计算机网络网络层实验3报告

马成 20307130112

整体部分

1. init()函数，建立了一下每一个路由器节点的父节点，然后由于观察到从一个路由器去一个ip主机的路线仅有一条，因此提前纪录了这一信息

```
def __init__(self, *args, **kwargs):
    super(ProjectController, self).__init__(*args, **kwargs)
    self.datapath_list = {}
    self.switches = []
    self.adjacency = defaultdict(dict)
    self.hosts = {'10.0.0.1': (1, 1), '10.0.0.2': (1, 2), '10.0.0.3': (2,
1), '10.0.0.4': (2, 2),
                  '10.0.0.5': (3, 1), '10.0.0.6': (3, 2), '10.0.0.7': (4,
1), '10.0.0.8': (4, 2),
                  '10.0.0.9': (5, 1), '10.0.0.10': (5, 2), '10.0.0.11':
(6, 1), '10.0.0.12': (6, 2),
                  '10.0.0.13': (7, 1), '10.0.0.14': (7, 2), '10.0.0.15':
(8, 1), '10.0.0.16': (8, 2)}
    self.father={1:(9,10),2:(9,10),3:(11,12),4:(11,12),5:(13,14),6:
(13,14),7:(15,16),8:(15,16),9:(17,18),10:(19,20),11:(17,18),12:(19,20),13:
(17,18),14:(19,20),15:(17,18),16:(19,20)}
    self.ip_son={
        9:{'10.0.0.1':1,'10.0.0.2':1,'10.0.0.3':2,'10.0.0.4':2},
        10:{'10.0.0.1':1,'10.0.0.2':1,'10.0.0.3':2,'10.0.0.4':2},
        11:{'10.0.0.5':3,'10.0.0.6':3,'10.0.0.7':4,'10.0.0.8':4},
        12:{'10.0.0.5':3,'10.0.0.6':3,'10.0.0.7':4,'10.0.0.8':4},
        13:{'10.0.0.9':5,'10.0.0.10':5,'10.0.0.11':6,'10.0.0.12':6},
        14:{'10.0.0.9':5,'10.0.0.10':5,'10.0.0.11':6,'10.0.0.12':6},
        15:{'10.0.0.13':7,'10.0.0.14':7,'10.0.0.15':8,'10.0.0.16':8},
        16:{'10.0.0.13':7,'10.0.0.14':7,'10.0.0.15':8,'10.0.0.16':8},
        17:
{'10.0.0.1':9,'10.0.0.2':9,'10.0.0.3':9,'10.0.0.4':9,'10.0.0.5':11,'10.0.0.6
':11,'10.0.0.7':11,'10.0.0.8':11,'10.0.0.9':13,'10.0.0.10':13,'10.0.0.11':13
,'10.0.0.12':13,'10.0.0.13':15,'10.0.0.14':15,'10.0.0.15':15,'10.0.0.16':15}
,
        18:
{'10.0.0.1':9,'10.0.0.2':9,'10.0.0.3':9,'10.0.0.4':9,'10.0.0.5':11,'10.0.0.6
':11,'10.0.0.7':11,'10.0.0.8':11,'10.0.0.9':13,'10.0.0.10':13,'10.0.0.11':13
,'10.0.0.12':13,'10.0.0.13':15,'10.0.0.14':15,'10.0.0.15':15,'10.0.0.16':15}
,
        19:
{'10.0.0.1':10,'10.0.0.2':10,'10.0.0.3':10,'10.0.0.4':10,'10.0.0.5':12,'10.0
.0.6':12,'10.0.0.7':12,'10.0.0.8':12,'10.0.0.9':14,'10.0.0.10':14,'10.0.0.11
':14,'10.0.0.12':14,'10.0.0.13':16,'10.0.0.14':16,'10.0.0.15':16,'10.0.0.16'
:16},
```

```
        20:{'10.0.0.1':10,'10.0.0.2': 10,'10.0.0.3': 10,'10.0.0.4':
10,'10.0.0.5': 12,'10.0.0.6': 12,'10.0.0.7': 12,'10.0.0.8': 12,'10.0.0.9':
14,'10.0.0.10': 14,'10.0.0.11': 14,'10.0.0.12': 14,'10.0.0.13':
16,'10.0.0.14': 16,'10.0.0.15': 16,'10.0.0.16': 16},
    }
    self.costs={}
    self.cnt=0
    self.path=defaultdict(list)
    self.key=[('10.0.0.12','10.0.0.16'),('10.0.0.12','10.0.0.1')] #学号:
20307130112
```

