[Packet Capture Using Wireshark]

- **과제 환경**

  Host OS: Windows 10

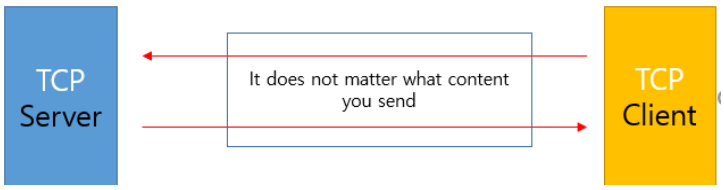
  Guest OS on Virtual machine: Oracle VM Virtual Box 6.0.4 with linux, ubuntu 18.04.2

  Packet Capture Tool: WireShark

- **1. TCP Server/Client Programming**

  : TCP packet analysis, result screenshot & report

  < Capture the TCP packets, Find the identifiers in the captured TCP packets and Explain why fields in the packet have that values. (addr, port num, eth type…) >

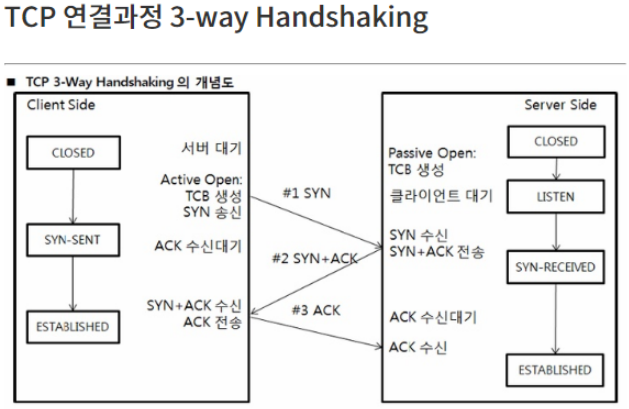


TCP sever Ip addr: 192.168.56.**242**

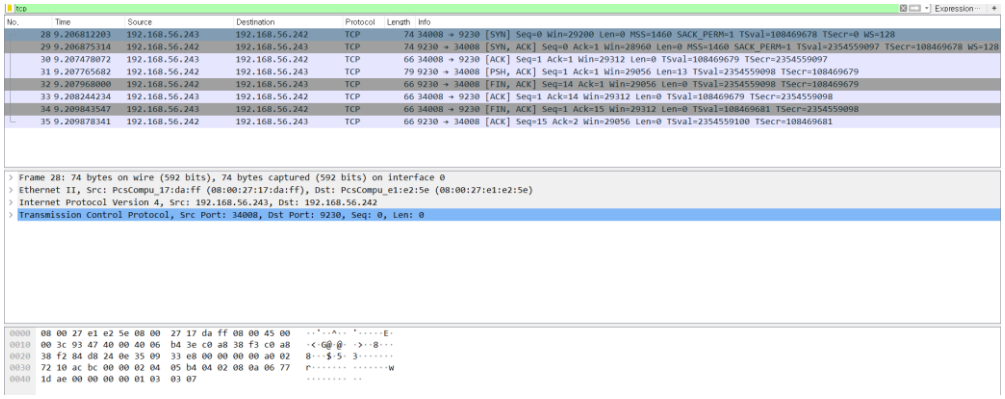
TCP client Ip addr: 192.168.56.**243**

Interface id: 0 (enp0s3)

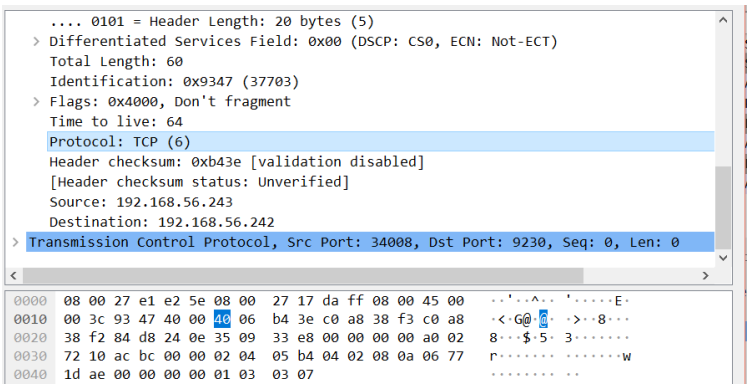
- **TCP sever & client connection 연결: 3-hand shaking**



