

ECE4016 Computer Networks Assignment 3

Network test tools

Yan Han 122090897

November 15, 2024

Contents

0. Environment	3
1 Network Test Tool Use	3
1.1 Commands inputs	3
1.2 Capture TCP packets	4
1.3 Capture UDP packets	7
1.4 Analyze protocol packets	8
1.5 Process of encapsulating and decapsulating packet	18

List of Tables

1	Summary of Common Network Commands	3
---	--	---

List of Figures

1	ifconfig	3
2	ping	4
3	nslookup	4
4	arp	4
5	netstat	5
6	traceroute	5
7	TCP sending packets	6
8	UDP sending packets	7
9	T-shark TCP sending packets	8
10	Wildshark TCP sending packets	9
11	T-shark TCP sending packets	12
12	Wildshark TCP sending packets	12
13	wildshark ARP sending packets	13
14	T-shark ARP sending packets	13
15	wildshark DNS sending packets	14
16	T-shark DNS sending packets	14
17	wildshark ICMP sending packets	15
18	T-shark ICMP sending packets	16
19	wildshark ICMP sending packets	17
20	T-shark ICMP sending packets	17

0. Environment

Ubuntu 22.04
Wireshark 4.0.10
macOS arm64

1 Network Test Tool Use

1.1 Commands inputs

Command	Primary Protocol	Purpose
ifconfig	System-level interaction	Configures and displays network interface settings.
ping	ICMP	Tests connectivity and measures RTT.
nslookup	DNS	Resolves domain names and queries DNS records.
arp	ARP	Maps IP addresses to MAC addresses on the local network.
netstat	TCP/UDP/ICMP	Monitors active connections, ports, and network statistics.
traceroute	ICMP/UDP	Traces the path packets take to a destination and measures hop latency.

Table 1: Summary of Common Network Commands

```
tsuzukii@tsuzukii: $ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
        inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
              ether 02:42:fe:1c:9c:54 txqueuelen 0 (以太网)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.44.128 netmask 255.255.255.0 broadcast 192.168.44.255
              inet6 fe80::58a2:917:e735:aa28 prefixlen 64 scopeid 0x20<link>
        ether 00:0c:29:d8:0e:36 txqueuelen 1000 (以太网)
        RX packets 2004 bytes 1816277 (1.8 MB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 1244 bytes 163495 (163.4 KB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
      inet 127.0.0.1 netmask 255.0.0.0
      inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (本地环回)
        RX packets 421 bytes 43805 (43.8 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 421 bytes 43805 (43.8 KB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figure 1: ifconfig

```
tsuzukii@tsuzukii:~$ ping www.baidu.com
PING www.a.shifen.com (157.148.69.80) 56(84) bytes of data.
64 bytes from 157.148.69.80 (157.148.69.80): icmp_seq=1 ttl=128 time=15.3 ms
64 bytes from 157.148.69.80 (157.148.69.80): icmp_seq=2 ttl=128 time=14.1 ms
64 bytes from 157.148.69.80 (157.148.69.80): icmp_seq=3 ttl=128 time=18.1 ms
64 bytes from 157.148.69.80 (157.148.69.80): icmp_seq=4 ttl=128 time=16.2 ms
64 bytes from 157.148.69.80 (157.148.69.80): icmp_seq=5 ttl=128 time=27.1 ms
```

Figure 2: ping

```
tsuzukii@tsuzukii:~$ nslookup
> www.baidu.com
Server:      127.0.0.53
Address:     127.0.0.53#53

Non-authoritative answer:
www.baidu.com canonical name = www.a.shifen.com.
Name:   www.a.shifen.com
Address: 157.148.69.80
Name:   www.a.shifen.com
Address: 157.148.69.74
Name:   www.a.shifen.com
Address: 2409:8c54:870:34e:0:ff:b024:1916
Name:   www.a.shifen.com
Address: 2409:8c54:870:67:0:ff:b0c2:ad75
```

Figure 3: nslookup

地址	类型	硬件地址	标志	Mask	接口
192.168.44.254	ether	00:50:56:fc:6a:43	C		ens33
_gateway	ether	00:50:56:e2:cf:cf	C		ens33

Figure 4: arp

1.2 Capture TCP packets

1. "354","20.304209","10.31.13.124","61.243.26.4","TCP","78","60439 → 443 [SYN]
Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=997099234 TSecr=0 SACK_PERM"
2. "363","20.336749","61.243.26.4","10.31.13.124","TCP","74","443 → 60439 [SYN,
ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1200 SACK_PERM TSval=3347572173 TSecr=997099234
WS=256"
3. "364","20.336798","10.31.13.124","61.243.26.4","TCP","66","60439 → 443 [ACK]
Seq=1 Ack=1 Win=131840 Len=0 TSval=997099267 TSecr=3347572173"
- 4."365","20.336960","10.31.13.124","61.243.26.4","TLSv1.3","583","Client Hello
(SNI=m7.music.126.net)"

Packet 1: SYN (Client to Server)

This packet is the initial SYN packet sent by the client (10.31.13.124) to establish a TCP connection with the server (61.243.26.4) on port 443. The client initiates the handshake with a sequence number of 0, indicating the start of the connection. The options include the maximum segment size (MSS) of 1460 bytes, the window scale factor of 64, and selective

```

tsuzuki@tsuzuki:~$ netstat
激活Internet连接 (w/o 服务器)
Proto Recv-Q Send-Q Local Address          Foreign Address        State
udp      0      0 tsuzuki:bootpc       192.168.44.254:bootps ESTABLISHED

