计网实验四报告

211220149 屈子康

2024 年 11 月 13 日

目录

1　转发表中匹配 IP 地址

1.1　转发表的生成

通过调用获取的路由器接口列表和读取 forwarding_table.txt 文件两个来源构造转发表

Listing 1: Forwarding Table

```
class IPForwardingTable:
class TableLine:
    def __init__(self, prefix, mask, next_hop, interface):
        self.prefix = prefix
        self.mask = mask
        self.next_hop = next_hop
        self.interface = interface

def __init__(self, router):
    self.table = {}
    self.router = router

    for port in self.router.net.ports():
        netmask = port.netmask
        ipaddr = port.ipaddr
        prefix = str(ipaddress.IPv4Address(int(ipaddr) & int(
           netmask)))
        interface_name = port.name

        prefix_net = IPv4Network(prefix + '/' + str(netmask))
        self.table[prefix_net] = IPForwardingTable.TableLine(prefix
           , str(netmask), '0.0.0.0', str(interface_name))

    with open('forwarding_table.txt', 'r') as table_file:
        lines = table_file.readlines()
        for line in lines:
            elements = line.split()

            prefix = elements[0]
            mask = elements[1]
            next_hop = elements[2]
            interface = elements[3]

            prefix_net = IPv4Network(prefix + '/' + mask)

            self.table[prefix_net] = IPForwardingTable.TableLine(
               prefix, mask, next_hop, interface)
```

1.2 IP 地址的匹配

首先通过 matches = destaddr in prefixnet 的方式找到所有的匹配项，然后从这些匹配项中找到掩码最大值对应的项

Listing 2: match ip

```
def get_match_line(self, target_ip):
    match_lines = []
    for prefix_net in self.table.keys():
        if target_ip in prefix_net:
            match_lines.append(self.table[prefix_net])
    # nothing in match lines
    if not match_lines:
        return None
    # find the max mask
    else:
        max_mask = max(int(IPv4Address(line.mask)) for line in
           match_lines)
        for match_line in match_lines:
            if int(IPv4Address(match_line.mask)) == max_mask:
                return match_line
```

2 处理数据包转发和 ARP 请求生成

2.1 解析以太网头

先处理 packet 的以太网头来丢掉目标地址不是该端口并且不是广播的包：

Listing 3: deal with ethernet header

```
    # deal with ethernet_header
    # first header is not ethernet header
    ethernet_header = packet[0]
    del packet[0]
    if not isinstance(ethernet_header, Ethernet):
        log_info("first header is not ethernet header")
        return
    # packet not for this port
    if ethernet_header.dst not in [port.ethaddr for port in self.
       net.ports() if port.name == ifaceName] and str(
            ethernet_header.dst) != 'ff:ff:ff:ff:ff:ff':
        log_info("packet not for this router")
        return
```

2.2 IPV4 数据包

2.2.1 处理 IPV4 数据包

识别到合法 ipv4 包后，查看 ipv4 包的下一跳转 ip 是否记录在 arp cache 上，如果是直接转发；如果不是，需要将该包放入等待列表中，并且由等待列表定时发送目标 ip 的 arp request 包

Listing 4: Basic Switch

```
def handle_ipv4_packet(self, receive_port_name, packet):
    log_info("!!###########(receive ipv4 packet) ")
    ipv4_header = packet.get_header(IPv4)
    # packet for router itself
    for port in self.net.ports():
        if str(port.ipaddr) == str(ipv4_header.dst):
            log_info("this packet is for router")
            return

    if ipv4_header is not None:
        ipv4_header.ttl -= 1
        log_info(f"ttl = {ipv4_header.ttl}")

    # if dst is in this router, ignore it
    if ipv4_header.dst in [port.ipaddr for port in self.net.ports()
       ]:
        return

    # pdb.set_trace()

    forward_table_match_line = self.forward_table.get_match_line(
       ipv4_header.dst)
    log_info(f"dest ip = {ipv4_header.dst}")
    if forward_table_match_line is not None:
        next_hop_ip = forward_table_match_line.next_hop
        log_info(f"next hop = {next_hop_ip}")
        if str(next_hop_ip) == '0.0.0.0':
            next_hop_ip = ipv4_header.dst

        interface_name = forward_table_match_line.interface
        log_info(f"interface's name = {interface_name}")

        # need to send arp request packet
        if not self.arp_table.is_ip_in_arp_table(next_hop_ip):
            for port in self.net.ports():
                # log_info(f"port's name = {port.name}")
                if port.name == interface_name:
                    self.ipv4_packet_waiting_list.add_packet(packet
                       , port, str(next_hop_ip))
```

```
                        break
            #
            else:
                log_info("ip-mac in arp cache")
                target_mac = self.arp_table.is_ip_in_arp_table(
                   next_hop_ip)
                for port in self.net.ports():
                    if port.name == interface_name:
                        ethernet_header = Ethernet(src=port.ethaddr,
                           dst=target_mac)
                        log_info(f"packet = {packet}")
                        log_info(f"ethernet_header = {ethernet_header}"
                           )
                        packet.prepend_header(ethernet_header)

                        log_info("***********************************(
                           send Ipv4 packet)")
                        log_info(f"new packet = {packet}")

                        self.router_send_packet(port, packet)
```

