

COSC364 RIPv2 Assignment

Jesse Sheehan (53366509)

Will Cowper (81163265)

May 3, 2019

Contents

1	Questions	4
1.1	Contribution	4
1.2	Reflection	4
1.3	Event Processing	4
1.4	Testing	5
2	Appendices	6
2.1	Source Code	6
2.1.1	src/__main__.py	6
2.1.2	src/bencode.py	7
2.1.3	src/config.py	12
2.1.4	src/protocol.py	15
2.1.5	src/routing_table_entry.py	17
2.1.6	src/routing_table.py	18
2.1.7	src/server.py	22
2.1.8	src/timer.py	28
2.1.9	src/utils.py	31
2.2	Configuration Files	33
2.2.1	configs/networks/figure1/1.conf	33
2.2.2	configs/networks/figure1/2.conf	33
2.2.3	configs/networks/figure1/3.conf	33
2.2.4	configs/networks/figure1/4.conf	33
2.2.5	configs/networks/figure1/5.conf	34
2.2.6	configs/networks/figure1/6.conf	34
2.2.7	configs/networks/figure1/7.conf	34
2.2.8	configs/networks/six-node/01.conf	34
2.2.9	configs/networks/six-node/02.conf	35
2.2.10	configs/networks/six-node/03.conf	35
2.2.11	configs/networks/six-node/04.conf	35
2.2.12	configs/networks/six-node/05.conf	35

2.2.13	configs/networks/six-node/06.conf	36
2.3	Other Files	37
2.3.1	tools/generate_network.py	37

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 53366500

Signature:

Date:

2-5-19 2/5/19

1 Questions

The configuration files of the example network can be found in sections 2.2.1 to 2.2.7. The following questions have been answered:

1.1 Contribution

The contribution toward the entire project was an even split. We both felt as though the work we 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 reduced the complexity that would have otherwise occurred. The Timer module has many features that we didn't end up using but could be useful in the future if we decided to continue developing this project. We also had a clean user-interface that clearly displays the current routing table and some other information about the router. Our protocol adds a CRC32 checksum to the data being sent, this is checked when the packet is received and dropped if it is incorrect. This protects our routers from receiving garbled packets.

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. We would also spend more time planning the project and understanding the exact steps required to implement the specification. Our current solution only receives at most 4096 bytes of data when reading a packet, this means that we can only send a maximum of 81 full router table entries in a packet. This could become an issue if we had a router with more than 80 directly connected neighbors but could be overcome by sending another packet with the remaining entries.

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. We created many configuration files for testing our config module. Some configuration files were well-formed and some were not. This allowed us to make sure that our config module worked as expected.

Once we had most of the program working we had to create test configuration files for entire networks. We found this process to be very tedious and so, wrote a program to generate these files for us (see section 2.3.1). Our testing became much easier after this as we didn't have to manually write these configuration files.

When testing an entire network, we would manually calculate the best routes by hand using the Bellman-Ford algorithm. We would then run the program and check that the routing table for each router converged to what we expected. We also tested our hypotheses for when an individual router was brought down using the same method. This would quite often reveal issues or bugs in our implementation that we could fix.

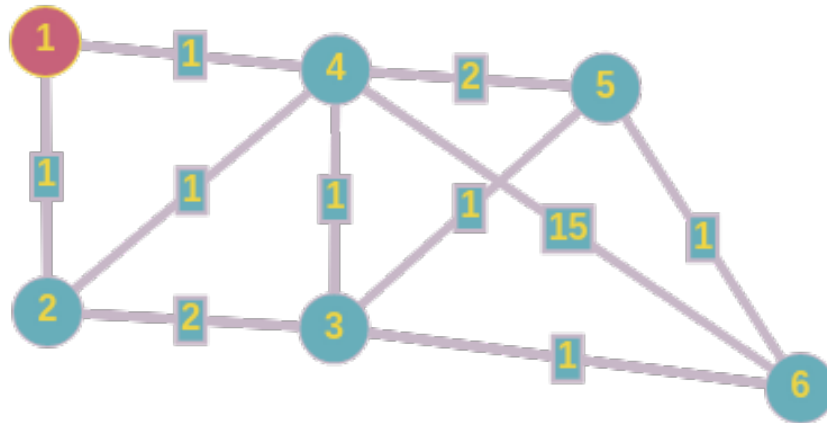


Figure 1: The "six-node" network as described in sections 2.2.8 to 2.2.13.

2 Appendices

2.1 Source Code

2.1.1 src/__main__.py

```
#!/usr/bin/python3
2
3 """
4
5     __main__.py
6
7     COSC364 RIP Assignment
8
9     Date: 02/05/2019
10
11     Written by:
12     - Will Cowper (81163265)
13     - Jesse Sheehan (53366509)
14 """
15
16 import sys
17 import os.path
18
19 import server
20 import config
21
22
23 def print_usage():
24     """
25     Prints the usage of the program.
26     """
27     print("usage: {0} <config_filename>".format(sys.argv[0]))
28
29
30 def print_filename_error(filename):
31     """
32     Prints a filename error.
33     """
34     print("Error: {0} doesn't exist.".format(filename))
35
36
37 def print_config_error():
38     """
39     Prints a configuration file error.
40     """
41     print("Error: Couldn't read the configuration file.")
42
43
44 def main():
45     """
46     The main entry point to the program.
47     """
```

```

48     """
50     if len(sys.argv) != 2:
51         print_usage()
52         return -1
53
54     filename = sys.argv[1]
55     file = None
56     conf = None
57
58     # accepts config from stdin
59     if filename == '-':
60         file = sys.stdin
61
62     # or from a file
63     else:
64         if not os.path.exists(filename):
65             print_filename_error(filename)
66             return -1
67         else:
68             file = open(filename, "r")
69
70     try:
71         print("Reading configuration file ... ", end='')
72         conf = config.Config()
73         conf.parse_file(file)
74         print("done!")
75
76     except:
77         print_config_error()
78         return -1
79
80     try:
81         print("Starting RIP router #" + str(conf.router_id))
82         s = server.Server(conf)
83         s.start()
84
85     # Ignore KeyboardInterrupts
86     except KeyboardInterrupt:
87         pass
88
89     # Re-raise other exceptions
90     except Exception as err:
91         raise err
92
93
94 if __name__ == "__main__":
95     main()

