实验报告

计算机网络-Lab4

专业：计算机科学与技术

任课教师：田臣

周迅 201220037

目录

一、 实验目的

模拟 IPv4 路由器，实现其基本功能的一部分。这一次，我们的路由器进化了——它要能够转发数据报，学会“最长前缀匹配”式查表；此外，它还要会发出 ARP 请求，来获取其他主机的未知 MAC。

二、 实验结果

- 完成了手册要求的基础功能，并通过官方测试集；
- 完成了实验报告（废话）；
- 设计了自己的测试集并通过；
- 使用 Wireshark 在仿真环境下抓包，验证路由器功能；
- 完成了多线程的可选任务，但难以检查其完全正确性，目前可以在 Mininet 搭建的仿真测试环境下正常工作。

三、 实验内容

1. Preparation

只要我们完成了 Lab3，那么这一阶段的事情又只剩下阅读手册了。

2. IP Forwarding Table Lookup

对于转发表，我们使用类来实现，且每一个表项（Entry）也用类来实现。每个表项中，需要有网络 IP、子网掩码、下一跳地址和转发接口。

```python
class ForwardingTableEntry:
    def __init__(self, network_ip: IPv4Address, subnet_mask: IPv4Address, next_hop
        self.ip_with_mask = IPv4Network(str(network_ip) + '/' + str(subnet_mask),
        self.next_hop_ip = next_hop_ip
        self.interface = interface
```

转发表，就是由这么一个个 Entry 组成，我们使用 list 来容纳，并提供使用文件和路由器端口两种初始化方式。

```python
class ForwardingTable:
    def __init__(self, file = None, router_interfaces = None):
        self.table = []
        # from file read forwarding table
        if file is not None:
            with open(file) as f:
```

匹配时，需遵循最长前缀匹配。所以，我们的列表应按前缀长度倒序排序。为了支持这一特性，重载表项的`__lt__()`方法，并在初始化转发表后进行排序。因为转发表是静态的，所以只要在初始时排一次即可。

```python
# sort by length of prefix, so that longest prefix match is supported
def __lt__(self, other):
    return self.ip_with_mask.prefixlen < other.ip_with_mask.prefixlen
```

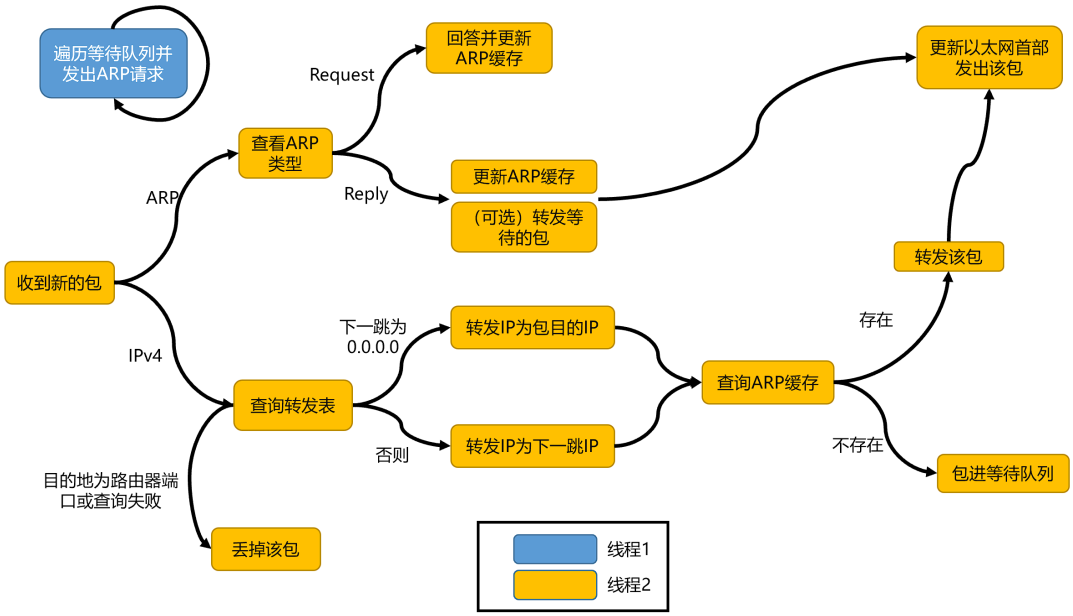
```python
# Sort the table by prefix length, so longer prefixe
self.table.sort(reverse=True)
```