2.2.2 waiting list 的实现

以 ip 作为处理主体，每隔 1 秒向记录中的 ip 发送 arp request，最多发送 5 次，ip 下记录着所有目标地址是该 ip 的 ipv4 数据包，如果相应的 arp reply 到来，waiting list 负责将该 ip 下所有的 ipv4 数据包发送出去

Listing 5: Basic Switch

```
class Ipv4PacketWaitingList:
    class ArpRequestMessage:
        def __init__(self, arp_request_packet, interface,
           last_request_time, request_times):
            self.arp_request_packet = arp_request_packet
            self.interface = interface
            self.last_request_time = last_request_time
            self.request_times = request_times
            self.last_refresh_time = time.time()

    def __init__(self, interval_time, max_request_times, router):
        self.interval_time = interval_time
        self.max_request_times = max_request_times
        self.waiting_list = []
        self.router = router
        self.ip2waiting_list = {}
        self.ip2arp_request = {}
        self.total_refresh_time = 0
```

```
def print_waiting_list(self):
    log_info("----------------------------------------------------(
       waiting list)")
    for key in self.ip2waiting_list.keys():
        log_info(f"ip:{key}, num:{len(self.ip2waiting_list[key])},
           times: {self.ip2arp_request[key].request_times}")
    log_info("----------------------------------------------------"
       )

# target ip must in waiting list, request times must < 5
def __waiting_list_send_arp_request(self, target_ip):
    self.router.router_send_packet(self.ip2arp_request[target_ip].
       interface,
                                   self.ip2arp_request[target_ip].
                                      arp_request_packet)
    self.ip2arp_request[target_ip].request_times += 1

    log_info("**************************(send arp request)")
    log_info(
        f"send arp request packet from {self.ip2arp_request[
           target_ip].interface.name} to {target_ip}, "
        f"already send arp request {self.ip2arp_request[target_ip].
           request_times} times, "
        f"time offset = {time.time() - self.ip2arp_request[
           target_ip].last_request_time}, "
        f"request times = {self.ip2arp_request[target_ip].
           request_times}")
    self.ip2arp_request[target_ip].last_request_time = time.time()

def send_arp_request_loop(self, next_hop_ip):
    # end loop situation 1
    if next_hop_ip not in self.ip2waiting_list.keys():
        return
    # end loop situation 2
    if self.ip2arp_request[next_hop_ip].request_times >= 5:
        self.del_packet(next_hop_ip)
    # send packet and prepare next loop
    else:
        # send arp request to next_hop_ip
        self.__waiting_list_send_arp_request(next_hop_ip)
        # next loop
        Timer(self.interval_time, Ipv4PacketWaitingList.
           send_arp_request_loop, args=(self, next_hop_ip)).start()

def add_packet(self, packet, interface, next_hop_ip):
        log_info(f"Waiting list: add dst({next_hop_ip}) on {interface}"
           )
        if next_hop_ip not in self.ip2waiting_list.keys():
            self.ip2waiting_list[next_hop_ip] = []
            self.ip2waiting_list[next_hop_ip].append(packet)
            # make arp request packet
            arp_request_packet = create_ip_arp_request(interface.
               ethaddr, interface.ipaddr, next_hop_ip)
            self.ip2arp_request[next_hop_ip] = self.ArpRequestMessage(
               arp_request_packet, interface,
                                                                       time
                                                                          .
                                                                          time
                                                                          ()
                                                                          ,

                                                                          0)




            # solution1: send arp request loop
            # self.send_arp_request_loop(next_hop_ip)
            # solution2: first send arp request then start refresh
               before each loop

            if self.ip2arp_request[next_hop_ip].request_times < 5:
                self.__waiting_list_send_arp_request(next_hop_ip)

            self.refresh()

        else:
            self.ip2waiting_list[next_hop_ip].append(packet)
        self.print_waiting_list()

    def del_packet(self, next_hop_ip):
        log_info(f"delete ip = {next_hop_ip}")
        del self.ip2arp_request[next_hop_ip]
        del self.ip2waiting_list[next_hop_ip]
        self.print_waiting_list()

    def is_ip_in_waiting_list(self, ip):
        return ip in self.ip2waiting_list.keys()

    def refresh(self):
        self.total_refresh_time += 1
        for key in self.ip2waiting_list.copy():
            if time.time() - self.ip2arp_request[key].last_refresh_time
```

```
                >= 0.2:
                log_info(
                    f"Refresh: offset time = {time.time() - self.
                       ip2arp_request[key].last_request_time}, ip = {
                       key},  "
                    f"request times = {self.ip2arp_request[key].
                       request_times}")
                self.ip2arp_request[key].last_refresh_time = time.time
                   ()
            if time.time() - self.ip2arp_request[key].last_request_time
                >= self.interval_time:
                # send new arp packet
                if self.ip2arp_request[key].request_times < 5:
                    log_info("*************************(send arp
                       request)")
                    log_info(
                        f"send arp request on {self.ip2arp_request[key
                           ].interface.name} to {key}, offset time = {
                           time.time() - self.ip2arp_request[key].
                           last_request_time}")
                    self.router.router_send_packet(self.ip2arp_request[
                       key].interface,
                                                   self.ip2arp_request[
                                                      key].
                                                      arp_request_packet
                                                      )
                    self.ip2arp_request[key].last_request_time = time.
                       time()
                    self.ip2arp_request[key].request_times += 1
                # delete item
                else:
                    log_info(f"delete ip = {key}")
                    del self.ip2arp_request[key]
                    del self.ip2waiting_list[key]

    def get_arp_reply(self, target_ip, target_mac, interface):
        # log_info(f"{len(self.waiting_list)}")
        log_info(f"get_arp_reply:target_ip={target_ip}, target_mac={
           target_mac}, interface={interface.name}")

        if self.is_ip_in_waiting_list(target_ip):
            log_info(f"offset time = {time.time() - self.ip2arp_request
               [target_ip].last_request_time}")
            log_info(f"arp reply for {target_ip} arrived!")
            if time.time() - self.ip2arp_request[target_ip].
               last_request_time > self.interval_time:
                log_info("arp reply packet out of waiting received time
                   , do not receive")
                return
            ethernet_header = Ethernet(src=self.ip2arp_request[
               target_ip].interface.ethaddr, dst=target_mac)
            for packet in self.ip2waiting_list[target_ip]:
                log_info(f"packet = {packet}")
                log_info(f"ethernet_header = {ethernet_header}")
                packet.prepend_header(ethernet_header)

                log_info("**********************************(send Ipv4
                    packet)")
                log_info(f"new packet = {packet}")

                self.router.router_send_packet(interface, packet)
            # done this ip
            del self.ip2waiting_list[target_ip]
            del self.ip2arp_request[target_ip]
```