```

../src/_main_.py

2.1.2 src/bencode.py

```
#!/usr/bin/python3
2
3 """
4
5     generate_network.py
6
7     COSC364 RIP Assignment
8
9     Date: 02/05/2019
10
11     Written by:
12     - Will Cowper (81163265)
13     - Jesse Sheehan (53366509)
14
15     A bencoding implementation based on the official specification (https://wiki.theory.org/index.php/BitTorrentSpecification#Bencoding)
16 """
17
18
19
20 def bencode(value):
21     """
22     Test Integer Encoding:
23     >>> bencode(42)
24     'i42e'
25     >>> bencode(0)
26     'i0e'
27     >>> bencode(-42)
28     'i-42e'
29
30     Test String Encoding:
31     >>> bencode("spam")
32     '4:spam'
33     >>> bencode("i")
34     '1:i'
35     >>> bencode("")
36     '0:'
37     >>> bencode("COSC364 is the greatest course evaaaa!")
38     '39:COSC364 is the greatest course evaaaa!'
39
40     Test List Encoding:
41     >>> bencode(["spam", 42])
42     'l4:spami42ee'
43
44     Test Dictionary Encoding:
45     >>> bencode({"bar": "spam", "foo": 42})
46     'd3:bar4:spam3:fooi42ee'
47
48 """
49
50 # integer encoding
51 if type(value) is int:
52     return "i" + str(value) + "e"
```



```

54 # string encoding
55 if type(value) is str:
56     return str(len(value)) + ":" + value
57
58 # list encoding
59 if type(value) is list:
60     return "l" + "".join(map(bencode, value)) + "e"
61
62 # dictionary encoding
63 if type(value) is dict:
64     # TODO: keys should be in alphabetical order
65     # TODO: check that the key is a string
66     return "d" + "".join([bencode(k) + bencode(v) for k, v in value.items()]) + "e"
67
68 raise ValueError(str(type(value)) +
69                  " must be one of int, str, list or dict")
70
71
72 def bdecode(string, returnLength=False):
73     """
74     >>> bdecode("i42e")
75     42
76     >>> bdecode("i0e")
77     0
78     >>> bdecode("i-42e")
79     -42
80
81     >>> bdecode("i42e", True)
82     (42, 4)
83     >>> bdecode("i0e", True)
84     (0, 3)
85     >>> bdecode("i-42e", True)
86     (-42, 5)
87
88     >>> bdecode("4:spam")
89     'spam'
90     >>> bdecode("1:i")
91     'i'
92     >>> bdecode("0:")
93     ''
94     >>> bdecode("39:COSC364 is the greatest course evaaaa!")
95     'COSC364 is the greatest course evaaaa!'
96
97     >>> bdecode("4:spam", True)
98     ('spam', 6)
99     >>> bdecode("1:i", True)
100    ('i', 3)
101     >>> bdecode("0:", True)
102    ('', 2)
103     >>> bdecode("39:COSC364 is the greatest course evaaaa!", True)
104    ('COSC364 is the greatest course evaaaa!', 42)

```

```

106 >>> bdecode("l4:spami42ee")
    ['spam', 42]
108 >>> bdecode("l4:spami42el9:more spam-42eee")
    ['spam', 42, ['more spam', -42]]
110
112 >>> bdecode("l4:spami42ee", True)
    (['spam', 42], 12)
114 >>> bdecode("l4:spami42el9:more spam-42eee", True)
    (['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 spam-42eeee")
    {'bar': 'spam', 'foo': 42, 'list': ['spam', 42, ['more spam', -42]]}
120
122 >>> bdecode("d3:bar4:spam3:fooi42ee", True)
    ({'bar': 'spam', 'foo': 42}, 22)
124 >>> bdecode("d3:bar4:spam3:fooi42e4:listl4:spami42el9:more spam-42eeee", True)
    ({'bar': 'spam', 'foo': 42, 'list': ['spam', 42, ['more spam', -42]]}, 58)

126 """

128 value = None
    length = 0
130
132 # integer decoding
    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     )

140     # get the integer from the string (this may throw a ValueError)
        value = int(string[1:end])

142     # update the length to account for the entire integer string
        length = end + 1
144
146 # string decoding
    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
157         value = string[length_end + 1:length_end + 1 + str_length]
158
159         # update the length to be the length of the string including the
160         # string length
161         length = length_end + 1 + str_length
162
163     # list decoding
164     elif string[0] == 'l':
165
166         # set the offset to 1 to account for the starting 'l'
167         offset = 1
168         value = []
169
170         while string[offset] != 'e':
171
172             # decode the inner value
173             inner_value, inner_length = bdecode(string[offset:], True)
174             offset += inner_length
175
176             # update the list
177             value.append(inner_value)
178
179         # update the length to account for the closing 'e'
180         length = offset + 1
181
182     # dictionary decoding
183     elif string[0] == 'd':
184
185         # set the offset to 1 to account for the starting 'd'
186         offset = 1
187
188         # in Python >= 3.6, the dictionary implementation remembers the
189         # insertion order
190         value = {}
191
192         while string[offset] != 'e':
193
194             # decode the key
195             inner_key, inner_length = bdecode(string[offset:], True)
196             offset += inner_length
197             # TODO: inner_key should be a string
198
199             # decode the value
200             inner_value, inner_length = bdecode(string[offset:], True)
201             offset += inner_length
202
203             # update the dictionary
204             value[inner_key] = inner_value
205
206     # TODO: validate that the keys are in alphabetical order
```

```
206         # update the length to account for the closing 'e'
207         length = offset + 1
208
209     # return the length as well if requested
210     if returnLength:
211         return value, length
212     else:
213         return value
214
215
216 if __name__ == "__main__":
217     import doctest
218     doctest.testmod()
```

