

Packet Sniffing with RAW Socket

유명성

1. Ethernet frame

Packet sniffing in linux

TCP/UDP Socket에선 프로토콜 헤더를 얻을 수 없다.

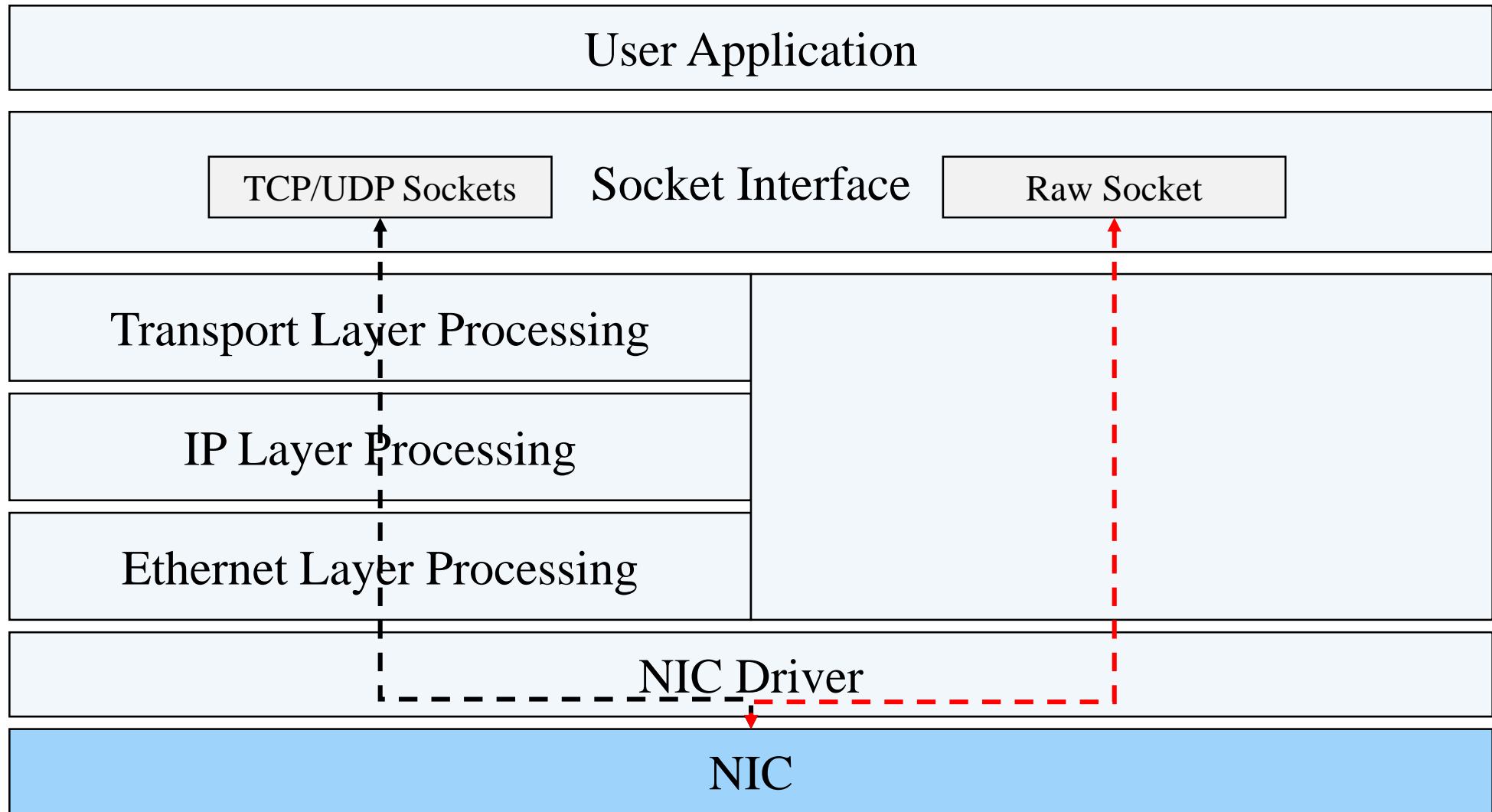
Packet에서 TC0 헤더를 얻고 싶다면 -> L4에 직접 접근

Packet에서 IP 헤더를 얻고 싶다면 -> L3에 직접 접근

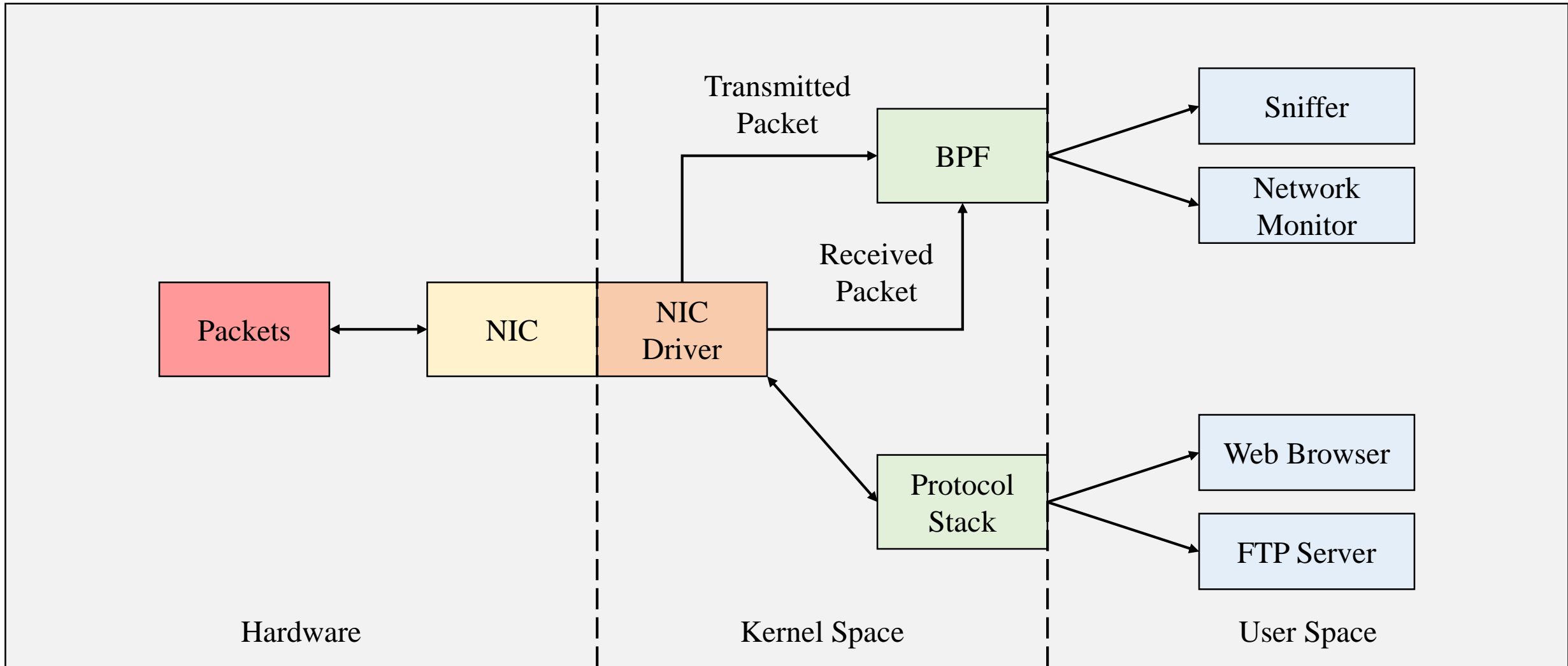
RAW_SOCKET

Packet에서 Ethernet 헤더를 얻고 싶다면 -> L2에 직접 접근 → PACKET_SOCKET

* PACKET_SOCKET은 RAW_SOCKET의 특수한 형태



Berkeley Packet Filter(BPF)



PAKCET_SOCK in linux

```
## Linux OS
ETH_P_ALL = 0x0003
sniff_sock = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(ETH_P_ALL))
# Optional
sniff_sock.bind("Interface name", 0) # NIC이름을 입력해 해당 NIC로 들어오는 패킷만 리스닝
# bind 하지 않으면 모든 NIC를 대상으로 리스닝
```

