南京大学本科生实验报告

课程名称：计算机网络　　任课教师：田臣/李文中　　助教：

学院	计算机科学与技术系	专业（方向）	计算机科学与技术
学号	202220013	姓名	徐简
Email	161200063@smail.nju.edu.cn	开始/完成日期	2021.4.8-2021.4.21

1. 实验名称

Lab 2: Respond to ARP

2. 实验目的

1. Handle ARP Requests
2. Cached ARP Table
3. Ultimate goal - an IPv4 router.

3. 实验内容、代码与结果

Step 1: Initialize required tables

Task 3 is just about adding a table feature to Task 2, so I think it is better to combine them into one part.

First, use `interfaces` method to initialize the Router class. Define an empty table `self.arp_table` to store a mapping in the Router between destination IP addresses and Ethernet MAC addresses.

```
def __init__(self, net: switchyard.llnetbase.LLNetBase):
        self.net = net
        self.interfaces=net.interfaces()
        self.ip_list=[intf.ipaddr for intf in self.interfaces]
        self.mac_list=[intf.ethaddr for intf in self.interfaces]
        self.arp_table={}
        #key ip, value mac
        # other initialization stuff here
```

Step 2: Handle ARP requests via cached ARP table

The handle logic after receiving a packet is described in the code annotations.

```
def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
        timestamp, ifaceName, packet = recv
        # TODO: your logic here
        log_debug("Got a packet:{}".format(str(packet)))
        log_info("Got a packet:{}".format(str(packet)))
        #the handle_packet function is called when the router receives a packet
        arp=packet.get_header(Arp)
        # get the arp header
        if arp is None:
            log_info("Not an arp packet") #get nothing from the arp header
        else:
            log_info("operation kind {}".format(str(arp.operation)))
            self.arp_table[arp.senderprotoaddr]=arp.senderhwaddr
            # store the source ip and mac
            if arp.operation==1:# API shows that it is an arp request
                log_info("arp requests")
                index =-1
                for i in range(len(self.ip_list)):
                    if self.ip_list[i]==arp.targetprotoaddr:
                        index =i
                        break
                # iterate the ip_list to match the target ip
                if index!= -1:# successfully match, need a reply
                    log_info("match packet")

 answer=create_ip_arp_reply(self.mac_list[index],arp.senderhwaddr,self.ip_list[
index],arp.senderprotoaddr)
                    self.net.send_packet(ifaceName,answer)#send back
                    log_info("send arp reply:{}".format(str(answer)))
            elif arp.operation==2:#API shows that it is an arp reply
                log_info("receive an arp reply")
                self.arp_table[arp.targetprotoaddr]=arp.targethwaddr
                # store the dest ip and mac in the table
            else:
                log_info("receive unknown arp")
        log_info("Table shown as follows:")
        #print the table every round
        for k,v in self.arp_table.items():
            print(k,"\t",v)
```

