# 南京大学本科生实验报告

课程名称: 计算机网络

任课教师: 黄程远

学院	计算机学院	专业 (方向)	计算机科学与技术
学号	221220147	姓名	杨洋
Email	3517098084@qq.com	开始/完成日期	2024.10.30/2024.11.7

#### 1. 实验名称

Lab 4: Forwarding Packets

### 2. 实验目的

建立转发表,并匹配目的地址,即实现 IP 转发表查找功能。发送 ARP 查询以创建新的以太网头部,并转发数据包,即实现基于 ARP 的数据包转发功能。

### 3. 实验内容

Task 1: Preparation

Task 2: IP Forwarding Table Lookup

Task 3: Forwarding the Packet and ARP

#### 4. 实验结果

测试结果如下

只有 bonus 多线程测试无法通过,其他的都通过了。

```
Router should try to receive a packet (ARP response), but then timeout

Router should send an ARP request for 10.10.50.250 on router-eth1

Router should try to receive a packet (ARP response), but then timeout

Router should send an ARP request for 10.10.50.250 on router-eth1

Router should try to receive a packet (ARP response), but then timeout

Router should send an ARP request for 10.10.50.250 on router-eth1

Router should send an ARP request for 10.10.50.250 on router-eth1

Router should try to receive a packet (ARP response), but then timeout

Router should try to receive a packet (ARP response), but then timeout

All tests passed!

(syenv) njucs@njucs-VirtualBox:~/workspace/lab-4-221220147$ Traceback (most recent call last):
```

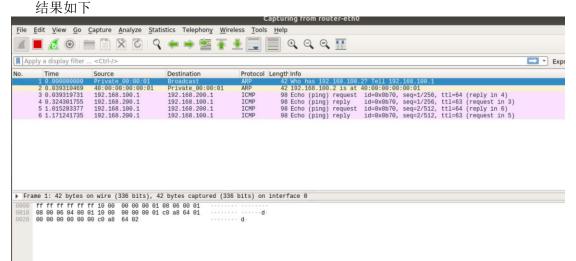
```
njucs@njucs-VirtualBox: ~/workspace/lab-4-221220147

File Edit View Search Terminal Help

192Ping request from 31.0.5.1 should arrive on eth5
193Ping request from 31.0.5.1 should arrive on eth5
194Ping request from 31.0.5.1 should arrive on eth5
195Ping request from 31.0.5.1 should arrive on eth5
195Ping request from 31.0.5.1 should arrive on eth5
196Ping request from 31.0.5.1 should arrive on eth5
197Ping request from 31.0.5.1 should arrive on eth6
198Phong request from 31.0.6.1 should arrive on eth6
208Ping request from 31.0.6.1 s
```

### Mininet 部署

观察 router 的 eth0 接口,由 server1 向 server2 发两个包。



当 Server1 需要与 Server2 建立连接时,它会首先发送一个 ARP 请求以获取 Server2 的 MAC 地址。Server2 收到这个请求后,会回复一个 ARP 响应,这个响应首先到达 Router,然后由 Router 转发给 Server1。在这个过程中,Router 记录下了 Server1 和 Server2 的 IP 地址与MAC 地址的映射关系。因此,当 Router 后续接收到来自 Server1 的数据包时,它能够直接利用之前缓存的地址信息将数据包转发给 Server2。

## 5. 核心代码

整体实现逻辑如下

ARPCache: 管理 ARP 缓存,存储 IP 地址和对应的 MAC 地址,并检查缓存是否超时。

Routeltem: 代表路由表中的一个条目,包含网络地址、子网掩码、待匹配 IP 和出口端口,用于判断 IP 地址是否匹配该路由。

RoutingTable:构建整个路由表,从文件加载路由规则,并提供查找最佳匹配路由条目的方法。

ARPRequestQueueltem: 处理单个ARP请求,包括请求本身和等待发送的IPv4数据包。

ARPRequestQueue : 管理 ARP 请求队列,处理 ARP 请求的添加、响应和重发。

Router 类:接收数据包。根据路由表和 ARP 缓存转发 IPv4 数据包。处理 ARP 请求和响应,更新 ARP 缓存。管理 ARP 请求队列,确保数据包在知道下一跳 MAC 地址后发送。

```
class RouteItem:
    def __init__(self, ip, mask, next_ip, portname):
        ipnum = IPv4Address(ip)
        masknum = IPv4Address(mask)
        self.ip = IPv4Address(int(ipnum) & int(masknum))
        self.mask = IPv4Address(masknum)
        if next_ip is not None:
            self.next_ip = IPv4Address(next_ip)
        else:
            self.next_ip = None
        self.portname = portname
        self.prefixnet = IPv4Network(str(self.ip) + '/' + mask)

def prefixlen(self):
        return self.prefixnet.prefixlen

def match(self, desaddr):
        return desaddr in self.prefixnet

def printf(self):
        log_info("ip:{} mask:{} nextdes:{} port:{}".format(self.ip, self.mask, self.next_ip, self.portname))
```

prefixlen 函数:这个函数的作用是返回当前路由表项的前缀长度。

match 函数:这个函数用于判断一个给定的目标 IP 地址是否与当前路由表项匹配。它通过比较目标 IP 地址和表项的网络地址来判断是否属于同一网络。如果目标 IP 地址落在当前表项定义的网络范围内,那么函数返回 True,表示匹配成功。

```
class RoutingTable:
   def __init__(self, interfaces):
    self.table = []
        f = open('forwarding table.txt', 'r')
            ip, mask, desaddr, iface = lines.strip().split(' ')
            self.table.append(RouteItem(ip, mask, desaddr, iface))
        for ifa in interfaces:
            self.table.append(RouteItem(str(ifa.ipaddr), str(ifa.netmask), None, ifa.name))
   def find match(self, desipaddr):
       maxlen = -1
        for it in self.table:
            if it.match(desipaddr) and it.prefixlen() > maxlen:
                maxlen = it.prefixlen()
        return res
   def printf(self):
        for it in self.table:
            it.printf()
```