然后，在匹配 IP 地址时，我们只要逐项检查就可以了。

```python
def match(self, ip_addr: IPv4Address):
    ''' find an entry to macth the given IP address '''
    for entry in self.table:
        if ip_addr in entry.ip_with_mask:
            return entry
    return None
```

3. Forwarding the Packet and ARP

完善转发表后，我们就要实现转发功能了。为了思路清晰，我们首先给出处理的流程表（由 PPT 支持，若不清晰，可查看同目录下的演示图片原图）。



对于一个普通的 packet，我们首先把它的 TTL 减 1，为后续老化机制的实现奠定基础。

```
ip_header = packet.get_header(IPv4)
if ip_header is None:
    return
ip_header.ttl -= 1
```

接着，我们上面实现的转发表就排上用场了。这里有特殊情况：如果该包的终点就是路由器某个端口，或者转发表中查无此“址”，那这个包就要丢掉。

```
# Check if the packet is destined for this router, drop it
if ip_header.dst in [iface.ipaddr for iface in self.net.interfaces()]:
    return

match_entry = self.forwarding_table.match(ip_header.dst)
# If there is no match in the table, just drop the packet
if match_entry is None:
    return
```

接下来，我们分两种情况讨论。如果说，包的目的地址与路由器某接口在同一子网下，那我们直接把这个目的 IP 作为我们转发的目的 IP。我们并不需要再次遍历路由器所有接口，因为这种情况下，匹配到的下一跳地址为“0.0.0.0”；否则（也就是可以看到一个正常的下一跳地址），就把下一跳地址作为转发的目的 IP。

```
forward_ip = (match_entry.next_hop_ip
              if match_entry.next_hop_ip != IPv4Address('0.0.0.0')
              else ip_header.dst)
```

IP 搞定后，就得把 MAC 整好。这时，我们上个实验的 ARP 缓存就派上用场了。查询缓存表，查到，那当然最好；查不到的话，就得问问了。

对于 ARP 中没有相应内容的情况，为了不延误手边工作，我们把包，连带其他一些信息（转发接口、时间、ARP 询问次数）放进等待集合，并以目的 IP 作为键值，便于查询。

```
self.waiting_packets[forward_ip] = (packet, match_entry.interface, time.time() - 1, 0)
```

每隔约 1s，路由器就会查看等待队列，针对每个目的 IP 发出 ARP 请求；问了 5 次还没回答的话，那就没耐心了，扔了得了。

```
time_out_record = []
for next_hop_ip, (packet, forward_interface, timestamp, times) i
    # we have tried 5 times, drop the packet
    if times >= 5:
        # del self.waiting_packets[next_hop_ip]
        time_out_record.append(next_hop_ip)
    # make a request every 1 second
    elif time.time() - timestamp >= 1.0:
        sender_mac = self.net.interface_by_name(forward_interfac
        sender_ip = self.net.interface_by_name(forward_interface
        arp_request = create_ip_arp_request(sender_mac, sender_i
        self.net.send_packet(forward_interface, arp_request)
        self.waiting_packets[next_hop_ip] = (packet, forward_int
# avoid delete while iterating
for next_hop_ip in time_out_record:
    del self.waiting_packets[next_hop_ip]
```

一旦获取了 MAC 地址（无论是通过 ARP 还是查询缓存），我们就把包发出去。注意一个细节：由于 MAC 地址在子网中才有其意义，所以在以太网首部中，我们还得把源 MAC 地址替换为转发端口的 MAC。

```
def forward_packet(self, packet, dst_mac, iface) -> None:
    eth_header = packet.get_header(Ethernet)
    if eth_header is None:
        return
    # Ethernet src also needs to be updated,
    # since the packet will enter another network.
    eth_header.src = self.net.interface_by_name(iface).ethaddr
    eth_header.dst = dst_mac
    self.net.send_packet(iface, packet)
```

