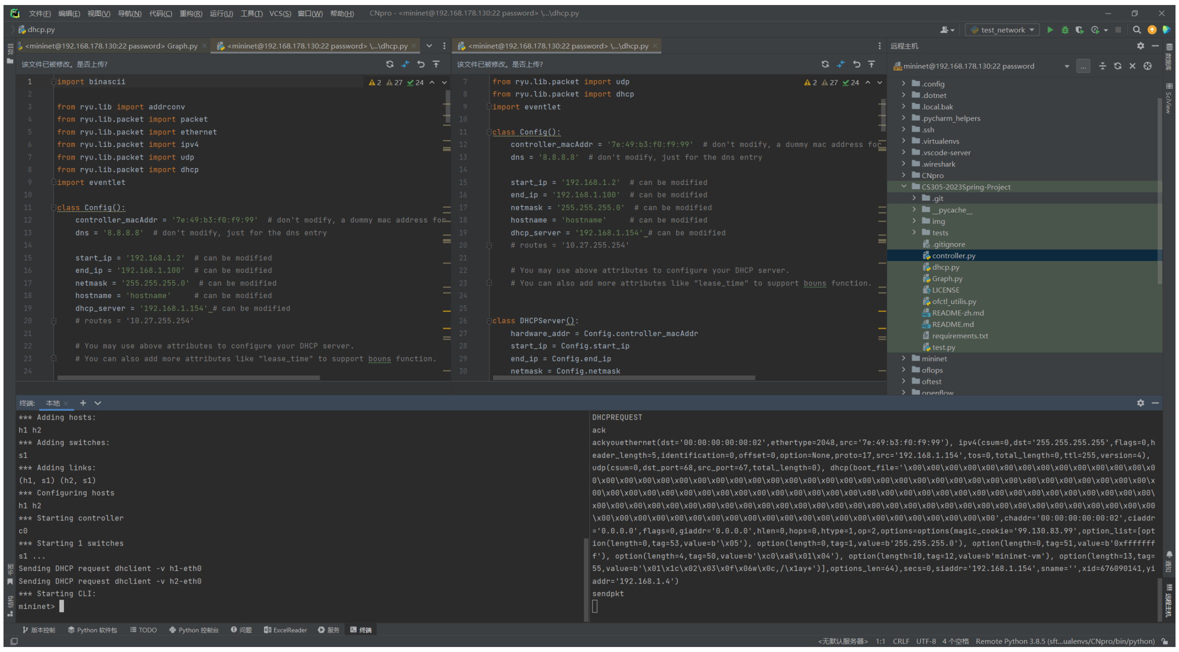
Computer Network Project Report

Group Member & Contribution Rate

- 杨宗奇 12111412 1/3
- 钟志源 12110517 1/3
- 刘晓群 12110943 1/3

Task 0: Environment Setup

- We use the software `VMware` and `virtualbox` to activate the visual machine mininet in our PC.
- We use `develop` tool in pycharm for the main interaction with the mininet.
- After the `ssh connection`, we can directly manipulate the inside terminal also get packet from the inner wireshark.
  ex. shown as following.



```python
import binascii

from ryu.lib import addrconv
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ipv4
from ryu.lib.packet import udp
from ryu.lib.packet import dhcp
import eventlet

class Config():
    controller_macAddr = '7e:49:b3:f0:f9:99'  # don't modify, a dummy mac address for
    dns = '8.8.8.8'  # don't modify, just for the dns entry

    start_ip = '192.168.1.2'  # can be modified
    end_ip = '192.168.1.100'  # can be modified
    netmask = '255.255.255.0'  # can be modified
    hostname = 'hostname'     # can be modified
    dhcp_server = '192.168.1.154' # can be modified
    # routes = '10.27.255.254'

    # You may use above attributes to configure your DHCP server.
    # You can also add more attributes like "lease_time" to support bouns function.
```

```python
from ryu.lib.packet import udp
from ryu.lib.packet import dhcp
import eventlet

class Config():
    controller_macAddr = '7e:49:b3:f0:f9:99'  # don't modify, a dummy mac address for
    dns = '8.8.8.8'  # don't modify, just for the dns entry

    start_ip = '192.168.1.2'  # can be modified
    end_ip = '192.168.1.100'  # can be modified
    netmask = '255.255.255.0'  # can be modified
    hostname = 'hostname'     # can be modified
    dhcp_server = '192.168.1.154' # can be modified
    # routes = '10.27.255.254'

    # You may use above attributes to configure your DHCP server.
    # You can also add more attributes like "lease_time" to support bouns function.


class DHCPServer():
    hardware_addr = Config.controller_macAddr
    start_ip = Config.start_ip
    end_ip = Config.end_ip
    netmask = Config.netmask
```

```
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
Sending DHCP request dhclient -v h1-eth0
Sending DHCP request dhclient -v h2-eth0
*** Starting CLI:
mininet> 
```

```
DHCPREQUEST
ack
ackyouethernet(dst='00:00:00:00:00:02',ethertype=2048,src='7e:49:b3:f0:f9:99'), ipv4(csum=0,dst='255.255.255.255',flags=0,h
eader_length=5,identification=0,offset=0,option=None,proto=17,src='192.168.1.154',tos=0,total_length=0,ttl=255,version=4),
udp(csum=0,dst_port=68,src_port=67,total_length=0), dhcp(boot_file='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00',chaddr='00:00:00:00:00:02',ciaddr
='0.0.0.0',flags=0,giaddr='0.0.0.0',hlen=0,hops=0,htype=1,op=2,options=options(magic_cookie='99.130.83.99',option_list=[opt
ion(length=0,tag=53,value=b'\x05'), option(length=0,tag=1,value=b'255.255.255.0'), option(length=0,tag=51,value=b'0xffffff
f'), option(length=4,tag=50,value=b'\xc0\xa8\x01\x04'), option(length=10,tag=12,value=b'mininet-vm'), option(length=13,tag=
55,value=b'\x01\x1c\x02\x03\x0f\x06w\x0c,/\x1ay*')],options_len=64),secs=0,siaddr='192.168.1.154',sname='',xid=676090141,yi
addr='192.168.1.4')
sendpkt
```

