

Lab4

231220098 李东龙

Task1 match the IP address.

```
def find_interface(self,dst_ip):
    index=None
    max_match=0
    dst_ip=IPv4Address(dst_ip)
    for item in self.forwarding_table:

        tempnetaddr=IPv4Network(f'{item[0]}/{item[1]}',strict=False)
        if dst_ip in tempnetaddr:
            if tempnetaddr.prefixlen>max_match:
                max_match=tempnetaddr.prefixlen
                index=item
    return index
```

```
myrouter.py x start_mininet.py
myrouter.py > Router > handle_packet
31 # 从 dataqueue 中删除已发送的数据包
32 self.dataqueue.remove(item)
33
34
35 if arp.senderprotoaddr in self.arprequestsend:
36     del self.arprequestsend[arp.senderprotoaddr]
37
38
39
40
41
42 def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
43     timestamp, ifaceName, packet = recv
44     # TODO: your logic here
45     arp=packet.get_header(Arp)
46     icmp=packet.get_header(ICMP)
47     eth=packet.get_header(Ethernet)
48     ipv4=packet.get_header(IPv4)
49     for ip_addr in list(self.arp_cache.keys()):
50         if time.time() - self.arp_cache[ip_addr][1] >= 100.0:
51             del self.arp_cache[ip_addr]
52     if eth.dst not in self.mac_list and eth.dst!="ff:ff:ff:ff:ff:ff":
53         log.info("如果以太网目标既不是广播地址也不是传入端口的 MAC，则路由器应始终丢弃它，而不是执行查找过程。")
54         return
55     if eth.dst!="ff:ff:ff:ff:ff:ff" and self.port_list[self.mac_list.index(eth.dst)]!=ifaceName:
56         log.info(f'throw ')
57         return
58     if packet[Ethernet].ethertype == EtherType.VLAN:
59         log.info(f'throw packet with vlan')
60         return
61     if arp:
62         self.handle_arp(arp,ifaceName)
63         return
64     if ipv4:
65         dst_ip=ipv4.dst
66         if dst_ip in self.ip_list:
67             log.info("如果数据包是针对路由器本身的（即目标地址位于路由器的接口之间），则只需丢弃/忽略该数据包。我们也将：")
68             return
69         index=self.find_interface(dst_ip)
70         if index is None:
71             log.info("如果表中没有匹配项，请暂时丢弃数据包。我们将在实验 5 中处理此问题。")
72             return
73         else:
74             #print(self.arp_cache)
75             ipv4.ttl-=1
76             if index[2]==IPv4Address('0.0.0.0'):
77                 next_hop_ip=ipv4.dst
78             else:
```

首先要获取包头进行判断。如果目标 mac 地址不是端口的 mac 地址之一，则直接忽略。如果没有找到匹配项，则到实验 5 再处理。如果 ip 目标地址就是端口的 ip 地址，则到实验 5 再处理。

关于如何找到匹配项。

通过 network 和 subnet address 构成子网，判断目标 ip 在不在里面。如果找到多个匹配项，则取子网前缀最长的。

Task2

我是如何数据包的发送和 arp 请求。

```

# other initialization stuff here
def send_arprequest(self, targetip, iface_name, packet):

    if targetip in self.arprequestsend:
        self.dataqueue.append({"arp": targetip, "iface": iface_name, "data": packet})
        return

    self.arprequestsend[targetip] = {"iface": iface_name, "retries": 1, "last_sent": time.time()}
    iface = self.net.interface_by_name(iface_name)
    arp_request = create_ip_arp_request(iface.ethaddr, iface.ipaddr, targetip)
    self.net.send_packet(iface_name, arp_request)
    print(f"{targetip} has sent arp request")

    self.dataqueue.append({"arp": targetip, "iface": iface_name, "data": packet})

def arp_respond_loop(self):
    current_time = time.time()
    for targetip, info in list(self.arprequestsend.items()):

        if current_time - info["last_sent"] >= 1:
            if info["retries"] >= 5:

                print(f"ARP 请求超过5次失败, 删除目标 IP: {targetip}")
                del self.arprequestsend[targetip]

                self.dataqueue = [entry for entry in self.dataqueue if entry["arp"] != targetip]
            else:

                iface_name = info["iface"]
                iface = self.net.interface_by_name(iface_name)
                arp_request = create_ip_arp_request(iface.ethaddr, iface.ipaddr, targetip)
                self.net.send_packet(iface_name, arp_request)
                info["retries"] += 1
                info["last_sent"] = current_time
                print(f"{targetip} 第 {info['retries']} 次重发 ARP 请求")

