

# Packet Sniffing with RAW Socket

유명성

# 1. Packet Sniffing

---

# 1. Packet Sniffing

## 1.1 Packet

### 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)

# 1. Packet Sniffing

---

## 1.2 Packet Sniffing

### Packet Sniffing

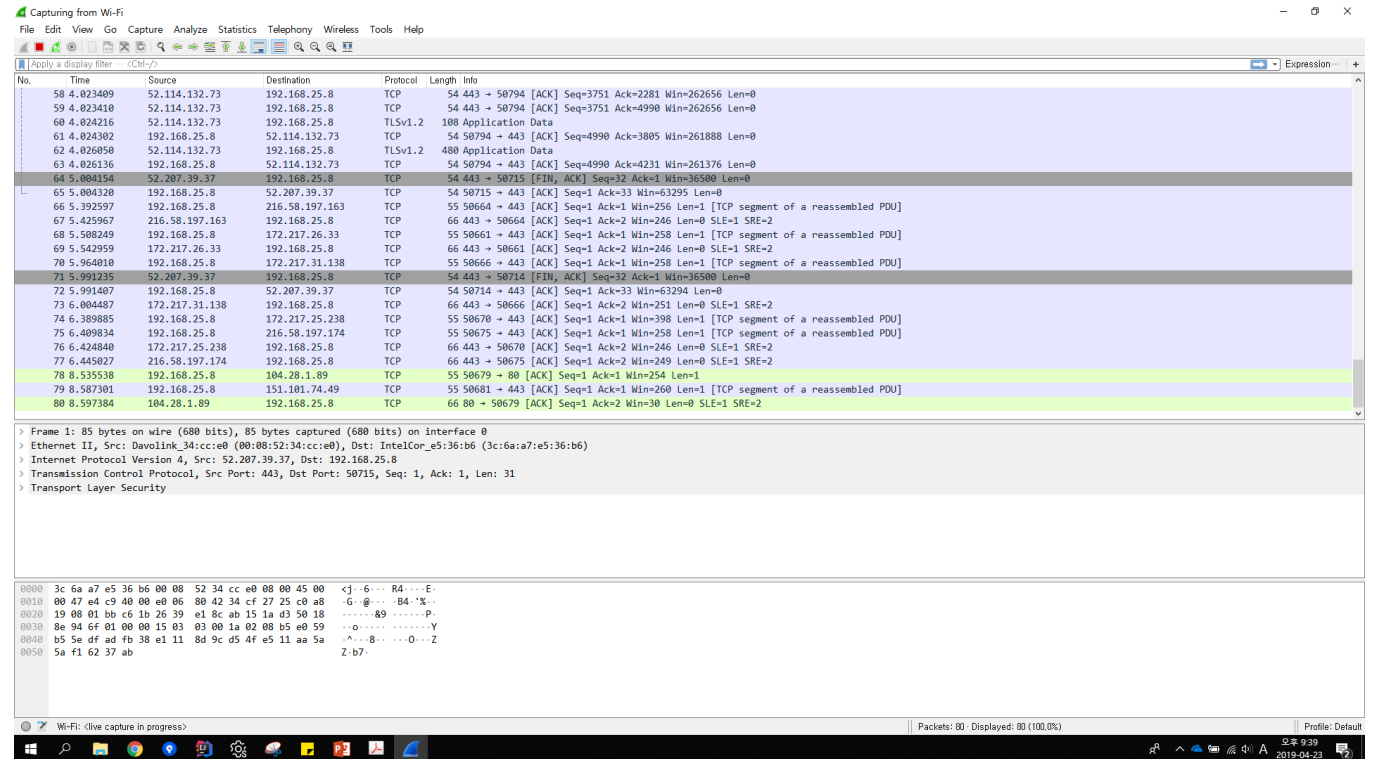
- ❖ 패킷 스니핑은 패킷 캡처, 패킷 분석으로도 불리며 네트워크 상에서 발생하는 일을 이해하기 위해 네트워크를 통해 전달되는 패킷을 수집하고 분석하는 행위이다.
- ❖ 주로 대역폭 분석 등 네트워크 상태를 분석을 위해 사용되며, 새로운 프로토콜 개발에도 쓰인다.
- ❖ 보안상 악의적인 공격을 탐지하고 공격자를 추적하기 위해서도 사용된다.
- ❖ 대표적인 패킷 스니핑 툴에 tcpdump, wireshark, kismet 등이 있다.

# 1. Packet Sniffing

## 1.3 Wireshark

### 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)



# 1. Packet Sniffing

---

## 1.4 pcap



### 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, Pcap, WinPcap, scapy
  - Java : jpcap, jNetPcap

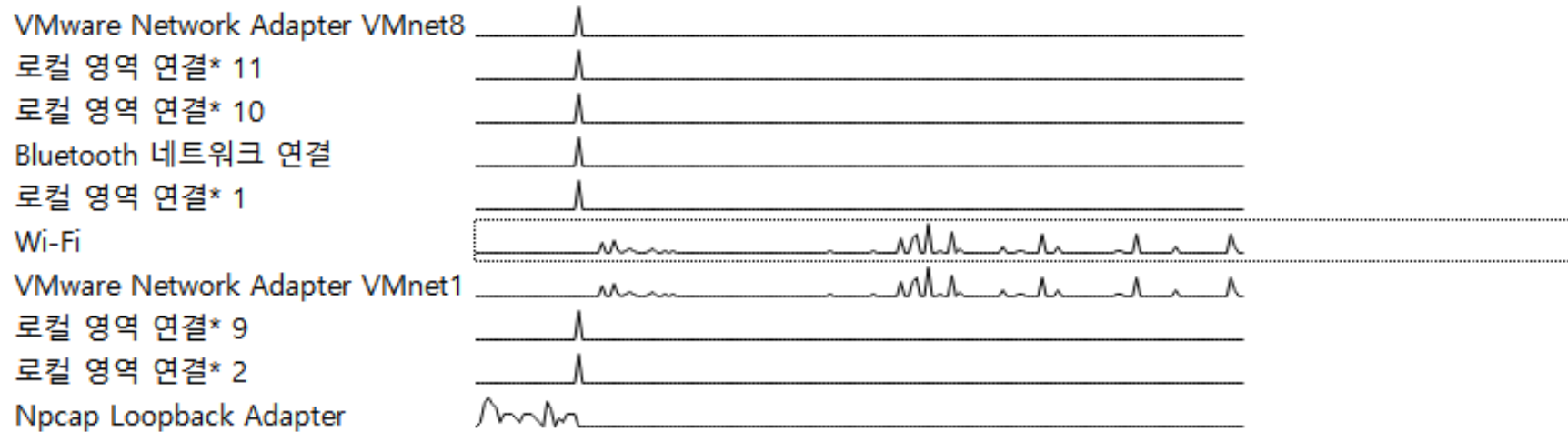
# 1. Packet Sniffing

## 1.5 packet capture

Welcome to Wireshark

### Capture

...using this filter:



원하는 인터페이스를 선택하여 패킷을 캡처할 수 있다.