../src/bencode.py

2.1.3 src/config.py

```
#!/usr/bin/python3
2
"""
4
6     config.py
8
10    COSC364 RIP Assignment
12
14    Date: 02/05/2019
16
18    Written by:
19    - Will Cowper (81163265)
20    - Jesse Sheehan (53366509)
21
22    """
23
24    import configparser
25    import os
26    import random
27
28    class Config:
29
30        """
31        Config class used for abstracting the stored config
32        """
33
34        def __init__(self):
35            self.router_id = 0
36            self.input_ports = []
37            self.output_ports = []
38            self.periodic_update = 0
```

```

34     def parse_file(self, file):
35         c = read_config_file(file)
36         self.router_id = c["routerId"]
37         self.input_ports = c["inputPorts"]
38         self.output_ports = [
39             OutputPort(o["outputPort"], o["cost"], o["routerId"]) for o in
c["outputPorts"]
40         ]
41         self.periodic_update = c["periodicUpdate"]
42
43     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)
45
46     def __repr__(self):
47         return self.__str__()
48
49
50 class OutputPort:
51     def __init__(self, port, cost, id):
52         self.router_id = id
53         self.port = port
54         self.cost = cost
55
56     def __str__(self):
57         return "OutputPort <id={0}, port={1}, cost={2}>".format(self.
router_id, self.port, self.cost)
58
59     def __repr__(self):
60         return self.__str__()
61
62
63 def read_config_file(file):
64     """
65     Parses a given file and returns a dict containing the routerID,
input ports
66     and output ports with their cost and next hop
67     """
68     #Create an instance of configparser object
69     config = configparser.ConfigParser()
70     config.read_file(file)
71     # dict declartion
72     router = {}
73     # Reading in each section of the config
74     routerId = (config.get('DEFAULT', 'router-id'))
75     inputPorts = (config.get('DEFAULT', 'input-ports'))
76     outputPorts = (config.get('DEFAULT', 'output-ports'))
77     # Checks config file for periodic timer override or defaults to
standard
78     periodicUpdate = config.get("DEFAULT", "periodic-update", fallback=3.0)
79
80     # Validating all parameters

```

```

82     router["routerId"] = check_router_id(routerId)
83     router["inputPorts"] = check_input_ports(inputPorts)
84     router["outputPorts"] = check_output_ports(router, outputPorts)
85     router["periodicUpdate"] = check_periodic_update(periodicUpdate)
86
87     return router
88
89 def check_periodic_update(periodicUpdate):
90     """
91     Reduces the chance of collisions and other nasties by implementing a
92     random wait to the periodicUpdate
93     """
94     return periodicUpdate + (random.random() * 2 - 1)
95
96 def check_router_id(routerId):
97     """
98     Takes a routerID string from the config and checks it
99     Returns it back as an int if its valid
100    """
101    try:
102        routerId = int(routerId)
103    except:
104        raise TypeError("RouterID must be an integer")
105    if (routerId > 64000 or routerId < 1):
106        raise ValueError("RouterID must be between 1 and 64000")
107    return routerId
108
109
110 def check_input_ports(inputPorts):
111     """
112     Takes a string of inputports from the config
113     Validates and then returns them as a list
114     """
115     try:
116         inputPorts = [int(port.strip()) for port in inputPorts.split(',')]
117     except:
118         raise TypeError("Input ports should be comma seperated ints")
119     for port in inputPorts:
120         if (port > 64000 or port < 1024):
121             raise ValueError("Port should be between 1024 and 64000")
122     if len(inputPorts) != len(set(inputPorts)):
123         raise ValueError("Ports should be unique")
124     return inputPorts
125
126
127 def check_output_ports(router, outputPorts):
128     """
129
130     Takes an incomplete router dict containing a routerID and input
131     ports
132     Tests the routerID and input ports against a list of outputPorts

