Lab Report

Lab Name Forwarding & ARP Request

Course	Computer Network
Major	Computer Science and Technology
Id	191220129
Name	Shangyu.Xing
Email	191220129@smail.nju.edu.cn
Date	2021.04

Contents

1 Objective

- Learn address resolution protocol and how to implement it;
- learn forwarding rules of a layer-3 router;
- learn to implement hardware logic using the Switchyard framework;
- learn to capture network package using wireshark.

2 Requirements

This lab requires to implement a router who can forward packets according to preset rules and send ARP requests when necessary.

- Implement a forwarding table;
- handle ARP request;
- try out multithreading.

3 Procedure

I completed all the tasks as required, **using multithreading as implementation**. In this section, I will explain how I did my work in detail.

3.1 Implement Forwarding Table

To build the forwarding table, I need to complete the following steps:

1. initialize the table according to interfaces on the router;
2. append the file content to it.

I used a list to store data. Every entry in the list consists 3 pieces – [prefix to match: IPv4Network, next hop address: IPv4Address, interface: str]. Note that after initialization the table must be sorted according to length of netmask (descending), since the match order is from head to tail.

```
def __init__(self, interfaces: list) -> None:
    for intf in interfaces:
        self.data.append([ip_interface(str(intf.ipaddr) + '/' +
            str(intf.netmask)).network, IPv4Address('0.0.0.0'),
            intf.name])
```

```python
    with open('forwarding_table.txt', 'r') as fp:
        lines = fp.readlines()
    # [network address, subnet address, next hop address, interface]
    for line in lines:
        strs = line.split(' ')
        if strs[3][-1] == '\n':
            strs[3] = strs[3][:-1]
        self.data.append([IPv4Network(strs[0] + '/' + strs[1]), IPv4Address(strs[2]), strs[3]])
    # sort by length: longest match
    self.data.sort(key=lambda entry: int(entry[0].netmask), reverse=True)
```

Besides, to facilitate querying, I added a query method. It returns next hop address and interface when queried with an IP address.

```python
def query(self, address: IPv4Address):
    # return [next hop address, interface] or None
    for entry in self.data:
        if address in entry[0]:
            return entry[1:]
    return None
```

3.2 Send ARP Request and Forward Packet

3.2.1 Forwarding Logic

A router should do the following to forward a packet:

1. decrease its ttl field in Network header;
2. query the forwarding table or send arp request for next hop address and interface;
3. query the arp table for mac address of the next hop ip;
4. modify src and dst field in the ethernet header and send it out of the interface.

```python
hip_idx = packet.get_header_index(IPv4)
if(hip_idx != -1):
        packet[hip_idx].ttl -= 1 # modify in the future
        hip = packet[hip_idx]
        ip_intf = self.forwardTable.query(hip.dst)
        if ip_intf == None:
                pass # modify in the future
        else:
                next_ip, next_intf = tuple(ip_intf)
                intf = self.net.interface_by_name(next_intf)
                if intf.ipaddr == next_ip:
                        pass # modify in the future
                else: # create Ethernet header and send out
                        if next_ip == IPv4Address('0.0.0.0'):
                                next_ip = hip.dst
                        mac = self.arpTable.query(next_ip)
                        if mac == None:
                                # construct arp request
                                self.arpQueue.insert(packet, intf, next_ip)
                        else: # forward directly
                                eth_idx = packet.get_header_index(Ethernet)
                                packet[eth_idx].src = intf.ethaddr
                                packet[eth_idx].dst = mac
                                self.net.send_packet(intf, packet)
                                log_info(f"directly forward {packet} out {intf}")
```

3.2.2 ARP Request and Reply Handling

To maintain a queue containing packets waiting for arp reply, we should create a python class for it. The class consists of a list containing waiting packets and other related data. It has the following methods:

- **send_arp_request** – send request and update timestamp and retry times

```python
def send_arp_request(self, index):
        # increase retry, remove if exceed, return new index
        entry = self.data[index]
        if entry[4] >= self.max_retry:
                del self.data[index]
                return index
        entry[3] = time()
        entry[4] += 1
        # send request
        ether = Ethernet()
        ether.src = entry[1].ethaddr
        ether.dst = 'ff:ff:ff:ff:ff:ff'
        ether.ethertype = EtherType.ARP
        arp = Arp(operation=ArpOperation.Request,
                        senderhwaddr=entry[1].ethaddr,
                        senderprotoaddr=entry[1].ipaddr,
                        targethwaddr='ff:ff:ff:ff:ff:ff',
                        targetprotoaddr=entry[2])
        arppacket = ether + arp
        self.net.send_packet(entry[1], arppacket)
        return index + 1
```

- **insert** – insert a packet waiting for reply

```
def insert(self, packet, intf, next_ip):
        log_info(f"{packet} waiting for arp reply")
        # insert a timeout entry to be retried
        self.data.append([packet, intf, next_ip, -2 * self.timeout, 0])
```

- **release** – modify and send out blocked packets once receiving an arp reply

```
def release(self, reply):
        # send out packet and remove entry if arp reply matches
        ip = reply.senderprotoaddr
        mac = reply.senderhwaddr
        i = 0
        while i < len(self.data):
                packet = self.data[i][0]
                if self.data[i][2] == ip:
                        intf = self.data[i][1]
                        # forward packet without updating arp table which is done in caller stage