2. 修改了link_delete_handler函数，否则会提前改变链接情况导致错误

```
@set_ev_cls(event.EventLinkDelete, MAIN_DISPATCHER)
def link_delete_handler(self, ev):
    pass
```

3. _packet_in_handler函数

   - 大部分其实是用的助教老师PPT上的内容
   - 就是获取发送和接收方地址，利用get_nxt函数获得通到下一个路由器所需要走的端口即可

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    msg = ev.msg                    #switch送来的事件ev,ev.msg 是表示packet_in数
据结构的一个对象
    datapath = msg.datapath         #msg.datapath是switch Datapath的一个对象，是
哪个switch发来的消息
    ofproto = datapath.ofproto      #协商的版本
    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocol(ethernet.ethernet) # 获取二层包头信息
    in_port = msg.match['in_port']
    if eth.ethertype == ether_types.ETH_TYPE_LLDP: # ignore lldp packet
        return
    src=None
    dst=None
    dpid=datapath.id
    match=None
    parser = datapath.ofproto_parser
    if eth.ethertype==ether_types.ETH_TYPE_IP:
        _ipv4=pkt.get_protocol(ipv4.ipv4)
        src=_ipv4.src
        dst=_ipv4.dst
        match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,
in_port=in_port, ipv4_src=src, ipv4_dst=dst)
    elif eth.ethertype==ether_types.ETH_TYPE_ARP:
        arp_pkt=pkt.get_protocol(arp.arp)
        src=arp_pkt.src_ip
        dst=arp_pkt.dst_ip
        match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_ARP,
in_port=in_port, arp_spa=src, arp_tpa=dst)
    else:
        return
    out_port=self.get_nxt(dpid,src,dst)
```

```
    actions = [datapath.ofproto_parser.OFPActionOutput(out_port)]
    # install a flow to avoid packet_in next time
    if out_port != ofproto.OFPP_FLOOD:
        self.add_flow(datapath,1,match,actions)
    data = None
    if msg.buffer_id == ofproto.OFP_NO_BUFFER:  # 还得把包送往该去的端口
        data = msg.data
    out = datapath.ofproto_parser.OFPPacketOut(
        datapath=datapath, buffer_id=msg.buffer_id, in_port=in_port,
        actions=actions, data=data)
    datapath.send_msg(out)
```

4. get_nxt函数

   - 如果当前请求的路径没有被计算就进入cal_path进行计算（这一部分会在之后说，因为需要根据不同的选择方式来写）
   - 之后遍历path，找到当前dpid所在的位置，如果不是最后一个位置，那么输出这个位置到计算过的下一条的端口号。否则直接输出到达目的主机的端口号

```
def get_nxt(self,dpid,src,dst):
    if (src,dst) not in self.path:
        self.cal_path(src,dst)
    now_path=self.path[(src,dst)]
    for i in range(len(now_path)):
        if now_path[i]==dpid:
            if i==len(now_path)-1:
                return self.hosts[dst][1]
            else:
                return self.adjacency[dpid][now_path[i+1]]
```

5. 一些工具函数

```
def ip2num(self,s:str):
    s=s.split('.')[-1]
    return int(s)

def print_path(self, key):
    print("h%d ->" % (self.ip2num(key[0])), end=" ")
    for i in self.path[key]:
        print("s%d ->" % (i), end=" ")
    print("h%d" % (self.ip2num(key[1])))
```

LPR

1. 代码部分

   - 循环查找，第一条一定是只连接src主机的路由器直接获取
   - 如果当前路由器是紧邻主机的路由器1 <= dpid and dpid <= 8，那么判断一下目标路由器是不是在这个路由器之下，如果是，就说明已经到达目的路由器，这个就是最后条了，直接退出。否则下一条就是当前路由器的左父节点
   - 如果不是紧邻主机的路由器，那么如果目的节点在他的ip_son中，就说明可以向下传了，直接转移到某个特定的儿子节点，否则下一条就是当前路由器的左父节点

