实验报告

一、实验名称

Lab 4: Forwarding Packets

二、实验目的

实现路由器的两个功能：

1. 接收并转发到达链路并且目的地是其他主机的数据包。转发过程中的一部分是在转发表中进行地址查询。
2. 为没有已知以太网MAC地址的IP地址发送ARP请求。路由器通常需要向其他主机发送数据包，需要以太网MAC地址来完成这个过程。

三、实验内容

Forwarding Table

根据自己的interface（自己的子网） 和 forwarding_table.txt

建立forwarding table，其中有四个表项，dest, mask, gateway, interface

计算前缀长度，对forwarding table进行排序，前缀长的放在前面，从前向后遍历时第一个匹配到的就是最长前缀匹配

IPV4 Packet

如果是发给自己ip的包，那么不处理

在forwarding table里面查找对应的表项，如果没有表项就不处理

确定next_hop_ip，有下一跳即为下一跳，没有即为目标ip

在arp_table中查找next_hop_mac，

找得到的话直接把包转发出去

找不到的话需要发arp请求来问，如果已经发过就不发了

(bonus：检查数据包的总长度是否等于以太网头的长度加上IPv4头中指定的总长度)

Arp Packet

判断ip是否是发给它的，如果不是就丢掉

然后把来源的ip和mac加入自己的arptable里面

对于arp请求，创建arp回复，填写好自己的mac地址并发回去

对于arp回复，查询等待队列中是否有相应ip的等待包，全都发出去并删除

（对于arp回复，如果源地址是广播地址，那么这个数据包可能是错误的或者被篡改过，应该被丢弃）

TimeOut

发送arp请求时，如果一直没有回复，每隔一秒发一次，最多发五次，还没有回复就把相应的包全都删掉

在包和包的间隙，等待下一个包时进行timeout的检查

HandlePacket

检查以太网地址，如果既不是发给自己的包，也不是广播的包，那么不处理

（检查ethertype是否为arp和ipv4，还有vlan等不处理）

分arp和ipv4两种情况处理，其余情况不处理

四、实验结果

Test

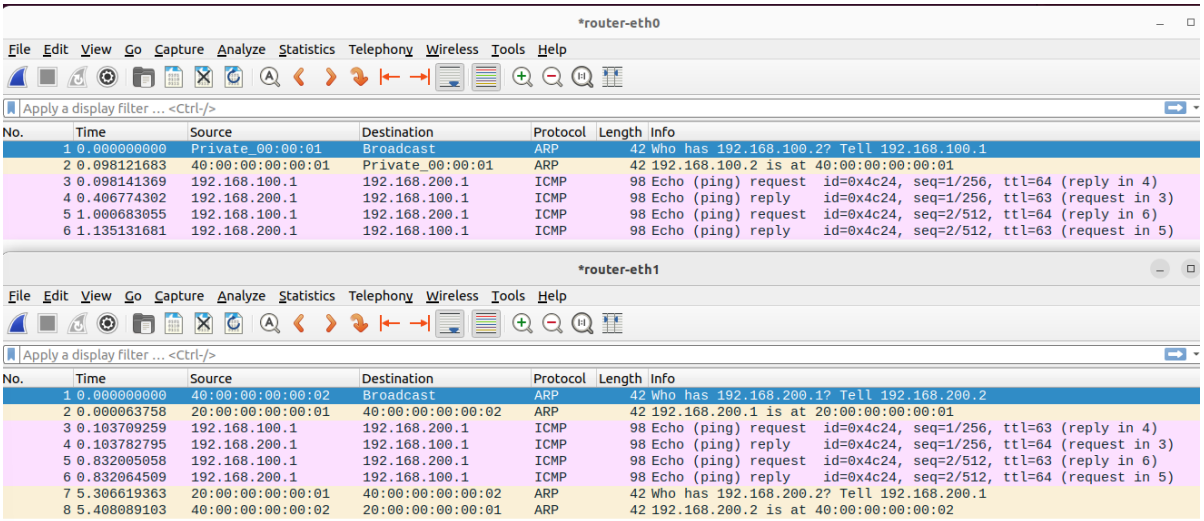
通过所有测试样例

```
1196Ping request from 31.0.5.1 should arrive on eth5
1197Ping request from 31.0.5.1 should arrive on eth5
1198Router should not do anything
1199Ping request from 31.0.6.1 should arrive on eth6
1200Ping request from 31.0.6.1 should arrive on eth6
1201Ping request from 31.0.6.1 should arrive on eth6
1202Ping request from 31.0.6.1 should arrive on eth6
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: V2FybWluZyB1cA==
1207Bonus: V2FybWVkIHVw
1208Bonus: V2h1dCBkJyB5YSBob3BlIHQnIGZpbmQgaGVyZT8=
1209Bonus: SGFsZndheQ==
1210Bonus: Tm90aGluJyBmb3IgeWEgdCcgZmluZCBoZXJlIQ==
1211Bonus: Q29uZ3JhdHMh


All tests passed!
```