1. Ethernet Frame

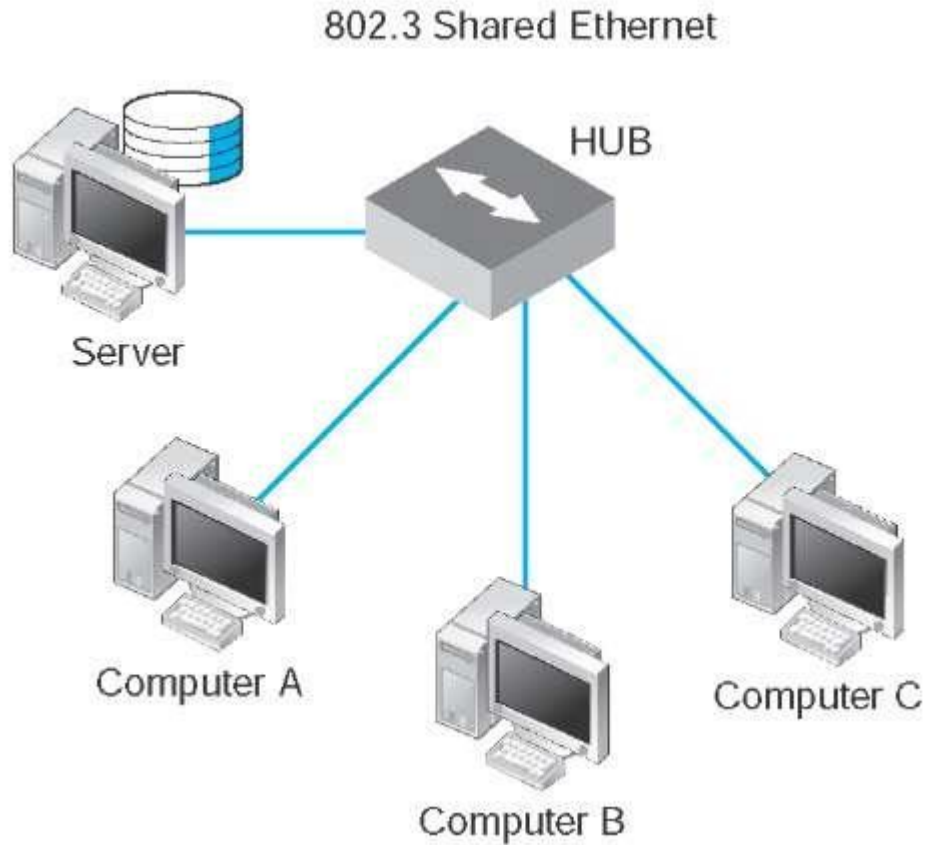
1.1 Ethernet

Ethernet

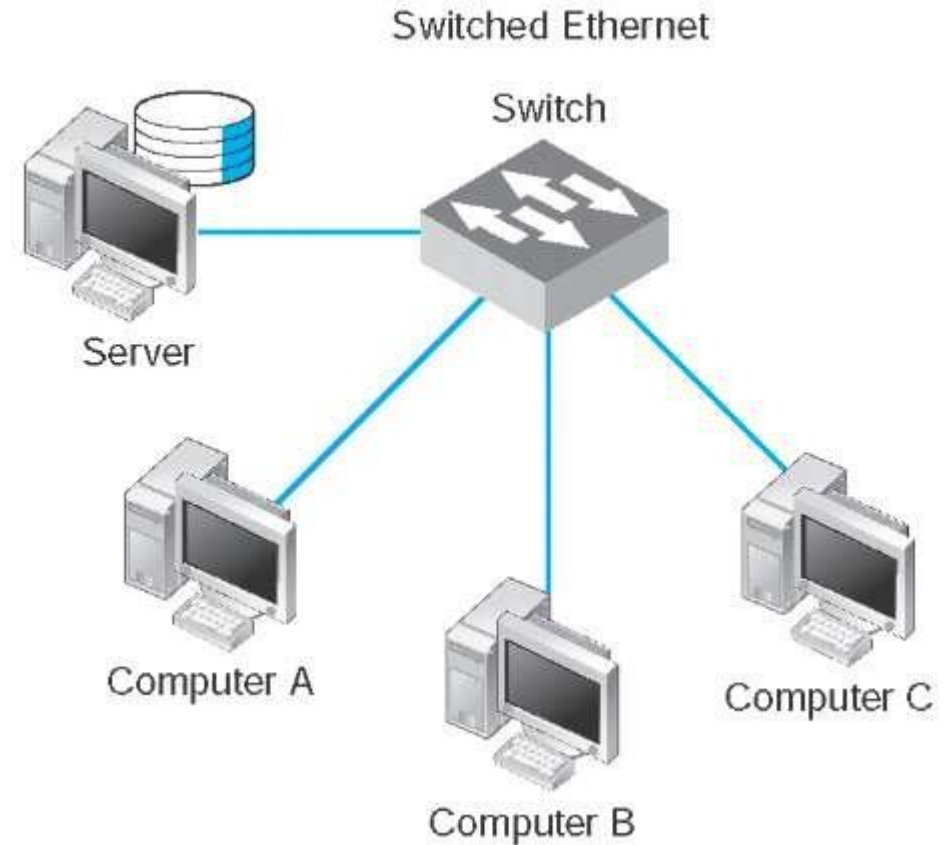
- ❖ 대표적인 LAN 내 통신 프로토콜, IEEE 802.3
- ❖ 비연결형 통신으로 비동기 직렬통신을 사용 -> 단순함과 저비용
- ❖ 부호화는 맨체스터 코드를 사용한다.
- ❖ 전송 매체를 공유하기 위해 CSMA/CD 사용(Half-Duplex) -> 스위치 네트워크에선 사용 안함(Full-Duplex)
- ❖ 최대전송단위(MTU) : 1500Byte(헤더 제외)
- ❖ 전송 속도