# 1. Packet Sniffing

## 1.5 packet capture

캡처된 패킷 목록

패킷 프로토콜 및  
필드 정보

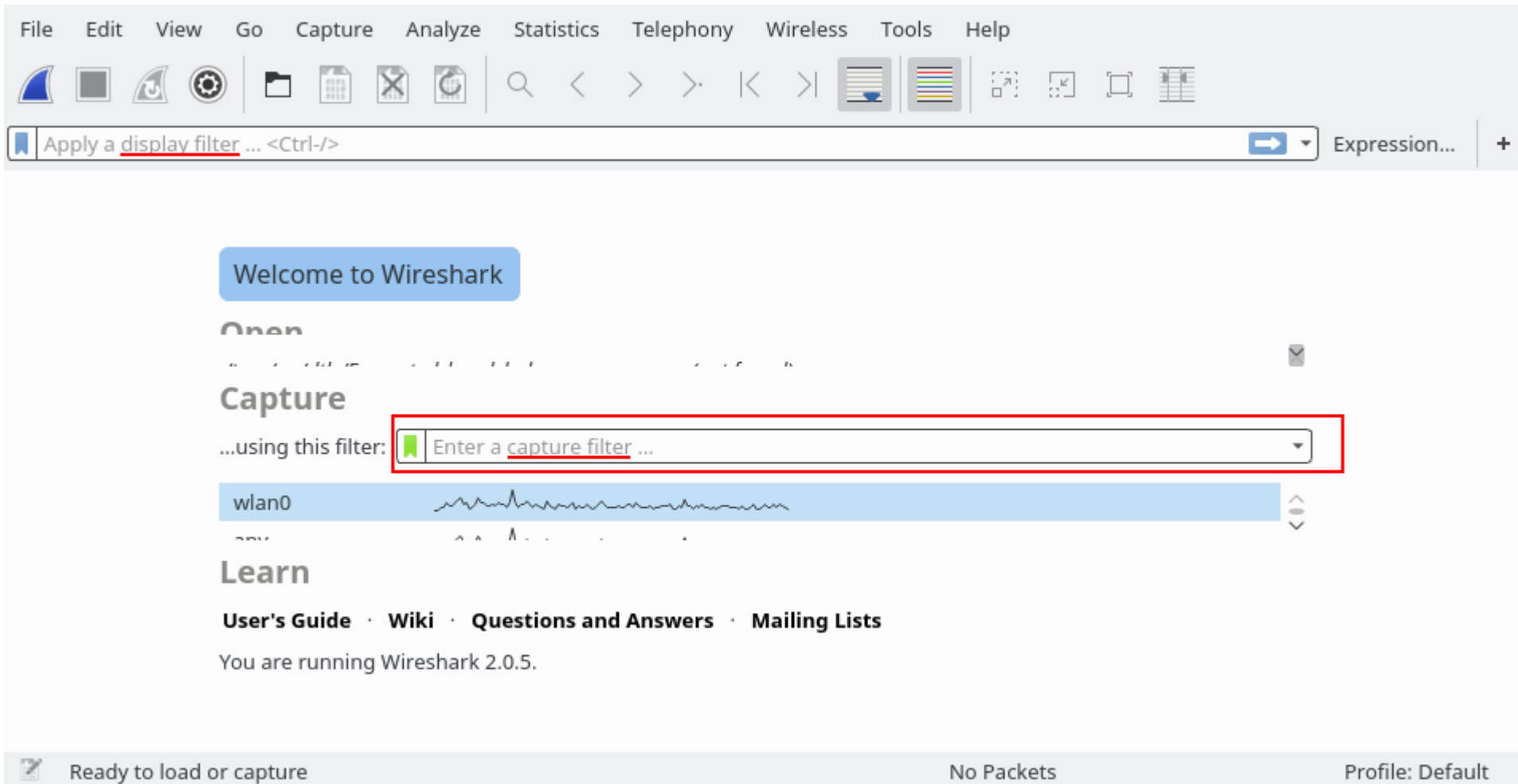
Raw Data

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
```

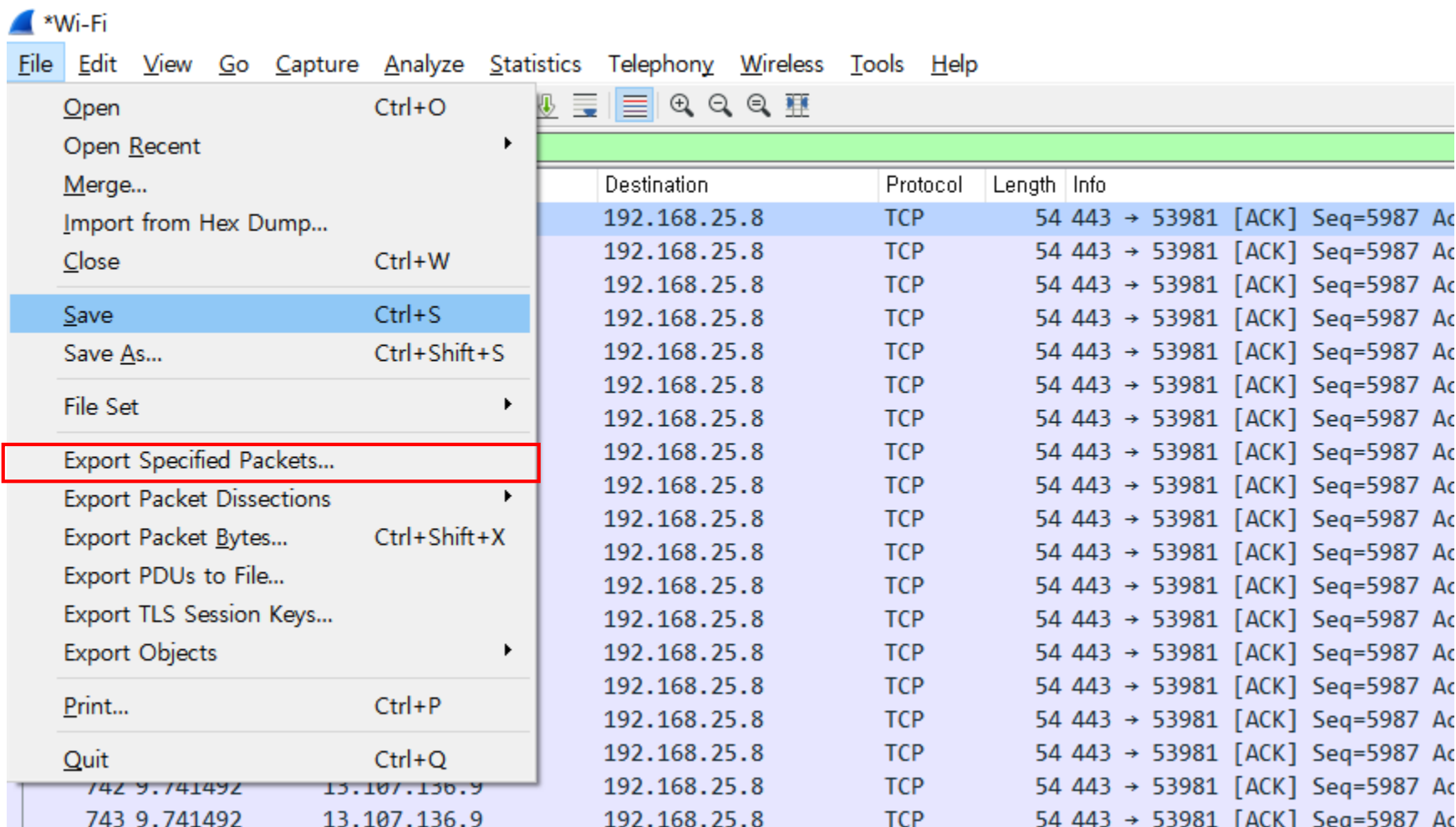
or

```
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

The image shows a Wi-Fi packet capture application window. The menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. The toolbar contains various icons for packet capture and analysis. The main window displays a list of captured packets with columns for No., Time, Source, Destination, Protocol, Length, and Info. A context menu is open over the packet list, showing options like Expand Subtrees, Collapse Subtrees, Expand All, Collapse All, Apply as Column, Apply as Filter (highlighted), Prepare a Filter, Conversation Filter, Colorize with Filter, Follow, Copy, Show Packet Bytes..., Export Packet Bytes..., Wiki Protocol Page, Filter Field Reference, Protocol Preferences, Decode As..., Go to Linked Packet, and Show Linked Packet in New Window. The packet list shows several TCP ACK packets from 13.107.136.9 to 192.168.25.8. The packet details pane at the bottom shows the selected packet's details, including Total Length, Identification, Flags, Time to live, Protocol, Header checksum, and Source/Destination addresses.

