COSC 364 RIP routing assginemnt

By Cui Yuan, ycu20 (63483319)

and Liu Yihong, yli227(49118489)

- **percentage contribution**
- **good point**
- **What could be improved**
- **Atomicity of event processing**
- **Testing**
- **Configuration file**
- **Source code**

percentage contribution

The percentage contributions for this assignment were roughly 50% for each of us.

Good point

The program has high readability to make people easy to understand. We wrote comment for each part of this program, make people easy to know the function. We strictly comply with the RIPv2 message format like the create header with command, version fields. We defined classes with encapsulation of responsibility for specific functions to makes it more object-oriented and easier to maintain. Also, we set checksum to verify it correctness.

What could be improved

We haven't done anything from chapter 4. We also should reduce the CPU load.

Atomicity of event processing

Atomicity problem occurs during the different router threads updating database, usually synchronized lock will help prevent this issue. However in this case, we didn't use threads. Instead, we use the select() statement to read available updates and process them sequentially. And we wait for the packets to be read so it do not interrupt any other operations to ensure all event processing in order.

Testing

```
destination: 2, first: 1, metric: 1, time: 0.0005946159362792969
destination: 3, first: 2, metric: 4, time: 0.0005900859832763672
destination: 4, first: 6, metric: 8, time: 2.0021913051605225
destination: 5, first: 6, metric: 6, time: 2.0021731853485107
destination: 6, first: 1, metric: 5, time: 2.0022666454315186
destination: 7, first: 1, metric: 8, time: 0.00010609626770019531
```

Fig.1 Router1

```
destination: 1, first: 2, metric: 1, time: 6.556510925292969e-05
destination: 3, first: 2, metric: 3, time: 0.00039744377136230469
destination: 4, first: 3, metric: 7, time: 0.00038266181945800781
destination: 5, first: 1, metric: 7, time: 4.863739013671875e-05
destination: 6, first: 1, metric: 6, time: 4.100799560546875e-05
destination: 7, first: 1, metric: 9, time: 3.361701965332031e-05
```

Fig.2 Router2

First we are going to test the routing table and the resulting routers. We except router1 should reach router 2,6 and 7 first, and metric will be 1, 5 and 6. For router2, it should reach router 1 and 3 first, and the metric will be 1 and 3.

From the fig.1 and fig.2 we can see the metric from router1 to router2 is exactly 1, and we can see the entry and metric for other routers are also correct as we excepted.

After this test, we are going to switch off the router2 and we expect other routers will converge again to become the ‘minimum hop’.

```
<socket.socket fd=5, family=AddressFamily.AF_INET, type=SocketKind.SOCK_DGRAM, p
roto=0, laddr=('127.0.0.1', 1026)>
destination: 2, first: 1, metric: 16, time: 27.049180507659912
destination: 3, first: 6, metric: 12, time: 7.939338684082031e-05
destination: 4, first: 6, metric: 8, time: 7.724761962890625e-05
destination: 5, first: 6, metric: 6, time: 7.486343383789062e-05
destination: 6, first: 1, metric: 5, time: 0.0001475811004638672
destination: 7, first: 1, metric: 8, time: 1.001596450805664

send packets to all outputs
dest 2 metric has become infinity
send packets to all outputs
destination: 2, first: 1, metric: 16, time: 27.050986766815186
-10.018841743469238
dest 2 has been removed
<socket.socket fd=4, family=AddressFamily.AF_INET, type=SocketKind.SOCK_DGRAM, p
roto=0, laddr=('127.0.0.1', 1025)>
destination: 3, first: 6, metric: 12, time: 1.0045225620269775
destination: 4, first: 6, metric: 8, time: 1.0045180320739746
destination: 5, first: 6, metric: 6, time: 1.004512071609497
destination: 6, first: 1, metric: 5, time: 1.0045812129974365
destination: 7, first: 1, metric: 8, time: 0.00010180473327636719
```

Fig.3 Router2 switch off

From fig.3 we can see when we switch off router2, we use split-horizon with poisoned-reverse to avoid looping. The metric become to infinity after time out period, the program find out this network can’t reach router2 so the entry table will remove router2 entry in garbage collection period and set the ‘minimum hop’ to a new path. It means the routers detect failures and recover correctly.

