COSC364 RIPv2 Assignment

Jesse Sheehan (53366509)
Will Cowper (81163265)

May 3, 2019

Contents

1 Questions **3**
1.1 Contribution . . . 3
1.2 Reflection . . . 3
1.3 Event Processing . . . 3
1.4 Testing . . . 3

2 Appendices **5**
2.1 Source Code . . . 5
2.1.1 src/__main__.py . . . 5
2.1.2 src/bencode.py . . . 6
2.1.3 src/config.py . . . 11
2.1.4 src/protocol.py . . . 14
2.1.5 src/routing_table_entry.py . . . 16
2.1.6 src/routing_table.py . . . 17
2.1.7 src/server.py . . . 20
2.1.8 src/timer.py . . . 26
2.1.9 src/utils.py . . . 29
2.2 Configuration Files . . . 31
2.2.1 configs/networks/figure1/1.conf . . . 31
2.2.2 configs/networks/figure1/2.conf . . . 31
2.2.3 configs/networks/figure1/3.conf . . . 31
2.2.4 configs/networks/figure1/4.conf . . . 31
2.2.5 configs/networks/figure1/5.conf . . . 32
2.2.6 configs/networks/figure1/6.conf . . . 32
2.2.7 configs/networks/figure1/7.conf . . . 32
2.3 Other Files . . . 33
2.3.1 tools/generate_network.py . . . 33

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 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 partial or total loss of marks, no grade being awarded and other serious consequences, including notification of the University Proctor.

Name: Will Cowper Jesse Sheehan

Student ID: 81163265 53366509

Signature: 

Date: 2-5-19 2/5/19

1 Questions

As required, the following questions have been answered:

1.1 Contribution

The contribution toward the entire project was an even split. Both partners felt as though the work they had contributed was worth 50% each.

1.2 Reflection

Some of the smaller modules in our codebase have been implemented quite well. For example, the Timer and Bencode modules have a very focused purpose and were discrete enough to be able to be doctested. We found that making use of recursion in the Bencode module prevented the complexity of its functions from becoming too high. The Timer modlue has many more features that what are being used in our project. It turned out to be too fully-featured and we could have done just as well with a stripped-back version of the same thing. We also had a good user-interface that clearly displays the current routing table. Our protocol module contains functions and classes that deal with preparing data for transmission, this includes a checksum that ensures that the data arrives intact or not at all.

The overall system design could be improved. We rewrote some modules several times in order to get it to feel as though it would be easy to work with. If we were to improve upon the current design, we could add more features, such as accepting keyboard commands while the router is running, etc. We would also spend more time planning the project and understanding the exact steps required to implement the specification.

1.3 Event Processing

Our entire program is based around a main loop that waits for incoming packets and if it doesn't receive any, it will do other things, such as updating the timers, updating the routing table, rendering the screen, etc. We use lists to ensure that our incoming packets are serviced in the order in which they arrive. When packets are processed, they may trigger updates to the routers neighbors. These updates are serviced after the periodic updates have finished being received. Once these updates have been sent to its neighbors, the router simply waits for more information to arrive.

In order to ensure the atomicity of events in our program we have made use of timer-driven functions and their timers in such a way that they do not interrupt other events. Our entire program is single-threaded so we don't need to worry about interruptions from other parts of the program.

1.4 Testing

Many of the smaller functions in the project were discrete enough that we could use doctests on them although we found that as the project grew, the complex dependencies

between the objects also grew. Once many functions in a module were working well and passing doctests, we were also able to test the module as a whole, by using many functions together and testing them. This led to us discovering that some of our functions weren't returning the correct values or weren't accepting the right parameters, etc. After getting an entire module working properly, we focussed on combining several modules together in a new module and testing that. Because of the amount of things we needed to test we didn't bother with writing any test cases. This was perhaps a poor decision in terms of testing.

Eventually we had most of the program working and had to create test configuration files for the router program to use. We found this process to be very tedious and so, wrote a program to generate these files for us. Our testing became much easier after this as we didn't have to manually write these configuration files.

2 Appendices

2.1 Source Code

2.1.1 src/__main__.py

```
#!/usr/bin/python3

"""

    __main__.py

    COSC364 RIP Assignment

    Date: 02/05/2019

    Written by:
     - Will Cowper (81163265)
     - Jesse Sheehan (53366509)

"""

import sys
import os.path

import server
import config


def print_usage():
    """
        Prints the usage of the program.
    """
    print("usage: {0} <config_filename>".format(sys.argv[0]))


def print_filename_error(filename):
    """
        Prints a filename error.
    """
    print("Error: {0} doesn't exist.".format(filename))


def print_config_error():
    """
        Prints a configuration file error.
    """
    print("Error: Couldn't read the configuration file.")


def main():
    """
        The main entry point to the program.
```

```
48      """
50      if len(sys.argv) != 2:
            print_usage()
52          return -1
54      filename = sys.argv[1]
        file = None
56      conf = None
58      # accepts config from stdin
        if filename == '--':
60          file = sys.stdin
62      # or from a file
        else:
64          if not os.path.exists(filename):
                print_filename_error(filename)
66              return -1
            else:
68              file = open(filename, "r")
70      try:
            print("Reading configuration file... ", end='')
72          conf = config.Config()
            conf.parse_file(file)
74          print("done!")
76      except:
            print_config_error()
78          return -1
80      try:
            print("Starting RIP router #" + str(conf.router_id))
82          s = server.Server(conf)
            s.start()
84
        # Ignore KeyboardInterrupts
86      except KeyboardInterrupt:
            pass
88
        # Re-raise other exceptions
90      except Exception as err:
            raise err
92
94  if __name__ == "__main__":
        main()
```