Mininet

server1 ping -c2 server2



eth0 - 1：server1向router发送了一个arp请求，请求router的mac

eth0 - 2：router向server1发送arp回复，回复router的mac（这时router保存了server1的mac）

eth0 - 3：server1向server2发送ping request，经router转发

eth1 - 1：router向server2发送arp请求，请求server2的mac

eth1 - 2：server2向router发送arp回复，回复server2的mac（这时server2保存了router的mac）

eth1 - 3：router给server2转发来自server1的ping request

eth1 - 4：server2向server1发送ping reply，经router转发

eth0 - 4：router给server1转发来自server2的ping reply

eth0 - 5：server1又向server2发了一个arp request，经router转发

eth1 - 5：router给server2转发来自server1的第二个ping request

eth1 - 6：server2向server1发送ping reply，经router转发

eth0 - 6：router给server1转发第二个来自server2的ping reply

eth1 - 7：server2向router发arp请求，确定和router之间的连接正常

eth1 - 8：router回复server2的arp请求

五、核心代码

```
#!/usr/bin/env python3

'''
Basic IPv4 router (static routing) in Python.
'''

import time
import switchyard
from switchyard.lib.userlib import *
from switchyard import llnetbase

class ForwardingTableEntry:
    def __init__(self, dest, mask, gateway, interface):
```

```
        self.dest = dest
        self.mask = mask
        self.gateway = gateway
        self.interface = interface
        self.prefixnet = IPv4Network("{}/{}".format(dest, mask), strict=False)

    def __lt__(self, other):
        return self.prefixnet.prefixlen > other.prefixnet.prefixlen

class Router(object):
    def __init__(self, net: switchyard.llnetbase.LLNetBase):
        self.net = net
        self.interfaces = self.net.interfaces()
        self.ip_list = [interface.ipaddr for interface in self.interfaces]
        self.arp_table = {}
        self.forwarding_table = []
        self.build_forwarding_table()
        self.waiting_packet = {} # ip - packet_list
        self.waiting_ip = {} # ip - (time, retries)

    def build_forwarding_table(self):
        for interface in self.interfaces:
            self.forwarding_table.append(ForwardingTableEntry(interface.ipaddr,
interface.netmask, None, interface.name))
        with open("forwarding_table.txt") as f:
            for line in f:
                dest, mask, gateway, interface = line.strip().split()
                self.forwarding_table.append(ForwardingTableEntry(dest, mask,
gateway, interface))
        self.forwarding_table.sort()

    def get_forwarding_entry(self, ipaddr):
        for entry in self.forwarding_table:
            if ipaddr in entry.prefixnet: # match
                return entry
        return None

    def handle_ipv4_packet(self, recv):
        timestamp, ifaceName, packet = recv
        ipv4 = packet.get_header(IPv4)
        eth = packet.get_header(Ethernet)
        dst_ip = ipv4.dst
        if len(eth) + ipv4.total_length != packet.size():
            return
        if dst_ip in self.ip_list:
            return
        entry = self.get_forwarding_entry(dst_ip)
        if entry is None:
            return

        next_hop_ip = dst_ip if entry.gateway is None else
ip_address(entry.gateway)
        next_hop_mac = self.arp_table.get(next_hop_ip)

        if next_hop_mac is not None:
```

```
            packet[Ethernet].src =
self.net.interface_by_name(entry.interface).ethaddr
            packet[Ethernet].dst = next_hop_mac
            packet[IPv4].ttl -= 1
            self.net.send_packet(entry.interface, packet)

        else:
            if next_hop_ip not in self.waiting_ip.keys():
                self.waiting_ip[next_hop_ip] = (time.time(), 1)
                arp_request =
create_ip_arp_request(self.net.interface_by_name(entry.interface).ethaddr,self.ne