```

```

132     Returns a list of outputPorts if they are all valid.
133
134     """
135     outportPortList = []
136     try:
137         outputPorts = [port.strip() for port in outputPorts.split(',')]
138     except:
139         raise TypeError(
140             "Output ports should be comma seperated in the form PORT-COST-
141             ID")
142     for output in outputPorts:
143         config = {}
144         output = output.split('-')
145         output = [int(i) for i in output]
146         config["cost"] = output[1]
147
148         if (output[0] > 64000 or output[0] < 1024):
149             raise ValueError("Port should be between 1024 and 64000")
150         if output[2] == router["routerId"]:
151             raise ValueError("Output port routerID matches own routerID")
152         if any(d.get('routerId', None) == output[2] for d in
153             outportPortList):
154             raise ValueError("RouterID already exists in output list")
155         config["routerId"] = output[2]
156         if output[0] in router["inputPorts"]:
157             raise ValueError("Output port is shared with an input port")
158         if any(d.get('outputPort', None) == output[0] for d in
159             outportPortList):
160             raise ValueError("OutputPort already in use")
161         config["outputPort"] = output[0]
162         outportPortList.append(config)
163
164     return outportPortList
165
166 def open_config_file(filePath):
167     """
168     Takes a filepath as argument, validates it and returns a Config
169     object
170     """
171     file = open(filePath, 'r')
172     if file.mode == 'r':
173         config = Config()
174         config.parse_file(file)
175     else:
176         print("Error opening file")
177     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)
14  - Jesse Sheehan (53366509)
16  """
18  import bencode
20  import binascii
22  __encoding = "utf-8"
24  def encode(data):
26      """
28      Encodes the raw data, including a checksum.
30      """
32      body = bencode.bencode(data).encode(__encoding)
34      crc = binascii.crc32(body)
36      return crc.to_bytes(4, "big") + body
38  def decode(data):
40      """
42      Decodes raw data, checks the validity and returns the dictionary
44      containing the data.
46      Returns None if the data is invalid.
48      """
50      try:
52          # get the CRC32 code
54          crc = int.from_bytes(data[:4], "big")
56          # get the body
58          body = data[4:]
60          # return None if the checksum is incorrect
62          if crc != binascii.crc32(body):
64              return None
66          # return the decoded data if the checksum is correct
68          else:
70              return bencode.bdecode(body.decode(__encoding))
72      except:
74          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
5      routing_table_entry.py
6
7      COSC364 RIP Assignment
8
9      Date: 02/05/2019
10
11      Written by:
12      - Will Cowper (81163265)

```

```

14         - Jesse Sheehan (53366509)
15     """
16
17     class RoutingTableEntry:
18         """
19         A RoutingTableEntry represents a RIP entry that resides in the routing
20         table
21         """
22
23         def __init__(self, destination, nextHop, cost):
24             self.destination = destination
25             self.nextHop = nextHop
26             self.cost = cost
27             self.age = 0.0
28             self.garbage = False
29
30         def __str__(self):
31             return "RouteTableEntry <destination={0}, nextHop={1}, cost={2},
32             age={3}, garbage={4}>".format(self.destination, self.nextHop, self.cost
33             , round(self.age, 2), self.garbage)
34
35         def __repr__(self):
36             return self.__str__()

```

../src/routing_table_entry.py

2.1.6 src/routing_table.py

```

1  #!/usr/bin/python3
2
3  """
4
5      routing_table.py
6
7      COSC364 RIP Assignment
8
9      Date: 02/05/2019
10
11      Written by:
12          - Will Cowper (81163265)
13          - Jesse Sheehan (53366509)
14  """
15
16
17
18  import os
19  from routing_table_entry import RoutingTableEntry
20  import config
21
22  class RoutingTable:

```

```

24     """
25     The RoutingTable represents the list of RoutingTableEntries for a
26     router.
27     """
28     def __init__(self, config, logging_function = None):
29         """
30         Creates a new RoutingTable based on the Config.
31         """
32         self.__routes = []
33         self.routerID = config.router_id
34         self.__logging_function = logging_function
35
36     def add_entry(self, destination, nextHop, totalCost):
37         """
38         Adds a new RoutingTableEntry to the RoutingTable.
39         """
40         route = RoutingTableEntry(destination, nextHop, totalCost)
41         self.__routes.append(route)
42
43     def set_garbage(self, routerID, isGarbage):
44         """
45         Sets the garbage flag of the entry.
46         """
47         index = self.get_index(routerID)
48         self.__routes[index].garbage = isGarbage
49         self.reset_age(routerID)
50         if isGarbage:
51             self.set_cost(routerID, 16)
52
53     def log(self, message):
54         if self.__logging_function is not None:
55             self.__logging_function(message)
56         else:
57             print(message)
58
59     def reset_age(self, routerID):
60         """
61         Resets the age of the entry to 0.
62         """
63         index = self.get_index(routerID)
64         self.__routes[index].age = 0.0
65
66     def increment_age(self, time):
67         """
68         Increments the age of all entries in the RoutingTable.
69         """
70         for entry in self.__routes:
71             if entry.destination != self.routerID:
72                 entry.age += time
73
74     def delete_entry(self, routerID):
75         """

```

```

76         Deletes an entry with the specific routerID from the
RoutingTable.
"""
78         index = self.get_index(routerID)
del self._routes[index]
80
82     def get_index(self, routerID):
"""
Gets the index of the entry with the routerID. Returns -1 if
not found.
"""
84         for i, route in enumerate(self._routes):
86             if route.destination == routerID:
return i
88         return -1 # Not found

90     def set_cost(self, routerID, cost):
"""
92         Sets the cost of the entry.
"""
94         index = self.get_index(routerID)
self._routes[index].cost = cost
96
98     def set_next_hop(self, routerID, nextHop):
"""
100         Sets the next hop of the entry.
"""
102         index = self.get_index(routerID)
self._routes[index].nextHop = nextHop

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

112         for route in self._routes:
114             if route.age > 10 and not route.garbage:
self.log("marked router " + str(route.destination) + " as
garbage")
self.set_garbage(route.destination, True)
triggered_routes.append(route)
116
118             if route.age > 20 and route.garbage:
self.log("purged router " + str(route.destination) + " from
database")
remove_routes.append(route)
120
122         if len(triggered_routes) != 0:
triggered_update_callback(triggered_routes)

