Packet Sniffing with RAW Socket

유명성

1. Packet Sniffing

Packet

❖ Packet은 네트워크에서 한 번에 전송되는 데이터의 전송 단위이다.

❖ 데이터를 쪼개서 전송하는 이유는 전송 매체를 더 효율적으로 사용하기 위해서이다.

❖ 100MByte의 데이터를 한 번에 전송하면 오류가 발생할 확률이 크고, 오류 발생 시 100MByte를 처음부터 다시 받아야한다.

❖ Ethernet과 같이 전송 매체를 공유하는 환경에서 이처럼 매체를 오랜 시간 독점해서 사용하는 것은 전체 네트워크의 성능에 악영향을 끼칠 수 있다.

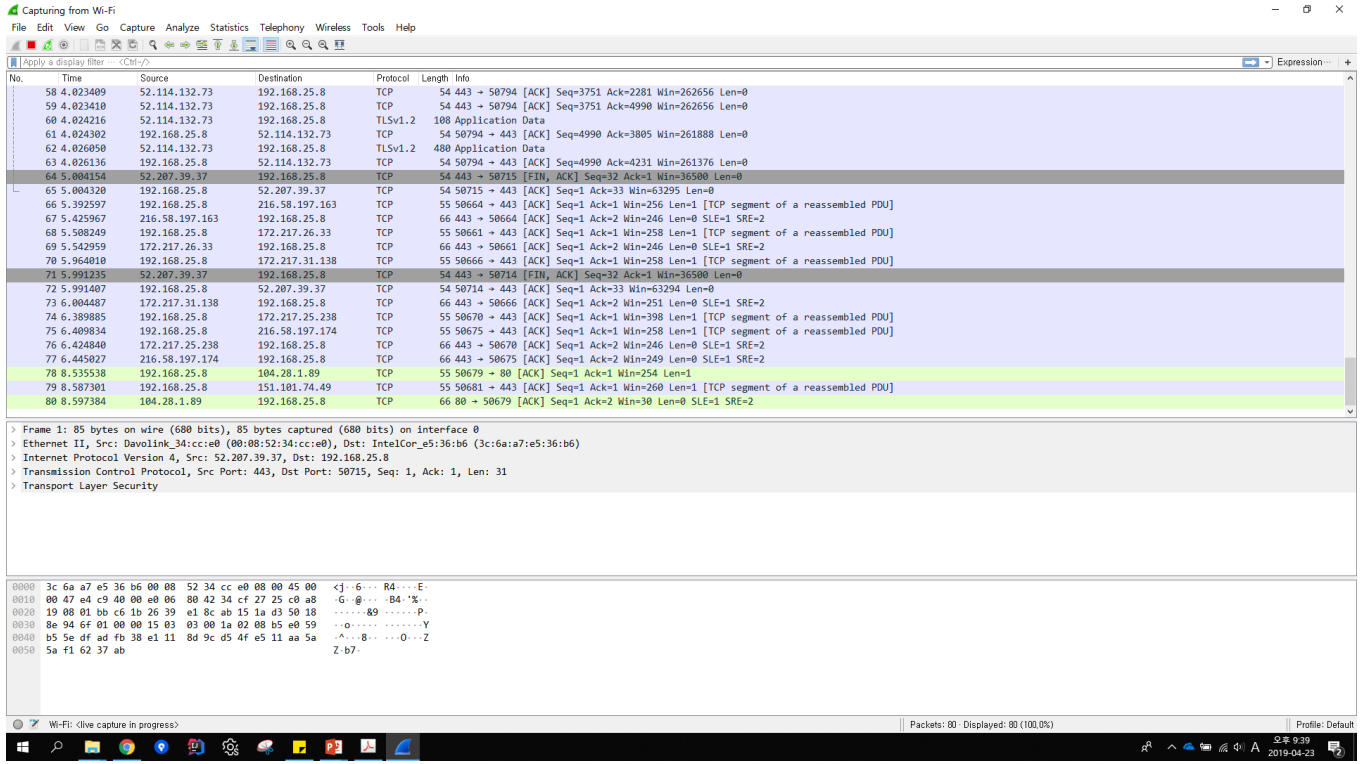
❖ 프로토콜별 데이터 전송 단위
- L2 : Frame
- L3 : Packet
- L4 : Segment(TCP), Message(UDP)

Packet Sniffing

- 패킷 스니핑은 패킷 캡쳐, 패킷 분석으로도 불리며 네트워크 상에서 발생하는 일을 이해하기 위해 네트워크를 통해 전달되는 패킷을 수집하고 분석하는 행위이다.
- 주로 대역폭 분석 등 네트워크 상태를 분석을 위해 사용되며, 새로운 프로토콜 개발에도 쓰인다.
- 보안상 악의적인 공격을 탐지하고 공격자를 추적하기 위해서도 사용된다.
- 대표적인 패킷 스니핑 툴에 tcpdump, wireshark, kismet 등이 있다.

Wireshark

- ❖ 오픈소스 GUI 패킷 스니핑 프로그램
- ❖ 내부적으로 Libpcap을 사용하며 GUI는 Qt로 작성되어 있어 Windows, Linux, MAC OS 등 다양한 운영체제에서 동작할 수 있다.(크로스 플랫폼)



pcap

❖ OS에서 패킷을 캡처할 수 있게 해주는 C 라이브러리로 tcpdump.org에서 제작하였다.

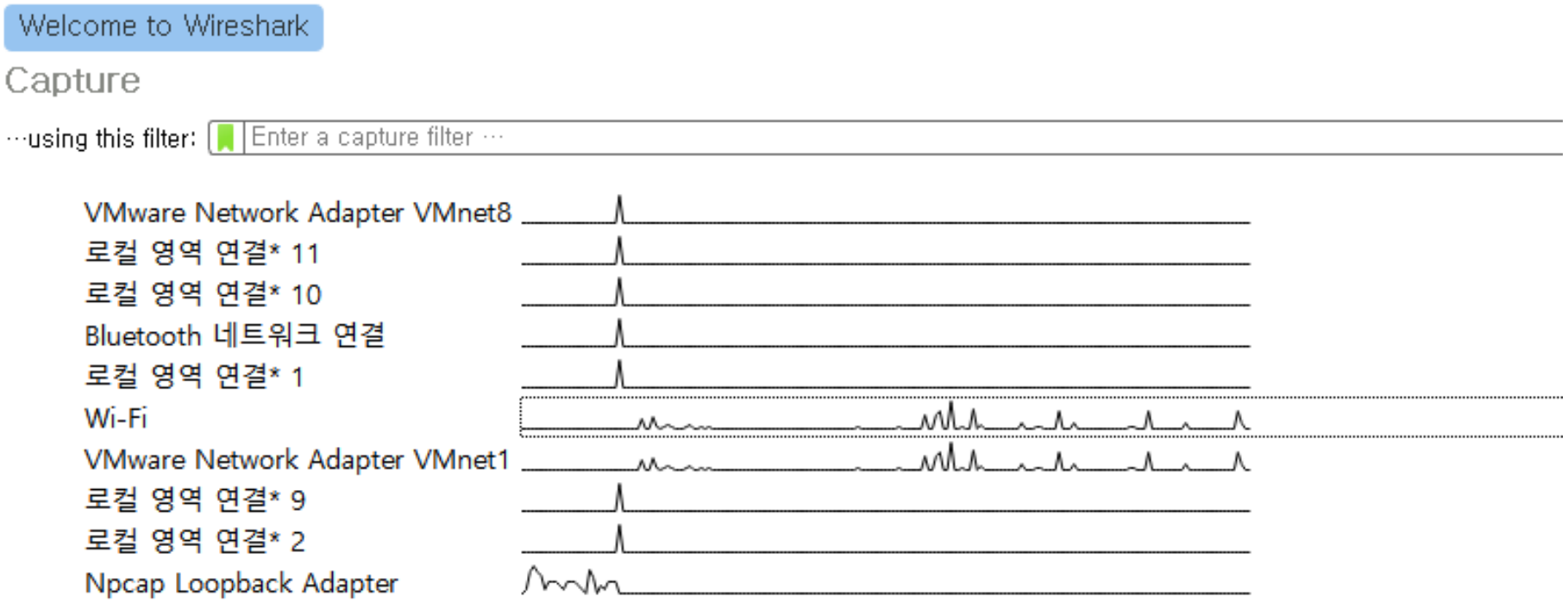
❖ Libpcap(Linux), Winpcap(Windows), Npcap(Windows) 등 다양한 버전이 존재한다.

❖ 다양한 언어에서 pcap을 wrapping한 라이브러리를 지원한다.

- C++ : Libtins
- Python : python-libpcap, Pcapy, WinPcapy, scapy
- Java : jpcap, jNetPcap

1. Packet Sniffing

1.5 packet capture



원하는 인터페이스를 선택하여 패킷을 캡처할 수 있다.