In our final test, we switch on the router2 again and expect that the network converges back into the initial state.

```
<socket.socket fd=4, family=AddressFamily.AF_INET, type=SocketKind.SOCK_DGRAM, p
roto=0, laddr=('127.0.0.1', 1025)>
destination: 3, first: 6, metric: 12, time: 0.00096893310546875
destination: 4, first: 6, metric: 8, time: 0.0009629726409912109
destination: 5, first: 6, metric: 6, time: 0.0009598731994628906
destination: 6, first: 1, metric: 5, time: 0.0010256767272949219
destination: 7, first: 1, metric: 8, time: 0.00011968612670898438

<socket.socket fd=3, family=AddressFamily.AF_INET, type=SocketKind.SOCK_DGRAM, p
roto=0, laddr=('127.0.0.1', 1024)>
destination: 2, first: 1, metric: 1, time: 1.8835067749023438e-05
destination: 3, first: 2, metric: 4, time: 2.3365020751953125e-05
destination: 4, first: 6, metric: 8, time: 0.30132365226745605
destination: 5, first: 6, metric: 6, time: 0.3013112545013428
destination: 6, first: 1, metric: 5, time: 0.30136871337890625
destination: 7, first: 1, metric: 8, time: 0.30045461654663086
```

Fig.4 Router2 switch on

From fig.4 we can see after we switch on router2, this network find out a new path can go through from router2, it re-calculate the paths and metrics and update them into this network. The routers re-converge correctly after revival.