```
def cal_path(self, src, dst):
    dpid=self.hosts[src][0]
    while True:
        self.path[(src,dst)].append(dpid)
        if(1 <= dpid and dpid <= 8):
            if (self.hosts[dst][0] == dpid):
                break
            else:
                dpid=self.father[dpid][0]
        else:
            if (dst in self.ip_son[dpid]):
                dpid=self.ip_son[dpid][dst]
            else:
                dpid=self.father[dpid][0]
    if (src,dst) in self.key:
        self.print_path((src,dst))
```

2. 结果

```
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
EventSwitchEnter<dpid=8, 4 ports>
EventSwitchEnter<dpid=10, 4 ports>
EventSwitchEnter<dpid=1, 4 ports>
EventSwitchEnter<dpid=14, 4 ports>
EventSwitchEnter<dpid=15, 4 ports>
EventSwitchEnter<dpid=13, 4 ports>
EventSwitchEnter<dpid=3, 4 ports>
EventSwitchEnter<dpid=19, 4 ports>
EventSwitchEnter<dpid=6, 4 ports>
EventSwitchEnter<dpid=7, 4 ports>
EventSwitchEnter<dpid=18, 4 ports>
EventSwitchEnter<dpid=5, 4 ports>
EventSwitchEnter<dpid=9, 4 ports>
EventSwitchEnter<dpid=17, 4 ports>
EventSwitchEnter<dpid=12, 4 ports>
EventSwitchEnter<dpid=4, 4 ports>
```

```
EventSwitchEnter<dpid=2, 4 ports>
EventSwitchEnter<dpid=11, 4 ports>
EventSwitchEnter<dpid=20, 4 ports>
EventSwitchEnter<dpid=16, 4 ports>
h12 -> s6 -> s13 -> s17 -> s15 -> s8 -> h16
h12 -> s6 -> s13 -> s17 -> s9 -> s1 -> h1
EventSwitchLeave<dpid=19, 0 ports>
EventSwitchLeave<dpid=6, 0 ports>
EventSwitchLeave<dpid=1, 0 ports>
EventSwitchLeave<dpid=10, 0 ports>
EventSwitchLeave<dpid=14, 0 ports>
EventSwitchLeave<dpid=3, 0 ports>
EventSwitchLeave<dpid=15, 4 ports>
EventSwitchLeave<dpid=8, 4 ports>
EventSwitchLeave<dpid=13, 4 ports>
EventSwitchLeave<dpid=7, 0 ports>
EventSwitchLeave<dpid=4, 0 ports>
EventSwitchLeave<dpid=12, 0 ports>
EventSwitchLeave<dpid=20, 0 ports>
EventSwitchLeave<dpid=5, 0 ports>
EventSwitchLeave<dpid=18, 0 ports>
EventSwitchLeave<dpid=17, 4 ports>
EventSwitchLeave<dpid=9, 4 ports>
EventSwitchLeave<dpid=2, 4 ports>
EventSwitchLeave<dpid=11, 4 ports>
EventSwitchLeave<dpid=16, 0 ports>
```

RSP

1. 代码部分
   - 和LPR基本类似，只是换成了随机选择一个父节点向上传递

```
def cal_path(self, src, dst):
    dpid=self.hosts[src][0]
    while True:
        self.path[(src,dst)].append(dpid)
        if(1 <= dpid and dpid <= 8):
            if (self.hosts[dst][0] == dpid):
                break
            else:
                dpid=self.father[dpid][random.randint(0,1)]
        else:
            if (dst in self.ip_son[dpid]):
                dpid=self.ip_son[dpid][dst]
            else:
                dpid=self.father[dpid][random.randint(0,1)]
    if (src,dst) in self.key:
        self.print_path((src,dst))
```

2. 结果