 - 10Mbps : Ethernet
 - 100Mbps : Fast Ethernet
 - 1Gbps : Gigabit Ethernet
 - 10Gbps : 10 Gigabit Ethernet

1. Ethernet Frame

1.1 Ethernet



Hub는 대역폭을 연결된 Host끼리 공유(Half-duplex)



Switch는 Host간 세그먼트 분리(Full-duplex)

1. Ethernet Frame

1.2 Ethernet Frame

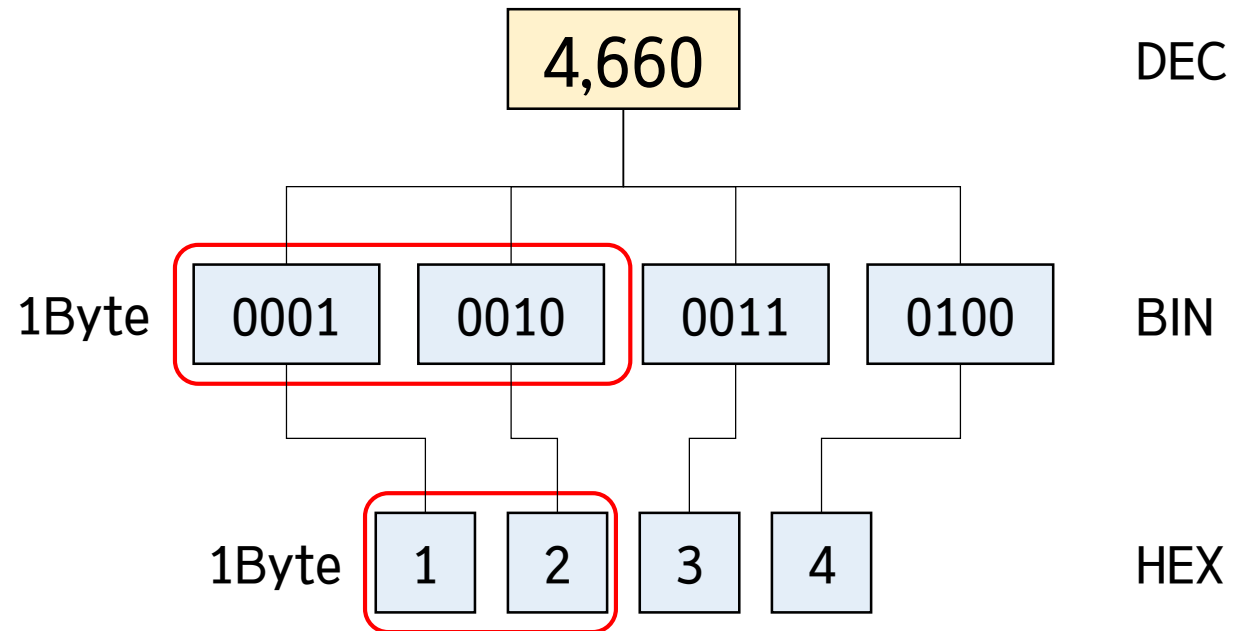
■ Ethernet Frame, IEEE 802.3(DIX 2.0)