```

```

124         for route in remove_routes:
125             self.__routes.remove(route)
126
127     def __getitem__(self, routerId):
128         """
129         Gets the entry with the given routerId.
130         """
131         index = self.get_index(routerId)
132         if index != -1:
133             return self.__routes[index]
134         return None
135
136     def __iter__(self):
137         """
138         Returns the iterator of the routes.
139         """
140         return iter(self.__routes)
141
142     def __len__(self):
143         """
144         Returns the number of routes this RoutingTable has.
145         """
146         return len(self.__routes)
147
148     def __str__(self):
149         """
150         Returns a human-readable RoutingTable that can be printed to
151         the terminal.
152         """
153         s = [
154             "+-----+-----+-----+-----+-----+",
155             "| Dest.      | Next Hop  | Total Cost | Age       | Garbage?  |",
156             "+-----+-----+-----+-----+-----+",
157             ]
158         for route in self.__routes:
159             s.append(" | {0:<10} | {1:<10} | {2:<10} | {3:<10} | {4:<10} | ".
160 format(
161             route.destination, route.nextHop, route.cost, round(route.
162 age, 2), route.garbage))
163             s.append("
164             +-----+-----+-----+-----+-----+
165             return os.linesep.join(s)
166
167 # runs a simple test
168 if __name__ == "__main__":
169     current_directory = os.path.dirname(__file__)
170     parent_directory = os.path.split(current_directory)[0]
171     file_path = os.path.join(parent_directory, 'configs/good/01.conf')

```

```
170     config = config.open_config_file(file_path)
171     r = RoutingTable(config)
172     print(r)
```

../src/routing_table.py

2.1.7 src/server.py

```
#!/usr/bin/python3
2
3 """
4
5     server.py
6
7     COSC364 RIP Assignment
8
9     Date: 02/05/2019
10
11     Written by:
12     - Will Cowper (81163265)
13     - Jesse Sheehan (53366509)
14 """
15
16 import socket
17 import select
18 import time
19
20 import timer
21 import routing_table
22 import routing_table_entry
23 import protocol
24 import utils
25 import bencode
26
27
28 def create_input_socket(port, host='localhost'):
29     """
30     Creates a new UDP socket.
31     """
32     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
33     sock.bind((host, port))
34     return sock
35
36
37
38 class Server:
39
40     def __init__(self, config):
41         """
42         Creates a new server with a configuration.
43         """
44         self.rt = routing_table.RoutingTable(config, self.log)
```

```

46         self.config = config
47         self.input_ports = []
48         self.periodic_timer = None
49         self.loglines = []
50
51     def print_display(self):
52         """
53         Displays useful information for the user.
54         """
55
56         # clear the screen
57         utils.clear_terminal()
58
59         # print info about this router
60         print("RIP Router #" + str(self.config.router_id))
61         print("Uptime: {0} seconds".format(
62             round(self.periodic_timer.getElapsed())))
63
64         # print the routing table
65         print(self.rt)
66
67         # print other info
68         print("Press Ctrl+C to quit")
69
70     def process_periodic_update(self, dt):
71         """
72         Called when the periodic timer is triggered.
73         """
74
75         # send destination, next hop and total cost of each routing entry
76         to each input port
77         sock = self.input_ports[0]
78
79         for output_port in self.config.output_ports:
80
81             # add self to the routes
82             routes = [{
83                 "destination": self.config.router_id,
84                 "cost": 0,
85                 "next-hop": self.config.router_id
86             }]
87
88             # if len(self.rt) == 0:
89             #     self.log("advertising self to " + str(output_port.
90 router_id))
91
92             for route in self.rt:
93
94                 cost = route.cost
95                 destination = route.destination
96
97                 # poison reverse by setting cost to 16 when announcing
98                 routes back from where they were learned
99                 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).
        """
        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            # Check route is valid before any processing is done

136            # route lists ourself as the destination (useless) or as the
            next hop (invalid) and should be dropped
            if route_destination == self.config.router_id or route_next_hop
            == self.config.router_id:
138                continue

140            # route contains a negative cost and should be dropped
            if (route_cost < 0) or (packet.link_cost < 0):
142                continue

```



```

144         # route is valid and should be processed
146         # total cost is the link cost added to the cost contained in
the packet
        total_destination_cost = route_cost + packet.link_cost
148
        is_destination_unreachable = (total_destination_cost >= 16)
150
        # clamp cost to maximum of 16
152        if is_destination_unreachable:
            total_destination_cost = 16
154
        # is the destination routerID known
156        is_destination_in_table = destination_entry is not None
158
        # New valid route
        if not is_destination_in_table and not
is_destination_unreachable:
160
            # put the destination in the table
            self.rt.add_entry(route_destination, route_next_hop,
162            total_destination_cost)
            self.log("added new router " + str(route_destination) + "
via " + str(route_next_hop) + " with a cost of " + str(
total_destination_cost))
164
            # Route already exists in table
166            elif is_destination_in_table:
168
                is_destination_garbage = destination_entry.garbage
170
                # Check for a better route.
                if total_destination_cost < destination_entry.cost:
172                    self.rt.set_cost(route_destination,
total_destination_cost)
                    self.rt.set_garbage(route_destination, False)
174                    self.rt.set_next_hop(route_destination, route_next_hop)
176
                    self.log("found new route to " + str(route_destination)
+ " via " + str(route_next_hop) + " with a cost of " + str(
total_destination_cost))
178
                # Check for worse route from the same hop
                elif route_next_hop == destination_entry.nextHop and
total_destination_cost > destination_entry.cost:
180                    if is_destination_unreachable:
                        # garbage it if we haven't seen it before,
otherwise ignore it
182                    if not is_destination_garbage:
                        self.rt.set_garbage(route_destination, True)
                        triggered_updates.append(destination_entry)
184                        self.log("processed a triggered update from " +
str(packet.routes[0]["next-hop"]) + " marked " + str(route_destination