No.	Time	Source	Destination	Protocol	Length	Info
689	9.734396	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=16420 Win=524032 Len=0
690	9.734397	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=17860 Win=525568 Len=0
695	9.7353		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=19300 Win=524032 Len=0
696	9.7353		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=25060 Win=525568 Len=0
697	9.7353		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=26500 Win=524032 Len=0
698	9.7353		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=27940 Win=525568 Len=0
699	9.7353		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=30820 Win=525568 Len=0
718	9.7370		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=32260 Win=524032 Len=0
719	9.7370		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=33700 Win=525568 Len=0
720	9.7370		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=35140 Win=524032 Len=0
721	9.7370		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=36580 Win=525568 Len=0
722	9.7370		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=38020 Win=524032 Len=0
723	9.7370		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=39460 Win=525568 Len=0
724	9.7370		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=40900 Win=524032 Len=0
739	9.7414		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=42340 Win=525568 Len=0
740	9.7414		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=43780 Win=524032 Len=0
741	9.7414		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=45220 Win=525568 Len=0
742	9.7414		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=48100 Win=525568 Len=0
743	9.7414		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=49540 Win=524032 Len=0
756	9.7431		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=50980 Win=525568 Len=0
757	9.7431		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=52420 Win=524032 Len=0
758	9.7431		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=61060 Win=525568 Len=0
765	9.7453		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=62500 Win=524032 Len=0
766	9.7453		.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=65380 Win=525568 Len=0

.... 0101  
Differentiated Services Code Point: 0, ECN: Not-ECT)  
Total Length: 40  
Identification: 0x7701 (30465)  
Flags: 0x4000, Don't fragment  
Time to live: 120  
Protocol: TCP (6)  
Header checksum: 0x1caa [validation disabled]  
[Header checksum status: Unverified]  
Source: 13.107.136.9  
Destination: 192.168.25.8

# 1. Packet Sniffing

## 1.5 packet capture

특정 TCP/UDP 스트림만  
출력할 수 도 있다.

The image shows the Wireshark network protocol analyzer interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. Below the menu is a toolbar with various icons. The main window is divided into three panes. The top pane shows a list of captured packets with columns for No., Time, Source, Destination, Protocol, Length, and Info. The middle pane shows a detailed view of the selected packet (No. 705), including the Ethernet II header, Internet Protocol Version 4 header, and TCP segment. The bottom pane shows the raw packet data in hexadecimal and ASCII. A context menu is open over the selected packet, with the 'Follow' option highlighted, and a sub-menu showing 'TCP Stream' selected.

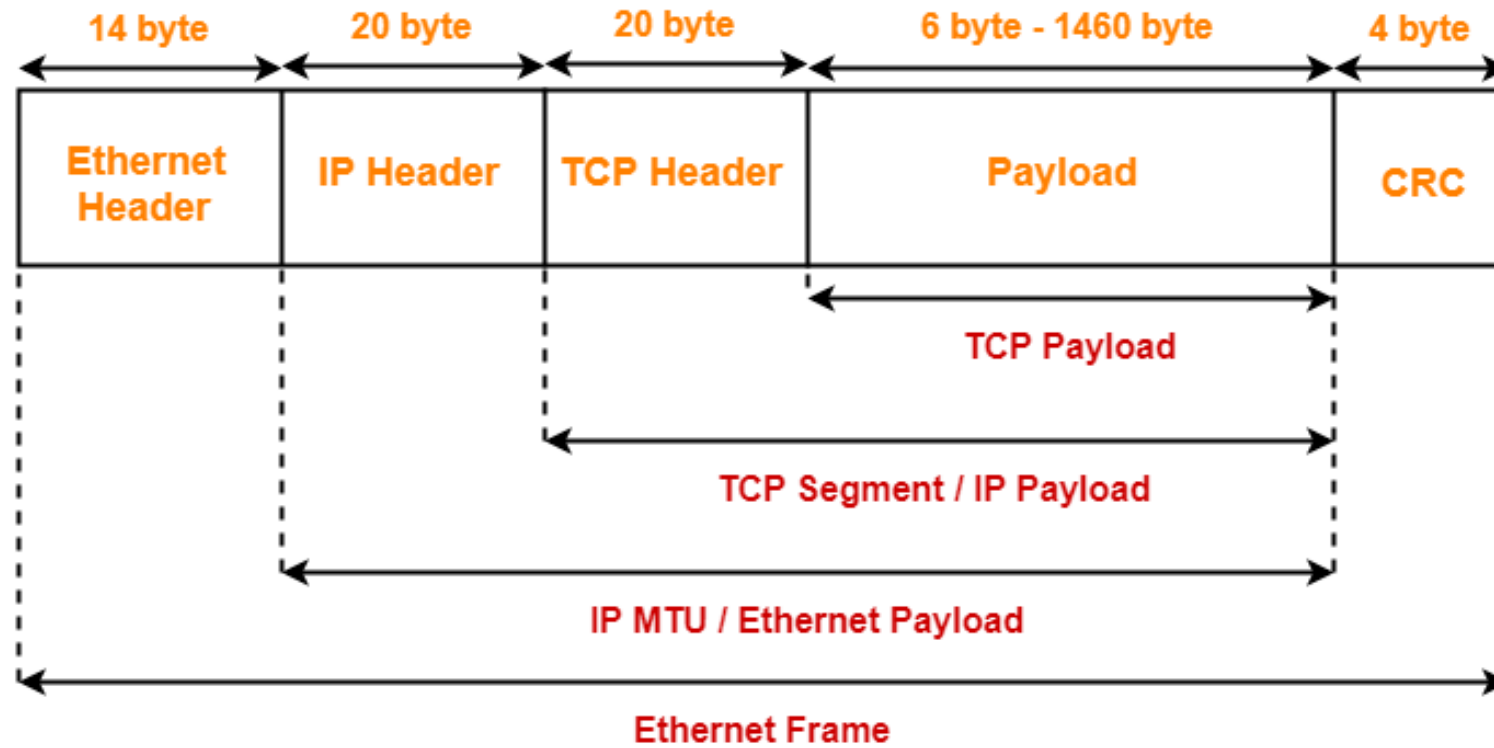
No.	Time	Source	Destination	Protocol	Length	Info
700	9.735393	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=49540 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
701	9.735394	192.168.25.8	13.107.136.9	TLSv1.2	1494	Application Data [TCP segment of a reassembled PDU]
702	9.735395	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=52420 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
703	9.735396	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=53860 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
704	9.735396	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=55300 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
705	9.735396	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=56740 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
706	9.735396	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=58180 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
707	9.735397	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=59620 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
708	9.735397	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=61060 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
709	9.735398	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=62500 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
710	9.735398	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=63940 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
711	9.735398	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=65380 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
712	9.735399	192.168.25.8	13.107.136.9	TLSv1.2	1494	Application Data [TCP segment of a reassembled PDU]
713	9.735399	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=68260 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
714	9.735399	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=69700 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
715	9.735400	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=71140 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
716	9.735400	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=72580 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
717	9.735400	192.168.25.8	13.107.136.9	TCP	1494	53981 → 443 [ACK] Seq=74020 Ack=5987 Win=65792 Len=1440 [TCP segment of a reassembled PDU]
718	9.737087	13.107.136.9	192.168.25.8	SCTP	54	443 → 53981 [ACK] Seq=5987 Ack=32260 Win=524032 Len=0
719	9.737087	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=33700 Win=525568 Len=0
720	9.737087	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=35140 Win=524032 Len=0
721	9.737087	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=36580 Win=525568 Len=0
722	9.737088	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=38020 Win=524032 Len=0
723	9.737088	13.107.136.9	192.168.25.8	TCP	54	443 → 53981 [ACK] Seq=5987 Ack=39460 Win=525568 Len=0

Frame 705: 1494 bytes on wire (11952 bits) captured (11952 bits) on interface 0  
Ethernet II, Src: Intel(R) Dual Band Wireless-AC 80, Dst: Davolink\_34:cc:e0 (00:08:52:34:cc:e0)  
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: 1480  
Identification: 0x4450 (17488)  
> 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..