2.3 arp 数据包

接收到合法的 arp 数据包后，根据 arp_header.operation 来确定是请求还是恢复数据包，从而交给不同的处理函数

2.3.1 处理 arp request 数据包

如果该 arp request 的目标 ip 是路由器的某个端口 ip，则发送 arp reply

Listing 6: Basic Switch

```
    def handle_arp_request_packet(self, receive_port_name, packet):
        log_info("!!###########(receive arp reply packet) ")
        arp_header = packet.get_header(Arp)

        sender_ip = arp_header.senderprotoaddr
        sender_mac = arp_header.senderhwaddr
        target_ip = arp_header.targetprotoaddr
        target_mac = Router.find_mac_by_ip(self, target_ip)
        log_info(f"sender ip={sender_ip}, sender_mac={sender_mac},
           target ip={target_ip}, target mac={target_mac}")

        # check arp header
        if target_ip not in [port.ipaddr for port in self.net.ports()]:
            log_info("arp request packet's target ip not in this router
               ")
            return
        # add ip-mac into arp cache
        self.arp_table.add_element(sender_ip, sender_mac)

        # need to replay
        if target_mac is not None:
            log_info("need to reply")
            # create_ip_arp_reply
            reply_packet = create_ip_arp_reply(target_mac, sender_mac,
               target_ip, sender_ip)
            log_debug(f"sender ip = {sender_ip}, sender mac = {
               sender_mac}")

            for interface in self.net.interfaces():
                if interface.name == receive_port_name:
                    log_info(f"send arp reply packet to {sender_ip}
                       from {interface.name}")
                    self.router_send_packet(interface, reply_packet)
```

2.3.2 处理 arp reply 数据包

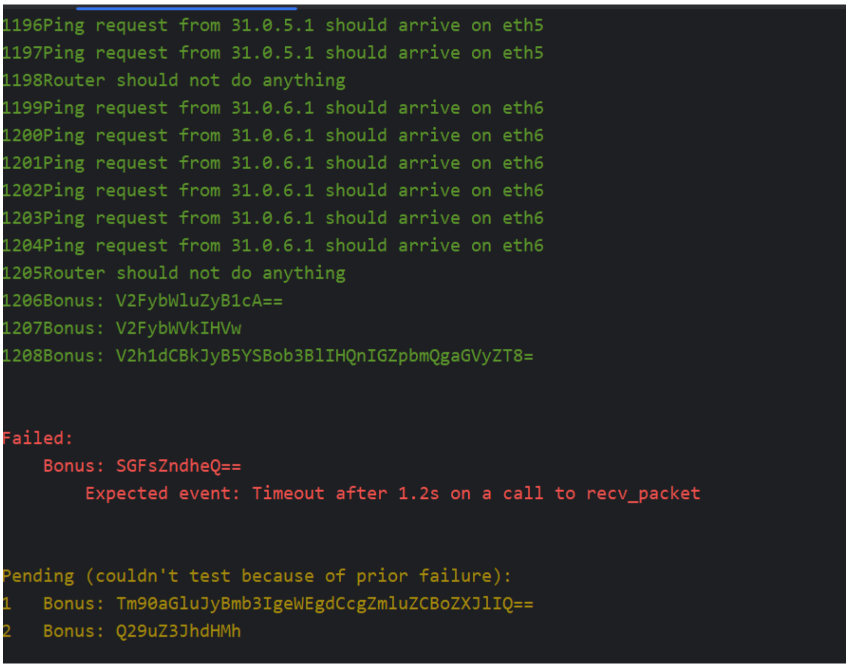
对收到的 arp reply 再进行一次检查，如果发现发送者的地址是广播地址，认为不合法，否则，将该 arp reply 的回复丢到 waiting list 中查看有无匹配项

Listing 7: Basic Switch

```
def handle_arp_reply_packet(self, receive_port_name, packet):
    log_info("!!###########(receive arp reply packet) ")

    arp_header = packet.get_header(Arp)

    sender_ip = arp_header.senderprotoaddr
    sender_mac = arp_header.senderhwaddr
    target_ip = arp_header.targetprotoaddr
    target_mac = arp_header.targethwaddr
    log_info(f"sender ip={sender_ip}, sender_mac={sender_mac},
       target ip={target_ip}, target mac={target_mac}")

    if str(sender_mac) == 'ff:ff:ff:ff:ff:ff':
        log_info("illegal arp reply packet")
        return

    # check arp header
    if target_ip not in [port.ipaddr for port in self.net.ports()]
       or target_mac not in [port.ethaddr for port in self.net.
       ports()]:
        log_info("arp reply packet not for this router")
        return
```