```

```

186         ) + " as garbage")
187
188         # We got a worse route from the samehop but its not
189         # infinite. As a neighbour we MUST update to the higher cost.
190         else:
191             self.rt.set_cost(route_destination,
192                             total_destination_cost)
193             self.rt.reset_age(route_destination)
194
195             # Check for worse route from a different hop and ignore it
196             elif total_destination_cost > destination_entry.cost:
197                 #self.log("Worse route to " + str(route_destination) +
198                 " ignoring it")
199                 continue
200
201             # Check for same route and keep it alive
202             elif route_next_hop == destination_entry.nextHop and
203             total_destination_cost == destination_entry.cost:
204                 # We dont want to keep alive infinite weight routes
205                 if not is_destination_garbage:
206                     self.rt.reset_age(route_destination)
207
208             if len(triggered_updates) > 0:
209                 self.log("sending triggered updates")
210                 self.process_triggered_updates(triggered_updates)
211
212             return None
213
214 def process_triggered_updates(self, routes):
215     """
216     Processes the triggered updates.
217     """
218     sock = self.input_ports[0]
219     for output_port in self.config.output_ports:
220
221         packet_routes = [{
222             "destination": route.destination,
223             "cost": 16,
224             "next-hop": self.config.router_id
225         } for route in routes]
226
227         p = protocol.Packet(output_port.cost, packet_routes)
228         sock.sendto(p.to_data(), ('localhost', output_port.port))
229
230 def start(self):
231     """
232     Starts the server.
233     """
234
235     # set up the input ports
236     self.input_ports = list(
237         map(create_input_socket, self.config.input_ports))

```

```

234     # start the periodic timer
235     self.periodic_timer = timer.Timer(
236         self.config.periodic_update, self.process_periodic_update)
237     self.periodic_timer.start()
238     self.periodic_timer.trigger()
239
240     # only block for a second at a time
241     blocking_time = 0.1
242
243     loop_time = time.time()
244
245     while self.input_ports:
246         readable, _writable, exceptional = select.select(
247             self.input_ports, [], self.input_ports, blocking_time)
248
249         # increment the age
250         dt = time.time() - loop_time
251         self.rt.increment_age(dt)
252
253         # redisplay the screen
254         self.print_display()
255
256         # update the timer, may call process_periodic_update
257         self.periodic_timer.update()
258
259         # may call process_triggered_updates
260         self.rt.update(self.process_triggered_updates)
261
262         # display the information log
263         print("")
264         print("Information Log:")
265         for line in self.loglines:
266             print(" ", line)
267
268         # iterate through all sockets that have data waiting on them
269         for sock in readable:
270             data, addr = sock.recvfrom(4096)
271             resp = self.process_incoming_data(addr, data)
272
273             if resp is not None:
274                 sock.sendto(resp, addr)
275
276         # removes a socket from the input list if it raised an error
277         for sock in exceptional:
278             if sock in self.input_ports:
279                 self.input_ports.remove(sock)
280                 sock.close()
281             raise Exception("A socket raised an error")
282
283         # update the loop time
284         loop_time = time.time()
285
286 if __name__ == "__main__":

