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　　助教：

学院	**计算机科学与技术系**	专业（方向）	**计算机科学与技术**
学号	**211220049**	姓名	**石璐**
Email	**211220049@smail.nju.edu.cn**	开始/完成日期	**2023 年 5 月 4 日**

1. 实验名称：Forwarding Packets

2. 实验目的

接收和转发到达链路并以其他主机为目的地的数据包。转发过程的一部分是在转发表中进行地址查找（"最长前缀匹配"查找）。

为没有已知以太网 MAC 地址的 IP 地址发出 ARP 请求。路由器经常需要向其他主机发送数据包，并需要以太网 MAC 地址来实现。

3. 实验内容

Task 1：IP Forwarding Table Lookup

✅ **Coding：**

1) Build Forwarding Table

- 定义转发表项

```
12  class ForwardingTableItem:
13      def __init__(self,ipaddr,mask,nexthop,interface):
14          self.ipaddr = IPv4Address(ipaddr)
15          self.mask = IPv4Address(mask)
16          self.nexthop = IPv4Address(nexthop)
17          self.interface = interface
```

- 建立转发表

（注：按前缀长度降序排序，便于"最长前缀匹配"查找）

```
23  class ForwardingTable:
24      def __init__(self,interfaces):
25          # from router interfaces
26          self.table = [ForwardingTableItem(intf.ipaddr,intf.netmask,'0.0.0.0',intf.name) \
27                        for intf in interfaces]
28          # from file
29          with open('forwarding_table.txt') as f:
30              for line in f.readlines():
31                  line = line.strip('\n')
32                  ipaddr,mask,nexthop,interface = line.split(' ')
33                  self.table.append(ForwardingTableItem(ipaddr,mask,nexthop,interface))
34          # sort forwarding table
35          self.table.sort(key=prefix_len,reverse=True)
```

2) Match Destination IP Addresses against Forwarding Table

由于转发表已经按前缀长度降序排序，因此顺序遍历的第一个匹配结果即为"最长前缀匹配"查找的结果。

```
23  class ForwardingTable:
24      def __init__(self,interfaces):…
41
42      def search(self,ipaddr):
43          for item in self.table:
44              prefixnet = IPv4Network(f'{item.ipaddr}/{item.mask}',strict=False)
45              if ipaddr in prefixnet:
46                  return item
47          return None
```

Task 2：Forwarding the Packet and ARP

☑ Coding：

- **等待队列 waitingtable**

  采用字典结构，key 为未知 mac 地址的 IP 地址，value 为 waitinglist 类，类中存储了 4 项信息：

  a) **time：**针对该 ip 地址最近一次发送 ARP request 包的时间

  b) **trycnt：**剩余可发送 request 的次数，初始为 5 次，减为 0 时将丢弃队列中等待的 packets

  c) **intf：**对应的路由器端口

  d) packets：等待发送到该 ip 的 packet 队列，保序

```
56  class Router(object):
57      def __init__(self, net: switchyard.llnetbase.LLNetBase):
58          ...
59          self.waitingtable = {}
```

```
49  class WaitingList:
50      def __init__(self,intf):
51          self.time = time.time()
52          self.trycnt = 5
53          self.intf = intf
54          self.packets = []
```

- **处理等待队列**

  每次通过主循环时处理队列中的项目，看看是否需要发送 ARP 请求重传或丢弃等待的 packets。

```
165     def start(self):
166         '''A running daemon of the router.
167         Receive packets until the end of time.
168         '''
169         while True:
170             try:
171                 self.handle_waitinglist()
172                 recv = self.net.recv_packet(timeout=1.0)
```

```
149     def handle_waitinglist(self):
150         for ip,wtlst in self.waitingtable.items():
151             if len(wtlst.packets) > 0 and time.time() - wtlst.time >= 1:
152                 # Retransimission
153                 if wtlst.trycnt > 0:
154                     arp_request = create_ip_arp_request(wtlst.intf.ethaddr,wtlst.intf.ipaddr,ip)
155                     log_info (f"Sending packet {arp_request} to {wtlst.intf.name}")
156                     self.net.send_packet(wtlst.intf, arp_request)
157                     self.waitingtable[ip].time = time.time()
158                     self.waitingtable[ip].trycnt -= 1
159                 # Drop packets
160                 else:
161                     self.waitingtable[ip].trycnt = 5
162                     self.waitingtable[ip].packets.clear()
```

