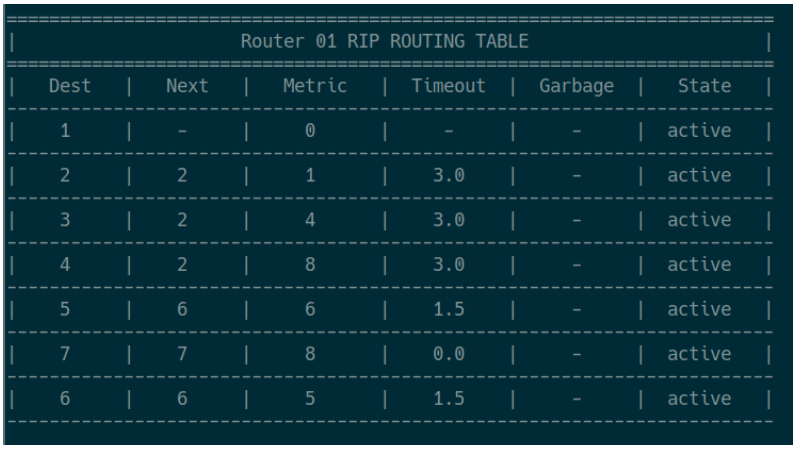
- Case 2: the shortest route after router removed

  Description: Check if a router would choose another optimal route after a neighbour through which a shortest route has passed is removed

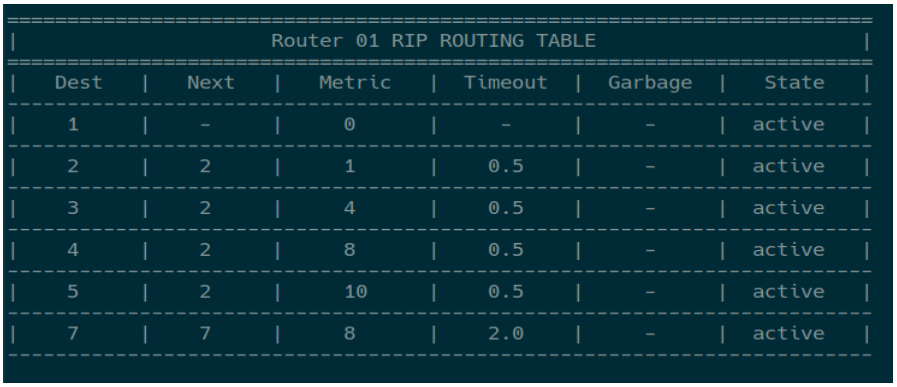
  Steps:
  - Start router 1-7
  - Wait until all routing tables converge
  - Remove router 6
  - Wait until all routing tables converge
  - Reconnect router 6
  - Wait until all routing tables converge

  Expected result:
  - Before removing router 6, router 1 chooses router 6 as the next hop to get to router 5
  - After removing router 6, router 1 chooses router 2 as the next hotp to get to router 5
  - After reconnecting router 6, router 1 again chooses router 6 as the next hop to get to router 5

  Status: Passed



Router 01 RIP ROUTING TABLE					
Dest	Next	Metric	Timeout	Garbage	State
1	-	0	-	-	active
2	2	1	3.0	-	active
3	2	4	3.0	-	active
4	2	8	3.0	-	active
5	6	6	1.5	-	active
7	7	8	0.0	-	active
6	6	5	1.5	-	active



Router 01 RIP ROUTING TABLE					
Dest	Next	Metric	Timeout	Garbage	State
1	-	0	-	-	active
2	2	1	0.5	-	active
3	2	4	0.5	-	active
4	2	8	0.5	-	active
5	2	10	0.5	-	active
7	7	8	2.0	-	active

Conclusion of test 3:

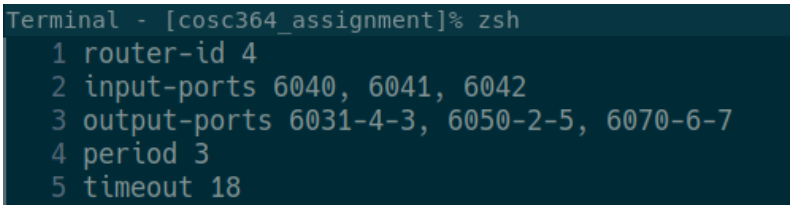
- When multiple paths are available to the same destination, a router can always choose the shortest route
- When a router through which a shortest route has passed is removed, a router can always choose another shortest route available to the same destination

- The routing tables can alway converge at last.

From test 1-3, we can conclude that the tested program can implement the rip protocol features specified in the assignment specs.

One example configuration file for the example network of Figure 1.