```

```

def handle_arp(self, arp, iface_name):
    if arp.targetprotoaddr not in self.ip_list:
        print("not for me")
        return
    if arp.operation==ArpOperation.Request:

        self.arp_cache[arp.senderprotoaddr]=[arp.senderhwaddr,time.time()]

        if arp.operation==ArpOperation.Request:
            reply_mac=self.mac_list[self.ip_list.index(arp.targetprotoaddr)]
            reply_packet=create_ip_arp_reply(reply_mac,arp.senderhwaddr,
            arp.targetprotoaddr,arp.senderprotoaddr)
            self.net.send_packet(iface_name,reply_packet)

    elif arp.operation==ArpOperation.Reply:
        if arp.targethwaddr==EthAddr('ff:ff:ff:ff:ff:ff') or arp.senderhwaddr==EthAddr('ff:ff:ff:ff:ff:ff'):
            log_info(f'ARP reply is not allowed')
            return
        self.arp_cache[arp.senderprotoaddr]=[arp.senderhwaddr,time.time()]
        for item in list(self.dataqueue):
            # 检查等待 ARP 响应的 IP 和接口是否匹配
            if item["arp"] == arp.senderprotoaddr and item["iface"] == iface_name:
                packet = item["data"]
                eth = packet.get_header(Ethernet)
                eth.dst = arp.senderhwaddr
                eth.src = self.net.interface_by_name(iface_name).ethaddr
                self.net.send_packet(iface_name, packet)
                # 从 dataqueue 中删除已发送的数据包
                self.dataqueue.remove(item)

        if arp.senderprotoaddr in self.arprequestsend:
            del self.arprequestsend[arp.senderprotoaddr]

```

```

myrouter.py x start_mininet.py
myrouter.py > Router > send_arprequest
149 for ip_addr in list(self.arp_cache.keys()):
150     if time.time() - self.arp_cache[ip_addr][1] >= 100.0:
151         del self.arp_cache[ip_addr]
152     if eth.dst not in self.mac_list and eth.dst!="ff:ff:ff:ff:ff:ff":
153         log_info("如果以太网目标既不是广播地址也不是传入端口的 MAC，则路由器应始终丢弃它，而不是执行查找过程。")
154         return
155     if eth.dst!='ff:ff:ff:ff:ff:ff' and self.port_list[self.mac_list.index(eth.dst)]!=ifaceName:
156         log_info(f'throw ')
157         return
158     if packet[Ethernet].ethertype == EtherType.VLAN:
159         log_info(f'throw packet with vlan')
160         return
161     if arp:
162         self.handle_arp(arp,ifaceName)
163         return
164     if ipv4:
165         dst_ip=ipv4.dst
166         if dst_ip in self.ip_list:
167             log_info("如果数据包是针对路由器本身的（即目标地址位于路由器的接口之间），则只需丢弃/忽略该数据包。我们也将：")
168             return
169         index=self.find_interface(dst_ip)
170         if index is None:
171             log_info("如果表中没有匹配项，请暂时丢弃数据包。我们将在实验 5 中处理此问题。")
172             return
173         else:
174             #print(self.arp_cache)
175             ipv4.ttl-=1
176             if index[2]==IPv4Address('0.0.0.0'):
177                 next_hop_ip=ipv4.dst
178             else:
179                 next_hop_ip=index[2]
180             if ipv4.src==IPv4Address('31.0.1.1'):
181                 print(ipv4.dst)
182                 iface_name=index[3]
183                 if next_hop_ip in self.arp_cache:
184                     with open("file.txt","a") as file:
185                         file.write(str(next_hop_ip)+'\n')
186                     file.close
187                 next_hop_mac=self.arp_cache[next_hop_ip][0]
188                 eth.dst=next_hop_mac
189                 eth.src=self.net.interface_by_name(iface_name).ethaddr
190                 self.net.send_packet(iface_name,packet)
191             else:
192                 self.send_arprequest(next_hop_ip,iface_name,packet)
193                 print(f"find next but no mac(next_hop_ip)")
194

