软件定义网络
实验二

Exp1:

首先,使用 topo 图:

```python
#!/usr/bin/env python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                         controller=RemoteController,
                         ip='127.0.0.1',
                         protocol='tcp',
                         port=6633)

    info( '*** Add switches\n')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch, dpid='0000000000000001')
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch, dpid='0000000000000002')
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch, dpid='0000000000000003')
    s4 = net.addSwitch('s4', cls=OVSKernelSwitch, dpid='0000000000000004')
    s5 = net.addSwitch('s5', cls=OVSKernelSwitch, dpid='0000000000000005')

    info( '*** Add hosts\n')
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)

    info( '*** Add links\n')
    net.addLink(h1, s1)
    net.addLink(s1, s4)
    net.addLink(s4, s5)
    net.addLink(s1, s2)
    net.addLink(s2, s3)
    net.addLink(s3, s5)
    net.addLink(s5, h2)

    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s1').start([c0])
    net.get('s2').start([c0])
    net.get('s3').start([c0])
    net.get('s4').start([c0])
    net.get('s5').start([c0])

    info( '*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

使用 dynamic_routes.py, 关键函数如下:

控制路径变化(每 3s 更换一次)

```
def install_path(self, src_dpid, dst_dpid, src_port, dst_port, ev, src, dst, pkt_ipv4, pkt_tcp):
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    mid_path = None
    if self.pathmod == 0 or self.pathmod == 1:
        self.pathmod = self.pathmod+1
        mid_path = self.short_path(src=src_dpid, dst=dst_dpid)
    elif self.pathmod == 2:
        self.pathmod = self.pathmod+1
        mid_path = self.long_path(src=src_dpid, dst=dst_dpid)
    elif self.pathmod == 3:
        self.pathmod = self.pathmod+1
        mid_path = [(5,2),(3,2),(3,1),(2,2),(2,1),(1,3)]
        if self.pathmod == 4:
            self.pathmod = 0

    if mid_path is None:
        return
    self.path = None
    self.path = [(src_dpid, src_port)] + mid_path + [(dst_dpid, dst_port)]

    self.logger.info("path : %s", str(self.path))

    for i in range(len(self.path) - 2, -1, -2):
        datapath_path = self.datapaths[self.path[i][0]]
        match = parser.OFPMatch(in_port=self.path[i][1], eth_src=src, eth_dst=dst, eth_type=0x0800,
                                ipv4_src=pkt_ipv4.src, ipv4_dst=pkt_ipv4.dst)

        if i < (len(self.path) - 2):
            actions = [parser.OFPActionOutput(self.path[i + 1][1])]
        else:
            actions = [parser.OFPActionSetField(eth_dst=self.ip_to_mac.get(pkt_ipv4.dst)),
                       parser.OFPActionOutput(self.path[i + 1][1])]

        self.add_flow(datapath_path, 100, match, actions, idle_timeout=0, hard_timeout=3)
    # time_install = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')
    # self.logger.info("time_install: %s", time_install)
```

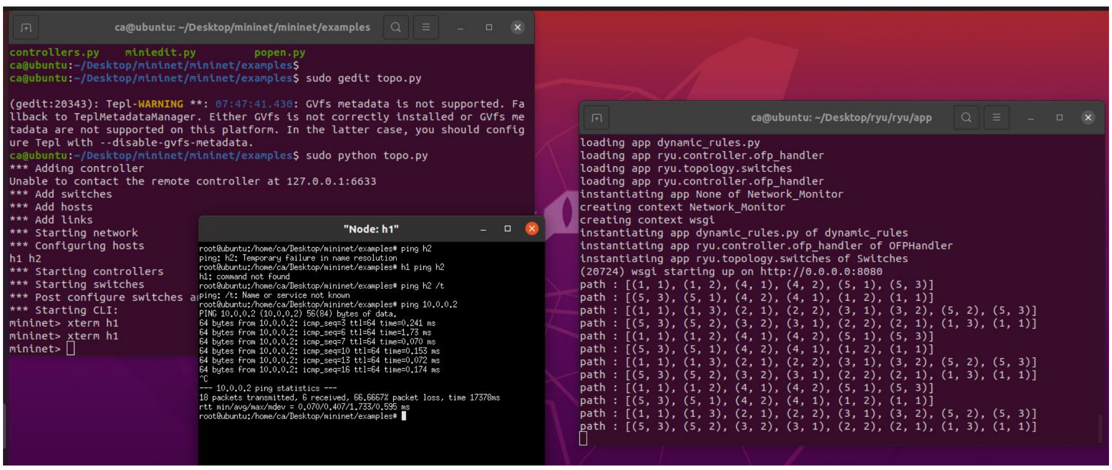
最短路径:

```
def short_path(self, src, dst, bw=0):
    if src == dst:
        return []
    result = defaultdict(lambda: defaultdict(lambda: None))
    distance = defaultdict(lambda: None)

    # the node is checked
    seen = [src]

    # the distance to src
    distance[src] = 0

    w = 1  # weight
    while len(seen) < len(self.src_links):
        node = seen[-1]
        if node == dst:
            break
        for (temp_src, temp_dst) in self.src_links[node]:
            if temp_dst not in seen:
                temp_src_port = self.src_links[node][(temp_src, temp_dst)][0]
                temp_dst_port = self.src_links[node][(temp_src, temp_dst)][1]
                if (distance[temp_dst] is None) or (distance[temp_dst] > distance[temp_src] + w):
                    distance[temp_dst] = distance[temp_src] + w
                    # result = {"dpid":(link_src, src_port, link_dst, dst_port)}
                    result[temp_dst] = (temp_src, temp_src_port, temp_dst, temp_dst_port)
        min_node = None
        min_path = 999
        # get the min_path node
        for temp_node in distance:
            if (temp_node not in seen) and (distance[temp_node] is not None):
                if distance[temp_node] < min_path:
                    min_node = temp_node
                    min_path = distance[temp_node]
        if min_node is None:
            break
        seen.append(min_node)

    path = []

    if dst not in result:
        return None

    while (dst in result) and (result[dst] is not None):
        path = [result[dst][2:4]] + path
        path = [result[dst][0:2]] + path
        dst = result[dst][0]
    #self.logger.info("path : %s", str(path))
    return path
```

最长路径(将权值取反即可)

```python
def long_path(self, src, dst, bw=0):
    if src == dst:
        return []
    result = defaultdict(lambda: defaultdict(lambda: None))
    distance = defaultdict(lambda: None)

    # the node is checked
    seen = [src]

    # the distance to src
    distance[src] = 0

    w = 1  # weight
    while len(seen) < len(self.src_links):
        node = seen[-1]
        if node == dst:
            break
        for (temp_src, temp_dst) in self.src_links[node]:
            if temp_dst not in seen:
                temp_src_port = self.src_links[node][(temp_src, temp_dst)][0]
                temp_dst_port = self.src_links[node][(temp_src, temp_dst)][1]
                if (distance[temp_dst] is None) or (distance[temp_dst] < distance[temp_src] + w):
                    distance[temp_dst] = distance[temp_src] + w
                    # result = {"dpid":(link_src, src_port, link_dst, dst_port)}
                    result[temp_dst] = (temp_src, temp_src_port, temp_dst, temp_dst_port)
        min_node = None
        min_path = 999
        # get the min_path node
        for temp_node in distance:
            if (temp_node not in seen) and (distance[temp_node] is not None):
                if distance[temp_node] < min_path:
                    min_node = temp_node
                    min_path = distance[temp_node]
        if min_node is None:
            break
        seen.append(min_node)

    path = []

