



# Lab Report

Lab Name Arp

Course Computer Network

Major Computer Science and Technology

Id 191220129

Name Shangyu.Xing

Email 191220129@smail.nju.edu.cn

Date 2021.04

# Contents

<b>1</b>	<b>Objective</b>	<b>2</b>
<b>2</b>	<b>Requirements</b>	<b>2</b>
<b>3</b>	<b>Procedure</b>	<b>2</b>
3.1	Handle ARP Request . . . . .	2
3.2	Cached ARP Table . . . . .	3
<b>4</b>	<b>Result</b>	<b>4</b>
4.1	Handle ARP Request . . . . .	4
4.2	Cached ARP Table . . . . .	6
<b>5</b>	<b>Summary</b>	<b>7</b>

# 1 Objective

- Learn address resolution protocol and how to implement it;
- 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 respond to ARP requests and cache an ARP table.

- Handle ARP request;
- implement a cached ARP table.

# 3 Procedure

In this section, I will explain how I implement the router in detail.

## 3.1 Handle ARP Request

The logic to handle ARP request is very simple. The procedure is listed as follows:

1. Upon receiving a packet, get its header to check whether it is an arp packet;
2. if it is, check all the router's interfaces. If there is a match, create the arp reply packet and send it out through the same interface which the arp request come in; else do nothing.

```
1 def handle_packet(self, recv: switchyard.llnetbase.  
    ReceivedPacket):  
2     timestamp, ifaceName, packet = recv  
3     # TODO: your logic here  
4     harp = packet.get_header(Arp)  
5     if harp is not None:  
6         for intf in self.net.interfaces():  
7             if intf.ipaddr == harp.targetprotoaddr:
```

```

8         response = create_ip_arp_reply(intf.ethaddr,
          harp.senderhwaddr, intf.ipaddr, harp.
            senderprotoaddr)
9         self.net.send_packet(ifaceName, response)

```

## 3.2 Cached ARP Table

To maintain an arp table, we should create a python class for it. The class consists of a constant indicating timeout value and a dict which stores (ip -> (mac, timestamp)) as an entry of table. It has the following methods:

- update – create an entry or update the timestamp of a specific entry;
- get – query for mac address with an ip;
- \_\_str\_\_ – its representation in str when printing.

```

1 class ArpTable:
2     data = {} # ip -> (mac, timestamp)
3     timeout = 10
4
5     def update(self, ip, mac):
6         self.data[ip] = (mac, time())
7
8     def get(self, mac):
9         t = self.data.get(mac)
10        if t == None or time() - t[1] >= self.timeout:
11            return None
12        return t[0]
13
14    def __str__(self) -> str:
15        res = ''
16        for key in self.data:
17            if time() - self.data[key][1] < self.timeout:
18                res += f'"{key.compressed}": '{self.data[key]
19                res += f'[0].toStr()}', "
20        return res

```

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

To update the table and print it out when an arp packet arrives, we should add the following lines at the end of function 'handle\_packet':

```

1 self.arpTable.update(harp.senderprotoaddr, harp.senderhwaddr)
2 log_info(str(self.arpTable))

```

## 4 Result

### 4.1 Handle ARP Request

Firstly I tested my code with switchyard testcases:

```
xsy@ASUS-VivoBook:~/Workspace/assignments/network/Lab-3-xingshangyu$ sudo swyard -t testcases/myrouter1_testscenario.srpy myrouter.py
[sudo] password for xsy:
19:11:35 2021/04/07 INFO Starting test scenario testcases/myrouter1_testscenario.srpy
19:11:35 2021/04/07 INFO '192.168.1.100': '30:00:00:00:00:01', '10.10.1.1': '60:00:de:ad:be:ef',
19:11:35 2021/04/07 INFO '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',
19:11:35 2021/04/07 INFO '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',

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!
```

Figure 1: Switchyard test result

To perform a test in mininet, I commanded server1 to ping the router 3 times and ran wireshark on server1. I got a result of 100% drop and this record:

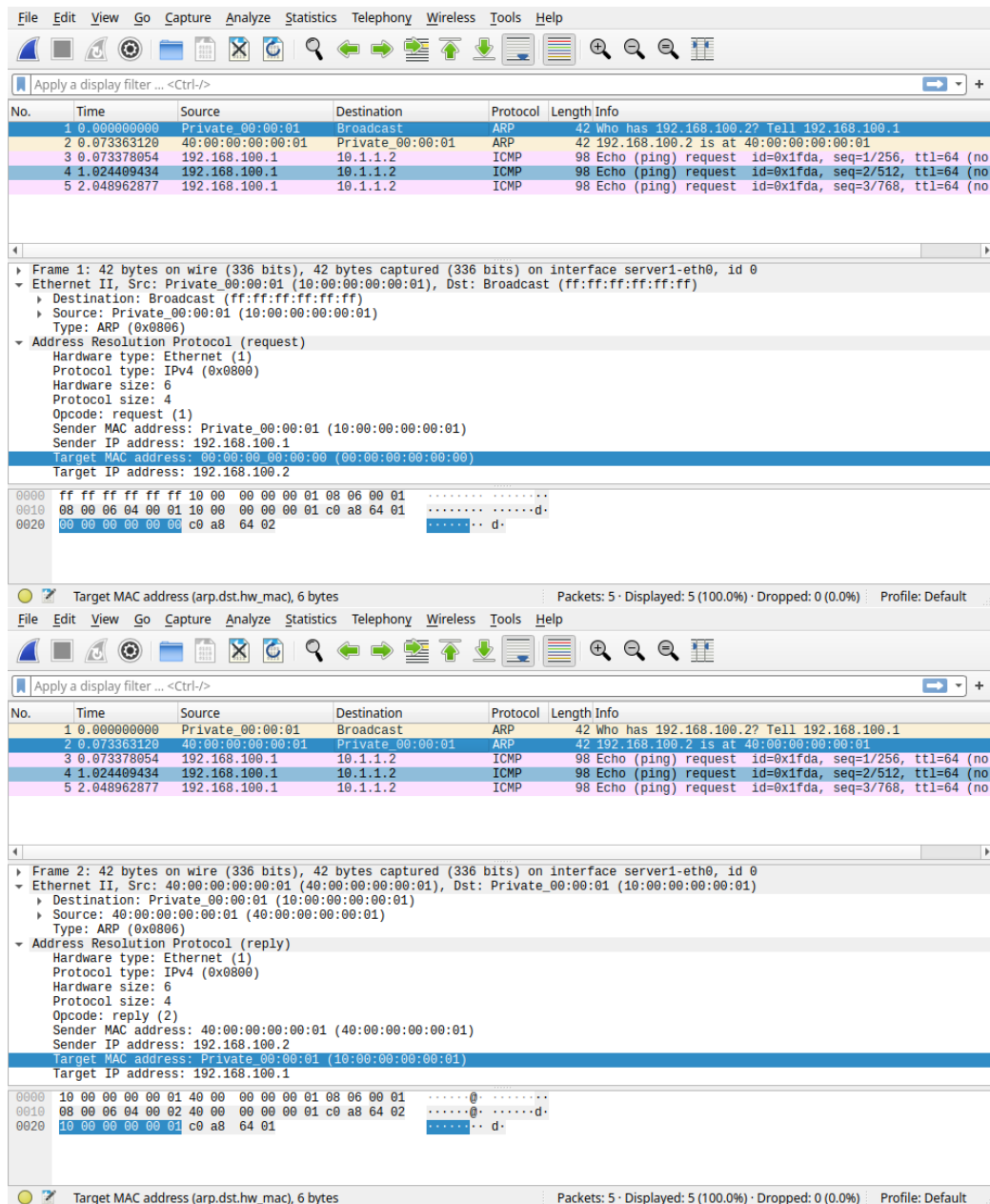


Figure 2: Wireshark capture result

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;
3. Server1 extract mac from the response, then sent echo requests packet to the router three times subsequently;
4. the router received echo requests, but didn't reply.