Configuration files

router1.txt

router-id 1

input-ports 1024, 1025, 1026

outputs 1027-1-2, 1039-8-7, 1037-5-6

router2.txt

router-id 2

input-ports 1027, 1028

outputs 1024-1-1, 1029-3-3

router3.txt

router-id 3

input-ports 1029, 1030

outputs 1028-3-2, 1031-4-4

router4.txt

router-id 4

input-ports 1031, 1032, 1033

outputs 1030-4-3, 1038-6-7, 1034-2-5

router5.txt

router-id 5

input-ports 1034, 1035

outputs 1033-2-4, 1036-1-6

router6.txt

router-id 6

input-ports 1036, 1037

outputs 1026-5-1, 1035-1-5

router7.txt

router-id 7

input-ports 1038, 1039

outputs 1032-6-4, 1025-8-1

Source code

```
        import socket
        from random import random
        from time import *
        from hashlib import md5
        import select
        import sys

        # GLOBAL VARIABLES
        INFINITY = 16
        LOCALHOST = "localhost"
        TIME_OUT = 30
        PERIOD = TIME_OUT / 6
        GARBAGE_COLLECTION_TIME = TIME_OUT / 3 * 2


        class Entry(object):
            """entry class"""
            def __init__(self, dest_node, first_node, metric, ti=
        ↳   None, flag=True):
                self.dest_node = dest_node  # destination router id
                self.first_node = first_node  # the first router 
        ↳       id to destination
                self.metric = metric  # metric of the route
                # time value when the entry is updated
                if not ti:
                    self.ti = time()
                else:
                    self.ti = ti
                self.garbage_collection_time = None
                self.flag = flag

            def alive_time(self):
                """return time after last updated"""
                return time() - self.ti

            def reset_time(self):
                """reset time"""
                self.ti = time()

            def __repr__(self):
```

```
            """change object to string"""
            return "destination: " + str(self.dest_node) + ↵
        ↳   ", " + "first: " + str(self.first_node) + ", " + ↵
        ↳   "metric: " + \
                   str(self.metric) + ", " + "time: " + str(↵
        ↳          self.alive_time())


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

            def __repr__(self):
                result = ''
                for entry in self.entries.values():
                    result += entry.__repr__() + "\n"
                return result

            def get_entry(self, dest_node):
                """return a entry by destination node"""
                return self.entries.get(dest_node)

            def update_entry(self, new_entry):
                """compare received entry with current entry, ↵
        ↳       then decide to update or not, return new entry ↵
        ↳       or None"""
                current_entry = self.get_entry(new_entry.dest_node)
                if not current_entry:  # if entry not existed, ↵
        ↳       add it to entry table
                    if new_entry.metric < INFINITY:
                        self.entries.update({new_entry.dest_node:↵
        ↳                new_entry})
                elif current_entry.first_node == new_entry.↵
        ↳       first_node:  # if current first node == new ↵
        ↳       first node, then update
                    if new_entry.metric >= INFINITY:  # for ↵
        ↳           garbage collection
                        new_entry.flag = current_entry.flag
                        new_entry.garbage_collection_time = ↵
```

```
                current_entry.garbage_collection_time
            self.entries.update({new_entry.dest_node:
            new_entry}))
        elif current_entry.metric > new_entry.metric:  # 
        if new metric < current metric, then update
            self.entries.update({new_entry.dest_node:
            new_entry})

    def remove_entry(self, dest_node):
        """remove entry from entry table"""
        if self.get_entry(dest_node):
            self.entries.pop(dest_node)


class Router(object):
    """router class"""
    def __init__(self, id, inputs, outputs):
        self.id = id
        self.inputs = inputs
        self.outputs = outputs
        self.entry_table = EntryTable()
        self.garbage_collection_time = 0
        self.input_sockets = []
        self.output_socket = None

    @staticmethod
    def create_checksum(payload):
        """create checksum by md5"""
        return md5(bytes(payload, "utf-8")).hexdigest()[:
        10]

    def verify_checksum(self, income):
        """verify check sum return true if correctness 
        otherwise false"""
        return income[:10] == self.create_checksum(income
        [10:])

    @staticmethod
    def create_socket(port_no):
        """create socket"""
```

```
                my_socket = None
                try:
                    my_socket = socket.socket(socket.AF_INET, ↵
↳               socket.SOCK_DGRAM)
                    return my_socket
                except socket.error as e:
                    print(e.strerror)
                    print("Port " + str(port_no) + " cannot be ↵
↳               created")
                    if my_socket:
                        try:
                            my_socket.close()
                        except socket.error:
                            print("Socket with " + str(port_no) +↵
↳                            " cannot be closed")

            def create_sockets(self):
                """create input sockets & out socket"""
                for port_no in self.inputs:
                    my_socket = self.create_socket(port_no)
                    self.input_sockets.append(my_socket)
                if len(self.input_sockets) != 0:
                    for i in range(len(self.input_sockets)):
                        self.input_sockets[i].bind((LOCALHOST, ↵
↳                       self.inputs[i]))
                    self.output_socket = self.input_sockets[0]

            def create_update_packet(self, output):
                """create an update packet
                    HEADER: command(1) version(1) send to router ↵
↳                   id(2)
                    ENTRY: source router id(2) route tag all ↵
↳                   zeros(2)
                           destination router id(4)
                           Subnet mask all zeros(4)
                           next hop(4)
                           metric(4)
                """
                payload = ""
                header = dec_to_bin(2, 8) + dec_to_bin(2, 8) + ↵
```

```
        dec_to_bin(output.dest_node, 16) + "\n"
        payload += header
        for key, entry in self.entry_table.entries.items():
            entry_header = dec_to_bin(self.id, 16) + 
            dec_to_bin(0, 16) + "\n"
            dest_node, metric, first_node = (entry.
            dest_node, entry.metric, entry.first_node)
            # Split Horizon with Poisoned Reverse
            if first_node == output.dest_node:  # if the 
            first node == dest node, then metric = 
            INFINITY
                metric = INFINITY
            entry_body = dec_to_bin(dest_node, 32) + "\n"
             + dec_to_bin(0, 32) + "\n" + dec_to_bin(
            first_node, 32)\
                + "\n" + dec_to_bin(metric, 32) + "\n"
            payload += entry_header + entry_body
        checksum = self.create_checksum(payload)
        return checksum + payload

    def send_packet(self, output):
        """send update packet to output"""
        packet = self.create_update_packet(output)
        self.output_socket.sendto(bytes(packet, 'utf-8'),