- **检查 Ethernet header**

```
76          # Drop the packet with Ethernet destination neither a broadcast address
77          # nor the MAC of the incoming port.
78          if ethheader and ethheader.dst != "ff:ff:ff:ff:ff:ff" \
79                       and ethheader.dst != self.net.interface_by_name(ifaceName).ethaddr:
80              return
81          # Drop the packet with VLAN
82          if ethheader.ethertype == EtherType.x8021Q:
83              return
```

- **处理 IPv4 packet**

i. 将 ttl 字段递减 1

ii. 在转发表中查找 ip 目标地址对应的表项

iii. 如果 nexthop 为 0.0.0.0，则下一跳的 ip 地址设置为表项 ipaddr 域

iv. 为要转发的 IP 数据包创建一个新的 Ethernet header。此时需要下一跳的 mac 地址，首先查询 arp 表，如果查到直接转发，如果未查到则将该数据包放入等待队列并发送 arp 请求报文。

```
113 if ipheader:
114     # Drop the packet for the router itself.
115     # TODO: handle in lab 5
116     if ipheader.dst not in self.ipaddrs:
117         packet[IPv4].ttl -= 1
118         # Process the packet not expired.
119         # TODO: handle in lab 5
120         if packet[IPv4].ttl > 0:
121             item = self.forwardingtable.search(packet[IPv4].dst)
122             # Drop the packet if there is no match in the table.
123             # TODO: handle in lab 5
124             if item:
125                 intf = self.net.interface_by_name(item.interface)
126                 # The IP address 0.0.0.0 means that if a packet's destination IP matches this n
127                 # just throw the packet out of this interface, expecting that the next hop is t
128                 dstip = item.nexthop
129                 if dstip == IPv4Address('0.0.0.0'):
130                     dstip = packet[IPv4].dst
```

```
131                 # search ARP Table for dst mac
132                 if dstip in self.arptable:
133                     dstmac = self.arptable[dstip]
134                     packet[0] = Ethernet(src=intf.ethaddr,dst=dstmac,ethertype=EtherType.IPv4)
135                     log_info (f"Sending packet {packet} to {intf.name}")
136                     self.net.send_packet(intf,packet)
137                 # send ARP request for dst mac
138                 else:
139                     if dstip not in self.waitingtable:
140                         self.waitingtable[dstip] = WaitingList(intf)
141                     if len(self.waitingtable[dstip].packets) == 0:
142                         arp_request = create_ip_arp_request(intf.ethaddr,intf.ipaddr,dstip)
143                         log_info (f"Sending packet {arp_request} to {intf.name}")
144                         self.net.send_packet(intf, arp_request)
145                         self.waitingtable[dstip].time = time.time()
146                         self.waitingtable[dstip].trycnt -= 1
147                     self.waitingtable[dstip].packets.append(packet)
```

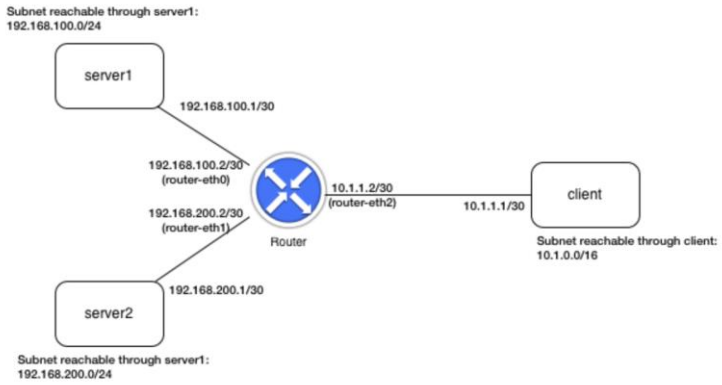
- 处理 ARP packet
  - i. 更新 ARP 表。
  - ii. 处理 ARP request。发送 ARP 回复报文。
  - iii. 处理 ARP reply。完成要转发的 IP 数据包的以太网头，并与等待队列中的 packets 一起发送。

```
84  # Only process the ARP packet when its destination IP is held by a port of the router.
85  if arpheader and arpheader.targetprotoaddr in self.ipaddrs:
86      # Cached ARP Table
87      if arpheader.operation == ArpOperation.Request or arpheader.senderhwaddr != "ff:ff:ff:ff:ff:ff":
88          self.arptable[arpheader.senderprotoaddr] = arpheader.senderhwaddr
89      # Handle ARP packet
90      if arpheader.targetprotoaddr in self.ipaddrs:
91          # Handle ARP request
92          if arpheader.operation == ArpOperation.Request:
93              intf = self.net.interface_by_name(ifaceName)
94              reply_mac = self.net.interface_by_ipaddr(arpheader.targetprotoaddr).ethaddr
95              arp_reply = create_ip_arp_reply(reply_mac, arpheader.senderhwaddr, \
96                          arpheader.targetprotoaddr, arpheader.senderprotoaddr)
97              log_info (f"Sending packet {arp_reply} to {intf.name}")
98              self.net.send_packet(intf, arp_reply)
99          # Handle ARP reply
100         elif arpheader.operation == ArpOperation.Reply:
101             if arpheader.senderhwaddr != "ff:ff:ff:ff:ff:ff":
102                 ipaddr,ethaddr = arpheader.senderprotoaddr,arpheader.senderhwaddr
103                 if len(self.waitingtable[ipaddr].packets) > 0:
104                     intf = self.waitingtable[ipaddr].intf
105                     e = Ethernet(src=intf.ethaddr,dst=ethaddr,ethertype=EtherType.IPv4)
106                     for waiting_packet in self.waitingtable[ipaddr].packets:
107                         waiting_packet[0] = e
108                         log_info (f"Sending packet {waiting_packet} to {intf.name}")
109                         self.net.send_packet(intf,waiting_packet)
110                     self.waitingtable[ipaddr].packets.clear()
111                     self.waitingtable[ipaddr].trycnt = 5
```

