364 Assignment 1 Report

Rip Routing Protocol

James Harris - 31202223
Lei Li - 49955811

1. Answers to questions

1.1. Percentage contribution of each partner, including list of contributions.

Lei Li:
50% contribution.

- Initial startup (socket creation, read configuration, etc.)
- Routing table
- Packet sending
- Packet processing
- Split horizon

James Harris:
50% contribution

- FSM code (states, transitions, etc.)
- Periodic updates
- Bug fixes/implementation improvements
- Report

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

The use of timestamps that are compared every time the EFSM reaches the Check Timers state allows timeouts and periodic updates to be delt with in a synchronous manor. The built-in time variance of this approach is beneficial as it helps to prevent routers from becoming synchronized.

The use of a 0.1 second timeout on socket select to prevent the code from becoming too CPU expensive. Initially the timeout was set to zero to make the check instantaneous, however, this caused the program to run through the states very fast, causing heavy CPU usage. By making the timeout 0.1 seconds, the program becomes dramatically less expensive to run, but still converges in a comparable similar time.

The design of the transitions of the Extended Finite State Machine, as discussed in 1.4.

1.3. Which aspects of your overall program (design or implementation) could be improved?

The router class is very large, containing many methods and variables, and some ifs that are deeply nested. This could be improved by splitting the router class up into some smaller classes. Another possible improvement would be to split up the nested if statements, so they are not so deep. These changes would help with the readability of the code.

1.4. How have you ensured atomicity of event processing?

The use of an extended finite state machine allows each event to be processed atomically, as other events cannot start processing until the current one has finished.

The design of the transitions of the EFSM, as seen in Figure 1, is such that getting stuck in detrimental loops is prevented. For example, if read messages returned to check sockets, then if a constant stream of messages arrived, the two states would get stuck in a loop and timers never get checked. By having Read Message move to Check Timers, we prevent this loop from occurring. A similar loop is avoided between Check Timers and Send Message.

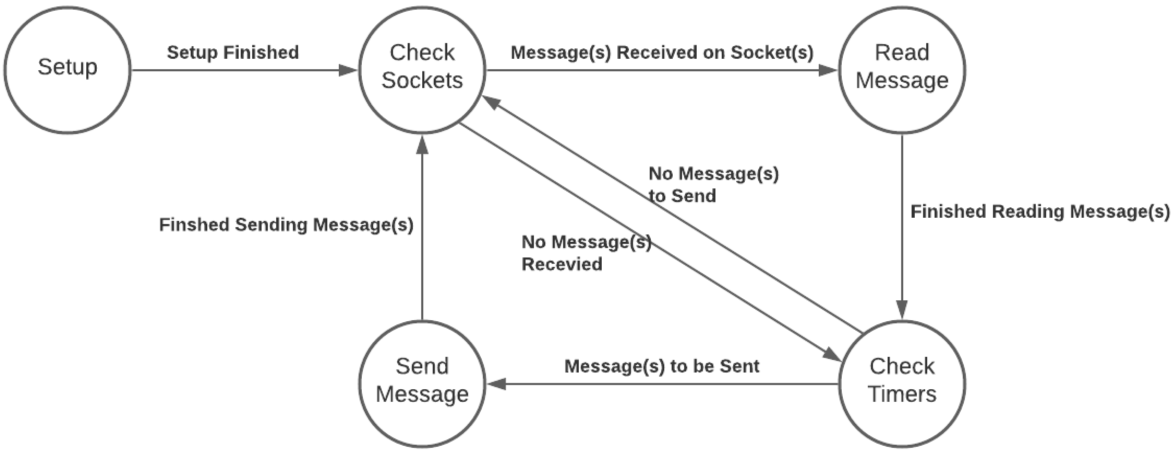

Figure 1 - Rip Router FSM Design

1.5. Have you identified any weakness of the RIP routing protocol?

The RIP protocol has no concept of how links are doing, other than them being alive or dead. This means that the protocol could find a slow link as the shortest hop path, which might not necessarily be the fastest path. There is also no-load balancing, as only one route to each destination is stored, meaning packets cannot be spread over different paths.

1.6. Discussion of Testing

Testing was performed on the RIP Routing Protocol implementation’s ability to converge initially in a series of network topologies, and its ability to reconverge after changes to the network’s topology, with routers being removed or added.