../src/__main__.py

2.1.2 src/bencode.py

```
#!/usr/bin/python3

"""

    generate_network.py

    COSC364 RIP Assignment

    Date: 02/05/2019

    Written by:
     - Will Cowper (81163265)
     - Jesse Sheehan (53366509)

    A bencoding implementation based on the official specification (https
    ://wiki.theory.org/index.php/BitTorrentSpecification#Bencoding)

"""


def bencode(value):
    """
    Test Integer Encoding:
    >>> bencode(42)
    'i42e'
    >>> bencode(0)
    'i0e'
    >>> bencode(-42)
    'i-42e'

    Test String Encoding:
    >>> bencode("spam")
    '4:spam'
    >>> bencode("i")
    '1:i'
    >>> bencode("")
    '0:'
    >>> bencode("COSC364 is the greatest course evarrrr!")
    '39:COSC364 is the greatest course evarrrr!'

    Test List Encoding:
    >>> bencode(["spam", 42])
    'l4:spami42ee'

    Test Dictionary Encoding:
    >>> bencode({"bar": "spam", "foo": 42})
    'd3:bar4:spam3:fooi42ee'

    """

    # integer encoding
    if type(value) is int:
        return "i" + str(value) + "e"
```

```
54      # string encoding
        if type(value) is str:
56          return str(len(value)) + ":" + value

58      # list encoding
        if type(value) is list:
60          return "l" + "".join(map(bencode, value)) + "e"

62      # dictionary encoding
        if type(value) is dict:
64          # TODO: keys should be in alphabetical order
            # TODO: check that the key is a string
66          return "d" + "".join([bencode(k) + bencode(v) for k, v in value.
    items()]) + "e"

68      raise ValueError(str(type(value)) +
                         " must be one of int, str, list or dict")
70

72  def bdecode(string, returnLength=False):
        """
74      >>> bdecode("i42e")
        42
76      >>> bdecode("i0e")
        0
78      >>> bdecode("i-42e")
        -42
80
        >>> bdecode("i42e", True)
82      (42, 4)
        >>> bdecode("i0e", True)
84      (0, 3)
        >>> bdecode("i-42e", True)
86      (-42, 5)

88      >>> bdecode("4:spam")
        'spam'
90      >>> bdecode("1:i")
        'i'
92      >>> bdecode("0:")
        ''
94      >>> bdecode("39:COSC364 is the greatest course evarrrr!")
        'COSC364 is the greatest course evarrrr!'
96
        >>> bdecode("4:spam", True)
98      ('spam', 6)
        >>> bdecode("1:i", True)
100     ('i', 3)
        >>> bdecode("0:", True)
102     ('', 2)
        >>> bdecode("39:COSC364 is the greatest course evarrrr!", True)
104     ('COSC364 is the greatest course evarrrr!', 42)
```

```
106     >>> bdecode("l4:spami42ee")
        ['spam', 42]
108     >>> bdecode("l4:spami42el9:more spami-42eee")
        ['spam', 42, ['more spam', -42]]
110
        >>> bdecode("l4:spami42ee", True)
112     (['spam', 42], 12)
        >>> bdecode("l4:spami42el9:more spami-42eee", True)
114     (['spam', 42, ['more spam', -42]], 30)

116     >>> bdecode("d3:bar4:spam3:fooi42ee")
        {'bar': 'spam', 'foo': 42}
118     >>> bdecode("d3:bar4:spam3:fooi42e4:listl4:spami42el9:more spami-42eeee
        ")
        {'bar': 'spam', 'foo': 42, 'list': ['spam', 42, ['more spam', -42]]}
120
        >>> bdecode("d3:bar4:spam3:fooi42ee", True)
122     ({'bar': 'spam', 'foo': 42}, 22)
        >>> bdecode("d3:bar4:spam3:fooi42e4:listl4:spami42el9:more spami-42eeee
        ", True)
124     ({'bar': 'spam', 'foo': 42, 'list': ['spam', 42, ['more spam', -42]]},
        58)

126     """

128     value = None
        length = 0
130
        # integer decoding
132     if string[0] == 'i':

134         # get the end of the integer string
            end = string.find('e')
136         if end == -1:
                raise ValueError(string[0:10] + "... is not a bencoded integer"
        )
138
            # get the integer from the string (this may throw a ValueError)
140         value = int(string[1:end])

142         # update the length to account for the entire integer string
            length = end + 1
144
        # string decoding
146     elif string[0].isnumeric():

148         # get the end of the string length
            length_end = string.find(':')
150         if length_end == -1:
                raise ValueError(string[0:10] + "... is not a bencoded string")
152
            # get the string length as an integer
```

```
154             str_length = int(string[0:length_end])
156             # get the actual string
                value = string[length_end + 1:length_end + 1 + str_length]
158             # update the length to be the length of the string including the
        string length
160             length = length_end + 1 + str_length
162         # list decoding
            elif string[0] == 'l':
164             # set the offset to 1 to account for the starting 'l'
166             offset = 1
                value = []
168             while string[offset] != 'e':
170                 # decode the inner value
172                 inner_value, inner_length = bdecode(string[offset:], True)
                    offset += inner_length
174                 # update the list
176                 value.append(inner_value)
178             # update the length to account for the closing 'e'
                length = offset + 1
180         # dictionary decoding
182         elif string[0] == 'd':
184             # set the offset to 1 to account for the starting 'd'
                offset = 1
186             # in Python >= 3.6, the dictionary implementation remembers the
188             # insertion order
                value = {}
190             while string[offset] != 'e':
192                 # decode the key
194                 inner_key, inner_length = bdecode(string[offset:], True)
                    offset += inner_length
196                 # TODO: inner_key should be a string
198                 # decode the value
                    inner_value, inner_length = bdecode(string[offset:], True)
200                 offset += inner_length
202                 # update the dictionary
                    value[inner_key] = inner_value
204             # TODO: validate that the keys are in alphabetical order
```

```
206
            # update the length to account for the closing 'e'
208         length = offset + 1

210     # return the length as well if requested
        if returnLength:
212         return value, length
        else:
214         return value


216
if __name__ == "__main__":
218     import doctest
        doctest.testmod()
```

