CS305 Project

SDN Simulation

Fan Site
12111624@mail.sustech.edu.cn
Xiao Jiachen
12112012@mail.sustech.edu.cn

Abstract

In this project, an Ryu Software-defined networking controller is implemented, which provides DHCP service and maintains shortest path scheduling flow tables. This report introduces the features, implement details and some testcases for the project.

Contents

1. Contributors

SID	Name	Contributions	Contribution Rate
12111624	Fan Site	Shortest-Path Switching	50%
12112012	Xiao Jiachen	DHCP	50%

2. Project Structure

```
SDN_SIMULATION
|  controller.py
|  device.py
|  dhcp.py
|  ofctl_utilis.py
|  topo_manager.py
|  README.md
|
├─doc
|      project_demo_instructions.md
|
└─tests
    ├─dhcp_test
    |      test_3hosts.py
    |      test_network.py
    |
    └─switching_test
            test_robust.png
            test_robust.py
            test_tree.png
            test_tree.py
            test_triangle.py
```

3. DHCP

The Dynamic Host Configuration Protocol (DHCP) process typically involves four main steps: DISCOVER, OFFER, REQUEST, ACK/NAK. The SDN controller replies an OFFER packet on receiving a DISCOVER packet, and replies an ACK/NAK packet for a REQUEST packet.

Offer: DHCP controller on the network receives the Discover message and respond with a DHCP Offer message. The Offer message contains an available IP address along with other configuration information such as subnet mask.

```
def assemble_offer(cls, pkt, datapath):
    eth = pkt.get_protocol(ethernet.ethernet)
    ip = pkt.get_protocol(ipv4.ipv4)
    udp_pkt = pkt.get_protocol(udp.udp)
    dhcp_pkt = pkt.get_protocol(dhcp.dhcp)
    new_ip = cls.get_available_ip()
    if new_ip == None:
```

```
        offer_pkt = cls.nack_pkt(dhcp_pkt, eth)
    else:
        offer_pkt = cls.ack_pkt(dhcp_pkt, eth, new_ip, '02')
    pkt = cls.convert_to_ethernet(offer_pkt, udp_pkt, ip, eth)
    return pkt
```

ACK/NAK: DHCP controller receive the Request message and check if the offered IP address is still available. If so, the controller responds with a DHCP ACK message, indicating that the client's request has been accepted. Otherwise, it sends a NAK message, indicating that the address requested has been occupied and the host needs to run the DHCP process again. ACK/NAK packets are generated by `ryu.lib.packet.dhcp.dhcp()`.

4. Shortest-Path Switching

4.1. Graph abstraction

The network consists of 3 types of components: hosts, switches and links. We first wrap up hosts and switches as `MyDevice` class. Then we construct a graph with devices as vertices, links as undirected edges(with different ports for different directions), and direct connection between hosts and adajcent switches also as edges. The abstract graph is stored in `topo_manager` class.

```
class MyDevice(object):
    def __init__(self, device):
        self.device = device
    def is_host(self):
        return isinstance(self.device, Host)
    def is_switch(self):
        return isinstance(self.device, Switch)

class topo_manager:
    def __init__(self):
          self.graph = {}
          self.vertex = []
          self.hosts = []
          self.switches = []
```

4.2. Event Handling

In topo_manager, there are some event handler functions:

```
def switch_enter(self, switch)
def switch_leave(self, switch)
def host_add(self, host, switch, port)
def link_add(self, dev1, dev2, port1, port2)
def link_delete(self, dev1, dev2)
def port_modify(self, port, state)
```

The ways to handle these events are:

- switch entering: simply wrap up the switch as device node, add this node to the graph.

- switch leave: remove the vertex in the graph with the same datapath id as the given switch, update topology flow table.
- host add: put the information of the new host into ARP table, update topology glow table.
- link add: add undirected edges(weight default set as 1) to the graph, update topology flow table.
- link delete: delete undirected edges according to given datapath id, update topology flow table.