The configuration file for router 4:



```
"""
COSC364 2022-S1 Assignment: RIP routing
Authors: MENG ZHANG (71682325), ZHENG CHAO (21671773)
File: rip_main.py
"""


# Import Modules
import sys
import time
import threading
from rip_init import rip_router_init


# Program Entry Point
if __name__ == "__main__":
    print("Starts RIP Daemon...")
    # get config file name
    try:
        if len(sys.argv) != 2:
            raise ValueError("Invalid argument for rip_main\n" +
                             "Rip router requires ONE config file")
        config_file_name = sys.argv[1]
    except ValueError as error:
        print(error)

    # Initialise a new Router object
    ROUTER = rip_router_init(config_file_name)

    # First advertise ROUTER itself immediately
    ROUTER.advertise_routes('all')
    ROUTER.print_routing_table()
    ROUTER.random_offset_period()

    # Start loop
    while True:
        ROUTER.receive_routes()
        ROUTER.advertise_all_routes_periodically()
        ROUTER.check_timeout_entries_periodically()
```

```python
"""
COSC364 2022-S1 Assignment: RIP routing
Authors: MENG ZHANG (71682325), ZHENG CHAO (21671773)
File: rip_init.py
"""


# Import Modules
from IO_parser import router_config
from rip_router import Router


# Initialise router
def rip_router_init(config_file_name):
    """
    Parameter:
    config_file_name: the name of the config file to initialise a new
    router object.

    Return: a new Router object
    """
    config = router_config(config_file_name)
    router = Router(config['router_id'],
                    config['input_ports'],
                    config['output_ports_metric_id'],
                    config['period'],
                    config['timeout'])
    print(f"Created Router {router.get_router_id()}")
    return router
```

```
"""
COSC364 2022-S1 Assignment: RIP routing
Authors: MENG ZHANG (71682325), ZHENG CHAO (21671773)
File: IO_parser.py
"""

def router_config(file_name):
    """
    Parameter:
    file_name: string
    file format:
    i.e.
    ------------------------------
    router-id 2
    input-ports 6020, 6021
    output-ports 6010-1-1, 6030-2-3
    period 3
    timeout 18
    ------------------------------

    Return: config_data
    a dictionary with 4 keys of router_id, input_ports, output_ports,
    timers
    i.e. {'router_id': 2, 'input_ports': [6020, 6021],
    'output_ports_metric_id': {6010: {'metric': 1, 'router_id': 1},
                               6020: {...}}, 'period': 3, 'timeout': 18}
    """
    raw_config = read_config(file_name)
    config_data = parse_config(raw_config)
    return config_data


def read_config(file_name):
    """
    Parameter:
    file_name: string.
    file format:
    i.e.
    ------------------------------
    router-id 2
    input-ports 6020, 6021
    output-ports 6010-1-1, 6030-2-3
    period 3
    timeout 18
    ------------------------------

    Return: a list of strings with 4 elements.
    i.e. ['router-id 2', 'input-ports 6020, 6021', 'output-ports 6010-1-1,
          6030-2-3', 'period 3', 'timeout 18']
    """
    try:
        with open(file_name) as config_file:
            raw_config = config_file.read().splitlines()
            return raw_config
    except FileNotFoundError:
        print("Error: the config file name is invalid")


def parse_config(raw_config):
    """
    Parameter:
    raw_config: a list of strings with 4 elements.
    i.e. ['router-id 2', 'input-ports 6020, 6021', 'output-ports 6010-1-1,
          6030-2-3', 'period 3', 'timeout 18']

    Return: config_data
```

```
    a dictionary with 4 keys of router_id, input_ports, output_ports,
    timers
    i.e. {'router_id': 2, 'input_ports': [6020, 6021],
          'output_ports_metric_id': {6010: {'metric': 1, 'router_id': 1},
                                     6020: {...}}, 'period': 3, 'timeout': 18}
    """
    try:
        # get router id
        router_id = parse_id(raw_config[0])
        # get input ports
        input_ports = parse_input_ports(raw_config[1])
        # check if input ports contains duplicate ports
        if contains_duplicates(input_ports):
            raise ValueError("The input ports contains duplicate ports")
        # get output ports
        output_ports, output_ports_metric_id = parse_output_ports(raw_config[2])
        # check if input ports and output ports contain duplicate ports
        if duplicate_lists(input_ports, output_ports):
            raise ValueError("The input ports and output ports contain duplicate ports")
        # get period
        period = parse_period(raw_config[3])
        # get timeout
        timeout = parse_timeout(raw_config[4])
        # check timeout vs period ratio
        if not is_valid_timer_ratio(period, timeout):
            raise ValueError("The ratio timeout vs period should be 6")
        # create coinfig_data dictionary
        config_data = {"router_id": router_id, "input_ports": input_ports,
                       "output_ports_metric_id": output_ports_metric_id,
                       "period": period, "timeout": timeout}
        return config_data
    except IndexError as ie:
        print(ie)
        print("Some value of the config file is not available")
    except ValueError as ve:
        print(ve)
        print("Some value of the config file is invalid")


def parse_id(raw_id):
    """
    Parameter:
    raw_id: a string
    i.e. 'router-id 2'

    Return: router_id
    an interger between 1 and 64000 i.e. 1
    """
    try:
        router_id = int(raw_id.split()[1])
        if (router_id < 1 or router_id > 64000):
            raise ValueError("Router ID value is out of bounds")
        return router_id
    except IndexError as e:
        print(e)
        print("The config router ID value is not available")
    except ValueError as e:
        print(e)
        print("The config router ID value must be an integer between 1 and 64000")


def parse_input_ports(raw_input_ports):
    """
    Parameter:
    raw_input_ports: a string
    i.e 'input-ports 6020, 6021'
```

```
    Return: input_ports
    a list of integers which are between 1024 and 64000
    i.e. [6020, 6021]
    """
    try:
        input_ports_temp = raw_input_ports.split()[1:]
        input_ports = []
        for port_str in input_ports_temp:
            port_int = int(port_str.strip(','))
            if (port_int < 1024 or port_int > 64000):
                raise ValueError("Input port value is out of bounds")
            input_ports.append(port_int)
        return input_ports
    except IndexError as e:
        print(e)
        print("The config input port value is not available")
    except ValueError as e:
        print(e)
        print("The config input port value must be an integer between 1024 and 64000")


def parse_output_ports(raw_output_ports):
    """
    Parameter:
    raw_input_ports: a string
    i.e 'output-ports 6010-1-1, 6030-2-3'

    Return: output_ports, output_ports_metric_id
    output_ports: a list of integers which are between 1024 and 64000
    i.e. [6010, 6030]
    output_ports_metric_id: a dict of dicts in which key is port number
    and each sub dict contains key(port)'s metric and id.
    Metric > 0, 1 <= ID <= 64000
    i.e. {6010: {'metric': 1, 'router_id': 1}, 6020: {...}}
    """
    try:
        output_ports_combo_temp = raw_output_ports.split()[1:]
        output_ports = []
        output_ports_metric_id = {}
        for port_combo_str in output_ports_combo_temp:
            port_combo_temp = port_combo_str.strip(',').split('-')
            port_int = int(port_combo_temp[0])
            metric_int = int(port_combo_temp[1])
            id_int = int(port_combo_temp[2])
            if (port_int < 1024 or port_int > 64000):
                raise ValueError("Ouput port value is out of bounds")
            if metric_int < 1:
                raise ValueError("Output port metric is out of bounds")
            if id_int < 1 or id_int > 64000:
                raise ValueError("Output id is out of bounds")
            output_ports.append(port_int)
            # output_ports_metric_id.append([port_int, metric_int, id_int])
            output_ports_metric_id[port_int] = {'metric': metric_int,
                                                'router_id': id_int}
        return output_ports, output_ports_metric_id
    except IndexError as e:
        print(e)
        print("The config output port value is not available")
    except ValueError as e:
        print(e)
        print("The config output ports must be fomatted as port-metric-id")
        print("The config output port value must be an integer between 1024 and 64000")
        print("The config output port metric must be an integer greater than 0")
        print("The config output port id must be an integer between 1 and 64000")