../src/bencode.py

2.1.3 src/config.py

```
#!/usr/bin/python3
2
"""
4
    config.py
6
    COSC364 RIP Assignment
8
    Date: 02/05/2019
10
    Written by:
12   - Will Cowper (81163265)
     - Jesse Sheehan (53366509)
14
"""
16
import configparser
18 import os
import random
20


22 class Config:

24      """
            Config class used for abstracting the stored config
26      """

28      def __init__(self):
            self.router_id = 0
30          self.input_ports = []
            self.output_ports = []
32          self.periodic_update = 0
```

```
34      def parse_file(self, file):
            c = read_config_file(file)
36          self.router_id = c["routerId"]
            self.input_ports = c["inputPorts"]
38          self.output_ports = [
                OutputPort(o["outputPort"], o["cost"], o["routerId"]) for o in
    c["outputPorts"]
40          ]
            self.periodic_update = c["periodicUpdate"]
42
        def __str__(self):
44          return "Config <id={0}, input_ports={1}, output_ports={2},
    periodic_update={3:.3}s>".format(self.router_id, self.input_ports, self
    .output_ports, self.periodic_update)

46      def __repr__(self):
            return self.__str__()
48


50  class OutputPort:
        def __init__(self, port, cost, id):
52          self.router_id = id
            self.port = port
54          self.cost = cost

56      def __str__(self):
            return "OutputPort <id={0}, port={1}, cost={2}>".format(self.
    router_id, self.port, self.cost)
58
        def __repr__(self):
60          return self.__str__()


62
    def read_config_file(file):
64      """
            Parses a given file and returns a dict containing the routerID,
    input ports
66          and output ports with their cost and next hop
        """
68      #Create an instance of configparser object
        config = configparser.ConfigParser()
70      config.read_file(file)
        # dict declartion
72      router = {}
        # Reading in each section of the config
74      routerId = (config.get('DEFAULT', 'router-id'))
        inputPorts = (config.get('DEFAULT', 'input-ports'))
76      outputPorts = (config.get('DEFAULT', 'output-ports'))
        # Checks config file for periodic timer override or defaults to
    standard
78      periodicUpdate = config.get("DEFAULT", "periodic-update", fallback=3.0)

80      # Validating all parameters
```

```
    router["routerId"] = check_router_id(routerId)
    router["inputPorts"] = check_input_ports(inputPorts)
    router["outputPorts"] = check_output_ports(router, outputPorts)
    router["periodicUpdate"] = check_periodic_update(periodicUpdate)

    return router


def check_periodic_update(periodicUpdate):
    """
        Reduces the chance of collisons and other nasties by implementing a
     random wait to the periodicUpdate
    """
    return periodicUpdate + (random.random() * 2 - 1)


def check_router_id(routerId):
    """
        Takes a routerID string from the config and checks it
        Returns it back as an int if its valid
    """
    try:
        routerId = int(routerId)
    except:
        raise TypeError("RouterID must be an integer")
    if (routerId > 64000 or routerId < 1):
        raise ValueError("RouterID must be between 1 and 64000")
    return routerId


def check_input_ports(inputPorts):
    """
        Takes a string of inputports from the config
        Validates and then returns them as a list
    """
    try:
        inputPorts = [int(port.strip()) for port in inputPorts.split(',')]
    except:
        raise TypeError("Input ports should be comma seperated ints")
    for port in inputPorts:
        if (port > 64000 or port < 1024):
            raise ValueError("Port should be between 1024 and 64000")
    if len(inputPorts) != len(set(inputPorts)):
        raise ValueError("Ports should be unique")
    return inputPorts


def check_output_ports(router, outputPorts):

    """
        Takes an incomplete router dict containing a routerID and input
    ports
        Tests the routerID and input ports against a list of outputPorts
```

```
132         Returns a list of outputPorts if they are all valid.

134     """
        outportPortList = []
136     try:
            outputPorts = [port.strip() for port in outputPorts.split(',')]
138     except:
            raise TypeError(
140             "Outport ports should be comma seperated in the form PORT-COST-
        ID")
        for output in outputPorts:
142         config = {}
            output = output.split('-')
144         output = [int(i) for i in output]
            config["cost"] = output[1]
146
            if (output[0] > 64000 or output[0] < 1024):
148             raise ValueError("Port should be between 1024 and 64000")
            if output[2] == router["routerId"]:
150             raise ValueError("Output port routerID matches own routerID")
            if any(d.get('routerId', None) == output[2] for d in
        outportPortList):
152             raise ValueError("RouterID already exists in output list")
            config["routerId"] = output[2]
154         if output[0] in router["inputPorts"]:
                raise ValueError("Outport port is shared with an input port")
156         if any(d.get('outputPort', None) == output[0] for d in
        outportPortList):
                raise ValueError("OutputPort already in use")
158         config["outputPort"] = output[0]
            outportPortList.append(config)
160
        return outportPortList
162
    def open_config_file(filePath):
164     """
            Takes a filepath as argument, validates it and returns a Config
        object
166     """
        file = open(filePath, 'r')
168     if file.mode == 'r':
            config = Config()
170         config.parse_file(file)
        else:
172         print("Error opening file")
        return config
```