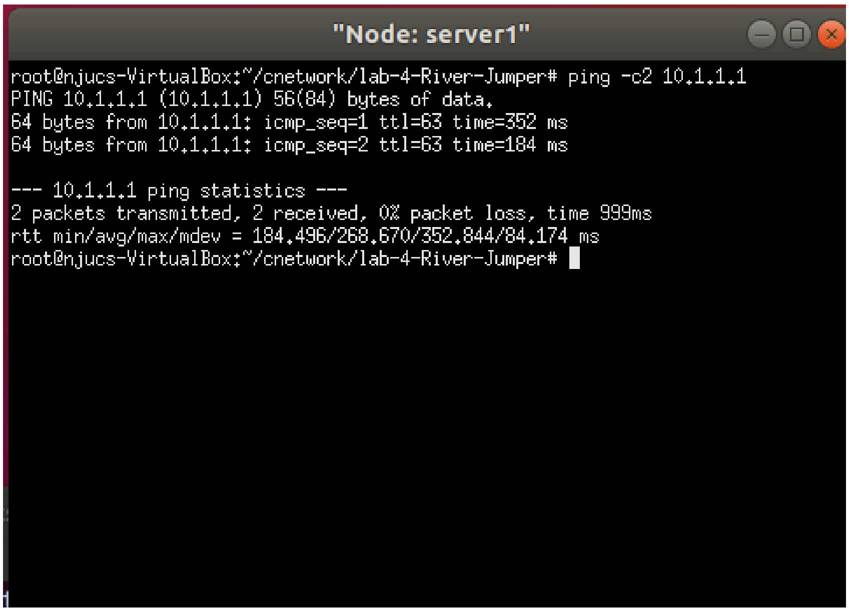
```
# add ip-mac into arp cache

self.arp_table.add_element(sender_ip, sender_mac)

# is this arp reply packet for this router?
for interface in self.net.interfaces():
    if interface.ethaddr == target_mac:
        self.ipv4_packet_waiting_list.get_arp_reply(str(
           sender_ip), sender_mac, interface)
```

3 样例测试结果

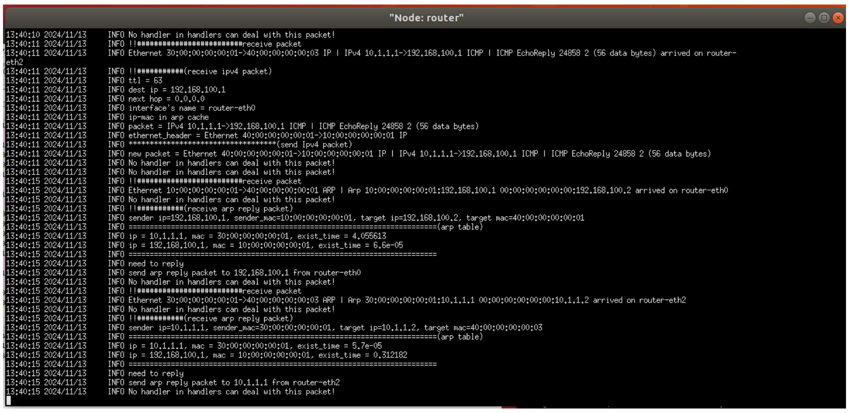


```
1196Ping request from 31.0.5.1 should arrive on eth5
1197Ping request from 31.0.5.1 should arrive on eth5
1198Router should not do anything
1199Ping request from 31.0.6.1 should arrive on eth6
1200Ping request from 31.0.6.1 should arrive on eth6
1201Ping request from 31.0.6.1 should arrive on eth6
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=


Failed:
    Bonus: SGFsZndheQ==
        Expected event: Timeout after 1.2s on a call to recv_packet


Pending (couldn't test because of prior failure):
1   Bonus: Tm90aGluJyBmb3IgeWEgdCcgZmluZCBoZXJlIQ==
2   Bonus: Q29uZ3JhdHMh
```