1. Packet Sniffing

1.5 packet capture

캡처된 패킷 목록

패킷 프로토콜 및 필드 정보

Raw Data

Apply a display filter ··· <Ctrl-/>

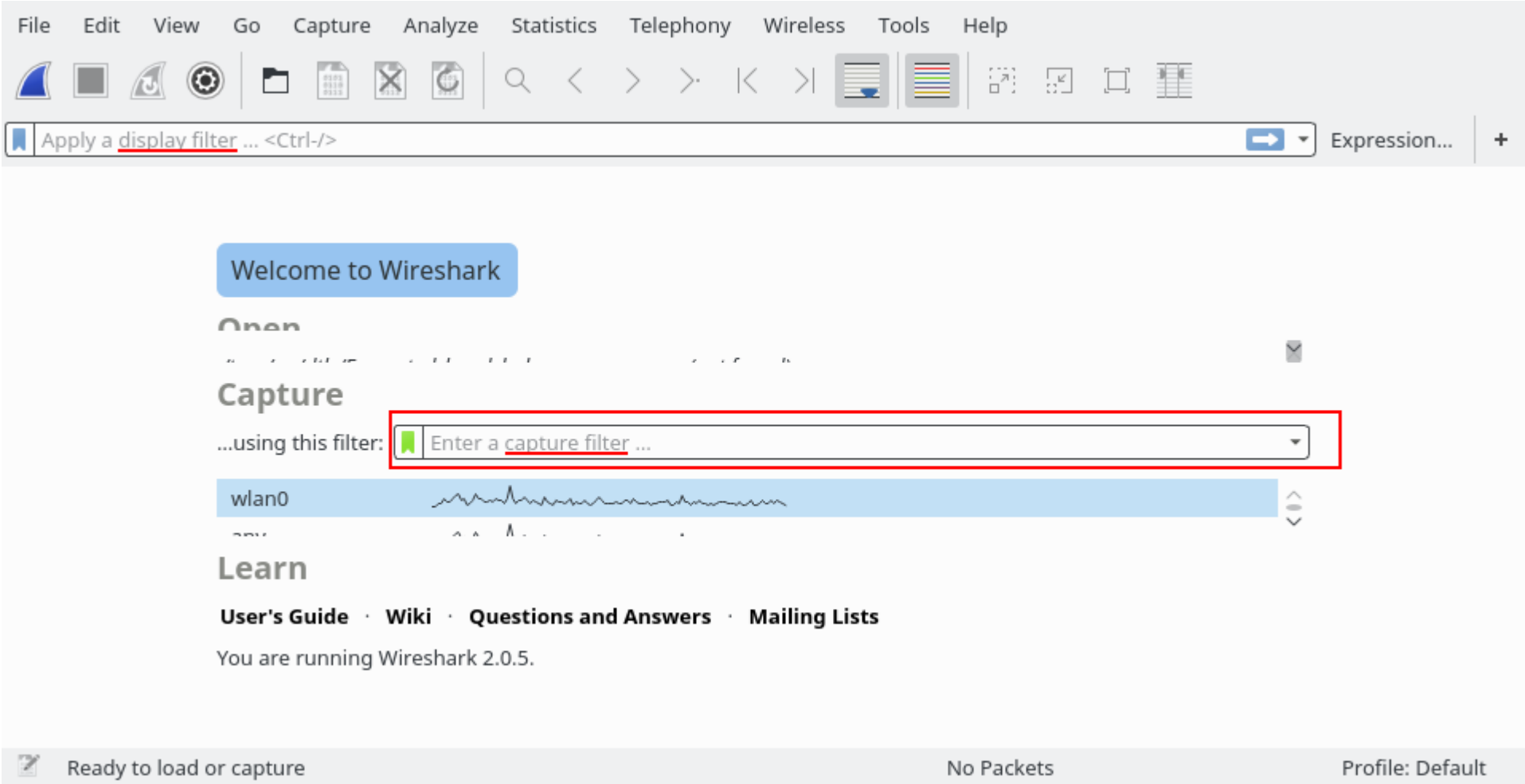
No.	Time	Source	Destination	Protocol	Length	Info
753	9.741544	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=108580 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
754	9.741544	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=110020 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
755	9.741545	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=111460 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
756	9.743186	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=50980 Win=525568 Len=0
757	9.743187	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=52420 Win=524032 Len=0
758	9.743187	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=61060 Win=525568 Len=0
759	9.743232	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=112900 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
760	9.743233	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=114340 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
761	9.743235	192.168.25.8	13.107.136.9	TLSv1.2	1494	Application Data [TCP segment of a reassembled PDU]
762	9.743235	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=117220 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
763	9.743235	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=118660 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
764	9.743236	192.168.25.8	13.107.136.9	TLSv1.2	961	Application Data
765	9.745347	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=62500 Win=524032 Len=0
766	9.745347	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=65380 Win=525568 Len=0
767	9.745347	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=66820 Win=524032 Len=0
768	9.745347	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=68260 Win=525568 Len=0
769	9.745348	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=69700 Win=524032 Len=0
770	9.745348	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=71140 Win=525568 Len=0
771	9.745348	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=74020 Win=525568 Len=0
772	9.745348	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=75460 Win=524032 Len=0
773	9.745349	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=79780 Win=525568 Len=0
774	9.746043	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=81220 Win=524032 Len=0
775	9.746043	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=82660 Win=525568 Len=0
776	9.746043	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=84100 Win=524032 Len=0

```
> Frame 769: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
> Ethernet II, Src: Davolink_34:cc:e0 (00:08:52:34:cc:e0), Dst: IntelCor_e5:36:b6 (3c:6a:a7:e5:36:b6)
> Internet Protocol Version 4, Src: 13.107.136.9, Dst: 192.168.25.8
> Transmission Control Protocol, Src Port: 443, Dst Port: 53981, Seq: 5987, Ack: 69700, Len: 0
```

```
0000  3c 6a a7 e5 36 b6 00 08  52 34 cc e0 08 00 45 00   <j··6··· R4····E·
0010  00 28 77 10 40 00 78 06  1c 9b 0d 6b 88 09 c0 a8   ·(w·@·x· ···k····
0020  19 08 01 bb d2 dd 25 a2  01 de d4 d2 de e4 50 10   ======%· ======P·
0030  07 ff 88 e0 00 00                                  ======
```