../src/config.py

2.1.4 src/protocol.py

```
#/usr/bin/python3
```

```
2
"""

4
    protocol.py

6
    COSC364 RIP Assignment

8
    Date: 02/05/2019

10
    Written by:
12   - Will Cowper (81163265)
     - Jesse Sheehan (53366509)

14
"""

16
import bencode
18 import binascii

20 __encoding = "utf-8"


22
def encode(data):
24     """
           Encodes the raw data, including a checksum.
26     """
       body = bencode.bencode(data).encode(__encoding)
28     crc = binascii.crc32(body)
       return crc.to_bytes(4, "big") + body
30


32 def decode(data):
       """
34         Decodes raw data, checks the validity and returns the dictionary
       containing the data.
           Returns None if the data is invalid.
36     """
       try:
38         # get the CRC32 code
           crc = int.from_bytes(data[:4], "big")
40
           # get the body
42         body = data[4:]

44         # return None if the checksum is incorrect
           if crc != binascii.crc32(body):
46             return None

48         # return the decoded data if the checksum is correct
           else:
50             return bencode.bdecode(body.decode(__encoding))
       except:
52         return None
```

```
54 class Packet:
55     """
56         A Packet is used to send and receive updates from other RIP routers
       .
57     """
58
59     def __init__(self, link_cost = -1, routes = []):
60         """
61             Creates a new Packet.
62         """
63         self.link_cost = link_cost
64         self.routes = routes
65
66     def from_data(self, data):
67         """
68             Sets the packet information from some raw data.
69             Returns True if successful.
70         """
71         d = decode(data)
72
73         if d is not None:
74             self.link_cost = d["link-cost"]
75             self.routes = d["routes"]
76             return True
77         else:
78             return False
79
80     def to_data(self):
81         """
82             Returns the raw data to be sent.
83         """
84         return encode({
85             "link-cost": self.link_cost,
86             "routes": self.routes
87         })
```

../src/protocol.py

2.1.5 src/routing_table_entry.py

```
1  #!/usr/bin/python3
2  """
3
4      routing_table_entry.py
5
6      COSC364 RIP Assignment
7
8      Date: 02/05/2019
9
10     Written by:
11      - Will Cowper (81163265)
12
```

```
     - Jesse Sheehan (53366509)

"""

class RoutingTableEntry:
    """
    A RoutingTableEntry represents a RIP entry that resides in the routing
    table
    """

    def __init__(self, destination, nextHop, cost):
        self.destination = destination
        self.nextHop = nextHop
        self.cost = cost
        self.age = 0.0
        self.garbage = False

    def __str__(self):
        return "RouteTableEntry <destination={0}, nextHop={1}, cost={2},
    age={3}, garbage={4}>".format(self.destination, self.nextHop, self.cost
    , round(self.age, 2), self.garbage)

    def __repr__(self):
        return self.__str__()
```