对于 ARP 请求，我们在上个实验已经搞定，不再赘述了。对于 ARP 回应，首要的便是更新缓存表。接着，查询等待队列，如有包等着回应的 MAC 地址，那就可以把包发出去了。

```
self.arp_tab[arp.senderprotoaddr] = arp.senderhwaddr
if self.muti_thread:
    self.lock.acquire()
if arp.senderprotoaddr in self.waiting_packets:
    packet, forward_interface, timestamp, times = self.waiting_packet
    self.forward_packet(packet, arp.senderhwaddr, forward_interface)
    del self.waiting_packets[arp.senderprotoaddr]
if self.muti_thread:
    self.lock.release()
```

4. More details

在获取 MAC 这一步，我们支持单线程和多线程两种实现。

对于单线程，只需在两处地方检查等待队列：

1) 每次处理完一个包；

2) 没有新的包到达（等待时间超过 1s）。

```
try:
    recv_pkt = self.net.recv_packet(timeout=1.0)
except NoPackets:
    # We should send arp request,
    # since last request may be lost
    if not self.muti_thread:
        self.make_arp_request()
    continue
```

多线程方式下，我们只额外需要创建一个线程，不断查询等待队列并发出询问（或者把包扔掉）。为了避免资源过多浪费，每轮查询后，该线程陷入 0.5s 的休眠。

```
while True:
    time_out_record = []
    for next_hop_ip, (packet, forward_interface
    self.lock.acquire()
    for next_hop_ip in time_out_record:
        del self.waiting_packets[next_hop_ip]
    self.lock.release()
    time.sleep(0.5)
```

多线程下，等待队列为临界资源（Critical Resource），安全起见，需要在每一处访

问该资源的地方加上互斥锁。

```
if forward_ip not in self.arp_tab:
    if self.muti_thread:
        self.lock.acquire()
    self.waiting_packets[forward_ip] = (packet, match_e
    if self.muti_thread:
        self.lock.release()
```

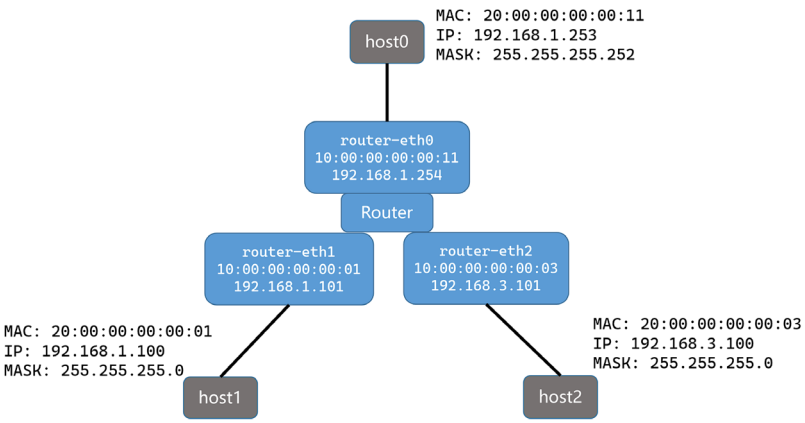
5. Testing

由于 Switchyard 测试脚本对收发数据包的顺序有严格的要求，所以该阶段首先使用单线程版测试。

首先，晒一下官方测试集的结果。

```
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!
```

在我们自己的测试中，首先给出该路由器所处状态。如下图，路由器连接 3 个子网，且 0 接口子网 IP 为 1 接口子网 IP 的“前缀”。



路由器转发表如下（忽略路由器端口）：

IP	Mask	Next Hop	Interface
192.168.1.0	255.255.255.0	192.168.1.100	router-eth1
192.168.1.252	255.255.255.252	192.168.1.253	router-eth0
192.168.3.0	255.255.255.0	192.168.3.100	router-eth2