# 1. Packet Sniffing

## 1.6 Ethernet sniffing

### TCP/IP Packet 구조

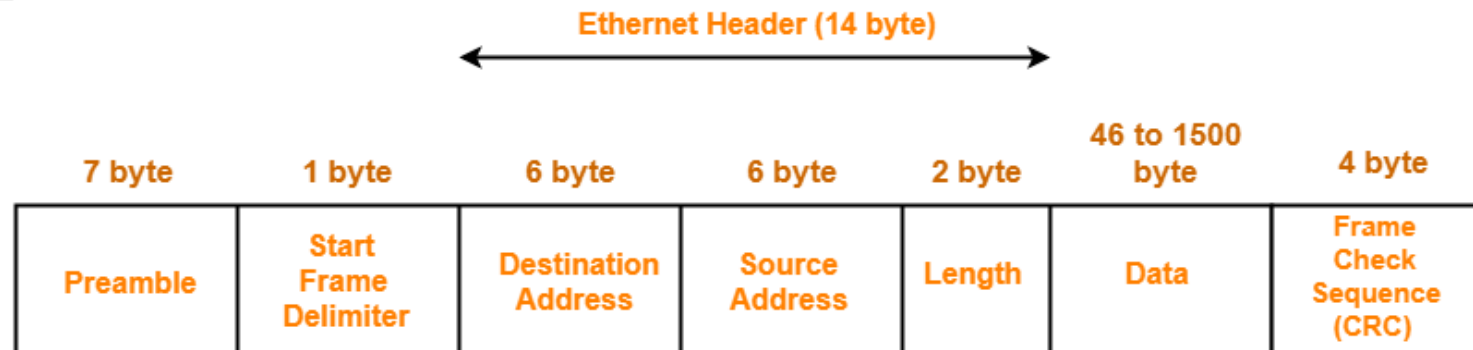


# 1. Packet Sniffing

## 1.6 Ethernet sniffing

```
> Frame 10: 55 bytes on wire (440 bits), 55 bytes captured (440 bits) on interface 0
  ✓ 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·1!· ....q·P·
0030  01 02 26 5a 00 00 00  ..&Z...
```



