COSC 364

Internet Technologies and Engineering

First Assignment

Jemin Lee

Xiang Ji 22246256

The percentage contribution:

Jemin Lee 50%

Xiang Ji 50%

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

We used a function to calculate the time. This way implements invalid timer and flush timer, includes start and cancel. Another thing we think is doing well is we used a function to check the configuration file, collect information and collect neighbor id and destination port in separate lists. The program is simple to understand, readable and maintainable. Each method implements only one function, such as add\_new\_route(),update\_route(),flush\_() and invalidate\_(). We have a recieve\_message function which to exclude routes that have the same destination port as the destination you are sending to. Another update\_table function is used to update table and creates a new instance of timer for running the invalidate and flush function.

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

How have you ensured atomicity of event processing?

We have a bind\_socket function for each port. In this way it guarantees every process is respective.

Our testing plans：

In fact, we did a lot of small tests when we wrote the program. In fact, we did a lot of small tests when we wrote the program. We will add print statements to display the results and compare them with the expected results to determine whether the program is right or not.

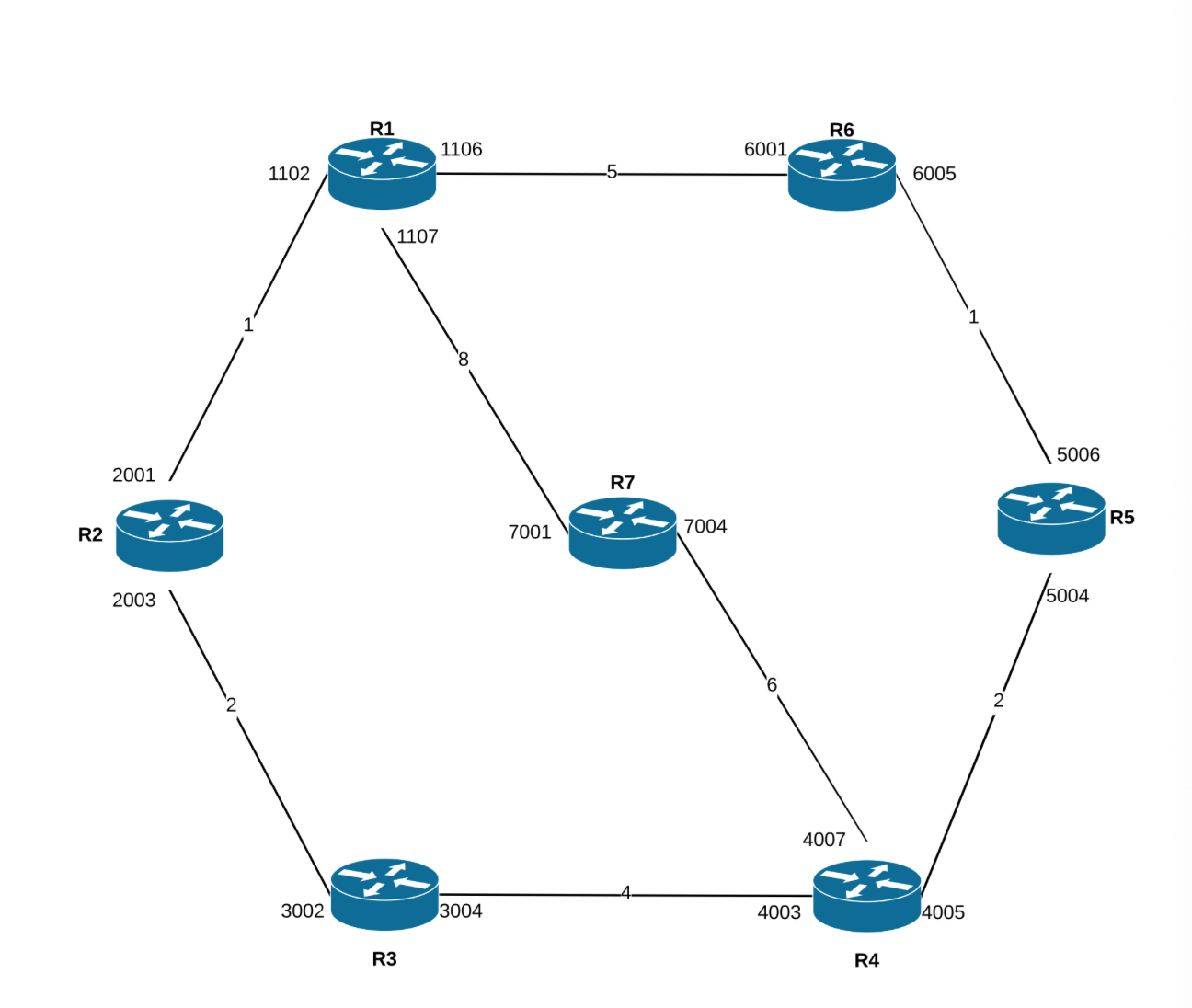
When we are ready, we start to do a lot of tests to check whether the program is working and performing correctly. First we check if the program can read all configure of router, then we check if we can create information and send it to each other. The next step we fixed some problems that occurred during the period.