The periodic updates time (and as they are multiples of periodic updates time, the timeout and garbage collection times) was set to be very small, such as 5 seconds, to allow the network to converge very quickly for the sake of testing.

For most tests, the example network topology given in the assignment handout, seen in Figure 2, was used.

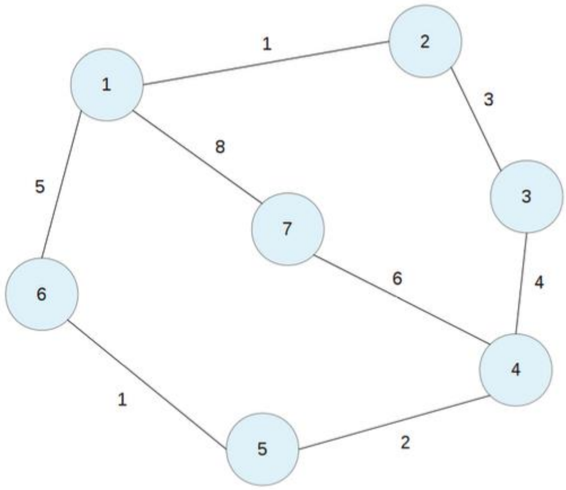

Figure 2: Example network for demonstration, sourced from assignment handout.

1.6.1 Convergence in the standard network

This test was to see if the protocol implementation would converge correctly in the standard network topology seen in Figure 2. The expected outcome of this test was manually calculated in the form of the expected routing tables after convergence. The actual routing tables after convergence has occurred were compared to the expected to see if the implementation was converging correctly. At points during development, they would not converge correctly which would indicate a problem with the implementation.

Additionally, this test covered the stability of the implantation by checking that a point of convergence was reached after some time, with the routing tables remaining the same.
At one point during development, before the introduction of split horizon with poison reverse and the introduction of variance to the timing of sending periodic updates, routers would occasionally get synchronized and mutually deceive each other in a loop.

1.6.2 Shutting down of routers

After convergence was achieved in the standard network routers would be shut off to see if the network would handle their disappearance correctly. Multiple tests were completed by shutting off different combinations of router(s). The expected result of each change would be calculated and compared with the actual result to see if the program was reconverging correctly.

The timeout time and garbage time were observed during reconvergence to check if routers were learning about the shutting off of a router properly. Oftentimes when there were errors in the implementation, the timers would start counting up in response to a router being shut off and then be reset, after being deceived by a message from another router.

Additionally, the time in which it took for the reconvergence to occur was observed, to see if it was converging as fast as expected. Unexpectedly long reconvergence was sign that something was going wrong, such as routers being deceived about the existence of a router that has been shut off. Split horizon with poison reverse was observed to reduce this reconvergence time greatly.

1.6.3 Bringing routers back up

After routers were shut down and the routing tables had reconverged to the expected state, the taken down router(s) would be brought back online. The convergence of the network back to the expected state was tested by comparing the expected outputs of the routing tables to the actual ones. This allowed for testing on whether the implementation could correctly handle routers coming back online. Also, whether it can handle new routers being brought online after initial convergence, as these are essentially the same case.

1.6.4 Restarting a router before it has been completely forgotten

This test was used to determine whether the implementation is capable of handling a router going offline and coming back online before it has been completely forgotten from the network. Router(s) would be taken offline and brought back online after different amounts of time (once the garbage flag had been set in the neighbor, once the garbage flag had been propagated through the network, once the neighbors had forgotten about the router but some other routers in the network had not yet) and the behavior of the network would be observed. It is expected that once the router is brought back online, its existence should propagate though the network. All that still know of the router should reset the timeout timer, garbage timer, and garbage flag for the route to the router. All those who had forgotten the router should relearn of its existence.

1.6.5 Isolated routers

This test was simply activating one router with no neighbors. It is expected that the routers routing table is empty and remains that way. The actual routing table was checked to be empty. This test actually failing for most of the development time as the router would populate its routing table from its own configuration file, however this was fixed. This test is used to ensure that the router starts up with the correct (empty) routing table.