1. Packet Sniffing

1.5 packet capture



1. Packet Sniffing

1.5 packet capture

Examples

Capture only traffic to or from IP address 172.18.5.4:

```
host 172.18.5.4
```

Capture traffic to or from a range of IP addresses:

```
net 192.168.0.0/24
```

or

```
net 192.168.0.0 mask 255.255.255.0
```

Capture traffic from a range of IP addresses:

```
src net 192.168.0.0/24
```

or

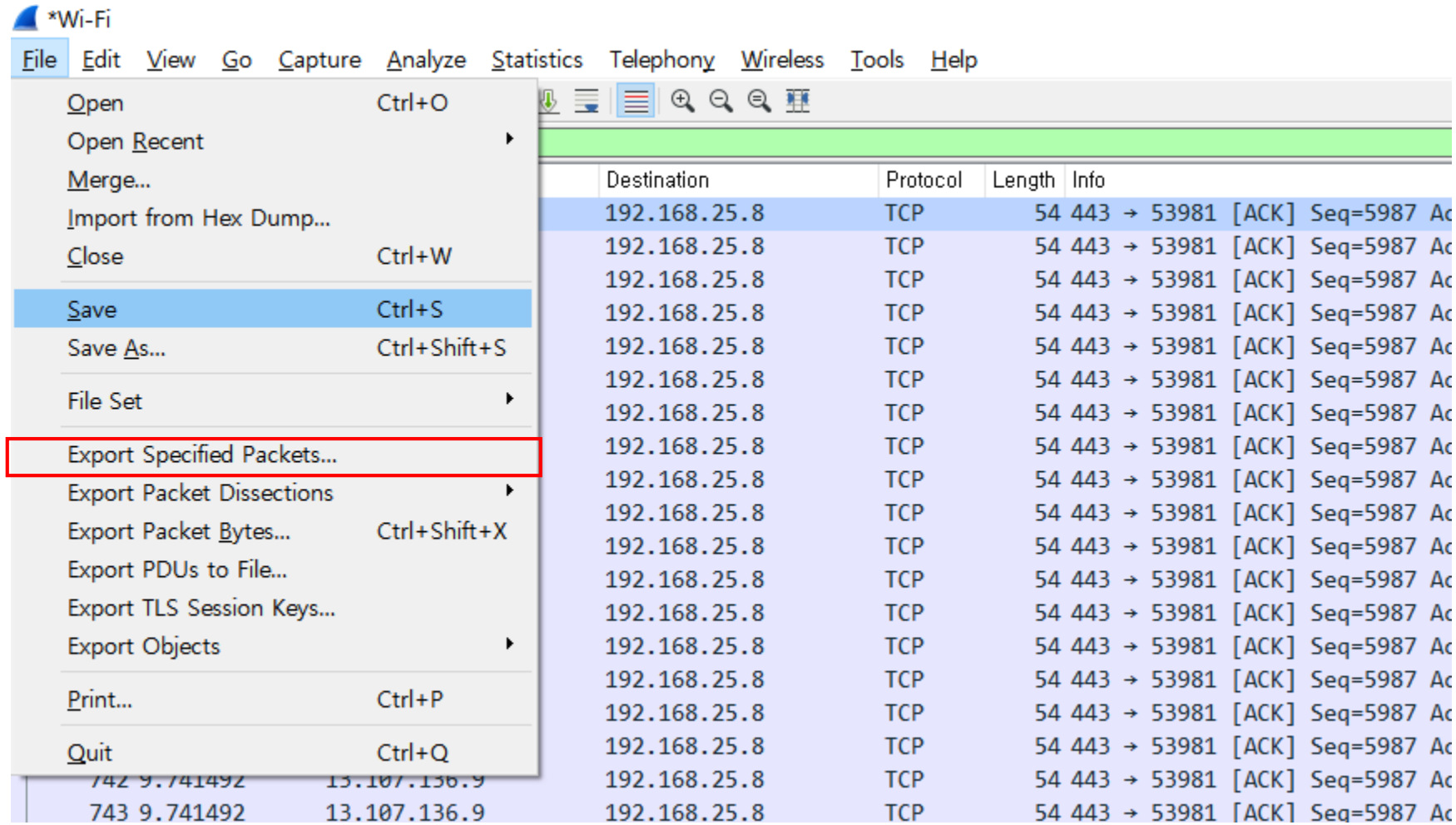
```
src net 192.168.0.0 mask 255.255.255.0
```

Libpcap의 pcap_filter를 이용해 원하는 패킷만 필터링할 수 있다.
http://www.tcpdump.org/manpages/pcap-filter.7.html