```

```
200
201 def parse_period(raw_period):
202     """
203     Parameter:
204     raw_period: a string
205     i.e. 'period 3'
206
207     Return: period
208     period: a positive integer
209     i.e. 3
210     """
211     try:
212         period = int(raw_period.split()[1])
213         if period < 1:
214             raise ValueError("Router period value is out of bounds")
215         return period
216     except IndexError as e:
217         print(e)
218         print("The config router period value is not available")
219     except ValueError as e:
220         print(e)
221         print("The config router timeout value must be a positive integer")
222
223 def parse_timeout(raw_timeout):
224     """
225     Parameter:
226     raw_timeout: a string
227     i.e. 'timeout 18'
228
229     Return: timeout
230     timeout: a positive integer
231     i.e. 18
232     """
233     try:
234         timeout = int(raw_timeout.split()[1])
235         if timeout < 1:
236             raise ValueError("Router timeout value is out of bounds")
237         return timeout
238     except IndexError as e:
239         print(e)
240         print("The config router timeout value is not available")
241     except ValueError as e:
242         print(e)
243         print("The config router timeout value must be a positive integer")
244
245
246
247 def contains_duplicates(lst):
248     """
249     Parameter:
250     lst: a list
251
252     Return: boolean
253     if the lst contains duplicates, return true, otherwise false
254     """
255     return len(set(lst)) != len(lst)
256
257 def duplicate_lists(lst1, lst2):
258     """
259     Parameters:
260     lst1: a list
261     lst2: a list
262
263     Return: boolean
264     if the two lists contains duplicate items, return true, otherwise false
265     """
266     return len(set(lst1).union(set(lst2))) != len(lst1) + len(lst2)
267
```

```
268
269 def is_valid_timer_ratio(period, timeout):
270     """
271     Parameters:
272     period: a positive integer
273     period: a positive integer
274
275     Return: boolean
        if timeout / period = 6, return true, otherwise false
        """
        return timeout / period == 6
```

```python
"""
COSC364 2022-S1 Assignment: RIP routing
Authors: MENG ZHANG (71682325), ZHENG CHAO (21671773)
File: rip_router.py
"""

# Import Modules
import time
import random
from datetime import datetime
from network_interface import Interface
from forwarding_route import Route
from rip_packet import RipPacket, RipEntry
from IO_formatter import routing_table_formatter


# Router Class
class Router:
    """
    An object that simulates a router with rip protocol
    """

    # Class attributes
    INFINITY = 16
    REGULAR_TIMER_OFFSET = 1.0

    def __init__(self, router_id,
                 inputs, outputs,
                 period, timeout):
        """
        the __* attributes are private attributes which can only be
        accessed by getter outside of class.

        Parameters:
        router_id: an integer, i.e. 1, 2, 3, etc
        inputs: a list of integers, i.e. [5001, 5002, 5003]
        outputs: a dictionary of dictionaries, i.e.
        {6010(port): {'metric': 1, 'router_id': 1},
         6030(port): {'metric': 2, 'router_id': 3},
              ... : {...}
        }
        period: an integer
        timout: an integer
        """
        # Instance attributes
        self.__router_id = router_id
        self.__split_horizon_poison_reverse = True
        self.__input_ports = inputs
        self.__output_ports = outputs
        self.__regular_advertise_timer = time.time()
        self.__default_period = period
        self.__period = period
        self.__trigger_advertise_timer = time.time()
        self.__default_triggered_updates_period = period / 6
        self.__triggered_updates_period = period / 6
        self.__timeout_check_timer = time.time()
        self.__timeout = timeout
        self.__garbage_collection_time = period * 4
        self.__interface = None
        self.__routing_table = {}
        # Initialisation
        self.init_interface(inputs)
        self.init_routing_table()
        self.random_offset_period()


