COSC 364

Internet Technologies and Engineering

Assignment 1 – RIP Protocol

Michael Toft Jensen – mto59 - 36622258

David Barclay – dlb70 – 18505639

Assignment Contribution:

Contribution to the assignment was an even 50%/50%

Good parts:

Program structure; Well defined classes with encapsulation of responsibility for specific functions. This makes it more object oriented and easier to maintain and navigate in the code. Though it makes the code a bit longer.

Commenting; Functions and methods are described with a comment defining the purpose of the method.

The way that the timing is handled allows for operations to happen out of order so that no operations (update sending, processing, garbage collection) have to wait for a long time. The program does not need to loop sequentially or use a counter to trigger operations, instead, they all rely on individual timers to trigger, meaning that timing is accurate and routers do not converge their timers.

The value of the infinity metric was chosen to be 30, as this is only slightly larger than the maximum path in the given network (5-6-1-7-4-3-2, metric 27).

What could be improved:

Could add checks on messages to confirm that the contents are in the correct order and have the correct values.

An error message should be printed when there are obvious conflicts in the configuration files.

Atomicity of event processing: All events occur in sequence completely.

Using the select() statement, we read in available updates and process them all sequentially (not necessarily in the order they arrived). If an update is corrupt, we drop it, meaning that we do not partially update.

By using the select() statement, we do not interrupt any other operations, and we wait for packets to be read. This ensures that all other operations finish completely before update processing occurs.

Testing: Create scenarios with predictable behaviour then test actual behaviour.

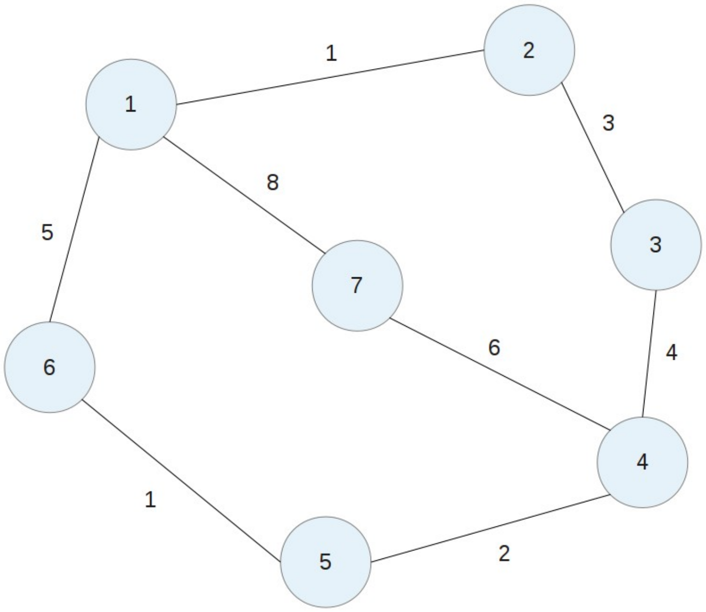
Configurations were created to test various aspects of the program. Networks were designed with specific attributes (combined with sequences of actions) that could exploit weaknesses in the protocol, as well as configurations that tested the basic functionality and robustness of the program. The configuration files were written in accordance to the network design, routing tables were predicted for the various states of the network, and then compared to the results generated by running the program with the designed configurations.

Test 0: Assignment.