1. Packet Sniffing

1.5 packet capture

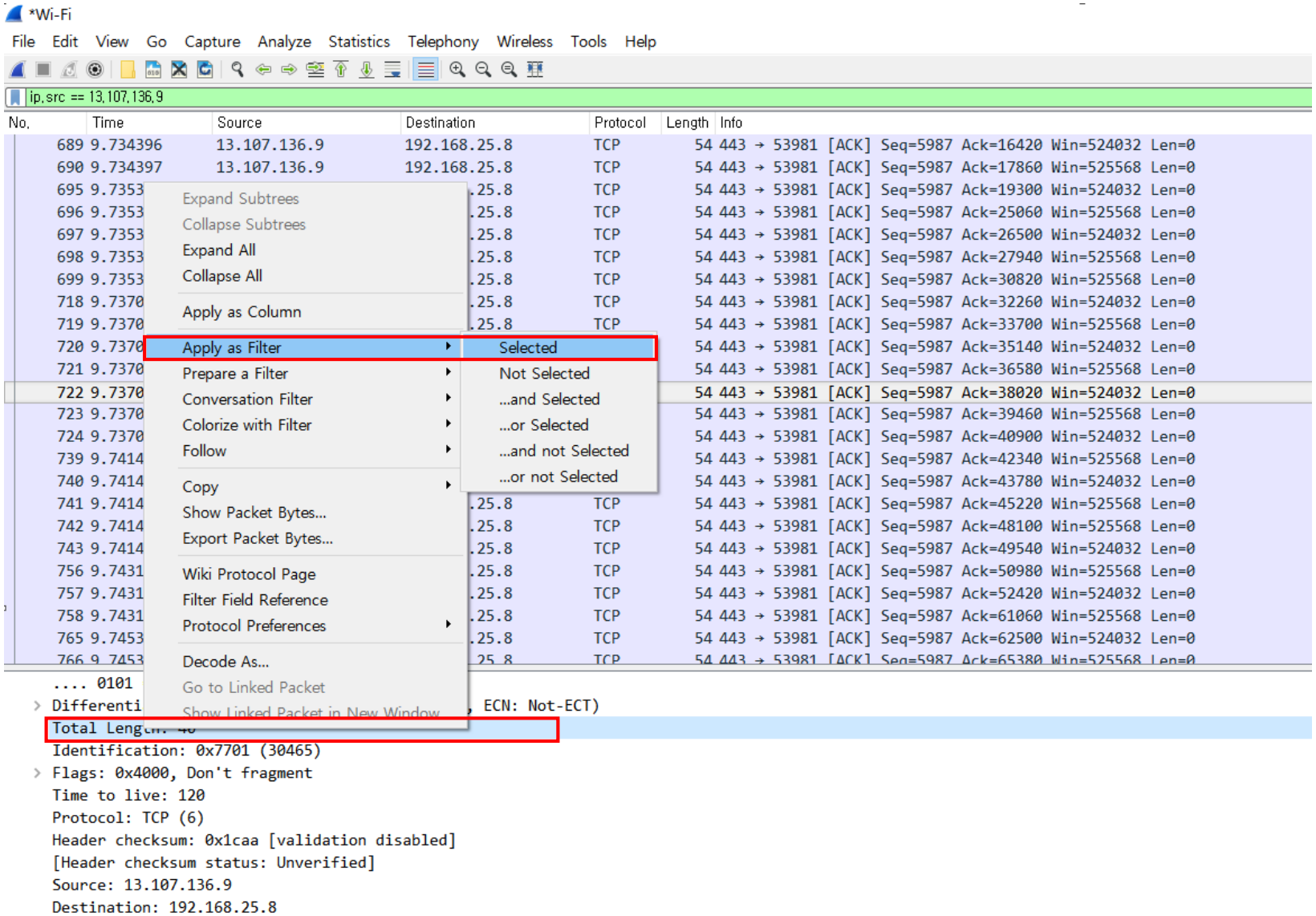
필터링한 패킷만 저장



1. Packet Sniffing

1.5 packet capture

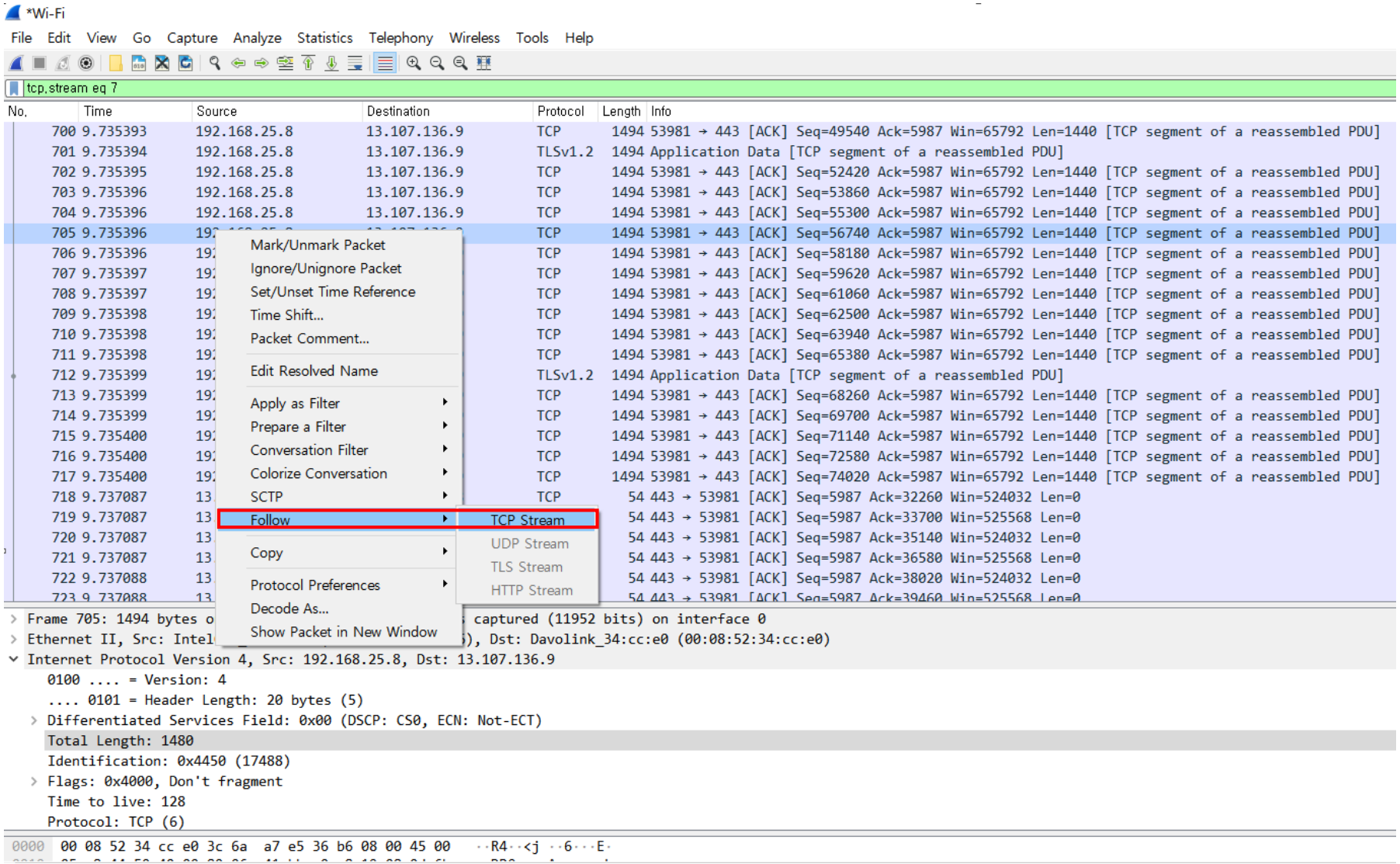
특정 필드 선택 후
필터 지정 가능



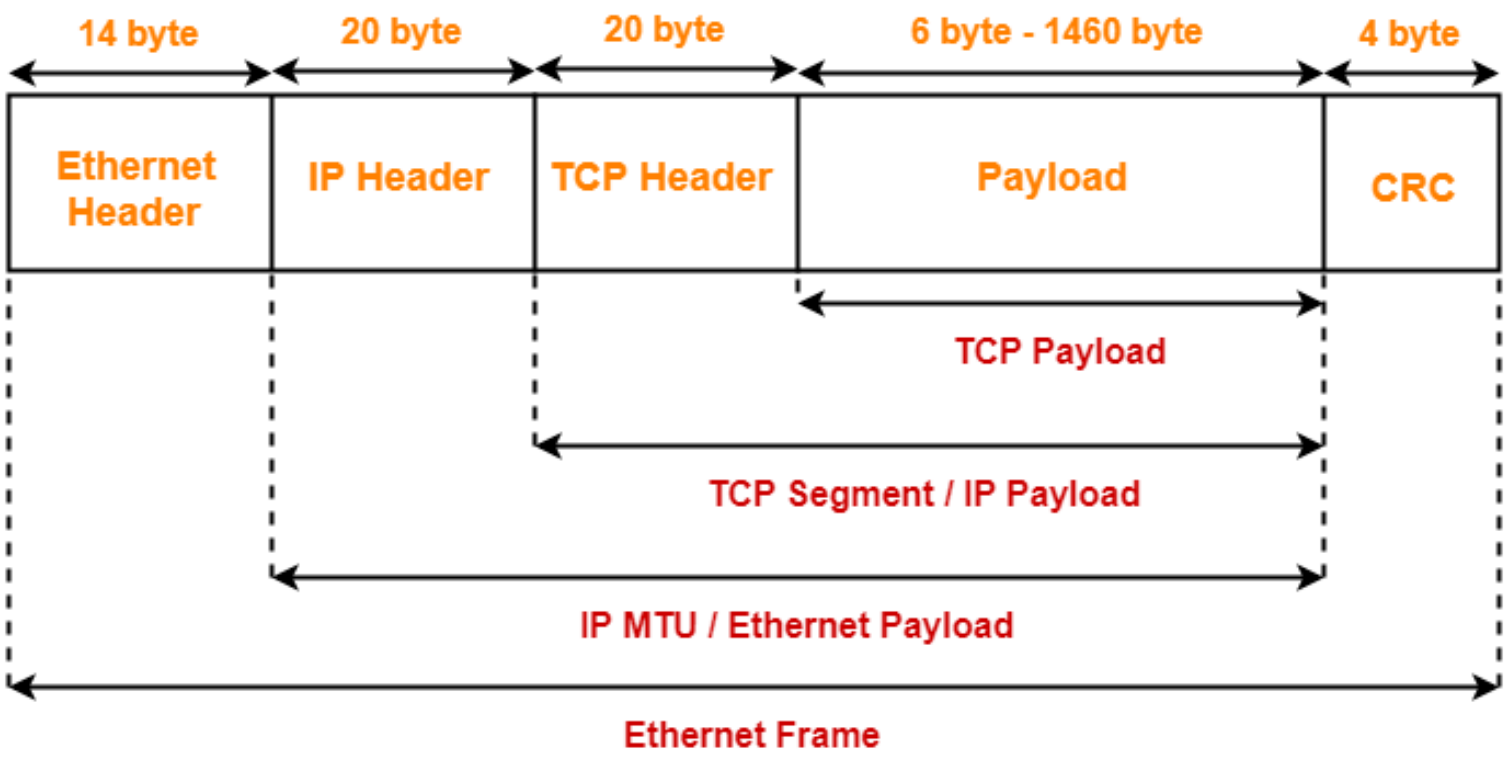
1. Packet Sniffing

1.5 packet capture

특정 TCP/UDP 스트림만 출력할 수 도 있다.



TCP/IP Packet 구조

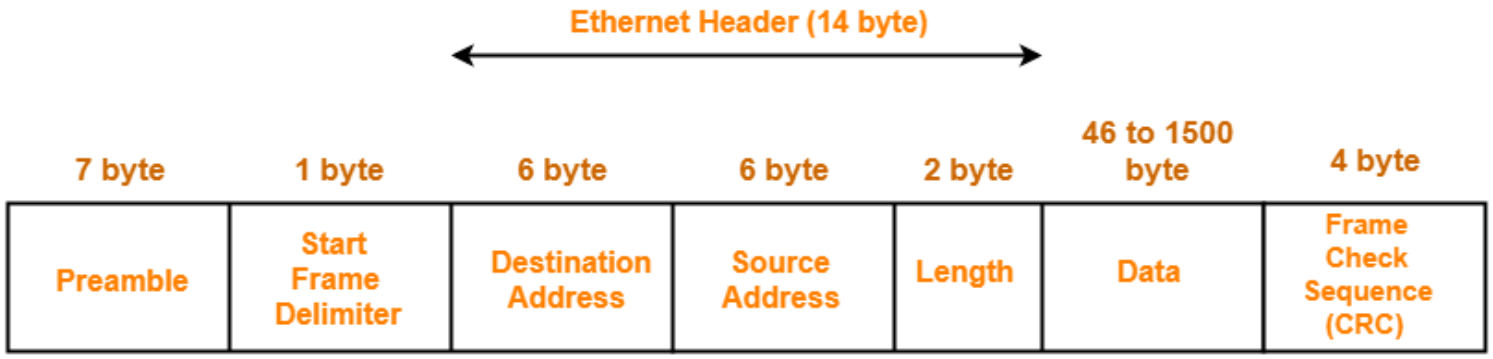


1. Packet Sniffing

1.6 Ethernet sniffing

```
> Frame 10: 55 bytes on wire (440 bits), 55 bytes captured (440 bits) on interface 0
v Ethernet II, Src: IntelCor_e5:36:b6 (3c:6a:a7:e5:36:b6), Dst: Davolink_34:cc:e0 (00:08:52:34:cc:e0)
  > Destination: Davolink_34:cc:e0 (00:08:52:34:cc:e0)
  > Source: IntelCor_e5:36:b6 (3c:6a:a7:e5:36:b6)
    Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 192.168.25.8, Dst: 108.177.125.188
> Transmission Control Protocol, Src Port: 58449, Dst Port: 5228, Seq: 1, Ack: 1, Len: 1
> Data (1 byte)
```

```
0000  00 08 52 34 cc e0 3c 6a  a7 e5 36 b6 08 00 45 00   ··R4··<j ··6···E·
0010  00 29 05 bf 40 00 80 06  30 f2 c0 a8 19 08 6c b1   ·)··@··· 0·····l·
0020  7d bc e4 51 14 6c 21 dd  93 08 a5 0b 71 aa 50 10   }··Q·l!· ····q·P·
0030  01 02 26 5a 00 00 00                               ··&Z···
```



