Computer Network* Lab 4

- **徐思源 191220133**
- Department of Computer Science and Technology
- Nanjing University
- 1357307497@qq.com

& 1 理论思路

- 本次实验主要就是实现一个有转发功能的路由器
- 经过路由器的包，分成ICMP和ARP两种：
  - 针对ICMP类型的包：路由器需要找到对应的路径将它发送出去，首先从forwarding_table中去寻找对应的路径，即下一跳的地址，然后根据得到的ip去缓存中找相应的mac地址（这里涉及到通过收发arp包找到mac地址），并将包发送出去
  - 针对ARP类型的包：当ip找不到对应的mac地址的时候，我们需要从发送请求类型的ARP包去询问对应的mac地址，收到回复类型的arp包的时候，需要将缓存中符合条件的数据包发送出去，收到请求类型的arp包和lab3一样

& 2 具体实现

- **初始化forwarding_table:**
  - 第一部分从txt文件中读入：每一项的四个内容分别是前缀，掩码，下一跳地址和发送出去的接口名，根据指导文件中的教学，对前缀和掩码进行处理就可以得到该路由器可以管辖的范围。（范围，下一跳地址，端口名）构成forwarding_table的一个表项
  - 对于路由器接口而言，接口对应的ip就是可以直接到达的范围，所以下一跳地址为None，说明而可以直接发送到目的ip上，根据接口处的ip地址和掩码处理得到前缀，之后和上面一种情况一样的处理就可以获得对应的forwarding_table表项
  - 需要注意的点是转化成ipv4的地址格式
  - 代码实现如下

```
class forwarding_table():
    table=[]
    def __init__(self,interfaces):
        f=open("forwarding_table.txt","r")
        for line in f:
            prefix,mask,next_hop,intf=line.strip().split(' ')
            prefixnet =
IPv4Network(str(IPv4Address(prefix))+'/'+str(IPv4Address(mask)))

 self.table.append(forwarding_item(prefixnet,IPv4Address(next_hop),i
ntf))
        f.close()
        for i in interfaces:
            y = (int(i.ipaddr) & int(i.netmask))
            prefixnet =
IPv4Network(str(IPv4Address(y))+'/'+str(IPv4Address(i.netmask)))

 self.table.append(forwarding_item(prefixnet,None,i.name))
```

- **完善forwarding_table的查找功能：**
  - 根据前面对于初始化的描述，我们的forwarding_table中存储的表项告诉我们的信息是，找到当前包所属管辖范围（即dest in prefixnet) 并把包从对应的端口发送到下一跳的地址上去，而管辖范围是可以有包含关系的，找到prefixlen最长的项就意味着找到范围最小的那个对应点，这样效率更高（指导文件的要求），返回找到的符合条件的表项
  - 代码实现如下：

```
class forwarding_table():
    table=[]
    //initial
    def getmatched(self, destaddr):
        maxlen=-1
        res=None
        for i in self.table:
            if i.match(destaddr) and i.getlen()>maxlen:
                maxlen=i.getlen()
                res=i
        return res
```

- **对收到的ipv4包进行处理：**
  - 首先根据目的地在forwarding_table当中查找对应的表项，找到匹配的那个，根据其中的端口名找到对应的端口interface，如果下一跳是None，说明就在当前路由器可以到达的范围内了，从缓存中获取目的地址的mac地址，下一跳目的地址就是包的目的ip，直接将包从对应端口发送出去就可以（这里需要考虑缓存中没有对应的mac地址，需要进一步发送ARP请求的情况，将在后续报告中阐述），否则说明我们需要跳到另一个区域中去（实现是跳到下一个路由器上去，所以也有一个具体的ip地址），同样获取对应的mac地址，进行发送
  - 代码实现如下：

```
if ipv4 is not None:              # recv a ipv4 packet
            matched = self.forwarding_table.getmatched(ipv4.dst)
            if matched is not None:
                try:
                    interface =
self.net.interface_by_name(matched.interface)
```