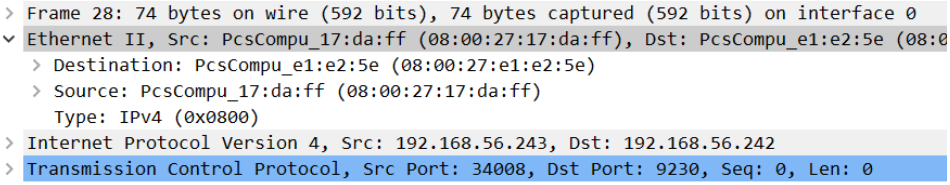
1) Client -> Server: SYN



3-핸드 쉐이크 과정의 첫번째 단계인, 클라이언트가 서버에게 SYN을 보내는 패킷이다. 첫번째로 address를 살펴보면, source address에 client(192.168.56.243)가 destination address에 server(192.168.56.242)에게 전송을 하고 있음을 알 수 있다. 이때, ip address는 dot(.)으로 구분된 32바이트의 주소 체계이며, 위의 스크린샷에서 볼 수 있듯이 ipv4에 해당한다. Port를 보면, 프로그램을 실행시킬 때 할당한 destination port는 9230(server용)으로 할당한 값으로 지정되어 있으며, source port는 34008(client용)로 자동 할당되었음 역시 확인할 수 있다. 또 SYN를 보면 시퀀스 번호가 0인데, 이는 TCP에서 세션을 성립할 때, 초기에 가장 먼저 보내는 패킷의 시퀀스 번호를 설정하여 세션을 연결하는데 사용됨도 알 수 있다.

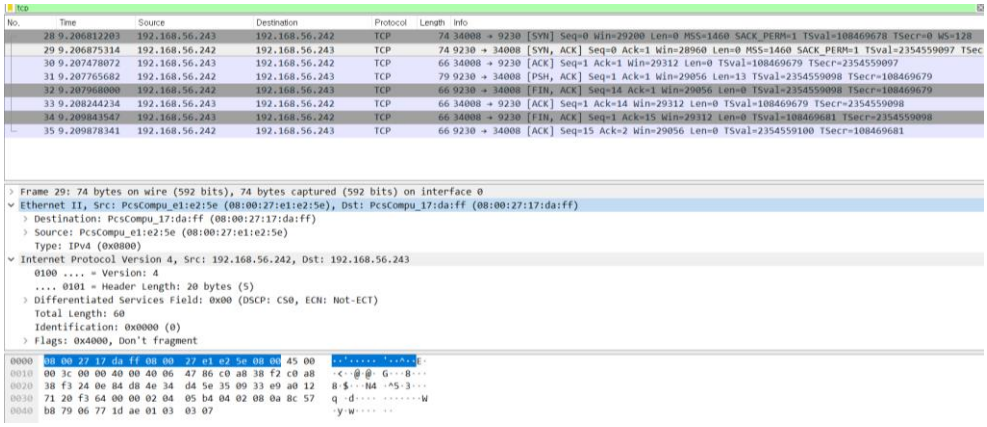


프로토콜 번호가 6번으로 TCP임을 알 수 있고,



Ethertype이 0x0800으로 IPv4를 상위 프로토콜로 하고 있음을 볼 수 있다.

2) Server -> Client: SYN, ACK



위는 connection을 연결하기 위한 두번째 패킷이며, 서버가 클라이언트에게 SYN, ACK를 보내는 부분이다. 1단계에서는 클라이언트가 서버에게 전송했지만, 2단계는 반대로 서버가 클라이언트에게 보내는 것이므로, 패킷의 src와 dest address가 위와는 반대로 server와 client의 주소로 매핑되어 있음을 확인할 수 있다.

```
Type: IPv4 (0x0800)  Protocol: TCP (6)
```

Ethertype이 0x0800으로 Ipv4임을, protocol 넘버가 6이므로 TCP임을 다시 한번 볼 수 있다.