文件(F) 编辑(E) 视图(V) 导航(N) 代码(C) 重构(R) 运行(U) 工具(T) VCS(S) 窗口(W) 帮助(H)
CNpro - <mininet@192.168.178.130:22 password> \...\dhcp.py
<mininet@192.168.178.130:22 password> Graph.py
<mininet@192.168.178.130:22 password> \...\dhcp.py
该文件已被修改。是否上传?
远程主机
mininet
oflops
oftest
终端: 本地
版本控制 Python 软件包 TODO Python 控制台 问题 ExcelReader 服务 终端
<无默认服务器> 1:1 CRLF UTF-8 4 个空格 Remote Python 3.8.5 (sft...ualenvs/CNpro/bin/python)

PS. We will highlight the bonus part in this report.

Task 1: DHCP

- The implementation of this task mainly includes
  - dhcp.py

- controller.py

controller.py

- In the controller, we mainly use the `packet_in_handler` function.
    - Divide different kind of message in the event
    - Provide parameters into the `DHCPServer` class.

```
msg = ev.msg
        datapath = msg.datapath
        pkt = packet.Packet(data=msg.data)
        pkt_dhcp = pkt.get_protocols(dhcp.dhcp)
        pkt_arp = pkt.get_protocols(arp.arp)
        inPort = msg.in_port
        if pkt_dhcp:
            DHCPServer.handle_dhcp(datapath, inPort, pkt)
```

dhcp.py

This class can be called and build either `offer packet` or `ack packet` according to the dhcp state.

```
dhcp_state = cls.get_state(cls, pkt_dhcp)
if dhcp_state == 'DHCPDISCOVER':
     cls._send_packet(datapath, port, cls.assemble_offer(pkt,datapath))
elif dhcp_state == 'DHCPREQUEST':
    cls._send_packet(datapath, port, cls.assemble_ack(pkt,datapath,port))
```

Attributes

```
controller_macAddr = '7e:49:b3:f0:f9:99'  # don't modify, a dummy mac address for fill the mac enrty
    dns = '8.8.8.8'  # don't modify, just for the dns entry

    start_ip = '192.168.1.2'  # can be modified
    end_ip = '192.168.1.100'  # can be modified
    netmask = '255.255.255.0'  # can be modified
    hostname = 'hostname'     # can be modified
    dhcp_server = '192.168.1.154' # can be modified
```

Functions

- `assemble_ack` assemble ack packet in response to `quest`
- `assemble_offer` assemble offer packet in response to `discover`
- `_send_packet` send packets out to the acquiring place.
- `get_state` Separate the head
  To better separate information that head contains about the client state, we write a `get_state` method.

```
def get_state(cls, pkt_dhcp):
    # 根据 Option 53
    dhcp_state = ord(
        [opt for opt in pkt_dhcp.options.option_list if opt.tag == 53][0].value)
    # 获取 DHCP 协议头中的 Option 53（消息类型），并转换为整数类型
    if dhcp_state == 1:
        state = 'DHCPDISCOVER'
    elif dhcp_state == 2:
        state = 'DHCPOFFER'
    elif dhcp_state == 3:
        state = 'DHCPREQUEST'
    elif dhcp_state == 5:
        state = 'DHCPACK'
    return state
```

The ip allocate issue

With the `start_ip` and `end_ip`, we define a range of IP addresses that the DHCP server can allocate. This range is specified in the configuration of the DHCP server.