                        log_info(f"{packet} released from {intf}")
                        hip_idx = packet.get_header_index(Ethernet)
                        packet[hip_idx].src = intf.ethaddr
                        packet[hip_idx].dst = mac
                        self.net.send_packet(intf, packet)
                        del self.data[i]
                i += 1
```

- **check_timeout** – check all entries in table and resend request if it timeout

```
def check_timeout(self):
        i = 0
        while i < len(self.data):
                if time() - self.data[i][3] >= self.timeout:
                        log_info(f"{self.data[i][0]} arp
                            retry")
                        i = self.send_arp_request(i)
                else:
                        i += 1
```

Note that when an entry timeout, it is not physically deleted; instead it is marked invalid, hiding from querying or printing.

3.3 Multithreading

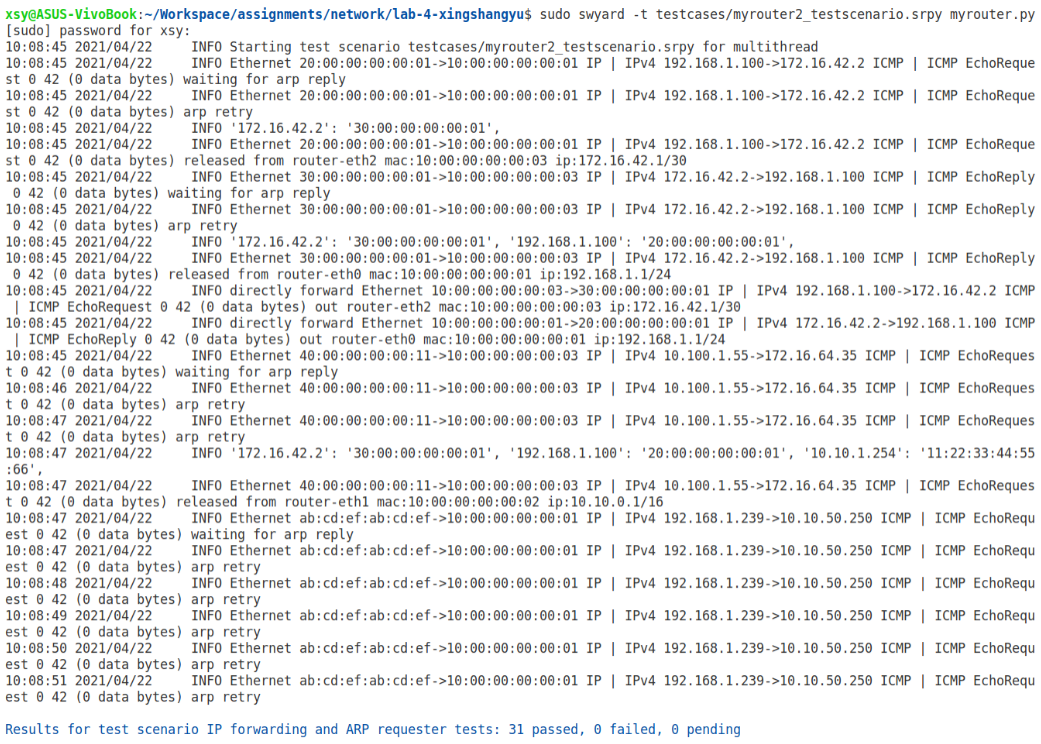
It is natural to implement another thread to maintain the queue and send arp request, while the main thread only handle received packets. If an arp reply is received, the main thread sends arp thread a signal noticing that an entry in table should be removed. Note that arp thread will terminate when main thread exits, so its property 'daemon' should be set to true.
The behavior of arp thread is very simple; it only repeatedly calls the method check_timeout of ArpQueue every 0.1s.

```
class ArpThread(threading.Thread):
    def __init__(self, queue: ArpQueue):
        threading.Thread.__init__(self, daemon=True)
        self.arpQueue = queue

    def run(self):
        while True:
            self.arpQueue.check_timeout()
            sleep(0.1)
```