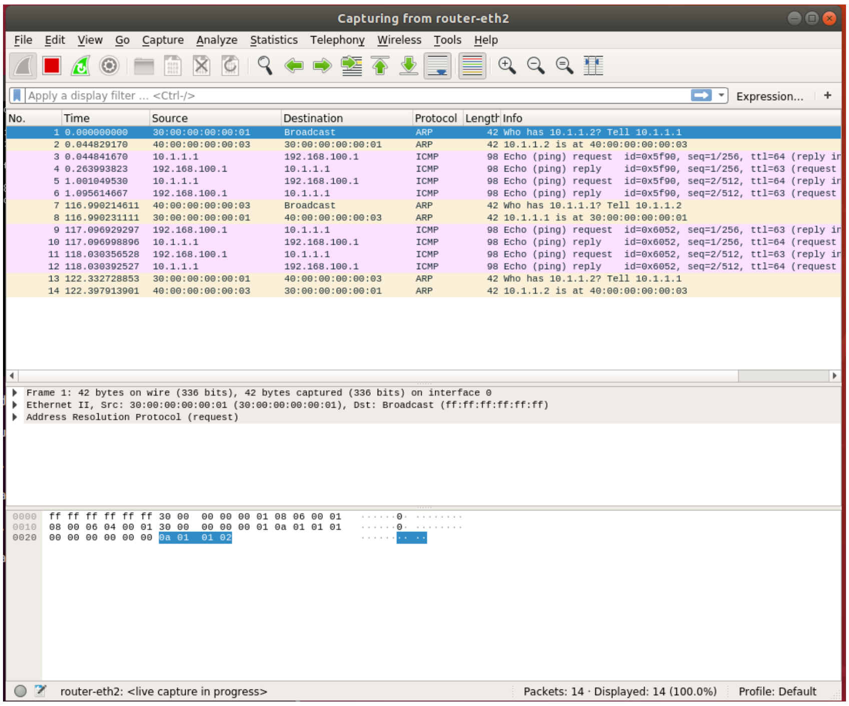
4 wireshark 中的结果



```
root@njucs-VirtualBox:~/cnetwork/lab-4-River-Jumper# ping -c2 10.1.1.1
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data.
64 bytes from 10.1.1.1: icmp_seq=1 ttl=63 time=352 ms
64 bytes from 10.1.1.1: icmp_seq=2 ttl=63 time=184 ms

--- 10.1.1.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 184.496/268.670/352.844/84.174 ms
root@njucs-VirtualBox:~/cnetwork/lab-4-River-Jumper# 
```



```
13:40:10 2024/11/13     INFO No handler in handlers can deal with this packet!
13:40:11 2024/11/13     INFO !!#########################receive packet
13:40:11 2024/11/13     INFO Ethernet 30:00:00:00:00:01->40:00:00:00:00:03 IP | IPv4 10.1.1.1->192.168.100.1 ICMP | ICMP EchoReply 24858 2 (56 data bytes) arrived on router-
eth2
13:40:11 2024/11/13     INFO !!###########(receive ipv4 packet)
13:40:11 2024/11/13     INFO ttl = 63
13:40:11 2024/11/13     INFO dest ip = 192.168.100.1
13:40:11 2024/11/13     INFO next hop = 0.0.0.0
13:40:11 2024/11/13     INFO interface's name = router-eth0
13:40:11 2024/11/13     INFO ip-mac in arp cache
13:40:11 2024/11/13     INFO packet = IPv4 10.1.1.1->192.168.100.1 ICMP | ICMP EchoReply 24858 2 (56 data bytes)
13:40:11 2024/11/13     INFO ethernet_header = Ethernet 40:00:00:00:00:01->10:00:00:00:00:01 IP
13:40:11 2024/11/13     INFO **********************************(send Ipv4 packet)
13:40:11 2024/11/13     INFO new packet = Ethernet 40:00:00:00:00:01->10:00:00:00:00:01 IP | IPv4 10.1.1.1->192.168.100.1 ICMP | ICMP EchoReply 24858 2 (56 data bytes)
13:40:11 2024/11/13     INFO No handler in handlers can deal with this packet!
13:40:11 2024/11/13     INFO No handler in handlers can deal with this packet!
13:40:15 2024/11/13     INFO !!#########################receive packet
13:40:15 2024/11/13     INFO Ethernet 10:00:00:00:00:01->40:00:00:00:00:01 ARP | Arp 10:00:00:00:00:01:192.168.100.1 00:00:00:00:00:00:192.168.100.2 arrived on router-eth0