```

```
64      def get_router_id(self):
65          """
66          router_id getter
67          """
68          return self.__router_id
69
70
71      def set_router_id(self, new_id):
72          """
73          router_id setter
74          """
75          self.__router_id = new_id
76
77
78      def get_input_ports(self):
79          """
80          router input_ports getter
81          """
82          return self.__input_ports
83
84
85      def set_input_ports(self, new_inputs):
86          """
87          router input_ports setter
88          """
89          self.__input_ports = new_inputs
90          self.init_interface(new_inputs)
91
92
93      def get_output_ports(self):
94          """
95          router output_ports getter
96          """
97          return self.__output_ports
98
99
100     def set_output_ports(self, new_outputs):
101         """
102         router output_ports setter
103         """
104         self.__output_ports = new_outputs
105
106
107     def get_period(self):
108         """
109         router period getter
110         """
111         return self.__period
112
113
114     def set_period(self, new_period):
115         """
116         router period setter
117         """
118         self.__period = new_period
119
120
121     def get_timeout(self):
122         """
123         router timeout getter
124         """
125         return self.__timeout
126
127
128     def set_timeout(self, new_timeout):
129         """
130         router timout setter
131
```

```
        """
        self.__timeout = new_timeout


    def get_interface(self):
        """
        router interface getter
        """
        return self.__interface


    def get_routing_table(self):
        """
        router routing_table getter
        """
        return self.__routing_table


    def print_routing_table(self):
        """
        Print the current self.__routing_table
        """
        print(routing_table_formatter(self.__router_id,
                                      self.__routing_table))


    def random_offset_period(self):
        """
        randomize self.__period +- TIMER_OFFSET
        """
        self.__period = self.__default_period +\
            random.uniform(-self.REGULAR_TIMER_OFFSET, \
                           +self.REGULAR_TIMER_OFFSET)
        print("Set Router regular update period to " + \
              f"{self.__period:.2f}")


    def random_triggered_updates_period(self):
        """
        randomize self.__triggered_updates_period
        """
        self.__triggered_updates_period = \
            self.__default_triggered_updates_period -\
            random.uniform(0, 0.4)
        print("Set Router triggered update period to " + \
              f"{self.__triggered_updates_period:.2f}")


    def init_interface(self, ports):
        """
        Create a new Interface object and set it as the default
        interface for the current Router object
        """
        self.__interface = Interface(ports)


    def init_routing_table(self):
        """
        Initialise the __routing_table attribute

        Route object format:
        route.next_hop: 2,
        route.metric: 1,
        route.timeout: 1234,
        route.garbage_collect_time: None(default)
        state: 'active'(default)
        """
```

```
        # Create a new Route object to router itself
        self_route = Route('-', 0, None)
        self.__routing_table[self.__router_id] = self_route


#---------------------------------------------------
# Above is the init implementation
#---------------------------------------------------


    def advertise_all_routes_periodically(self):
        """
        Call advertise_all_routes() periodcally by self.__period

        Use random.random() to calculate offset for self.__period
        in order to avoid synchronized update messages which can lead
        to unnecessary collisions on broadcast networks.
        """
        now = time.time()
        if now - self.__regular_advertise_timer >= self.__period:
            self.advertise_routes('all')
            self.print_routing_table()
            self.__regular_advertise_timer = now
            self.random_offset_period()


    def advertise_updated_routes(self):
        """
        advertise the updated routes to all neighbours
        """
        now = time.time()
        if now - self.__trigger_advertise_timer >= \
           self.__triggered_updates_period:
            self.advertise_routes('update')
            self.print_routing_table()
            self.__trigger_advertise_timer = now
            self.random_triggered_updates_period()


    def advertise_routes(self, mode):
        """
        parameter:
        mode: a string 'all' / 'update'
        get the latest advertising rip packet from
        update_packet() & triggered_packet() methods and
        advertise the packet to all the neighbours (ouput ports)

        need to add a parameter for updata_packet/triggered_packet
        """
        try:
            ports_num = len(self.__output_ports)
            if ports_num < 1:
                raise ValueError("No output port/socket available")
            for dest_port, metric_id in self.__output_ports.items():
                packet = self.update_packet(metric_id['router_id'], mode)
                if packet is None:
                    print("A packet without entry. Stop Sending")
                    return
                self.__interface.send(packet, dest_port)
                current_time = datetime.now().strftime('%H:%M:%S.%f')[:-4]
                if mode == 'all':
                    message = "Sends all routes to Router"
                else:
                    message = 'Sends triggred update to Router'
                print(message +
                      f"{metric_id['router_id']} " +
                      f"[{dest_port}] at {current_time}")
```

```
            # clear flags of "update"
            for route in self.__routing_table.values():
                if mode == 'update' and route.state == 'updated':
                    route.state = 'active'
        except ValueError as error:
            print(error)


    def update_packet(self, receiver_id, mode):
        """
        parameter:
        receiver_port

        Process the current routing table data and convert it into
        a rip format packet for advertise_all_routes() method
        """
        # Create RipEntries for all the routes
        entries = []
        for dest, route in self.__routing_table.items():

            if mode == "update" and route.state == "active":
                continue
            metric = route.metric
            # split_horizon_poison_reverse
            if self.__split_horizon_poison_reverse and\
               route.next_hop == receiver_id:
                metric = self.INFINITY
            entry = RipEntry(dest, metric)