活跃的UNIX域套接字 (w/o 服务器)
Proto RefCnt Flags     Type      State      I-Node    路径
unix   3      [ ]      流        已连接    31397    /run/user/1000/at-spi/bus
unix   3      [ ]      流        已连接    30019    /run/user/1000/wayland-0
unix   3      [ ]      流        已连接    31060
unix   3      [ ]      流        已连接    27895    /run/dbus/system_bus_socket
unix   3      [ ]      流        已连接    18290    /run/dbus/system_bus_socket
unix   3      [ ]      流        已连接    18602
unix   3      [ ]      流        已连接    17281
unix   3      [ ]      流        已连接    29054    /run/user/1000/bus
unix   3      [ ]      流        已连接    27905
unix   3      [ ]      流        已连接    34742
unix   3      [ ]      流        已连接    29082
unix   3      [ ]      流        已连接    27588    /run/user/1000/bus
unix   3      [ ]      流        已连接    29824
unix   3      [ ]      流        已连接    29394
unix   3      [ ]      流        已连接    31014    /run/dbus/system_bus_socket
unix   3      [ ]      流        已连接    29937
unix   3      [ ]      流        已连接    26350    /run/systemd/journal/stdout
unix   3      [ ]      流        已连接    21729    /run/systemd/journal/stdout
unix   3      [ ]      流        已连接    18566
unix   3      [ ]      流        已连接    28447    /run/user/1000/pulse/native
unix   3      [ ]      流        已连接    28173
unix   3      [ ]      流        已连接    29070
unix   3      [ ]      流        已连接    27512    /run/systemd/journal/stdout
unix   3      [ ]      流        已连接    31008

```

Figure 5: netstat

```

tsuzuki@tsuzuki:~$ traceroute www.baidu.com
traceroute to www.baidu.com (157.148.69.80), 30 hops max, 60 byte packets
1 _gateway (192.168.44.2)  0.113 ms  0.098 ms  0.103 ms
2 * * *
3 * * *
4 * * *
5 * * *
6 * * *
7 * * *
8 * * *
9 * * *
10 * * *
11 * * *
12 * * *
13 * * *
14 * * *
15 * * *
16 * * *
17 * * *^Z

```

Figure 6: traceroute

acknowledgment (SACK) support.

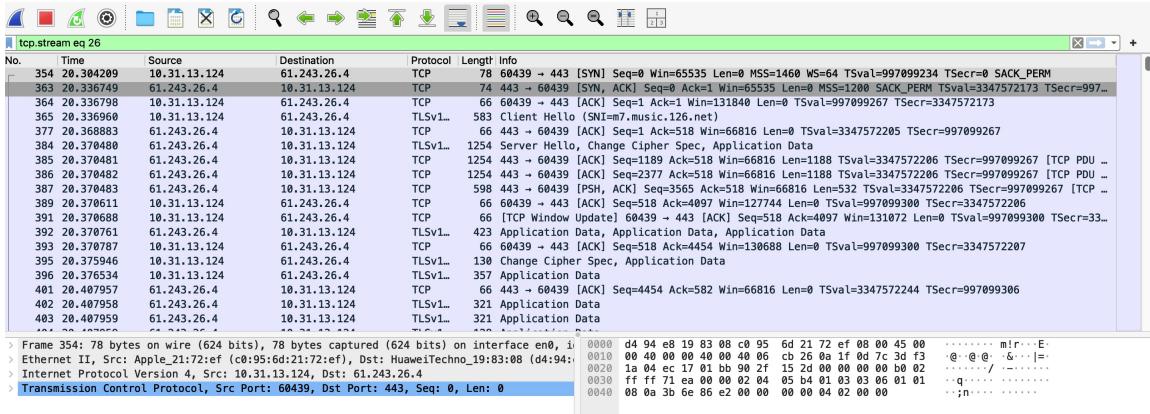


Figure 7: TCP sending packets

Packet 2: SYN, ACK (Server to Client)

This packet is sent by the server in response to the client's SYN packet, acknowledging the client's connection request. The sequence number is set to 0, and the acknowledgment number is 1, confirming the receipt of the client's SYN packet. The server specifies a maximum segment size (MSS) of 1200 bytes and a window scale factor of 256. The SACK option is enabled, and the timestamp reflects the server's round-trip time.

Packet 3: ACK (Client to Server)

This packet is the final acknowledgment from the client, confirming the receipt of the server's SYN-ACK packet. The acknowledgment number is set to 1, indicating the successful receipt of the server's SYN (with a sequence number of 0). The sequence number of 1 reflects the continuation of the TCP flow. No application data is included in this packet.

Packet 4: Client Hello (TLS Handshake)

Once the TCP connection is established, the client initiates the TLS handshake by sending a "Client Hello" message. The SNI (Server Name Indication) indicates that the client is attempting to connect to the server at `m7.music.126.net`. The message includes information about the supported encryption protocols and cipher suites, as well as other details necessary for establishing a secure encrypted connection.

Summary

- **Three-Way Handshake Process:**

- **Step 1:** The client sends a **SYN** packet to initiate the connection.
- **Step 2:** The server responds with a **SYN, ACK** packet to acknowledge the client's request.
- **Step 3:** The client sends an **ACK** packet to confirm the connection is established.

- **TLS Handshake Begins:** After the TCP connection is established, the client sends a **Client Hello** message to start the TLS handshake, preparing for secure communication. Then the data starts to transmit.

1.3 Capture UDP packets