IEEE 802.3 Ethernet Frame Format

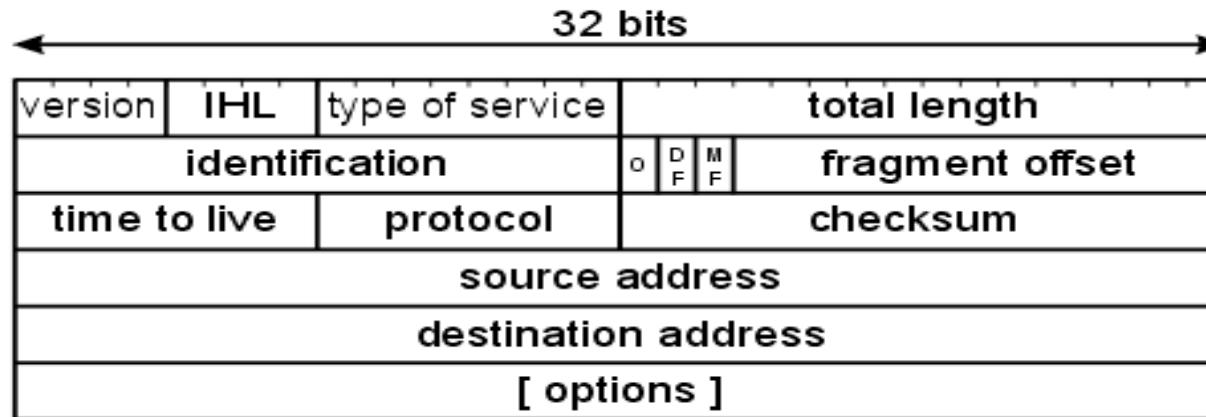
# 1. Packet Sniffing

## 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
▼ 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
```

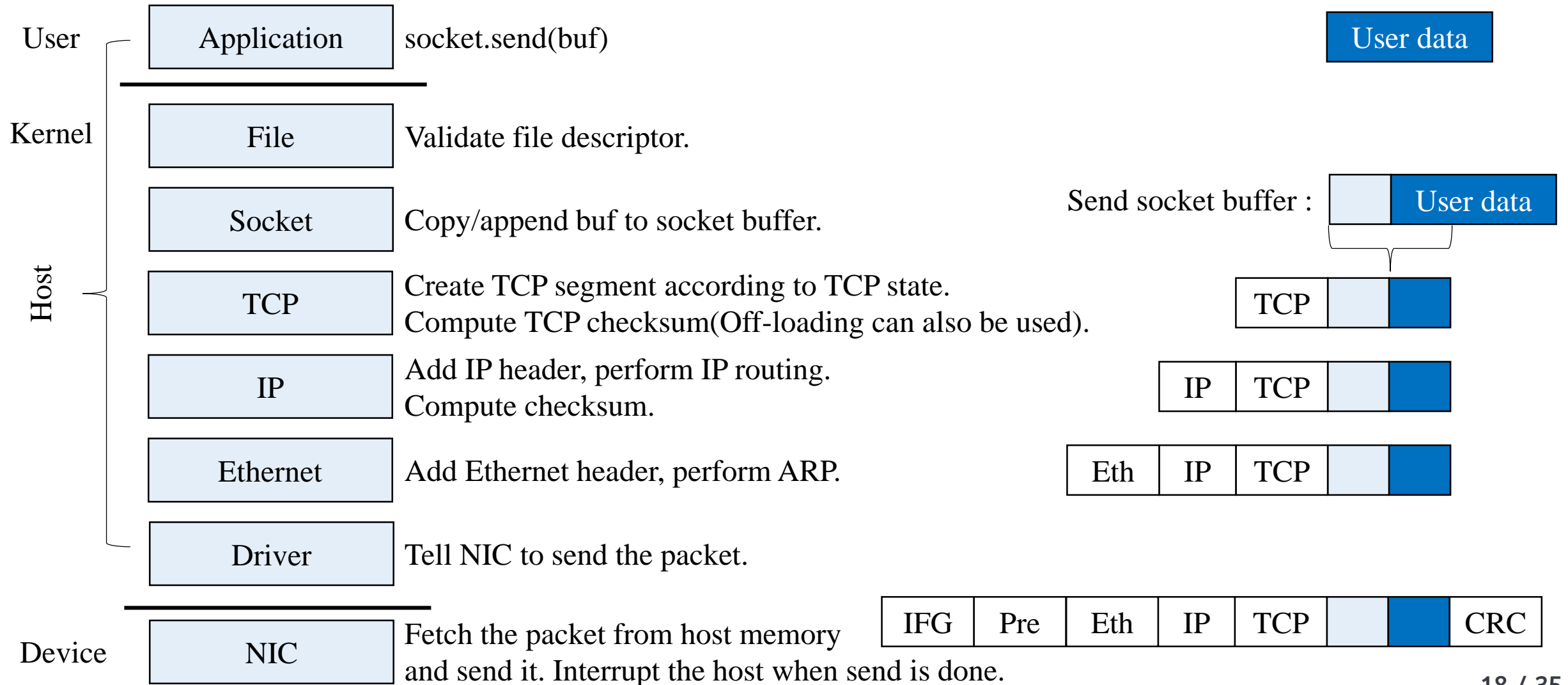
0030	01 00 83 0b 00 00	47 45 54 20 2f 20 48 54 54 50	.....GET / HTTP
0040	2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 67 69 6c 67		/1.1..Host: gilg
0050	69 6c 2e 6e 65 74 0d 0a 43 6f 6e 6e 65 63 74 69		il.net.. Connecti
0060	6f 6e 3a 20 6b 65 65 70 2d 61 6c 69 76 65 0d 0a		on: keep -alive..
0070	55 70 67 72 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

---

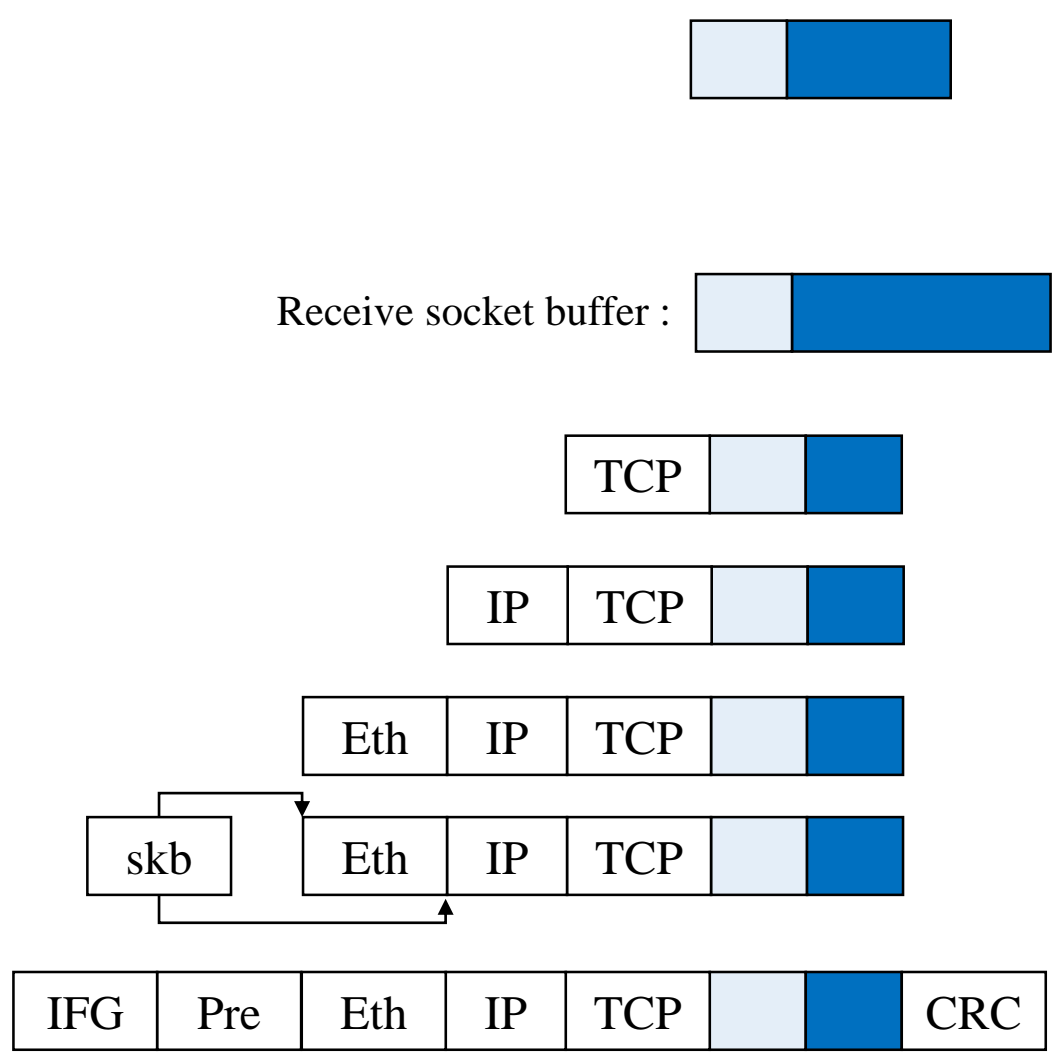
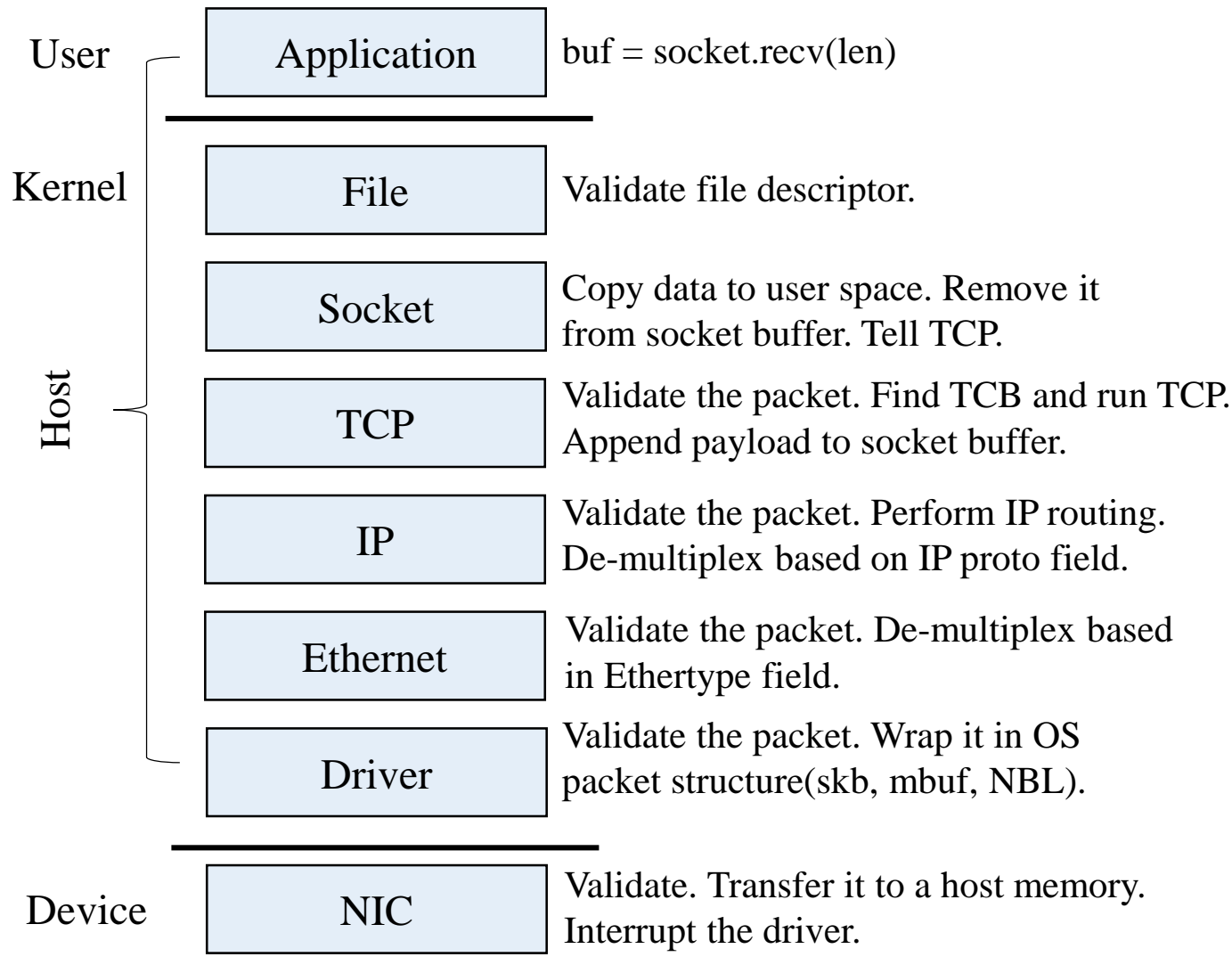
## 2. Raw Socket

### 2.1 kernel protocol stack(when sending a packet)



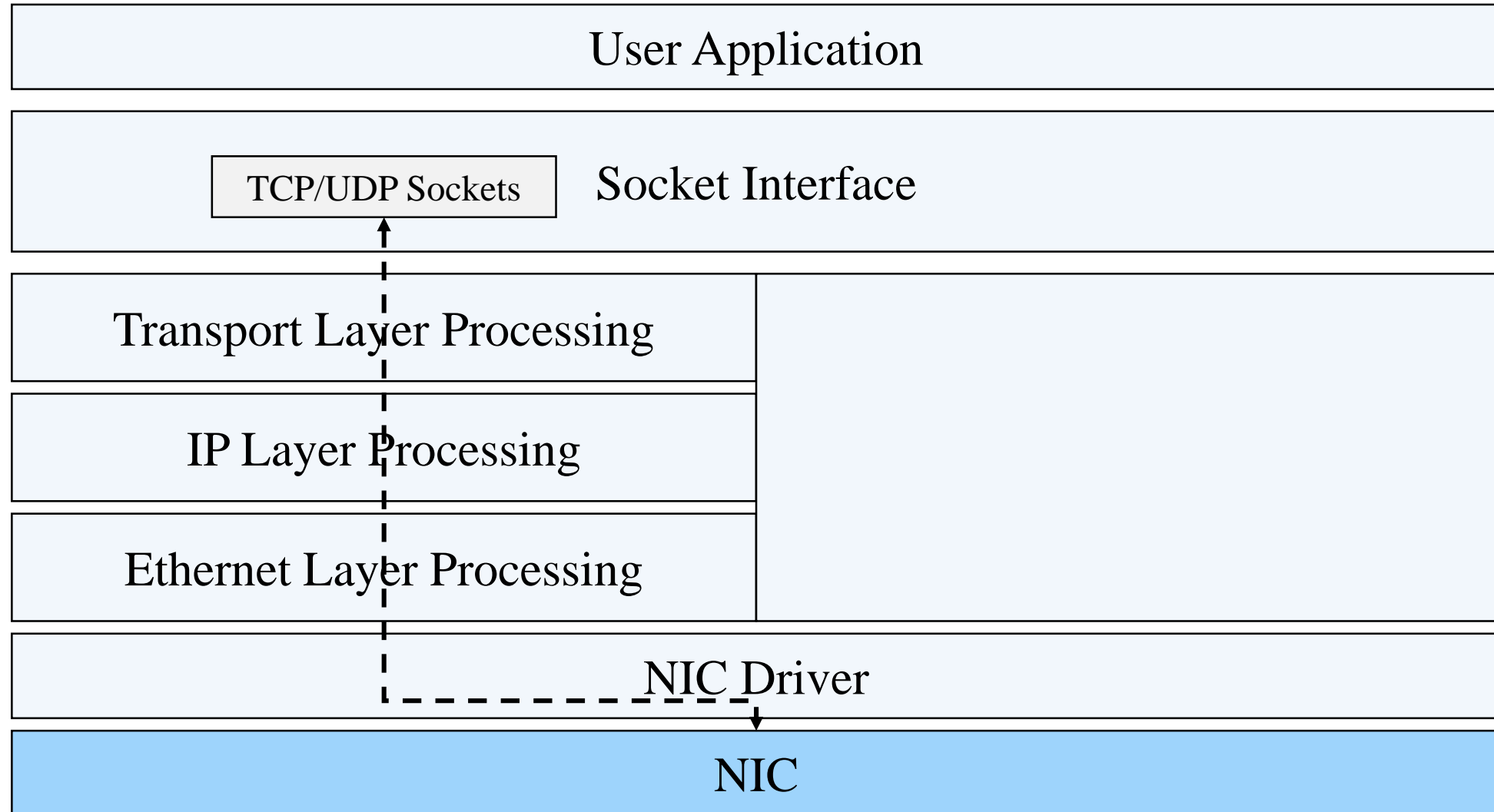
