

H2 南京大学本科生实验报告

课程名称：计算机网络 任课教师：李文中

学院	计算机科学与技术系	专业（方向）	计算机科学与技术系
学号	181860109	姓名	吴润泽
Email	181860109@smail.nju.edu.cn	开始/完成日期	2020/4/8-2020/4/11

H3 1. 实验名称：Forwarding Packets

H3 2. 实验目的

1. 实现路由器的转发和路由功能
2. 理解路由器转发表的具体作用并利用
3. 掌握网络层基础的通信流程

H3 3. 实验过程

H4 Task 2 实现路由表项查询

H5 a. 实现原理

1. 建立路由表：从给的 *forwarding_table* 文件和路由器端口所在的子网建立即可，对于路由器直接相连的子网，下一跳标记为 '#'；
2. 路由表匹配：遍历所有的路由表项，如果匹配则与当前匹配的最大前缀比较，如果孩子网前缀长度大，则更新匹配到的路由表项；如果匹配均失败，返回空。

H5 b. 代码编写

```
def built_router_table(self, filename):
    #通过文件进行读取
    myfile = open(filename, 'r')
    for line in myfile.readlines():
        line = line.strip().split()#去掉首尾额外空格，并将其分割
        if len(line) == 4:
            self.router_table.append((line[0], line[1], line[2],
line[3]))
    myfile.close()
    #通过遍历路由器接口读取
    for intf in self.net.interfaces():
        self.router_table.append(
            (str(intf.ipaddr), str(intf.netmask), '#',
str(intf.name)))

def match_subnet(self, dst_ip):
    maxlen, tar_route=0, None
    for item in self.router_table:
        subnet = IPv4Network(item[0] + '/' + item[1], False)
        if dst_ip in subnet:
            if maxlen < subnet.prefixlen:#更新当前匹配到的最长前缀
项
                tar_route, maxlen = item, subnet.prefixlen
    return tar_route
```

H4 Task3 实现转发数据包和发送Arp请求

H5 模块1: IP数据报的接收和转发

H6 a. 实现原理

1. 每当到达一个IP数据包, 则将其目的地址与路由表匹配, 如果匹配失败则丢弃;
2. 判断目的地址是否直接在某一子网中, 如果是则下一跳地址改为目的地址;
3. 判断下一跳地址是否在ARP表中, 如果在直接进行转发, *goto 7*;
4. 否则, 判断是否存在下一跳不在任意子网的情况 (防止转发表存在错误);
5. 如果查询下一跳的ARP请求已经发过, 则不再发送;
6. 将该包以及应发送ARP的信息, 加入数据包缓存队列, 等待ARP回复;
7. 将以太网头的源地址改为将转发数据包的端口, 目的地址改为下一跳地址, 同时IP头ttl减1。

H6 b. 代码编写

```
def process_IP_Packet(self, packet):# 进行IP数据包的处理
    tar_route = self.match_subnet(dst_ip)
    if tar_route is None:
        pass
    else:
        ...#以太网包头src和下一跳nexthop的确定
        if nexthop in self.arp_table:#若已知下一跳mac地址
            self.IP_forward(packet,port,dst_mac)
        else:#加入缓冲队列
            if tar_route[2]!='#'
            and self.match_subnet(nexthop) is None:
                pass#error! 没有匹配到下一跳的子网
            if nexthop in self.mycache.cache_packet:
                has_same_arp=True
            request_pkt=
            AddPacket(src_mac,src_ip,nexthop,packet,port)
            #若未发过ARP请求
            send_packet(request_pkt[1], request_pkt[0])

def IP_forward(self, packet, port, dst_mac):#将数据包转发
    packet[Ethernet].dst, packet[IPv4].ttl =
    dst_mac, packet[IPv4].ttl - 1
    self.net.send_packet(port, packet)
```

H5 模块2: 数据包缓存队列

H6 a. 实现原理

1. 数据结构: 以字典的方式存储, 关键字为应查询的ip地址, 值为一个列表。列表第一项记录该待发送队列的停留时间, 查询次数和发送ARP请求需要的信息, 其余项为依次加入的数据包;
2. 每加入一项, 则发回应发送的ARP请求包以及端口, 由调用逻辑判断是否需要发送;

