ECE4016 Assignment 03: Part 1. Network Test Experiment

Jiarui Chen 120090361

Linux basic network test commands

1. ifconfig

This command is used to view and change the configuration of the network interface on the system.

Syntax:

```
ifconfig [-v] [-a] [-s] [interface]
```

Directly running `ifconfig` with no parameter displays information about all network interfaces currently in operation.

```
mumeicc@CC-VM:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::9155:86e5:8f21:4b94  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:b8:9c:67  txqueuelen 1000  (Ethernet)
        RX packets 60241  bytes 60157486 (60.1 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 34638  bytes 2279035 (2.2 MB)
        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  (Local Loopback)
        RX packets 762  bytes 87807 (87.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 762  bytes 87807 (87.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Here, `enp0s3` and `lo` are active network interfaces on the system.

In the output, we have

- `mtu`: maximum transmission unit
- `inet`, `netmask`, `broadcast` and `inet6`: IPv4, netmask, broadcast and IPv6 address
- `txqueuelen`: length of transmit queue
- `ether`: hardware address

Running `ifconfig` with parameter will have

- Viewing the configuration of all interfaces: `ifconfig -a`

```
mumeicc@CC-VM:~$ ifconfig -a
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::9155:86e5:8f21:4b94  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:b8:9c:67  txqueuelen 1000  (Ethernet)
        RX packets 2584  bytes 2025523 (2.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2979  bytes 246987 (246.9 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  (Local Loopback)
        RX packets 421  bytes 45342 (45.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 421  bytes 45342 (45.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

- Viewing the configuration of a specific interface: `ifconfig [interface]`

```
mumeicc@CC-VM:~$ ifconfig enp0s3
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::9155:86e5:8f21:4b94  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:b8:9c:67  txqueuelen 1000  (Ethernet)
        RX packets 4354  bytes 2376883 (2.3 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5213  bytes 473207 (473.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

2. ping

This command is used to test the reachability of a host on an IP network by sending an ICMP echo request to the target host and waiting for an ICMP echo reply. Based on **ICMP (Internet Control Message Protocol)**.

We can simply run the command by `ping [address]`:

```
mumeicc@CC-VM:~$ ping www.bilibili.com
PING a.w.bilicdn1.com (14.17.92.70) 56(84) bytes of data.
64 bytes from 14.17.92.70 (14.17.92.70): icmp_seq=1 ttl=48 time=6.59 ms
64 bytes from 14.17.92.70 (14.17.92.70): icmp_seq=2 ttl=48 time=7.10 ms
64 bytes from 14.17.92.70 (14.17.92.70): icmp_seq=3 ttl=48 time=6.35 ms
64 bytes from 14.17.92.70 (14.17.92.70): icmp_seq=4 ttl=48 time=6.63 ms
64 bytes from 14.17.92.70 (14.17.92.70): icmp_seq=5 ttl=48 time=6.61 ms
64 bytes from 14.17.92.70 (14.17.92.70): icmp_seq=6 ttl=48 time=6.06 ms
^C
--- a.w.bilicdn1.com ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5009ms
rtt min/avg/max/mdev = 6.062/6.556/7.099/0.313 ms
```

After sending the request, we receive lines contain the reply messages, including IP, count of test, TTL and RTT (round trip time).

After interrupting by `Ctrl+C`, we have the overall data, including packet number, loss rate, total time and RTT status (min, average, max and mdev RTT).

We can specify the number of package sent to the host by `ping -c [number] [address]`, for example:

```
mumeicc@CC-VM:~$ ping -c 5 www.bilibili.com
PING a.w.bilicdn1.com (14.17.92.74) 56(84) bytes of data.
64 bytes from 14.17.92.74 (14.17.92.74): icmp_seq=1 ttl=48 time=7.28 ms
64 bytes from 14.17.92.74 (14.17.92.74): icmp_seq=2 ttl=48 time=7.57 ms
64 bytes from 14.17.92.74 (14.17.92.74): icmp_seq=3 ttl=48 time=7.35 ms
64 bytes from 14.17.92.74 (14.17.92.74): icmp_seq=4 ttl=48 time=7.36 ms
64 bytes from 14.17.92.74 (14.17.92.74): icmp_seq=5 ttl=48 time=7.24 ms

--- a.w.bilicdn1.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 7.241/7.359/7.565/0.112 ms
```

3.nslookup

This command is used to querying the mapping between domain name and IP address, or other DNS records. Based on **DNS protocol**.

We can run it by `nslookup [address]`:

```
mumeicc@CC-VM:~$ 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: 14.215.177.39
Name:   www.a.shifen.com
Address: 14.215.177.38
```

In the output, we have

- `Server, Address`: the DNS server and address it is querying
- `Name, Address`: the CNAME record and IP address

Directly run `nslookup` will be an interactive query:

```
mumeicc@CC-VM:~$ 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: 14.215.177.38
Name:   www.a.shifen.com
Address: 14.215.177.39
> exit
```

4. arp

This command manipulates the system's **ARP (Address Resolution Protocol)** cache. The primary function of ARP is to resolve the IP address of a system to its address.

Syntax:

```
arp [-v] [-i if] [-H type] [-a] [hostname]
```

Example:

```
mumeicc@CC-VM:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
_gateway                 ether   52:54:00:12:35:02   C                     enp0s3
```

This checks the ARP info: IP address and Hardware address mapping, corresponding network interface.

5. netstat

This command displays network connections for Transmission Control Protocol, a number of network interfaces and network protocol statistics.

Example:

```
mumeicc@CC-VM:~$ netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 CC-VM:37912             api.snapcraft.io:https  ESTABLISHED
tcp        0      0 CC-VM:57240             api.snapcraft.io:https  ESTABLISHED
tcp        0      0 CC-VM:49492             api.snapcraft.io:https  ESTABLISHED
udp        0      0 CC-VM:bootpc            _gateway:bootps         ESTABLISHED
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  2      [ ]         DGRAM                    20065    /run/user/1000/systemd/
notify
unix  3      [ ]         DGRAM      CONNECTED     15765    /run/systemd/notify
unix  2      [ ]         DGRAM                    15783    /run/systemd/journal/sy
slog
unix  17     [ ]         DGRAM      CONNECTED     15792    /run/systemd/journal/de
v-log
unix  9      [ ]         DGRAM      CONNECTED     15794    /run/systemd/journal/so
cket
unix  3      [ ]         STREAM     CONNECTED     57083
unix  3      [ ]         STREAM     CONNECTED     22390    /run/dbus/system_bus_so
cket
unix  3      [ ]         STREAM     CONNECTED     21391
unix  3      [ ]         STREAM     CONNECTED     19451    /run/systemd/journal/st
dout
unix  3      [ ]         STREAM     CONNECTED     23959
```

(Only a part of the output here)

It shows the statistics of all active Internet connections and UNIX domain sockets.

Adding parameter `-s` will display the statistics of network protocols.

```
mumeicc@CC-VM:~$ netstat -s
Ip:
    Forwarding: 2
    83162 total packets received
    1 with invalid addresses
    0 forwarded
    0 incoming packets discarded
    83159 incoming packets delivered
    68127 requests sent out
    20 outgoing packets dropped
Icmp:
    82 ICMP messages received
    0 input ICMP message failed
    ICMP input histogram:
        destination unreachable: 41
        echo replies: 41
    125 ICMP messages sent
    0 ICMP messages failed
    ICMP output histogram:
        destination unreachable: 43
        echo requests: 82
IcmpMsg:
        InType0: 41
        InType3: 41
        OutType3: 43
```

(Only a part of output here)

6. traceroute

This command displays possible routes and measuring transit delays of packets across an IP network. It is based on **ICMP (Internet Control Message Protocol)**.