# 2. Raw Socket

## 2.1 kernel protocol stack(when receiving a packet)



## 2. Raw Socket

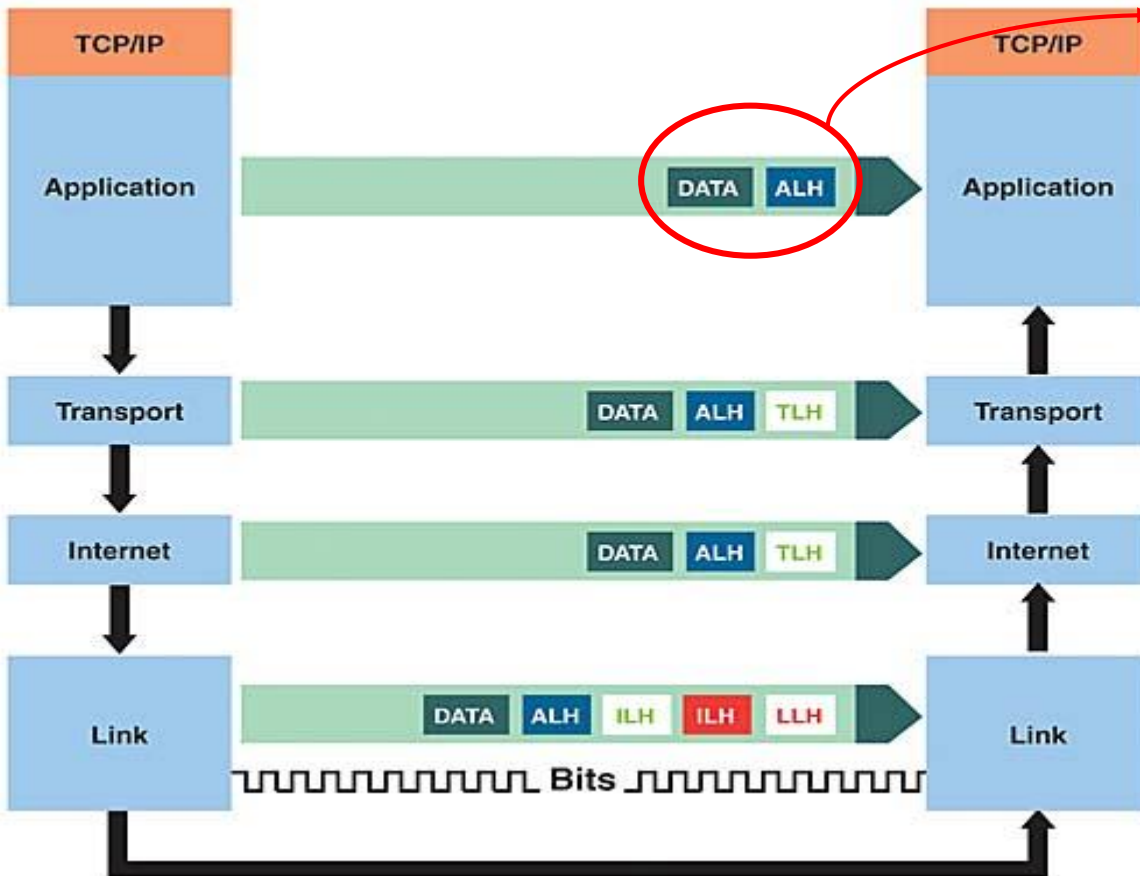
### 2.2 Socket interface



## 2. Raw Socket

### 2.2 Socket interface

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

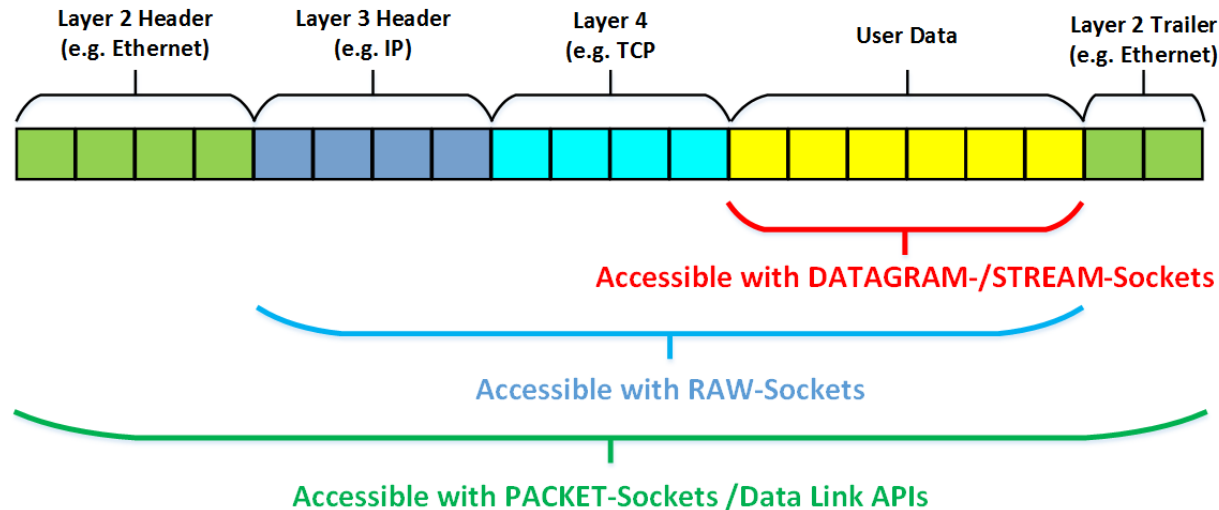


## 2. Raw Socket

### 2.3 Raw Socket

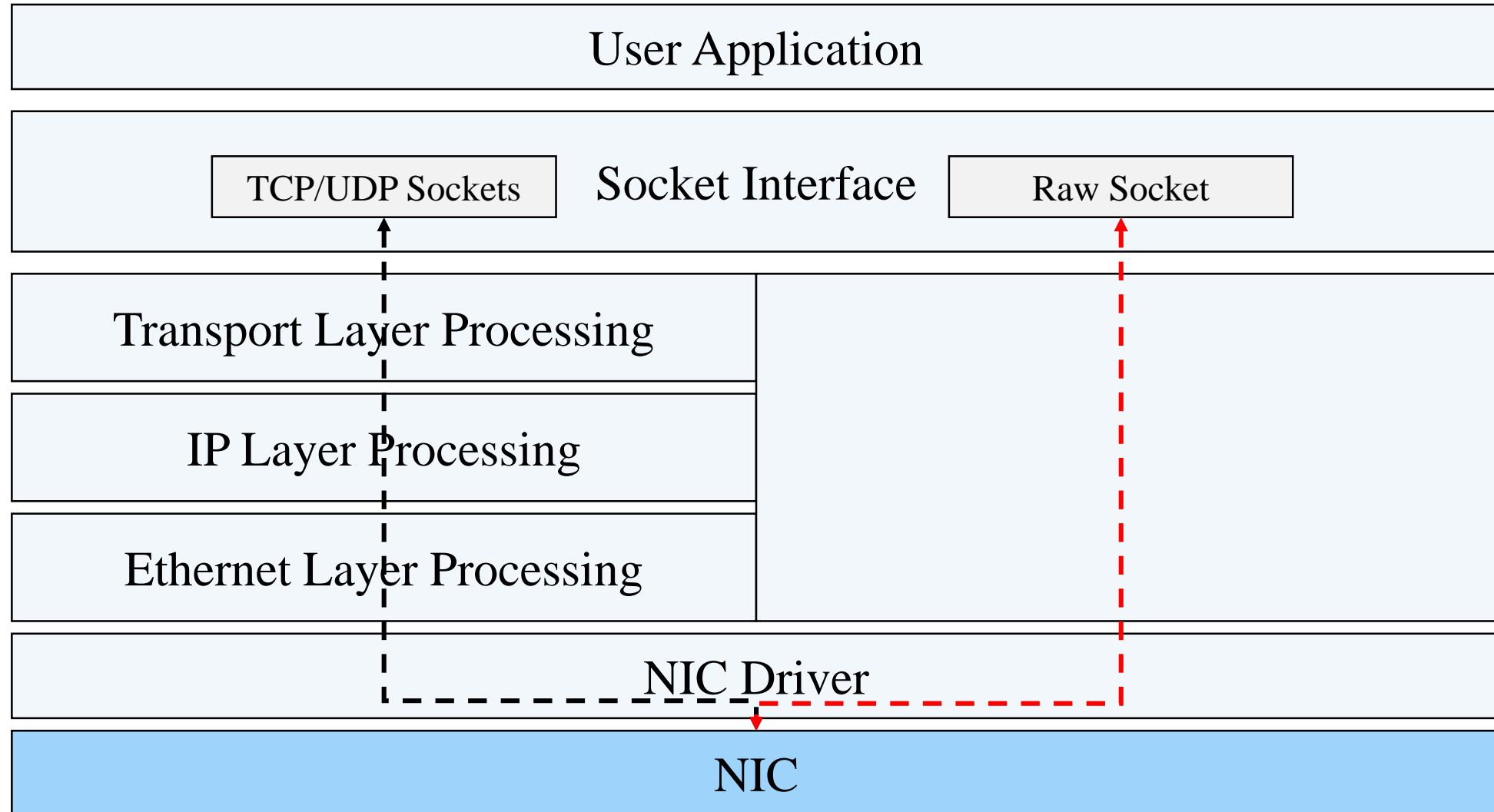
#### Raw Socket

- ❖ 인터넷 소켓의 한 종류로 IP를 사용하는 프로토콜의 L3 계층 이후 헤더를 직접 조작할 수 있는 소켓.
- ❖ ICMP, IGMP 등의 프로토콜을 사용해 패킷을 보내고 받을 때 사용한다.
- ❖ Raw Socket을 사용하려면 관리자(Root, Administrator) 권한이 필요하다.
- ❖ OS 별로 API 형태가 다르다.
- ❖ Packet Socket의 특수한 형태인 Raw Socket으로 L2 계층 이후의 헤더를 직접 조작할 수 있다.(UNIX)



## 2. Raw Socket

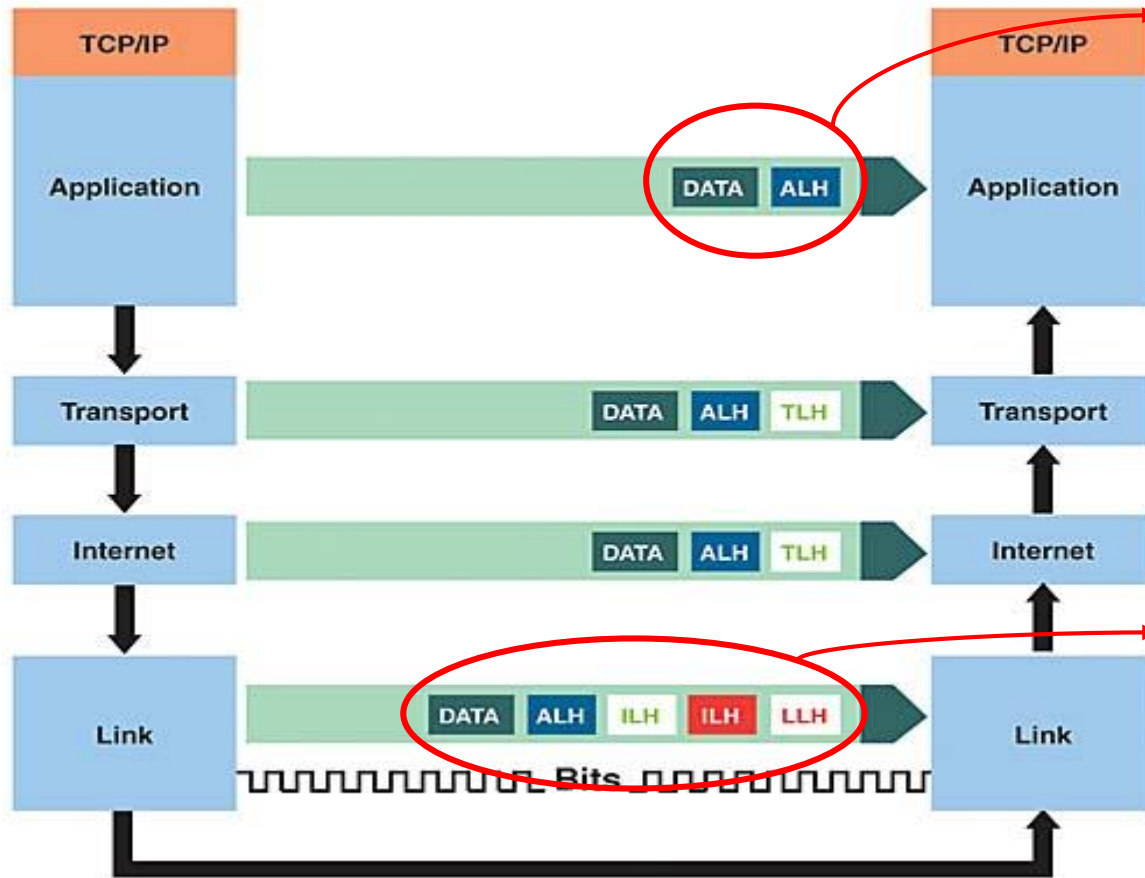
### 2.3 Raw Socket





## 2. Raw Socket

### 2.3 Raw Socket



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