- ❖ Preamble : (101010101...) 비트동기를 위해 56비트동안 '1', '0' 반복
- ❖ SFD : (10101011) 프레임 동기를 위한 식별 문자(0xAB)
- ❖ DA / SA : Destination Address, Source Address
- ❖ Len / Type : 0x600 이하 -> Length, 0x600 이상 -> Ethertype(IPv4=0x0800)
- ❖ FCS : 오류 검출을 위한 CRC 코드



1. Ethernet Frame

1.2 Ethernet Frame



1. Ethernet Frame

1.2 Ethernet Frame

- ▶ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
- ▼ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
 - ▶ Destination: 00:00:00_00:00:00 (00:00:00:00:00:00)
 - ▶ Source: 00:00:00_00:00:00 (00:00:00:00:00:00)
 - Type: IPv4 (0x0800)
- ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- ▶ Transmission Control Protocol, Src Port: 53924, Dst Port: 8888, Seq: 0, Len: 0

0000	00 00 00 00 00 00	00 00 00 00 00 00	08 00	45 00E.
0010	00 3c 85 09 40 00	40 06 b7 b0 7f 00	00 01 7f 00		<..@.@.....
0020	00 01 d2 a4 22 b8	7d b9 81 4d 00 00	00 00 a0 02		..."}..M.....
0030	aa aa fe 30 00 00	02 04 ff d7 04 02	08 0a d8 66		...0.....f
0040	8e 15 00 00 00 00	01 03 03 07		

DA : loopback

SA : loopback

Ethertype :IPv4

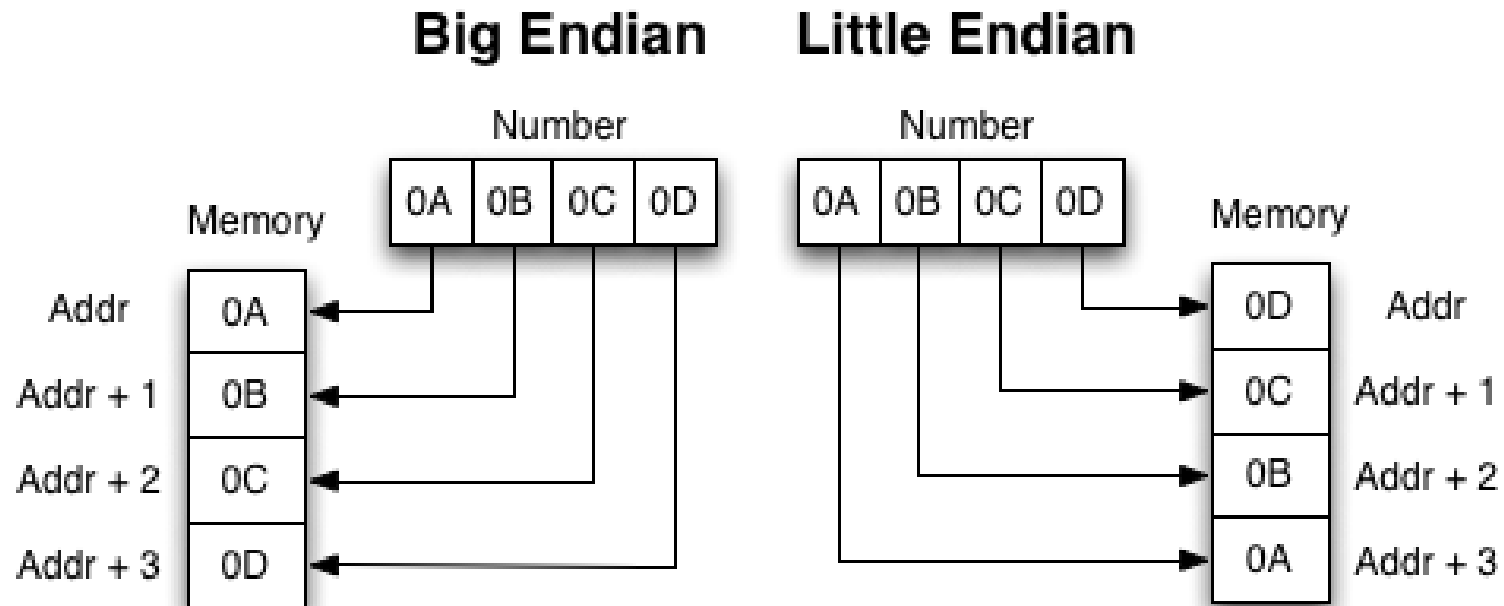
Preamble 및 FCS 등의 데이터는 NIC가 처리한 뒤 드라이버로 전달하지 않는다.

1. Ethernet Frame

1.3 Byte Ordering

Byte Ordering

- ❖ System이 메모리에 데이터를 저장하는 순서
- ❖ 시스템간 저장순서가 다르면 같은 값을 서로 다른 값으로 인식
- ❖ Intel x86, x64 및 AMD 계열 CPU는 Little Endian



1. Ethernet Frame

1.3 Byte Ordering

```
1  #include <stdio.h>
2  #include <ctype.h>
3  #include "dumpcode.h"
4
5  int main(int argc, char *argv[]){
6      char str[] = "Famous Network";
7      int i = 100;
8
9      printf("%s\n", str);
10     dumpcode((unsigned char *)&str, 16);
11
12     printf("%d\n", i);
13     dumpcode((unsigned char *)&i, 4);
14
15     return 0;
16 }
```

1. Ethernet Frame

1.3 Byte Ordering

```
root@kali:~/Downloads# ./a.out
Famous Network
0xbfad5cb1 46 61 6d 6f 75 73 20 4e 65 74 77 6f 72 6b 00 dc Famous Network..
100
0xbfad5cac 64 00 00 00 d...
root@kali:~/Downloads#
```

- string은 1byte 단위로 메모리에 적재되기 때문에 byte order와 무관
- int는 4Byte 단위로 적재, little endian임으로 첫 번째 Byte인 64가 가장 앞에 적재
- a = 0x1234(4660)이면 34(2) 12(1)

1. Ethernet Frame

1.4 Network Byte Order

Network byte order

- ❖ Network byte order는 Big endian([RFC1700](#))
- ❖ Network byte order가 Big endian인 것에 성능상의 이유는 없다.
- ❖ 따라서 각 Host는 네트워크로 데이터 전송 시 Big endian으로 변환하여 전송.
- ❖ 수신하는 Host가 Network byte order를 자신의 Byte order에 맞게 변환.
- ❖ String으로 전송하면 메모리에 1Byte 단위로 적재되기 때문에 Byte order에 영향 받지 않는다.