For test update\_table function, start only two routers, let one router learn a new route, and then turn off the other router. So we can see if the router invalidate, and flush the aged route

The next test is to remove one or two routers. Through this test, if the routers can communicate with each other, then we can confirm that the program is correct. Then, the remaining routers can converge and create a new link table. After conducting this test, we found that after removing some routers, the desired results were not achieved. Therefore, we try to find the wrong code and fix it.

In the final test, we reopened the closed router. Then we observed that the router reached the expected goal.

network for demonstration :



Router.py

import configparser

import pickle

import sys

import select

import time

from threading import Timer

import socket

from datetime import datetime

class routeTimer(object):

'''

this class is used to implement

- invalid timer

- flush timer

'''

def \_\_init\_\_(self, interval, f, \*args, \*\*kwargs):

'''

parameter:

- interval, time it should wait before running the function

- function

- args, parameter that the function processes

'''

self.interval = interval

self.f = f

self.args = args

self.kwargs = kwargs

self.timer = None

def call(self):

self.f(\*self.args, \*\*self.kwargs)

def cancel(self):

'''

stop the timer

'''

if self.timer is not None:

self.timer.cancel()

else:

pass

def start(self):

'''

start the timer

'''

self.timer = Timer(self.interval, self.call)

self.timer.start()

class RIP\_demon(object):

'''

router class

'''

def \_\_init\_\_(self, file):

self.file = file

self.router\_id = None

self.ingress = []

self.ingress\_sockets = []

self.neighbor\_port = []

self.neighbor\_id = []

self.routes = {}

self.triggered\_message = []

self.config = configparser.ConfigParser()

self.drop = []

self.learned\_routers = []

self.route\_message = None

self.sender\_id = None

self.current\_table = None

self.invalid\_timer = {}

self.flush\_timer = {}

def load\_startup(self):

'''

reads the initial config file

'''

self.config.read(self.file)

#does checks for the config file

if len(self.config.get("router-id", "id")) < 1:

print('\nInvalid router id')

return

if len(self.config.items("input-ports")) < 1:

print('\nInvalid input-ports')

return

if len(self.config.items("output-ports")) < 1:

print('\nInvalid output-ports')

return

else:

self.router\_id = int(self.config.get("router-id", "id"))

#collects input ports into list

for key,value in self.config.items("input-ports"):

self.ingress.append(value)

#collect neighbor id and destination port in seperate lists

for key,value in self.config.items("output-ports"):

line\_in\_output = value.split('-')

self.neighbor\_id.append(line\_in\_output[0])

self.neighbor\_port.append(line\_in\_output[2])

sender\_router = 'router{}'.format(line\_in\_output[0])

def show\_routes(self):