../src/routing_table_entry.py

2.1.6 src/routing_table.py

```
#!/usr/bin/python3

"""

    routing_table.py

    COSC364 RIP Assignment

    Date: 02/05/2019

    Written by:
     - Will Cowper (81163265)
     - Jesse Sheehan (53366509)

"""


import os
from routing_table_entry import RoutingTableEntry
import config


class RoutingTable:
```

```
24      """
            The RoutingTable represents the list of RoutingTableEntries for a
        router.
26      """

28      def __init__(self, config):
            """
30              Creates a new RoutingTable based on the Config.
            """
32          self.__routes = []
            self.routerID = config.router_id
34
        def add_entry(self, destination, nextHop, totalCost):
36          """
                Adds a new RoutingTableEntry to the RoutingTable.
38          """
            route = RoutingTableEntry(destination, nextHop, totalCost)
40          self.__routes.append(route)

42      def set_garbage(self, routerID, isGarbage):
            """
44              Sets the garbage flag of the entry.
            """
46          index = self.get_index(routerID)
            self.__routes[index].garbage = isGarbage
48          self.reset_age(routerID)
            if isGarbage:
50              self.set_cost(routerID, 16)

52      def reset_age(self, routerID):
            """
54              Resets the age of the entry to 0.
            """
56          index = self.get_index(routerID)
            self.__routes[index].age = 0.0
58
        def increment_age(self, time):
60          """
                Increments the age of all entries in the RoutingTable.
62          """
            for entry in self.__routes:
64              if entry.destination != self.routerID:
                    entry.age += time
66
        def delete_entry(self, routerID):
68          """
                Deletes an entry with the specific routerID from the
        RoutingTable.
70          """
            index = self.get_index(routerID)
72          del self.__routes[index]

74      def get_index(self, routerID):
```

```
        """
            Gets the index of the entry with the routerID. Returns -1 if not found.
        """
        for i, route in enumerate(self.__routes):
            if route.destination == routerID:
                return i
        return -1  # Not found

    def set_cost(self, routerID, cost):
        """
            Sets the cost of the entry.
        """
        index = self.get_index(routerID)
        self.__routes[index].cost = cost

    def set_next_hop(self, routerID, nextHop):
        """
            Sets the next hop of the entry.
        """
        index = self.get_index(routerID)
        self.__routes[index].nextHop = nextHop

    def update(self, triggered_update_callback):
        """
            Performs house-keeping on the entries.
            The `triggered_update_callback` is for performing triggered updates.
        """
        remove_routes = []
        triggered_routes = []

        for route in self.__routes:
            if route.age > 10 and not route.garbage:
                self.set_garbage(route.destination, True)
                triggered_routes.append(route)

            if route.age > 20 and route.garbage:
                remove_routes.append(route)

        if len(triggered_routes) != 0:
            triggered_update_callback(triggered_routes)

        for route in remove_routes:
            self.__routes.remove(route)

    def __getitem__(self, routerId):
        """
            Gets the entry with the given routerId.
        """
        index = self.get_index(routerId)
        if index != -1:
            return self.__routes[index]
```

```
126             return None
128         def __iter__(self):
                """
130                 Returns the iterator of the routes.
                """
132             return iter(self.__routes)
134         def __len__(self):
                """
136                 Returns the number of routes this RoutingTable has.
                """
138             return len(self.__routes)
140         def __str__(self):
                """
142                 Returns a human-readable RoutingTable that can be printed to
        the terminal.
                """
144             s = [
                    "
        +-------------+-------------+-------------+-------------+-------------+",
146                 "| Dest.       | Next Hop    | Total Cost | Age         | Garbage?
           |",
                    "
        +-------------+-------------+-------------+-------------+-------------+"
148             ]
                for route in self.__routes:
150                 s.append("| {0:<10} | {1:<10} | {2:<10} | {3:<10} | {4:<10} |".
        format(
                        route.destination, route.nextHop, route.cost, round(route.
        age, 2), route.garbage))
152             s.append("
        +-------------+-------------+-------------+-------------+-------------+")
                return os.linesep.join(s)
154
156 # runs a simple test
    if __name__ == "__main__":
158     current_directory = os.path.dirname(__file__)
        parent_directory = os.path.split(current_directory)[0]
160     file_path = os.path.join(parent_directory, 'configs/good/01.conf')
        config = config.open_config_file(file_path)
162     r = RoutingTable(config)
        print(r)
```