4.3. Dijkstra

For the shortest-path switching function, we implemented Dijkstra algorithm to schedule routes between every pair of hosts. The pseudocode is shown as follows:

```
function Dijkstra(graph, source):
    Initialize distances: dist[node] = infinity for all nodes in graph
    Set distance from source to itself: dist[source] = 0
    Initialize an empty priority queue: pq

    Enqueue source with distance 0 into pq

    while pq is not empty:
        Dequeue a node, current, from pq

        if current has been visited:
            Continue to the next iteration

        Mark current as visited

        for each neighbor in graph[current]:
            Calculate new distance: new_dist = dist[current] + weight(current,
neighbor)

            if new_dist < dist[neighbor]:
                Update dist[neighbor] to new_dist

                next hop of neighbor <- current

                Enqueue neighbor with distance new_dist into pq

    Return dist
```

5. Testcases

5.1. DHCP

The following result is obtained from `dhcp_test/test_network.py`. All functions of DHCP services are correctly implemented, allowing modifying the ip address pool, subnet mask, lease time etc.

```
mininet> h1 ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        ether 00:00:00:00:00:01  txqueuelen 1000  (Ethernet)
        RX packets 20  bytes 1200 (1.2 KB)
        RX errors 0  dropped 17  overruns 0  frame 0
        TX packets 1  bytes 58 (58.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 1: h1 ip info

```
mininet> h2 ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.2  netmask 255.0.0.0  broadcast 10.255.255.255
        ether 00:00:00:00:00:02  txqueuelen 1000  (Ethernet)
        RX packets 21  bytes 1260 (1.2 KB)
        RX errors 0  dropped 17  overruns 0  frame 0
        TX packets 1  bytes 58 (58.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 2: h2 ip info

5.2. Shortest-Path Switching

The complex testcase is a complete binary tree rooted at s1, with an extra link between s4 and s6.

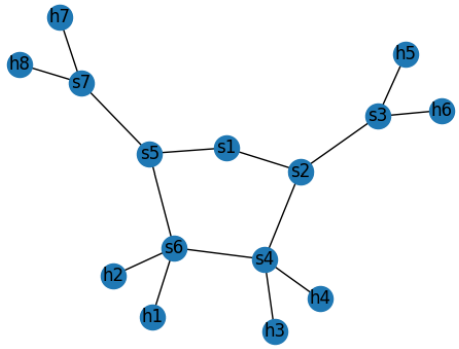

Figure 3: complex testcase

The flow table is listed in Section 8.1.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
```

Figure 4: `pingall` result

The routing and flow tables after all types of operations passed all tests of the lab presentation.

6. Bonus Tasks

6.1. DHCP

Besides the basic function, we add lease duration in DHCP's options, this means after a specific time, the IP address will be recycled. After that, the client should send an REQUEST to further use this IP.

The DHCP Server itself will create an IP pool to know which IP was used so it will not allocate duplicate IP. And get_avaliable_ip can get a IP that no one use(or exceed lease time), declare_use_ip will confirm that a host will use a IP.

The testcase for bonus task is designed as: 3 hosts sending dhcp discover messages with 10s timeout, but the lease time is 256s, and the ip pool size is only 2, so the 3rd host cannot get ip address. For the running log, please see Section 8.2.

(h1, h2, h3 are hosts, ip pool: 10.0.0.2 10.0.0.3, command timeout = 10s, lease time = 256s)

7. Summary

In this project, we implemented an Ryu Software-defined networking controller with basic DHCP services and maintaining shortest path scheduling flow tables by Dijkstra. We have a further understanding of how SDN works from this experience. Great thanks to Prof. Li Zhuozhao, TA Wang Qing, SAs and everyone else who provides support to this project and computer network learning!

8. Appendix

8.1. Flow Table For Testcase