1. Ethernet Frame




1.4 Network Byte Order

함수명	용도
htons	호스트 byte order를 네트워크 byte order로 변경, 2Byte 크기
htonl	호스트 byte order를 네트워크 byte order로 변경, 4Byte 크기
ntohs	네트워크 byte order를 호스트 byte order로 변경, 2Byte 크기
ntohl	네트워크 byte order를 호스트 byte order로 변경, 4Byte 크기





htons & ntohs





- If your host is little-endian:

ntohs ( ) →  

htons ( ) →  

- If your host is big-endian:

ntohs ( ) →  

htons ( ) →  

1. Ethernet Frame

1.4 Network Byte Order

```
import socket
import sys

print(sys.byteorder) # little

h_s = 0x1234 # 4660
h_l = 0x12345678 # 305419896

print(socket.htons(h_s)) # 34 12 -> 13330
print(socket.htonl(h_l)) # 78 56 34 12 -> 2018915348

n_s = 0x3412
n_l = 0x78563412

print(socket.ntohl(n_s)) # 12 34 -> 1305419896
print(socket.ntohl(n_l)) # 12 34 56 78 -> 305419896
```

1. Ethernet Frame

1.4 Network Byte Order

```
def my_htons(item):  
    left = (a << 8) & 0xff00  
    right = a >> 8  
  
    return right + left
```

0x1234	0001 0010 0011 0100	
	0011 0100 0000 0000	8bit shift left and masking(0xff00)
	+ 0000 0000 0001 0010	8bit shift right
0x3412	0011 0100 0001 0010	

1. Ethernet Frame

1.4 Network Byte Order

socket.htons

==

socket.ntohs

htons(l)과 ntohs(l)은 같은 동작을 수행, 의미적 구분을 위해 이름을 나눔.

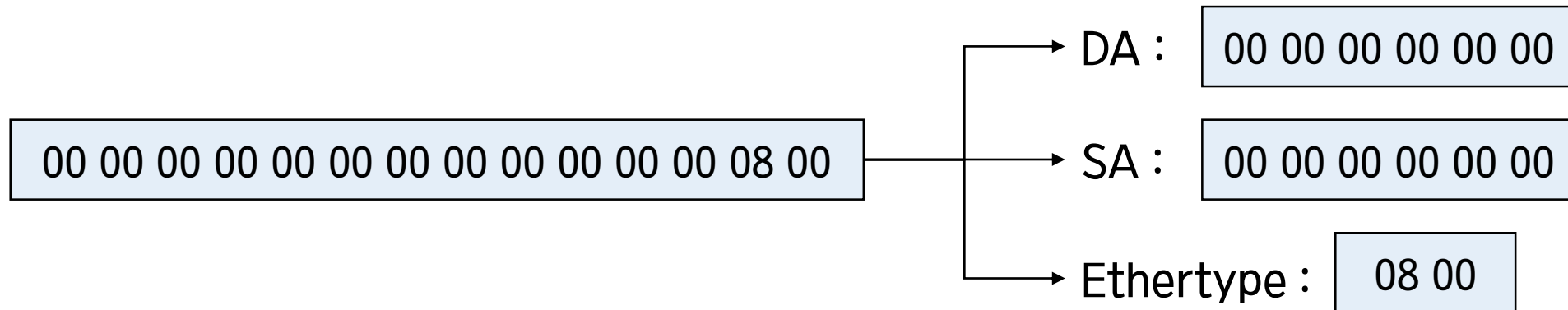
2. Ethernet frame parsing

2. Ethernet Frame parsing

2.1 Frame parsing

■ Frame parsing

- ❖ 캡처한 프레임을 분석하기 좋은 형태로 바꾸는 작업
- ❖ 주로 프로토콜 헤더의 필드별로 파싱해 구조체 및 클래스 형태로 변환
- ❖ string 타입이 아닌 필드는 Byte order를 구분해서 변환(ex, Ethertype)



2. Ethernet Frame parsing

2.2 Struct module

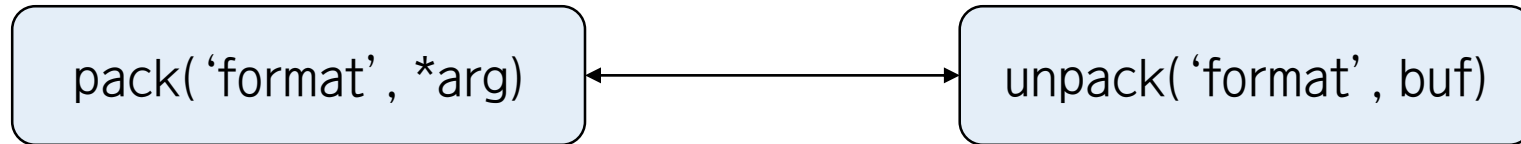
Struct module

- ❖ Encoding된 이진 데이터를 python bytes 데이터로 해석해주는 모듈.
- ❖ <https://docs.python.org/3/library/struct.html>
- ❖ Python Bytes로 표현된 C-구조체와 Python 값 사이의 변환을 수행.
- ❖ 네트워크 등 다른 소스에 저장된 Binary 데이터를 처리할 때 사용
- ❖ 주요 메소드
 - `struct.pack(format, v1, v2, ...)` : format에 맞게 Python 값 v1, v2, ...를 연결한 Bytes를 리턴
 - `struct.unpack(format, buf)` : buf를 format에 맞는 Python 값으로 쪼개 Tuple을 리턴