4 Result

4.1 Testcase

Firstly I tested my code with switchyard testcases:



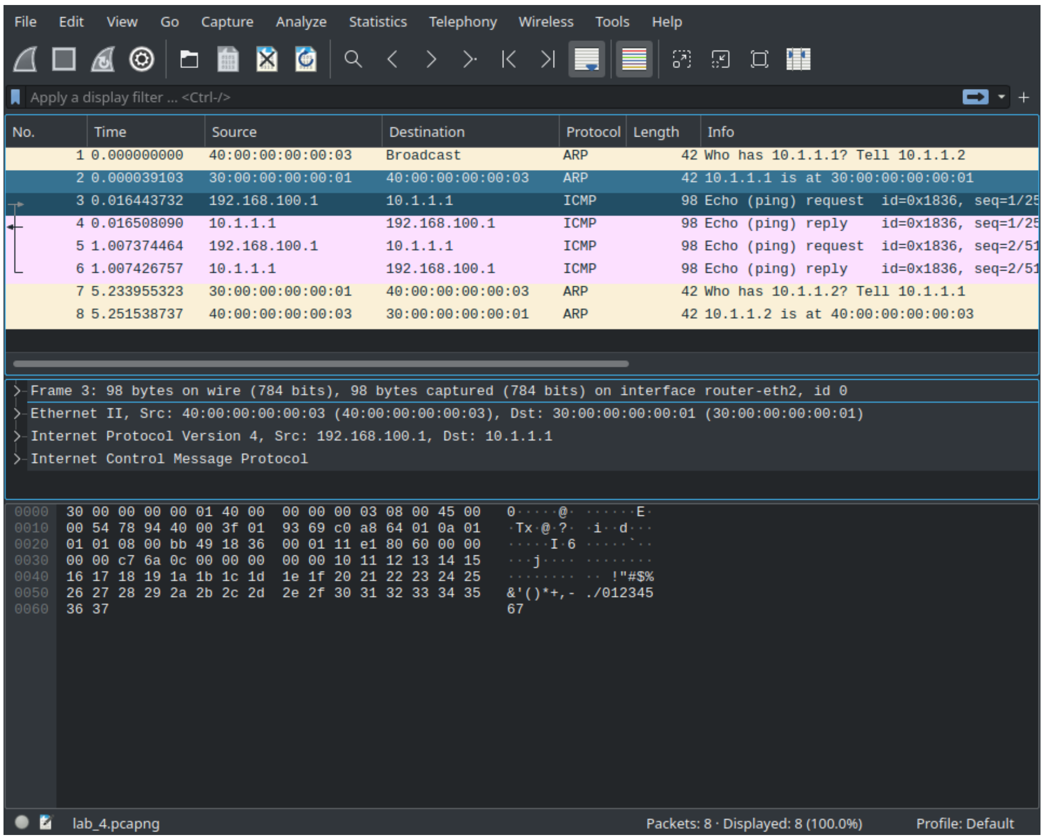
```
xsy@ASUS-VivoBook:~/Workspace/assignments/network/lab-4-xingshangyu$ sudo swyard -t testcases/myrouter2_testscenario.srpy myrouter.py
[sudo] password for xsy:
10:08:45 2021/04/22     INFO Starting test scenario testcases/myrouter2_testscenario.srpy for multithread
10:08:45 2021/04/22     INFO Ethernet 20:00:00:00:00:01->10:00:00:00:00:01 IP | IPv4 192.168.1.100->172.16.42.2 ICMP | ICMP EchoReque
st 0 42 (0 data bytes) waiting for arp reply
10:08:45 2021/04/22     INFO Ethernet 20:00:00:00:00:01->10:00:00:00:00:01 IP | IPv4 192.168.1.100->172.16.42.2 ICMP | ICMP EchoReque
st 0 42 (0 data bytes) arp retry
10:08:45 2021/04/22     INFO '172.16.42.2': '30:00:00:00:00:01',
10:08:45 2021/04/22     INFO Ethernet 20:00:00:00:00:01->10:00:00:00:00:01 IP | IPv4 192.168.1.100->172.16.42.2 ICMP | ICMP EchoReque
st 0 42 (0 data bytes) released from router-eth2 mac:10:00:00:00:00:03 ip:172.16.42.1/30
10:08:45 2021/04/22     INFO Ethernet 30:00:00:00:00:01->10:00:00:00:00:03 IP | IPv4 172.16.42.2->192.168.1.100 ICMP | ICMP EchoReply
 0 42 (0 data bytes) waiting for arp reply
10:08:45 2021/04/22     INFO Ethernet 30:00:00:00:00:01->10:00:00:00:00:03 IP | IPv4 172.16.42.2->192.168.1.100 ICMP | ICMP EchoReply
 0 42 (0 data bytes) arp retry
10:08:45 2021/04/22     INFO '172.16.42.2': '30:00:00:00:00:01', '192.168.1.100': '20:00:00:00:00:01',
10:08:45 2021/04/22     INFO Ethernet 30:00:00:00:00:01->10:00:00:00:00:03 IP | IPv4 172.16.42.2->192.168.1.100 ICMP | ICMP EchoReply
 0 42 (0 data bytes) released from router-eth0 mac:10:00:00:00:00:01 ip:192.168.1.1/24
10:08:45 2021/04/22     INFO directly forward Ethernet 10:00:00:00:00:03->30:00:00:00:00:01 IP | IPv4 192.168.1.100->172.16.42.2 ICMP
 | ICMP EchoRequest 0 42 (0 data bytes) out router-eth2 mac:10:00:00:00:00:03 ip:172.16.42.1/30
10:08:45 2021/04/22     INFO directly forward Ethernet 10:00:00:00:00:01->20:00:00:00:00:01 IP | IPv4 172.16.42.2->192.168.1.100 ICMP