IEEE 802.3 Ethernet Frame Format

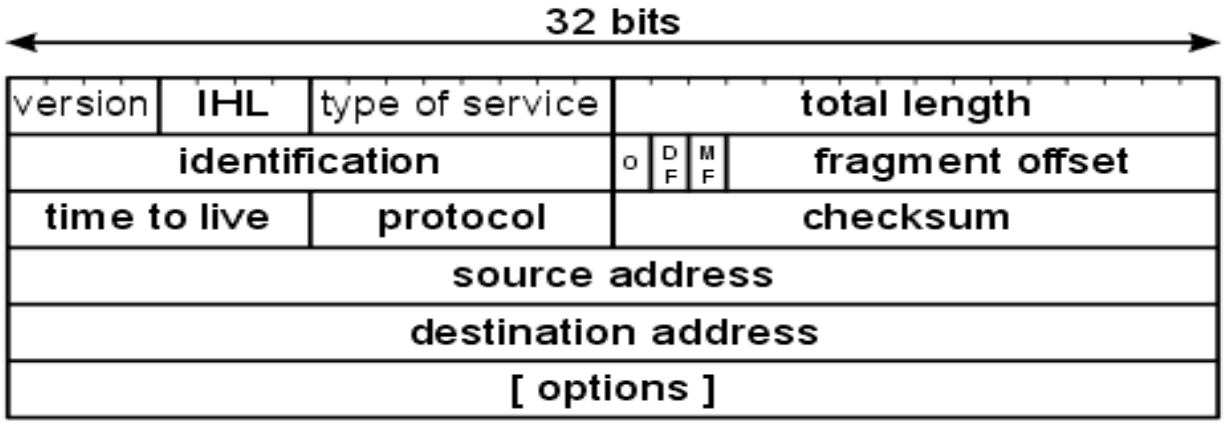
1. Packet Sniffing

1.7 IP sniffing

```
> Frame 11847: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
> Ethernet II, Src: IntelCor_e5:36:b6 (3c:6a:a7:e5:36:b6), Dst: Davolink_34:cc:e0 (00:08:52:34:cc:e0)
v Internet Protocol Version 4, Src: 192.168.25.8, Dst: 13.107.136.9
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 40
    Identification: 0x005b (91)
  > Flags: 0x4000, Don't fragment
    Time to live: 128
    Protocol: TCP (6)

0000  00 08 52 34 cc e0 3c 6a  a7 e5 36 b6 08 00 45 00   ··R4··<j ··6···E·
0010  00 28 00 5b 40 00 80 06  8b 50 c0 a8 19 08 0d 6b   ·(·[@··· ·P·····k
0020  88 09 ed ec 01 bb 06 ab  c0 1e c2 ab 4b 5f 50 10   ······ ····K_P·
0030  01 02 7b 31 00 00                                  ··{1··
```

IP header format



1. Packet Sniffing

1.8 HTTP sniffing

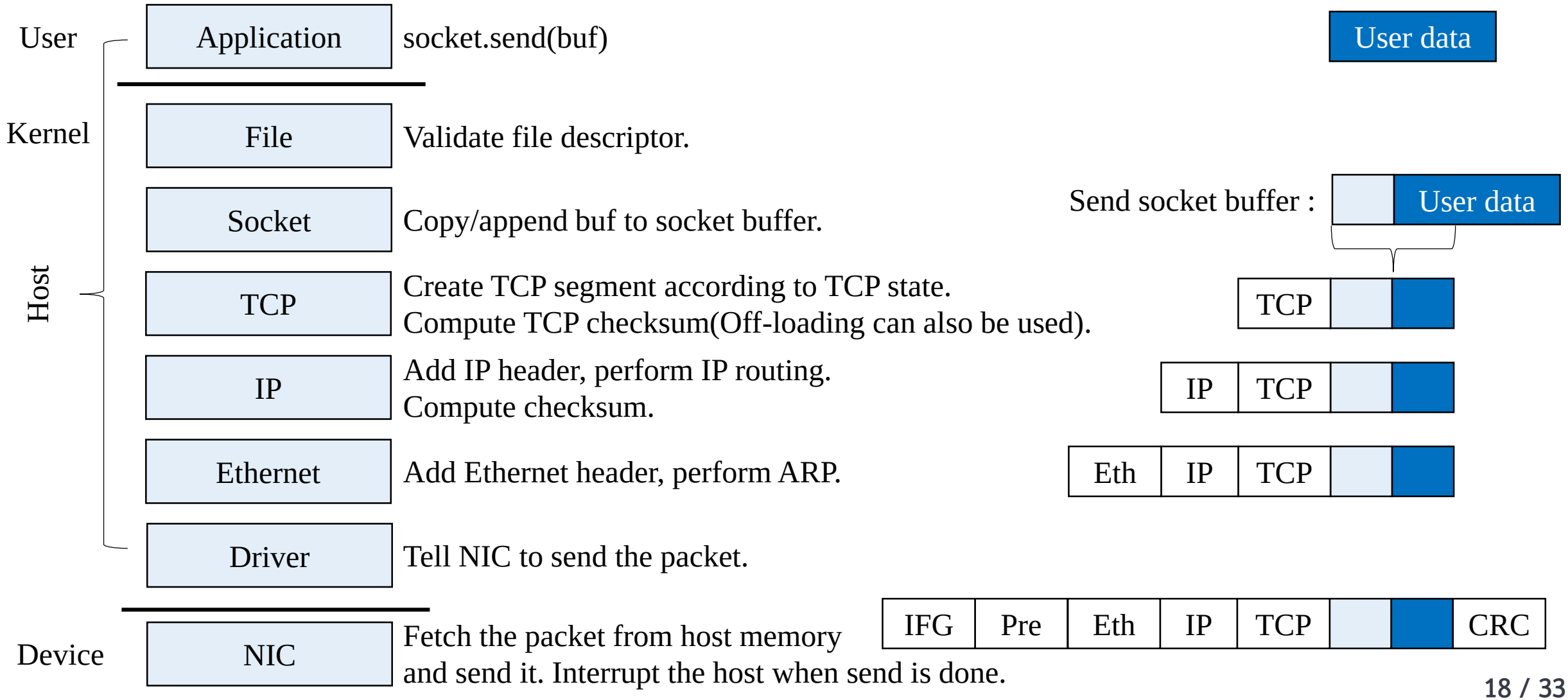
```
> Frame 475: 529 bytes on wire (4232 bits), 529 bytes captured (4232 bits) on interface 0
> Ethernet II, Src: IntelCor_e5:36:b6 (3c:6a:a7:e5:36:b6), Dst: Davolink_34:cc:e0 (00:08:52:34:cc:e0)
> Internet Protocol Version 4, Src: 192.168.25.8, Dst: 175.213.35.39
> Transmission Control Protocol, Src Port: 61332, Dst Port: 80, Seq: 1, Ack: 1, Len: 475
v Hypertext Transfer Protocol
  > GET / HTTP/1.1\r\n
    Host: gilgil.net\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    DNT: 1\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.103 Safari/537.36\r\n
```

```
0030  01 00 83 0b 00 00 47 45  54 20 2f 20 48 54 54 50   ......GE T / HTTP
0040  2f 31 2e 31 0d 0a 48 6f  73 74 3a 20 67 69 6c 67   /1.1··Ho st: gilg
0050  69 6c 2e 6e 65 74 0d 0a  43 6f 6e 6e 65 63 74 69   il.net·· Connecti
0060  6f 6e 3a 20 6b 65 65 70  2d 61 6c 69 76 65 0d 0a   on: keep -alive··
0070  55 70 67 72 61 64 65 2d  49 6e 73 65 63 75 72 65   Upgrade- Insecure
0080  2d 52 65 71 75 65 73 74  73 3a 20 31 0d 0a 44 4e   -Request s: 1··DN
0090  54 3a 20 31 0d 0a 55 73  65 72 2d 41 67 65 6e 74   T: 1··Us er-Agent
00a0  3a 20 4d 6f 7a 69 6c 6c  61 2f 35 2e 30 20 28 57   : Mozill a/5.0 (W
00b0  69 6e 64 6f 77 73 20 4e  54 20 31 30 2e 30 3b 20   indows N T 10.0;
00c0  57 69 6e 36 34 3b 20 78  36 34 29 20 41 70 70 6c   Win64; x 64) Appl
```

