

EL5373

INTERNET ARCHITECTURE AND PROTOCOLS

Runze Dong

N10264442

rd1711@nyu.edu

Workstation: APAH Othello_I

MAC: f8:0f:41:c4:7f:aa

IP: 128.238.66.104

Lab Report 2

Due Sept 30, 2014

Exercise 1

The host has 3 interfaces, eth0 (Ethernet), lo (Local loopback) and wlan0 (WLAN).

The Ethernet interface reads and sends Ethernet frames from or to the medium.

WLAN interface support data send/receive in a wireless distribution system.

The loopback interface is used for debugging. It behaves as a separate data link interface, but packets sent to it will be put in the IP input queue and will not be transmitted on the medium.

Ethernet MTU 1500 bytes

Local loopback MTU 65536 bytes

WLAN MTU 1500 bytes

The network is subnetted.

It has subnet mask 255.255.255.0. The IP address 128.238.66.104 is a Class B IP. It has 16 bits Network ID. So this subnet mask results in 8 bits subnet ID and 8 bits host ID.

-1

Exercise 2

The 127.0.0.1 interface is on.

We can't see any ICMP message sent from host in the tcpdump output. Because 127.0.0.1 is local loopback address. When we ping this address, the message will not be sent outside through ethernet interface. In generally, tcpdump listens to ethernet interface activity. So it can't capture any data or ICMP message.

Exercise 4

The **target IP address** in ARP request is 128.238.66.105

At the MAC layer, the **destination Ethernet address** in ARP request is broadcast ff: ff: ff: ff: ff: ff

The **frame type** in the Ethernet frame is ARP 0x0806.

128.238.66.105 send the ARP reply.

ARP Request

```
▼ Address Resolution Protocol (request)
  Hardware type: Ethernet (1)
  Protocol type: IP (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (1)
  Sender MAC address: WistronI_c4:7f:aa (f8:0f:41:c4:7f:aa)
  Sender IP address: 128.238.66.104 (128.238.66.104)
  Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
  Target IP address: 128.238.66.105 (128.238.66.105)
```

ARP Reply

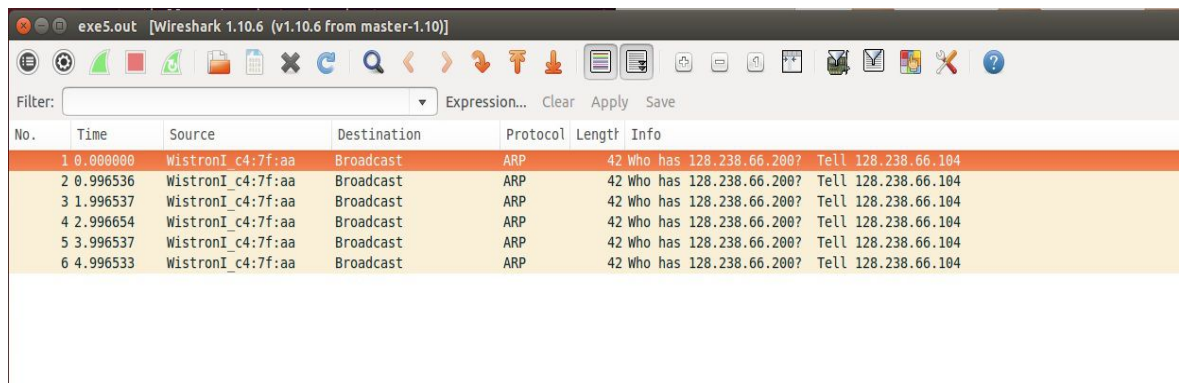
```
▼ Address Resolution Protocol (reply)
  Hardware type: Ethernet (1)
  Protocol type: IP (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: reply (2)
  Sender MAC address: WistronI_c4:7f:a8 (f8:0f:41:c4:7f:a8)
  Sender IP address: 128.238.66.105 (128.238.66.105)
  Target MAC address: WistronI_c4:7f:aa (f8:0f:41:c4:7f:aa)
  Target IP address: 128.238.66.104 (128.238.66.104)
```

```
guest@othello1: ~
guest@othello1:~$ arp -a
? (128.238.66.103) at f8:0f:41:c3:88:0d [ether] on eth0
? (128.238.66.105) at <incomplete> on eth0
172-27-222-50.DYNAPOOL.NYU.EDU (172.27.222.50) at 78:6c:1c:e0:5e:05 [ether] on wlan0
NYUNY-VL1125-GW.WIRELESS.NET.NYU.EDU (172.27.222.1) at 00:00:5e:00:01:53 [ether] on wlan0
guest@othello1:~$
```

```
guest@othello1: ~
guest@othello1:~$ arp -a
? (128.238.66.103) at f8:0f:41:c3:88:0d [ether] on eth0
? (128.238.66.105) at f8:0f:41:c4:7f:a8 [ether] on eth0
172-27-222-50.DYNAPOOL.NYU.EDU (172.27.222.50) at 78:6c:1c:e0:5e:05 [ether] on wlan0
NYUNY-VL1125-GW.WIRELESS.NET.NYU.EDU (172.27.222.1) at 00:00:5e:00:01:53 [ether]
on wlan0
guest@othello1:~$
```