find\_match 函数:这个函数的目的是找到与给定目标 IP 地址匹配的、前缀长度最大的路由表项。函数遍历路由表中的所有项,找到与目标 IP 地址匹配且前缀长度最大的项,并返回这个项。

```
class ARPRequestQueueItem:
    def __init__(self, arp, port):
        self.arp = arp
        self.pkt = []
        self.time = time.time()
        self.times = 1
        self.port = port

def add_packet(self, pkt):
        self.pkt.append(pkt)

def matches(self, senderaddr):
        return self.arp.get_header(Arp).targetprotoaddr == senderaddr
```

arp: 存储 ARP 请求的数据包。

pkt: 存储等待发送的数据包列表。

time: 记录请求被加入队列的时间。

times: 记录请求发送的次数。

port: 记录发送请求的端口。

```
class ARPRequestQueue:
      self.arp q = []
      self.net = net
   def add_item(self, arp, pkt, port):
      for item in self.arp_q:
          if item.arp.get_header(Arp).targetprotoaddr == arp.get_header(Arp).targetprotoaddr:
             item.add_packet(pkt)
      log info("!!!")
      newitem = ARPRequestQueueItem(arp, port)
      newitem.add_packet(pkt)
      self.arp q.append(newitem)
      self.net.send_packet(port, arp)
  def getreply(self, arp, interface):
       log_info("Reply! {}".format(arp.senderprotoaddr))
       for it in self.arp q:
           if it.matches(arp.senderprotoaddr):
                for pkt in it.pkt:
                     pkt[0].dst = arp.senderhwaddr
                    pkt[0].src = interface.ethaddr
                     self.net.send packet(interface, pkt)
                self.arp q.remove(it)
                return True
   def resend(self):
       now = time.time()
       for it in self.arp q:
           if it.times >= 5:
                self.arp q.remove(it)
           elif now - it.time > 1.0:
                log info("Fa Le")
                log info(it.arp.get header(Arp).targetprotoaddr)
                self.net.send packet(it.port, it.arp)
                it.time = time.time()
                it.times += 1
```

add item 函数: 向队列中添加一个新的 ARP 请求队列项。

getreply 函数: 当收到 ARP 响应时,根据响应中的发送者 IP 地址找到对应的请求 队列项,并发送等待的数据包。

resend 函数: 定期检查队列中的请求是否超时,如果超时则重新发送 ARP 请求。

```
def __init__(self, net: switchyard.llnetbase.LLNetBase):
    self.net = net
    self.interfaces = self.net.interfaces()
    self.arp_record = ARPCache()
    self.route_record = RoutingTable(self.interfaces)
    self.arp_queue = ARPRequestQueue(self.net)
def process_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
    timestamp, ifaceName, packet = recv
        interface = self.net.interface by name(ifaceName)
    if interface == None or (packet[0].dst != interface.ethaddr and packet[0].dst != 'ff:ff:ff:ff:ff:ff'):
    log_info(packet.headers())
    log_info(interface.name)
    log_info(interface.ipaddr)
    log_info(interface.ethaddr)
    arp = packet.get_header(Arp)
    ipv4 = packet.get_header(IPv4)
    if arp is not None:
    log_info("have arp arp protal is {}".format(arp.targetprotoaddr))
           port = self.net.interface_by_ipaddr(arp.targetprotoaddr)
            port = None
        log info(port)
        if port != None:
            if arp.operation == 1:
                self.arp_record.add_item(arp.senderprotoaddr, arp.senderhwaddr)
                forward_item = self.route_record.find_match(arp.senderprotoaddr)
                if forward item is not None:
                    replypkt = create_ip_arp_reply(port.ethaddr, arp.senderhwaddr, arp.targetprotoaddr, arp.send
self.net.send_packet(interface, replypkt)
                e:
    if arp.senderhwaddr != 'ff:ff:ff:ff:ff':
        self.arp_record.add_item(arp.senderprotoaddr, arp.senderhwaddr)
                    self.arp_queue.getreply(arp, interface)
    if ipv4 is not None:
         forward_item = self.route_record.find_match(ipv4.dst)
         if forward_item is None:
              ipv4.ttl -= 1
              if ipv4.ttl <= 0:
                   if forward item.next ip is None:
                       nextip = ipv4.dst
                       nextip = forward item.next ip
                   port = self.net.interface_by_name(forward_item.portname)
                   nextmac = self.arp_record.get_item(nextip)
                       packet[0].dst = nextmac
                       packet[0].src = port.ethaddr
                       self.net.send_packet(port, packet)
log_info("???")
                       log_info(ipv4.dst)
                       log_info(forward_item.portname)
                       log_info(nextip)
                       flag = True
                       for item in self.interfaces:
                           if item.ipaddr == nextip:
                                flag = False
                           arp_request = create_ip_arp_request(port.ethaddr, port.ipaddr, nextip)
                           self.arp_queue.add_item(arp_request, packet, port)
```

process\_packet 函数: 处理接收到的每个数据包,包括 ARP 请求和 IPv4 数据包。

run 函数: 持续运行,接收数据包并处理。

stop 函数:停止路由器的运行。