```
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
```

```
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
EventSwitchEnter<dpid=7, 4 ports>
EventSwitchEnter<dpid=16, 4 ports>
EventSwitchEnter<dpid=2, 4 ports>
EventSwitchEnter<dpid=15, 4 ports>
EventSwitchEnter<dpid=19, 4 ports>
EventSwitchEnter<dpid=14, 4 ports>
EventSwitchEnter<dpid=8, 4 ports>
EventSwitchEnter<dpid=1, 4 ports>
EventSwitchEnter<dpid=13, 4 ports>
EventSwitchEnter<dpid=4, 4 ports>
EventSwitchEnter<dpid=11, 4 ports>
EventSwitchEnter<dpid=6, 4 ports>
EventSwitchEnter<dpid=18, 4 ports>
EventSwitchEnter<dpid=10, 4 ports>
EventSwitchEnter<dpid=20, 4 ports>
EventSwitchEnter<dpid=3, 4 ports>
EventSwitchEnter<dpid=17, 4 ports>
EventSwitchEnter<dpid=12, 4 ports>
EventSwitchEnter<dpid=5, 4 ports>
EventSwitchEnter<dpid=9, 4 ports>
h12 -> s6 -> s13 -> s18 -> s15 -> s8 -> h16
h12 -> s6 -> s14 -> s20 -> s10 -> s1 -> h1
EventSwitchLeave<dpid=19, 0 ports>
EventSwitchLeave<dpid=6, 0 ports>
EventSwitchLeave<dpid=1, 0 ports>
EventSwitchLeave<dpid=14, 0 ports>
EventSwitchLeave<dpid=15, 0 ports>
EventSwitchLeave<dpid=8, 4 ports>
EventSwitchLeave<dpid=13, 4 ports>
EventSwitchLeave<dpid=3, 4 ports>
EventSwitchLeave<dpid=10, 4 ports>
EventSwitchLeave<dpid=7, 0 ports>
EventSwitchLeave<dpid=4, 4 ports>
EventSwitchLeave<dpid=5, 4 ports>
EventSwitchLeave<dpid=20, 4 ports>
EventSwitchLeave<dpid=12, 4 ports>
EventSwitchLeave<dpid=2, 0 ports>
EventSwitchLeave<dpid=17, 4 ports>
```

```
EventSwitchLeave<dpid=18, 4 ports>
EventSwitchLeave<dpid=16, 4 ports>
EventSwitchLeave<dpid=11, 4 ports>
EventSwitchLeave<dpid=9, 0 ports>
```

LLR

1. 代码部分
   - cal_cost用于计算一个路由器到一个主机ip的路径中最大流量，保证dpid路由器是这个主机的祖先节点
   - cal_cost2用于计算src和dst主机节点经过dpid路由器的路径中最大流量，保证dpid是两个主机的LCA
   - cal_path部分
     - 第一个if就是在判断两个主机是不是的LCA是不是在第一层路由器，如果是就只需要一条
     - 第二个elif判断两个主机的LCA是不是在第二层，如果是他们又两个选择，直接分别计算两个选择的代价选择即可
     - else表示两个主机的LCA在第三层，那么他们又17~20这个4个选择，一次考虑后选出
     - 对于已经选好的路径，给路径中每一条链路的流量加1

```
def cal_cost(self,dpid:int,ip:str):
    if self.hosts[ip][0]==dpid:
        return 0
    ans=0
    while self.hosts[ip][0]!=dpid:
        son=self.ip_son[dpid][ip]
        ans=max(ans,self.costs[(dpid,son)])
        dpid=son
    return ans

def cal_cost2(self,dpid:int,ip1:str,ip2:str):
    return max(self.cal_cost(dpid,ip1),self.cal_cost(dpid,ip2))

def cal_path(self, src, dst):
    dpid1=self.hosts[src][0]
    dpid2=self.hosts[dst][0]
    if dpid1==dpid2:
        self.path[(src,dst)].append(dpid1)
    elif self.father[dpid1]==self.father[dpid2]:
        fa=self.father[dpid1]
        self.path[(src,dst)].append(dpid1)
        if self.cal_cost2(fa[0],src,dst)<self.cal_cost2(fa[1],src,dst):
            self.path[(src,dst)].append(fa[0])
        else:
            self.path[(src,dst)].append(fa[1])
        self.path[(src,dst)].append(dpid2)
    else:
        mincost=100000000
        mindpid=-1
        for i in range(17,21):
            # print(i,self.cal_cost2(i,src,dst))
            if self.cal_cost2(i,src,dst)<mincost:
                mincost=self.cal_cost2(i,src,dst)
```