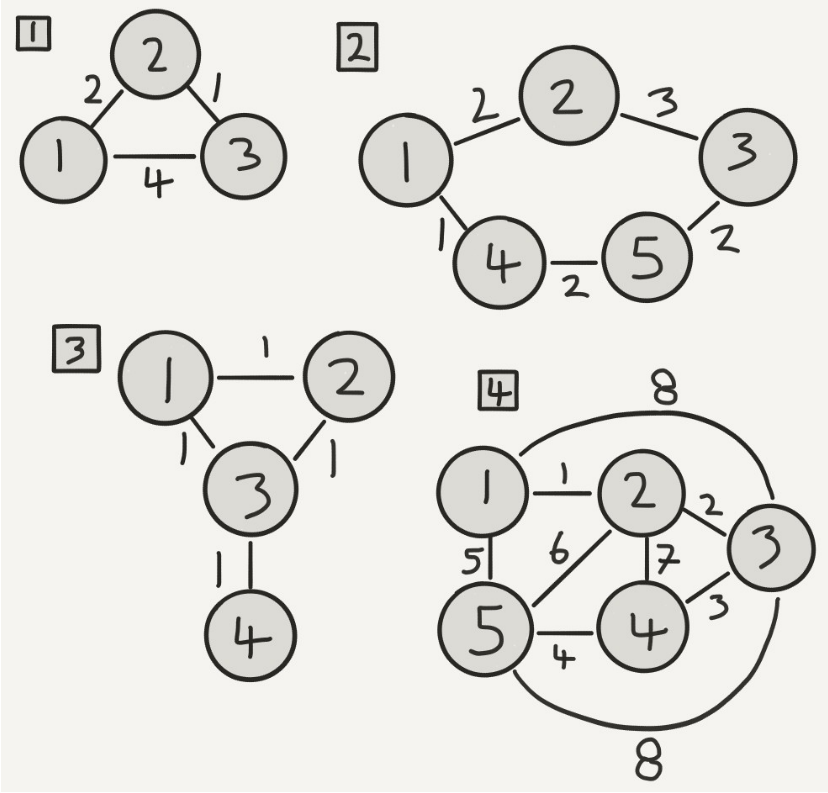
Using the given configuration in the assignment specification, various combinations of routers were launched, killed, and revived, to test whether or not the program would fail or calculate incorrect routes.

A bug was discovered in the code by using the default configuration files.

By launching routers 1,2,3,4,7, letting them converge, and killing 7, a routing loop occurred. The bug was fixed in commit 521ebb3 which affected lines 204-205 of the code. The code was using the destination of a route instead of the first hop to determine whether on not to poison.



Custom Test Case network graphs.



Test 1: Trivial 3 node network. Test is designed for testing basic functionality.

Test Case 1

```
file: Testcase1/1.cfg.
router-id 1
input-ports 10012 10013
outputs 10021-2-2 10031-4-3
```

Routing Tables

Router 1	Dest	First	Metric
	2	2	2
	3	2	3

```
file: Testcase1/2.cfg.
router-id 2
input-ports 10021 10023
outputs 10012-2-1 10032-1-3
```

Router 2	Dest	First	Metric
	1	1	2
	3	3	1

```
file: Testcase1/3.cfg.
router-id 3
input-ports 10031 10032
outputs 10013-4-1 10023-1-2
```

Router 3	Dest	First	Metric
	1	2	3
	2	2	1

Test 2: Equal cost path.

This test case is designed to show that routes are not chosen in preference to minimizing hops. It is assumed that routers are initialized in increasing numerical order, so that results are consistent. Routers are first allowed to converge, then router 2 is killed temporarily to force routers 1 and 3 to route through 4 and 5 to reach each other. After routers converge while 2 is dead, revive router 2. No subsequent change should be observed as the metrics of routes 1-2-3 and 1-4-5-3 are both equal, and the protocol does not prefer minimum hop routing, only minimum cost. (note that other routes of equal distance may be inconsistent with the routes shown in the figure.)

Test Case 2	Routing Tables				After router 2 fails and restarts			
file: Testcase2/1.cfg.	Router 1	Dest	First	Metric	Router 1	Dest	First	Metric
router-id 1		2	2	2		2	2	2
input-ports 10012 10014		3	2	5		3	4	5
outputs 10021-2-2 10041-1-4		4	4	1		4	4	1
		5	4	3		5	4	3
file: Testcase2/2.cfg.	Router 2	Dest	First	Metric	Router 2	Dest	First	Metric
router-id 2		1	1	2		1	1	2
input-ports 10021 10023		3	3	3		3	3	3
outputs 10012-2-1 10032-3-3		4	1	3		4	1	3
		5	3	5		5	3	5
file: Testcase2/3.cfg.	Router 3	Dest	First	Metric	Router 3	Dest	First	Metric
router-id 3		1	2	5		1	2	5
input-ports 10032 10035		2	2	3		2	2	3
outputs 10023-3-2 10053-2-5		4	5	4		4	5	4
		5	5	2		5	5	2
file: Testcase2/4.cfg.	Router 4	Dest	First	Metric	Router 4	Dest	First	Metric
router-id 4		1	1	1		1	1	1
input-ports 10041 10045		2	1	3		2	1	3
outputs 10014-1-1 10054-2-5		3	5	4		3	5	4
		5	5	2		5	5	2
file: Testcase2/5.cfg.	Router 5	Dest	First	Metric	Router 5	Dest	First	Metric
router-id 5		1	4	3		1	4	3
input-ports 10054 10053		2	3	5		2	3	5
outputs 10045-2-4 10035-2-3		3	3	2		3	3	2
		4	4	2		4	4	2