H6 b. 代码编写

```

class PktCache:
    def __init__(self):
        self.cache_packet = dict()
    def AddPacket(self, src_mac, src_ip, dst_ip, packet, port):
        if dst_ip not in self.cache_packet:
            self.cache_packet[dst_ip] = list()
            self.cache_packet[dst_ip].append(
                [time.time(), 1, src_ip, src_mac, port])
        self.cache_packet[dst_ip].append(packet)
        return (create_ip_arp_request(src_mac, src_ip, dst_ip), port)

```

H5 模块3：缓存队列ARP请求的处理

H6 a. 实现原理

1. 将缓存队列按照第一次ARP发送请求的时间来排序，以保证发送顺序正确性；
2. 如果发送次数多于四次或者存活时间大于4秒，则将该队列中对应列表中所有包丢弃；
3. 否则，将查询队列每项ip地址的ARP请求包放入待查询队列，并返回；
4. 在路由接收包的循环中，若1s内没有收到包则调用缓存队列的ARP请求函数；

H6 b. 代码编写

```

class PktCache:
    def GapArpQuery(self):
        now = time.time()
        cache_packet_list = list(self.cache_packet.items())
        cache_packet_list.sort(key=lambda x: x[1][0][0])
        arp_packets = list()
        for item in cache_packet_list:
            if item[1][0][1] <= 4
            and now - item[1][0][0] - item[1][0][1] >= 0.0:
                ...# port, src_mac, src_ip, nexthop相应的赋值
                arp_packets.append(
                    (create_ip_arp_request(src_mac, src_ip, nexthop)
                     , port))
                self.cache_packet[item[0]][0][1] += 1
            elif item[1][0][1] >= 5 or now - item[1][0][0] >= 5.0:
                self.cache_packet.pop(item[0])
        return arp_packets

class Router(object):
    while True:
        try:
            timestamp, dev, pkt = self.net.recv_packet(timeout=1.0)
        except NoPackets:
            self.arp_repeat()#调用处

```

H5 模块4：ARP表更新的处理

H6 a. 实现原理

1. 每当ARP请求或者ARP回复到达路由器时，都应更新ARP表；
2. 判断该ARP包的发送方是否在待查询队列中，若不在则返回；
3. 否则得到缓冲队列中对应的列表，并获取应转发数据包的端口，依次发送；

4. 最终将该ip地址对应的缓存列表清空。

H6 b. 代码编写

```
class PktCache:
    def GetArpReply(self, get_ip):
        if get_ip in self.cache_packet:
            return self.cache_packet[get_ip]
        return list()

class Router(object):
    def add_arp_table(self, src_ip, src_mac):
        self.arp_table[src_ip] = (src_mac, time.time())#更新ARP表
        if src_ip in self.mycache.cache_packet:
            cache_pkts = self.mycache.GetArpReply(src_ip)
            port = cache_pkts[0][4] #得到对应的端口，并发送
            for i in range(1, len(cache_pkts)):
                self.IP_forward(cache_pkts[i], port, src_mac)
            self.mycache.cache_packet.pop(src_ip)

    def process_arp_reply(self, port, packet):
        ...# 得到ARP回复包中的相应数据src_mac, src_ip, dst_mac, dst_ip
        # 进行缓冲队列是否命中的检查
        self.add_arp_table(src_ip, src_mac)
        return

    def process_arp_request(self, port, packet):
        ...# 得到ARP请求包中的相应数据src_mac, src_ip, dst_mac, dst_ip
        # 进行缓冲队列是否命中的检查
        self.add_arp_table(src_ip, src_mac)
        ...# 检测目的ip是否为路由器端口，并发送相应的ARP回复
```

