Computer Networking-Lab-Repot

课程名称：**计算机网络** 任课教师：**田臣/李文中**

学院	Dept. of Computer Science and Technology	专业（方向）	CS
学号	181830044	姓名	董宸郅
Email	pdon#foxmail.com	开始/完成日期	4.9/4.22

实验名称：IPv4 Router: Forwarding Packets

实验目的

- 进一步完善**Router**功能
- 深入理解**Router**的转发机制
- 学会合理组织代码结构以同时处理收发任务
- 加强在实验环境下调试代码的能力

实验内容

理论知识

ARP

地址解析协议(Address Resolution Protocol)，是根据IP地址获取物理地址的一个TCP/IP协议。主机发送信息时将**包含目标IP地址的ARP请求广播到局域网络上的所有主机**，并接收返回消息，以此确定目标的物理地址；收到返回消息后将该IP地址和物理地址**存入本机ARP缓存中并保留一定时间**，下次请求时直接查询ARP缓存以节约资源。

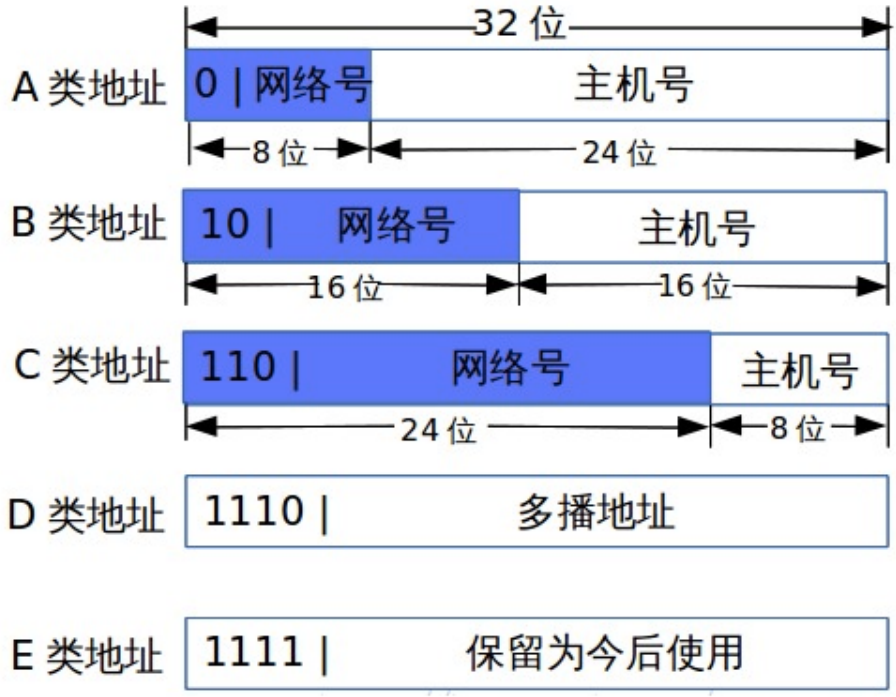
路由表

在路由器中保存着各种传输路径的相关数据，从而为经过路由器的每个数据包寻找一条最佳的传输路径。在每个表项中，一般含有**dst IP、interface、next hop IP、subnet mask**等关键信息。

IPv4地址格式

IPv4使用32位（4字节）地址，由网络号与主机号组成，分为五类（如下图所示）：

IP地址分为五大类：A类、B类、C类、D类和E类，如下图所示：



子网掩码

是一种用来指明一个IP地址的哪些位标识的是主机所在的子网，以及哪些位标识的是主机的位掩码。通过计算机的子网掩码判断两台计算机是否属于同一网段的方法是，将计算机的IP地址和子网掩码转换为二进制的形式，然后进行与运算，如果得出的结果是相同的，那么这两台计算机就属于同一网段。

最长前缀匹配

路由表中的每个表项都指定了一个网络，一个目的地址可能与多个表项匹配。最明确的一个表项（即子网掩码最长的一个）就叫做最长前缀匹配，这个表项也是路由表中与目的地址的高位匹配得最多的表项。