```

```
pass
```

```
../src/server.py
```

2.1.8 src/timer.py

```
#!/usr/bin/python3
2
3 """
4
5     timer.py
6
7     COSC364 RIP Assignment
8
9     Date: 02/05/2019
10
11     Written by:
12         - Will Cowper (81163265)
13         - Jesse Sheehan (53366509)
14 """
15
16 import time
17
18 class Timer:
19
20     def __init__(self, period, callback):
21         """
22         """
23         Creates a new Timer with a period and a callback.
24         """
25         self.__period = period
26         self.__callback = callback
27         self.__started = False
28         self.__startedTime = 0
29         self.__paused = False
30         self.__pausedTime = 0
31         self.__updateTime = 0
32
33     def start(self):
34         """
35         """
36         Starts the timer.
37         """
38         if not self.__started:
39             t = time.time()
40             self.__started = True
41             self.__startedTime = t
42             self.__paused = False
43             self.__pausedTime = 0
44             self.__updateTime = t
45
46     def stop(self):
47         """
48         """
```

```
48         """
50         Stops the timer.
52         """
54         if self.__started:
55             self.__started = False
56             self.__startedTime = 0
57             self.__paused = False
58             self.__pausedTime = 0
59             self.__updateTime = 0
60
61     def reset(self):
62         """
63         Resets the timer.
64         """
65         if self.__started:
66             self.stop()
67             self.start()
68
69     def pause(self):
70         """
71         Pauses the timer.
72         """
73         if self.__started and not self.__paused:
74             self.__paused = True
75             self.__pausedTime = time.time() - self.__startedTime
76             self.__startedTime = 0
77
78     def resume(self):
79         """
80         Resumes the timer.
81         """
82         if self.__started and self.__paused:
83             self.__startedTime = time.time() - self.__pausedTime
84             self.__paused = False
85             self.__pausedTime = 0
86
87     def update(self):
88         """
89         Updates the timer. May call its callback.
90         """
91         if self.__started and not self.__paused:
92             t = time.time()
93             dt = t - self.__updateTime
94             if dt > self.__period:
95                 self.__updateTime = t
96                 self.__callback(dt)
97
98     def trigger(self):
99         """
100         Forcefully call the callback.
101         """
102         if self.__started and not self.__paused:
103             t = time.time()
104             dt = t - self.__updateTime
```

```

100         self.__updateTime = t
101         self.__callback(dt)
102
103     def getElapsed(self):
104         """
105         Returns the time elapsed in seconds.
106         """
107         if self.__started:
108             if self.__paused:
109                 return self.__pausedTime
110             else:
111                 return time.time() - self.__startedTime
112         return 0.0
113
114     def isStarted(self):
115         """
116         Returns True if the timer has been started.
117
118         >>> t = Timer(10, None)
119         >>> t.isStarted()
120         False
121         >>> t.start()
122         >>> t.isStarted()
123         True
124         >>> t.stop()
125         >>> t.isStarted()
126         False
127         """
128         return self.__started
129
130     def isPaused(self):
131         """
132         Returns True if the timer has been paused.
133
134         >>> t = Timer(10, None)
135         >>> t.isPaused()
136         False
137         >>> t.start()
138         >>> t.isPaused()
139         False
140         >>> t.pause()
141         >>> t.isPaused()
142         True
143         >>> t.resume()
144         >>> t.isPaused()
145         False
146         >>> t.stop()
147         >>> t.isPaused()
148         False
149         >>> t.start()
150         >>> t.pause()
151         >>> t.isPaused()
152         True

```

```

154         >>> t.stop()
155         >>> t.isPaused()
156         False
157         """
158         return self._paused and self._started
159
160     def __str__(self):
161         """
162         Returns a string representation of the timer.
163         """
164         return "Timer <period={0:.3}s, started={1}, paused={2}, elapsed
165         ={3:.3}s>".format(self._period, self._started, self._paused, self.
166         getElapsed())
167
168     def __repr__(self):
169         """
170         Returns a string representation of the timer.
171         """
172         return self.__str__()
173
174 # run doctests
175 if __name__ == "__main__":
176     import doctest
177     doctest.testmod()

```

../src/timer.py

2.1.9 src/utls.py

```

2  #!/usr/bin/python3
3  """
4
5      utils.py
6
7      COSC364 RIP Assignment
8
9      Date: 02/05/2019
10
11     Written by:
12     - Will Cowper (81163265)
13     - Jesse Sheehan (53366509)
14     """
15
16 import os
17
18 def clear_terminal():
19     """
20     Clears the terminal based on the type of operating system.
21     """
22

```

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

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

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

../src/utils.py

2.2 Configuration Files

2.2.1 configs/networks/figure1/1.conf

```
1 ; configs/networks/figure1/1.conf
2 ; created with tools/generate_network.py
3
4 [DEFAULT]
5 router-id = 1
6 input-ports = 55501, 55503, 55505
7 output-ports = 55500-1-2, 55502-5-6, 55504-8-7
```

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

2.2.2 configs/networks/figure1/2.conf

```
1 ; configs/networks/figure1/2.conf
2 ; created with tools/generate_network.py
3
4 [DEFAULT]
5 router-id = 2
6 input-ports = 55500, 55507
7 output-ports = 55501-1-1, 55506-3-3
```

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

2.2.3 configs/networks/figure1/3.conf

```
1 ; configs/networks/figure1/3.conf
2 ; created with tools/generate_network.py
3
4 [DEFAULT]
5 router-id = 3
6 input-ports = 55506, 55509
7 output-ports = 55507-3-2, 55508-4-4
```

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

2.2.4 configs/networks/figure1/4.conf

```
1 ; configs/networks/figure1/4.conf
2 ; created with tools/generate_network.py
3
4 [DEFAULT]
5 router-id = 4
6 input-ports = 55508, 55511, 55513
7 output-ports = 55509-4-3, 55510-2-5, 55512-6-7
```

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

2.2.5 configs/networks/figure1/5.conf

```
1 ; configs/networks/figure1/5.conf
2 ; created with tools/generate_network.py
3
4 [DEFAULT]
5 router-id = 5
6 input-ports = 55510, 55515
7 output-ports = 55511-2-4, 55514-1-6
```

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

2.2.6 configs/networks/figure1/6.conf

```
1 ; configs/networks/figure1/6.conf
2 ; created with tools/generate_network.py
3
4 [DEFAULT]
5 router-id = 6
6 input-ports = 55502, 55514
7 output-ports = 55503-5-1, 55515-1-5
```

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

2.2.7 configs/networks/figure1/7.conf

```
1 ; configs/networks/figure1/7.conf
2 ; created with tools/generate_network.py
3
4 [DEFAULT]
5 router-id = 7
6 input-ports = 55504, 55512
7 output-ports = 55505-8-1, 55513-6-4
```

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

2.2.8 configs/networks/six-node/01.conf

```
1 ; section 3.1.1 of the routing booklet
2
3 [DEFAULT]
4 router-id = 1
5 input-ports = 5512, 5514
6 output-ports = 5521-1-2, 5541-1-4
```

../configs/networks/six-node/01.conf

2.2.9 configs/networks/six-node/02.conf

```
2 ; section 3.1.1 of the routing booklet
3
4 [DEFAULT]
5 router-id = 2
6 input-ports = 5521, 5524, 5523
7 output-ports = 5512-1-1, 5542-1-4, 5532-2-3
```