```
1. "311","0.400531","127.0.0.1","127.0.0.1","UDP","78","54870 → 12345 Len=46"
2. "312","0.400631","127.0.0.1","127.0.0.1","UDP","62","12345 → 54870 Len=30"
3. "1117","1.403724","127.0.0.1","127.0.0.1","UDP","78","54870 → 12345 Len=46"
```

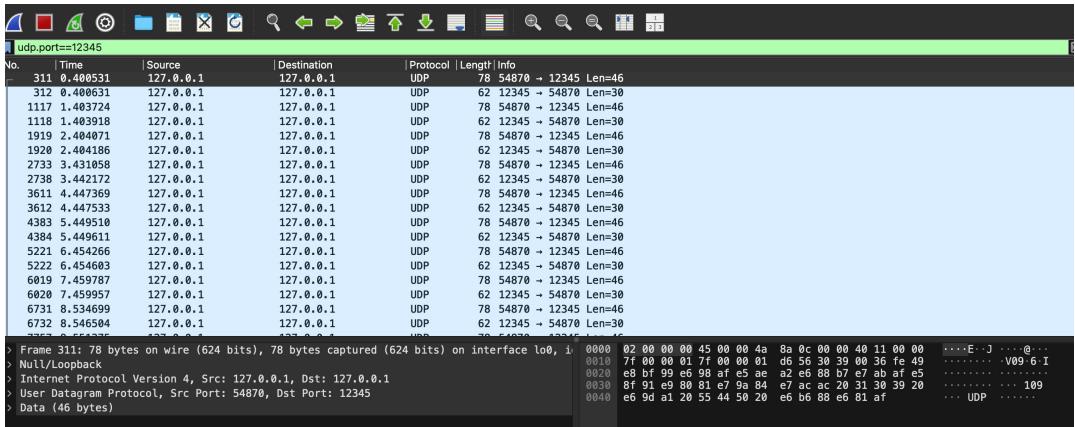


Figure 8: UDP sending packets

Unlike TCP, UDP (User Datagram Protocol) does not establish a connection with a three-way handshake. Instead, it operates in a connectionless manner, where each packet is sent independently, without the need for acknowledgment or retransmission. Here is an example of a basic UDP exchange:

Packet 1: Client to Server (UDP Request)

This packet is an initial UDP request sent by the client (127.0.0.1) to the server (127.0.0.1) on port 12345. UDP does not require a connection setup; thus, the client sends data directly to the server without prior handshakes. The client's source port is dynamically assigned (54870 in this case), and the packet length is 46 bytes.

Packet 2: Server to Client (UDP Response)

In response, the server sends a UDP packet back to the client on the client's source port (54870). This response packet contains 30 bytes of data. Since UDP is stateless, each packet exchange is independent, and the server is not required to maintain any connection information for the client.

Packet 3: Client to Server (Additional UDP Request)

The client sends another UDP request to the server on port 12345. Each request is treated as an independent transmission, so the client can continue to send packets without waiting for prior responses.

Summary

- Connectionless Transmission:** UDP does not establish a connection, unlike TCP's three-way handshake. Each packet is transmitted independently.
- Unreliable Delivery:** Since UDP does not provide acknowledgment, packets may be lost without notification. It is up to the application layer to handle any necessary retransmission.
- Efficient for Real-Time Applications:** UDP's low overhead makes it suitable for applications requiring speed and low latency, such as real-time audio and video streaming or online gaming.

1.4 Analyze protocol packets

TCP