实验步骤（含测试结果与关键代码）

Task 2: IP Forwarding Table Lookup

Coding

将初始化路由表的任务单独成Router类的一个method：

```
def init_fwd_tab(self):
    '''
    Initialize the forwarding table from 2 sources:
        1. the router's own interfaces
        2. the file named 'forwarding_table.txt'
    A simple list is used to hold each entry in the following order:
        1. network address
        2. subnet mask
        3. next hop IP
        4. intf to forward the packet
```

```
    '''
    # add src 1
    for intf in self.intfs:
        entry = []
        entry.append(str(intf.ipaddr))
        entry.append(str(intf.netmask))
        entry.append('0.0.0.0') # next hop is NONE in this case
        entry.append(intf.name)
        self.fwd_tab.append(entry)

    # add src 2
    for line in open("forwarding_table.txt"):
        entry = line.split()
        self.fwd_tab.append(entry)
```

使用最长前缀匹配的方式判断路由表中是否有符合目标IP的表项，同时丢掉发给路由器本身的包：

```
# 2. handle IPv4 packets
ipv4 = pkt.get_header(IPv4)
if (ipv4):
    # drop the pkt if the dst belong to the router itself
    if ipv4.dst not in self.ipaddrs:
        # denote the longest prefix match so far
        longest = 0
        # denote the updated matched entry
        matched_entry = []
        for entry in self.fwd_tab:
            prefixnet = IPv4Network(entry[0] + '/' + entry[1],
strict=False)
            if (ipv4.dst in prefixnet) and (longest < prefixnet.prefixlen):
                longest = prefixnet.prefixlen
                matched_entry = entry

```

Task 3: Forwarding the Packet and ARP

Coding

准备工作，新增一个等待队列来存放还未收到ARP reply的表项，使用list实现，组成见注释：

```
# init the waiting queue for ARP reply
# composition of every entry in queue:
#   1. the next hop ip addr (used for requesting the corresponding mac addr)
#   2. last_request_time (interval < 1)
#   3. cnt-->times of requesting (cnt <= 5)
#   4. matched forwarding table entry
#   5. IPv4 packet(x)    original packet
#   6. ARP request packet
self.wait_q = []
```

以下为收发IPv4与ARP的总体代码逻辑：

- **收到了包**
  - **是ARP**

- **request**
  - 在Lab 3中已实现过

```
if (arp.operation == ArpOperation.Request):
    # add a new entry into the ARP cache table or just update a recorded one
    self.arp_tab[arp.senderprotoaddr] = arp.senderhwaddr
    log_info("Cached ARP table updated: {}".format(str(self.arp_tab)))

    # drop it if target ip does not exist here
    if arp.targetprotoaddr in self.ipaddrs:
        wanted_macaddr = self.net.interface_by_ipaddr(arp.targetprotoaddr).ethaddr
        arp_reply = create_ip_arp_reply(wanted_macaddr, arp.senderhwaddr, arp.targetprotoaddr, arp.senderprotoaddr)
        self.net.send_packet(dev, arp_reply)
```

- **reply**
  - 将**IP-ARP**对加入ARP缓存，并找出这是waiting queue中哪个表项正在等待的ARP回复；找到后，修改原来的Ethernet报头，将其发出，并从队列中将这个表项删除

```
if (arp.operation == ArpOperation.Request):
    # add a new entry into the ARP cache table or just update a recorded one
    self.arp_tab[arp.senderprotoaddr] = arp.senderhwaddr
    log_info("Cached ARP table updated: {}".format(str(self.arp_tab)))

    # drop it if target ip does not exist here
    if arp.targetprotoaddr in self.ipaddrs:
        wanted_macaddr = self.net.interface_by_ipaddr(arp.targetprotoaddr).ethaddr
        arp_reply = create_ip_arp_reply(wanted_macaddr, arp.senderhwaddr, arp.targetprotoaddr, arp.senderprotoaddr)
        self.net.send_packet(dev, arp_reply)
```

- **是IPv4**
  - 终点不是路由器本身且经最长前缀匹配，路由表中存在匹配的项

```
# drop the pkt if the dst belong to the router itself
if ipv4.dst not in self.ipaddrs:
    # denote the longest prefix match so far
    longest = 0
    # denote the updated matched entry
    matched_entry = []
    for entry in self.fwd_tab:
        prefixnet = IPv4Network(entry[0] + '/' + entry[1], strict=False)
```

```
        if (ipv4.dst in prefixnet) and (longest < prefixnet.prefixlen):
            longest = prefixnet.prefixlen
            matched_entry = entry
    # drop the pkt if no matches found
    if matched_entry != [] :
        # assume ttl >= 0
        pkt.get_header(IPv4).ttl -= 1
        intf = self.net.interface_by_name(matched_entry[3])
    # 1. the dst is within the subnet to which the intf belong,
    #    which means the next hop is the dst
    if matched_entry[2] == '0.0.0.0':
        next_hop_ip = ipv4.dst
    # 2. the next hop is an IP address on a router through which the destination is reachable
    else:
        next_hop_ip = IPv4Address(matched_entry[2])
```

