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?

In our program We made some judgment in the function receive\_message() and use select to complete non-blocking work. so that the code execution is more concise.

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

How have you ensured atomicity of event processing?

We use select for each socket to implement non-blocking and we make use of built in timer tasks to handle time related events.

Our testing plans：

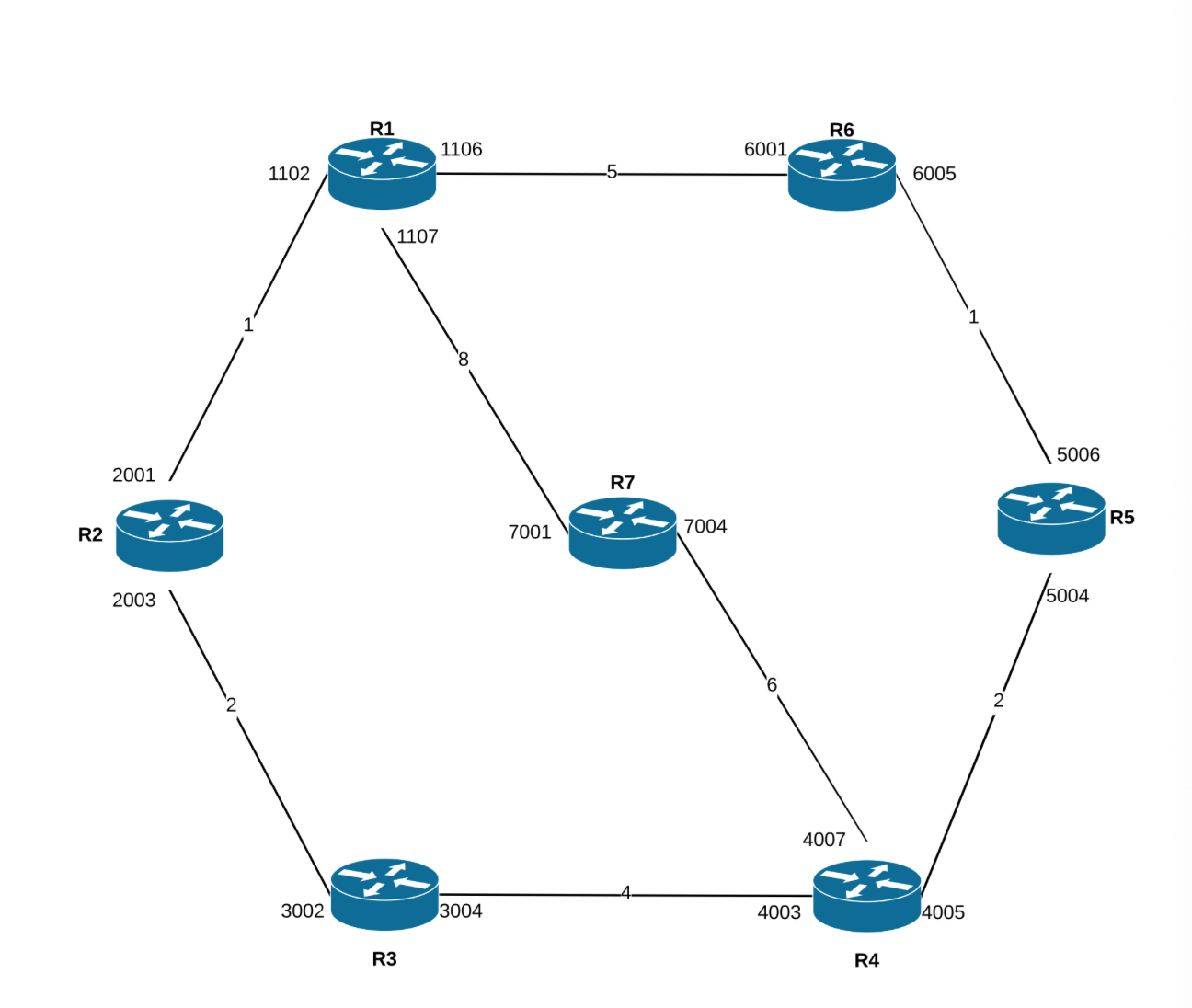
In fact, we did a lot of small tests when we wrote the program. In fact, we did a lot of small tests when we wrote the program. We will add print statements to display the results and compare them with the expected results to determine whether the program is right or not.