```
Capturing on 'Wi-Fi: en0'
  1  0.000000 10.31.13.124 > 208.67.228.220 TCP 78 55806 > 443 [SYN] Seq=0 Win=65535 MSS=1460 WS=64 TStamp=4115658
  2  0.00006 10.31.13.124 > 111.124.200.65 TCP 54 55793 > 443 [ACK] Seq=1 Ack=1 Win=4096 Len=0
  3  0.00012000 10.31.13.124 > 113.248.75.249 TCP 78 55810 > 88 [SYN] Seq=0 Win=65535 MSS=1460 WS=64 TStamp=19237388
  4  0.022666 111.124.200.65 > 10.31.13.124 TCP 60 [TCP ACKed unseen segment] 443 > 55793 [ACK] Seq=1 Ack=2 Win=6 Len=0
  5  0.022400 111.124.200.65 > 113.248.75.249 TCP 54 55742 > 443 [ACK] Seq=1 Ack=1 Win=4096 Len=0
  6  0.103761 113.248.75.249 > 10.31.13.124 TCP 60 [TCP ACKed unseen segment] 443 > 55742 [ACK] Seq=1 Ack=2 Win=501 Len=0
  7  0.195327 10.31.13.124 > 113.248.75.249 TCP 54 55745 > 443 [ACK] Seq=1 Ack=1 Win=4096 Len=0
  8  0.195218 10.31.13.124 > 113.248.75.249 TCP 54 55744 > 443 [ACK] Seq=1 Ack=1 Win=4096 Len=0
  9  0.219637 113.248.75.249 > 10.31.13.124 TCP 60 [TCP ACKed unseen segment] 443 > 55744 [ACK] Seq=1 Ack=2 Win=501 Len=0
  10  0.221524 113.248.75.249 > 10.31.13.124 TCP 60 [TCP ACKed unseen segment] 443 > 55745 [ACK] Seq=1 Ack=2 Win=501 Len=0
  11  0.249411 10.31.13.124 > 113.248.75.249 TCP 54 55746 > 443 [ACK] Seq=1 Ack=1 Win=4096 Len=0
  12  0.249499 10.31.13.124 > 113.248.75.249 TCP 54 55743 > 443 [ACK] Seq=1 Ack=1 Win=4096 Len=0
  13  0.277779 113.248.75.249 > 10.31.13.124 TCP 60 [TCP ACKed unseen segment] 443 > 55743 [ACK] Seq=1 Ack=2 Win=501 Len=0
  14  0.277783 113.248.75.249 > 10.31.13.124 TCP 60 [TCP ACKed unseen segment] 443 > 55746 [ACK] Seq=1 Ack=2 Win=501 Len=0
  15  0.345647 10.31.13.124 > 113.248.75.249 TCP 78 [TCP Retransmission] 55810 > 88 [SYN] Seq=0 Win=65535 MSS=1460 WS=64 TStamp=1923739194 TSecr=0 SACK_PERM
  16  0.772830 10.31.13.124 > 20.42.73.28 TCP 54 55774 > 443 [ACK] Seq=1 Ack=1 Win=2048 Len=0
  17  0.772830 10.31.13.124 > 128.233.64.168 TCP 114 54531 > 34043 [PSH, ACK] Seq=1 Ack=1 Win=2048 Len=48 TStamp=35496786 TSecr=3640897676
  18  0.930163 10.31.13.124 > 128.233.64.168 TCP 129 54531 > 34043 [PSH, ACK] Seq=49 Ack=1 Win=2048 Len=43 TStamp=35496786
  19  0.945670 128.233.64.168 > 10.31.13.124 TCP 66 34043 > 54531 [ACK] Seq=1 Ack=49 Win=4 Len=0 TStamp=3640111552 TSecr=3549678677
  20  0.945675 128.233.64.168 > 10.31.13.124 TCP 66 34043 > 54531 [ACK] Seq=1 Ack=112 Win=4 Len=0 TStamp=3640111552 TSecr=3549678677
  21  0.984985 10.31.13.124 > 113.248.75.249 TCP 78 [TCP Retransmission] 55810 > 88 [SYN] Seq=0 Win=65535 MSS=1460 WS=64 TStamp=1923739837 TSecr=0 SACK_PERM
  22  0.984985 10.31.13.124 > 113.248.75.249 TCP 54 55774 > 443 [ACK] Seq=1 Ack=2 Win=16383 Len=0 TStamp=35496786 TSecr=3640897676
  23  1.184476 10.31.13.124 > 113.248.75.249 TCP 54 [TCP Dup ACK 5#1] 55742 > 443 [ACK] Seq=1 Ack=1 Win=4096 Len=0
  24  1.125642 113.248.75.249 > 10.31.13.124 TCP 60 [TCP Dup ACK 6#1] 443 > 55742 [ACK] Seq=1 Ack=2 Win=501 Len=0
  25  1.128891 128.233.64.168 > 10.31.13.124 TCP 274 34043 > 54531 [PSH, ACK] Seq=1 Ack=112 Win=4 Len=208 TStamp=3640111552 TSecr=3549678677
  26  1.128649 10.31.13.124 > 128.233.64.168 TCP 66 54531 > 34043 [ACK] Seq=112 Ack=289 Win=2044 Len=0 TStamp=3549678876