Test 3: Split Horizon with Poisoned Reverse demonstration.

This configuration is designed to test the robustness of the Split Horizon with Poisoned Reverse. Routers are first allowed to converge, then router 3 is killed. If routers 1 and 2 detect the failure at the same time, and have both recently received a periodic update from each other, then they may set their route to 4 through each other. With poisoned reverse, in the next update received by each router, the routers will simply set the metric to router 4 to infinity. Without poisoned reverse, the route to 4 will persist until it times out as the routers do not report it to each other.

This test is very hard to reproduce as it relies on timers being synchronised.

```
Test Case 3                               Routing Tables                 |   Immediately after router 3 fails.
file: Testcase3/1.cfg.                    Router 1  Dest  First  Metric  |   Router 1  Dest  First  Metric
router-id 1                                            2      2       1  |                2      2       1
input-ports 10012 10013                                3      3       1  |
outputs 10021-1-2 10031-1-3                            4      3       2  |                4      2       2
                                                                         |
file: Testcase3/2.cfg.                    Router 2  Dest  First  Metric  |   Router 2  Dest  First  Metric
router-id 2                                            1      1       1  |                1      1       1
input-ports 10021 10023                                3      1       1  |
Outputs 10012-1-1 10032-1-3                            4      3       2  |                4      1       2
                                                                         |
file: Testcase3/3.cfg.                    Router 3  Dest  First  Metric  |   After poisoned reverse update
router-id 3                                            1      1       1  |   Router 1  Dest  First  Metric
input-ports 10031 10032 10034                          2      2       1  |                2      2       1
outputs 10013-1-1 10023-1-2 10043-1-4                  4      4       1  |
                                                                         |                4      2      30
file: Testcase3/4.cfg.                    Router 4  Dest  First  Metric  |
router-id 4                                            1      3       2  |   Router 2  Dest  First  Metric
input-ports 10043                                      2      3       2  |                1      1       1
outputs 10034-1-3                                      3      3       1  |
                                                                         |                4      1      30
```

Test 4: (Almost) Fully connected network (5 nodes)

This test case is almost trivial, but is designed to test the robustness of the protocol with a highly connected network.

Test Case 4

Routing Tables

```
file: Testcase4/1.cfg.
router-id 1
input-ports 10012 10013 10015
outputs 10021-1-2 10013-8-3 10051-5-5
```

Router 1	Dest	First	Metric
	2	2	1
	3	2	3
	4	2	6
	5	5	5

```
file: Testcase4/2.cfg.
router-id 2
input-ports 10021 10023 10024 10025
outputs 10012-1-1 10032-2-3 10042-7-4 10052-6-5
```

Router 2	Dest	First	Metric
	1	1	1
	3	3	2
	4	3	5
	5	5	6

```
file: Testcase4/3.cfg.
router-id 3
input-ports 10031 10032 10043 10053
outputs 10013-8-1 10023-2-2 10043-3-4 10053-8-5
```

Router 3	Dest	First	Metric
	1	2	3
	2	2	2
	4	4	3
	5	4	7

```
file: Testcase4/4.cfg.
router-id 4
input-ports 10042 10043 10045
outputs 10024-7-2 10034-3-3 10054-4-5
```

Router 4	Dest	First	Metric
	1	3	6
	2	3	5
	3	3	3
	5	5	4

```
file: Testcase4/5.cfg.
router-id 5
input-ports 10051 10052 10053 10054
outputs 10015-5-1 10025-6-2 10035-8-3 10045-4-4
```