```
*** s1 ------------------------------------------------------------------------
    cookie=0x0,    duration=14.780s,    table=0,...,ip,dl_dst=00:00:00:00:00:07
actions=output:"s1-eth2"
    cookie=0x0,    duration=14.780s,    table=0,...,ip,dl_dst=00:00:00:00:00:08
actions=output:"s1-eth2"
    cookie=0x0,    duration=14.709s,    table=0,...,ip,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
    cookie=0x0,    duration=14.708s,    table=0,...,ip,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth1"
*** s2 ------------------------------------------------------------------------
    cookie=0x0,    duration=14.783s,    table=0,...,ip,dl_dst=00:00:00:00:00:05
actions=output:"s2-eth3"
    cookie=0x0,    duration=14.783s,    table=0,...,ip,dl_dst=00:00:00:00:00:06
actions=output:"s2-eth3"
    cookie=0x0,    duration=14.783s,    table=0,...,ip,dl_dst=00:00:00:00:00:03
actions=output:"s2-eth3"
    cookie=0x0,    duration=14.783s,    table=0,...,ip,dl_dst=00:00:00:00:00:04
actions=output:"s2-eth3"
    cookie=0x0,    duration=14.783s,    table=0,...,ip,dl_dst=00:00:00:00:00:07
actions=output:"s2-eth1"
    cookie=0x0,    duration=14.782s,    table=0,...,ip,dl_dst=00:00:00:00:00:08
actions=output:"s2-eth1"
    cookie=0x0,    duration=14.736s,    table=0,...,ip,dl_dst=00:00:00:00:00:01
actions=output:"s2-eth2"
    cookie=0x0,    duration=14.736s,    table=0,...,ip,dl_dst=00:00:00:00:00:02
```

```
actions=output:"s2-eth2"
*** s3 ------------------------------------------------------------------------
    cookie=0x0,    duration=14.808s,    table=0,...,ip,dl_dst=00:00:00:00:00:01
actions=output:"s3-eth2"
    cookie=0x0,    duration=14.808s,    table=0,...,ip,dl_dst=00:00:00:00:00:05
actions=output:"s3-eth1"
    cookie=0x0,    duration=14.808s,    table=0,...,ip,dl_dst=00:00:00:00:00:06
actions=output:"s3-eth1"
    cookie=0x0,    duration=14.808s,    table=0,...,ip,dl_dst=00:00:00:00:00:03
actions=output:"s3-eth1"
    cookie=0x0,    duration=14.807s,    table=0,...,ip,dl_dst=00:00:00:00:00:04
actions=output:"s3-eth1"
    cookie=0x0,    duration=14.803s,    table=0,...,ip,dl_dst=00:00:00:00:00:07
actions=output:"s3-eth1"
    cookie=0x0,    duration=14.803s,    table=0,...,ip,dl_dst=00:00:00:00:00:08
actions=output:"s3-eth1"
    cookie=0x0,    duration=14.802s,    table=0,...,ip,dl_dst=00:00:00:00:00:02
actions=output:"s3-eth3"
*** s4 ------------------------------------------------------------------------
    cookie=0x0,    duration=14.778s,    table=0,...,ip,dl_dst=00:00:00:00:00:01
actions=output:"s4-eth1"
    cookie=0x0,    duration=14.778s,    table=0,...,ip,dl_dst=00:00:00:00:00:02
actions=output:"s4-eth1"
    cookie=0x0,    duration=14.778s,    table=0,...,ip,dl_dst=00:00:00:00:00:05
actions=output:"s4-eth4"
    cookie=0x0,    duration=14.777s,    table=0,...,ip,dl_dst=00:00:00:00:00:06
actions=output:"s4-eth4"
    cookie=0x0,    duration=14.777s,    table=0,...,ip,dl_dst=00:00:00:00:00:03
actions=output:"s4-eth2"
    cookie=0x0,    duration=14.777s,    table=0,...,ip,dl_dst=00:00:00:00:00:07
actions=output:"s4-eth4"
    cookie=0x0,    duration=14.777s,    table=0,...,ip,dl_dst=00:00:00:00:00:08
actions=output:"s4-eth4"
    cookie=0x0,    duration=14.777s,    table=0,...,ip,dl_dst=00:00:00:00:00:04
actions=output:"s4-eth3"
*** s5 ------------------------------------------------------------------------
    cookie=0x0,    duration=14.800s,    table=0,...,ip,dl_dst=00:00:00:00:00:07
actions=output:"s5-eth3"
    cookie=0x0,    duration=14.800s,    table=0,...,ip,dl_dst=00:00:00:00:00:08
actions=output:"s5-eth3"
    cookie=0x0,    duration=14.799s,    table=0,...,ip,dl_dst=00:00:00:00:00:01
actions=output:"s5-eth1"
    cookie=0x0,    duration=14.799s,    table=0,...,ip,dl_dst=00:00:00:00:00:02
actions=output:"s5-eth1"
    cookie=0x0,    duration=14.799s,    table=0,...,ip,dl_dst=00:00:00:00:00:05
actions=output:"s5-eth2"
    cookie=0x0,    duration=14.799s,    table=0,...,ip,dl_dst=00:00:00:00:00:06
actions=output:"s5-eth2"
    cookie=0x0,    duration=14.799s,    table=0,...,ip,dl_dst=00:00:00:00:00:03
actions=output:"s5-eth2"
    cookie=0x0,    duration=14.798s,    table=0,...,ip,dl_dst=00:00:00:00:00:04
```