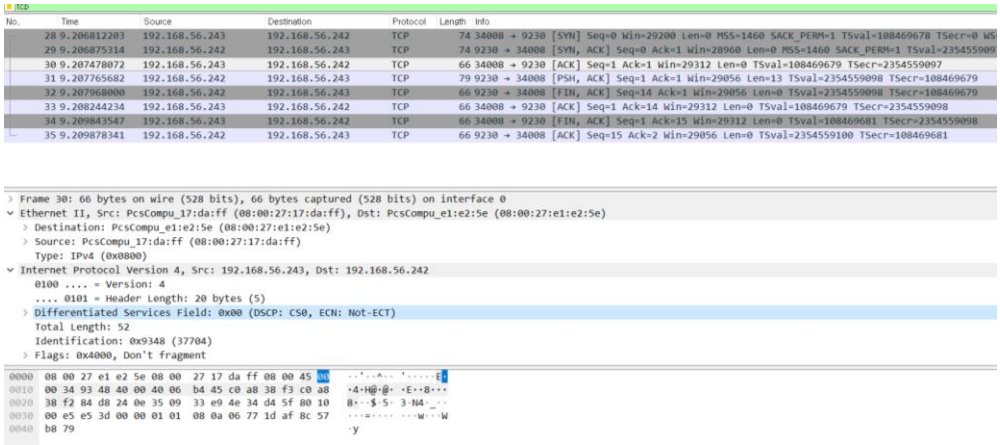
```
Transmission Control Protocol, Src Port: 9230, Dst Port: 34008, Seq: 0, Ack: 1,
   Source Port: 9230
   Destination Port: 34008
   [Stream index: 0]
   [TCP Segment Len: 0]
   Sequence number: 0    (relative sequence number)
   [Next sequence number: 0    (relative sequence number)]
   Acknowledgment number: 1    (relative ack number)
```

Port와 seq, ack를 살펴보면, server가 client에게 보내는 것이므로, 이전에 정해진 server의 port 9230, client의 port 34008가 앞서 클라이언트가 서버에게 보낼 때와 반대로 포트번호도 src, dest가 각 server, client로 매핑되었다. 여기 스크린샷에서 ACK를 보면, 상대방으로부터 패킷을 받았다는 것(이전 과정)을 확인할 수 있는데, ACK가 1이 되었기 때문이다. 이는 일반적으로 TCP계층에서 시퀀스번호에 1을 더하여 보낸다고 한다.

```
[SEQ/ACK analysis]
    [This is an ACK to the segment in frame: 28]
    [The RTT to ACK the segment was: 0.000063111 seconds]
    [iRTT: 0.000665869 seconds]
```

위의 스크린샷으로부터, 커넥션 연결을 위한 첫번째 핸드쉐이킹 과정, 프레임 28번에 대한 ACK였음을 알 수 있다.

3) Client -> Server: ACK



Connection 성립을 위한 3-핸드쉐이킹 과정의 마지막 단계인, 클라이언트가 서버에게 ACK를 보내는 패킷의 스크린샷이다. address부터 확인하면, 클라이언트의 ip 주소와 서버의 ip주소가 source와 destination에 매핑되어 있어서 클라이언트가 서버에게 전송하는 것임을 알 수 있다.

```
Type: IPv4 (0x0800) Protocol: TCP (6)
```

역시나 이서타입이 IPv4에 해당하는 값을, 프로토콜 넘버는 TCP에 해당하는 값을 담고 있다.

```
Transmission Control Protocol, Src Port: 34008, Dst Port: 9
    Source Port: 34008
    Destination Port: 9230
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 1    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
```

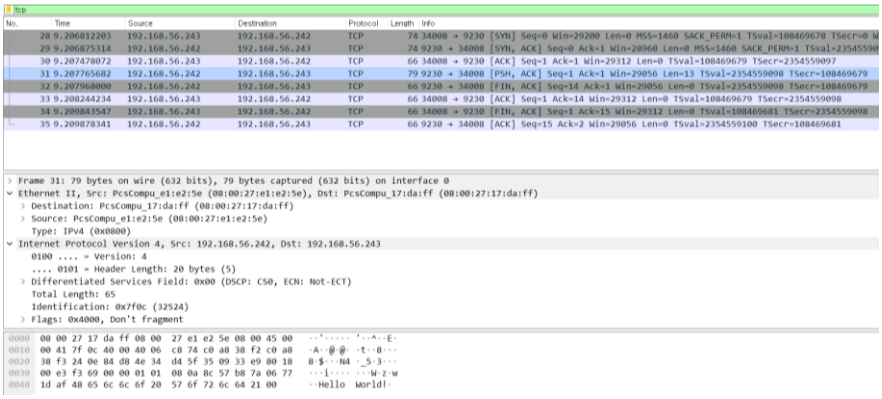
Port를 살펴보면, source port와 destination port가 클라이언트, 서버의 port 넘버에 각 잘 매핑되어있고,

```
[SEQ/ACK analysis]
    [This is an ACK to the segment in frame: 29]
    [The RTT to ACK the segment was: 0.000602758 seconds]
    [iRTT: 0.000665869 seconds]
```