 | ICMP EchoReply 0 42 (0 data bytes) out router-eth0 mac:10:00:00:00:00:01 ip:192.168.1.1/24
10:08:45 2021/04/22     INFO Ethernet 40:00:00:00:00:11->10:00:00:00:00:03 IP | IPv4 10.100.1.55->172.16.64.35 ICMP | ICMP EchoReques
t 0 42 (0 data bytes) waiting for arp reply
10:08:46 2021/04/22     INFO Ethernet 40:00:00:00:00:11->10:00:00:00:00:03 IP | IPv4 10.100.1.55->172.16.64.35 ICMP | ICMP EchoReques
t 0 42 (0 data bytes) arp retry
10:08:47 2021/04/22     INFO Ethernet 40:00:00:00:00:11->10:00:00:00:00:03 IP | IPv4 10.100.1.55->172.16.64.35 ICMP | ICMP EchoReques
t 0 42 (0 data bytes) arp retry
10:08:47 2021/04/22     INFO '172.16.42.2': '30:00:00:00:00:01', '192.168.1.100': '20:00:00:00:00:01', '10.10.1.254': '11:22:33:44:55
:66',
10:08:47 2021/04/22     INFO Ethernet 40:00:00:00:00:11->10:00:00:00:00:03 IP | IPv4 10.100.1.55->172.16.64.35 ICMP | ICMP EchoReques
t 0 42 (0 data bytes) released from router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16
10:08:47 2021/04/22     INFO Ethernet ab:cd:ef:ab:cd:ef->10:00:00:00:00:01 IP | IPv4 192.168.1.239->10.10.50.250 ICMP | ICMP EchoRequ
est 0 42 (0 data bytes) waiting for arp reply
10:08:47 2021/04/22     INFO Ethernet ab:cd:ef:ab:cd:ef->10:00:00:00:00:01 IP | IPv4 192.168.1.239->10.10.50.250 ICMP | ICMP EchoRequ
est 0 42 (0 data bytes) arp retry
10:08:48 2021/04/22     INFO Ethernet ab:cd:ef:ab:cd:ef->10:00:00:00:00:01 IP | IPv4 192.168.1.239->10.10.50.250 ICMP | ICMP EchoRequ
est 0 42 (0 data bytes) arp retry
10:08:49 2021/04/22     INFO Ethernet ab:cd:ef:ab:cd:ef->10:00:00:00:00:01 IP | IPv4 192.168.1.239->10.10.50.250 ICMP | ICMP EchoRequ
est 0 42 (0 data bytes) arp retry
10:08:50 2021/04/22     INFO Ethernet ab:cd:ef:ab:cd:ef->10:00:00:00:00:01 IP | IPv4 192.168.1.239->10.10.50.250 ICMP | ICMP EchoRequ
est 0 42 (0 data bytes) arp retry
10:08:51 2021/04/22     INFO Ethernet ab:cd:ef:ab:cd:ef->10:00:00:00:00:01 IP | IPv4 192.168.1.239->10.10.50.250 ICMP | ICMP EchoRequ
est 0 42 (0 data bytes) arp retry

Results for test scenario IP forwarding and ARP requester tests: 31 passed, 0 failed, 0 pending
```