We use the ip_pool to solve the issue of ip allocate which is also the main function of our dhcp design. In initialization, the server will create a list of IP `ip_pool` and a set `ip_used` in this range and sort them in ascending order. `ip_pool` is used to keep track of available IP addresses, and `ip_used` is used to keep track of IP that have been allocated to clients.

When a new client sends a `discover` message, the DHCP server will check the length of the list. If the list is empty(no available IP), the server will return `0.0.0.0` to the client. Otherwise, the server will allocate the first IP address in `ip_pool` to the client and remove it from the list,(pop in python) and add this ip to `ip_used`.

When a new client sends a `request`, the server will assign the requested ip to the client. In this way, according to the RFC protocol design, we ensure that DHCP does not duplicate IP assignments.

```
# get the ip pool
    ip_pool = []
    arr1 = start_ip.split('.')
    begin = int(''.join(arr1[3]))
    arr2 = end_ip.split('.')
    end = int(''.join(arr2[3]))
    for i in range(begin, end + 1):
        ip = start_ip[:-len(arr1[3])] + str(i)
        if ip not in ip_pool:
            ip_pool.append(ip)
    ip_pool.sort()
    print(ip_pool)
```

The Release Time Issue

We implemented the functionality of DHCP lease duration.

- DHCP Server Response: The DHCP server receives the client's lease duration request and responds with a DHCP offer message that includes the lease duration it is willing to assign. The server sets the "DHCP Option 51" field in the DHCP offer message to indicate the lease duration it is providing.

```
lease_time = 6000
lease_time_option = dhcp.option(tag=51, value=lease_time.to_bytes(4,
byteorder='big'))
```

- Client Lease Acceptance: The client receives the DHCP offer message from the server, including the lease duration. The client then sends a DHCP request message to the server to accept the offered lease duration.
- DHCP Server Lease Confirmation: The DHCP server receives the client's DHCP request message, confirming the lease duration. The server updates its lease database to record the lease start time and the assigned lease duration.
- Lease Renewal: As the lease duration approaches expiration, the client may choose to renew the lease by sending a DHCP request

message to the server, requesting an extension of the lease duration. The server can respond with a DHCP ACK message, granting the lease renewal and updating the lease duration.

- Lease Expiration: Once the lease duration has expired, the DHCP server can release the IP address and reclaim it for potential assignment to other clients. The server marks the lease as expired in its lease database.

Things We've Inserted

To make the packet we send more reliable, we constructed all necessary options.

Options Number	Usage	length
1	Set subnet mask option.	4 bytes
3	Set gateway address option.	must be a multiple of 4 bytes DNS Server
6	Set DNS server address option.	must be a multiple of 4 bytes DNS Server
12	Set domain name option.	
15	Set domain name suffix option.	not fixed
28	Set the broadcast ip	
50	Set requested IP option.	
51	Set IP address lease time option.	4 bytes (in seconds)
53	Set DHCP message type.	1 byte
54	Set server identifier.	
55	Set requested parameter list option. The client uses this option to specify which network configuration parameters it wants to obtain from the server.	
61	Set client identifier option.	

Task2: Shortest Path Switching

- The implementation of this task mainly includes
  - controller.py
  - Graph.py

Graph.py

It is a class described by ourselves, which is an abstract data structure aimed to store the topology of the net work and to run shoretest path algorithm.