  27  1.229754 10.31.13.124 > 113.248.75.249 TCP 54 [TCP Dup ACK 8#1] 55744 > 443 [ACK] Seq=1 Ack=1 Win=4096 Len=0
  28  1.222142 10.31.13.124 > 113.248.75.249 TCP 54 [TCP Dup ACK 7#1] 55745 > 443 [ACK] Seq=1 Ack=1 Win=4096 Len=0
  29  1.246553 113.248.75.249 > 10.31.13.124 TCP 60 [TCP Dup ACK 9#1] 443 > 55744 [ACK] Seq=1 Ack=2 Win=501 Len=0
  30  1.258466 113.248.75.249 > 10.31.13.124 TCP 60 [TCP Dup ACK 10#1] 443 > 55745 [ACK] Seq=1 Ack=2 Win=501 Len=0
```

Figure 9: T-shark TCP sending packets

1. SYN (Client to Server):

- Source IP:** 10.31.13.124 (Client)
- Destination IP:** 61.243.26.4 (Server)
- Protocol:** TCP

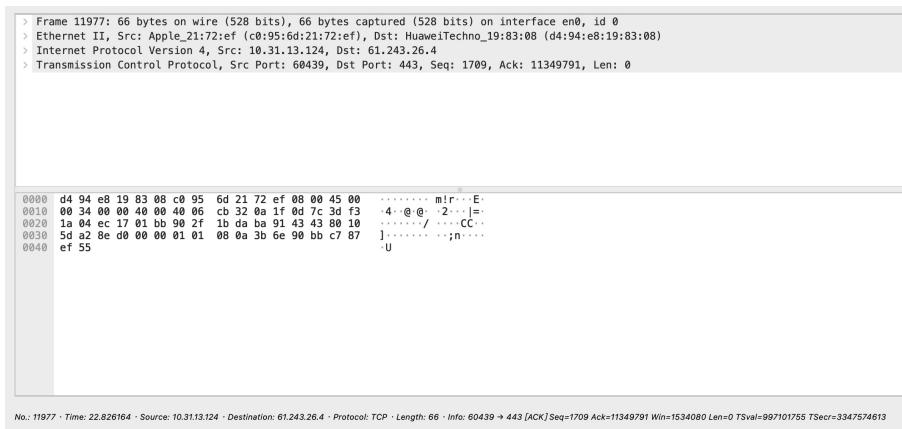


Figure 10: Wildshark TCP sending packets

- **Source Port:** 60439
- **Destination Port:** 443
- **Sequence Number (Seq):** 0
- **Acknowledgment Number (Ack):** 0
- **Window Size (Win):** 65535
- **Data Length (Len):** 0 bytes
- **Options:**
 - **MSS:** 1460 bytes
 - **Window Scale (WS):** 64
 - **Timestamp (TSval/TSecr):** 997099234 / 0
 - **SACK Permitted:** Yes

2. SYN, ACK (Server to Client):

- **Source IP:** 61.243.26.4 (Server)
- **Destination IP:** 10.31.13.124 (Client)
- **Protocol:** TCP
- **Source Port:** 443
- **Destination Port:** 60439
- **Sequence Number (Seq):** 0
- **Acknowledgment Number (Ack):** 1

- **Window Size (Win):** 65535
- **Data Length (Len):** 0 bytes
- **Options:**
 - **MSS:** 1200 bytes
 - **Window Scale (WS):** 256
 - **Timestamp (TSval/TSecr):** 3347572173 / 997099234
 - **SACK Permitted:** Yes

3. ACK (Client to Server):

- **Source IP:** 10.31.13.124 (Client)
- **Destination IP:** 61.243.26.4 (Server)
- **Protocol:** TCP
- **Source Port:** 60439
- **Destination Port:** 443
- **Sequence Number (Seq):** 1
- **Acknowledgment Number (Ack):** 1
- **Window Size (Win):** 131840
- **Data Length (Len):** 0 bytes
- **Timestamp (TSval/TSecr):** 997099267 / 3347572173

4. Client Hello (TLS Handshake):

- **Source IP:** 10.31.13.124 (Client)
- **Destination IP:** 61.243.26.4 (Server)
- **Protocol:** TLSv1.3
- **Source Port:** 60439
- **Destination Port:** 443
- **Data Length (Len):** 583 bytes
- **Content:** Client Hello (SNI=m7.music.126.net)

5. Server Application Data:

- **Source IP:** 61.243.26.4 (Server)

- **Destination IP:** 10.31.13.124 (Client)
- **Protocol:** TCP
- **Source Port:** 443
- **Destination Port:** 60439
- **Sequence Number (Seq):** 1189
- **Acknowledgment Number (Ack):** 518
- **Window Size (Win):** 66816 bytes
- **Data Length (Len):** 1188 bytes
- **Timestamp Value (TSval):** 3347572206
- **Timestamp Echo Reply (TSecr):** 997099267
- **Content:**

6. ACK (Client to Server):

- **Source IP:** 10.31.13.124 (Client)
- **Destination IP:** 61.243.26.4 (Server)
- **Protocol:** TCP
- **Source Port:** 60439
- **Destination Port:** 443
- **Sequence Number (Seq):** 518
- **Acknowledgment Number (Ack):** 4454
- **Window Size (Win):** 130688 bytes
- **Data Length (Len):** 0 bytes
- **Timestamp Value (TSval):** 997099300
- **Timestamp Echo Reply (TSecr):** 3347572207
- **Content:** TCP acknowledgment for previously received data