H5 c. 实现测试

H6 I. test scenario测试

给定测试文件测试结果

运行 `swyard -t routertests2.srpy myrouter.py`, 结果如下:

```
Results for test scenario IP forwarding and ARP requester tests: 31 passed, 0 failed, 0 pending

Passed:
1 IP packet to be forwarded to 172.16.42.2 should arrive on
  router-eth0
2 Router should send ARP request for 172.16.42.2 out router-
  eth2 interface
3 Router should receive ARP response for 172.16.42.2 on
  router-eth2 interface
4 IP packet should be forwarded to 172.16.42.2 out router-eth2
5 IP packet to be forwarded to 192.168.1.100 should arrive on
  router-eth2
6 Router should send ARP request for 192.168.1.100 out router-
  eth0
7 Router should receive ARP response for 192.168.1.100 on
  router-eth0
8 IP packet should be forwarded to 192.168.1.100 out router-
  eth0
9 Another IP packet for 172.16.42.2 should arrive on router-
  eth0
10 IP packet should be forwarded to 172.16.42.2 out router-eth2
   (no ARP request should be necessary since the information
   from a recent ARP request should be cached)
11 IP packet to be forwarded to 192.168.1.100 should arrive on
  router-eth2
12 IP packet should be forwarded to 192.168.1.100 out router-
  eth0 (again, no ARP request should be necessary since the
  information from a recent ARP request should be cached)
13 An IP packet from 10.100.1.55 to 172.16.64.35 should arrive
  on router-eth1
14 Router should send an ARP request for 10.10.1.254 on router-
  eth1
15 Application should try to receive a packet, but then timeout
16 Router should send another an ARP request for 10.10.1.254 on
  router-eth1 because of a slow response
17 Router should receive an ARP response for 10.10.1.254 on
  router-eth1
18 IP packet destined to 172.16.64.35 should be forwarded on
  router-eth1
19 An IP packet from 192.168.1.239 for 10.200.1.1 should arrive
  on router-eth0. No forwarding table entry should match.
20 An IP packet from 192.168.1.239 for 10.10.50.250 should
  arrive on router-eth0.
21 Router should send an ARP request for 10.10.50.250 on
  router-eth1
22 Router should try to receive a packet (ARP response), but
  then timeout
23 Router should send an ARP request for 10.10.50.250 on
  router-eth1
24 Router should try to receive a packet (ARP response), but
  then timeout
25 Router should send an ARP request for 10.10.50.250 on
  router-eth1
26 Router should try to receive a packet (ARP response), but
  then timeout
27 Router should send an ARP request for 10.10.50.250 on
  router-eth1
28 Router should try to receive a packet (ARP response), but
  then timeout
29 Router should send an ARP request for 10.10.50.250 on
  router-eth1
30 Router should try to receive a packet (ARP response), but
  then timeout
31 Router should try to receive a packet (ARP response), but
  then timeout

All tests passed!
```

测试文档说明

路由器端口沿用所给定的测试模板，同时测试代码行数较多且较为相似，故仅说明其主要功能。

测试代码检测以下方面的正确性：

1. 已知转发端口情况下，对包头的修改处理；
2. 其Arp请求和回复响应功能，以及重复请求的处理和丢弃；
3. 路由表匹配时直接命中某一路由子网，或者下一跳在某子网这两种情况的处理；
4. 路由器中缓存队列各项Arp请求发送的处理。

- case 1：发送的目的 ip 为路由器端口，直接丢弃：

```
req_ping = mk_ping(" ", ' ', '1.1.1.1', '172.16.42.1')
s.expect(PacketInputEvent('router-eth0', req_ping,
display=IPv4))
s.expect(PacketInputTimeoutEvent(1.5))
```

- case 2：一个不在路由器子网中的 ip 向子网特定ip发送 ping 请求：

1. 该ip 不在转发表文件中，而在路由器端口所在子网，故没有下一跳；
2. 首先在对应子网端口发送Arp请求，询问该ip的mac地址；
3. 设定该ip的mac地址，并发送Arp回复到该端口；
4. 路由器修改数据包以太网头和 ttl ，并从该端口转发；
5. 主机发送对应的ping回复，由于目的ip不在子网，故直接丢弃。

```

req_ping = mk_ping('', '', '2.2.2.2', '10.10.1.254')
s.expect(PacketInputEvent('router-eth0', req_ping,
display=IPv4))
otroarp = mk_arpreq("10:00:00:00:00:02", "10.10.0.1",
"10.10.1.254")
s.expect(PacketOutputEvent('router-eth1', otroarp,
display=Arp))
otroarpresponse = mk_arpresp(otroarp, "11:00:00:00:00:00")
s.expect(PacketInputEvent("router-eth1", otroarpresponse,
display=Arp))
req_ping := req_ping.ttl-1
s.expect(PacketOutputEvent('router-eth1', req_ping,
display=IPv4))
rep_ping = mk_ping("11:00:00:00:00:00",
"10:00:00:00:00:02", "10.10.1.254", '2.2.2.2', True)
s.expect(PacketInputEvent('router-eth1', rep_ping,
display=IPv4))
s.expect(PacketInputTimeoutEvent(1))