13:40:15 2024/11/13     INFO No handler in handlers can deal with this packet!
13:40:15 2024/11/13     INFO !!###########(receive arp reply packet)
13:40:15 2024/11/13     INFO sender ip=192.168.100.1, sender_mac=10:00:00:00:00:01, target ip=192.168.100.2, target mac=40:00:00:00:00:01
13:40:15 2024/11/13     INFO ========================================================================(arp table)
13:40:15 2024/11/13     INFO ip = 10.1.1.1, mac = 30:00:00:00:00:01, exist_time = 4.055613
13:40:15 2024/11/13     INFO ip = 192.168.100.1, mac = 10:00:00:00:00:01, exist_time = 6.6e-05
13:40:15 2024/11/13     INFO ========================================================================
13:40:15 2024/11/13     INFO need to reply
13:40:15 2024/11/13     INFO send arp reply packet to 192.168.100.1 from router-eth0
13:40:15 2024/11/13     INFO No handler in handlers can deal with this packet!
13:40:15 2024/11/13     INFO !!#########################receive packet
13:40:15 2024/11/13     INFO Ethernet 30:00:00:00:00:01->40:00:00:00:00:03 ARP | Arp 30:00:00:00:00:01:10.1.1.1 00:00:00:00:00:00:10.1.1.2 arrived on router-eth2
13:40:15 2024/11/13     INFO No handler in handlers can deal with this packet!
13:40:15 2024/11/13     INFO !!###########(receive arp reply packet)
13:40:15 2024/11/13     INFO sender ip=10.1.1.1, sender_mac=30:00:00:00:00:01, target ip=10.1.1.2, target mac=40:00:00:00:00:03
13:40:15 2024/11/13     INFO ========================================================================(arp table)
13:40:15 2024/11/13     INFO ip = 10.1.1.1, mac = 30:00:00:00:00:01, exist_time = 5.7e-05
13:40:15 2024/11/13     INFO ip = 192.168.100.1, mac = 10:00:00:00:00:01, exist_time = 0.312182
13:40:15 2024/11/13     INFO ========================================================================
13:40:15 2024/11/13     INFO need to reply
13:40:15 2024/11/13     INFO send arp reply packet to 10.1.1.1 from router-eth2
13:40:15 2024/11/13     INFO No handler in handlers can deal with this packet!
```