1.6.6 Routers in pairs

This test is activating neighboring routers. The expected routing table is just one route to the neighbor with the correct metric and that neighbor as the next hop. This was compared to the actual routing table to determine if the test passes. This test is used to check if two neighbors are communicating with each other correctly and are capable of acknowledging the others existence even with poison reverse.

1.6.7 Routers >15 metric distance away

This test involves setting up a network with routers that are connected but >15 metric distance away from each other. An example of how this was done is by shutting off routers 5 and 2 in the example network from Figure 2. The expected outcome is that routes with a metric of over 15 do not appear in the routing tables. For example, router 1 should know router 4 exists and has a cost of 14 but should not know router 3 exists, as the path would be 18 which is greater than 15. This test ensures that the 15-hop max (or whatever maximum distance is set in the configuration) is actually enforced.

1.6.8 Stub Routers

Another network configuration that was tested was stub routers. This is when a router has only one neighbor. An example of this is in the example network in Figure 2 shutting down router 6, which makes router 5 a stub network off of router 4. Router 5 then has only router 4 to learn about the network's topology. It is expected that the network will reconverge, with router 5 forgetting about router 6 and learning new paths to router 1 and router 2. This test allows us to ensure that stub routers learn correct information from their neighbor. An additional test of shutting down the stub routers only neighbor and checking to see whether or not the stub forgot about the entire network was employed.

1.6.9 Multiple equivalent choices

This test is used to determine if routers handle multiple same length paths correctly. It is expected that the router will take the first shortest path it hears about, and then not change, as the other path although equivalent in length is not better so is disregarded. The example network from Figure 2 is already setup to test his, with equal cost paths from between routers 1 and 4 on both sides of the loop. The correct behavior is tested by starting the routers up in an order that means the routers both choose one path (for example starting all routers expect 6 and then starting 6) and checking that they stick to this path and do not change when the alternative equivalent path is heard about.

1.6.10 Conclusions of Testing

The broad range of tests used allow the correctness of the behavior implementation to be verified. Tests for convergence, handling of the shutdown and restart of routers, handling of greater than 15 hop routes, and other edge cases are checked for. However, these tests only check that the routing protocol behaves in the expected way, they do not check whether the implementation actually follows the RIP routing protocol.

2. Example config file:

The following is an example of the contents of the router configuration file for router 1. The format is Json and uses three parameters. First the router_id which is the id of the router and has an integer value. Second is the input_ports field, which is an array of ports we expect to receive packets from. Lastly is the outputs field, which is an object containing pairs of output port ids and arrays the format [metric, neighboring router id].

```
{
   "router_id": 1,
   "input_ports": [6002, 6006, 6007],
   "outputs": {"8002": [1, 2], "8006": [5, 6], "8007": [8, 7]}
}
```

