C:\Users\uberT\Desktop\cosc364-rip-assignment\rip.py

```
import sys
import socket
import select
import random
import time
from copy import deepcopy
from Config_file_reader import *

BUFFER_SIZE = 1024

routingTable = {}


def createSockets(input_ports):

    input_sockets = []

    for port in input_ports:
        try:
            # Bind a socket for each port on localhostS
            UDP_socket = socket.socket(family=socket.AF_INET, type=socket.
SOCK_DGRAM)
            UDP_socket.bind(("localhost", port))
            input_sockets.append(UDP_socket)
        except:
            print("Error starting socket on port " + str(port))
    return input_sockets


def resetUpdateTimer(updateTimer):
    return time.time() + updateTimer

def checkPeriodicUpdate(nextUpdate):
    # Predicate to see if it is time for a periodic update.
    if time.time() >= nextUpdate:
        return True
    else:
        return False

def resetTimeout(routerId, timeoutTimer):
    routingTable[routerId]["timeout"] = time.time() + timeoutTimer

def checkTimeout(routerId, timeoutTimer, garbageTimer):
    # if the garbage flag is true
    if routingTable[routerId]["garbage"]:
        # if timer has gone off
        if time.time() >= routingTable[routerId]["timeout"]:
            # delete the entry
            del(routingTable[routerId])
    else:
        # timer has gone off
```

```
                if time.time() >= routingTable[routerId]["timeout"]:
                    setGarbage(routerId, garbageTimer)

        def createTableEntry(routerId, metric, nextHopId, nextHop, timeoutTimer):


            data = {
                "metric" : metric,
                "nextHop": nextHop,
                "nextHopId": nextHopId,
                "timeout" : time.time() + timeoutTimer,
                "garbage" : False,
                "infiniteRouteFlag" : False
            }

            routingTable[routerId] = data

        def printTable():
            print(  " ↵
        ↳   +-------------+--------+----------+-------------+---------+---------+----------↵
        ↳   -----------+\n",
                    "| Destination | Metric | Next Hop | Next Hop ID | Timeout | Garbage ↵
        ↳           | Infinite Route Flag |\n",
                    ↵
        ↳           "+-------------+--------+----------+-------------+---------+---------+-↵
        ↳           --------------------+")

            copiedTable = deepcopy(routingTable)
            for link in routingTable:
                data = routingTable[link]
                timeout = int(data["timeout"] - time.time())
                if data["garbage"]:
                    garbage = timeout
                    timeout = '-'
                else:
                    garbage = '-'
                if not data["infiniteRouteFlag"]:
                    infiniteRouteFlag = "Not Set"
                else:
                    infiniteRouteFlag = "Set"
                print(" |{0:^13}|{1:^8}|{2:^10}|{3:^13}|{4:^9}|{5:^9}|{6:^21}|".format(↵
        ↳       link, data["metric"], data["nextHop"], data["nextHopId"], timeout, garbage↵
        ↳       , infiniteRouteFlag))
            print(" ↵
        ↳   +-------------+--------+----------+-------------+---------+---------+----------↵
        ↳   -----------+")

        def createUpdatePacket(senderId, destId, routingTable, triggeredUpdate=False):
            packet = bytearray()
            COMMAND = 2
            VERSION = 2
```

```
            senderIdBytes = senderId.to_bytes(2, 'big')
            packet.append(COMMAND.to_bytes(1, 'big')[0])
            packet.append(VERSION.to_bytes(1, 'big')[0])
            for i in senderIdBytes: packet.append(i)
            for i in routingTable:
                if not triggeredUpdate:
                    createRouteEntry(routingTable, packet, destId, i)
                else:
                    if(routingTable[i]["infiniteRouteFlag"]):
                        createRouteEntry(routingTable, packet, destId, i)
            return packet

        def createRouteEntry(routingTable, packet, destId, routeEntry):
            AFI = 2
            AFIBytes = AFI.to_bytes(2, 'big')
            for i in AFIBytes: packet.append(i)
            for i in range(0, 2): packet.append(0x00)
            routeDestBytes = routeEntry.to_bytes(4, 'big')
            for i in routeDestBytes: packet.append(i)
            for i in range(0, 8): packet.append(0x00)
            if(routingTable[routeEntry]["nextHopId"] == destId):
                metric = 16
                metricBytes = metric.to_bytes(4, 'big')
            else:
                metricBytes = routingTable[routeEntry]["metric"].to_bytes(4, 'big')
            for i in metricBytes: packet.append(i)


        def processPacket(senderPort, packet, timeoutTimer, garbageTimer, outputPorts, ↵
        ↳currentRouterId):
            # discard invalid packets
            if not checkPacket(packet):
                print("discarded an invalid packet")
                return
            entryCount = int((len(packet) - 4) / 20)
            entries = packet[4:]

            senderId = bytearray()
            senderId.append(packet[2])
            senderId.append(packet[3])
            senderId = int.from_bytes(senderId,"big")

            for neighbourPort, neighbourMetric, neighbourRouterId in outputPorts:
                if(senderId == neighbourRouterId):
                    senderMetric = neighbourMetric

            if senderId not in routingTable.keys():
                createTableEntry(senderId, senderMetric, senderId, senderPort, timeoutTimer)
            else:
                for neighbourPort, neighbourMetric, neighbourRouterId in outputPorts:
                    if senderId == neighbourRouterId and neighbourMetric < routingTable[↵
```

```
            senderId]["metric"]:
                routingTable[senderId]["metric"] = neighbourMetric
                routingTable[senderId]["nextHopId"] = neighbourRouterId
                routingTable[senderId]["nextHop"] = neighbourPort
        routingTable[senderId]["garbage"] == False
        resetTimeout(senderId, timeoutTimer)

    for i in range(entryCount):
        destination = bytearray()
        metric = bytearray()
        for j in range(4, 8):
            destination.append(entries[20 * i + j])
        for j in range(16, 20):
            metric.append(entries[20 * i + j])


        destination = int.from_bytes(destination, "big")
        metric = int.from_bytes(metric, "big")

        distance = senderMetric + metric

        if destination in routingTable.keys():

            if distance < routingTable[destination]["metric"] or routingTable[
            destination]["nextHopId"] == senderId:
                routingTable[destination]["infiniteRouteFlag"] = False
                routingTable[destination]["metric"] = distance
                routingTable[destination]["nextHopId"] = senderId
                routingTable[destination]["nextHop"] = senderPort

            if(distance < 16 and routingTable[destination]["nextHopId"] ==
            senderId):
                resetTimeout(destination, timeoutTimer)
                routingTable[destination]["garbage"] = False

        elif(destination != currentRouterId and not distance > 15):
            createTableEntry(destination, distance, senderId, senderPort,
            timeoutTimer)

        try:
            if routingTable[destination]["metric"] > 15 and routingTable[
            destination]["garbage"] == False:
                setGarbage(destination, garbageTimer)
                routingTable[destination]["infiniteRouteFlag"] = True
        except:
            #doesn't work if destination == currentRouterId as it won't have a
            routing entry
            buffer = 0

    def checkPacket(packet):
        entryCount = int((len(packet) - 4) / 20)
```

```
            if packet[0] != 2 and packet[1] != 2:
                return False
            entries = packet[4:]
            for i in range(entryCount):
                metric = bytearray()
                for j in range(16, 20):
                    metric.append(entries[20 * i + j])
                metric = int.from_bytes(metric, "big")
                if metric < 1 or metric > 16:
                    return False

            return True


        def setGarbage(routerId, garbageTimer):
            routingTable[routerId]["garbage"] = True
            routingTable[routerId]["timeout"] = time.time() + garbageTimer
            routingTable[routerId]["metric"] = 16



        def main():

            routerId, inputPorts, outputPorts, timerValues = readConfig(sys.argv[1])
            """
            print("Input Ports:", end=" ")
            for i in inputPorts: print(i, end=", ")
            print()
            print("Output Ports:", end=" ")
            for i in outputPorts: print(i, end=", ")
            print()
            print("Timer Values:", timerValues)
            """
            inputSockets = createSockets(inputPorts)
            outputSocket = [inputSockets[0]]
            updateTimer, timeoutTimer, garbageTimer = timerValues
            nextUpdate = resetUpdateTimer(updateTimer)
            incomingQueue = []

            while True:
                read, write, special = select.select(inputSockets, outputSocket, [])

                for i in read:
                    try:
                        data, addr = i.recvfrom(BUFFER_SIZE)
                        incomingQueue.append((data, addr))
                    except:
                        # print("Error recieving a packet")
                        buffer = 0

                if checkPeriodicUpdate(nextUpdate):
```

```
                    # send update
                    nextUpdate = resetUpdateTimer(updateTimer)
                    print(" Router ID:", routerId)
                    printTable()
                    j = 0
                    for i in outputPorts:
                        bytesToSend = createUpdatePacket(routerId, i[2], routingTable)
                        inputSockets[j].sendto(bytesToSend, ('localhost', i[0]))
                        j += 1

                for message in incomingQueue:
                    data, addr = incomingQueue.pop(0)
                    port = addr[1]
                    processPacket(port, data, timeoutTimer, garbageTimer, outputPorts, ↵
        ↳           routerId)

                sendTriggeredUpdate = False
                for i in routingTable:
                    if(routingTable[i]["infiniteRouteFlag"] == True):
                        sendTriggeredUpdate = True
                if(sendTriggeredUpdate):
                    j = 0
                    for i in outputPorts:
                        bytesToSend = createUpdatePacket(routerId, i[2], routingTable, ↵
        ↳               sendTriggeredUpdate)
                        inputSockets[j].sendto(bytesToSend, ('localhost', i[0]))
                        j += 1
                    for i in routingTable: routingTable[i]["infiniteRouteFlag"] = False

                for entry in deepcopy(routingTable):
                    checkTimeout(entry, timeoutTimer, garbageTimer)




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