测试样例包括以下几个：

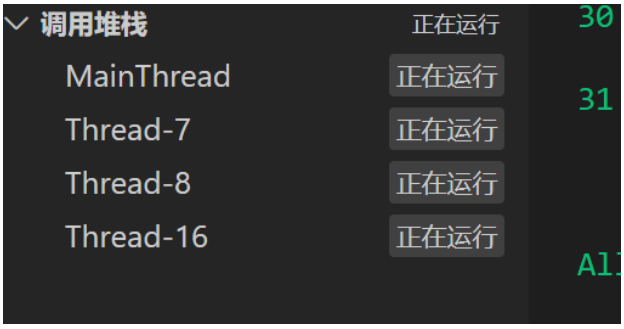
1) host0 向路由器发送 ARP 请求，以获得 eth0 接口的 MAC；同时，路由器应记住 host0 的 MAC；
2) host2 向 host0 发送一个普通 IPv4 包，路由器应且仅应转发该包（同时 TTL 减 1）；
3) host0 向 host1 发送一个普通 IPv4 包，路由器应询问 host1 的 MAC，得到回复后再转发；
4) host2 向 host1 发送一个普通 IPv4 包，路由器应且仅应转发该包（同时 TTL 减 1）；
5) host0 向 host2 发送一个普通 IPv4 包，路由器 ARP 请求 host2 的 MAC 地址，未得到回复，重复 5 次后丢掉该包，不再请求。

代码实现可以看 Github，这里贴个测试通过的结果。

```
11  Other hosts should not receive any packet
12  A packet should arrive on router-eth2
13  A packet should be forwarded out router-eth1
14  Other hosts should not receive any packet
15  A packet should arrive on router-eth0
16  An ARP request should be sent out router-eth2
17  resend ARP request
18  resend ARP request
19  resend ARP request
20  resend ARP request
21  Packet should be dropped, the router will nerver request


All tests passed!
```

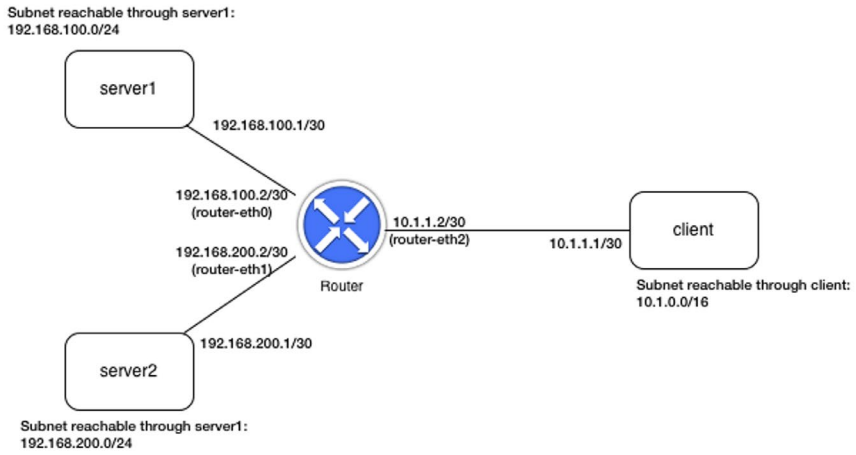
接着，我们来看看多线程的测试。好吧，测试都可以过，但不知道为啥似乎停不下来，得我们自己按以下终止键……（没关系，我们可以在仿真测试环境下做）

```
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!

(swmtenv) njucs@njucs-VirtualBox:~/NetLabWork/lab-04-SkyerWalkery$
```