t.interface_by_name(entry.interface).ipaddr,next_hop_ip)
                self.net.send_packet(entry.interface, arp_request)
            if next_hop_ip not in self.waiting_packet.keys():
                self.waiting_packet[next_hop_ip] = []
            self.waiting_packet[next_hop_ip].append(packet)


    def handle_arp_packet(self, recv):
        timestamp, ifaceName, packet = recv
        arp = packet.get_header(Arp)
        eth = packet.get_header(Ethernet)
        src_ip = arp.senderprotoaddr
        src_mac = arp.senderhwaddr
        dst_ip = arp.targetprotoaddr
        if dst_ip not in self.ip_list:
            return

        if arp.operation == ArpOperation.Request:
            self.arp_table[src_ip] = src_mac
            arp_reply =
create_ip_arp_reply(self.net.interface_by_ipaddr(dst_ip).ethaddr, src_mac,
dst_ip, src_ip)
            self.net.send_packet(ifaceName, arp_reply)

        elif arp.operation == ArpOperation.Reply:
            if eth.src == 'ff:ff:ff:ff:ff:ff':
                return
            self.arp_table[src_ip] = src_mac
            if src_ip in self.waiting_ip.keys():
                for packet in self.waiting_packet[src_ip]:
                    packet[Ethernet].src =
self.net.interface_by_name(ifaceName).ethaddr
                    packet[Ethernet].dst = src_mac
                    packet[IPv4].ttl -= 1
                    self.net.send_packet(ifaceName, packet)
                del self.waiting_packet[src_ip]
                del self.waiting_ip[src_ip]

    def handle_timeout(self):
        for ip in list(self.waiting_ip.keys()):
            timestamp = self.waiting_ip[ip][0]
            retries = self.waiting_ip[ip][1]
            if time.time() - timestamp > 1:
                if retries >= 5:
                    del self.waiting_ip[ip]
```

```
                    del self.waiting_packet[ip]
                else:
                    self.waiting_ip[ip] = (time.time(), retries + 1)
                    arp_request = create_ip_arp_request(

 self.net.interface_by_name(self.get_forwarding_entry(ip).interface).ethaddr,

 self.net.interface_by_name(self.get_forwarding_entry(ip).interface).ipaddr,
                        ip
                    )
                    self.net.send_packet(self.get_forwarding_entry(ip).interface,
arp_request)


    def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
        timestamp, ifaceName, packet = recv
        arp = packet.get_header(Arp)
        eth = packet.get_header(Ethernet)
        ipv4 = packet.get_header(IPv4)

        if eth.dst != self.net.interface_by_name(ifaceName).ethaddr and eth.dst
!= 'ff:ff:ff:ff:ff:ff':
            return
        if eth.ethertype != EtherType.ARP and eth.ethertype != EtherType.IPv4:
            return

        if arp:
            self.handle_arp_packet(recv)
            return
        if ipv4:
            self.handle_ipv4_packet(recv)
            return


    def start(self):
        '''A running daemon of the router.
        Receive packets until the end of time.
        '''
        while True:
            try:
                recv = self.net.recv_packet(timeout=1.0)
            except NoPackets:
                self.handle_timeout()
                continue
            except Shutdown:
                break

            self.handle_packet(recv)

        self.stop()

    def stop(self):
        self.net.shutdown()


def main(net):
```

```
    '''
    Main entry point for router.  Just create Router
    object and get it going.
    '''
    router = Router(net)
    router.start()
```

六、总结与感想