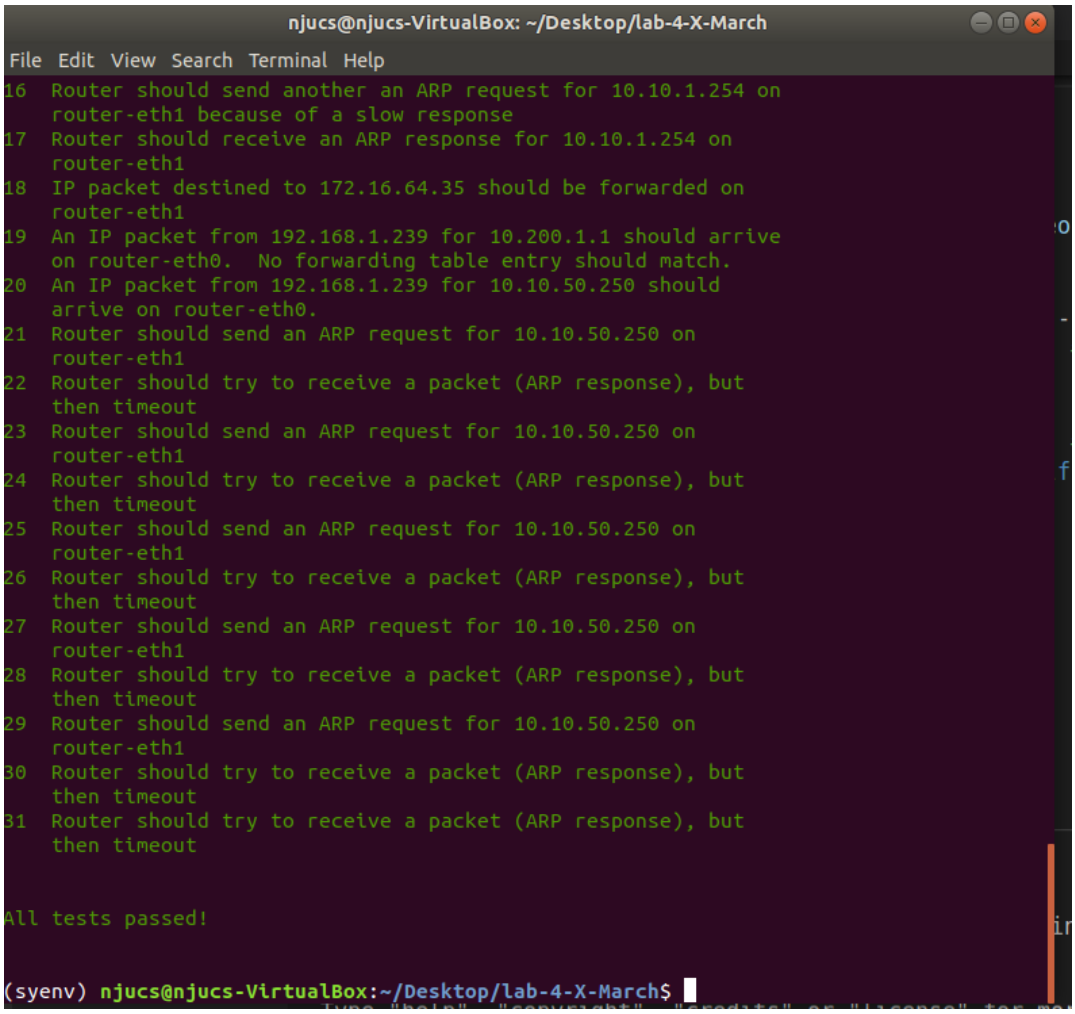
```
                except KeyError:
                    interface = None
                ipv4.ttl -= 1
                if ipv4.ttl <= 0:
                    pass
                if matched.next_hop is not None: #jump to next_hop
                    next = matched.next_hop
                    targetmac =
self.ip_mac_table.get(matched.next_hop)
                else: #send directly
                    next = ipv4.dst
                    targetmac = self.ip_mac_table.get(ipv4.dst)
                if targetmac is not None:
                    packet[0].dst = targetmac
                    packet[0].src = interface.ethaddr
                    # send
                    self.net.send_packet(matched.interface,packet)
                else:
                    # send arp and add it into queue to wait for
reply。。。
 else:
                pass # not found match ip in forwading table
```