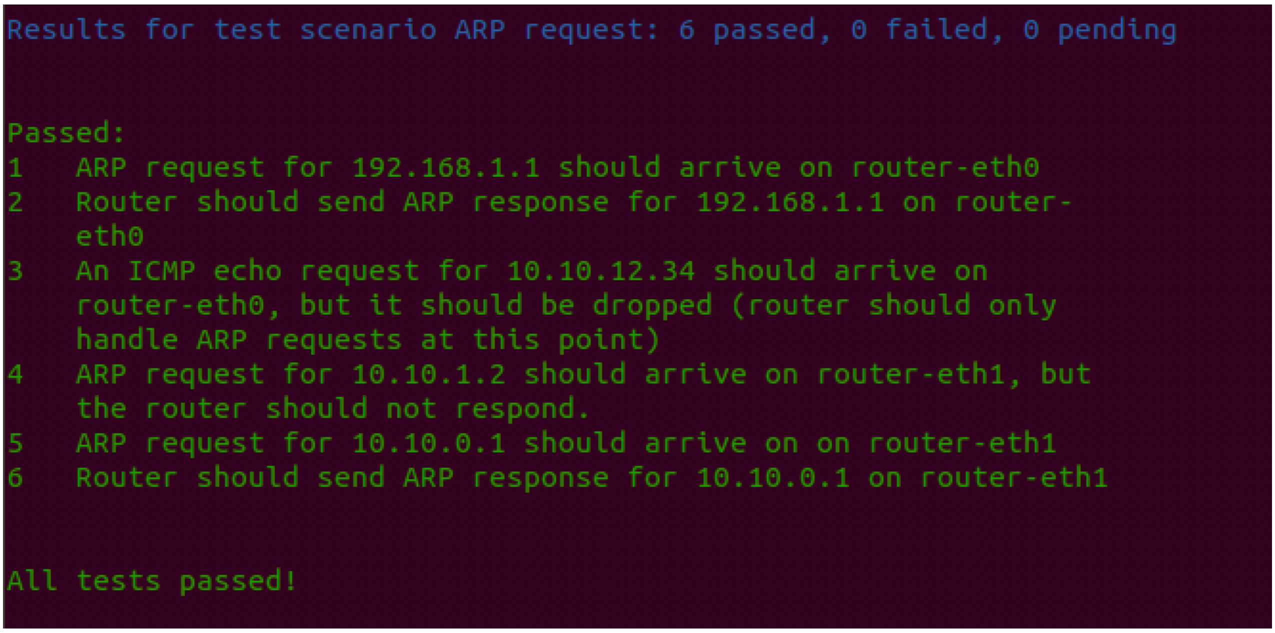
Step 3: Test

`$ swyard -t testcases/myrouter1_testscenario.srpy myrouter.py` and results are shown below.

```
17:07:09 2021/04/19     INFO Starting test scenario testcases/myrouter1_testscenario.srpy
17:07:09 2021/04/19     INFO Got a packet:Ethernet 30:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 30:00:00:00:00:01:192.168.1.100 ff:ff:ff:ff:ff:ff:192.168.1.1
17:07:09 2021/04/19     INFO operation kind ArpOperation.Request
17:07:09 2021/04/19     INFO arp requests
17:07:09 2021/04/19     INFO match packet
17:07:09 2021/04/19     INFO send arp reply:Ethernet 10:00:00:00:00:01->30:00:00:00:00:01 ARP | Arp 10:00:00:00:00:01:192.168.1.1 30:00:00:00:00:01:192.168.1.100

//received a packet and found out it was an arp request.
//sent a reply and update the arp_table

17:07:09 2021/04/19     INFO Table shown as follows:
192.168.1.100    30:00:00:00:00:01
17:07:09 2021/04/19     INFO Got a packet:Ethernet ab:cd:ef:00:00:01->10:00:00:00:00:01 IP | IPv4 192.168.1.242->10.10.12.34 ICMP | ICMP EchoRequest 0 42 (13 data bytes)
17:07:09 2021/04/19     INFO Not an arp packet
17:07:09 2021/04/19     INFO Table shown as follows:
192.168.1.100    30:00:00:00:00:01

//received a ICMP EchoRequest, not an arp packet, so drop the packet

17:07:09 2021/04/19     INFO Got a packet:Ethernet 60:00:de:ad:be:ef->ff:ff:ff:ff:ff:ff ARP | Arp 60:00:de:ad:be:ef:10.10.1.1 ff:ff:ff:ff:ff:ff:10.10.1.2
17:07:09 2021/04/19     INFO operation kind ArpOperation.Request