2. Ethernet Frame parsing

2.2 Struct module

여러 Python Value를 엔디안, 구조체 정렬 등을
고려해 Bytes로 바꿀 때



네트워크 등에서 수신한 Bytes를
엔디안, 구조체 정렬 등을 고려해 Python Value로 바꿀 때

2. Ethernet Frame parsing

2.2 Struct module

Character	Byte order	Size	Alignment
@	native	native	native
=	native	standard	none
<	little-endian	standard	none
>	big-endian	standard	none
!	network (= big-endian)	standard	none

컴파일러가 구조체의 크기를
일정한 크기로 패딩하는 것


```
4  struct MyType1{
5      char a;
6      double b;
7      int c;
8  };
9
10 struct MyType2{
11     char a;
12     int c;
13     double b;
14 };
```

$\text{char}(1) + \text{double}(8) + \text{int}(4) = 13\text{Bytes?}$

$\text{MyType1} : 8 + 8 + 8 = 24\text{Bytes}$

$\text{MyType2} : 8 + 8 + 8 = 16\text{Bytes}$



가장 큰 멤버의 크기를 기준으로 배치된다.
작은 크기의 멤버순으로 선언해야 한다.

2. Ethernet Frame parsing

2.2 Struct module

Format	C Type	Python type	Standard size	Notes
x	pad byte	no value		
c	char	bytes of length 1	1	
b	signed char	integer	1	(1),(3)
B	unsigned char	integer	1	(3)
?	_Bool	bool	1	(1)
h	short	integer	2	(3)
H	unsigned short	integer	2	(3)
i	int	integer	4	(3)
I	unsigned int	integer	4	(3)

2. Ethernet Frame parsing

2.2 Struct module

L	unsigned long	integer	4	(3)
q	long long	integer	8	(2), (3)
Q	unsigned long long	integer	8	(2), (3)
n	ssize_t	integer		(4)
N	size_t	integer		(4)
e	(7)	float	2	(5)
f	float	float	4	(5)
d	double	float	8	(5)
s	char []	bytes		
p	char []	bytes		
P	void *	integer		(6)

2. Ethernet Frame parsing

2.3 Ethernet header parsing

■ Ethernet header parsing

- ❖ DA : 6Byte, 1Byte 원소 6개의 배열 -> 각 원소는 1Byte임으로 endian 변환 X
- ❖ SA : 6Byte, 1Byte 원소 6개의 배열 -> 각 원소는 1Byte임으로 endian 변환 X
- ❖ Ether_type : 2Byte, unsigned short -> 2Byte를 해석해야 하기 때문에 endian 변환 필요

DA : 00 00 00 00 00 00

SA : 00 00 00 00 00 00

Ethertype : 08 00

```
ether = struct.unpack('!6B6BH', raw_data)
```

! : Network byte order

6B : unsigned char(1Byte) * 6

H : unsigned short(2Byte)

2. Ethernet Frame parsing

2.3 Ethernet header parsing

```
def make_ethernet_header(raw_data):  
    → ether = struct.unpack('!6B6BH', raw_data)  
    → return { 'dst': '%02x:%02x:%02x:%02x:%02x:%02x' % ether[:6],  
               'src': '%02x:%02x:%02x:%02x:%02x:%02x' % ether[6:12],  
               'ether_type': ether[12] }
```

```
ether : (0, 80, 86, 253, 7, 92, 0, 12, 41, 68, 94, 27, 2048)  
dst : 00:50:56:fd:07:5c  
src : 00:0c:29:44:5e:1b  
ether_type : 2048
```

단, MAC 주소처럼 내부적으로 비교연산 등에 자주 사용되는 값은 string으로 바뀌어서 저장할 필요가 없다.

```
ETH_P_ALL = 0x0003
```

```
ETH_SIZE = 14
```

```
def make_ethernet_header(raw_data):
```

```
    ether = struct.unpack('!6B6BH', raw_data)
```

```
    return {'dst': '%02x:%02x:%02x:%02x:%02x:%02x' % ether[:6],
```

```
            'src': '%02x:%02x:%02x:%02x:%02x:%02x' % ether[6:12],
```

```
            'ether_type': ether[12]}
```

```
def sniffing(nic):
```

```
    if os.name == 'nt':
```

```
        address_family = socket.AF_INET
```

```
        protocol_type = socket.IPPROTO_IP
```

```
    else:
```

```
        address_family = socket.AF_PACKET
```

```
        protocol_type = socket.ntohs(ETH_P_ALL)
```

```
    with socket.socket(address_family, socket.SOCK_RAW, protocol_type) as sniffe_sock:
```

```
        sniffe_sock.bind((nic, 0))
```

```
        if os.name == 'nt':
```

```
            sniffe_sock.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)
```

```
            sniffe_sock.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)
```

```
            data, _ = sniffe_sock.recvfrom(65535)
```

```
            ethernet_header = make_ethernet_header(data[:ETH_SIZE])
```

```
            for item in ethernet_header.items():
```

```
                print('{0} : {1}'.format(item[0], item[1]))
```

```
            if os.name == 'nt':
```

```
                sniffe_sock.ioctl(socket.SIO_RCVALL, socket.RCVALL_OFF)
```