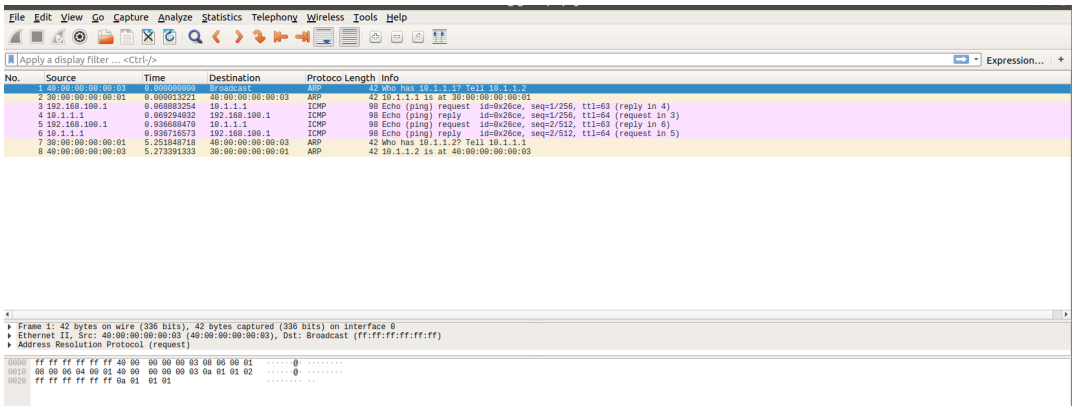
- **对上述情况中没有在缓存中找到mac地址，需要发送ARP请求的情况进行处理：**
  - 我们已知通过哪个端口可以到达目的地，但是并不知道具体的地址，首先构建ARP请求的包，询问这个具体的mac地址，从已知的端口发送出去
    - ```
      arp_request =
      create_ip_arp_request(interface.ethaddr,interface.ipaddr,next)
      self.net.send_packet(matched.interface,arp_request)
      ```
  - 由于获取到mac地址之前不能直接发送这个包，我们要对没有发送出去的包进行缓存，构建一个队列来缓存这些数据包，队列的一项保存如下的数据，由于根据指导文件，5次发送arp请求没有reply才能丢弃这个数据包，所以我们需要记录发送的arp请求包来方便下一次发送，存储需要发送的数据包（其中，由于我们已经找到了将这个包发送出去的那个对应端口，这里我们可以先更新它的src，dst留作收到reply时赋值），记录发送出去的那个端口名，记录时间戳，发送arp请求的次数，初始化0，
    - ```
      class q_item():
          def __init__(self,arp_request,packet,next,interface,ifaceName):
              self.arp=arp_request #该数据包发送出去的arp请求包
              self.packet=packet #数据包
              self.next_hop=next #下一跳
              self.out=interface #发送端口
              self.src=ifaceName #从哪个端口来
              self.time=time.time() #时间戳
              self.sendtimes=0 #请求次数
      ```
  - 将这个未发送的数据包存储在队列当中：
    - ```
      packet[0].src=interface.ethaddr
      self.q.append(q_item(arp_request,packet,next,matched.interface,iface
      Name))
      ```
      ```

- **针对收到arp回复的包进行处理：**
  - 收到reply说明之前出现过有数据包因为没有找到对应的mac地址而不能发送的情况，所以这里我们遍历之前构建的缓存队列，找到符合条件的数据包（即下一跳地址为reply发送方地址的情况），对目的地址的mac进行赋值（也就是reply包发送方的mac地址）全部从对应端口发送出去：
  - 代码实现如下：
    - ```
      if arp.operation == 2: #reply send the packets
          for i in range(len(self.qi)-1,-1,-1):
              if arp.senderprotoaddr == self.q[i].next_hop:
                   # creat eth header
                   self.q[i].packet[0].dst = arp.senderhwaddr
                   # send
                   self.net.send_packet(self.q[i].out,self.q[i].packet)
                   # pop
                   self.q.pop(i)
      ```
- **针对收到arp请求的情况和lab3一样，略**
- **针对超时和五次请求没有收到回复的情况的处理：**
  - 发送请求次数已经超过5次，直接丢弃，否则，如果等待响应的时间超过1，再将arp请求发送出去，同时请求次数加1
  - 代码实现如下：
    - ```
      # timeout and resend
              now = time.time()
              for i in range(len(self.q)-1,-1,-1):
                  if self.q[i].sendtimes >= 4: #send 5 times drop the
      packet
                      self.q.pop(i)
                      continue
                  if now - self.q[i].time > 1: # Timeout , resend arp pkt
                      self.net.send_packet(self.q[i].out,self.q[i].arp)
                      self.q[i].sendtimes+=1
      ```
  - 需要注意的是，这个分析不管有没有收到包都要持续进行，所以需要在收不到包的情况下也加上这个步骤，如图：
    - ```
      while True:
          try:
              recv = self.net.recv_packet(timeout=1.0)
          except NoPackets:
              now = time.time()
              for i in range(len(self.q)-1,-1,-1):
                  if self.q[i].sendtimes >= 4: #send 5 times drop the packet
                      self.q.pop(i)
                      continue
                  if now - self.q[i].time > 1: # Timeout , resend arp pkt
                      self.net.send_packet(self.q[i].out,self.q[i].arp)
                      self.q[i].sendtimes+=1
              continue
          except Shutdown:
              break

          self.handle_packet(recv)
      ```
      ```

& 3 实验结果和分析

- **测试用例：**
  - 

- **mininet运行和抓包结果分析：**
  - 
    - 用server1 ping client，首先server1发送广播信号请求router的地址，并将icmp数据包发送给router，之后router从另一个端口去询问client的地址并且在得到回复之后发送icmp数据包给client，符合我们代码的实现