Attributes

```
class Graph:
def __init__(self):
self.nodes = []
self.edges = [[-1 for i in range(MAX_V)] for j in range(MAX_V)]
self.port = [[-1 for i in range(MAX_V)] for j in range(MAX_V)]
self.port_on = [[False for i in range(MAX_V)] for j in range(MAX_V)]
self.vis = []
```

- MAX_V: The maximum of the number of nodes
- nodes: The switches in the network
- edges: The link weight between two switches
- port: port[i][j] denotes which port of switch i used to connect to switch j
- port_on: port_on[i][j] denotes whether port j of i is on
- vis: boolean list for dijkstra and SPFA
- paths: shortest paths between any two switches

Functions

- dijskstra(self, src)
  - find shortest paths of the single source src.
  - uses PriorityQueue to optimize the time complexity
  - record the previous node of each node in the shortest path and join them to form a path then store it in the self.paths

- SPFA(self, src)
  - find shortest paths of the single source src.
  - uses Queue to implement SPFA algorithm and store the shortest paths into self.paths
- floyd(self, src)
  - uses floyd algorithm to find shortest paths between any two switches
- shortest_path(self, src, dst)
  - check if their is a shortest path between src and dst, return the path and its length, otherwise return an empty list and -1

controller.py

- In the controller, we mainly finish two things
  - send arp response messages to the hosts who invokes arp requests to let them learn the arp record and store the records in the local arp cache.
  - update the topology of abstract data structure, apply the shortest path algorithm, reset and install the flow table on all the switches, meanwhile print them out in the controller console.

Send ARP Response

- First we create an class $MyMap$ to store the relationship of hosts' ip and mac address, and get information of that from the initial arping of hosts.
- Then upon saving the arp request, we send an arp response according to the message in the arp packet.

```
ofctl = OfCtl_v1_0(datapath, self.logger)
ofctl.send_arp(arp_opcode=ARP_REPLY, vlan_id=VLANID_NONE,
dst_mac=src_mac, sender_mac=dst_mac,
sender_ip=dst_ip, target_mac=src_mac, target_ip=src_ip,
src_port=OFPP_CONTROLLER,
output_port=inPort)
```

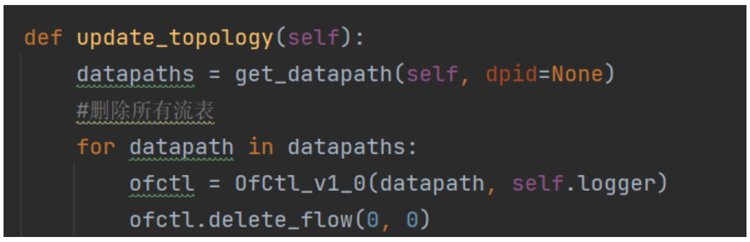
- dst_mac: the destination mac address for which the arp response should send to
- sender_mac, sender_ip, target_mac, target_ip: fields of arp packet, the target should be the host who send arp request, and the sender

should be the destination host that sender request for.

- src_port, out_port: To the switch who send the arp request to controller, the src_port of the arp response is OFPP_CONTROLLER, and it should be send to the host, so the out_port is the port which connected to the host.

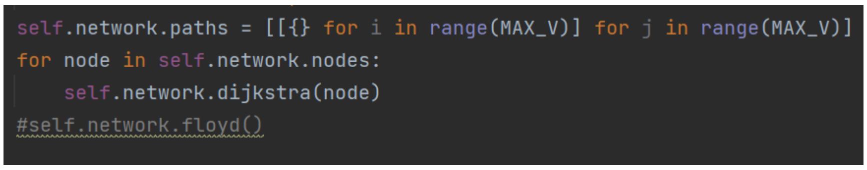
Update_topology

- first we delete all previous flow table



```
def update_topology(self):
    datapaths = get_datapath(self, dpid=None)
    #删除所有流表
    for datapath in datapaths:
        ofctl = OfCtl_v1_0(datapath, self.logger)
        ofctl.delete_flow(0, 0)
```

- Then we run one of three shortest path algorithms
  for single source algorithms dijkstra and SPFA, traverse each node as the src.
  for floyd, only run once



```
self.network.paths = [[{} for i in range(MAX_V)] for j in range(MAX_V)]
for node in self.network.nodes:
    self.network.dijkstra(node)
#self.network.floyd()
```

- get_out_port(self, datapath, src, dst)
  - in order to set flow table, we need to know which port of a certain switch connecting to the next hop in the shortest path of src to dst.