Raw 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 Packet Socket을 만들며, 바인드된 인터페이스가  
송/수신하는 모든 패킷을 애플리케이션에 전달

## 2. Raw Socket

### 2.3 Raw Socket

#### Raw Socket in Linux

- ❖ Raw Socket 생성 시 Root 권한 필요.
- ❖ Port는 더 이상 통신의 종단점이 아님 -> Raw Socket은 Driver 레벨에서 동작
- ❖ bind(), connect() API를 호출할 필요가 없다. -> bind()의 경우 특정 주소 및 NIC에 Raw Socket을 매핑하기 위해 사용될 수 있다.
- ❖ listen(), accept() API는 동작하지 않는다. -> Server/Client 개념이 사라짐.
- ❖ 기본적으로 IP 헤더는 커널이 생성하지만, IP\_HDRINCL 옵션을 통해 사용자가 직접 생성할 수 있다.
- ❖ Raw Socket 생성 시 IP 기반 프로토콜을 지정하면 해당 프로토콜만 수신한다.
- ❖ IPPROTO\_RAW를 지정하면 모든 IP based 프로토콜을 직접 작성해 보낼 수 있지만, 받을 수는 없다.
- ❖ L2에 접근하거나, 모든 IP based 프로토콜을 수신하려면 Packet Socket을 사용해야 한다.

## 2. Raw Socket

### 2.4 Creating a raw socket

#### 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
```