../src/routing_table.py

2.1.7 src/server.py

```
#!/usr/bin/python3
2
```

```
"""

    server.py

    COSC364 RIP Assignment

    Date: 02/05/2019

    Written by:
     - Will Cowper (81163265)
     - Jesse Sheehan (53366509)

"""

import socket
import select
import time

import timer
import routing_table
import routing_table_entry
import protocol
import utils
import bencode


def create_input_socket(port, host='localhost'):
    """
        Creates a new UDP socket.
    """
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.bind((host, port))
    return sock


class Server:

    def __init__(self, config):
        """
            Creates a new server with a configuration.
        """
        self.rt = routing_table.RoutingTable(config)
        self.config = config
        self.input_ports = []
        self.periodic_timer = None
        self.loglines = []

    def print_display(self):
        """
            Displays useful information for the user.
        """

        # clear the screen
```

```
56          utils.clear_terminal()

58          # print info about this router
            print("RIP Router #" + str(self.config.router_id))
60          print("Uptime: {0} seconds".format(
                round(self.periodic_timer.getElapsed())))
62
            # print the routing table
64          print(self.rt)

66          # print other info
            print("Press Ctrl+C to quit")
68
        def process_periodic_update(self, dt):
70          """
                Called when the periodic timer is triggered.
72          """
            # send destination, next hop and total cost of each routing entry
    to each input port
74          sock = self.input_ports[0]

76          for output_port in self.config.output_ports:

78              # add self to the routes
                routes = [{
80                      "destination": self.config.router_id,
                        "cost": 0,
82                      "next-hop": self.config.router_id
                    }]
84
                if len(self.rt) == 0:
86                  self.log("advertising self to " + str(output_port.router_id
    ))

88              for route in self.rt:

90                  cost = route.cost
                    destination = route.destination
92
                    # poison reverse by setting cost to 16 when announcing
    routes back from where they were learned
94                  if self.rt[destination].nextHop == output_port.router_id:
                        cost = 16
96
                    routes.append({
98                      "destination": destination,
                        "cost": cost,
100                     "next-hop": self.config.router_id
                    })
102
                packet = protocol.Packet(output_port.cost, routes)
104             sock.sendto(packet.to_data(), ('localhost', output_port.port))
```

```
106     def log(self, message):
            """
108             Writes to the information log (for a maximum of 10 lines).
            """
110         self.loglines.append(message)
            while len(self.loglines) > 10:
112             self.loglines = self.loglines[1:]

114     def process_incoming_data(self, addr, data):
            """
116             Called when incoming data is received. The data returned from
        this function is sent back through the socket. If None is returned,
        nothing will be sent.
            """
118
            triggered_updates = []
120         packet = protocol.Packet()

122         if not packet.from_data(data):
                self.log("invalid packet hash")
124             return

126         for route in packet.routes:

128             route_destination = route["destination"]
                route_cost = route["cost"]
130             route_next_hop = route["next-hop"]

132             destination_entry = self.rt[route_destination]

134             is_destination_in_table = destination_entry is not None

136             is_destination_unreachable = route_cost >= 16

138             total_destination_cost = route_cost + packet.link_cost

140             if route_destination == self.config.router_id:
                    continue
142
                # New valid route
144             if not is_destination_in_table and not
        is_destination_unreachable:

146                 # put the destination in the table
                    self.rt.add_entry(route_destination, route_next_hop,
        total_destination_cost)
148                 self.log("added new router " + str(route_destination) + "
        via " + str(route_next_hop) + " with a cost of " + str(
        total_destination_cost))

150             # Route already exists in table
                elif is_destination_in_table:
152
```

```
                is_destination_garbage = destination_entry.garbage

                # Check for a better route.
                if total_destination_cost < destination_entry.cost:
                    self.rt.set_cost(route_destination, total_destination_cost)
                    self.rt.set_garbage(route_destination, False)
                    self.rt.set_next_hop(route_destination, route_next_hop)

                    self.log("found new route to " + str(route_destination) + " via " + str(route_next_hop) + " with a cost of " + str(total_destination_cost))

                # Check for worse route from the same hop
                elif route_next_hop == destination_entry.nextHop and total_destination_cost > destination_entry.cost:
                    if is_destination_unreachable:
                        # garbage it if we haven't seen it before, otherwise ignore it
                        if not is_destination_garbage:
                            self.rt.set_garbage(route_destination, True)
                            triggered_updates.append(destination_entry)
                            self.log("processed a triggered update from " + str(packet.routes[0]["next-hop"]) + " marked " + str(route_destination) + " as garbage")

                    # We got a worse route from the samehop but its not infinite. As a neighbour we MUST update to the higher cost.
                    else:
                        self.rt.set_cost(route_destination, total_destination_cost)
                        self.rt.reset_age(route_destination)

                # Check for worse route from a different hop and ignore it
                elif total_destination_cost > destination_entry.cost:
                    #self.log("Worse route to " + str(route_destination) + " ignoring it")
                    continue

                # Check for same route and keep it alive
                elif route_next_hop == destination_entry.nextHop and total_destination_cost == destination_entry.cost:
                    # We dont want to keep alive infinite weight routes
                    if not is_destination_garbage:
                        self.rt.reset_age(route_destination)

        if len(triggered_updates) > 0:
            self.log("sending triggered updates")
            self.process_triggered_updates(triggered_updates)

        return None

    def process_triggered_updates(self, routes):
```

```
        """
196             Processes the triggered updates.
        """
198         sock = self.input_ports[0]
            for output_port in self.config.output_ports:
200
                packet_routes = [{
202                     "destination": route.destination,
                        "cost": 16,
204                     "next-hop": self.config.router_id
                    } for route in routes]
206
                p = protocol.Packet(output_port.cost, packet_routes)
208             sock.sendto(p.to_data(), ('localhost', output_port.port))

210     def start(self):
            """
212             Starts the server.
            """
214
            # set up the input ports
216         self.input_ports = list(
                map(create_input_socket, self.config.input_ports))
218
            # start the periodic timer
220         self.periodic_timer = timer.Timer(
                self.config.periodic_update, self.process_periodic_update)
222         self.periodic_timer.start()
            self.periodic_timer.trigger()
224
            # only block for a second at a time
226         blocking_time =1

228         loop_time = time.time()

230         while self.input_ports:
                readable, _writable, exceptional = select.select(
232                 self.input_ports, [], self.input_ports, blocking_time)

234             # increment the age
                dt = time.time() - loop_time
236             self.rt.increment_age(dt)

238             # redisplay the screen
                self.print_display()
240
                # update the timer, may call process_periodic_update
242             self.periodic_timer.update()

244             # may call process_triggered_updates
                self.rt.update(self.process_triggered_updates)
246
                # display the information log
```

```
248             print("")
                print("Information Log:")
250             for line in self.loglines:
                    print(" ", line)
252
                # iterate through all sockets that have data waiting on them
254             for sock in readable:
                    data, addr = sock.recvfrom(4096)
256                 resp = self.process_incoming_data(addr, data)

258                 if resp is not None:
                        sock.sendto(resp, addr)
260
                # removes a socket from the input list if it raised an error
262             for sock in exceptional:
                    if sock in self.input_ports:
264                     self.input_ports.remove(sock)
                        sock.close()
266                 raise Exception("A socket raised an error")

268             # update the loop time
                loop_time = time.time()
270
if __name__ == "__main__":
272     pass
```