```

- case 3: 在路由器子网中的源和目的进行ping通信，但是目的不进行Arp回复：
 1. 大体逻辑与case2 相同，但没有Arp回复，发送五次Arp请求后，将包丢弃。

```

req_ping = mk_ping("22:00:00:00:00:00",
'ff:ff:ff:ff:ff:ff', '192.168.1.2', '10.10.1.254')
s.expect(PacketInputEvent('router-eth0', req_ping,
display=IPv4))
req_ping = mk_ping('10:00:00:00:00:02', '11:00:00:00:00:00',
'192.168.1.2', '10.10.1.254', ttl=63)
...
s.expect(PacketOutputEvent('router-eth0', req_arp,
display=Arp))
repeat 4 times:
    s.expect(PacketInputTimeoutEvent(1.5))
    s.expect(PacketOutputEvent('router-eth0', req_arp,
display=Arp))

```

- case 4: 目的与源的mac地址均已经获知，检测其通信功能是否正常，总共有两次接收和两次发送（请求和回复），：

```

req_ping = mk_ping('33:00:00:00:00:00', '10:00:00:00:00:03',
'172.16.42.2', '10.10.1.254')
...#对应的Ip发送和回复包的接收和转发
s.expect(PacketInputEvent("router-eth2", req_ping,
display=IPv4))
s.expect(PacketOutputEvent("router-eth1", req_ping,
display=IPv4))
s.expect(PacketInputEvent("router-eth1", rep_ping,
display=IPv4))
s.expect(PacketOutputEvent("router-eth2", rep_ping,
display=IPv4))

```

- case 5: 目的地址相同的多个数据包到达路由器，，每等待1s都会有ARP请求包发送，在Arp回复到达后按到达顺序发送：

```

req_ping_1 =
mk_ping('33:00:00:00:00:00', '10:00:00:00:00:03',
        '172.16.42.2', '172.16.128.1', ttl=5)

req_ping_2 =
mk_ping('11:00:00:00:00:00', '10:00:00:00:00:02',
        '10.10.1.254', '172.16.128.1', ttl=4)

req_ping_3 =
mk_ping('33:00:00:00:00:00', '10:00:00:00:00:03',
        '172.16.42.2', '172.16.128.1', ttl=3)

...

s.expect(PacketInputTimeoutEvent(1), 'waiting for arp
reply')

s.expect(PacketOutputEvent("router-eth1", req_arp_1,
display=Arp))