2. Raw Socket

2. Raw Socket

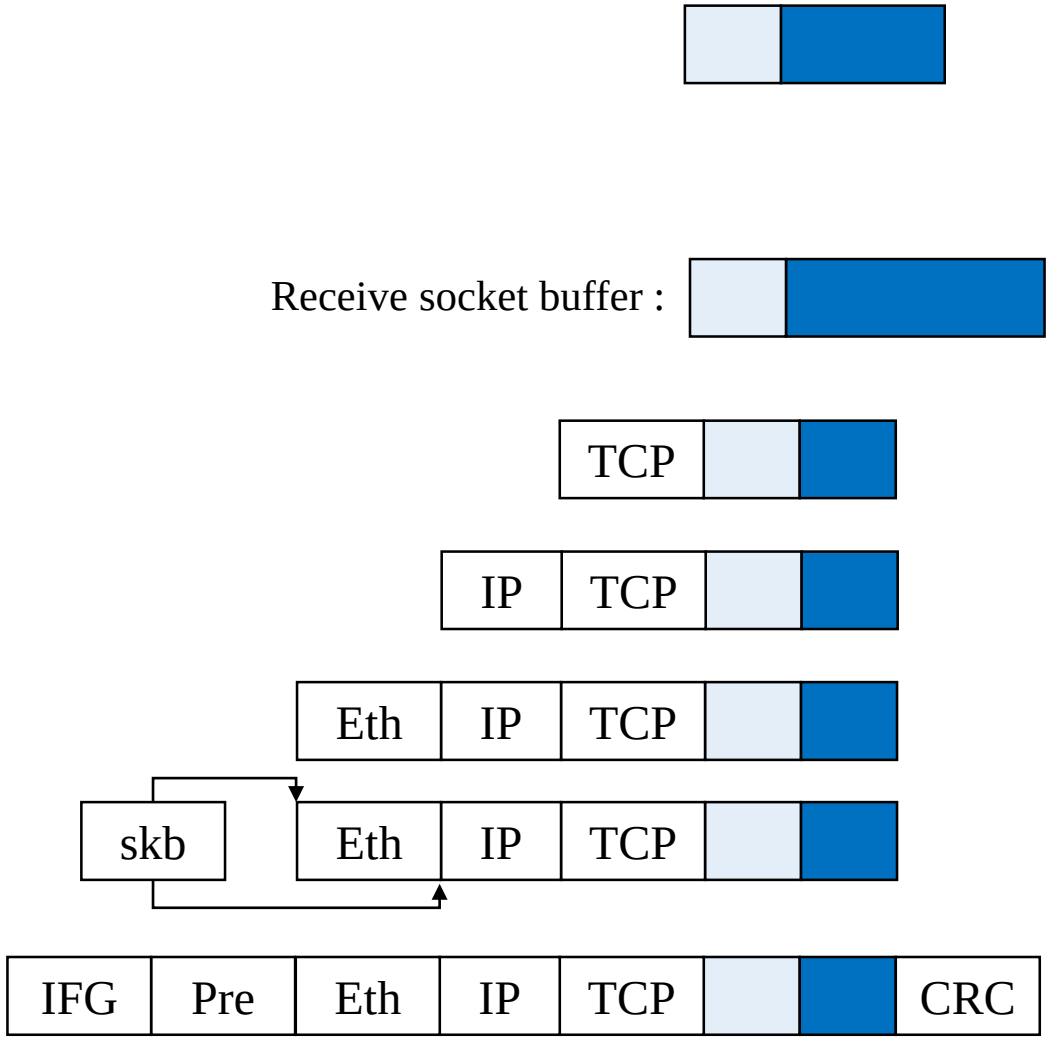
2.1 kernel protocol stack(when sending a packet)



2. Raw Socket

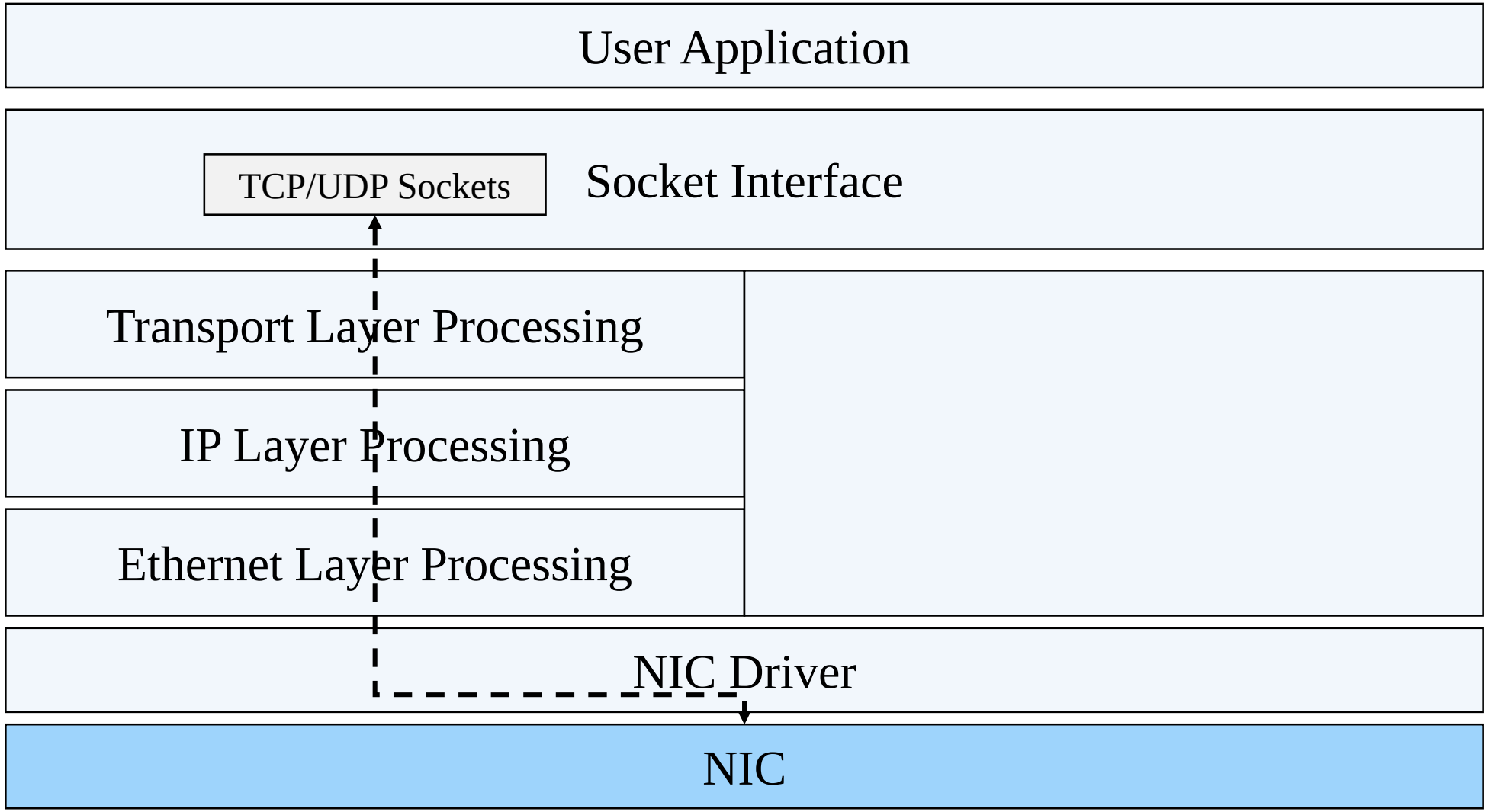
2.1 kernel protocol stack(when receiving a packet)

User	Application	buf = socket.recv(len)
Kernel	File	Validate file descriptor.
Host	Socket	Copy data to user space. Remove it from socket buffer. Tell TCP.
	TCP	Validate the packet. Find TCB and run TCP. Append payload to socket buffer.
	IP	Validate the packet. Perform IP routing. De-multiplex based on IP proto field.
	Ethernet	Validate the packet. De-multiplex based in Ethertype field.
	Driver	Validate the packet. Wrap it in OS packet structure(skb, mbuf, NBL).
Device	NIC	Validate. Transfer it to a host memory. Interrupt the driver.