../src/server.py

2.1.8 src/timer.py

```
#!/usr/bin/python3
2
"""
4
    timer.py
6
    COSC364 RIP Assignment
8
    Date: 02/05/2019
10
    Written by:
12   - Will Cowper (81163265)
     - Jesse Sheehan (53366509)
14
"""
16
import time
18
class Timer:
20
    def __init__(self, period, callback):
22      """
```

```
            Creates a new Timer with a period and a callback.
        """
        self.__period = period
        self.__callback = callback
        self.__started = False
        self.__startedTime = 0
        self.__paused = False
        self.__pausedTime = 0
        self.__updateTime = 0

    def start(self):
        """
            Starts the timer.
        """
        if not self.__started:
            t = time.time()
            self.__started = True
            self.__startedTime = t
            self.__paused = False
            self.__pausedTime = 0
            self.__updateTime = t

    def stop(self):
        """
            Stops the timer.
        """
        if self.__started:
            self.__started = False
            self.__startedTime = 0
            self.__paused = False
            self.__pausedTime = 0
            self.__updateTime = 0

    def reset(self):
        """
            Resets the timer.
        """
        if self.__started:
            self.stop()
            self.start()

    def pause(self):
        """
            Pauses the timer.
        """
        if self.__started and not self.__paused:
            self.__paused = True
            self.__pausedTime = time.time() - self.__startedTime
            self.__startedTime = 0

    def resume(self):
        """
            Resumes the timer.
```

```
76          """
            if self.__started and self.__paused:
78              self.__startedTime = time.time() - self.__pausedTime
                self.__paused = False
80              self.__pausedTime = 0

82      def update(self):
            """
84              Updates the timer. May call its callback.
            """
86          if self.__started and not self.__paused:
                t = time.time()
88              dt = t - self.__updateTime
                if dt > self.__period:
90                  self.__updateTime = t
                    self.__callback(dt)
92
        def trigger(self):
94          """
                Forcefully call the callback.
96          """
            if self.__started and not self.__paused:
98              t = time.time()
                dt = t - self.__updateTime
100             self.__updateTime = t
                self.__callback(dt)
102
        def getElapsed(self):
104         """
                Returns the time elapsed in seconds.
106         """
            if self.__started:
108             if self.__paused:
                    return self.__pausedTime
110             else:
                    return time.time() - self.__startedTime
112         return 0.0

114     def isStarted(self):
            """
116             Returns True if the timer has been started.

118         >>> t = Timer(10, None)
            >>> t.isStarted()
120         False
            >>> t.start()
122         >>> t.isStarted()
            True
124         >>> t.stop()
            >>> t.isStarted()
126         False
            """
128         return self.__started
```

```
130     def isPaused(self):
            """
132             Returns True if the timer has been paused.

134         >>> t = Timer(10, None)
            >>> t.isPaused()
136         False
            >>> t.start()
138         >>> t.isPaused()
            False
140         >>> t.pause()
            >>> t.isPaused()
142         True
            >>> t.resume()
144         >>> t.isPaused()
            False
146         >>> t.stop()
            >>> t.isPaused()
148         False
            >>> t.start()
150         >>> t.pause()
            >>> t.isPaused()
152         True
            >>> t.stop()
154         >>> t.isPaused()
            False
156         """
            return self.__paused and self.__started
158
        def __str__(self):
160         """
                Returns a string representation of the timer.
162         """
            return "Timer <period={0:.3}s, started={1}, paused={2}, elapsed
        ={3:.3}s>".format(self.__period, self.__started, self.__paused, self.
        getElapsed())
164
        def __repr__(self):
166         """
                Returns a string representation of the timer.
168         """
            return self.__str__()
170
# run doctests
172 if __name__ == "__main__":
        import doctest
174     doctest.testmod()
```

../src/timer.py

2.1.9 src/utils.py

```
#!/usr/bin/python3

"""

    utils.py

    COSC364 RIP Assignment

    Date: 02/05/2019

    Written by:
     - Will Cowper (81163265)
     - Jesse Sheehan (53366509)

"""

import os

def clear_terminal():
    """
        Clears the terminal based on the type of operating system.
    """

    # the terminal clear command for linux
    if os.name == "posix":
        os.system("clear")

    # the console cls command for windows
    elif os.name == "nt":
        os.system("cls")

    # otherwise, just print 25 newlines
    else:
        for _ in range(25):
            print("")
```

../src/utils.py

2.2 Configuration Files

2.2.1 configs/networks/figure1/1.conf

```
; configs/networks/figure1/1.conf
; created with tools/generate_network.py

[DEFAULT]
router-id = 1
input-ports = 55501, 55503, 55505
output-ports = 55500-1-2, 55502-5-6, 55504-8-7
```

../configs/networks/figure1/1.conf

2.2.2 configs/networks/figure1/2.conf

```
; configs/networks/figure1/2.conf
; created with tools/generate_network.py

[DEFAULT]
router-id = 2
input-ports = 55500, 55507
output-ports = 55501-1-1, 55506-3-3
```