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.

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 RIP\_demon(object):

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.message = []

self.config = configparser.ConfigParser()

self.drop = []

self.learned\_routers = []

def load\_startup(self):

self.config.read(self.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"))

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

self.ingress.append(value)

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])

def open\_ports(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 show\_routes(self):

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 {} invalidated'.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):

'''

update self.output\_lines

update with the config files, or

update with the seperate source

'''

sender = {}

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

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

self.routes.update({key:value})

self.message.append(sender)

self.message.append(self.routes)

def send\_message(self):

'''

send to neighbors

'''

now = datetime.now().time()

#create message

update\_message = pickle.dumps(self.message)

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

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)))

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

def recieve\_message(self):

'''

update routing config

update output config

implement Split Horizon (kinda, filters route to itself, could be better if the message itself doesn't contain that bit, yeah)

'''

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

print("listening for messages...")

def update\_table(route, sender):

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

print("===============Update Routes=============")

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

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

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

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

r\_id = route\_data[0]

cost = route\_data[1]

port = route\_data[2]

next\_hop = route\_data[3]

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

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

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

to\_sender\_cost = to\_sender\_route[1]

potential\_new\_cost = int(to\_sender\_cost) + int(cost)

def flush\_(router):

now = datetime.now().time()

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

with open(self.file, 'w') as configfile:

self.config.write(configfile)

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

print(now)

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

return

flush\_timer = Timer(60, flush\_,[router\_num])

sender\_flush\_timer = Timer(60, flush\_,[sender\_id\_num])

def invalidate\_(router):

now = datetime.now().time()

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

print("===========Invalidation============")

print(now)

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

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

with open(self.file, 'w') as configfile:

self.config.write(configfile)

flush\_timer.start()

return

invalid\_timer = Timer(180, invalidate\_,[router\_num])

sender\_invalid\_timer = Timer(180, invalidate\_,[sender\_id\_num])

sender\_invalid\_timer.cancel()

sender\_flush\_timer.cancel()

sender\_invalid\_timer.start()

print('Invalid timer started for {}'.format(sender\_id\_num))

if router\_num in current\_table:

current\_route = current\_table[router\_num]

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

current\_cost = current\_route\_data[1]

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

current\_next\_hop = current\_route\_data[3]

if current\_next\_hop == "N/A":

print("here")

print(route)

pass

else:

if int(current\_cost) <= potential\_new\_cost:

invalid\_timer.cancel()

flush\_timer.cancel()

invalid\_timer.start()

print("No new routes")

return

else:

new\_cost = potential\_new\_cost

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

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

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

invalid\_timer.cancel()

flush\_timer.cancel()

invalid\_timer.start()

return

else:

new\_cost = potential\_new\_cost

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

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

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

invalid\_timer.start()

print('Invalid timer started for {}'.format(router\_num))

return

def triggered\_update():

'''

triggered update

'''

return

message, \_, \_ = select.select(self.ingress\_sockets, [], [], 30)

current\_time = time.time()

for s in message:

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

data, addr = s.recvfrom(1024)

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

sender = message\_data[0]

update = message\_data[1]

for r\_id in update:

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

if router == r\_id:

#drop the route destined to myself

pass

else:

update\_table(update[r\_id], sender)

return

def main():

print('RIP Router Demon')

print('RIP Version: 2')

config\_file = sys.argv[1]

demon = RIP\_demon(config\_file)

def update\_timer():

demon.create\_message()

demon.send\_message()

demon.recieve\_message()

demon.show\_routes()

Timer(30, update\_timer).start()

demon.load\_startup()

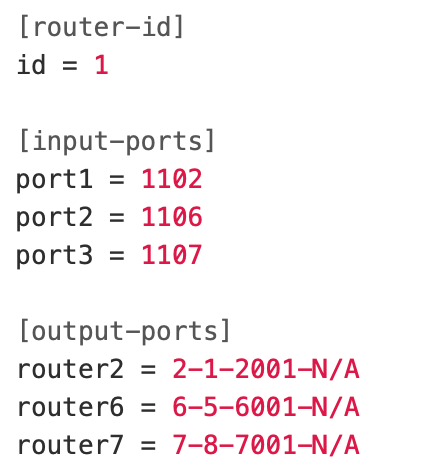
demon.open\_ports()

demon.show\_routes()