17:07:09 2021/04/19     INFO arp requests
17:07:09 2021/04/19     INFO Table shown as follows:
192.168.1.100    30:00:00:00:00:01
10.10.1.1    60:00:de:ad:be:ef

//no match, but update the arp_table

17:07:09 2021/04/19     INFO Got a packet:Ethernet 70:00:ca:fe:c0:de->ff:ff:ff:ff:ff:ff ARP | Arp 70:00:ca:fe:c0:de:10.10.5.5 ff:ff:ff:ff:ff:ff:10.10.0.1
17:07:09 2021/04/19     INFO operation kind ArpOperation.Request
17:07:09 2021/04/19     INFO arp requests
17:07:09 2021/04/19     INFO match packet
```

```
17:07:09 2021/04/19     INFO send arp reply:Ethernet 10:00:00:00:00:02->70:00:ca:fe:c0:de ARP | Arp 10:00:00:00:00:02:10.10.0.1 70:00:ca:fe:c0:de:10.10.5.5
17:07:09 2021/04/19     INFO Table shown as follows:
192.168.1.100    30:00:00:00:00:01
10.10.1.1    60:00:de:ad:be:ef
10.10.5.5    70:00:ca:fe:c0:de

//received an arp request, matched and sent a arp reply
//updated the arp_table


Results for test scenario ARP request: 6 passed, 0 failed, 0 pending


Passed:
1    ARP request for 192.168.1.1 should arrive on router-eth0
2    Router should send ARP response for 192.168.1.1 on router-
     eth0
3    An ICMP echo request for 10.10.12.34 should arrive on
     router-eth0, but it should be dropped (router should only
     handle ARP requests at this point)
4    ARP request for 10.10.1.2 should arrive on router-eth1, but
     the router should not respond.
5    ARP request for 10.10.0.1 should arrive on on router-eth1
6    Router should send ARP response for 10.10.0.1 on router-eth1


All tests passed!
```