```

测试结果

运行 `swyard -t lab_4routertests.py myrouter_to`，结果如下：

```

Results for test scenario Router stage 2 additional test 1: 3/ passed, 0 failed, 0 pending

Passes!
1 send A ping request to 172.16.42.1(router-eth2) arrive on
router-eth0
2 The dest is one of the interface, so router just drop it
3 send A ping request to 10.10.1.254 arrive on router-eth0
4 send A ping request to 10.10.1.254 leave out on router-eth1
5 Router receive an ARP request for 10.10.1.254 on router-
eth1 and prepare send the ping request to 10.10.1.254
6 forward 2.2.2.2 to 10.10.1.254 ping request leave out on
router-eth1
7 10.10.1.254 send ping reply to 2.2.2.2 arrive on router-eth1
8 Application should try to receive a packet, but then timeout
9 send A ping request to 10.10.1.254 arrive on router-eth0
10 forward 192.168.1.2 to 10.10.1.254 ping request leave out on
router-eth1
11 10.10.1.254 send ping reply to 192.168.1.2 arrive on router-
eth1
12 send Arp request for 192.168.1.2 leave out on router-eth0
13 Application should try to receive arp reply, but then
timeout
14 send Arp request for 192.168.1.2 leave out on router-eth0
15 Application should try to receive arp reply, but then
timeout
16 send Arp request for 192.168.1.2 leave out on router-eth0
17 Application should try to receive arp reply, but then
timeout
18 send Arp request for 192.168.1.2 leave out on router-eth0
19 Application should try to receive arp reply, but then
timeout
20 send Arp request for 192.168.1.2 leave out on router-eth0
21 Router should receive an unsolicited ARP response for
172.16.42.2 on router-eth1 and prepare send the ping request
to 10.10.1.254
22 172.16.42.2 ping for 10.10.1.254 arrive on router-eth2
23 172.16.42.2 ping for 10.10.1.254 leave out on router-eth1
24 10.10.1.254 response for 172.16.42.2 arrive on router-eth1
25 10.10.1.254 response for 172.16.42.2 leave out on router-
eth2
26 172.16.42.2 ping for 172.16.128.1 arrive on router-eth2
27 look up neighbor 10.10.1.254's mac
28 10.10.1.254 ping for 172.16.128.1 arrive on router-eth1
29 waiting for arp reply
30 repeat send arp request for neighbor 10.10.1.254's mac
31 172.16.42.2 ping for 172.16.128.1 arrive on router-eth2
32 waiting for arp reply
33 repeat send arp request for neighbor 10.10.1.254's mac
34 10.10.1.254 arp reply arrive an router-eth1
35 172.16.42.2 ping for 172.16.128.1 leave out on router-eth1
through 10.10.1.254
36 10.10.1.254 ping for 172.16.128.1 leave out on router-eth1
through 10.10.1.254
37 second 172.16.42.2 ping for 172.16.128.1 leave out on
router-eth1 through 10.10.1.254

All tests passed!

```

H6 II. 抓包测试

运行mininet，开启router运行 `swyard myrouter.py`，并在wireshark中监视 `eth0`和`eth2`的抓包情况；

运行 `xterm server1`，在server1中 `ping -c1 10.1.1.1`，观察两个端口抓包结果：

Capturing from router-eth2					Capturing from router-eth0				
No.	Time	Source	Destination	Protocol Length Info	No.	Time	Source	Destination	Protocol Length Info
0	0.00000000	40:00:00:00:00:03	10.1.1.1	ICMP	42	0.00000000	Private: 00:00:00:00:00:00	Broadcast	ARP
2	0.000018247	30:00:00:00:00:01	40:00:00:00:00:03	ARP	42	0.000018247	Private: 00:00:00:00:00:00	Private: 00:00:00:00:00:00	ARP
3	0.10514355	192.168.100.1	10.1.1.1	ICMP	98	0.000018247	192.168.100.1	10.1.1.1	ICMP
4	0.10537474	10.1.1.1	192.168.100.1	ICMP	98	0.000018247	10.1.1.1	192.168.100.1	ICMP
5	0.16690800	30:00:00:00:00:01	40:00:00:00:00:03	ARP	42	0.000018247	10.1.1.1	192.168.100.1	ICMP
6	0.22493816	40:00:00:00:00:03	30:00:00:00:00:01	ARP	42	0.000018247	10.1.1.1	192.168.100.1	ICMP

首先在端口 `eth2`（与10.1.1.1）相连的接口上收到了查询10.1.1.1的ARP请求结果，并向 `40:00:00:00:00:03` (`ip: 192.168.100.1`) 发送ARP回复；此后server1发送ping请求与client进行通信，并成功收到回复。

观察路由中的调试信息，也收到了相应的ARP请求和回复信息。

```

20:05:15 2020/04/11 INFO Got a ARP Request
20:05:15 2020/04/11 INFO router-eth0 10:00:00:00:01 192.168.100.1 00:00:00:00:00:00 192.168.100.2 1
20:05:15 2020/04/11 INFO update {IPv4Address('192.168.100.1'): (EthAddr('10:00:00:00:00:01'), 1586606715, 818919)}
20:05:15 2020/04/11 INFO Ethernet 40:00:00:00:00:01->10:00:00:00:00:01 ARP | Arp 40:00:00:00:00:01:192.168.100.2 10:00:00:00:00:01:192.168.100.1
catch an IP packet Ethernet 10:00:00:00:00:01->40:00:00:00:00:01 IP | IPv4 192.168.100.1->10.1.1.1 ICMP | ICMP EchoRequest 8230 1 (56 data bytes)
20:05:15 2020/04/11 INFO IPv4 pkt src: 192.168.100.1 dst: 10.1.1.1 ttl: 64 protocol: 1
Arp request: (<switchyard.lib.packet.packet.Packet object at 0x7fbb598027f0>, 'router-eth2')
20:05:16 2020/04/11 INFO Got a ARP Reply
20:05:16 2020/04/11 INFO router-eth2 30:00:00:00:00:01 10.1.1.1 40:00:00:00:00:00:00:00 10.1.1.2 2
20:05:16 2020/04/11 INFO update {IPv4Address('192.168.100.1'): (EthAddr('10:00:00:00:00:01'), 1586606715, 818919), IPv4Address('10.1.1.1'): (EthAddr('30:00:00:00:00:01'), 1586606716, 023538)}