'''

reads all existing routes in the output ports and format them in a way that is readable and print

'''

now = datetime.now().time()

print("=================Show Routes=================")

print(now)

print('Router ID: {}'.format(self.router\_id))

for key, value in self.config.items("output-ports"):

line\_in\_output = value.split('-')

ID = line\_in\_output[0]

metric = line\_in\_output[1]

out\_port = line\_in\_output[2]

next\_hop = line\_in\_output[3]

if metric == "16" and out\_port == "N/A" and next\_hop == "N/A":

print('route to router {} possibly down'.format(ID))

else:

if next\_hop == "N/A":

print(ID + ' directly connected, ' + out\_port)

else:

print(ID + ' reachable via Port ' + out\_port + ', Next Hop: ' + next\_hop + ' Metric ' + metric)

def create\_message(self, dest\_port):

'''

create message

exclude routes that have the same destination port as the destination you are sending to

'''

source = {}

routes = {}

source.update({self.router\_id:"update"})

for key,value in self.config.items("output-ports"):

'''

exclude if those routes were learnt, or configured via a port,

that the message is destined to the same port.

(implementing split horizon)

'''

if dest\_port in value:

pass

else:

routes.update({key:value})

message = []

message.append(source)

message.append(routes)

return message

def send\_message(self):

'''

sending messages to neighbor ports

'''

now = datetime.now().time()

print("==========Send Message===========")

for port in self.neighbor\_port:

#create message

message = self.create\_message(port)

update\_message = pickle.dumps(message)

#sending router's whole output contents in pickle

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sock.sendto(update\_message, ("127.0.0.1", int(port)))

print("Update message sent to port " + port + ", " + str(now))

def bind\_socket(self):

for port in self.ingress:

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sock.bind(("127.0.0.1", int(port)))

self.ingress\_sockets.append(sock)

def recieve\_message(self):

print("==========Ports Open===========")

print("listening for messages...")

#wait for 30 seconds for messages

message, \_, \_ = select.select(self.ingress\_sockets, [], [], 12)

#current set to now

current\_time = time.time()

#iterate through the current message

for s in message:

print("===============Ports Closed===============")

data, addr = s.recvfrom(1024)

#load the message

message\_data = (pickle.loads(data))

#set the current sender\_id

'''

example:

self.message = [{2:"update message"}, {router1:route1, router2:route2}]

'''

self.sender\_id = list(message\_data[0].keys())[0]

update = message\_data[1]

print('message from {},'.format(self.sender\_id))

print(update)

for r\_id in update:

self.route\_message = update[r\_id]

self.update\_table()

def flush\_(self, router):

'''

flush a route

'''

now = datetime.now().time()

print("===========Flush Route===========")

self.config.remove\_option("output-ports", router)

print(now)

print('route to {} deleted'.format(router))

def invalidate\_(self, router):

'''

invalidate a route

calls flush\_timer

'''

now = datetime.now().time()

invalidated\_route = '{}-{}-{}-{}'.format(router[-1], 16, 'N/A', 'N/A')

self.config.set("output-ports", router, invalidated\_route)

print(now)

print('route invalidated: {}'.format(invalidated\_route))

#call flush timer

self.flush\_timer[router].start()

def add\_new\_route(self, r\_id, new\_cost, port, sender\_id):

new\_route = '{}-{}-{}-{}'.format(r\_id, new\_cost, port, sender\_id)

print('new route : {}'.format(new\_route))

dest\_router = 'router{}'.format(r\_id)

self.config.set("output-ports", dest\_router, new\_route)

def update\_route(self, r\_id, new\_cost, port, sender\_id):

if port is None:

#nothing happens to the current table

print("No new routes")

else:

#update the current route

new\_route = '{}-{}-{}-{}'.format(r\_id, new\_cost, port, sender\_id)

print('new route : {}'.format(new\_route))

dest\_router = 'router{}'.format(r\_id)

self.config.set("output-ports", dest\_router, new\_route)