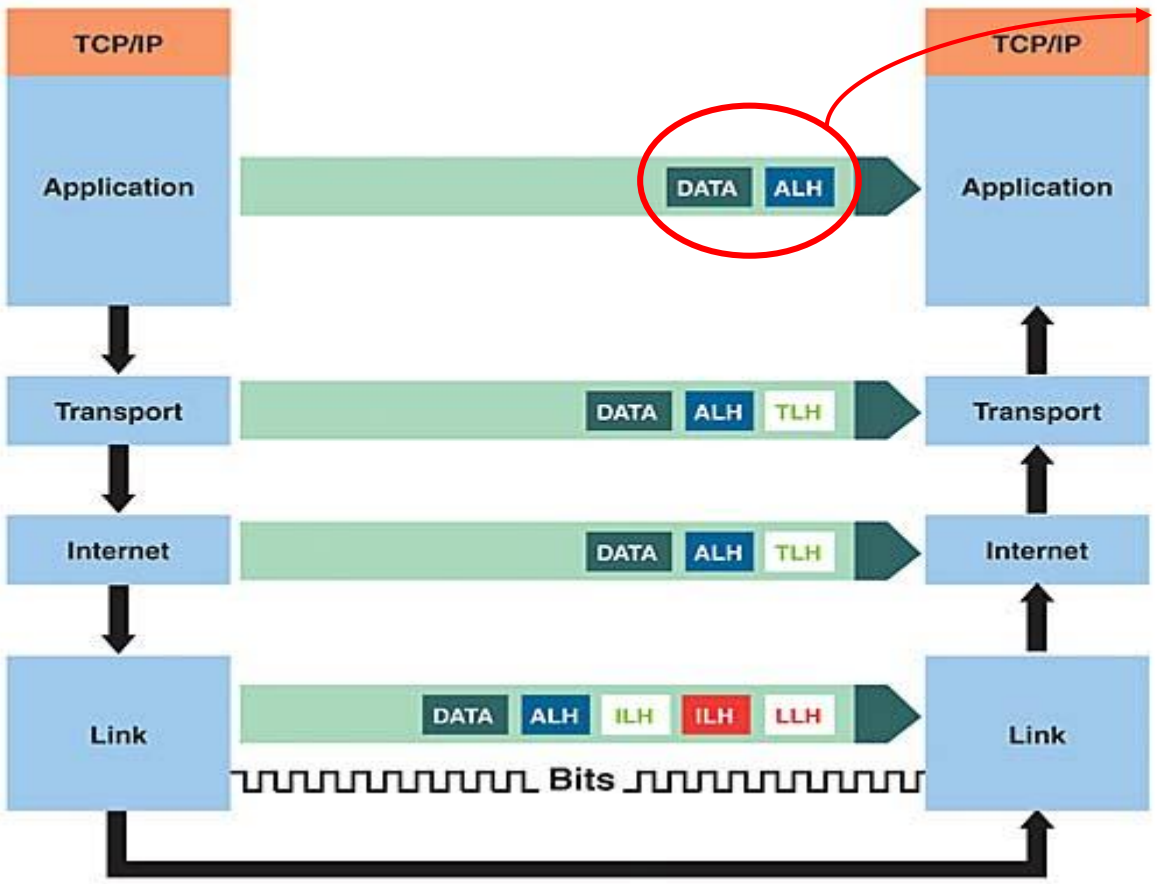
2. Raw Socket

2.2 Socket interface



2. Raw Socket

2.2 Socket interface



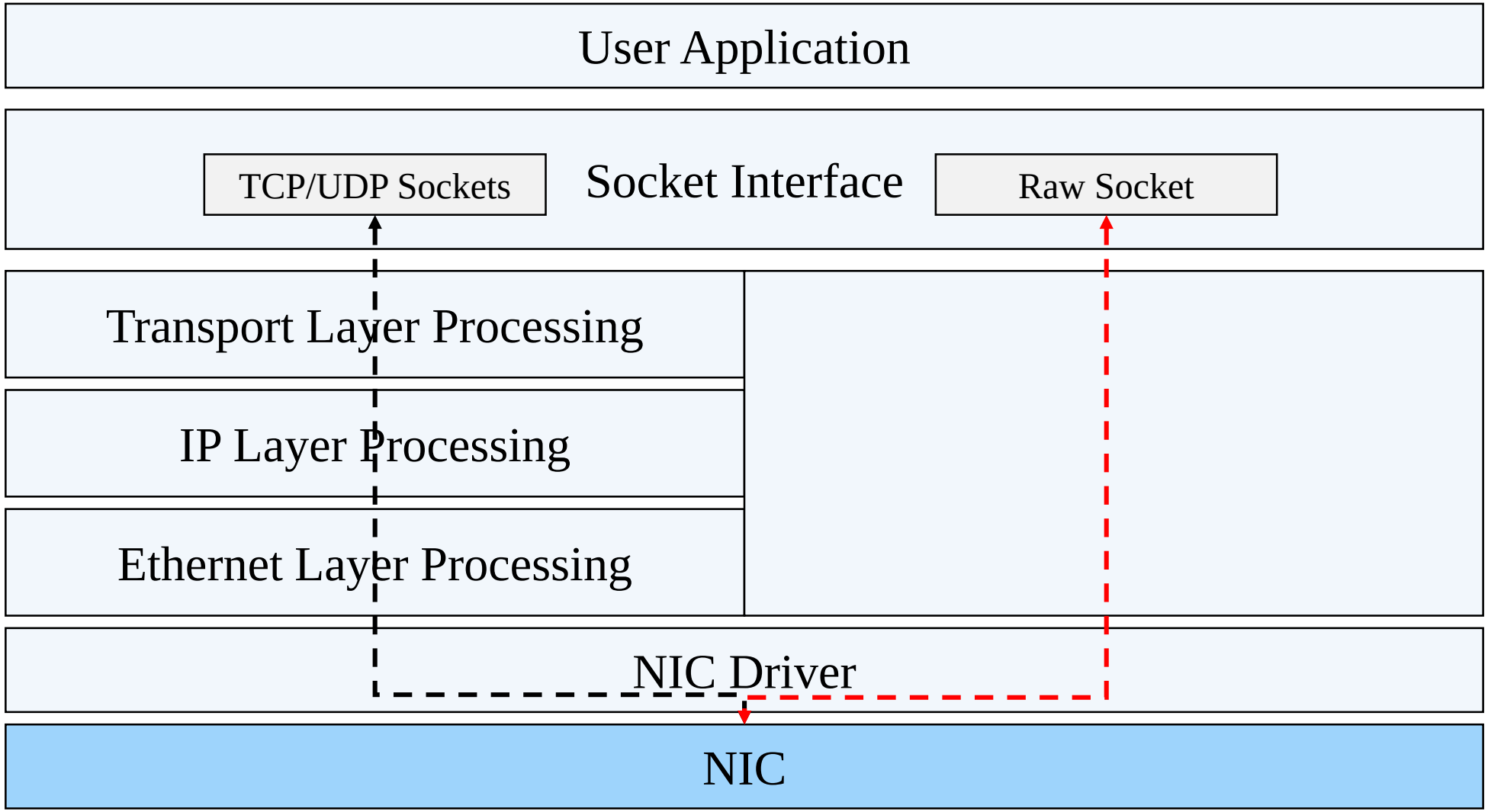
일반 Socket은 응용프로그램에 응용계층 데이터만 전달한다.
모든 프로토콜 헤더는 프로토콜 스택에서 처리하며 제거된다.

Raw Socket

- raw 소켓은 어느 특정한 프로토콜 용의 전송 계층 포매팅 없이 인터넷 프로토콜 패킷을 직접적으로 주고 받게 해주는 인터넷 소켓이다.
- 새로운 프로토콜을 설계하여 구현하거나, 네트워크 장비를 만드는 경우와 같이 패킷을 세밀하게 조작할 때 사용하는 특수한 Socket이다.
- 커널 수준에서만 다룰 수 있었던 프로토콜 헤더를 직접 다룰 수 있다. 또한 사용자 응용프로그램에서 Raw Socket을 통해 직접 만든 패킷을 전송할 수 있다.
- 일반적으로 Raw Socket을 사용하기 위해서는 관리자(root) 권한이 필요하다.