            entries.append(entry)

        # Create RipPacket
        packet = RipPacket(entries, self.__router_id)
        packet_bytes = packet.packet_bytes()
        return packet_bytes


#---------------------------------------
# Above is sender implementation
#---------------------------------------


    def receive_routes(self):
        """
        Receive the routes update from neighbours (input ports)

        The implementation is in a while loop and should be called with
        a separate thread from the main thread
        """
        # The __interface only listen to the input ports
        # print(f"Listening to ports at {time.ctime()}")
        packets_list = self.__interface.receive()
        for raw_packet in packets_list:
            self.process_received_packet(raw_packet)


    def process_received_packet(self, raw_packet):
        """
        Process the received packet and call update_routing_table()
        if necessary

        Parameter: packet
        an array of bytes
        """
        # Check if raw_packet valid in RipPacket and RipEntry classes
        # Process the raw_packet if valid,
        # and return (True, RipPacket object)
        # otherwise, return (False, router_id)
```

```
        is_valid, rip_packet = RipPacket.decode_packet(raw_packet)
        if is_valid:
            # update routing_table if incoming packet is valid
            print(f'Received update from Router {rip_packet.router_id}')
            self.update_routing_table(rip_packet)
        else:
            # drop the packet if incoming packet is invalid
            print(f'Drop invalid packet from Router {rip_packet}')


    def update_routing_table(self, rip_packet):
        """
        check all the entries in rip_packet object, and update current
        routing table if necessary

        Parameter:
        rip_packet: a valid rip_packet object

        Reture: boolean
        return True if new route added, otherwise False
        """
        # get metric from sender
        sender_id = rip_packet.router_id
        metric_to_sender = None
        for neighbour in self.__output_ports.values():
            if neighbour['router_id'] == sender_id:
                metric_to_sender = neighbour['metric']
        for entry in rip_packet.entries:
            # update the metric for each entry
            # by adding the metric to sender
            # metric = min(metric + metric_to_sender, 16(infinity))
            updated_metric = min(entry.metric + metric_to_sender,
                                 self.INFINITY)
            #if route to dest is unavailable in __routing_table
            if updated_metric != self.INFINITY and\
               not entry.dest in self.__routing_table.keys():
                self.__routing_table[entry.dest] = \
                    Route(sender_id, updated_metric, time.time())
                # Triggered update for new route
                # self.__routing_table[entry.dest].state = 'updated'
                # print("Triggerd update for new route")
                # self.advertise_updated_routes()
            elif entry.dest in self.__routing_table:
                self.update_availabe_route(entry,
                                           updated_metric,
                                           sender_id)


    def update_availabe_route(self, entry, updated_metric, sender_id):
        """
        Parameters:
        entry: a RipEntry object
        sender_id: the router id from which the entry is sent
        """
        # if route to dest is available in __routing_table

        # 1. if packet is from the same router as
        # existing router, reinitialize the timeout anyway
        from_same_router = sender_id == \
            self.__routing_table[entry.dest].next_hop
        is_timeout = not \
            self.__routing_table[entry.dest].garbage_collect_time is \
            None
        if from_same_router:
            self.__routing_table[entry.dest].timeout = time.time()
```

```
        # 2. compare metrics
        new_metric = updated_metric
        old_metric = self.__routing_table[entry.dest].metric
        have_differnt_metrics = new_metric != old_metric
        is_lower_new_metric = new_metric < old_metric
        is_almost_timeout = \
            not self.__routing_table[entry.dest].timeout is None and \
            not is_timeout and \
            (time.time() - self.__routing_table[entry.dest].timeout) \
            >= self.__timeout / 2

        if from_same_router and have_differnt_metrics:
            self.__routing_table[entry.dest].metric = new_metric
            if not is_timeout and new_metric == self.INFINITY:
                self.__routing_table[entry.dest].garbage_collect_time \
                    = time.time()
                # Triggered update for invalid route
                self.__routing_table[entry.dest].state = 'dying'
                print("triggered update for invalid route")
                self.advertise_updated_routes()
            elif is_timeout:
                self.__routing_table[entry.dest].garbage_collect_time \
                    = None
                self.__routing_table[entry.dest].state = 'active'

        elif is_lower_new_metric:
            self.__routing_table[entry.dest].metric = new_metric
            self.__routing_table[entry.dest].next_hop = sender_id
            self.__routing_table[entry.dest].timeout = time.time()
            if is_timeout:
                self.__routing_table[entry.dest].garbage_collect_time \
                    = None
                self.__routing_table[entry.dest].state = 'active'
            # Triggered update
            # self.__routing_table[entry.dest].state = 'updated'
            # print("triggered updated route from different router with lower metric")
            # self.advertise_updated_routes()
        elif not from_same_router and \
             not have_differnt_metrics and \
             not is_timeout and is_almost_timeout:
            self.__routing_table[entry.dest].next_hop = sender_id
            self.__routing_table[entry.dest].timeout = time.time()


#---------------------------------------
# Above is receiver implementation
#---------------------------------------

    def check_timeout_entries_periodically(self):
        """
        call check_timeout_entries() every default_period
        """
        now = time.time()
        if now - self.__timeout_check_timer >= self.__default_period:
            self.check_timeout_entries()
            self.__timeout_check_timer = now


    def check_timeout_entries(self):
        """
        Check the timeout of each entry in __routing_table

        if an entry is timeout, start its garbage_collect_time
        """
        current_time = datetime.now().strftime('%H:%M:%S.%f')[:-4]
        print(f"Checking timeout entries at {current_time}")