```
(base) felixyan9MacBook-Air-16 ~ % tshark -i lo -f "port 12345" -P
Capturing on 'Loopback: lo0' (127.0.0.1)
  1  0.000000 127.0.0.1 > 127.0.0.1  UDP 62 12345 > 12345 Len=46
  2  0.000174 127.0.0.1 > 127.0.0.1  UDP 62 12345 > 54870 Len=38
  3  1.005905 127.0.0.1 > 127.0.0.1  UDP 78 54870 > 12345 Len=46
  4  1.006078 127.0.0.1 > 127.0.0.1  UDP 62 12345 > 12345 Len=46
  5  2.008444 127.0.0.1 > 127.0.0.1  UDP 78 54870 > 12345 Len=46
  6  2.008568 127.0.0.1 > 127.0.0.1  UDP 62 12345 > 54870 Len=38
  7  3.013332 127.0.0.1 > 127.0.0.1  UDP 78 54870 > 12345 Len=46
  8  3.013500 127.0.0.1 > 127.0.0.1  UDP 62 12345 > 54870 Len=38
  9  4.016737 127.0.0.1 > 127.0.0.1  UDP 62 12345 > 12345 Len=46
 10  4.016737 127.0.0.1 > 127.0.0.1  UDP 62 12345 > 54870 Len=38
 11  5.021387 127.0.0.1 > 127.0.0.1  UDP 78 54870 > 12345 Len=46
 12  5.021688 127.0.0.1 > 127.0.0.1  UDP 62 12345 > 54870 Len=38
 13  6.024459 127.0.0.1 > 127.0.0.1  UDP 78 54870 > 12345 Len=46
 14  6.024646 127.0.0.1 > 127.0.0.1  UDP 62 12345 > 54870 Len=38
```

Figure 11: T-shark TCP sending packets

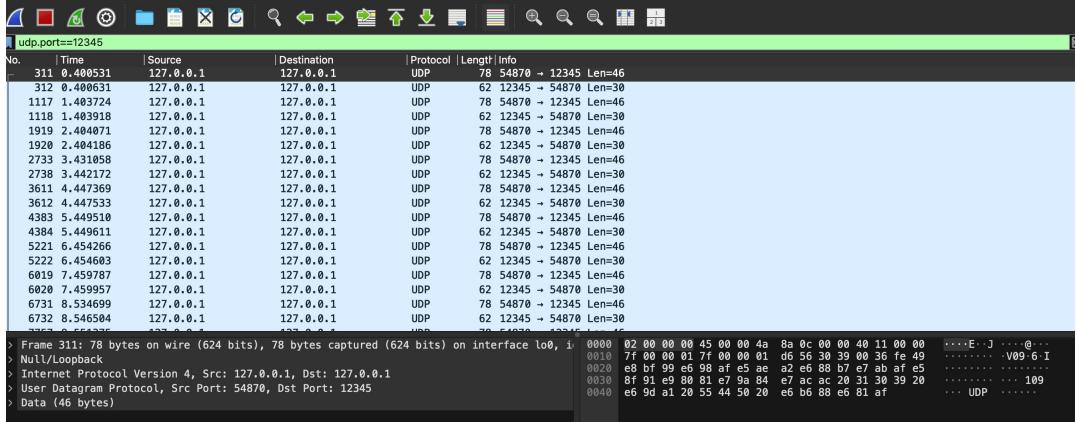


Figure 12: Wildshark TCP sending packets

UDP

1. UDP Request (Client to Server):

- **Source IP:** 127.0.0.1 (Client)
- **Destination IP:** 127.0.0.1 (Server)
- **Protocol:** UDP
- **Source Port:** 54870
- **Destination Port:** 12345
- **Data Length (Len):** 46 bytes
- **Content:** This packet is a UDP request sent by the client to the server on port 12345. UDP is connectionless, so no handshake is required, and the data is sent directly.

2. UDP Response (Server to Client):

- **Source IP:** 127.0.0.1 (Server)
- **Destination IP:** 127.0.0.1 (Client)
- **Protocol:** UDP
- **Source Port:** 12345

- **Destination Port:** 54870
- **Data Length (Len):** 30 bytes
- **Content:** This packet is a UDP response from the server to the client. Since UDP is stateless, the server sends the response without needing to maintain session information for the client.

ARP

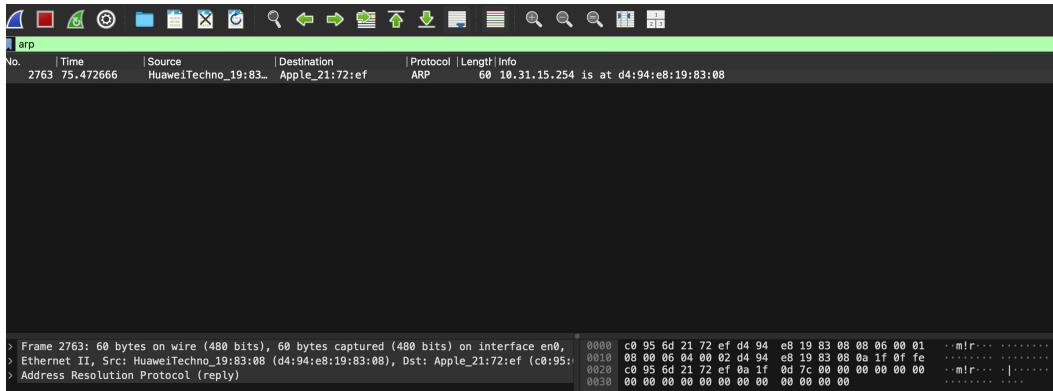


Figure 13: wildshark ARP sending packets

```
(base) felixyan@MacBook-Air-16 ~ % tshark -f "arp" -P
Capturing on 'Wi-Fi: en0'
1  0.000000 HuaweiTechno_19:83:08 > Apple_21:72:ef ARP 60 10.31.15.254 is at d4:94:e8:19:83:08
```

Figure 14: T-shark ARP sending packets