Exercise 5

While ARP request is broadcast to try establish connection with non-existence host, 128.238.66.200, it will attempt 6 times to send request APR and stop.



The image shows a Wireshark capture of network traffic. The filter is set to 'Filter:'. The packet list shows 6 ARP requests from source 'WistronI_c4:7f:aa' to destination 'Broadcast'. The protocol is ARP, and the length is 42. The info column shows 'Who has 128.238.66.200? Tell 128.238.66.104'.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	WistronI_c4:7f:aa	Broadcast	ARP	42	Who has 128.238.66.200? Tell 128.238.66.104
2	0.996536	WistronI_c4:7f:aa	Broadcast	ARP	42	Who has 128.238.66.200? Tell 128.238.66.104
3	1.996537	WistronI_c4:7f:aa	Broadcast	ARP	42	Who has 128.238.66.200? Tell 128.238.66.104
4	2.996654	WistronI_c4:7f:aa	Broadcast	ARP	42	Who has 128.238.66.200? Tell 128.238.66.104
5	3.996537	WistronI_c4:7f:aa	Broadcast	ARP	42	Who has 128.238.66.200? Tell 128.238.66.104
6	4.996533	WistronI_c4:7f:aa	Broadcast	ARP	42	Who has 128.238.66.200? Tell 128.238.66.104

Exercise 8

ICMP echo request and ICMP echo reply are used by ping.

In order to test the accessibility to remote host, ping command will send ICMP echo request message to the host. And if the host is available, it will feedback ICMP echo reply message.

```
guest@othello1: ~  
0x0010: 80ee 4264 0800 e03b 176c 0058  
19:30:05.331871 f8:0f:41:c3:86:c6 > f8:0f:41:c4:7f:aa, ethertype IPv4 (0x0800),  
length 60: 128.238.66.100 > 128.238.66.104: ICMP echo reply, id 5996, seq 88, le  
ngth 8  
0x0000: 4500 001c 7a33 0000 4001 7a05 80ee 4264  
0x0010: 80ee 4268 0000 e83b 176c 0058 0000 0000  
0x0020: 0000 0000 0000 0000 0000 0000 0000  
19:30:06.330266 f8:0f:41:c4:7f:aa > f8:0f:41:c3:86:c6, ethertype IPv4 (0x0800),  
length 42: 128.238.66.104 > 128.238.66.100: ICMP echo request, id 5996, seq 89,  
length 8  
0x0000: 4500 001c 8eca 4000 4001 256e 80ee 4268  
0x0010: 80ee 4264 0800 e03a 176c 0059  
19:30:06.331069 f8:0f:41:c3:86:c6 > f8:0f:41:c4:7f:aa, ethertype IPv4 (0x0800),  
length 60: 128.238.66.100 > 128.238.66.104: ICMP echo reply, id 5996, seq 89, le  
ngth 8  
0x0000: 4500 001c 7a34 0000 4001 7a04 80ee 4264  
0x0010: 80ee 4268 0000 e83a 176c 0059 0000 0000  
0x0020: 0000 0000 0000 0000 0000 0000 0000  
19:30:07.330269 f8:0f:41:c4:7f:aa > f8:0f:41:c3:86:c6, ethertype IPv4 (0x0800),  
length 42: 128.238.66.104 > 128.238.66.100: ICMP echo request, id 5996, seq 90,  
length 8  
0x0000: 4500 001c 8ecb 4000 4001 256d 80ee 4268  
0x0010: 80ee 4264 0800 e039 176c 005a  
19:30:07.331195 f8:0f:41:c3:86:c6 > f8:0f:41:c4:7f:aa, ethertype IPv4 (0x0800),
```

```
guest@othello1: ~  
guest@othello1:~$ ping -sv 128.238.66.100  
PING 128.238.66.100 (128.238.66.100) 0(28) bytes of data.  
8 bytes from 128.238.66.100: icmp_seq=1 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=2 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=3 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=4 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=5 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=6 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=7 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=8 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=9 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=10 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=11 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=12 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=13 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=14 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=15 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=16 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=17 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=18 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=19 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=20 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=21 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=22 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=23 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=24 ttl=64  
8 bytes from 128.238.66.100: icmp_seq=25 ttl=64
```