```
actions=output:"s5-eth2"
*** s6 ------------------------------------------------------------------------
    cookie=0x0,    duration=14.848s,    table=0,...,ip,dl_dst=00:00:00:00:00:01
actions=output:"s6-eth4"
    cookie=0x0,    duration=14.847s,    table=0,...,ip,dl_dst=00:00:00:00:00:02
actions=output:"s6-eth4"
    cookie=0x0,    duration=14.843s,    table=0,...,ip,dl_dst=00:00:00:00:00:05
actions=output:"s6-eth2"
    cookie=0x0,    duration=14.842s,    table=0,...,ip,dl_dst=00:00:00:00:00:06
actions=output:"s6-eth3"
    cookie=0x0,    duration=14.824s,    table=0,...,ip,dl_dst=00:00:00:00:00:07
actions=output:"s6-eth1"
    cookie=0x0,    duration=14.824s,    table=0,...,ip,dl_dst=00:00:00:00:00:08
actions=output:"s6-eth1"
    cookie=0x0,    duration=14.824s,    table=0,...,ip,dl_dst=00:00:00:00:00:03
actions=output:"s6-eth4"
    cookie=0x0,    duration=14.824s,    table=0,...,ip,dl_dst=00:00:00:00:00:04
actions=output:"s6-eth4"
*** s7 ------------------------------------------------------------------------
    cookie=0x0,    duration=14.833s,    table=0,...,ip,dl_dst=00:00:00:00:00:01
actions=output:"s7-eth1"
    cookie=0x0,    duration=14.833s,    table=0,...,ip,dl_dst=00:00:00:00:00:02
actions=output:"s7-eth1"
    cookie=0x0,    duration=14.833s,    table=0,...,ip,dl_dst=00:00:00:00:00:05
actions=output:"s7-eth1"
    cookie=0x0,    duration=14.833s,    table=0,...,ip,dl_dst=00:00:00:00:00:06
actions=output:"s7-eth1"
    cookie=0x0,    duration=14.833s,    table=0,...,ip,dl_dst=00:00:00:00:00:03
actions=output:"s7-eth1"
    cookie=0x0,    duration=14.833s,    table=0,...,ip,dl_dst=00:00:00:00:00:04
actions=output:"s7-eth1"
    cookie=0x0,    duration=14.833s,    table=0,...,ip,dl_dst=00:00:00:00:00:07
actions=output:"s7-eth2"
    cookie=0x0,    duration=14.833s,    table=0,...,ip,dl_dst=00:00:00:00:00:08
actions=output:"s7-eth3"
```

8.2. DHCP log

```
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
```

```
*** Starting 1 switches
s1 ...
Sending DHCP request dhclient -v h1-eth0
Sending DHCP request dhclient -v h2-eth0
Sending DHCP request dhclient -v h3-eth0
*** Starting CLI:
mininet> h1 ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.2  netmask 255.255.255.0  broadcast 10.0.0.255
        ether 00:00:00:00:00:01  txqueuelen 1000  (Ethernet)
        RX packets 3  bytes 704 (704.0 B)
        RX errors 0  dropped 1  overruns 0  frame 0
        TX packets 2  bytes 684 (684.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet> h2 ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.3  netmask 255.255.255.0  broadcast 10.0.0.255
        ether 00:00:00:00:00:02  txqueuelen 1000  (Ethernet)
        RX packets 4  bytes 1026 (1.0 KB)
        RX errors 0  dropped 1  overruns 0  frame 0
        TX packets 3  bytes 1026 (1.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet> h3 ifconfig
h3-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 00:00:00:00:00:03  txqueuelen 1000  (Ethernet)
        RX packets 4  bytes 936 (936.0 B)
        RX errors 0  dropped 1  overruns 0  frame 0
        TX packets 3  bytes 1026 (1.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
```

```
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```