```
"2763", "75.472666", "HuaweiTechno_19:83:08", "Apple_21:72:ef", "ARP", "60", "10.31.15.254
is at d4:94:e8:19:83:08"
```

- **Source MAC Address:** HuaweiTechno_19:83:08(d4:94:e8:19:83:08)
- **Destination MAC Address:** Apple_21:72:ef
- **Protocol:** ARP
- **Data Length (Len):** 60 bytes
- **Operation:** Reply
- **Sender IP Address:** 10.31.15.254
- **Sender MAC Address:** d4:94:e8:19:83:08
- **Content:** Indicates that the IP address 10.31.15.254 is associated with the MAC address d4:94:e8:19:83:08

*1. ARP Request When a device (e.g., 10.31.15.1) needs to communicate with another device on the same network (e.g., 10.31.15.254), it first checks if the MAC address for the target IP is known.

- If the MAC address is unknown, the device sends a broadcast ARP request, asking, "Who has IP 10.31.15.254? Please send me your MAC address."

*2. ARP Response Devices receiving this broadcast check if their IP address matches 10.31.15.254.

- If the IP address matches, the device responds with an ARP reply, as seen in this packet.
- The response format is, "10.31.15.254 is at d4:94:e8:19:83:08," which is sent back directly to the requester.
- The requester then stores the IP-MAC mapping (e.g., 10.31.15.254 → d4:94:e8:19:83:08) in its ARP table for future communication.

DNS

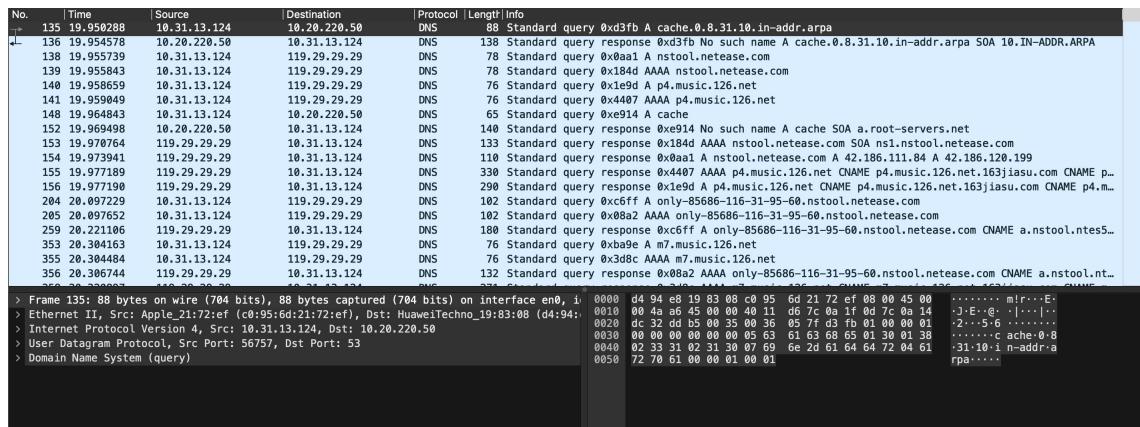


Figure 15: wildshark DNS sending packets

```
(base) felixyuan@MacBook-Air-16 ~ % tshark -f "port 53" -P
Capturing on 'wlan_11-16' (10.31.13.124)
1 0.000000 10.31.13.124 > 223.5.5.5 DNS 73 Standard query 0x5471 A www.baidu.com
2 0.009193 223.5.5.5 > 10.31.13.124 DNS 132 Standard query response 0x6471 A www.baidu.com CNAME www.a.shifen.com A 183.2.172.42 A 183.2.172.185
3 0.033780 10.31.13.124 > 223.5.5.5 DNS 76 Standard query 0xe636 AAAA pss.bdstatic.com
4 0.044026 10.31.13.124 > 223.5.5.5 DNS 76 Standard query 0x9977 A pss.bdstatic.com
5 0.044026 10.31.13.124 > 223.5.5.5 DNS 68 Standard query 0x9977 AAAA pss.bdstatic.com CNAME pss.bdstatic.com.abdydns.com CNAME opencdnbdps...
ss.jomodns.com AAAA 240e:97d1:10:1a80:1:b73d:b124 AAAA 240e:94:988:21:0:7af:1:824
6 0.055000 223.5.5.5 > 10.31.13.124 DNS 183 Standard query response 0x9977 A pss.bdstatic.com CNAME pss.bdstatic.com.a.bdydns.com CNAME opencdnbdps...
jomodns.com A 111.177.8.36 A 121.14.156.36
7 0.629796 10.31.13.124 > 223.5.5.5 DNS 82 Standard query 0xa673 AAAA hectorstatic.baidu.com
8 0.630000 10.31.13.124 > 223.5.5.5 DNS 82 Standard query 0xa673 AAAA hectorstatic.baidu.com
9 0.639781 223.5.5.5 > 10.31.13.124 DNS 254 Standard query response 0xa673 AAAA hectorstatic.baidu.com CNAME hectorstatic.baidu.com.a.bdydns.com CNAM...
E opencdnbdv.jomodns.com CNAME opencdnbdv.bd.woaa.cn AAAA 2409:876:5088:ed::b7ff:fd2d4 A hectorstatic.baidu.com CNAME hectorstatic.baidu.com.a.bdydns.com CNAME o...
10 0.642078 223.5.5.5 > 10.31.13.124 DNS 238 Standard query response 0xb4ed A hectorstatic.baidu.com CNAME hectorstatic.baidu.com.a.bdydns.com CNAME o...
11 0.642078 223.5.5.5 > 10.31.13.124 DNS 238 Standard query response 0xb4ed A hectorstatic.baidu.com CNAME hectorstatic.baidu.com.a.bdydns.com CNAME o...
12 0.655195 10.31.13.124 > 223.5.5.5 DNS 73 Standard query 0xbfb5 AAAA mbd.baidu.com
13 0.665799 223.5.5.5 > 10.31.13.124 DNS 132 Standard query response 0xdcd3 A mbd.baidu.com CNAME mbd.n.shifen.com A 14.215.182.57 A 14.215.182.19
14 0.665715 223.5.5.5 > 10.31.13.124 DNS 156 Standard query response 0xbfb5 AAAA mbd.baidu.com CNAME mbd.n.shifen.com AAAAA 240e:ff:e020:9ab:0:ff:b03a:...
24b5 AAAA 240e:ff:e020:957:0:ff:b02b:6114
```

Figure 16: T-shark DNS sending packets

"135", "19.950288", "10.31.13.124", "10.20.220.50", "DNS", "88", "Standard query 0xd3fb A cache.0.8.31.10.in-addr.arpa"

- **Source IP:** 10.31.13.124 (Client)
- **Destination IP:** 10.20.220.50 (DNS Server)
- **Protocol:** DNS
- **Source Port:** 60439
- **Destination Port:** 53
- **Transaction ID:** 0xd3fb
- **Flags:** Standard query
- **Query Type:** A (Address record)
- **Query Name:** cache.0.8.31.10.in-addr.arpa
- **Data Length (Len):** 88 bytes
- **Content:** DNS standard query for reverse DNS lookup

This packet is a DNS query sent by the client to a DNS server at IP address 10.20.220.50. The query is a standard request to resolve the reverse DNS for the IP address “10.31.8.0” by querying the “cache.0.8.31.10.in-addr.arpa” domain. The transaction ID 0xd3fb uniquely identifies this DNS query to help match it with the server’s response. The query type “A” requests the IPv4 address associated with this reverse DNS name.

ICMP

"85", "5.858982", "10.31.13.124", "119.29.29.29", "ICMP", "70", "Destination unreachable (Port unreachable)"

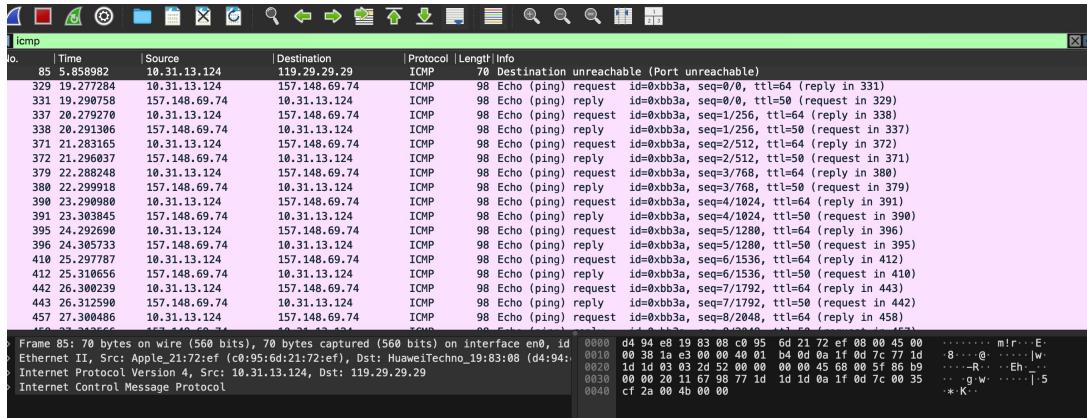


Figure 17: wildshark ICMP sending packets

- **Source IP:** 119.29.29.29