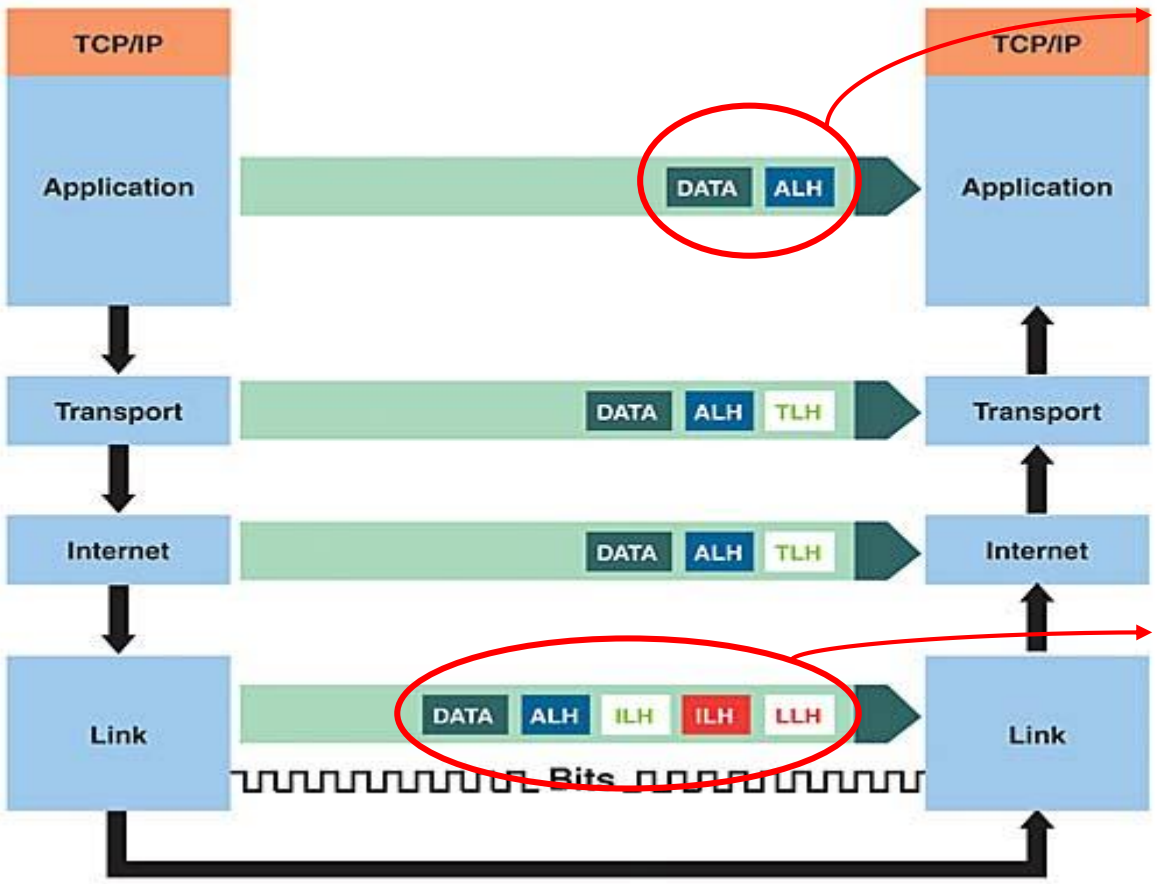
2. Raw Socket

2.3 Raw Socket



2. Raw Socket

2.3 Raw Socket



일반 Socket은 응용프로그램에 응용계층 데이터만 전달한다.
모든 프로토콜 헤더는 프로토콜 스택에서 처리하며 제거된다.

Row Socket은 헤더를 모두 포함한 데이터를 전달한다.

2. Raw Socket

2.4 Creating a raw socket

IPPROTO_ICMP

IPPROTO_UDP

IPPROTO_TCP

ETH_P_IP

ETH_P_ALL

```
socket.socket(${family}, socket.SOCK_RAW, ${protocol})
```

AF_INET L3 이후의 프로토콜 헤더 포함

AF_PACKET L2 이후의 프로토콜 헤더 포함

AF_PACKET은 바인드된 인터페이스가 송/수신하는 모든 패킷을 애플리케이션에 전달

Raw Socket in Linux

- ❖ 일반적으로 아래와 같이 Raw Socket을 생성하면 모든 타입의 프레임을 수집할 수 있다.
- ❖ ETH_P_ALL은 Python에 선언이 되어 있지 않기 때문에 C 언어 라이브러리의 값을 따로 선언해야 한다.
- ❖ AF_PACKET으로 생성한 Raw Socket은 특정 NIC에 바인드할 수 있다. 이때 NIC 이름을 사용한다.
- ❖ 바인드할 경우 해당 NIC에서 수신되는 패킷만 가져오며, 바인드하지 않을 경우 모든 NIC에서 수신되는 패킷을 가져온다.
- ❖ 일반적으로 바인딩할 때 포트번호는 0(Default)를 사용한다.

```
## 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
```

Raw Socket in Linux

```
# AF_INET(인터넷 소켓)을 이용한 RAW_SOCKET in linux with python 3.x
socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_TCP) # TCP Packet
socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_UDP) # UDP Packet
socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_ICMP) # ICMP Packet

# AF_PACKET(패킷 소켓)을 이용한 RAW_SOCKET in linux with python 3.x
socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.IPPROTO_TCP) # TCP Packet
socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.IPPROTO_UDP) # UDP Packet
socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.IPPROTO_ICMP) # ICMP Packet
socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(0x0800)) # IP Packet -> ETH_P_IP
socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(0x0003)) # Every Packets -> ETH_P_ALL
```

Raw Socket in Windows

- ❖ 일반적으로 아래와 같이 Raw Socket을 생성하면 모든 IP 패킷을 수집할 수 있다.
- ❖ Windows에선 AF_PACKET을 사용할 수 없다.
- ❖ AF_INET만 사용하며 때문에 특정 NIC의 IP 주소에 바인딩해야 한다.

```
## Windows NT
sniff_sock = socket.socket(socket.AF_INET,socket.SOCK_RAW,socket.IPPROTO_IP)
sniff_sock.bind(("NIC IP address",0))
sniff_sock.setsockopt(socket.IPPROTO_IP,socket.IP_HDRINCL,1)
sniff_sock.ioctl(socket.SIO_RCVALL,socket.RCVALL_ON)
```

```python
ETH_P_ALL = 0x0003

def sniffing(nic):
    if os.name == 'nt':
        address_familiy = socket.AF_INET
        protocol_type = socket.IPPROTO_IP
    else:
        address_familiy = socket.AF_PACKET
        protocol_type = socket.ntohs(ETH_P_ALL)

    with socket.socket(address_familiy, 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)

        for p in data:
            print("%02x" % p, end=" ")

        if os.name == 'nt':
            sniffe_sock.ioctl(socket.SIO_RCVALL,socket.RCVALL_OFF)
```

2. Raw Socket

2.5 Raw socket example

root@ubuntu: /home/famous/Desktop/TA/socket/assignment/assignment_8

File Edit View Search Terminal Tabs Help

root@ubuntu: /home/f... | root@ubuntu: /home/f... | root@ubuntu: /home/f... | root@ubuntu: /home/f...

```
root@ubuntu:/home/famous/Desktop/TA/socket/assignment/assignment_8# python3 raw_sniffer.py -i ens33
00 50 56 fd 07 5c 00 0c 29 44 5e 1b 08 00 45 00 00 45 1f d8 40 00 40 11 08 f4 c0 a8 c8 88 c0 a8 c8
02 9f 8b 00 35 00 31 12 1f 69 0f 01 00 00 01 00 00 00 00 00 00 0b 74 63 6d 69 74 78 75 72 64 73 6a
0b 6c 6f 63 61 6c 64 6f 6d 61 69 6e 00 00 rooroooroooroooroooroooroooroooroooroooroooroot@urooroot@uroo
root@ubuntu:/home/famous/Desktop/TA/socket/assignment/assignment_8#
```

관리자: Anaconda Prompt

```
(base) C:₩Users₩Famous₩OneDrive - 충북대학교₩학교공부₩4학년 1학기₩컴퓨터네트워크TA₩8주차>python raw_socket_sniffer.py -i 192.168.25.8
45 00 00 33 42 b2 40 00 80 11 17 70 c0 a8 19 08 ac d9 1a 0e d9 f3 01 bb 00 1f 3a 88 0c 56 2d 1a 20 8a 09 4a 50 34 07 fb 1e c5 0e f2 0f
 38 e4 fa 72 82 f9
(base) C:₩Users₩Famous₩OneDrive - 충북대학교₩학교공부₩4학년 1학기₩컴퓨터네트워크TA₩8주차>
```

```python
def dumpcode(buf):
    print("%7s"% "offset ", end='')

    for i in range(0, 16):
        print("%02x " % i, end='')

        if not (i%16-7):
            print("- ", end='')

    print("")

    for i in range(0, len(buf)):
        if not i%16:
            print("0x%04x" % i, end= ' ')

        print("%02x" % buf[i], end= ' ')

        if not (i % 16 - 7):
            print("- ", end='')

        if not (i % 16 - 15):
            print(" ")

    print("")
```

2. Raw Socket

2.5 Raw socket example

```
root@ubuntu:/home/famous/Desktop/TA/socket/assignment/assignment_8# python3 raw_sniffer.py -i lo
offset 00 01 02 03 04 05 06 07 - 08 09 0a 0b 0c 0d 0e 0f
0x0000 00 00 00 00 00 00 00 00 - 00 00 00 00 08 00 45 00
0x0010 00 3c 47 4e 40 00 40 06 - f5 6b 7f 00 00 01 7f 00
0x0020 00 01 d7 ee 22 b8 55 4c - f1 98 00 00 00 00 a0 02
0x0030 aa aa fe 30 00 00 02 04 - ff d7 04 02 08 0a da 5d
0x0040 55 29 00 00 00 00 01 03 - 03 07
root@ubuntu:/home/famous/Desktop/TA/socket/assignment/assignment_8#
```

Assignment #8

- 수업 Github assignment_8에 있는 raw_sniffer.py를 사용한 패킷 분석
  - Linux에서 수행할 것
  - raw_niffer.py로 Assignment#2(문자열 거꾸로 전송)가 실행되면서 서버-클라이언트간 주고받은 첫 번째 TCP 패킷을 캡처해 사진 첨부(문자열은 팀 이름을 전달)
  - 캡처한 패킷을 상세히 분석
  - 보고서는 2장 내로 작성
- 팀 대표가 barcel@naver.com으로 제출 (5.7일까지)
  - Title : [컴퓨터네트워크][학번][이름][과제_N]
  - Content : github repo url

    팀명 : 길동이네

    팀원 : 홍길동(학번), 고길동(학번)