

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。

```
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

All tests passed!

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


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。

RouteItem：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。

ARPRequestQueueItem：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')
        for lines in f:
            ip, mask, desaddr, iface = lines.strip().split(' ')
            self.table.append(RouteItem(ip, mask, desaddr, iface))
        f.close()
        for ifa in interfaces:
            self.table.append(RouteItem(str(ifa.ipaddr), str(ifa.netmask), None, ifa.name))

    def find_match(self, desipaddr):
        maxlen = -1
        res = None
        for it in self.table:
            if it.match(desipaddr) and it.prefixlen() > maxlen:
                res = it
                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:
    def __init__(self, net):
        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)
                return True
        log_info("!!!")
        newitem = ARPRequestQueueItem(arp, port)
        newitem.add_packet(pkt)
        self.arp_q.append(newitem)
        self.net.send_packet(port, arp)
        return False

    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。

`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。