ACK도 살펴보면, 바로 이전 단계인 29번 프레임에서 서버가 클라이언트에게 전송한 패킷에 대한 응답임을 확인할 수 있고, ACK는 수신자의 입장에서 송신자로부터 앞으로 받아야할 다음 데이터의 seq인데, 앞선 29번 프레임에서 ACK가 1이었으므로, 프레임 30의 seq는 1인 것이 일치한다. 3-way-handshaking 과정에서 seq가 initial sequence number의 기능을 함을 1, 2, 3 단계를 통해서 확인하였다.

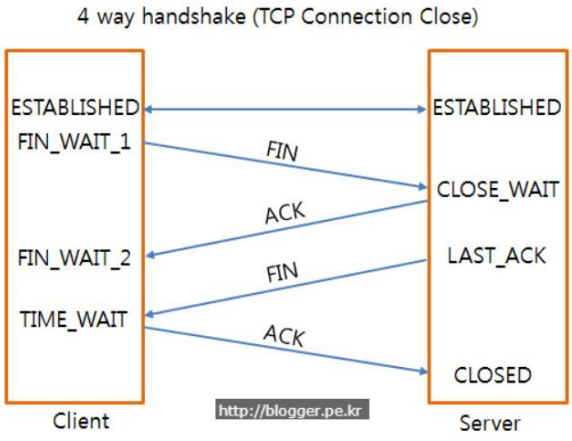
4) Connection between Client & Server done!

Cf. PSH 플래그



프레임 31을 보면, 서버의 주소와 포트가 source에, 클라이언트의 주소와 포트가 destination에 매핑되어있고, PSH 플래그가 설정되어있음을 볼 수 있다. 이는 송신자 입장에서는 application layer은 이 데이터를 버퍼에 쌓지 말고 즉시 보내라는 뜻으로 TCP layer에 알리는 기능을 하고, 수신자 입장에서는 이 데이터를 버퍼에 쌓지 말고 즉시 application layer로 올리라고 TCP layer가 인식하게하는 역할을 한다. 이 패킷은 연결이 설정된 후에, 데이터 전송이 발생했음을 알 수 있다. 이때, 즉시 전송되어야 한다는 것은, 송신자가 더 이상 송신할 데이터가 없다는 것을 수신자에 알리기 위함으로 해석할 수 있으며, 데이터전송이 끝나게 됨을 생각할 수 있다.

- **전송을 다하면, connection 종료 과정**



위는 TCP connection 종료의 일반적인 4-way handshaking 과정이다.

Cf. 서버가 순차적으로 ACK, FIN을 클라이언트에게 보내야 일반적이지만, 서버나 클라이언트 프로그램의 버그 혹은 운영체제의 버그로 ACK, FIN을 모두 전송하는 경우가 많다고 한다. 또 대부분 클라이언트가 종료하지만, 서버와 클라이언트 모두 종료가 가능하다.

아래는 위의 패킷에 이어 실제로 캡쳐한 패킷의 TCP connection 종료 과정이다.

(Frame 33 -> Frame 32 -> Frame 34 -> Frame 35)

1) Client -> Server: ACK

```
> Frame 33: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
> Ethernet II, Src: PcsCompu_17:da:ff (08:00:27:17:da:ff), Dst: PcsCompu_e1:e2:5e (08:00:27:e1:e2:5e)
> Internet Protocol Version 4, Src: 192.168.56.243, Dst: 192.168.56.242
> Transmission Control Protocol, Src Port: 34008, Dst Port: 9230, Seq: 1, Ack: 14, Len: 0
```

```
Type: IPv4 (0x0800)   Protocol: TCP (6)
```

위의 33번 패킷에서는 클라이언트가 서버에게 ACK를 보내고 있다. 따라서 source address, port와 destination address, port에는 각 클라이언트와 서버의 주소 및 포트 넘버가 매핑되어있고 이서타입을 통해 IPv4임을, 프로토콜 필드를 통해 TCP layer의 패킷임을 확인할 수 있다.

```
v [SEQ/ACK analysis]
    [This is an ACK to the segment in frame: 31]
    [The RTT to ACK the segment was: 0.000478552 seconds]
    [iRTT: 0.000665869 seconds]
```

앞의 31번 packet의 ACK가 1이었으므로, 이어지는 패킷 33에서는 seq가 1임이고, ACK가 14임으로 뒤에 이어질 패킷의 seq가 14임을 예상할 수 있다. 이때 ACK는 이전 패킷인 31번 패킷에 대한 응답이었음을 확인할 수 있다.