```

首先对包进行判断。如果包的以太网目标 mac 地址异常，则直接返回。

如果是 arp 包，则进入 arp 的判断。

如果是 ip 数据包，则进入 ip 的路由。

Arp 判断：如果目标地址不在端口 mac 之中，则直接返回。然后将 ip 与 mac 的信息加入到 arp_cache 表中。如果是 arprequest，则直接构造一个 arpreply 包回应即可。如果是 arp reply 包，就要找对应 ip 的数据包，把数据包全部发送出去，然后不要往对应的 ip 继续发送 arp 包了，把对应的 ip 在 arp 发送表里删除。

Arp 循环：对于一个 ip，所有数据包共享 5 次 arp 的发送，抄过 5 次，则所有的数据包都要删除，对应的 arp 发送表里面也要删除。遍历 arp 表，时间间隔大于 1s，则重发，次数+1。

Arp 发送：如果对应的 ip 没有在 arp 发送表里面，则加入，并且发送 arprequest，并添加待发数据包。如果对应的 ip 已经在 arp 里面了，则只要添加待发数据包。

Ip 数据发送：首先根据 find_interface 找到对应的接口。然后看接口的 next_hop_ip。如果是 0.0.0.0，则目的 ip 是 packet 的目的 ip，否则，就是 next_hop_ip。然后去 arp_cache 根据 ip 找 mac。如果找到了 ip，就直接发送了，如果没有找到，就调用 arp 发送。

```

File Edit Selection View Go Debug Terminal Help
start_mininet.py - lab-4-qlwy66 - Visual Studio Code

EXPLORER
  OPEN EDITORS
    myrouter.py 9*,U
    start_mininet.py 7,U
  LAB-4-QLWY66
    __pycache__
    .github
    testcases
    test_submit.py
    testscenario2_advanc...
    testscenario2.srpy
    cat U
    file.txt U
    forwarding_table... U
    myrouter.py 9*,U
    README.md
    result.txt U
    start_mininet.py 7,U

OUTLINE

TERMINAL
  1: bash
  1202Ping request from 31.0.6.1 should arrive on eth6
  1203Ping request from 31.0.6.1 should arrive on eth6
  1204Ping request from 31.0.6.1 should arrive on eth6
  1205Router should not do anything
  1206Bonus: V2FybWluZyB1cA==
  1207Bonus: V2FybWVkaHVw
  1208Bonus: V2h0CBkYyB5YSBob3BlIHQnIGZpbmQgaGVyZT8=

Failed:
  Bonus: SGFsZndheQ==
  Expected event: Timeout after 1.2s on a call to recv_packet

Pending (couldn't test because of prior failure):
  1 Bonus: Tm90aGluZyBmb3IgeWEgdCcgZmluZCB0ZXJlIQ==
  2 Bonus: Q29uZ3JhdHMH

*****

```

Client

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	40:00:00:00:00:03	Broadcast	ARP	42	who has 10.1.1.1? Tell 10.1.1.2
2	0.000032093	30:00:00:00:00:01	40:00:00:00:00:03	ARP	42	10.1.1.1 is at 30:00:00:00:00:01
3	0.102979837	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request id=0x185d, seq=1/256, ttl=63 (reply in 4)
4	0.103024573	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply id=0x185d, seq=1/256, ttl=64 (request in 3)
5	0.302038148	30:00:00:00:00:01	40:00:00:00:00:03	ARP	42	who has 10.1.1.2? Tell 10.1.1.1
6	0.375338511	40:00:00:00:00:03	30:00:00:00:00:01	ARP	42	10.1.1.2 is at 40:00:00:00:00:03