```
                mindpid=i
        self.path[(src,dst)]=[dpid1,self.ip_son[mindpid]
[src],mindpid,self.ip_son[mindpid][dst],dpid2]
    now_path=self.path[(src,dst)]
    for i in range(len(now_path)-1):
        self.costs[(now_path[i],now_path[i+1])]+=1
        self.costs[(now_path[i+1],now_path[i])]+=1
    if self.cnt<10:
        self.print_path((src,dst))
        self.cnt+=1
```

2. 结果

```
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
EventSwitchEnter<dpid=6, 4 ports>
EventSwitchEnter<dpid=15, 4 ports>
EventSwitchEnter<dpid=7, 4 ports>
EventSwitchEnter<dpid=13, 4 ports>
EventSwitchEnter<dpid=8, 4 ports>
EventSwitchEnter<dpid=4, 4 ports>
EventSwitchEnter<dpid=1, 4 ports>
EventSwitchEnter<dpid=19, 4 ports>
EventSwitchEnter<dpid=14, 4 ports>
EventSwitchEnter<dpid=10, 4 ports>
EventSwitchEnter<dpid=3, 4 ports>
EventSwitchEnter<dpid=12, 4 ports>
EventSwitchEnter<dpid=11, 4 ports>
EventSwitchEnter<dpid=17, 4 ports>
EventSwitchEnter<dpid=5, 4 ports>
EventSwitchEnter<dpid=20, 4 ports>
EventSwitchEnter<dpid=9, 4 ports>
EventSwitchEnter<dpid=18, 4 ports>
EventSwitchEnter<dpid=2, 4 ports>
EventSwitchEnter<dpid=16, 4 ports>
h1 -> s1 -> s9 -> s17 -> s11 -> s3 -> h5
h5 -> s3 -> s12 -> s19 -> s10 -> s1 -> h1
```

```
h1 -> s1 -> s9 -> s17 -> s11 -> s3 -> h6
h6 -> s3 -> s12 -> s19 -> s10 -> s1 -> h1
h2 -> s1 -> s9 -> s17 -> s11 -> s3 -> h6
h6 -> s3 -> s12 -> s19 -> s10 -> s1 -> h2
h2 -> s1 -> s9 -> s17 -> s11 -> s4 -> h7
h7 -> s4 -> s12 -> s19 -> s10 -> s1 -> h2
h3 -> s2 -> s9 -> s18 -> s11 -> s4 -> h7
h7 -> s4 -> s12 -> s20 -> s10 -> s2 -> h3
EventSwitchLeave<dpid=19, 0 ports>
EventSwitchLeave<dpid=6, 0 ports>
EventSwitchLeave<dpid=1, 0 ports>
EventSwitchLeave<dpid=14, 0 ports>
EventSwitchLeave<dpid=10, 0 ports>
EventSwitchLeave<dpid=3, 0 ports>
EventSwitchLeave<dpid=13, 0 ports>
EventSwitchLeave<dpid=8, 0 ports>
EventSwitchLeave<dpid=15, 0 ports>
EventSwitchLeave<dpid=7, 0 ports>
EventSwitchLeave<dpid=4, 0 ports>
EventSwitchLeave<dpid=20, 0 ports>
EventSwitchLeave<dpid=5, 0 ports>
EventSwitchLeave<dpid=12, 0 ports>
EventSwitchLeave<dpid=17, 0 ports>
EventSwitchLeave<dpid=18, 0 ports>
EventSwitchLeave<dpid=2, 0 ports>
EventSwitchLeave<dpid=16, 0 ports>
EventSwitchLeave<dpid=11, 0 ports>
EventSwitchLeave<dpid=9, 0 ports>
```