2) Server -> Client: FIN, ACK

```
> Frame 32: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
> Ethernet II, Src: PcsCompu_e1:e2:5e (08:00:27:e1:e2:5e), Dst: PcsCompu_17:da:ff (08:00:27:17:da:ff)
> Internet Protocol Version 4, Src: 192.168.56.242, Dst: 192.168.56.243
v Transmission Control Protocol, Src Port: 9230, Dst Port: 34008, Seq: 14, Ack: 1, Len: 0
```

```
Type: IPv4 (0x0800)   Internet Protocol Version 4, Src: 192.168.56.242, Dst: 192.168.56.243
                        0100 .... = Version: 4

Protocol: TCP (6)     Source Port: 9230
                      Destination Port: 34008
```

서버가 클라이언트에게 전송하고 있으니까, source address, port와 destination address, port가 각 서버와 클라이언트에 매핑되고 있다. 또 이서타입은 IPv4이며, 프로토콜도 역시 TCP임을 확인할 수 있다. 이때 앞의 패킷에서 예상했듯이, 앞의 패킷의 ACK가 14임으로 이 패킷의 seq는 14이며, ACK는 1로 함으로써 이어지는 패킷의 seq가 1이 될 것임을 다시 한번 예상할 수 있다.

```
.... .... ...1 = Fin: Set
v [Expert Info (Chat/Sequence): Connection finish (FIN)]
    [Connection finish (FIN)]
```

여기서는 FIN 플래그가 세팅되어 있는데, 이는 연결 종료 요청을 뜻한다. 세션 연결을 종료할 때 사용되며, 더 이상 전송할 데이터가 없다는 것을 나타낸다. 서버가 클라이언트에게 종료 요청을 했으며, 후에 클라이언트가 이를 ACK로 받아들이고

connection을 절반 폐쇄한다.

3) Client -> Server: FIN, ACK

```
> Frame 34: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
> Ethernet II, Src: PcsCompu_17:da:ff (08:00:27:17:da:ff), Dst: PcsCompu_e1:e2:5e (08:00:27:e1:e2:5e)
> Internet Protocol Version 4, Src: 192.168.56.243, Dst: 192.168.56.242
> Transmission Control Protocol, Src Port: 34008, Dst Port: 9230, Seq: 1, Ack: 15, Len: 0
```

Addr와 port를 먼저 보자면, 클라이언트가 서버에게 보내는 것이므로 source와 destination의 주소와 포트는 각 클라이언트와 서버에 해당한다. 역시나 IPv4이고 TCP 프로토콜임을 알 수 있다. 앞 패킷의 ACK가 1이었으므로, 이 패킷의 seq는 1이고 ACK를 15로 설정해서 마지막 패킷의 seq를 15라고 예상할 수 있다. 여기서 클라이언트가 서버가 요청한 종료 요청을 받아들이고, 클라이언트에서 서버로의 연결도 종료 요청을 보낸다.

```
v [SEQ/ACK analysis]
    [This is an ACK to the segment in frame: 32]
    [The RTT to ACK the segment was: 0.001875547 seconds]
    [iRTT: 0.000665869 seconds]
```

여기서 언급한 ACK가 이전의 32번 프레임의 FIN에 해당하는 응답임을 확인할 수 있다.

4) Server -> Client: ACK

⇨ Client로부터 마지막 ACK를 수신한 Server가 소켓을 closed한다.