2. Ethernet Frame parsing

2.3 Ethernet header parsing

```
root@ubuntu:/home/famous/Desktop/TA/socket/assignment/assignment_9# python3 ether_sniffer.py -i ens33
dst : 00:50:56:fd:07:5c
src : 00:0c:29:44:5e:1b
ether_type : 2048
root@ubuntu:/home/famous/Desktop/TA/socket/assignment/assignment_9#
```

```
root@ubuntu:/home/famous/Desktop/TA/socket/assignment/assignment_9# python3 ether_sniffer.py -i lo
dst : 00:00:00:00:00:00
src : 00:00:00:00:00:00
ether_type : 2048
root@ubuntu:/home/famous/Desktop/TA/socket/assignment/assignment_9#
```

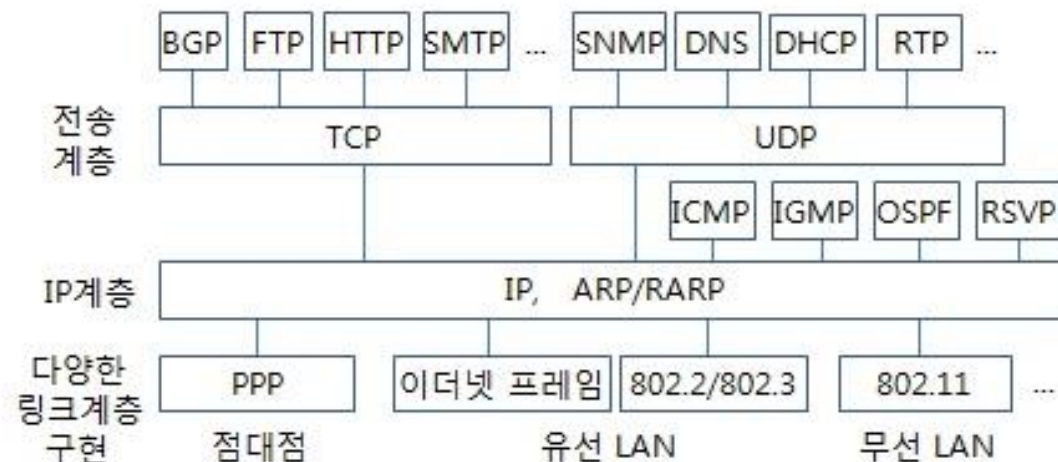

3. IP packet parsing

3. IP packet parsing

3.1 IP

■ IP(Internet Protocol)

- ❖ TCP/IP 패킷 스위칭 네트워크에서 서로 다른 호스트가 정보를 주고 받는데 사용하는 프로토콜
- ❖ TCP/IP 네트워크에서 주소지정(Addressing) 및 라우팅(Routing)을 담당
- ❖ Best-Effort Service : 오류제어 및 흐름제어를 수행하지 않는다.
- ❖ Connectionless : 비연결형 프로토콜, Unreliable : 소실, 중복, 지연, 순서 바뀔 수 있다.
- ❖ TCP, UDP, ICMP, IGMP 등 다양한 프로토콜이 IP 프로토콜을 통해 전송된다.