Analysis of the result is marked by `//` annotations.



```
Results for test scenario ARP request: 6 passed, 0 failed, 0 pending


Passed:
1   ARP request for 192.168.1.1 should arrive on router-eth0
2   Router should send ARP response for 192.168.1.1 on router-
    eth0
3   An ICMP echo request for 10.10.12.34 should arrive on
    router-eth0, but it should be dropped (router should only
    handle ARP requests at this point)
4   ARP request for 10.10.1.2 should arrive on router-eth1, but
    the router should not respond.
5   ARP request for 10.10.0.1 should arrive on on router-eth1
6   Router should send ARP response for 10.10.0.1 on router-eth1


All tests passed!
```

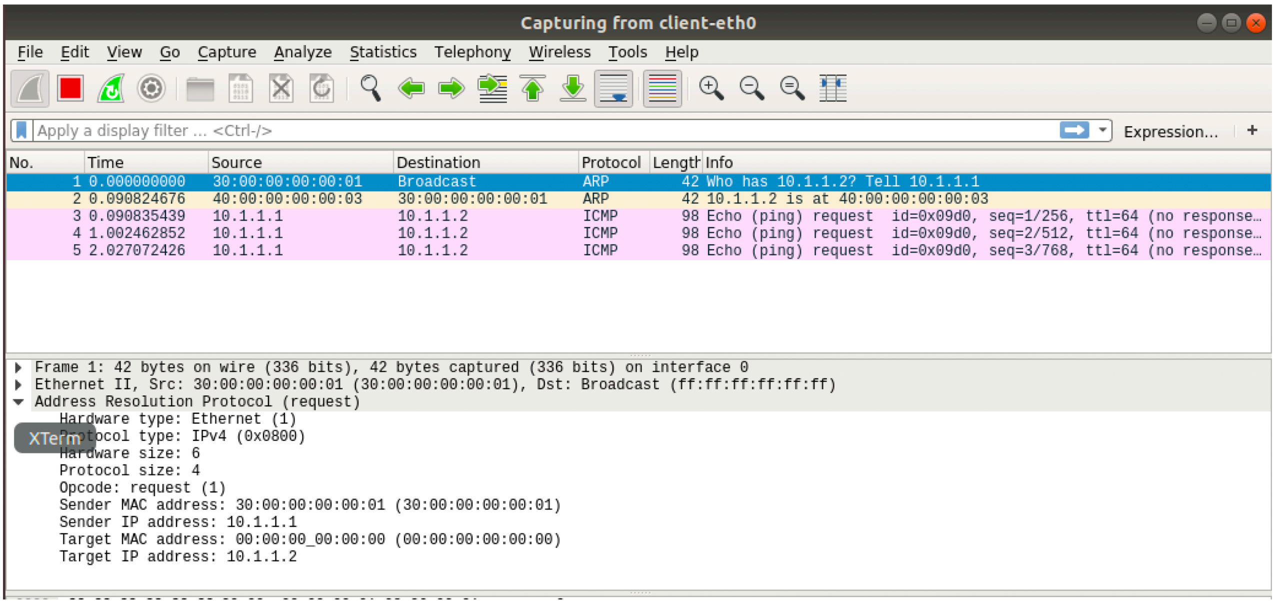
Step 4: Deploying

- `sudo python start_mininet.py`
- `client# ping -c3 10.1.1.2`

As you can see in the Wireshark capture window, the router initially received an ARP request for its own IP address and sent back an ARP request correctly. Then it received an ICMP echo request and nothing else happened.

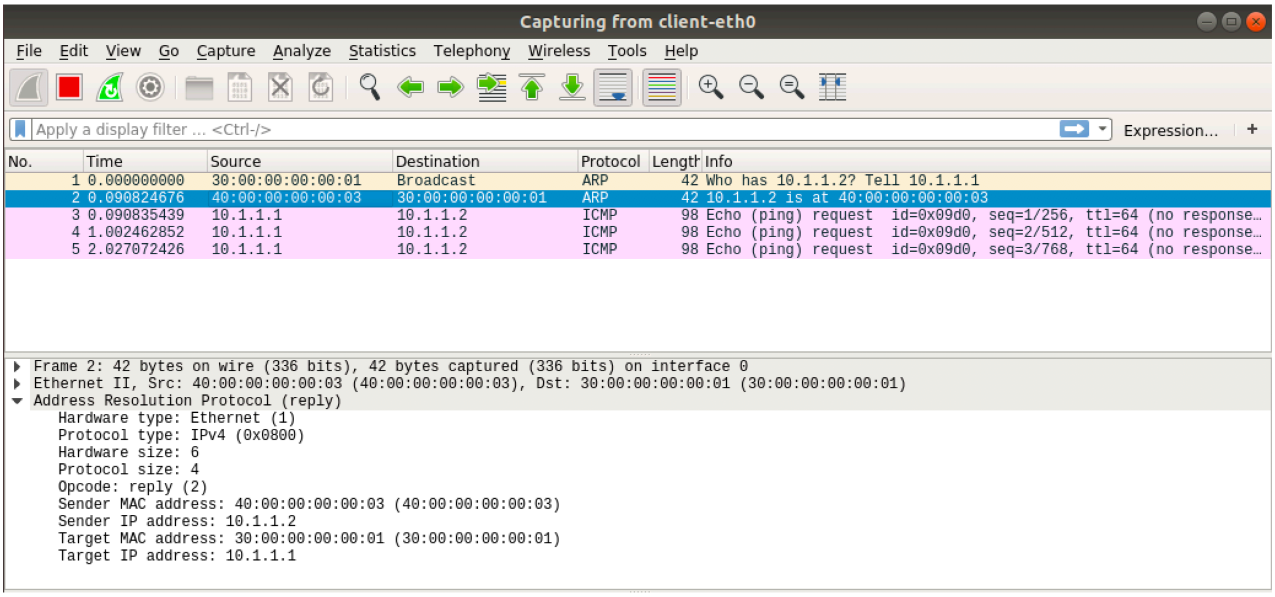
Click the ARP request packet the first line in the capture window, and the "target MAC address" is currently all zeroes, since this is the address being requested.