Figure 1: Switchyard test result

It seems that switchyard testing framework works well with multithreading!

4.2 Deployment

To perform a test in mininet, I commanded server1 to ping client 2 times and ran wireshark on router's interface eth-2. I got a result of 0% drop and this record:



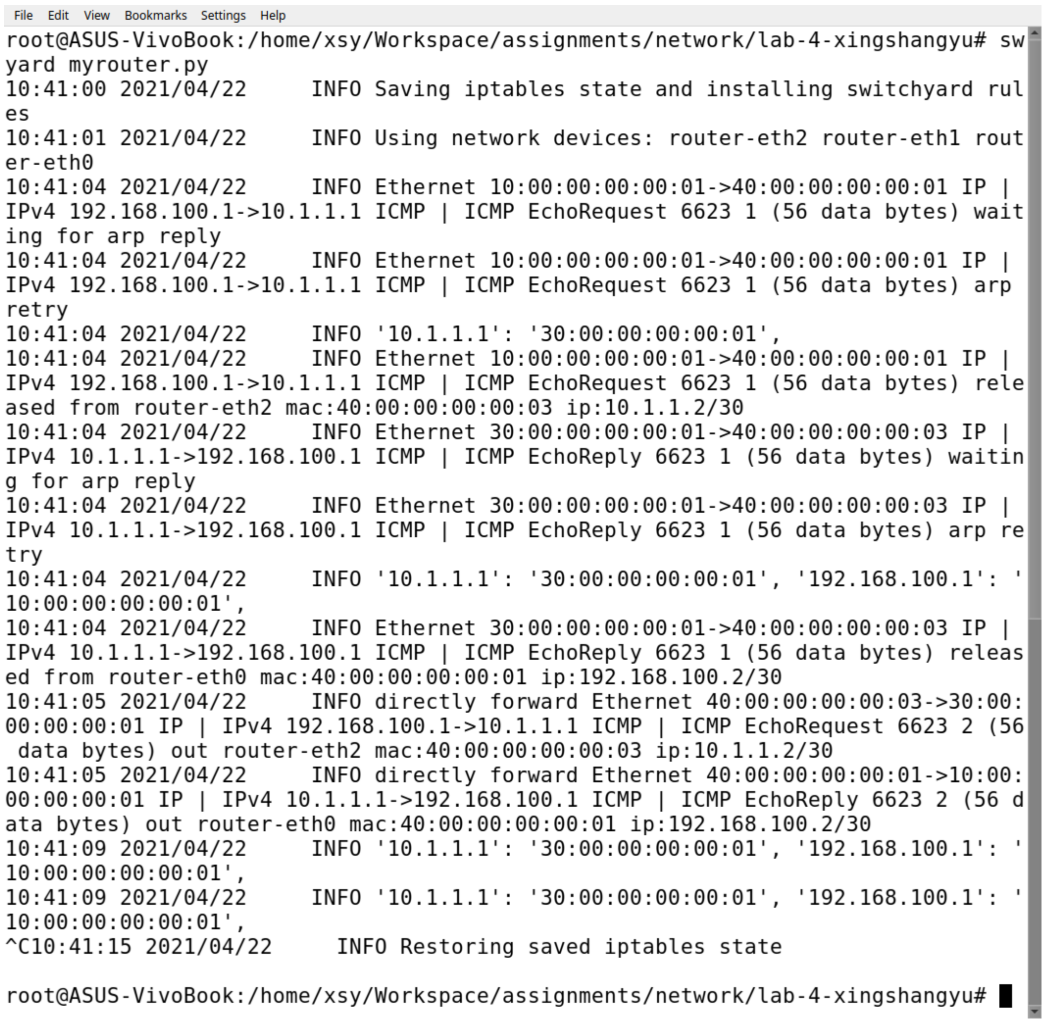
```
No.  Time         Source              Destination         Protocol Length Info
   1 0.000000000  40:00:00:00:00:03   Broadcast           ARP      42 Who has 10.1.1.1? Tell 10.1.1.2
   2 0.000039103  30:00:00:00:00:01   40:00:00:00:00:03   ARP      42 10.1.1.1 is at 30:00:00:00:00:01
   3 0.016443732  192.168.100.1       10.1.1.1            ICMP     98 Echo (ping) request  id=0x1836, seq=1/25
   4 0.016508090  10.1.1.1            192.168.100.1       ICMP     98 Echo (ping) reply    id=0x1836, seq=1/25
   5 1.007374464  192.168.100.1       10.1.1.1            ICMP     98 Echo (ping) request  id=0x1836, seq=2/51
   6 1.007426757  10.1.1.1            192.168.100.1       ICMP     98 Echo (ping) reply    id=0x1836, seq=2/51
   7 5.233955323  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
   8 5.251538737  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 3: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface router-eth2, id 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)
> Internet Protocol Version 4, Src: 192.168.100.1, Dst: 10.1.1.1
> Internet Control Message Protocol
```

```
0000  30 00 00 00 00 01 40 00  00 00 00 03 08 00 45 00   0·····@· ······E·
0010  00 54 78 94 40 00 3f 01  93 69 c0 a8 64 01 0a 01   ·Tx·@·?· ·i··d···
0020  01 01 08 00 bb 49 18 36  00 01 11 e1 80 60 00 00   ·····I·6 ·····`··
0030  00 00 c7 6a 0c 00 00 00  00 00 10 11 12 13 14 15   ···j···· ········
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ········ ·· !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                              67
```