# def triggered\_update(self, router, flush\_timer):

# update\_message = '{}-{}-{}-{}'.format(router, 16, 'N/A', 'N/A')

# update\_message = pickle.dumps(update\_message)

# print("==========Triggered Update===========")

# for port in self.neighbor\_port:

# #sending router's whole output contents in pickle

# sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

# sock.sendto(update\_message, ("127.0.0.1", int(port)))

def update\_table(self):

'''

takes a class variable self.message\_route = e.g "2-3-3002-4"

'''

current\_table = dict(self.config.items("output-ports"))

route\_message\_data = self.route\_message.split("-")

to\_sender\_route\_data = current\_table['router{}'.format(self.sender\_id)].split('-')

sender\_router = 'router{}'.format(self.sender\_id)

route\_message\_router = 'router{}'.format(route\_message\_data[0])

potential\_new\_cost = int(route\_message\_data[1]) + int(to\_sender\_route\_data[1]) + 1 #hop count 1

#write on our config file with key = reachable id and value = altered route line

print('updating route, message from {}'.format(sender\_router))

router\_id = route\_message\_data[0]

new\_cost = potential\_new\_cost

next\_hop = self.sender\_id

#check if the entry exists in the current table, if so,

if route\_message\_router in current\_table.keys():

current\_route = current\_table[route\_message\_router]

current\_route\_data = current\_route.split('-')

if current\_route\_data[2] != "N/A" and current\_route\_data[3] == "N/A":

pass

else:

#it compares the cost, if current entry is better,

if int(current\_route\_data[1]) <= potential\_new\_cost:

#destination port is set to none

dest\_port = None

else:

#if the new cost is better, the destination port is set to the new one

dest\_port = current\_route\_data[2]

self.update\_route(router\_id, new\_cost, dest\_port, next\_hop)

'''

to test this,

start only two routers,

let one router learn a new route,

turn off the other router,

and then see if the router invalidate, and flush the aged route.

'''

#entry router is in flush\_timer, so cancel the timer

self.flush\_timer[route\_message\_router].cancel()

self.invalid\_timer[route\_message\_router].cancel()

self.invalid\_timer[route\_message\_router].start()

else:

#if the new entry came in,

dest\_port = route\_message\_data[2]

#calls add\_new\_route

self.add\_new\_route(router\_id, new\_cost, to\_sender\_route\_data[2], next\_hop)

#creates a new instance of timer for running the invalidate and flush function

#put into a dictionary with the entry router name as the key

self.invalid\_timer[route\_message\_router] = routeTimer(45, self.invalidate\_, route\_message\_router)

self.flush\_timer[route\_message\_router] = routeTimer(60, self.flush\_, route\_message\_router)

#starts the invalid timer for the specific route

self.invalid\_timer[route\_message\_router].start()

print('Invalid timer started for {}'.format(route\_message\_router))

def main():

print('RIP Router Demon')

print('RIP Version: 2')

config\_file = sys.argv[1]

demon = RIP\_demon(config\_file)

def update\_timer():

'''

function called every 30 seconds, implementing periodic update

'''

demon.send\_message()

demon.recieve\_message()

demon.show\_routes()

#timer execute every 30 sec

Timer(12, update\_timer).start()

demon.load\_startup()

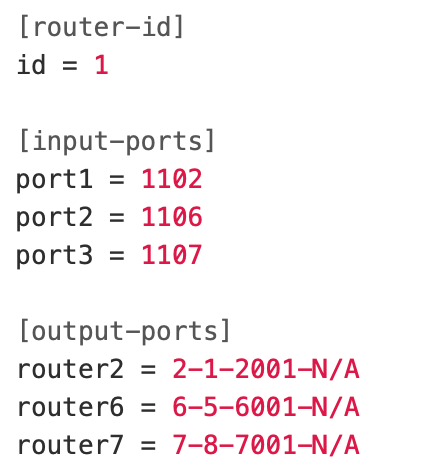
demon.bind\_socket()

demon.show\_routes()