Click the ARP response packet and all the addresses in the ARP header are now filled in, as expected.



```
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 0.090824676  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 0.090835439  10.1.1.1            10.1.1.2            ICMP        98 Echo (ping) request  id=0x09d0, seq=1/256, ttl=64 (no response…
    4 1.002462852  10.1.1.1            10.1.1.2            ICMP        98 Echo (ping) request  id=0x09d0, seq=2/512, ttl=64 (no response…
    5 2.027072426  10.1.1.1            10.1.1.2            ICMP        98 Echo (ping) request  id=0x09d0, seq=3/768, ttl=64 (no response…
```

```
▶ 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)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: 30:00:00:00:00:01 (30:00:00:00:00:01)
    Sender IP address: 10.1.1.1
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 10.1.1.2
```

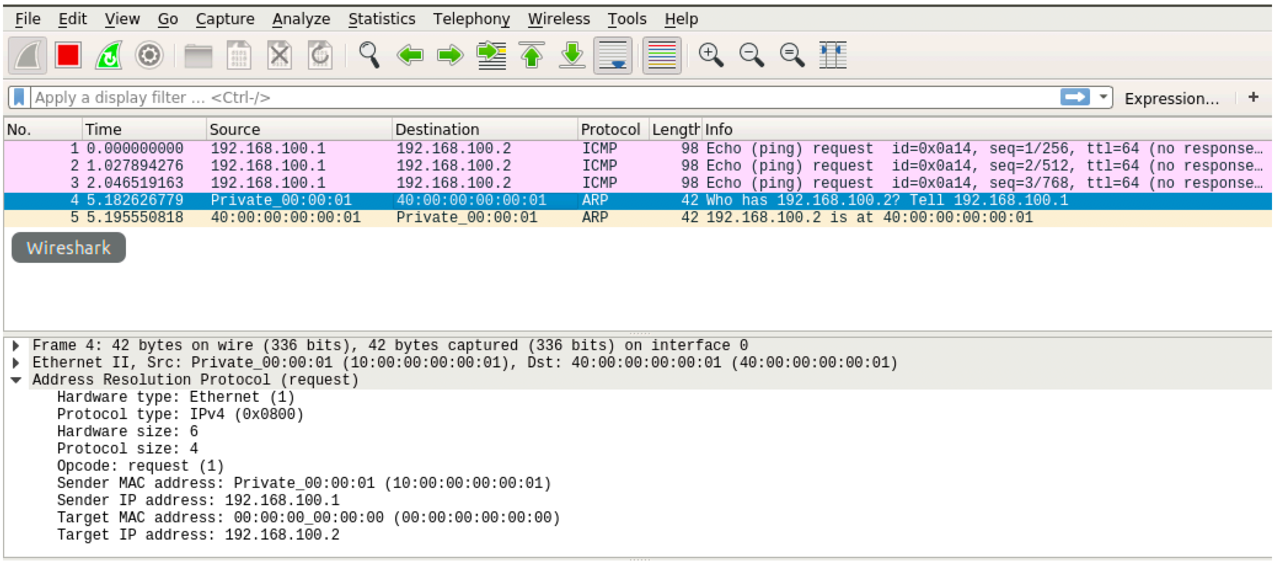


```
No.   Time         Source              Destination         Protocol Length Info
    1 0.000000000  30:00:00:00:00:01   Broadcast           ARP         42 Who has 10.1.1.2? Tell 10.1.1.1
    2 0.090824676  40:00:00:00:00:03   30:00:00:00:00:01   ARP         42 10.1.1.2 is at 40:00:00:00:00:03
    3 0.090835439  10.1.1.1            10.1.1.2            ICMP        98 Echo (ping) request  id=0x09d0, seq=1/256, ttl=64 (no response…
    4 1.002462852  10.1.1.1            10.1.1.2            ICMP        98 Echo (ping) request  id=0x09d0, seq=2/512, ttl=64 (no response…
    5 2.027072426  10.1.1.1            10.1.1.2            ICMP        98 Echo (ping) request  id=0x09d0, seq=3/768, ttl=64 (no response…
```

```
▶ Frame 2: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▶ Ethernet II, Src: 40:00:00:00:00:03 (40:00:00:00:00:03), Dst: 30:00:00:00:00:01 (30:00:00:00:00:01)
▼ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: 40:00:00:00:00:03 (40:00:00:00:00:03)
    Sender IP address: 10.1.1.2
    Target MAC address: 30:00:00:00:00:01 (30:00:00:00:00:01)
    Target IP address: 10.1.1.1
```