```

```
        entries_to_remove = []
        for dest_id, entry in self.__routing_table.items():
            if not entry.timeout is None and \
               entry.garbage_collect_time is None and \
               time.time() - entry.timeout >= self.__timeout:
                entry.garbage_collect_time = time.time()
                entry.metric = self.INFINITY
                entry.state = 'dying'
                # Triggered update
                print("Triggered update for invalid route")
                self.advertise_updated_routes()

            if not entry.garbage_collect_time is None and \
                (time.time() - entry.garbage_collect_time) \
                >= self.__garbage_collection_time:
                entries_to_remove.append(dest_id)

        for dest_id in entries_to_remove:
            self.__routing_table.pop(dest_id)
            print(f"Removed timeout route to {dest_id}")
            self.print_routing_table()


#---------------------------------------------------------
# Above is timeout and garbage_collection implementation
#---------------------------------------------------------

    def __str__(self):
        return ("Router: {0}\n"
                "Input Ports: {1}\n"
                "Output Ports: {2}\n"
                "Period: {3}\n"
                "Timeout: {4}").format(self.__router_id,
                                       self.__input_ports,
                                       self.__output_ports,
                                       self.__period,
                                       self.__timeout)
```

```python
"""
COSC364 2022-S1 Assignment: RIP routing
Authors: MENG ZHANG (71682325), ZHENG CHAO (21671773)
File: router_interface.py
"""

# Import Modules
import socket
import select


# Router Network Interface Class
class Interface:
    """
    A router interface object which includes:
    * Multiple sockets with corresponding ports as instance attributes
    * A series of methods for socket operations:
    - send(port),
    - receive(port)
    """
    def __init__(self, ports):
        """
        Parameters: ports
        ports: a list of integers of port number
        """
        self.host = "127.0.0.1" # local host
        self.select_timeout = 0.5 # default 0.5
        self.ports = ports # input ports
        self.sending_port = ports[0] # set 1st port as the sending port
        self.ports_sockets = {} # input ports and sockets
        self.init_sockets()

    def init_sockets(self):
        """
        Parameter: ports
        ports: a list of integers of ports

        Return: port_socket
        port_socket: a list of
        """
        try:
            for port in self.ports:
                udp_socket = socket.socket(socket.AF_INET,
                                           socket.SOCK_DGRAM)
                udp_socket.bind((self.host, port))
                # udp_socket.setblocking(0)  # blocking switch

                self.ports_sockets[port] = udp_socket
        except socket.error as error:
            print("Failed to initialise sockets for ports\n", error)

    def get_ports_sockets(self):
        """
        ports_sockets getter
        """
        return self.ports_sockets

    def receive(self):
        """
        Using select() to monitor a list of ports and receive the port
        with readable data

        Parameter: sockets
        ports: a list of socket objects

        Return: (data, port)
```

```
        """
        sockets = []
        for input_socket in self.ports_sockets.values():
            sockets.append(input_socket)
        sockets_to_read = (select.select(sockets, [], [], \
                                         self.select_timeout))[0]
        data_list = []
        for socket_to_read in sockets_to_read:
            # get the receiving port number which the socket binds
            # port = socket_to_read.getsockname()
            # get data from socket
            data = socket_to_read.recv(1024)
            data_list.append(data)
        return data_list

    def send(self, data_bytes, dest_port):
        """
        Parameter: data_bytes
        data_bytes: data in bytes format
        i.e. data can be the update packet from router
        """
        try:
            sending_socket = self.ports_sockets[self.sending_port]
            dest = (self.host, dest_port)
            sending_socket.sendto(data_bytes, dest)
        except KeyError:
            print("The port for sending packet does not exist")
        except socket.error as error:
            print("Can't send packet with the socket\n" + error)

    def __str__(self):
        return ("Host: {0}\n"
                "Ports: {1}\n"
                "Ports_Sockets: {2}").format(self.host,
                                             self.ports,
                                             self.ports_sockets)
```

```python
"""
COSC364 2022-S1 Assignment: RIP routing
Authors: MENG ZHANG (71682325), ZHENG CHAO (21671773)
File: forwarding_route.py
"""

# Route Class
class Route:
    """
    A Route class for creating entries of RIP routing table

    Why we use a class instead of a dictionary/list for route?
    * Compared to list we can quickly get a route value by name
    instead of number index. i.e. route.nexthop
    * Compared to dict/list, a Route class avoid accidental
    modification.