3. Source Code:

```python
#!/usr/bin/env python3
import sys
import time
import json
import copy
from socket import *
from select import select
from random import randint

HOST = '127.0.0.1'  # localhost
INFINITY = 16

PERIODIC_UPDATES = 4
TIMEOUTS = PERIODIC_UPDATES * 6
GARBAGE_COLLECTION = PERIODIC_UPDATES * 4

DEBUG_MODE_VERBOSITY = 0


# https://www.youtube.com/watch?v=E45v2dD3IQU for how to make FSM in
python


class State(object):
    """ Generic state class """

    def __init__(self, fsm):
        self.fsm = fsm

    def enter(self):
        pass

    def execute(self):
        pass

    def exit(self):
        pass


class SetUp(State):
    """ Setup state of FSM """

    def __init__(self, fsm):
        super(SetUp, self).__init__(fsm)

    def execute(self):
        print_debug_message("Performing setup", 4)

        self.fsm.configuration()
        self.fsm.socket_setup()
        # self.fsm.routing_table_setup()

        # Transition to to_check_sockets
        self.fsm.to_transition("to_check_sockets")

    def exit(self):
        print_debug_message("Finished setup", 4)


class CheckSockets(State):
```

```python
    """ Check sockets state of FSM """

    def __init__(self, fsm):
        super(CheckSockets, self).__init__(fsm)

    def enter(self):
        print_debug_message("Checking sockets", 4)

    def execute(self):
        msg_received = self.fsm.check_receive()

        if msg_received:
            self.fsm.to_transition("to_reading_message")
        else:
            self.fsm.to_transition("to_check_timers")

    def exit(self):
        print_debug_message("Finished checking sockets ", 4)


class ReadingMessage(State):
    """ Reading message state of FSM """

    def __init__(self, fsm):
        super(ReadingMessage, self).__init__(fsm)

    def enter(self):
        print_debug_message("Reading message(s)", 4)

    def execute(self):
        self.fsm.process_res_pkt()
        self.fsm.to_transition("to_check_timers")

    def exit(self):
        print_debug_message("Message(s) read", 4)


class SendingMessage(State):
    """ Sending message state of FSM """

    def __init__(self, fsm):
        super(SendingMessage, self).__init__(fsm)

    def enter(self):
        print_debug_message("Sending message", 4)

    def execute(self):
        self.fsm.send_packet()
        self.fsm.to_transition("to_check_sockets")

    def exit(self):
        print_debug_message("Message sent", 4)


class CheckTimers(State):
    """ Check timers state of FSM
        Check the different types of timers (periodic timer, timeout
timer, garbage collection timer)
        and handle events appropriately"""

    def __init__(self, fsm):
```

```
        super(CheckTimers, self).__init__(fsm)

    def enter(self):
        print_debug_message("Checking timers", 4)

    def execute(self):
        self.fsm.check_time_stamp()

        if self.fsm.should_send_message:
            self.fsm.should_send_message = False
            self.fsm.to_transition("to_sending_message")
        else:
            self.fsm.to_transition("to_check_sockets")

    def exit(self):
        print_debug_message("Finished checking timers", 4)


class Transition(object):
    """ Transition for FSM """

    def __init__(self, to_state):
        self.to_state = to_state

    def execute(self):
        """ Execute this transition """
        print_debug_message("Transitioning to state
{}".format(self.to_state), 4)


class Fsm(object):
    """ A finite state machine"""

    def __init__(self):
        # self.router = router
        self.transitions = {}
        self.states = {}
        self.curState = None
        self.transition = None

    def add_transition(self, transition_name, transition):
        """ Add transition to FSM """
        self.transitions[transition_name] = transition

    def add_state(self, state_name, state):
        """ Add state of FSM """
        self.states[state_name] = state

    def set_state(self, state_name):
        self.curState = self.states[state_name]

    def to_transition(self, to_transition):
        self.transition = self.transitions[to_transition]

    def execute(self):
        if self.transition:
            self.curState.exit()
            self.transition.execute()
            self.set_state(self.transition.to_state)
            self.curState.enter()
            self.transition = None
```

```
        self.curState.execute()


class RipRouteEntry:
    """ A single RIP route entry"""

    def __init__(self, dest_id, next_hop, metric):
        """ Initialization"""

        self.dest_id = dest_id
        self.next_hop = next_hop
        self.metric = metric

        self.timeout_timestamp = time.time()  # timeout_timestamp of
current time

        self.garbage_timestamp = 0
        self.garbage = False

    def to_json(self):
        return "{{\"dest_id\": {}, \"next_hop\": {}, \"metric\":
{}}}".format(self.dest_id, self.next_hop, self.metric)


class Router(Fsm):
    """ Router"""

    def __init__(self, config_filename):
        """ Initialisation"""

        super().__init__()
        self.router_id = None
        self.input_ports = []
        self.outputs = {}
        self.conf = {}
        self.sockets = []
        self.received_datagram = []
        self.routing_table = {}
        # self.res_skt init in function socket_setup
        self.res_skt = None
        self.neighbours_id = []