Router 5	Dest	First	Metric
	1	1	5
	2	2	6
	3	4	7
	4	4	4

Full code base can be found here:
https://github.com/dlb70/rip

```
import sys
from time import time,sleep
from random import random
from hashlib import md5
import socket
from select import select


### GLOBALS ###
LOCALHOST = "127.0.0.1"
CONFIGFILE = sys.argv[1]
BUFSIZE = 1023      # Maximum bytes read by socket.recvfrom()
TIMER = 6           # Time between periodic updates
TIMEOUT = TIMER/6.0 # Timout length for select()
ENTRY_TIMEOUT = TIMER * 6 # Timeout length for entry invalidation
GARBAGE = TIMER * 6 # Timer for garbage collection
INFINITY = 30       # Metric representing infinity.



class Entry(object):
    def __init__(self, dest, first, metric, t=None):
        self.dest = dest # Destination dest (Router.id)
        self.first = first # First first along route (Router.id)
        self.metric = metric # The metric of this route
        if (t == None):
            t = time()
        self.time = t # The time value of when this entry was last udated

    def __repr__(self):
        rstr = ""
        rstr += "dest:" + str(self.dest) + ' '
        rstr += "first:" + str(self.first) + ' '
        rstr += "metric:" + str(self.metric) + ' '
        rstr += "time:" + str(self.timer())[:4] + ''
        return rstr

    def timer(self):
        """ Returns the time since last updated in seconds. """
        return time() - self.time


class EntryTable(object):
    def __init__(self):
        self.entries = {}

    def __repr__(self):
        rstr = str(self.entries)
        return rstr
```

```python

    def __str__(self):
        return repr(self)

    def tostr(self):
        rstr = ""
        for entry in self.getEntries():
            rstr += str(entry) + '\n'
        return rstr

    def destinations(self):
        """ Returns a sorted list of destinations in the table. """
        return sorted(self.entries.keys())

    def getEntry(self, dest):
        """ Returns an entry from the table specified by destination. """
        return self.entries.get(dest)

    def getEntries(self):
        """ Iterator function to return all entries in order. """
        for dest in self.destinations():
            yield self.entries.get(dest)

    def update(self, entry): ##############################################
        """ Tests to see wether the new entry should be added.
            Adds the entry to the table (replacing older entry if there).
            Returns the new entry if added, or None if not added.
        """
        current = self.getEntry(entry.dest)

        if (current == None): # If no record of destination
            if (entry.metric < INFINITY):
                self.entries.update({entry.dest:entry})
            return entry

        elif (current.first == entry.first): # If it came from the first hop
            if (entry.metric > INFINITY):
                entry.metric = INFINITY
            self.entries.update({entry.dest:entry})
            return entry

        elif (entry.metric < current.metric):
            self.entries.update({entry.dest:entry})
            return entry

        return None


    def removeEntry(self, dest):
        """ Removes an entry from the table by destination """
        if (self.entries.get(dest) == None):
            return None
        else:
            return self.entries.pop(dest)


class Output(object):
    """ Class defining the attributes of an output.
        Attributes refer to the destination router and its edge cost.
    """

    def __init__(self, string):
        """ Takes a string input of the form "port-metric-dest" """
        elements = string.split('-')
        self.port   = int(elements[0])
        self.metric = int(elements[1])
        self.dest   = int(elements[2]) # the Router.id of the router

    def __repr__(self):
        rstr = "("
        rstr += "port:" + str(self.port) + ','
        rstr += "metric:" + str(self.metric) + ','
        rstr += "dest:" + str(self.dest) + ')'
        return rstr

class Router(object):
    def __init__(self, rtrid, inputPorts, outputs):
        """ A Router.
            id          - is the int ID of the router.
            inputPorts  - is a list of ints which are ports on which to listen.
            outputs     - is a dict of the form {dest:Output},
                where 'dest' is the destination id.
            outputSocket    - is the socket on which updates are sent from.
        """
        self.id = rtrid
        self.entryTable = EntryTable()
        self.inputPorts = inputPorts
        self.inputSockets = []
        self.outputs = outputs
        self.outputSocket = None
        self.garbageTimer = 0

    def show(self):
        print("ID: " + str(self.id))
        print("Table: ")
        for entry in self.entryTable.getEntries():
            print(" - " + str(entry))