    i.e. What if we accidentally do: route[error_key] = error
    """
    def __init__(self, next_hop, metric,
                 timeout, garbage_collect_time=None,
                 state = 'active'):
        """
        parameters:
        next_hop: an integer of router ID, i.e. 2, 3
        metric: an integer, i.e. 1, 5, 7
        timeout: the current time obtained by time.time()
        garbage_collect_time: None or the current time
        state: a string, i.e. 'active', 'dying', 'updated'
        """
        self.next_hop = next_hop
        self.metric = metric
        self.timeout = timeout
        self.garbage_collect_time = garbage_collect_time
        self.state = state
```

```
"""
COSC364 2022-S1 Assignment: RIP routing
Authors: MENG ZHANG (71682325), ZHENG CHAO (21671773)
File: rip_packet.py
"""
# RipPacket Class
class RipPacket:
    """
    A class for creating RIP update packet and provide methods to
    encode/decode outputing/incoming packets
    """
    # class attributes
    HEADER_LEN = 4
    ENTRY_LEN = 20


    def __init__(self, entries, router_id, command=2, version=2):
        """
        Parameters:
        entries: a list of rip entry objects
        router_id: the sender ID,  an integer between 1 and 64000
        (use the 16-bit wide all-zero field)
        command: an integer,
        i.e. 2 represents 'response'; 1 represents 'request'
        version: an integer, i.e. 1, 2(default)
        """
        # instance attributes
        self.command = command
        self.version = version
        self.router_id = router_id
        self.entries = entries


    @classmethod
    def decode_packet(cls, raw_packet):
        """
        Parameter:
        raw_packet: a packet of bytes
        i.e.
        HEADER:
        [command(1 byte), version(1), sender_id(2)]
        ENTRY:
        [afi(2 bytes), padding(2)
        dest(4)
        padding(4)
        padding(4)
        metric(4)]

        Return (True, RipPacket object) if raw_packet is valid,
        otherwise return (False, sender_id)
        """
        # Header: 4 bytes [0:4]
        command = raw_packet[0]
        version = raw_packet[1]
        sender_id = (raw_packet[2] << 8) + raw_packet[3]
        entries_num = int(len(raw_packet[4:]) / cls.ENTRY_LEN)
        # check header validity
        if not cls.is_valid_header(command, version,
                                   sender_id, entries_num):
            print("Broken packet:", "invalid header")
            return (False, sender_id)
        # Entries: n * 20 bytes [4:]
        # decode each entry
        entries = []
        for i in range(4, len(raw_packet), cls.ENTRY_LEN):
            raw_entry = raw_packet[i:i+cls.ENTRY_LEN]
```

```
            entry = RipEntry.decode_enty(raw_entry)
            # check entry validity
            # invalid entry is represented as None
            if entry is None:
                print("Broken packet:", "invalid entry")
                return (False, sender_id)
            entries.append(entry)
        rip_packet = RipPacket(entries, sender_id)
        return (True, rip_packet)


    def packet_bytes(self):
        """
        Return a rip packet in array of bytes
        """
        return self.header_bytes() + self.entries_bytes()


    def header_bytes(self):
        """
        Common header: 4 bytes in total
        [command(1 byte), version(1), sender_id(2)]

        command:2 (1 byte)
        version: 2 (1 byte)
        rouer_id: 1-64000 (2 bytes)

        Return a 4-byte rip header.
        """
        command_byte = self.command.to_bytes(1, byteorder='big')
        version_byte = self.version.to_bytes(1, byteorder='big')
        sender_id_bytes = self.router_id.to_bytes(2, byteorder='big')
        header = command_byte + version_byte + sender_id_bytes
        return header


    def entries_bytes(self):
        """
        Return a list of 20-byte rip entry
        """
        entries = bytes()
        for entry in self.entries:
            entries += entry.entry_bytes()
        return entries

    @classmethod
    def is_valid_header(self, command, version, router_id, entries_num):
        """
        check if a packet is valid
        """
        is_valid_command = command == 2
        if (not is_valid_command):
            print(f"invalid header command: {command}")
        is_valid_version = version == 2
        if (not is_valid_version):
            print(f"invalid header version: {version}")
        is_valid_id = 1 <= router_id <= 64000
        if (not is_valid_id):
            print(f"invalid header id: {router_id}")
        is_valid_entries_num = 1 <= entries_num <= 25
        if (not is_valid_entries_num):
            print(f"invalid header entries num: {entries_num}")
        return is_valid_command and\
               is_valid_version and\
               is_valid_id and\
               is_valid_entries_num
```

```python


# RipEntry Class
class RipEntry:
    """
    A class for creating entry objects in a rip packet
    """

    # class attributes
    PADDING_2BYTES = (0).to_bytes(2, byteorder='big')
    PADDING_4BYTES = (0).to_bytes(4, byteorder='big')


    def __init__(self, dest, metric, afi=2):
        """
        Parameters:
        dest: an integer, router_id of destination
        metric: an integer between 1 and 16 (inclusive)
        AFI: Address FAmily Identifier