- ARP缓存中已有目标IP对应的MAC地址

```
# if the IP-MAC pair is already recorded in the ARP cache table,
# then no need for an ARP request
if next_hop_ip in self.arp_tab:
    dst_macaddr = self.arp_tab[next_hop_ip]
    pkt.get_header(Ethernet).src = intf.ethaddr
    pkt.get_header(Ethernet).dst = dst_macaddr
    self.net.send_packet(matched_entry[3], pkt)
```

- ARP缓存中没有有目标IP对应的MAC地址，发送ARP request并将其加入队列

```
# ARP request is necessary when none recorded
else:
    # create a new ARP request using the handy API
    arp_rqst = create_ip_arp_request(intf.ethaddr, intf.ipaddr, next_hop_ip)
    self.net.send_packet(matched_entry[3], arp_rqst)
    # add this request into waiting queue
    new_entry = [next_hop_ip, time.time(), 1, matched_entry, pkt, arp_rqst]
    self.wait_q.append(new_entry)
```

- **没收到包**

```
# when no pkt is received, especially ARP
else:
    # update every entry state in the waiting queue
    for entry in self.wait_q[:]:
        # ARP reply is not received after 1s
        if time.time() - entry[1] > 1:
            # ARP request already sent exactly 5 times
            if (entry[2] >= 5):
                self.wait_q.remove(entry)
```

```
            # still be able to send ARP request
            else:
                entry[1] = time.time()
                entry[2] += 1
                self.net.send_packet((entry[3])[3], entry[5])
```

Testing

```
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
```

Deploying

在server1的cli中输入以下命令：

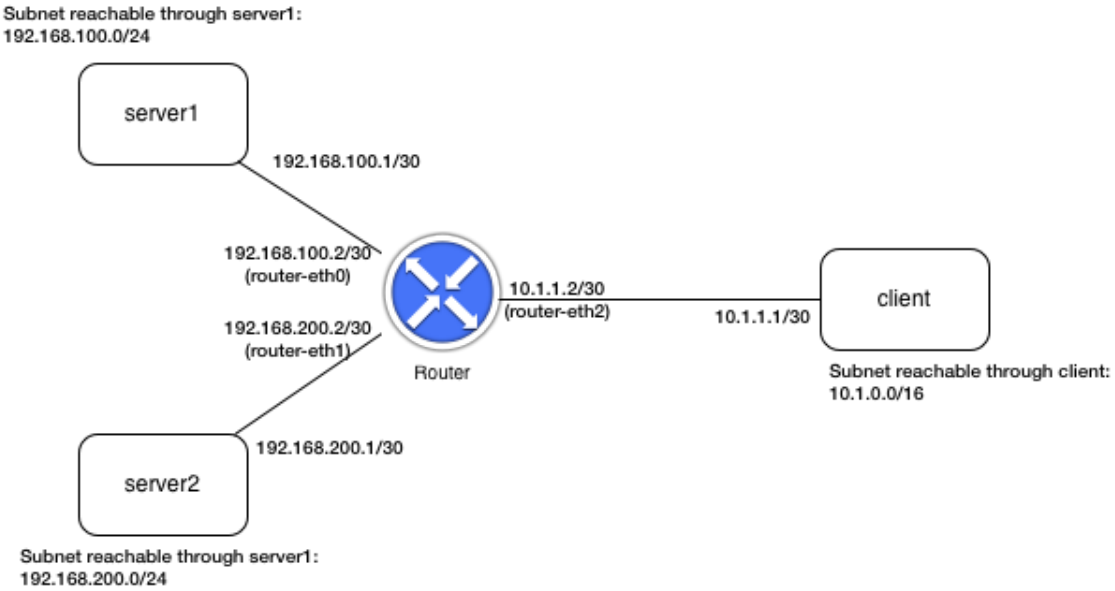
```
# from server1 to server2
ping -c2 192.168.200.1
```