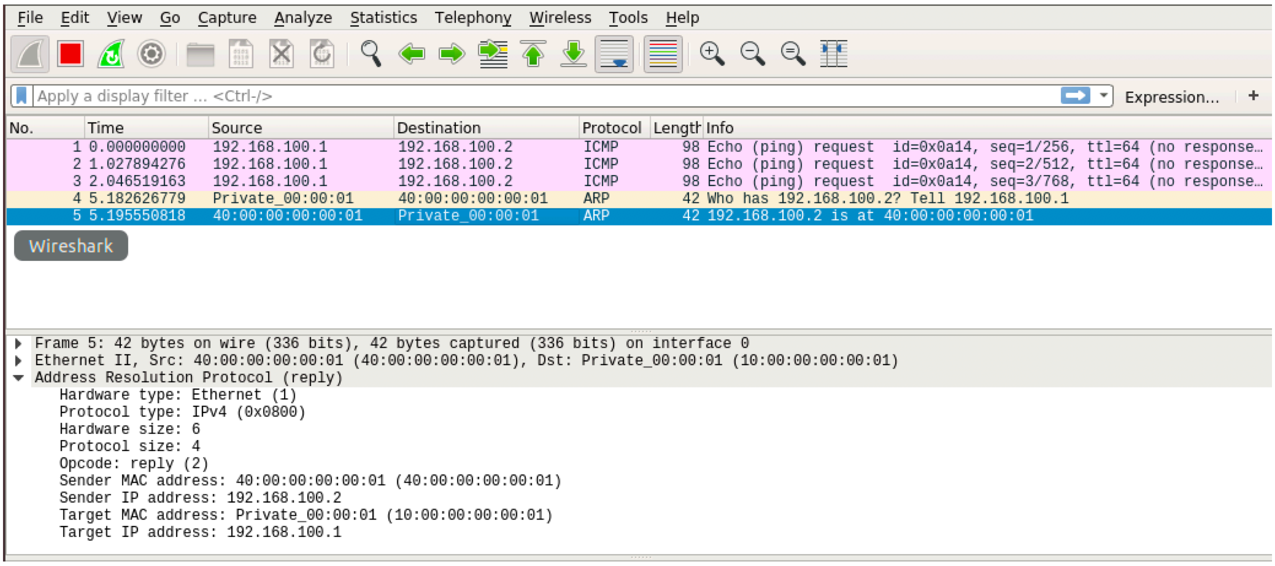
- `sudo python start_mininet.py`
- `server1 ping -c3 192.168.100.2`

The results are similar to the client ping operation above. The router received an arp request with empty target Mac address first and then sent a reply back with all information filled in. Besides, ICMP packets are not handled in this section.


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+
Wireshark

```
No.   Time          Source               Destination          Protocol Length Info
    1 0.000000000   192.168.100.1        192.168.100.2        ICMP        98 Echo (ping) request  id=0x0a14, seq=1/256, ttl=64 (no response…
    2 1.027894276   192.168.100.1        192.168.100.2        ICMP        98 Echo (ping) request  id=0x0a14, seq=2/512, ttl=64 (no response…
    3 2.046519163   192.168.100.1        192.168.100.2        ICMP        98 Echo (ping) request  id=0x0a14, seq=3/768, ttl=64 (no response…
    4 5.182626779   Private_00:00:01     40:00:00:00:00:01    ARP         42 Who has 192.168.100.2? Tell 192.168.100.1
    5 5.195550818   40:00:00:00:00:01    Private_00:00:01     ARP         42 192.168.100.2 is at 40:00:00:00:00:01
```