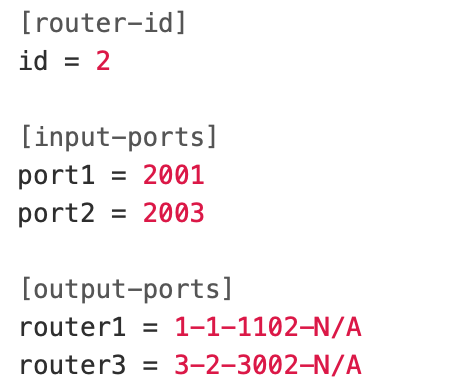
update\_timer()

main()

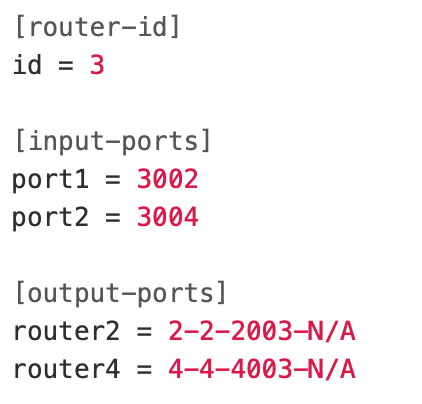
config1.ini



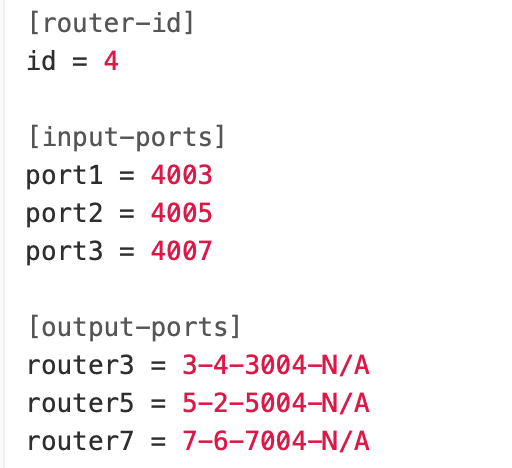
config2.ini



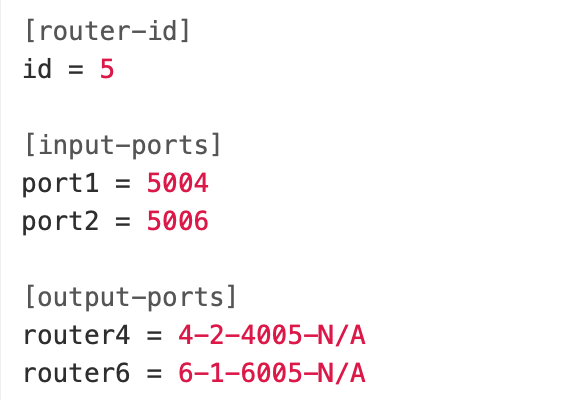
config3.ini



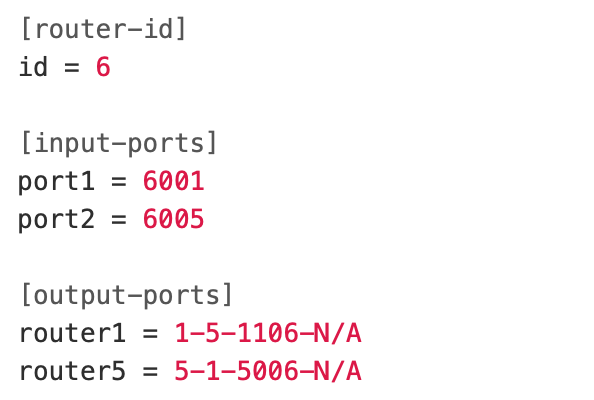
config4.ini



config5.ini



config6.ini



config7.ini