        self.config_filename = config_filename

        self.time_of_last_periodic = time.time()
        self.should_send_message = False

        self.next_periodic_update_timeout = PERIODIC_UPDATES

        # States
        self.add_state("SetUp", SetUp(self))
        self.add_state("CheckSockets", CheckSockets(self))
        self.add_state("ReadingMessage", ReadingMessage(self))
        self.add_state("CheckTimers", CheckTimers(self))
        self.add_state("SendingMessage", SendingMessage(self))

        # Transitions
        self.add_transition("to_check_sockets",
Transition("CheckSockets"))
        self.add_transition("to_reading_message",
Transition("ReadingMessage"))
        self.add_transition("to_check_timers",
```

```
Transition("CheckTimers"))
        self.add_transition("to_sending_message",
Transition("SendingMessage"))

        self.set_state("SetUp")

    def configuration(self):
        """Read config"""
        try:
            with open(self.config_filename, "r") as read_file:
                self.conf = json.load(read_file)
        except FileNotFoundError:
            print("Configuration failed. No such file or directory.")
        else:
            # print(self.conf)

            if 'router_id' in self.conf:
                self.get_id()
            else:
                print("router_id is required")
                sys.exit()

            if 'input_ports' in self.conf:
                self.get_input_ports()
            else:
                print("input_ports is required")
                sys.exit()

            if 'outputs' in self.conf:
                self.get_outputs()
            else:
                print("outputs is required")
                sys.exit()

            print("configuration file uploaded successful")
            print(self)
            print("neighbours: {}".format(self.neighbours_id))

    def run(self):
        """ Main loop """
        while True:
            self.execute()

    def get_id(self):
        """ Read the router if from the configuration file"""
        if 1 <= self.conf["router_id"] <= 64000:
            self.router_id = self.conf["router_id"]
        else:
            print("Invalid Router ID Number")
            sys.exit()

    def get_input_ports(self):
        """ Read input ports from the configuration file"""
        for port_num in self.conf["input_ports"]:
            if 1024 <= port_num <= 64000 and port_num not in
self.input_ports:
                self.input_ports.append(port_num)
            else:
                print("Invalid Port Number found")
                sys.exit()
```

```python
    def get_outputs(self):
        """ Read outputs from the configuration file"""
        for key, value in self.conf["outputs"].items():
            try:
                output_port = int(key)
                metric = value[0]
                peer_router_id = value[1]
            except Exception:
                print("Invalid configuration file syntax.")
                sys.exit()
            else:
                if (1024 <= output_port <= 64000
                        and 1 <= metric < INFINITY
                        and 1 <= peer_router_id <= 64000
                        and output_port not in self.outputs):
                    self.outputs[output_port] = [metric, peer_router_id]
                    self.neighbours_id.append(peer_router_id)
                else:
                    print("Invalid outputs value.")
                    sys.exit()

    def __repr__(self):
        """Configuration details of a router"""
        return ("Router {} created\nInput Ports: {}\nOutputs: {}"
                .format(self.router_id, self.input_ports, self.outputs))

    def socket_setup(self):
        """ Bind one socket to each input port
        and specify one of the input sockets to be used for sending UDP
packets to neighbours"""

        # create socket for each input port
        for i in range(len(self.input_ports)):
            try:
                udp_socket = socket(AF_INET, SOCK_DGRAM, 0)
            except Exception:
                print("Failed to create socket.")
                sys.exit()
            else:
                self.sockets.append(udp_socket)
                print("Socket {} created.".format(self.input_ports[i]))

        # bind port to socket
        if len(self.sockets) == len(self.input_ports):
            for i in range(len(self.sockets)):
                try:
                    self.sockets[i].bind((HOST, self.input_ports[i]))
                except Exception:
                    print("ERROR on binding.")
                    sys.exit()
                else:
                    self.res_skt = self.sockets[0]
                    print("Socket {} bind
complete.".format(self.input_ports[i]))
        else:
            print("Socket setup failed.")
            sys.exit()

    # When we switch a router on, the routing table should be empty, so
we don't need this function
    # def routing_table_setup(self):
```

```
    #     """ Init routing table"""
    #     for key, value in self.outputs.items():
    #         init_entry = RipRouteEntry(value[1], value[1], value[0])
    #         self.routing_table[value[1]] = init_entry
    #     self.print_routing_table()

    def check_receive(self):
        """ receive packets.
            Returns boolean: True if packet(s) received, False if no