## 2. Raw Socket

### 2.4 Creating a raw socket

#### ■ 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
```

## 2. Raw Socket

---

### 2.3 Raw Socket



#### Raw Socket in Windows

- ❖ 타 OS와 마찬가지로 Administrator 권한 필요
- ❖ Raw Socket으로 TCP 패킷을 보낼 수 없다.
- ❖ 유효하지 않은(자신의 것이 아닌) src IP를 가진 UDP 패킷을 Raw Socket으로 보낼 수 없다.
- ❖ 보유중인 NIC와 매칭되지 않은 IP를 가진 모든 UDP 패킷은 Raw Socket으로 보내지지 않고 Drop된다.
- ❖ Windows는 Packet Socket을 제공하지 않는다.

## 2. Raw Socket

### 2.4 Creating a raw socket

#### ■ Raw Socket in Windows

- ❖ 일반적으로 아래와 같이 Raw Socket을 생성하면 모든 IP 패킷을 수집할 수 있다.
- ❖ Windows에선 AF\_PACKET(Packet Socket)을 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)
```

```
5  ETH_P_ALL = 0x0003
6
7  def sniffing(nic):
8      if os.name == 'nt':
9          address_familiy = socket.AF_INET
10         protocol_type = socket.IPPROTO_IP
11     else:
12         address_familiy = socket.AF_PACKET
13         protocol_type = socket.ntohs(ETH_P_ALL)
14
15     with socket.socket(address_familiy, socket.SOCK_RAW, protocol_type) as sniffed_socket:
16         sniffed_socket.bind((nic, 0))
17
18         if os.name == 'nt':
19             sniffed_socket.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)
20             sniffed_socket.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)
21
22         data, _ = sniffed_socket.recvfrom(65535)
23
24         for p in data:
25             print("%02x" % p, end=" ")
26
27         if os.name == 'nt':
28             sniffed_socket.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... x root@ubuntu: /home/f... x root@ubuntu: /home/f... x root@ubuntu: /home/f... x
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 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 roorroorroorroorroorroorroorroorroorroorroorroorroorroorroorro
root@ubuntu: /home/famous/Desktop/TA/socket/assignment/assignment_8#
```

```
관리자: Anaconda Prompt
(base) C:\Users\Famous\OneDrive - 충북대학교\공부\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\공부\4학년 1학기\컴퓨터네트워크TA\8주차>
```

```
7 def dumpcode(buf):
8     print("%7s"% "offset ", end='')
9
10    for i in range(0, 16):
11        print("%02x " % i, end='')
12
13        if not (i%16-7):
14            print("- ", end='')
15
16    print("")
17
18    for i in range(0, len(buf)):
19        if not i%16:
20            print("0x%04x" % i, end= ' ')
21
22            print("%02x" % buf[i], end= ' ')
23
24            if not (i % 16 - 7):
25                print("- ", end='')
26
27            if not (i % 16 - 15):
28                print(" ")
29
30    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#
```

## 2. Raw Socket

---

### 2.6 Assignment 8

## Assignment #8

- 수업 Github assignment\_8에 있는 raw\_socket\_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(표지 제외)
- 팀 대표가 [barcel@naver.com](mailto:barcel@naver.com)으로 제출 (5.7일까지)
  - Title : [컴퓨터네트워크][학번][이름][과제\_N]
  - Content : github repo url

팀명 : 길동이네

팀원 : 홍길동(학번), 고길동(학번)