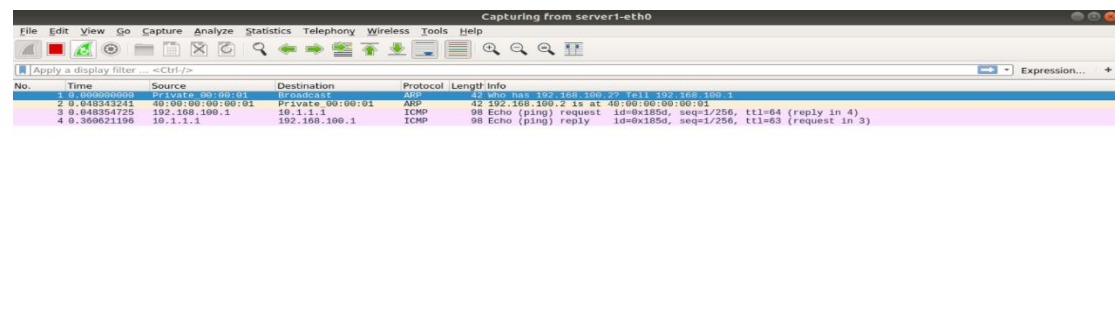
Router-eth0

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	40:00:00:00:00:03	Broadcast	ARP	42	who has 10.1.1.1? Tell 10.1.1.2
2	0.048332285	40:00:00:00:00:01	Private 00:00:00:01	ARP	42	192.168.100.2 is at 40:00:00:00:00:01
3	0.048398189	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request id=0x185d, seq=1/256, ttl=64 (reply in 4)
4	0.302038148	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply id=0x185d, seq=1/256, ttl=63 (request in 3)

Router-eth2

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	40:00:00:00:00:03	Broadcast	ARP	42	who has 10.1.1.1? Tell 10.1.1.2
2	0.000040856	30:00:00:00:00:01	40:00:00:00:00:03	ARP	42	10.1.1.1 is at 30:00:00:00:00:01
3	0.102979814	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request id=0x185d, seq=1/256, ttl=63 (reply in 4)
4	0.103024201	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply id=0x185d, seq=1/256, ttl=64 (request in 3)
5	0.302038142	30:00:00:00:00:01	40:00:00:00:00:03	ARP	42	who has 10.1.1.2? Tell 10.1.1.1
6	0.375338236	40:00:00:00:00:03	30:00:00:00:00:01	ARP	42	10.1.1.2 is at 40:00:00:00:00:03

server1



The image shows a Wireshark packet capture window titled "Capturing from server1-eth0". The interface includes a menu bar (File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help) and a toolbar. Below the toolbar is a display filter set to "Apply a display filter ... <Ctrl-/>". The packet list pane shows four captured packets:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Private::1	Private::1	ARP	42	Ethertype 102.168.100.2 is at 48:00:00:00:00:01
2	0.045343244	48:00:00:00:00:01	Private::1	ARP	42	Ethertype 102.168.100.2 is at 48:00:00:00:00:01
3	0.048354725	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request id=0x185d, seq=1/256, ttl=64 (reply in 4)
4	0.369621196	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply id=0x185d, seq=1/256, ttl=63 (request in 3)

我在 server1 里面 ping client。

首先，server1 不知道 router 接口的 mac 地址，于是有 arp-reply 和 arp-request。router 学习到了 server1 的 mac 信息。然后 server1 发送 ip 数据包到 router 的 eth-0。router 把 ip 包转到了 eth-2。此时又不知道 client 的 mac 地址，于是又有了 arp-reply 和 arp-request。然后数据包从 eth-2 发出，到达了 client。Client 原路返回 ip 数据包，最终到达了 server1。