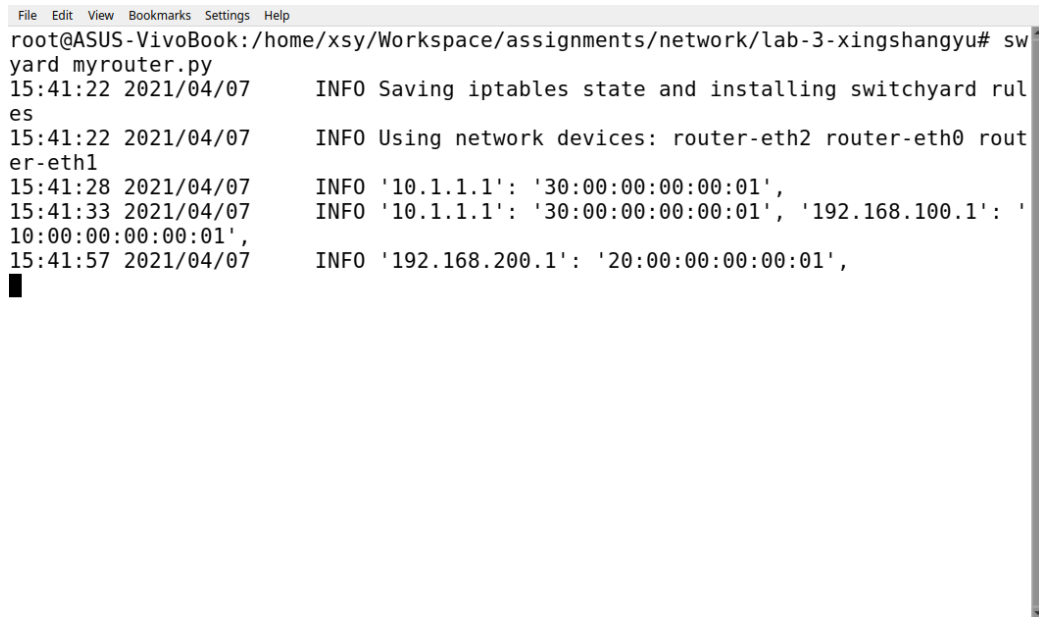
As a result, wireshark could capture arp request/response and echo request but no echo response.

## 4.2 Cached ARP Table

I set the timeout value to 10 second, and made the following commands:

```
1 client ping -c 1 <router>
2 (wait 5s)
3 server1 ping -c 1 <router>
4 (wait 15s)
5 server2 ping -c 1 <router>
```

Router's log (which contains the cached table at different timestamp):



```
root@ASUS-VivoBook:/home/xsy/Workspace/assignments/network/lab-3-xingshangyu# sw
yard myrouter.py
15:41:22 2021/04/07      INFO Saving iptables state and installing switchyard rul
es
15:41:22 2021/04/07      INFO Using network devices: router-eth2 router-eth0 rout
er-eth1
15:41:28 2021/04/07      INFO '10.1.1.1': '30:00:00:00:00:01',
15:41:33 2021/04/07      INFO '10.1.1.1': '30:00:00:00:00:01', '192.168.100.1': '
10:00:00:00:00:01',
15:41:57 2021/04/07      INFO '192.168.200.1': '20:00:00:00:00:01',
```

Figure 3: Router's log in mininet

As can be seen in the picture above, the entries in the arp table did not timeout in 5s but timeout after 15s. The detailed procedure is this:

1. Client broadcast arp packet, and the router stored client's (ip -> (mac, timestamp)) in its arp table;
2. The router made an arp response to client;
3. Client sent echo requests packet to the router, but it didn't reply;
4. Server1 did the same thing, and the router stored server1's (ip -> (mac, timestamp)). At this time, the previous entry (which is the client's information) had not timeout, so the table now contained 2 entries;
5. Server2 did the same thing, and the router stored server2's (ip -> (mac, timestamp)). At this time, the previous entries (which are the client's and server1's information) had timeout, so the table now contained 1 entry.

## 5 Summary

- Knowing how to use tools effectively will greatly enhance working efficiency;



- English reading and writing skills are important.