../configs/networks/six-node/02.conf

2.2.10 configs/networks/six-node/03.conf

```
2 ; section 3.1.1 of the routing booklet
3
4 [DEFAULT]
5 router-id = 3
6 input-ports = 5532, 5534, 5535, 5536
7 output-ports = 5523-2-2, 5543-1-4, 5553-1-5, 5563-1-6
```

../configs/networks/six-node/03.conf

2.2.11 configs/networks/six-node/04.conf

```
2 ; section 3.1.1 of the routing booklet
3
4 [DEFAULT]
5 router-id = 4
6 input-ports = 5541, 5542, 5543, 5545, 5546
7 output-ports = 5514-1-1, 5524-1-2, 5534-1-3, 5554-2-5, 5564-15-6
```

../configs/networks/six-node/04.conf

2.2.12 configs/networks/six-node/05.conf

```
2 ; section 3.1.1 of the routing booklet
3
4 [DEFAULT]
5 router-id = 5
6 input-ports = 5553, 5554, 5556
7 output-ports = 5535-1-3, 5545-2-4, 5565-1-6
```

../configs/networks/six-node/05.conf

2.2.13 configs/networks/six-node/06.conf

```
2 ; section 3.1.1 of the routing booklet
3
4 [DEFAULT]
5 router-id = 6
6 input-ports = 5563, 5564, 5565
7 output-ports = 5536-1-3, 5546-15-4, 5556-1-5
```

../configs/networks/six-node/06.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
2
3 """
4
5     generate_network.py
6
7     COSC364 RIP Assignment
8
9     Date: 02/05/2019
10
11     Written by:
12     - Will Cowper (81163265)
13     - Jesse Sheehan (53366509)
14 """
15
16 import os
17 import sys
18
19
20 def get_network_name():
21     network_name = None
22     while network_name is None:
23         try:
24             network_name = input("Enter network name: ")
25             network_name = network_name.strip()
26             if not network_name.isalnum():
27                 print("Network name must be alpha-numeric")
28                 network_name = None
29         except:
30             print("ASD")
31             return None
32     return network_name
33
34
35 def get_router_ids():
36     router_ids = []
37     while len(router_ids) == 0:
38         try:
39             line = input("Enter router ids seperated by spaces: ")
40             router_ids = [id for id in line.strip().split(" ")]
41             is_valid = True
42             for id in router_ids:
43                 if not id.isalnum():
44                     is_valid = False
45                     break
46
```

```

48         if not is_valid:
49             print("All ids must be alpha-numeric")
50             router_ids = None
51     except:
52         return None
53     return router_ids
54
55 def get_link_cost(fromId, toId):
56     link_cost = None
57     while link_cost is None:
58         try:
59             line = input("Enter link cost between routers '" +
60                           str(fromId) + "' and '" + str(toId) + "': ")
61             line = line.strip()
62             if not line.isnumeric() or int(line) < 0:
63                 print("Link cost must be a positive integer (or 0 for
64 infinity)")
65             else:
66                 link_cost = int(line)
67         except Exception as e:
68             print(e)
69             return None
70     return link_cost
71
72 def main():
73     network_name = get_network_name()
74     if network_name is None:
75         return
76
77     router_ids = get_router_ids()
78     if router_ids is None:
79         return
80
81     configs = {}
82     port_number_max = 55500
83     for index, fromId in enumerate(router_ids):
84         for toId in router_ids[index + 1:]:
85             link_cost = get_link_cost(fromId, toId)
86             if link_cost is None:
87                 return
88
89             if link_cost == 0:
90                 continue
91
92             to_port_number = port_number_max
93             port_number_max += 1
94             from_port_number = port_number_max
95             port_number_max += 1
96
97             if fromId not in configs:
98                 configs[fromId] = {"output-ports": []},

```

```

100         "input-ports": [], "router-id": fromId}
101     configs[fromId]["output-ports"].append(
102         (to_port_number, link_cost, toId))
103     configs[fromId]["input-ports"].append(from_port_number)
104
105     if toId not in configs:
106         configs[toId] = {"output-ports": [],
107                         "input-ports": [], "router-id": toId}
108     configs[toId]["output-ports"].append(
109         (from_port_number, link_cost, fromId))
110     configs[toId]["input-ports"].append(to_port_number)
111
112 # assign port numbers
113 root_path = os.path.join("configs", "networks", network_name)
114 if not os.path.exists(root_path):
115     os.mkdir(root_path)
116     print("Created directory", root_path)
117
118 for key in configs:
119     config = configs[key]
120     filename = os.path.join(root_path, config["router-id"] + ".conf")
121     with open(filename, "w") as f:
122         f.write("; " + filename + "\n")
123         f.write("; created with tools/generate_network.py\n")
124         f.write("\n")
125         f.write("[DEFAULT]\n")
126         f.write("router-id = " + str(config["router-id"]) + "\n")
127         f.write("input-ports = " + ", ".join([str(x)
128             for x in config["input-
129 ports"]])) + "\n")
130         f.write("output-ports = " + ", ".join([str(x[0]) + "-" + str(x
131 [1]) + "-" + str(x[2])
132             for x in config["output-
133 ports"]])) + "\n")
134         f.write("\n")
135         print("Created", filename)
136
137 # print("Creating ", network_name, "with", configs)
138
139 if __name__ == "__main__":
140     main()

```

../tools/generate_network.py