lab_4.pcapng   Packets: 8 · Displayed: 8 (100.0%)   Profile: Default

Figure 2: Wireshark capture result



```
root@ASUS-VivoBook:/home/xsy/Workspace/assignments/network/lab-4-xingshangyu# sw
yard myrouter.py
10:41:00 2021/04/22     INFO Saving iptables state and installing switchyard rul
es
10:41:01 2021/04/22     INFO Using network devices: router-eth2 router-eth1 rout
er-eth0
10:41:04 2021/04/22     INFO Ethernet 10:00:00:00:00:01->40:00:00:00:00:01 IP |
IPv4 192.168.100.1->10.1.1.1 ICMP | ICMP EchoRequest 6623 1 (56 data bytes) wait
ing for arp reply
10:41:04 2021/04/22     INFO Ethernet 10:00:00:00:00:01->40:00:00:00:00:01 IP |
IPv4 192.168.100.1->10.1.1.1 ICMP | ICMP EchoRequest 6623 1 (56 data bytes) arp
retry
10:41:04 2021/04/22     INFO '10.1.1.1': '30:00:00:00:00:01',
10:41:04 2021/04/22     INFO Ethernet 10:00:00:00:00:01->40:00:00:00:00:01 IP |
IPv4 192.168.100.1->10.1.1.1 ICMP | ICMP EchoRequest 6623 1 (56 data bytes) rele
ased from router-eth2 mac:40:00:00:00:00:03 ip:10.1.1.2/30
10:41:04 2021/04/22     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 6623 1 (56 data bytes) waitin
g for arp reply
10:41:04 2021/04/22     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 6623 1 (56 data bytes) arp re
try
10:41:04 2021/04/22     INFO '10.1.1.1': '30:00:00:00:00:01', '192.168.100.1': '
10:00:00:00:00:01',
10:41:04 2021/04/22     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 6623 1 (56 data bytes) releas
ed from router-eth0 mac:40:00:00:00:00:01 ip:192.168.100.2/30
10:41:05 2021/04/22     INFO directly forward Ethernet 40:00:00:00:00:03->30:00:
00:00:00:01 IP | IPv4 192.168.100.1->10.1.1.1 ICMP | ICMP EchoRequest 6623 2 (56
 data bytes) out router-eth2 mac:40:00:00:00:00:03 ip:10.1.1.2/30
10:41:05 2021/04/22     INFO directly forward 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 6623 2 (56 d
ata bytes) out router-eth0 mac:40:00:00:00:00:01 ip:192.168.100.2/30
10:41:09 2021/04/22     INFO '10.1.1.1': '30:00:00:00:00:01', '192.168.100.1': '
10:00:00:00:00:01',
10:41:09 2021/04/22     INFO '10.1.1.1': '30:00:00:00:00:01', '192.168.100.1': '
10:00:00:00:00:01',
^C10:41:15 2021/04/22     INFO Restoring saved iptables state

root@ASUS-VivoBook:/home/xsy/Workspace/assignments/network/lab-4-xingshangyu# 
```

Figure 3: Router's log

What happened in the network was this:

1. server1 broadcast arp packet, and the router found that it was directed at it;
2. The router made an arp response to server1 and cached server1's (ip, mac) pair in its arp table;
3. server1 extract mac from the response, then sent echo requests packet to the router;
4. the router looked for its forwarding table and got next hop address, but it didn't konw the mac address;
5. the router sent an arp request to next hop and got a reply;
6. the router cached arp data in its arp table and forwarded the packet;
7. client made an echo reply;
8. the router looked for its forwarding table and got next hop address which is server1;
9. the router retrieved mac address of server1 from its cached arp table and forwarded the packet;
10. repeated the above procedure for the second echo request, but no arp requests should be made because all the data needed was in arp table.

5 Summary

- Knowing how to use tools effectively will greatly enhance working efficiency;
- Object-oriented programming is very suitable for implementing data structure such as arp table, forwarding table and arp queue;
- English reading and writing skills are important.