        """
        self.dest = dest
        self.metric = metric
        self.afi = afi


    @classmethod
    def decode_enty(cls, raw_entry):
        """
        Parameter:
        raw_entry: an entry of bytes
        i.e.
        ENTRY:
        [afi(2 bytes), padding(2)
        dest(4)
        padding(4)
        padding(4)
        metric(4)]

        Return RipEntry object if raw_entry is valid,
        otherwise return None
        """
        # afi: 2 bytes [0:3]
        afi = (raw_entry[0] << 8) + raw_entry[1]
        # dest: 4 bytes but practically take 2 bytes [4:8]
        if (raw_entry[4] != 0 or
            raw_entry[5] != 0):
            print("Invalid dest of entry")
            return None
        dest = (raw_entry[6] << 8) + raw_entry[7]
        # metric 4 bytes but practically take 1 byte [16:]
        if (raw_entry[16] != 0 or
            raw_entry[17] != 0 or
            raw_entry[18] != 0):
            print("Invalid metric of entry")
            return None
        metric = raw_entry[19]
        entry = RipEntry(dest, metric, afi)
        if not entry.is_valid_entry():
            return None
        return entry


    def entry_bytes(self):
        """
        Rip entry: 20 bytes each
        [afi(2 bytes), padding(2)
        dest(4)
```

```
        padding(4)
        padding(4)
        metric(4)]

        afi: 2 (2 bytes)
        dest: 1-64000 (4 bytes)
        metric: 1-16 (4 bytes)
        padding: 0 (2 or 4 bytes)
        """
        afi_bytes = self.afi.to_bytes(2, byteorder='big')
        dest_bytes = self.dest.to_bytes(4, byteorder='big')
        metric_bytes = self.metric.to_bytes(4, byteorder='big')
        entry = afi_bytes + self.PADDING_2BYTES +\
                        dest_bytes +\
                        self.PADDING_4BYTES +\
                        self.PADDING_4BYTES +\
                        metric_bytes
        return entry

    def is_valid_entry(self):
        """
        check if an entry is valid
        """
        is_valid_dest = 1 <= self.dest <= 64000
        is_valid_metric = 0 <= self.metric <= 16
        is_valid_afi = self.afi == 2
        return is_valid_dest and is_valid_metric and is_valid_afi

    def set_metric_infinite(self):
        """
        set the metric to be infinite(16)
        """
        self.metric = 16

    def increment_metric(self):
        """
        add 1 to metric
        """
        self.metric += 1
```

```python
"""
COSC364 2022-S1 Assignment: RIP routing
Authors: MENG ZHANG (71682325), ZHENG CHAO (21671773)
File: IO_formatter.py
"""
import time

def routing_table_formatter(router_id, table):
    """
    Parameters:
    table: a dictionary of routes
    {id1: Route object, id2: Route object, ...}

    Return:
    table: a formatted string which contains data of the table
    """
    # Get header
    header = table_header_formatter(router_id)
    # Get content
    content = table_content_formatter(table)
    return header + content


def talbe_border_formatter(length):
    """
    return two formatted routing table borders
    """
    border = length * '-'
    double_border = length * '='
    return border, double_border


def table_header_formatter(router_id):
    """
    return a formatted routing table header
    """
    border, double_border = talbe_border_formatter(72)
    title = f'Router {router_id:02} RIP ROUTING TABLE'
    padded_title = '|' + 21 * ' ' + title + 22 * ' ' + '|'
    labels = "|  Dest  |  Next  |  Metric  | Timeout | Garbage |  State  |"
    header = '\n' + double_border + '\n' +\
        padded_title + '\n' +\
        double_border + '\n' +\
        labels + '\n' +\
        border + '\n'
    return header


def table_content_formatter(table):
    """
    retrun formatted routing table content
    """
    border = talbe_border_formatter(72)[0]
    content = ""
    for dest, rip_route in table.items():
        next_hop = rip_route.next_hop
        metric = rip_route.metric

        timeout = rip_route.timeout
        if not timeout is None:
            timeout = round(time.time() - rip_route.timeout, 2)
            timeout_str = f'{timeout:^11.1f}'
        else:
            timeout_str = 5 * ' ' +  '-' + 5 * ' '

        garbage_collect_time = rip_route.garbage_collect_time
```

```
67          if not garbage_collect_time is None:
68              garbage_collect_time = int(time.time() - garbage_collect_time)
69              gc_str = f'{garbage_collect_time:^11.0f}'
70          else:
71              gc_str = 5 * ' ' +  '-' + 5 * ' '
72          state = rip_route.state
73          content += f'|{dest:^10}|{next_hop:^10}|{metric:^12}|'+\
74              f'{timeout_str}|{gc_str}|{state:^11}|' + '\n' +\
75              border + '\n'
76      return content
```

Plagiarism Declaration

This form needs to accompany your COSC 364 assignment submission.

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

I hereby declare that:

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

Name: MENG ZHANG

Student ID: 71682325

Signature: 

Date: 26/04/2022

Plagiarism Declaration

This form needs to accompany your COSC 364 assignment submission.

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

I hereby declare that:

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

Name: Zheng Chao

Student ID: 21671773

Signature: zheng chao

Date: 26/4/2022