클라이언트가 FIN 세그먼트를 보내고 서버는 ACK 전송이 가능하다.

```
> Frame 35: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
> Ethernet II, Src: PcsCompu_e1:e2:5e (08:00:27:e1:e2:5e), Dst: PcsCompu_17:da:ff (08:00:27:17:da:ff)
> Internet Protocol Version 4, Src: 192.168.56.242, Dst: 192.168.56.243
> Transmission Control Protocol, Src Port: 9230, Dst Port: 34008, Seq: 15, Ack: 2, Len: 0
```

위의 스크린샷을 보면, 연결 종료 과정의 마무리로 서버가 클라이언트에게 ACK를 보낸다. 따라서 source와 destination의 address, port에는 각 서버와 클라이언트의 주소와 포트 번호가 해당되고, 위에서 IPv4와 TCP를 역시나 확인할 수 있다.

```
v [SEQ/ACK analysis]
    [This is an ACK to the segment in frame: 34]
    [The RTT to ACK the segment was: 0.000034794 seconds]
    [iRTT: 0.000665869 seconds]
```

이때의 ACK가 이전의 34번 프레임의 FIN에 해당하는 응답임을 위의 스크린샷에서 확인할 수 있다.

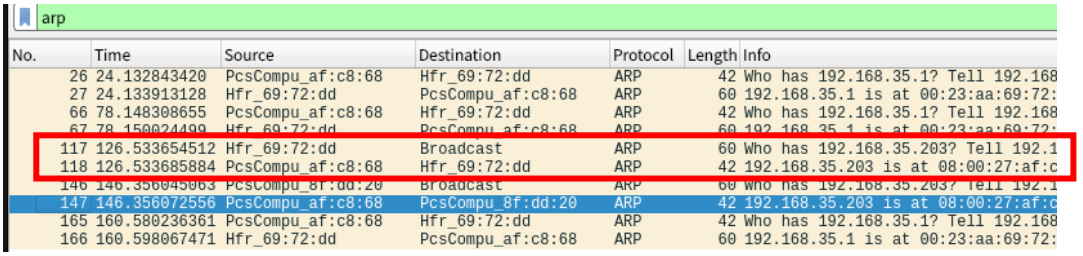
➔ 위의 8개의 TCP 패킷을 통해서, TCP connection 연결, 데이터 전송, connection 종료에 대한 과정을 identifier를 통해 분석해보았다. 아래는 ARP 패킷의 Request, Reply에 대한 분석이다.

- **2. ARP Packet Capture with Request and Reply**

: ARP packet analysis, result screenshot & report

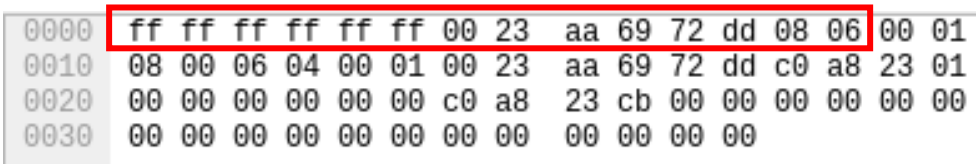
< Capture the ARP Request & Reply packet (with the pictures), Find the identifiers in the captured ARP packets, Explain why the fields in the packet have that values. >

두 host 의 IP 주소와 MAC 주소는 아래와 같다.
호스트 A 의 IP 주소 192.168.35.**1** MAC 주소 **00:23:aa:69:72:dd**
호스트 B 의 IP 주소 192.168.35.**203** MAC 주소 **08:00:27:af:c8:68**



위를 보면 ARP 요청, 응답 패킷들은 쌍을 이루고 있음을 알 수 있다. 이 중 117번, 118번 패킷을 분석해보았다. ARP 패킷 분석은 헤더 양식에 주소를 채워보면서 분석해보았다.

아래는 ARP request에 해당하는 117번 packet이다.



빨간 박스에 해당하는 필드는 이더넷 헤더이며 그 후는 ARP 헤더이다.