```

以及对应的ping ICMP包。

```

20:05:16 2020/04/11 INFO Ethernet 40:00:00:00:00:03->30:00:00:00:00:01 IP | IPv4 192.168.100.1->10.1.1.1 ICMP | ICMP EchoRequest 8230 1 (56 data bytes)
catch an IP packet Ethernet 30:00:00:00:00:01->40:00:00:00:00:03 IP | IPv4 10.1.1.1->192.168.100.1 ICMP | ICMP EchoReply 8230 1 (56 data bytes)
20:05:16 2020/04/11 INFO IPv4 pkt src: 10.1.1.1 dst: 192.168.100.1 ttl: 64 protocol: 1
20:05:16 2020/04/11 INFO Ethernet 40:00:00:00:00:01->10:00:00:00:00:01 IP | IPv4 10.1.1.1->192.168.100.1 ICMP | ICMP EchoReply 8230 1 (56 data bytes)

```

紧接着在server1中运行 `ping -c 1 10.1.1.3`，可以观察到，在发送了五次查询 10.1.1.3 的ARP请求包后，便停止发包，ping失败，因为没有信息可以得知除了 `server1, server2, client` 以为结点的mac地址。与文档描述相符，故测试正确。

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.100.1	192.168.100.1	ARP	42	Who has 10.1.1.1? at 00:00:00:00:00:01
2	0.000000	Private_08:00:00:00:00:01	Private_08:00:00:00:00:01	ARP	42	Who has 10.1.1.1? at 00:00:00:00:00:01
3	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
4	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
5	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
6	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
7	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
8	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
9	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
10	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
11	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
12	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
13	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
14	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
15	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
16	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
17	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
18	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
19	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
20	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
21	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
22	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
23	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
24	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
25	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
26	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
27	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
28	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
29	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
30	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
31	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
32	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
33	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
34	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
35	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
36	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
37	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
38	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
39	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
40	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
41	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
42	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
43	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
44	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
45	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
46	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
47	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
48	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
49	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
50	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
51	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
52	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
53	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
54	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
55	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
56	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
57	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
58	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
59	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
60	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
61	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
62	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
63	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
64	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
65	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
66	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
67	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
68	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
69	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
70	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
71	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
72	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
73	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
74	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
75	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
76	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
77	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
78	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
79	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
80	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
81	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
82	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
83	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
84	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
85	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
86	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
87	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
88	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
89	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
90	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
91	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
92	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
93	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
94	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
95	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
96	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
97	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
98	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1
99	0.000000	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request 10-0-2026, seq=1
100	0.000000	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply 10-0-2026, seq=1

H3 4. 总结与感想

这次实验的整体难度直线上升，从前期的理解，板块设计，功能划分，以及运行调试都充满着艰难险阻。

本着先实现再优化的想法，将所有结构揉在一个类中，很多功能相似代码重复的地方，代码逻辑异常混乱。在调试时，最开始使用输出调试法，由于输出内容过多过乱，测试一度陷入僵局，并且遇到很多不明所以的错误信息，反复查阅switchyard文档并改用ide debug测试才变得顺利。在理论知识的学习过程中，很多事情逻辑都是理解个大概，细节很少去考虑，在实现匹配路由表和转发时才发现事情没有那么简单。将代码模块分成两个类，让结构变得更清晰。

实验的完成让人很有成就感，希望可以保持对这种知识探究的渴望和乐趣。

H3 5. 文档结构