Capturing from router-eth2
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Expression…
+

```
No.   Time          Source              Destination         Protocol  Length Info
    1 0.000000000   30:00:00:00:00:01   Broadcast           ARP       42 Who has 10.1.1.2? Tell 10.1.1.1
    2 0.044829170   40:00:00:00:00:03   30:00:00:00:00:01   ARP       42 10.1.1.2 is at 40:00:00:00:00:03
    3 0.044841670   10.1.1.1            192.168.100.1       ICMP      98 Echo (ping) request  id=0x5f90, seq=1/256, ttl=64 (reply i
    4 0.263993823   192.168.100.1       10.1.1.1            ICMP      98 Echo (ping) reply    id=0x5f90, seq=1/256, ttl=63 (request
    5 1.001049530   10.1.1.1            192.168.100.1       ICMP      98 Echo (ping) request  id=0x5f90, seq=2/512, ttl=64 (reply i
    6 1.095614667   192.168.100.1       10.1.1.1            ICMP      98 Echo (ping) reply    id=0x5f90, seq=2/512, ttl=63 (request
    7 116.990214611 40:00:00:00:00:03   Broadcast           ARP       42 Who has 10.1.1.1? Tell 10.1.1.2
    8 116.990231111 30:00:00:00:00:01   40:00:00:00:00:03   ARP       42 10.1.1.1 is at 30:00:00:00:00:01
    9 117.096929297 192.168.100.1       10.1.1.1            ICMP      98 Echo (ping) request  id=0x6052, seq=1/256, ttl=63 (reply i
   10 117.096998896 10.1.1.1            192.168.100.1       ICMP      98 Echo (ping) reply    id=0x6052, seq=1/256, ttl=64 (request
   11 118.030356528 192.168.100.1       10.1.1.1            ICMP      98 Echo (ping) request  id=0x6052, seq=2/512, ttl=63 (reply i
   12 118.030392527 10.1.1.1            192.168.100.1       ICMP      98 Echo (ping) reply    id=0x6052, seq=2/512, ttl=64 (request
   13 122.332728853 30:00:00:00:00:01   40:00:00:00:00:03   ARP       42 Who has 10.1.1.2? Tell 10.1.1.1
   14 122.397913901 40:00:00:00:00:03   30:00:00:00:00:01   ARP       42 10.1.1.2 is at 40:00:00:00:00:03
```

```
▸ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▸ Ethernet II, Src: 30:00:00:00:00:01 (30:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Address Resolution Protocol (request)
```

```
0000  ff ff ff ff ff ff 30 00  00 00 00 01 08 06 00 01   ······0· ········
0010  08 00 06 04 00 01 30 00  00 00 00 01 0a 01 01 01   ······0· ········
0020  00 00 00 00 00 00 0a 01  01 02                     ········ ··
```

```
router-eth2: <live capture in progress>        Packets: 14 · Displayed: 14 (100.0%)        Profile: Default
```