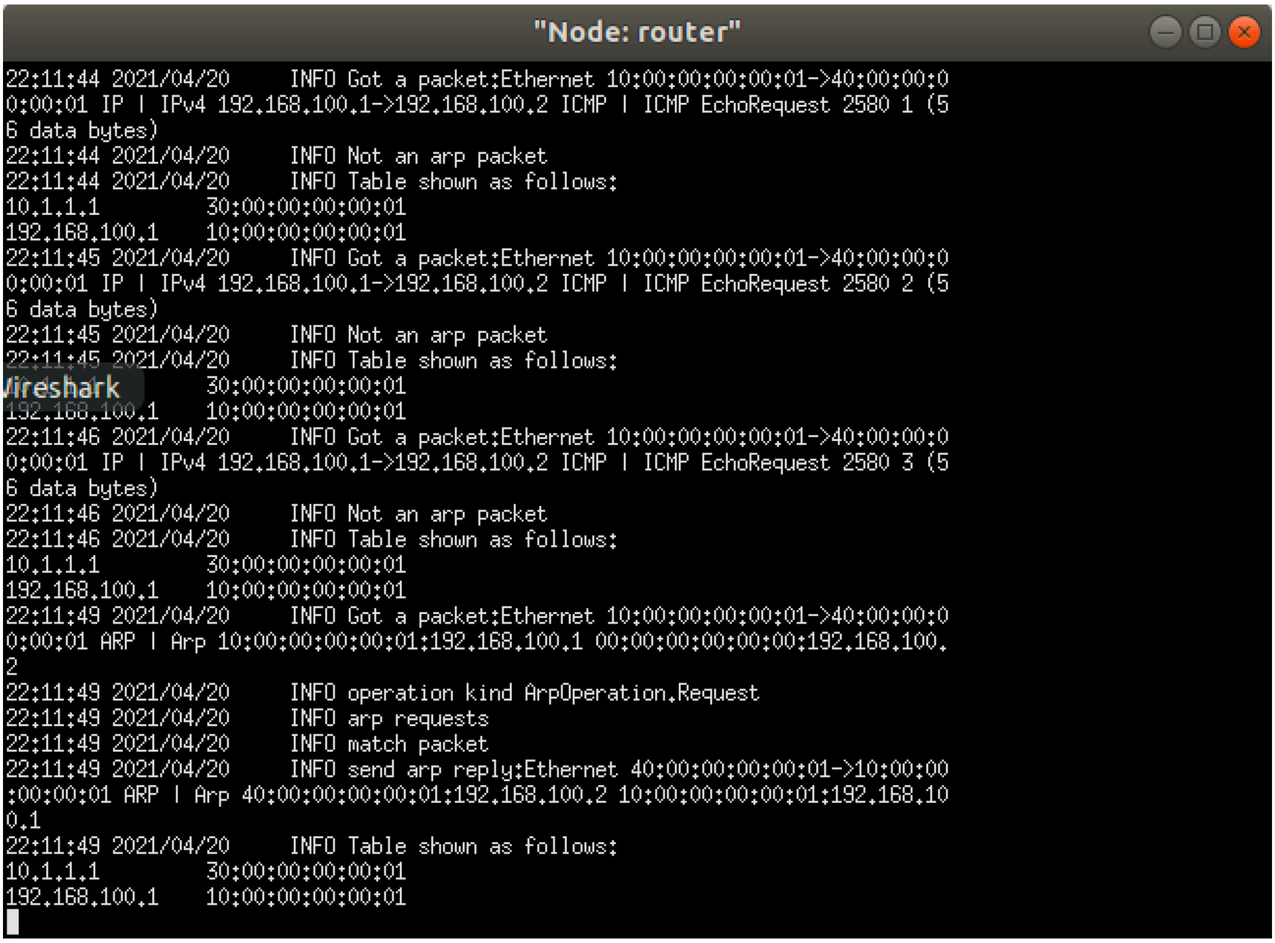
```
▸ Frame 4: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▸ Ethernet II, Src: Private_00:00:01 (10:00:00:00:00:01), Dst: 40:00:00:00:00:01 (40:00:00:00:00:01)
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: Private_00:00:01 (10:00:00:00:00:01)
    Sender IP address: 192.168.100.1
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.100.2
```