packet"""

        # 1 second timeout on select to prevent the program from
becoming cpu intensive
        readable, writable, exceptional = select(self.sockets, [], [],
1)
        print_debug_message("Readable" + str(readable), 3)
        if readable:
            for sock in readable:
                data, src_addr = sock.recvfrom(1024)
                self.received_datagram.append((data, src_addr))
            print_debug_message(self.received_datagram, 3)

            return True
        else:
            return False

    def respond_packet(self, copy_of_routing_table):
        """format: json string
        {src_id: xxx,
         entries:
             [
                 entry1,
                 entry2,
                 ...
             ]
        }"""

        pkt = "{" + "\"src_id\": {},".format(self.router_id) + "}"

        if len(copy_of_routing_table) > 0:
            pkt = "{" + "\"src_id\": {},".format(self.router_id) +
"\"entries\": ["

            for dest_id in copy_of_routing_table.keys():
                pkt = pkt + copy_of_routing_table[dest_id].to_json() +
","
            pkt = pkt[:-1] + "]}"

        print_debug_message("respond pkt", 2)
        print_debug_message(pkt, 2)

        return pkt.encode('utf-8')

    def send_packet(self):
        """ send a update packet to all neighbours """
        for key, value in self.outputs.items():
            current_version = self.split_horizon(value[1])
            pkt = self.respond_packet(current_version)

            self.res_skt.sendto(pkt, (HOST, key))
```

```
    def split_horizon(self, neighbour_id):
        """Set the metrics of the routes learned from neighbour to
infinity"""
        copy_of_routing_table = copy.deepcopy(self.routing_table)

        for dest_id, entry in copy_of_routing_table.items():
            if dest_id != neighbour_id and entry.next_hop ==
neighbour_id:
                entry.metric = INFINITY

        return copy_of_routing_table

    def keep_neighbour_alive(self, neighbour_id):
        """ If neighbour still in routing table, reset the timeout_timer
of that neighbour"""
        if neighbour_id in self.routing_table.keys():
            self.routing_table[neighbour_id].timeout_timestamp =
time.time()

    def process_res_pkt(self):
        """update routing table"""

        for data in self.received_datagram:
            if data[0]:
                msg = eval(data[0].decode('utf-8'))
                print_debug_message("Packet Message Piece: " + str(msg),
1)
                src = msg["src_id"]

                if src in self.neighbours_id:
                    metric_to_neighbour =
self.find_metric_to_neighbour(src)  # update the entry to neighbour and
return the metric
                    self.keep_neighbour_alive(src)
                    if "entries" in msg:
                        for entry in msg["entries"]:
                            dest_id = entry["dest_id"]
                            if dest_id != self.router_id:
                                received_metric = int(entry["metric"])
                                if 1 <= received_metric <= INFINITY:   #
validation
                                    new_metric = min(received_metric +
metric_to_neighbour, INFINITY)
                                    if self.is_new_dest(dest_id):   # if
it is a new route
                                        if new_metric < INFINITY:
                                            # add this route to the
routing table if the metric is not infinity
                                            new_entry =
RipRouteEntry(dest_id, src, new_metric)
                                            self.routing_table[dest_id]
= new_entry

                                    else:  # if there is an existing
route
                                        if new_metric < INFINITY:
                                            print_debug_message(
                                                "Updating existing
entry: {} {} {}".format(dest_id, src, new_metric), 3)

self.update_an_entry(dest_id, src, new_metric)
```

```
                                                    else:
                                                        # Only if received from the
one who sent to use
                                                        # if the new metric is
infinity, update metric
                                                        # set the garbage flag, send
message
                                                        if
self.routing_table[dest_id].next_hop == src and self.routing_table[
                                                            dest_id] \
                                                                .metric != INFINITY:

print_debug_message("Marking for deletion: {}".format(dest_id), 3)
                                                            self.should_send_message
= True

self.mark_entry_for_deletion(dest_id)

            self.received_datagram.remove(data)
            self.print_routing_table()

    def find_metric_to_neighbour(self, neighbour_id):
        """if there's not an entry to the given neighbour, create one;
         if there's an entry to the given neighbour, update the time
stamp;
         return the metric"""
        found = False
        metric = None

        for value in self.outputs.values():
            if neighbour_id == value[1]:
                metric = value[0]