6. Deploying

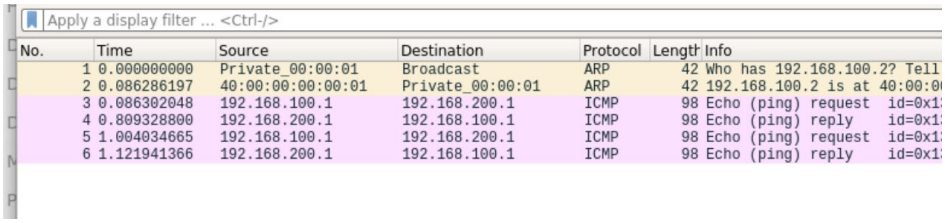
在仿真测试环境下，我们的配置和转发表如下图：



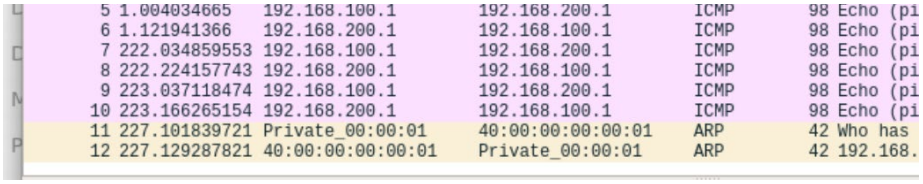
IP	Mask	Next Hop	Interface
192.168.100.0	255.255.255.0	192.168.100.1	router-eth0
192.168.200.0	255.255.255.0	192.168.200.1	router-eth1
10.1.0.0	255.255.0.0	10.1.1.1	router-eth2

由于我们需要从 server1 发出 ping，所以使用 Wireshark 聚焦 router-eth0。

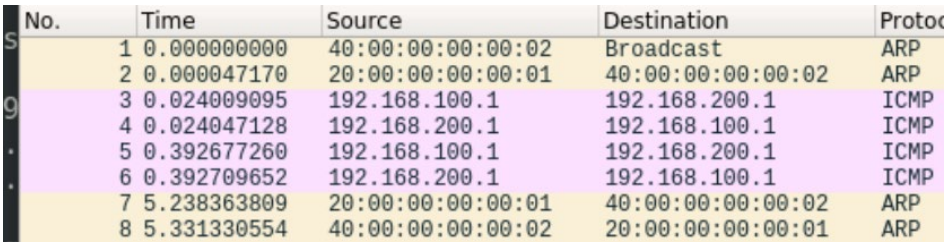
首先，`server1 ping -c2 192.168.200.1`。如图，server1 会发出 ARP 请求，询问 router-eth0 的 MAC，获取后发出 ICMP 包。



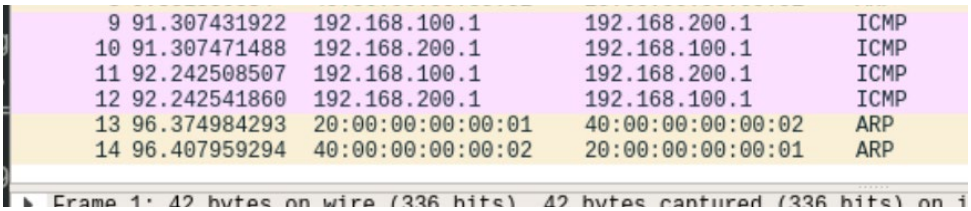
再次 `server1 ping -c2 192.168.200.1`。最后一轮 ARP 是对 MAC 地址的确认。



我们从 server2 的视角。如图，路由器会询问 server2 的 MAC，获取后转发包；最后 server2 发出确认路由器 MAC。



再次 `server1 ping -c2 192.168.200.1`。这一次，我们的路由器没有再询问了（因为已经在 ARP 缓存表中了）。



上述两次仿真测试观察均已通过 Wireshark 导出抓包文件，详情可参考同目录下的 `lab_4_1.pcapng` 和 `lab_4_2.pcapng`。

上述测试使用的是多线程版。由此，我们在有限、简单的仿真环境下验证了多线程版的部分正确性（起码暂时没啥 BUG）。

四、 核心代码

要不您往上边翻一翻，都有截图；或者，上 Github 看看代码？

五、 总结与感想

- 这次实验，无论是任务量还是难度，相比起上一个实验都大幅增长（还好提前开始做的），感觉可以把两个实验并到一起，然后给 4 周的时间；
- 发出 ARP 请求这块，从表现上讲就适合多线程，也算略微巩固了 OS 课程中并发的有关知识；
- 对于包的首部在传输过程中的变化，以及各个协议的适用范围等，都有了更深的体会（STFW 比书本好用多了）。