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+
Wireshark

```
No.   Time          Source               Destination          Protocol Length Info
    1 0.000000000   192.168.100.1        192.168.100.2        ICMP        98 Echo (ping) request  id=0x0a14, seq=1/256, ttl=64 (no response…
    2 1.027894276   192.168.100.1        192.168.100.2        ICMP        98 Echo (ping) request  id=0x0a14, seq=2/512, ttl=64 (no response…
    3 2.046519163   192.168.100.1        192.168.100.2        ICMP        98 Echo (ping) request  id=0x0a14, seq=3/768, ttl=64 (no response…
    4 5.182626779   Private_00:00:01     40:00:00:00:00:01    ARP         42 Who has 192.168.100.2? Tell 192.168.100.1
    5 5.195550818   40:00:00:00:00:01    Private_00:00:01     ARP         42 192.168.100.2 is at 40:00:00:00:00:01
```

```
▸ Frame 5: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▸ Ethernet II, Src: 40:00:00:00:00:01 (40:00:00:00:00:01), Dst: Private_00:00:01 (10:00:00:00:00:01)
▾ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: 40:00:00:00:00:01 (40:00:00:00:00:01)
    Sender IP address: 192.168.100.2
    Target MAC address: Private_00:00:01 (10:00:00:00:00:01)
    Target IP address: 192.168.100.1
```

The log output of the router also matches the Wireshark capture results.



```
22:11:44 2021/04/20     INFO Got a packet:Ethernet 10:00:00:00:00:01->40:00:00:0
0:00:01 IP | IPv4 192.168.100.1->192.168.100.2 ICMP | ICMP EchoRequest 2580 1 (5
6 data bytes)
22:11:44 2021/04/20     INFO Not an arp packet
22:11:44 2021/04/20     INFO Table shown as follows:
10.1.1.1         30:00:00:00:00:01
192.168.100.1    10:00:00:00:00:01
22:11:45 2021/04/20     INFO Got a packet:Ethernet 10:00:00:00:00:01->40:00:00:0
0:00:01 IP | IPv4 192.168.100.1->192.168.100.2 ICMP | ICMP EchoRequest 2580 2 (5
6 data bytes)
22:11:45 2021/04/20     INFO Not an arp packet
22:11:45 2021/04/20     INFO Table shown as follows:
10.1.1.1         30:00:00:00:00:01
192.168.100.1    10:00:00:00:00:01
22:11:46 2021/04/20     INFO Got a packet:Ethernet 10:00:00:00:00:01->40:00:00:0
0:00:01 IP | IPv4 192.168.100.1->192.168.100.2 ICMP | ICMP EchoRequest 2580 3 (5
6 data bytes)
22:11:46 2021/04/20     INFO Not an arp packet
22:11:46 2021/04/20     INFO Table shown as follows:
10.1.1.1         30:00:00:00:00:01
192.168.100.1    10:00:00:00:00:01
22:11:49 2021/04/20     INFO Got a packet:Ethernet 10:00:00:00:00:01->40:00:00:0
0:00:01 ARP | Arp 10:00:00:00:00:01:192.168.100.1 00:00:00:00:00:00:192.168.100.
2
22:11:49 2021/04/20     INFO operation kind ArpOperation.Request
22:11:49 2021/04/20     INFO arp requests
22:11:49 2021/04/20     INFO match packet
22:11:49 2021/04/20     INFO send arp reply:Ethernet 40:00:00:00:00:01->10:00:00
:00:00:01 ARP | Arp 40:00:00:00:00:01:192.168.100.2 10:00:00:00:00:01:192.168.10
0.1
22:11:49 2021/04/20     INFO Table shown as follows:
10.1.1.1         30:00:00:00:00:01
192.168.100.1    10:00:00:00:00:01
█
```

4. 总结与感想

Lab 3 is not that hard, compared to Lab 2. It is more like an introduction to the following steps of building a full IPv4 router. Can't wait to explore Lab4 and Lab5, and finally see my router running correctly.