router-eth0处的wireshark截图：

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.100.1	192.168.200.1	ICMP	98	Echo (ping) request id=0x4704, seq=1/256, ttl=64 (reply in 4)
2	0.286059400	40:00:00:00:00:01	Broadcast	ARP	42	Who has 192.168.100.1? Tell 192.168.100.2
3	0.286076344	Private_00:00:01	40:00:00:00:00:01	ARP	42	192.168.100.1 is at 10:00:00:00:00:01
4	0.396012359	192.168.200.1	192.168.100.1	ICMP	98	Echo (ping) reply id=0x4704, seq=1/256, ttl=63 (request in 1)
5	0.999867444	192.168.100.1	192.168.200.1	ICMP	98	Echo (ping) request id=0x4704, seq=2/512, ttl=64 (reply in 6)
6	1.146002314	192.168.200.1	192.168.100.1	ICMP	98	Echo (ping) reply id=0x4704, seq=2/512, ttl=63 (request in 5)
7	5.140823270	Private_00:00:01	40:00:00:00:00:01	ARP	42	Who has 192.168.100.2? Tell 192.168.100.1
8	5.150273569	40:00:00:00:00:01	Private_00:00:01	ARP	42	192.168.100.2 is at 40:00:00:00:00:01

router-eth1处的wireshark截图：

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	40:00:00:00:00:02	Broadcast	ARP	42	Who has 192.168.200.1? Tell 192.168.200.2
2	0.000019506	20:00:00:00:00:01	40:00:00:00:00:02	ARP	42	192.168.200.1 is at 20:00:00:00:00:01
3	0.103810557	192.168.100.1	192.168.200.1	ICMP	98	Echo (ping) request id=0x4704, seq=1/256, ttl=63 (reply in 4)
4	0.103844287	192.168.200.1	192.168.100.1	ICMP	98	Echo (ping) reply id=0x4704, seq=1/256, ttl=64 (request in 3)
5	0.957946370	192.168.100.1	192.168.200.1	ICMP	98	Echo (ping) request id=0x4704, seq=2/512, ttl=63 (reply in 6)
6	0.958002345	192.168.200.1	192.168.100.1	ICMP	98	Echo (ping) reply id=0x4704, seq=2/512, ttl=64 (request in 5)
7	5.319511119	20:00:00:00:00:01	40:00:00:00:00:02	ARP	42	Who has 192.168.200.2? Tell 192.168.200.1
8	5.384171639	40:00:00:00:00:02	20:00:00:00:00:01	ARP	42	192.168.200.2 is at 40:00:00:00:00:02



首先，第一个ICMP request到达Router-eth0端口，在经过判断后，Router得出路由表里含有符合这个目标IP(192.168.200.1)的网段信息，而且这个目标IP与Router-eth1端口**直接相连**，但没有对应的MAC地址。于是Router在eth1端口向外发送ARP request，得到答复后，向server2转发了那个ICMP request。在server2收到这个request之后，向server1发去一个对应的reply，Router同样判断这个包可以被转发，且由于之前已经记录过server1的IP-MAC对，所以不需要发送ARP request而直接发给server1。

接下来还有第二次ICMP request和reply，所不同的是由于Router中**已经存有相应IP-MAC对**，故不需要发送ARP包，其余原理一致。

总结与感想

这次实验的代码量比前几次要多一些，但我数了一下也就几十行，对比其他科目就也还行（不是）。这次的难点在于要提前理清流程图，**合理设计好代码结构**，以保证收发数据包同时进行。

在我刚开始设计的时候，并没有想到在if gotpkt之后加一个与之平行的else来更新没有收到ARP reply的队列，而是把这个任务合并在if gotpkt里面，和收包同时进行。事实证明这样是错误的，在test scenerio中测试时，可以发现这样会导致有ARP reply不能收到。

我第一次调整时，把更新队列的代码复制到了与if gotpkt平行的地方，但仍保留了原来位置的代码，这样通过了测试。但后来我想，应该不太可能两个地方要放同样的代码，于是试着分别在两处安插了断点，再次运行测试，发现自始至终只有下方新增的代码在发挥作用，于是删去了上方代码。经过思考，我理解了功能逻辑就应该是现在这样，并在上方报告处给出了相应的逻辑流程。

最后想说，课程的进度还是很快的，很多时候会有跟不上节奏的感觉。而且，课本中的诸多细节其实在lab中被简化了，我们做的只是最简单的版本，所以还是要多看书，争取更全面地理解网络的知识体系！！