        for entry in self.routing_table.values():
            if entry.dest_id == neighbour_id:
                found = True
                entry.metric = metric
                # we update the timeout timer in keep_neighbour_alive
function already
                # entry.timeout_timestamp = time.time()

        if not found:
            new_entry = RipRouteEntry(neighbour_id, neighbour_id,
metric)
            self.routing_table[neighbour_id] = new_entry

        return metric

    def is_new_dest(self, dest_id):
        """check if an destination is in the routing table"""
        if dest_id != self.router_id:
            for dest_id_current in self.routing_table.keys():
                if dest_id_current == dest_id:
                    return False
            return True
        else:
            return False

    def update_an_entry(self, dest_id, next_hop, new_metric):
        """update an old entry"""
```

```
        if dest_id != self.router_id:
            entry = self.routing_table[dest_id]

            if entry.next_hop == next_hop:  # the datagram is from the
same router

                entry.timeout_timestamp = time.time()  # update
timeout_timestamp in no condition

                if entry.metric != new_metric:
                    # if the datagram is from the same router and the
metric is different
                    entry.metric = new_metric

            else:
                if entry.metric > new_metric:  # if the datagram is from
another router and the new metric is
                    # less than the current route

                    # Update the route (timer, metric and next hop)
                    entry.timeout_timestamp = time.time()  # update
timeout_timestamp
                    entry.metric = new_metric
                    entry.next_hop = next_hop

    def mark_entry_for_deletion(self, dest_id):
        """start the deletion process if the new metric is infinity"""

        if dest_id != self.router_id:
            self.routing_table[dest_id].garbage = True
            self.routing_table[dest_id].metric = INFINITY
            self.routing_table[dest_id].garbage_timestamp = time.time()
# update garbage_timestamp for garbage collect

    def print_routing_table(self):
        """Print the routing table, if routing table has changed"""
        print("-" * 10 + "Router " + str(self.router_id) + " Routing
Table" + "-" * 10)
        print(
            "{:<12} {:<9} {:<8} {:<12} {:<12}
{:<10}".format("Destination", "Next Hop", "Metric", "Timeout T",
                                                        "Garbage
T", "Garbage-Flag"))
        for entry in self.routing_table.values():
            print("{:<12} {:<9} {:<8} {:<12} {:<12} {:<10}"
                  .format(entry.dest_id,
                          entry.next_hop,
                          entry.metric,
                          round(time.time() - entry.timeout_timestamp,
8),
                          (round(time.time() - entry.garbage_timestamp,
8), 0)[not entry.garbage],
                          entry.garbage))

    def check_time_stamp(self):
        """Check timestamps and update garbage flag and delete expired
entry if necessary."""

        current_time = time.time()

        # Check if new periodic update should be sent
```

```
        if self.time_of_last_periodic < current_time -
self.next_periodic_update_timeout:
            self.time_of_last_periodic = time.time()
            self.should_send_message = True  # Periodic update timer has
elapsed so we should send message
            print_debug_message("Periodic", 1)
            self.next_periodic_update_timeout = PERIODIC_UPDATES *
(randint(8, 12) / 10)
            self.print_routing_table()

        for dest_id in list(self.routing_table.keys()):
            entry = self.routing_table[dest_id]
            time_since_timeout = int(current_time -
entry.timeout_timestamp)
            time_since_garbage = int(current_time -
entry.garbage_timestamp)

            if not entry.garbage and time_since_timeout > TIMEOUTS:

                entry.garbage = True
                entry.metric = INFINITY
                entry.garbage_timestamp = time.time()  # update
timeout_timestamp for garbage collection
                self.should_send_message = True  # Route metric has been
set to infinity, so we should send an update

            elif (entry.garbage and time_since_garbage <=
GARBAGE_COLLECTION
                  and entry.metric < INFINITY):
                # when a new route replaced the one that is about to be
deleted, clear the garbage flag
                entry.garbage = False
                entry.garbage_timestamp = 0

            elif entry.garbage and time_since_garbage >
GARBAGE_COLLECTION:

                self.routing_table.pop(entry.dest_id)


def print_debug_message(message, verbosity):
    """ Print message with timeout_timestamp if in debug mode"""
    if verbosity <= DEBUG_MODE_VERBOSITY:
        print("{}: {}".format(time.time(), message))


if __name__ == "__main__":
    """start up a router"""

    router = Router(str(sys.argv[-1]))
    router.run()
```

4. Plagiarism Declaration Forms:

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: James Harris

Student ID: 31202223

Signature:

Date: 26/04/2021

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: Lei Li

Student ID: 49955811

Signature: Lei Li

Date: April 26 2021