    def showIO(self):
        print("Inputs: " + str(self.inputPorts))
        print("Outputs: ")
        for output in sorted(self.outputs.keys()):
            print(" - " + str(self.outputs.get(output)))

    def checksum(self, payload):
        """ Returns a checksum of the payload """
        return md5(bytes(payload, 'utf-8')).hexdigest()[:10]

    def verifies(self, message):
        """ Returns True if a packet is valid, False if invalid """
        return (message[:10] == self.checksum(message[10:]))

    def openSocket(self, port):
        """ Open a socket for the router on the integer port. """
        try:
            s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            s.bind((LOCALHOST, port))
            print("Opened socket on port " + str(port))
            return s
        except:
            print("Could not open socket on port " + str(port))
            self.close()

    def openInputSockets(self):
        """ Creates a list of opened sockets using the asigned ports. """
        for port in self.inputPorts:
            sock = self.openSocket(port)
            self.inputSockets.append(sock)
        return self.inputSockets

    def openOutputSocket(self):
        """ Allocates the first input socket as the output socket.
            Does not actually open a socket.
        """
        self.outputSocket = self.inputSockets[0]
        return self.outputSocket

    def createUpdate(self, output): #######################################
        """ Create an update message (as a string) from the router's
            information and the routing table.

            Message format:
            '''
            HEADER self.id output.dest\n
            ENTRY self.id 0\n
            ENTRY dest metric\n
            ENTRY dest metric\n
            '''
        """
        # The first newline is to split the checksum from the message
        # Header = src dest
        message  = "\nHEADER " + str(self.id) + ' ' + str(output.dest) + '\n'
        message += "ENTRY " + str(self.id) + " 0\n"
        for entry in self.entryTable.getEntries():
            dest, metric, first = (entry.dest, entry.metric, entry.first)
            if (first == output.dest): # Split Horizon with Poisoned Reverse
```

```
                metric = INFINITY
            message += "ENTRY " + str(dest) + ' ' + str(metric) + '\n'

        checksum = self.checksum(message)
        return (checksum + message)


    def sendUpdate(self, output):
        """ Send a update message to the defined output. This involves sending
            a packet identifying the sender, and the sender's entry table.
            The entry table sent will include an entry for the sender with a
            metric set to zero.

            Split horizon - Avoid loop creation by not sending routes that have
                their first hop set to the defined output.
            Poizoned reverse - Instead of just removing those routes, set their
                metric to infinity (A constant INFINITY in reality)
        """
        message = self.createUpdate(output)
        self.outputSocket.sendto(bytes(message,'utf-8'),
                                    (LOCALHOST,output.port))

    def process(self, message): ###########################################
        """ Takes an update message as a string and processes it.
            If the checksum does not match the payload, drop the packet.
            Returns True on success
        """
        if not (self.verifies(message)):
            print("Invalid Checksum. Packed dropped!")
            return None # Drop the packet

        lines = message.split('\n')
        for line in lines[1:]: # Skip checksum
            line = line.split(' ')

            if (line[0] == ''): # End of message
                return None

            elif (line[0] == "HEADER"):
                source = int(line[1])
                routerid = int(line[2])
                if (routerid != self.id):
                    print("Message not destined for me. Dropping packet!")
                output = self.outputs.get(source)

            elif (line[0] == "ENTRY"):
                dest = int(line[1])
                if (dest != self.id): # Do not add ourselves
                    metric = int(line[2]) + output.metric
                    newEntry = Entry(dest, source, metric)
                    self.entryTable.update(newEntry)

        return None

    def recieveUpdate(self, sock):
        """ Reads a packet from the socket 'sock'
            Returns a tuple containing the str message and tuple address
        """
        packet = sock.recvfrom(BUFSIZE)
        address = packet[1]
        message = packet[0].decode(encoding='utf-8')
        #print("Packet recieved from " + address[0] + ':' + str(address[1]))#DBG
        return message