    if dst not in result:
        return None

    while (dst in result) and (result[dst] is not None):
        path = [result[dst][2:4]] + path
        path = [result[dst][0:2]] + path
        dst = result[dst][0]
    #self.logger.info("path : %s", str(path))
    return path
@set_ev_cls(ofp_event.EventOFPStateChange, [MAIN_DISPATCHER, DEAD_DISPATCHER])
```

得到结果如图所示:



Exp1 成功!

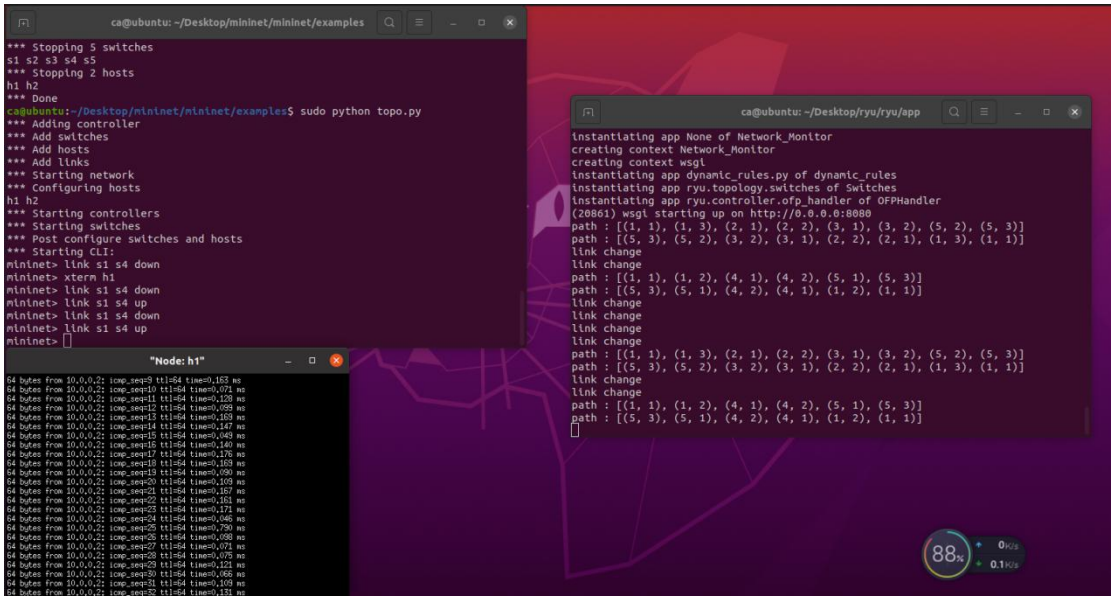
Exp2:

增加了手动链路转换机制

```python
# these two functions need to be coded in your own way
    def delete_flow(self, datapath, priority, match, actions, dst_port, idle_timeout=0, hard_timeout=0):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        mod = parser.OFPFlowMod(datapath=datapath,
                                # table_id=ofproto.OFPTT_ALL,
                                command=ofproto.OFPFC_DELETE,
                                priority=priority, match=match,
                                idle_timeout=idle_timeout,
                                hard_timeout=hard_timeout,
                                # out_port=ofproto.OFPP_ANY,
                                out_port = dst_port,
                                out_group=ofproto.OFPG_ANY,
                                instructions=inst)
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPortStatus, [CONFIG_DISPATCHER, MAIN_DISPATCHER, DEAD_DISPATCHER, HANDSHAKE_DISPATCHER])
    def get_OFPPortStatus_msg(self, ev):
        msg = ev.msg
        datapath = ev.msg.datapath
        dpid = msg.datapath.id
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        port = msg.desc.port_no
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        # key1 = (dpid, port)
        # dpidPort_to_ip=self.network_monitor.DpidPort_to_ip[key1]
        reason = msg.reason
        if reason == ofproto.OFPPR_MODIFY:
            if self.path:
                print("link has changed")
                for i in range(1, len(self.path) - 2, 1):
                    self.delete_flow(datapath=datapath, match=match, actions=actions, priority=100, dst_port=self.path[i + 1][1])
```

函数 1 用于删除流表,函数二则是在 down 或 up 后进行相应的删除操作

实验结果:



Exp2 成功!!