<table>
<tr><td colspan="6">Ethernet 패킷분석</td></tr>
<tr><td colspan="6">Destination MAC Add (6 byte)</td></tr>
<tr><td colspan="6">ff ff ff ff ff ff (브로드캐스트)</td></tr>
<tr><td colspan="6">Source MAC Add (6 byte)</td></tr>
<tr><td colspan="6">00 23 aa 69 72 dd (호스트 A PC 의 MAC 주소)</td></tr>
<tr><td colspan="6">Type (2 byte)</td></tr>
<tr><td colspan="6">08 06 (ARP 헤더)</td></tr>
<tr><td colspan="6">ARP 헤더</td></tr>
<tr><td>H/W Type
(2 byte)</td><td>protocol type
(2 byte)</td><td>H/W Length
(1byte)</td><td>protocol length
(1 byte)</td><td colspan="2">OP
(2 byte)</td></tr>
<tr><td>00 01
Ethernet : 1</td><td>08 00
IPv4</td><td>06</td><td>04</td><td colspan="2">00 01
ARP request</td></tr>
<tr><td colspan="4">SA (Sender ethernet address) (6 byte)</td><td colspan="2">Sender IP address
(4 byte)</td></tr>
<tr><td colspan="4">00 23 aa 69 72 dd
호스트 A PC 의 MAC 주소</td><td colspan="2">C0 a8 23 01
192.168.35.1</td></tr>
<tr><td colspan="4">DA (Target ethernet address) (6 byte)</td><td colspan="2">Target IP address
(4 byte)</td></tr>
<tr><td colspan="4">00 00 00 00 00 00
대상의 MAC 주소를 모르니 의미 없는 값을 넣는다.</td><td colspan="2">c0 a8 23 cb
192.168.35.203
찾는 호스트 B PC 의 IP</td></tr>
</table>

호스트 A는 호스트 B의 MAC 주소를 알기 위해서 로컬 네트워크에 broadcast(ff ff ff ff ff ff) 패킷을 보낸다. 117번 packet의 info에 보면 Who has 호스트 B의 IP주소? Tell 호스트 A의 IP주소라고 되어있다.

아래는 ARP reply에 해당하는 118번 packet이다.

```
0000  00 23 aa 69 72 dd 08 00  27 af c8 68 08 06 00 01
0010  08 00 06 04 00 02 08 00  27 af c8 68 c0 a8 23 cb
0020  00 23 aa 69 72 dd c0 a8  23 01
```

마찬가지로, 빨간 박스 안은 이더넷 헤더이고 그 후는 ARP 헤더이다.

<table>
<tr><td colspan="6">Ethernet 패킷분석</td></tr>
<tr><td colspan="6">Destination MAC Add (6 byte)</td></tr>
<tr><td colspan="6">00 23 aa 69 72 dd (호스트 A PC 의 MAC 주소)</td></tr>
<tr><td colspan="6">Source MAC Add (6 byte)</td></tr>
<tr><td colspan="6">08 00 27 af c8 68
호스트 B PC 의 MAC 주소</td></tr>
<tr><td colspan="6">Type (2 byte)</td></tr>
<tr><td colspan="6">08 06 (ARP 헤더)</td></tr>
<tr><td colspan="6">ARP 헤더</td></tr>
<tr><td>H/W Type
(2 byte)</td><td>protocol type
(2 byte)</td><td>H/W
Length
(1byte)</td><td>protocol
length
(1 byte)</td><td>OP
(2 byte)</td><td></td></tr>
<tr><td>00 01
Ethernet : 1</td><td>08 00
IPv4</td><td>06</td><td>04</td><td>00 02
ARP reply</td><td></td></tr>
<tr><td colspan="4">SA (Sender ethernet address) (6 byte)</td><td colspan="2">Sender IP address
(4 byte)</td></tr>
<tr><td colspan="4">08 00 27 af c8 68
호스트 B PC 의 MAC 주소</td><td colspan="2">c0 a8 23 cb
192.168.35.203
보내는 B 의 IP</td></tr>
<tr><td colspan="4">DA (Target ethernet address) (6 byte)</td><td colspan="2">Target IP address
(4 byte)</td></tr>
<tr><td colspan="4">00 23 aa 69 72 dd
호스트 A PC 의 MAC 주소</td><td colspan="2">c0 a8 23 01
192.168.35.1
A 의 IP</td></tr>
</table>

호스트 B의 IP주소를 갖는 호스트 B가 자신의 MAC주소를 호스트 A에게 보내줌을 볼 수 있다. 이 패킷의 info에 보면 B의 IP주소 at B의 MAC주소로, B의 MAC주소를 얻었음을 알 수 있다.

➔ 패킷 분석에 따르면, ARP request는 MAC 주소를 알기 위해 보내는 것으로 이더넷 헤더의 목적지 MAC 주소에 broadcast주소(ff ff ff ff ff f)를 담아 보낸다. Broadcast에 모든 호스트들은 패킷을 보고 자신의 IP를 갖는 패킷을 수집하여 ARP request를 보냈던 호스트에게 자신의 MAC 주소를 보낸다.

-과제 끝입니다. 고맙습니다😊-