Exercise 10

We can't see any traffic sent on the network.

Because 128.238.60.100 is not in the same subnet with 128.238.66.104. We can't connect with another host in the different subnet.

```
guest@othello1: ~  
0 packets captured  
0 packets received by filter  
0 packets dropped by kernel  
guest@othello1:~$ ifconfig  
eth0      Link encap:Ethernet  HWaddr f8:0f:41:c4:7f:aa  
          inet addr:128.238.66.104  Bcast:128.238.66.255  Mask:255.255.255.0  
          inet6 addr: fe80::fa0f:41ff:fec4:7faa/64 Scope:Link  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:2936 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:1601 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:286962 (286.9 KB)  TX bytes:144482 (144.4 KB)  
          Interrupt:20 Memory:f7d00000-f7d20000  
  
lo        Link encap:Local Loopback  
          inet addr:127.0.0.1  Mask:255.0.0.0  
          inet6 addr: ::1/128 Scope:Host  
          UP LOOPBACK RUNNING  MTU:65536  Metric:1  
          RX packets:1797 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:1797 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:0  
          RX bytes:335544 (335.5 KB)  TX bytes:335544 (335.5 KB)  
  
guest@othello1: ~  
guest@othello1:~$ ping 128.238.60.100  
PING 128.238.60.100 (128.238.60.100) 56(84) bytes of data.
```

ICMP type 3. Code fields, 1.

Exercise 11

In first case, one of the two hosts with duplicate IP (fenchi, 128.238.66.104) telnets successfully with a host with unique IP (kenchi, 128.238.66.106).

The ARP table indicates the 128.238.66.104 points to fenchi's MAC address, f8:0f:41:c4:7f:a8. Then the other host (yachi, 128.238.66.104) also telnets the same host (kenchi, 128.238.66.106), it replaces the previous host and the ARP table update, 128.238.66.104 points to yachi's MAC address f8:0f:41:c4:7f:aa.

When two hosts with duplicate IP try to telnet with other hosts. The last host attempting to telnet will replace the previous. And the ARP table also update with the new MAC address of last host.

In second case, kenchi telnets fenchi.

In third case, guchi telnets yachi.

For second&third cases, it depends on whose ARP reply come first. we find that when a client tries to telnet some hosts which have duplicate IP each other, all of these hosts will send ARP reply to that client. And the host whose ARP reply is received firstly by the client will telnet the client successfully. The other hosts will be detected as duplicate IP and rejected to connect.

Exercise 12

Case 1. Yachi can ping one of the hosts that have the correct subnet mask.

In order to check whether fenchi is in the same subnet with itself, yachi will make & operation between fenchi's IP and its subnet mask.

yachi XXX.104 (0110 1000) & XXX.240(1111 0000) = 0110 0000

fenchi XXX.105 (0110 1001) & XXX.240(1111 0000) = 0110 0000

So yachi will think fenchi has the same Network ID and send message to fenchi correctly.

Case 2. Network can't be reachable.

guchi XXX.121 (0111 1001) & XXX.240(1111 0000) = 0111 0000

fenchi XXX.105 (0110 1001) & XXX.240(1111 0000) = 0110 0000

As I explain in case 1, guchi will find fenchi isn't in the same subnet with itself because 0111 doesn't match 0110. guchi can't reach fenchi.

Case 3. One host with correct subnet mask can ping yachi normally.

The correct subnet mask is 255.255.255.000. For each four host, they are considered in the same subnet with this mask. and yachi can also reply normally.

Case 4. Network unreachable.

Because of guchi's wrong subnet 255.255.255.240 and its IP 128.238.66.121, it will cause guchi fail to reply others hosts.