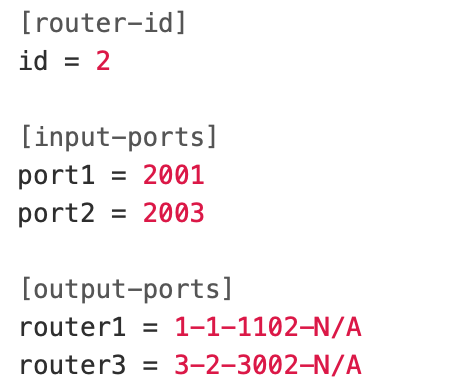
update\_timer()

main()

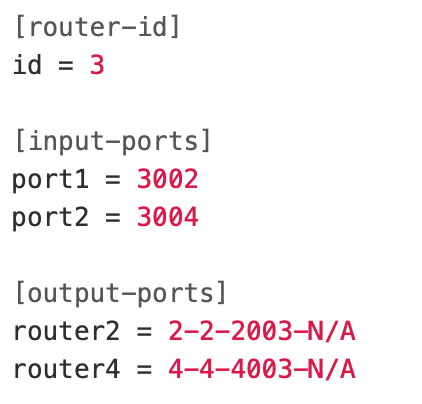
config1.ini



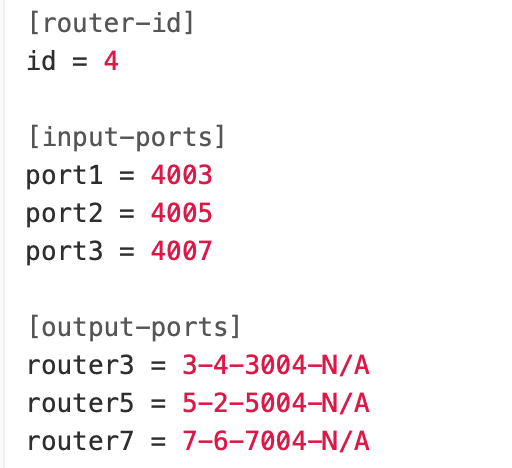
config2.ini



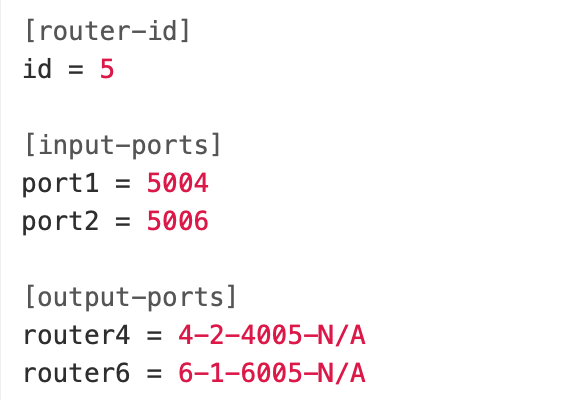
config3.ini



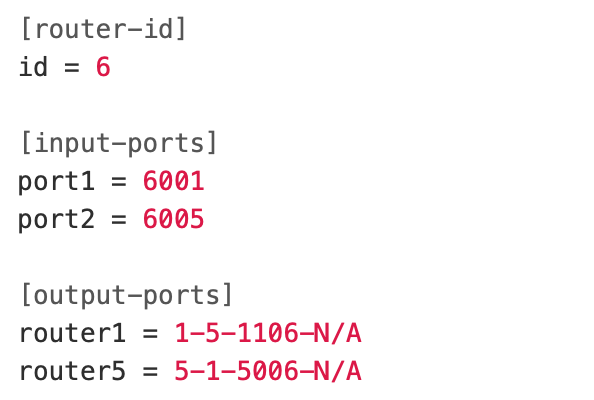
config4.ini



config5.ini



config6.ini



config7.ini