✅ **Testing:**

```
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: V2FybSB1cA==
1207Bonus: Q29vbCBkb3du
1208Bonus: V2h1dCBkJyB5YSBob3BlIHQnIGZpbmQgaGVyZT8=


Failed:
    Bonus: Tm90aGluJyBmb3IgeWEgdCcgZmluZCBoZXJlIQ==
        Expected event: recv_packet Ethernet
        30:00:00:00:05:02->20:00:00:00:00:05 IP | IPv4
        15.0.0.92->192.168.129.2 UDP | UDP 0->0 | RawPacketContents
        (35 bytes) b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'... on
        eth5


Pending (couldn't test because of prior failure):
1   Q29uZ3JhdHMh


************************************************************
Your code didn't crash, but a test failed.
************************************************************

This is the Switchyard equivalent of the blue screen of death.
As far as I can tell, here's what happened:

    Expected event:
        Bonus: Tm90aGluJyBmb3IgeWEgdCcgZmluZCBoZXJlIQ==

    Failure observed:
        send_packet was called instead of recv_packet Ethernet
        30:00:00:00:05:02->20:00:00:00:00:05 IP | IPv4
        15.0.0.92->192.168.129.2 UDP | UDP 0->0 | RawPacketContents
        (35 bytes) b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'... on
        eth5
```

☑ **Deploying：执行 server1# ping -c2 10.1.1.1**



i. server1(192.168.100.1)欲向 client 发送 ICMP 包，首先要向 router 的 eth0 端口发送，因此 server1 询问 eth0 端口(192.168.100.2)的 mac 地址，在 eth0 收到一条 192.168.100.1 对 192.168.100.2 的 ARP 请求报文。

ii. router 回复上述 ARP 请求，在 eth0 发送一条 192.168.100.2 对 192.168.100.1 的 ARP 回复报文。

iii. server1 得到 eth0 的 mac 地址后开始发送 ICMP 报文，因此在 eth0 收到一条 192.168.100.1 对 10.1.1.1 的 ICMP 报文。

iv. router 查询转发表后将该报文导向 eth2 端口准备发向 client，但是要先询问 10.1.1.1 的 mac 地址，因此在 eth2 发送一条 10.1.1.2 对 10.1.1.1 的 ARP 请求报文，同时将 ICMP 报文缓存起来。

v. client 回复上述 ARP 请求，在 eth2 收到一条 10.1.1.1 对 10.1.1.2 的 ARP 回复报文。

vi. router 得到 client 的 mac 地址后发送之前缓存的 ICMP 报文，因此在 eth2 发送一条 192.168.100.1 对 10.1.1.1 的 ICMP 报文。

vii. 接下来，client 回复该 ICMP 报文，因此在 eth2 收到 ICMP 回复报文，再向 eth0 转发该报文到 server1。

viii. 在前面接受 ARP 报文时 router 的 ARP 表学习了 client 和 server1 的 mac 地址，因此第二次 ICMP 报文的请求和回复可以直接经由 router 进行转发，不需要再发送 ARP 请求。

ix. 最后 eth2 中的 2 条 ARP 报文用于定期检查缓存。