../configs/networks/figure1/2.conf

2.2.3 configs/networks/figure1/3.conf

```
; configs/networks/figure1/3.conf
; created with tools/generate_network.py

[DEFAULT]
router-id = 3
input-ports = 55506, 55509
output-ports = 55507-3-2, 55508-4-4
```

../configs/networks/figure1/3.conf

2.2.4 configs/networks/figure1/4.conf

```
; configs/networks/figure1/4.conf
; created with tools/generate_network.py

[DEFAULT]
router-id = 4
input-ports = 55508, 55511, 55513
output-ports = 55509-4-3, 55510-2-5, 55512-6-7
```

../configs/networks/figure1/4.conf

2.2.5 configs/networks/figure1/5.conf

```
; configs/networks/figure1/5.conf
; created with tools/generate_network.py

[DEFAULT]
router-id = 5
input-ports = 55510, 55515
output-ports = 55511-2-4, 55514-1-6
```

../configs/networks/figure1/5.conf

2.2.6 configs/networks/figure1/6.conf

```
; configs/networks/figure1/6.conf
; created with tools/generate_network.py

[DEFAULT]
router-id = 6
input-ports = 55502, 55514
output-ports = 55503-5-1, 55515-1-5
```

../configs/networks/figure1/6.conf

2.2.7 configs/networks/figure1/7.conf

```
; configs/networks/figure1/7.conf
; created with tools/generate_network.py

[DEFAULT]
router-id = 7
input-ports = 55504, 55512
output-ports = 55505-8-1, 55513-6-4
```

../configs/networks/figure1/7.conf

2.3 Other Files

2.3.1 tools/generate_network.py

The following file will interactively prompt the user for information about a network. It will then create all the necessary configuration files for the network to run.

```
#!/usr/bin/python3
"""

    generate_network.py

    COSC364 RIP Assignment

    Date: 02/05/2019

    Written by:
     - Will Cowper (81163265)
     - Jesse Sheehan (53366509)

"""

import os
import sys


def get_network_name():
    network_name = None
    while network_name is None:
        try:
            network_name = input("Enter network name: ")
            network_name = network_name.strip()
            if not network_name.isalnum():
                print("Network name must be alpha-numeric")
                network_name = None
        except:
            print("ASD")
            return None
    return network_name


def get_router_ids():
    router_ids = []
    while len(router_ids) == 0:
        try:
            line = input("Enter router ids seperated by spaces: ")
            router_ids = [id for id in line.strip().split(" ")]
            is_valid = True
            for id in router_ids:
                if not id.isalnum():
                    is_valid = False
                    break
```

```
48              if not is_valid:
                    print("All ids must be alpha-numeric")
50                  router_ids = None
            except:
52              return None
        return router_ids
54


56  def get_link_cost(fromId, toId):
        link_cost = None
58      while link_cost is None:
            try:
60              line = input("Enter link cost between routers '" +
                             str(fromId) + "' and '" + str(toId) + "': ")
62              line = line.strip()
                if not line.isnumeric() or int(line) < 0:
64                  print("Link cost must be a positive integer (or 0 for
    infinity)")
                else:
66                  link_cost = int(line)
            except Exception as e:
68              print(e)
                return None
70      return link_cost


72
    def main():
74      network_name = get_network_name()
        if network_name is None:
76          return

78      router_ids = get_router_ids()
        if router_ids is None:
80          return

82      configs = {}
        port_number_max = 55500
84      for index, fromId in enumerate(router_ids):
            for toId in router_ids[index + 1:]:
86              link_cost = get_link_cost(fromId, toId)
                if link_cost is None:
88                  return

90              if link_cost == 0:
                    continue
92
                to_port_number = port_number_max
94              port_number_max += 1
                from_port_number = port_number_max
96              port_number_max += 1

98              if fromId not in configs:
                    configs[fromId] = {"output-ports": [],
```

```
                                        "input-ports": [], "router-id": fromId}
            configs[fromId]["output-ports"].append(
                (to_port_number, link_cost, toId))
            configs[fromId]["input-ports"].append(from_port_number)

            if toId not in configs:
                configs[toId] = {"output-ports": [],
                                 "input-ports": [], "router-id": toId}
            configs[toId]["output-ports"].append(
                (from_port_number, link_cost, fromId))
            configs[toId]["input-ports"].append(to_port_number)

    # assign port numbers
    root_path = os.path.join("configs", "networks", network_name)
    if not os.path.exists(root_path):
        os.mkdir(root_path)
        print("Created directory", root_path)

    for key in configs:
        config = configs[key]
        filename = os.path.join(root_path, config["router-id"] + ".conf")
        with open(filename, "w") as f:
            f.write("; " + filename + "\n")
            f.write("; created with tools/generate_network.py\n")
            f.write("\n")
            f.write("[DEFAULT]\n")
            f.write("router-id = " + str(config["router-id"]) + "\n")
            f.write("input-ports = " + ", ".join([str(x)
                                                  for x in config["input-
    ports"]]) + "\n")
            f.write("output-ports = " + ", ".join([str(x[0]) + "-" + str(x
    [1]) + "-" + str(x[2])
                                                   for x in config["output-
    ports"]]) + "\n")
            f.write("\n")
            print("Created", filename)

    # print("Creating ", network_name, "with", configs)


if __name__ == "__main__":
    main()
```

../tools/generate_network.py