    def broadcast(self):
        """ Send an update message to all outputs """
        for output in self.outputs.keys():
            self.sendUpdate(self.outputs.get(output))

    def wait(self, timeout):
        """ Waits for an incoming packet. Returns a list of update messages
            to be processed, or None if there were no queued packets.
        """
        read, written, errors = select(self.inputSockets,[],[],timeout)
        if (len(read) > 0):
            messages = []
            for sock in read:
                message = self.recieveUpdate(sock)
                messages.append(message)
            return messages
        else:
            return None

    def garbageCollect(self): #############################################
        """ Removes expired entries from the entry table.
            Before removing the entries, set their metric to INFINITY
                and broadcast an update.
            Returns a list of removed entries.
        """
        self.garbageTimer += GARBAGE
        expired = []
        for entry in self.entryTable.getEntries():
            if (entry.timer() > ENTRY_TIMEOUT):
                expired.append(entry.dest)
                entry.metric = INFINITY
        if (len(expired) != 0):
            self.broadcast()     # Broadcast Update
            self.show()
            print()
            for dest in expired: # Remove Entries
                self.entryTable.removeEntry(dest)
        return expired
```

```python
    def close(self):
        """ close all sockets """
        try:
            for sock in self.inputSockets:
                sock.close()
        except:
            print("WARNING!!! Could not exit cleanly! " +
                "Sockets may still be open!")
            return 1
        return 0



def createRouter(cfg):
    """ Wrapper function for creating a Router object from a
        configuration file
    """
    l = cfg.readline().strip('\n')
    while (l != ""):
        if l.startswith("router-id"):
            rtrid = int(l.split(' ')[1])

        if l.startswith("input-ports"):
            inputs = l.strip("input-ports ").split(' ')
            inputs = list(map(int, inputs))

        if l.startswith("outputs"):
            outputList = l.strip("outputs ").split(' ')
            outputs = {}
            for string in outputList:
                newOutput = Output(string)
                outputs.update({newOutput.dest:newOutput})

        l = cfg.readline().strip('\n')
    return Router(rtrid,inputs,outputs)



def main(router):
    router.openInputSockets()
    router.openOutputSocket()
    print("INITIALIZED.\n\n")
    router.broadcast()
    t = time()
    router.garbageTimer = time() + GARBAGE
    while True:
        # Do main router update message
        if ((time() - t) >= TIMER):
            # router.show()
            t += TIMER + (random() - 0.5) * (TIMER * 0.4) # Randomises timer
            router.broadcast()
            router.show()
            print()

        # Wait for incoming packets
        packets = router.wait(TIMEOUT)
        if (packets != None):
            for packet in packets:
                router.process(packet)

        # Do garbage collection
        if (time() - router.garbageTimer >= GARBAGE):
            # timer reset is called from garbageCollect()
            print("GARBAGE COLLECTION")
            n = len(router.garbageCollect())
            print("GARBAGE COLLECTION: Removed " + str(n) + "entries.")

        #if (router.garbageCollect() != None):
        #    print("GARBAGE COLLECTION")

if (__name__ =="__main__"):
    print("\nReading from config file: " + CONFIGFILE)
    configFile = open(CONFIGFILE,'r')
    print("INITIALIZING...")
    router = createRouter(configFile)
    configFile.close()
    router.show()
    router.showIO()
    try:
        main(router)
    except(KeyboardInterrupt, SystemExit):
        print("\nrecieved interrupt, closing...  ")
        router.close()
        print("done.")
        exit(0)
```

Configuration files for the main assignment configuration.

```
Format
file: folder/id.cfg
router-id X          (X = router ID)
input-ports 100XY    (Y = neighbor)
outputs 100YX-M-Y    (M = metric)
EOF

file: config/1.cfg
router-id 1
input-ports 10012 10016 10017
outputs 10021-1-2 10061-5-6 10071-8-7

file: config/2.cfg
router-id 2
input-ports 10021 10023
outputs 10012-1-1 10032-3-3

file: config/3.cfg
router-id 3
input-ports 10032 10034
outputs 10023-3-2 10043-4-4

file: config/4.cfg
router-id 4
input-ports 10043 10045 10047
outputs 10034-4-3 10054-2-5 10074-6-7

file: config/5.cfg
router-id 5
input-ports 10054 10056
outputs 10045-2-4 10065-1-6

file: config/6.cfg
router-id 6
input-ports 10061 10065
outputs 10016-5-1 10056-1-5

file: config/7.cfg
router-id 7
input-ports 10071 10074
outputs 10017-8-1 10047-6-4
```