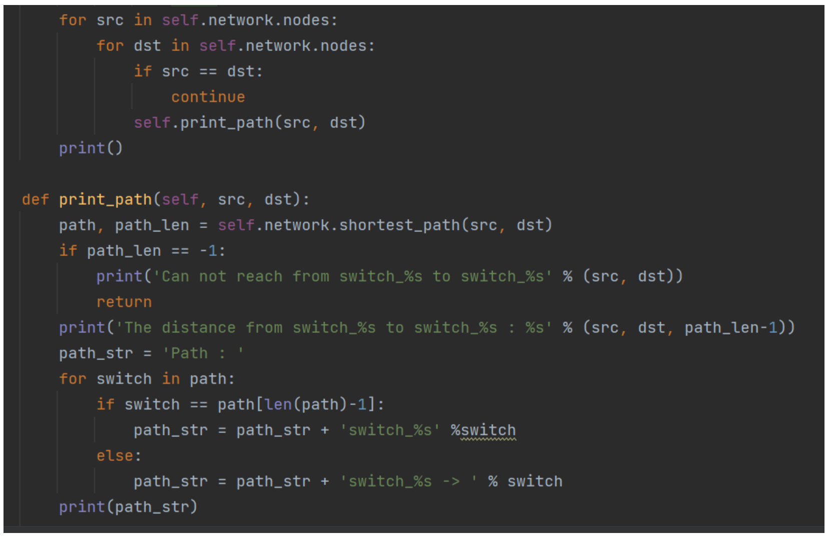
```
def get_out_port(self, datapath, src, dst):
        dpid = datapath.id
        path, path_len = self.network.shortest_path(src, dst)
        next_hop = path[path.index(dpid) + 1]
        out_port = self.network.port[dpid][next_hop]
        return out_port
```

- Then we traverse each host as the dst, each switch as the src, get the shortest path and set the flow table using ofctl.
- Note that the switch that directly connecting to the host don't get the out_port from the shorest path but from the attribute of host.

```
for host in self.hosts:
dst_mac = host.mac
```

```
dst = host.port.dpid
for datapath in datapaths:
src = datapath.id
ofp_parser = datapath.ofproto_parser
if src != dst:
out_port = self.get_out_port(datapath, src, dst)
actions = [ofp_parser.OFPActionOutput(out_port)]
ofctl = OfCtl_v1_0(datapath, self.logger)
ofctl.set_flow(0, 0, dl_dst=dst_mac, actions=actions)
else:
ofctl = OfCtl_v1_0(datapath, self.logger)
actions = [ofp_parser.OFPActionOutput(host.port.port_no)]
ofctl.set_flow(0, 0, dl_dst=dst_mac, actions=actions)
```

- Each time of updating the topology, we print all the shortest paths.



```
    for src in self.network.nodes:
        for dst in self.network.nodes:
            if src == dst:
                continue
            self.print_path(src, dst)
    print()

def print_path(self, src, dst):
    path, path_len = self.network.shortest_path(src, dst)
    if path_len == -1:
        print('Can not reach from switch_%s to switch_%s' % (src, dst))
        return
    print('The distance from switch_%s to switch_%s : %s' % (src, dst, path_len-1))
    path_str = 'Path : '
    for switch in path:
        if switch == path[len(path)-1]:
            path_str = path_str + 'switch_%s' %switch
        else:
            path_str = path_str + 'switch_%s -> ' % switch
    print(path_str)
```

After doing the above, we can achieve task2

Problems and Conclusion

We have met with a lot of problems since we start assembling the python environment. It is not easy to go over everything related to find a good way to manipulate the inner files in another operating system.

Packet Construction

About how we construct the `offer packet` and `ack packet`, the encode problem is what the most time consuming. Everything seems correct, but it doesn't output the right message. Then we reread the `rfc document` and fixed this bug.

Packet Assemble

It could be troublesome to assemble the packet manually. To check whether we assemble the packet correctly, we installed `wireshark` in the VM and use it to capture the packet along the test process. We can examine the packet in `wireshark` and see if the message is correct.

No new ip issue

When all ips in the ip_pool is occupied but there is new discover request coming up, the dhcp server will ignore this discover packet.

Problems in Task2

In task2, we encounter the following problems

- Initially, we don't understand the process of this task and what things should be done in the code.
  Solution: After exploration and consultation we figure it out.
- Unfamiliar with the library functions and classes, have no ideas how to translate the logical thoughts into code.
  Solutions: With continuous tries, we gradually have command in these functions and classes and succeed in finish this task.
- Unable to detect the details inside the mininet, thus hard to debugging.
  Solutions: We tries to print out the necessary message in the controller, and print the topology and flow tables of the network using mininet CLI to debug.

reference

[1] Ryu's API documentation
https://ryu.readthedocs.io/en/latest/index.html
[2] Mininet's document
https://github.com/mininet/mininet/wiki/Documentation
[3] Mininet source code https://github.com/mininet/mininet
[4] Openflow quick start
https://homepages.dcc.ufmg.br/~mmvieira/cc/OpenFlow%20Tutorial%20-%20OpenFlow%20Wiki.htm