```
[base] Felixyan@MacBook-Air-16 ~ % tshark -f "icmp" -P
Capturing on 'Wi-Fi: en0'
  1. 0.000000 10.31.13.124 > 183.240.98.198 ICMP 98 Echo (ping) request id=0x8627, seq=0/0, ttl=64
  2. 0.071112 183.240.98.198 > 10.31.13.124 ICMP 98 Echo (ping) reply id=0x8627, seq=0/0, ttl=47 (request in 1)
  3. 1.084895 10.31.13.124 > 183.240.98.198 ICMP 98 Echo (ping) request id=0x8627, seq=1/256, ttl=64
  4. 1.076464 183.240.98.198 > 10.31.13.124 ICMP 98 Echo (ping) reply id=0x8627, seq=1/256, ttl=46 (request in 3)
  5. 2.083377 10.31.13.124 > 183.240.98.198 ICMP 98 Echo (ping) request id=0x8627, seq=2/512, ttl=46
  6. 2.083382 183.240.98.198 > 10.31.13.124 ICMP 98 Echo (ping) reply id=0x8627, seq=2/512, ttl=47 (request in 5)
  7. 3.011816 10.31.13.124 > 183.240.98.198 ICMP 98 Echo (ping) request id=0x8627, seq=3/768, ttl=64
  8. 4.016480 10.31.13.124 > 183.240.98.198 ICMP 98 Echo (ping) reply id=0x8627, seq=4/1024, ttl=64
  9. 4.016480 10.31.13.124 > 183.240.98.198 ICMP 98 Echo (ping) request id=0x8627, seq=4/1024, ttl=64 (request in 8)
  10. 5.023884 10.31.13.124 > 183.240.98.198 ICMP 98 Echo (ping) reply id=0x8627, seq=5/1280, ttl=64
  11. 5.023897 183.240.98.198 > 10.31.13.124 ICMP 98 Echo (ping) reply id=0x8627, seq=5/1280, ttl=47 (request in 10)
```

Figure 18: T-shark ICMP sending packets

- **Destination IP:** 10.31.13.124
- **Protocol:** ICMP
- **Data Length (Len):** 70 bytes
- **ICMP Type:** 3 (Destination Unreachable)
- **ICMP Code:** 3 (Port Unreachable)
- **Content:** This packet indicates that the destination port on the target machine (10.31.13.124) is unreachable. The originating host (119.29.29.29) attempted to send data to a closed port, and this ICMP message was generated in response to inform the sender of the issue.

This ICMP message is a “Destination Unreachable” response with a code indicating that the specific port on the destination is unreachable. This type of message typically occurs when a device tries to reach a UDP port on a remote host that is not open or actively listening, and the host replies with this ICMP packet to notify the sender.

HTTP

1. "113", "4.357961", "10.31.13.124", "140.207.56.109", "HTTP", "1035", "POST /mmtls/077e9958 HTTP/1.1 "
2. "129", "4.521459", "140.207.56.109", "10.31.13.124", "HTTP", "228", "HTTP/1.1 200 OK "

HTTP POST Request:

- **Source IP:** 10.31.13.124 (Client)
- **Destination IP:** 140.207.56.109 (Server)
- **Protocol:** HTTP
- **Data Length (Len):** 1035 bytes
- **Request Method:** POST
- **Request URI:** /mmtls/077e9958
- **HTTP Version:** HTTP/1.1

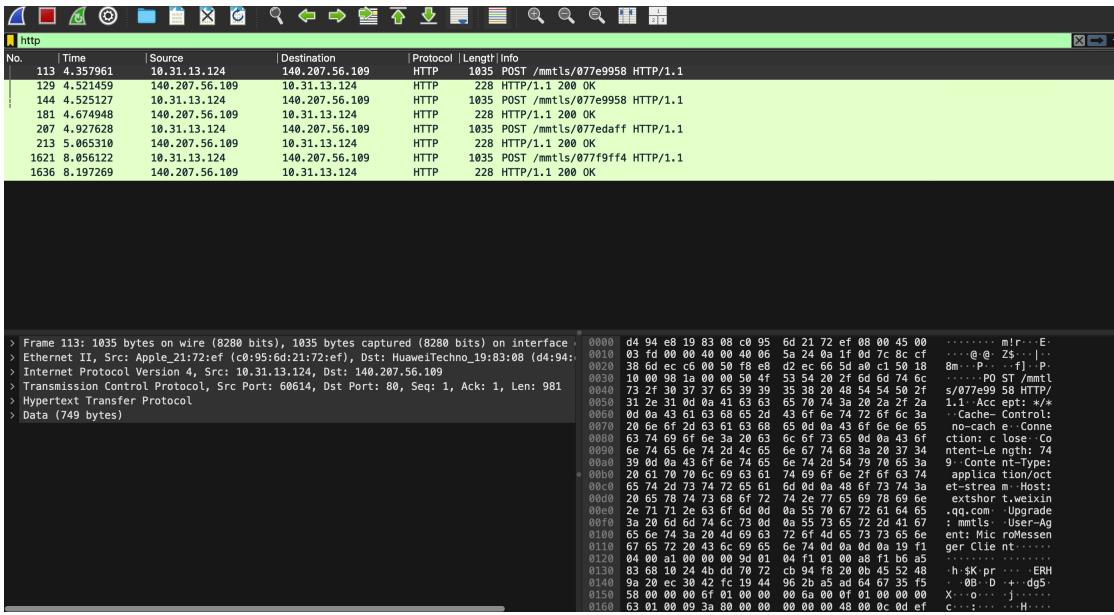


Figure 19: wildshark ICMP sending packets

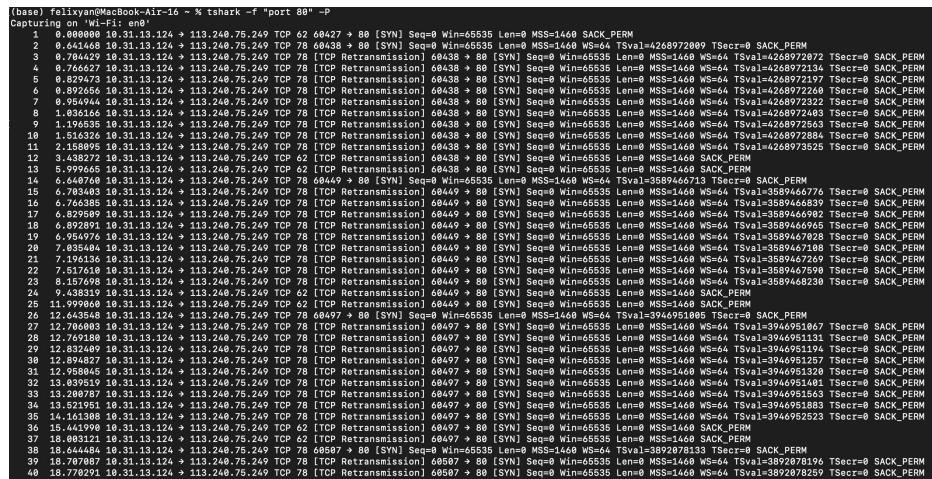


Figure 20: T-shark ICMP sending packets

- Content:** This packet represents an HTTP POST request from the client to the server, targeting the URI /mmtls/077e9958. The client is attempting to send data to the server as part of this request.

HTTP Response (200 OK):

- Source IP:** 140.207.56.109 (Server)
- Destination IP:** 10.31.13.124 (Client)
- Protocol:** HTTP
- Data Length (Len):** 228 bytes

- **Status Code:** 200 OK
- **HTTP Version:** HTTP/1.1
- **Content:** This packet is an HTTP response from the server, with a status code 200 OK, indicating that the server has successfully processed the client's POST request.

These packets represent an HTTP request-response pair, where the client sends an HTTP POST request to the server (Packet 113), and the server responds with a successful HTTP 200 OK status (Packet 129). This sequence indicates that the data sent by the client was successfully received and processed by the server.

Difference between real life and textbook

The packets I capture may not look exactly like what I learned in textbooks because real-world networks can be unpredictable. Here are a few simple reasons:

- **1. Network Issues:** Packets can be lost, delayed, or arrive out of order due to congestion or different paths in the network.
- **2. Device Optimizations:** Some devices change packets to improve speed or efficiency, such as combining smaller packets or handling checksums differently.
- **3. Security Layers:** Firewalls, NAT, and proxies may alter packets to protect the network, or encryption can hide the data.
- **4. Protocol Variations:** Real networks may use optional or extended parts of protocols not always covered in textbooks.

These factors mean that captured packets often have differences, making real-world packet data look less "perfect" than in theory.

1.5 Process of encapsulating and decapsulating packet

Encapsulation Process

- **Application Layer:** Data is generated at the application layer. For example, in an HTTP request, the application layer creates a message that includes the requested URL, headers, and additional data. This data is passed to the transport layer.
- **Transport Layer:** The transport layer (e.g., TCP or UDP) adds a header to the application data. This header includes:
 - **Source Port and Destination Port:** Identifies the sender and receiver processes.
 - **Sequence Number (for TCP):** Ensures data is delivered in the correct order.
 - **Checksum:** Provides error-checking functionality.

- **Network Layer:** The network layer (e.g., IP) adds an IP header to the segment or datagram. This header contains:
 - **Source and Destination IP Addresses:** Used to route the data to its intended destination.
 - **Other Fields:** Such as protocol type and time-to-live (TTL) to manage routing and prevent infinite loops.

The resulting data unit at this stage is called a "packet."

Decapsulation Process (From Packet to Data)

- **Network Layer:** The network layer receives the packet and removes the IP header. It interprets:
 - **Source and Destination IP Addresses:** Verifies if the packet is intended for the local host.
 - **Protocol Field:** Indicates the transport layer protocol (e.g., TCP or UDP).
- **Transport Layer:** The transport layer removes its header. It uses:
 - **Port Numbers:** Identifies the target application process.
 - **Sequence Numbers (for TCP):** Ensures the data is in order.
 - **Checksum:** Verifies data integrity.

The application data is extracted and passed to the application layer.

- **Application Layer:** The application layer processes the data.