         (LOCALHOST, output.port_no))

    def process(self, income):
        """deal with update message
           check if successfully processing return true, 
           else drop and return false
        """
        if not self.verify_checksum(income):
            print("Cannot pass checksum")
            return False
        lines = income.split("\n")
        if len(lines) != 0:
            header = lines[0]
            body = lines[1:]
        else:
            print("no payload")
```

```
                    return False
                if len(body) == 0:
                    print("no entry")
                    return False
                command = bin_to_dec(header[10:18])
                version = bin_to_dec(header[18:26])
                output_router_id = bin_to_dec(header[26:])
                if output_router_id != self.id:
                    print("packet is not for this router")
                    return False
                if command != 2:
                    print("incorrect command")
                    return False
                if version != 2:
                    print("incorrect version")
                    return False
                entry_number = len(body) // 5
                for i in range(entry_number):
                    source_node = bin_to_dec(body[i * 5][:16])
                    output = self.outputs.get(source_node)
                    dest_node = bin_to_dec(body[i * 5 + 1])
                    metric = bin_to_dec(body[i * 5 + 4])
                    if dest_node != self.id:
                        new_metric = metric + output.metric
                        if new_metric > INFINITY:
                            new_metric = INFINITY
                        new_entry = Entry(dest_node, source_node,↲
↳                        new_metric)
                        self.entry_table.update_entry(new_entry)
                    else:
                        if self.entry_table.get_entry(source_node):
                            self.entry_table.get_entry(↲
↳                            source_node).reset_time()  # reset ↲
↳                            time if neighbour router
                        else:
                            new_entry = Entry(source_node, self.↲
↳                            id, metric)
                            self.entry_table.update_entry(↲
↳                            new_entry)  # create entry if not ↲
↳                            existed
```

```
                print(self.entry_table)
                return True

            @staticmethod
            def receive_packet(my_socket):
                """receive packet and read"""
                packet = my_socket.recvfrom(1024 * 4)
                return packet[0].decode(encoding='utf-8')  # ↵
↳           convent md5 to string

            def send_packets_by_outputs(self):
                """Send an update to all routers in outputs"""
                print("send packets to all outputs")
                for key in self.outputs.keys():
                    self.send_packet(self.outputs.get(key))

            def timeout(self):
                """set time out entry metric to infinity"""
                for entry in self.entry_table.entries.values():
                    if entry.alive_time() > TIME_OUT:
                        entry.metric = INFINITY
                        print("dest {} metric has become ↵
↳                       infinity".format(entry.dest_node))
                        self.send_packets_by_outputs()

            def garbage_collection(self):
                """remove expired entry from table"""
                keys = []
                for key, entry in self.entry_table.entries.items():
                    if entry.flag:
                        if entry.metric >= INFINITY:
                            print("GC start")
                            entry.garbage_collection_time = time()
                            entry.flag = False
                        elif entry.garbage_collection_time and time()↵
↳                        - entry.garbage_collection_time > ↵
↳                       GARBAGE_COLLECTION_TIME and\
                                not entry.flag:
                            print(entry)
                            print(entry.garbage_collection_time - ↵
```

```
                time())
                keys.append(key)
        for k in keys:
            self.entry_table.remove_entry(k)
            print("dest {} has been removed".format(k))


class Output(object):
    """output object"""
    def __init__(self, port_no, metric, dest_node):
        self.port_no = int(port_no)
        self.metric = int(metric)
        self.dest_node = int(dest_node)


def dec_to_bin(dec_num, number):
    """convert decimal to binary"""
    bin_num = bin(int(dec_num))
    return "0" * (number - len(bin_num) + 2) + bin_num[2:]


def bin_to_dec(bin_num):
    """convert binary to decimal"""
    return int(bin_num, 2)


def create_router(config_file):
    """create router by config file"""
    lines = config_file.readlines()
    router_id = int(lines[0].replace("router-id ", "").strip('\n'))
    input_ports = []
    inputs_str = lines[1].replace("input-ports ", "").strip('\n')
    inputs = inputs_str.split(", ")
    for ip in inputs:
        input_ports.append(int(ip))
    outputs = {}
    outputs_str = lines[2].replace("outputs ", "").strip('\n')
```

```
            output_list = outputs_str.split(", ")
            entries = {}
            for op in output_list:
                elements = op.split("-")
                port_no = int(elements[0])
                metric = int(elements[1])
                dest_node = int(elements[2])
                output = Output(port_no, metric, dest_node)
                outputs.update({dest_node: output})
                entry = Entry(dest_node, router_id, metric)
                entries.update({dest_node: entry})
            new_router = Router(router_id, input_ports, outputs)
            new_router.entry_table.entries = entries
            return new_router


        def main():
            config_name = sys.argv[1]
            # config_name = "router1.txt"
            with open(config_name, 'r') as config_file:
                router = create_router(config_file)
            router.create_sockets()
            router.send_packets_by_outputs()
            t = time()
            random_period = (random() * 0.4 + 0.8) * PERIOD
            while True:
                if time() - t >= random_period:  # periodically ↵
        ↳       send updates to output
                    router.send_packets_by_outputs()
                    t += random_period
                    # print(random_period)
                    random_period = (random() * 0.4 + 0.8) * ↵
        ↳           PERIOD  # change random period when send ↵
        ↳           updates
                    router.timeout()
                    router.garbage_collection()
                try:
                    r_list, w_list, e_list = select.select(router↵
        ↳           .input_sockets, [], [], 1)
                    if r_list:
```

```
                    for read in r_list:
                        print(read)
                        packet = router.receive_packet(read)
                        router.process(packet)
            except KeyboardInterrupt:
                print("program has been stopped by Ctrl+C")
                sys.exit()
            except:
                print("Unexpected error: " + sys.exc_info()[0])
                raise


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