3. IP packet parsing

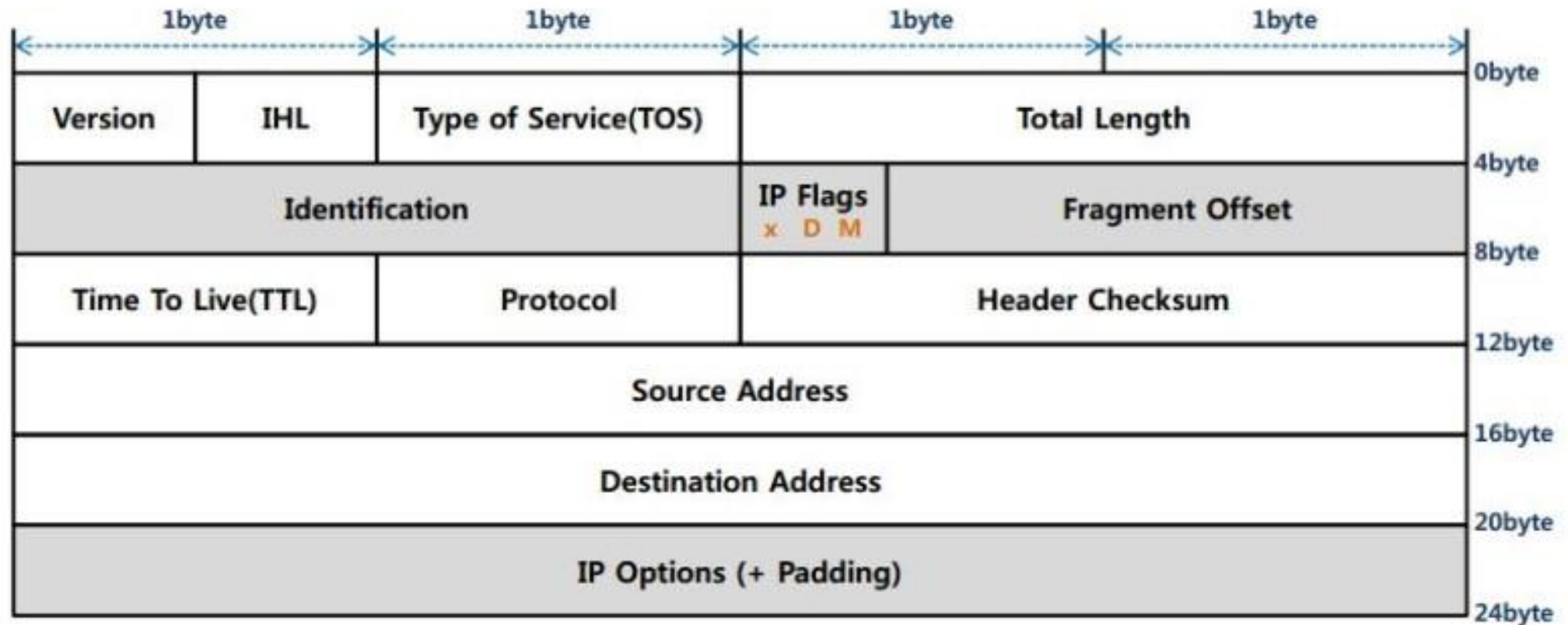
3.2 IPv4 header

■ IPv4 header field

- ❖ Version(4) : 현재는 IPv4임으로 4
- ❖ Header Length(4) : 워드 단위로 표기된 IPv4 헤더의 길이, 최소값 $5(4 * 5 = 20)$
- ❖ Type of Service(8) : QoS 등을 위한 필드, 현재 사용 X
- ❖ Total Packet Length(16) : Byte 단위로 표기된 헤더 및 페이로드를 포함한 전체 IP 패킷의 길이
- ❖ Fragment Identifier(16) : 단편화 일련번호
- ❖ Fragmentation Flag(3) : 단편화 옵션
- ❖ Fragmentation Offset(13) : 단편화 패킷 내 오프셋
- ❖ TTL(Time To Live, 8) : IP 패킷의 수명, 홉마다 1씩 감소
- ❖ Protocol Identifier(8) : 어느 상위계층 프로토콜이 IP 패킷에 들어있는지 나타냄
- ❖ Checksum(16) : 헤더 체크섬
- ❖ Source / Destination Address(32) : 송신지, 수신지 IP 주소
- ❖ IP 옵션 및 패딩 -> 거의 사용X

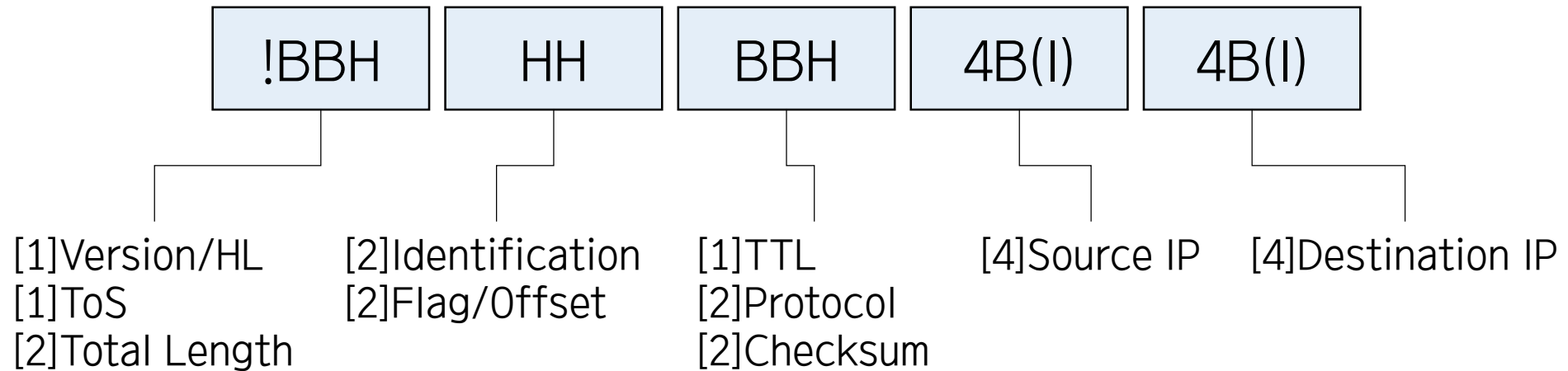
3. IP packet parsing

3.2 IPv4 header



3. IP packet parsing

3.3 IPv4 header parsing



3. IP packet parsing

3.4 Assignment 9

Assignment #9

- Linux에서 IP Packet을 수신해 Ethernet 헤더, IP 헤더, 페이로드 를 출력하는 프로그램 작성.
 - AF_PACKET을 사용하고 PROTOCOL_TYPE은 ETH_P_ALL을 사용.
 - Ethernet 헤더 파싱 후 Ether_type을 통해 IP 패킷인지 검사 후 IP 패킷일 때만 출력
 - IP 헤더는 헤더의 길이를 먼저 구한 뒤 옵션을 제외한 길이에 맞게 파싱
 - While 루프를 통해 여러 번 동작하도록 작성
 - 프로그램 실행 뒤 google.com에 PING을 1번 보낸 결과를 캡처해 첨부
- 팀 대표가 barcel@naver.com으로 제출 (5.14일까지)
 - Title : [컴퓨터네트워크][학번][이름][과제_N]
 - Content : github repo url

팀명 : 길동이네

팀원 : 홍길동(학번), 고길동(학번)

3. IP packet parsing

3.4 Assignment 9

```
[2] IP_PACKET-----  
  
Ethernet Header  
[dst] 00:0c:29:44:5e:1b  
[src] 00:50:56:fd:07:5c  
[ether_type] 2048  
  
IP HEADER  
[version] 4  
[header_length] 5  
[tos] 0  
[total_length] 84  
[id] 20301  
[flag] 0  
[offset] 0  
[ttl] 128  
[protocol] 1  
[checksum] 21019  
[src] 8.8.8.8  
[dst] 192.168.200.136  
  
Raw Data  
offset 00 01 02 03 04 05 06 07 - 08 09 0a 0b 0c 0d 0e 0f  
0x0000 00 0c 29 44 5e 1b 00 50 - 56 fd 07 5c 08 00 45 00  
0x0010 00 54 4f 4d 00 00 80 01 - 52 1b 08 08 08 08 c0 a8  
0x0020 c8 88 00 00 61 cd 21 e0 - 00 01 fb 44 c5 5c 00 00  
0x0030 00 00 f6 dc 06 00 00 00 - 00 00 10 11 12 13 14 15  
0x0040 16 17 18 19 1a 1b 1c 1d - 1e 1f 20 21 22 23 24 25  
0x0050 26 27 28 29 2a 2b 2c 2d - 2e 2f 30 31 32 33 34 35  
0x0060 36 37
```