```
mumeicc@CC-VM:~$ traceroute -q 1 -m 16 example.com
traceroute to example.com (93.184.216.34), 16 hops max, 60 byte packets
 1  _gateway (10.0.2.2)  0.264 ms
 2  *
 3  *
 4  *
 5  *
 6  *
 7  *
 8  *
 9  *
10  *
11  *
12  *
13  *
14  *
15  *
16  *
```

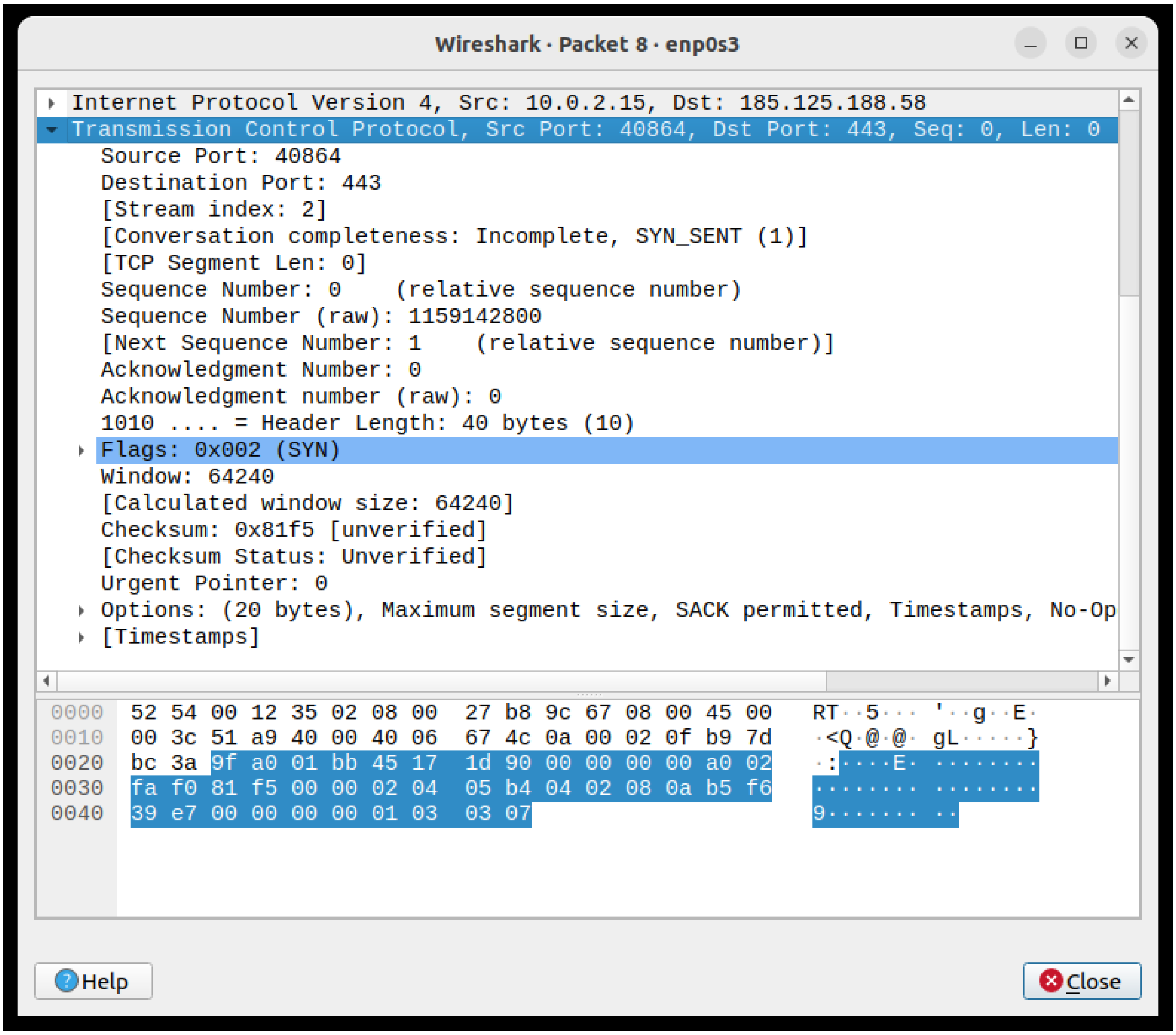
In the example, we use `traceroute` to find routes to `example.com`, with 16 maximum hops, 1 query per hop.

Wireshark Packet Capture (Including TCP/UDP)

1. TCP

TCP capture by Wireshark

Using Wireshark to capture TCP packet in Linux:



TCP header here is 40 bytes:

- Source port: 2 bytes, here `0x9fa0`
- Destination port: 2 bytes, here `0x01bb`
- Sequence number: 4 bytes
- Acknowledge number: 4 bytes
- Flags: 2 bytes, here `0xa002`
- Window: 2 bytes, here `0xfaf0`
- Checksum: 2 bytes, here `0x81f5`
- Urgent point: 2 bytes, here `0x0000`
- Options: 20 bytes

TCP capture by tshark command

```
mumeicc@CC-VM:~$ tshark -i enp0s3 -f "tcp"
Capturing on 'enp0s3'
 ** (tshark:4642) 18:38:42.341687 [Main MESSAGE] -- Capture started.
 ** (tshark:4642) 18:38:42.341790 [Main MESSAGE] -- File: "/tmp/wireshark_enp0s3
TV8TW1.pcapng"
    1 0.000000000    10.0.2.15 → 216.239.36.117 TCP 74 48622 → 443 [SYN] Seq=0 W
in=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1410925909 TSecr=0 WS=128
    2 0.000027271    10.0.2.15 → 216.239.36.117 TCP 74 48636 → 443 [SYN] Seq=0 W
in=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1410925909 TSecr=0 WS=128
    3 16.127986325    10.0.2.15 → 216.239.36.117 TCP 74 [TCP Retransmission] [TC
P Port numbers reused] 48636 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PER
M=1 TSval=1410942037 TSecr=0 WS=128
    4 16.128008336    10.0.2.15 → 216.239.36.117 TCP 74 [TCP Retransmission] [TC
P Port numbers reused] 48622 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PER
M=1 TSval=1410942037 TSecr=0 WS=128
```

TCP Connection Process

The packets sending between server and client captured by Wireshark:

Time	Source	Destination	Protocol	Length	Info
141.184199927	10.0.2.15	142.251.43.2	TCP	74	42554 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_P
141.225287378	142.251.43.2	10.0.2.15	TCP	60	443 → 42554 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=
141.225315491	10.0.2.15	142.251.43.2	TCP	54	42554 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0

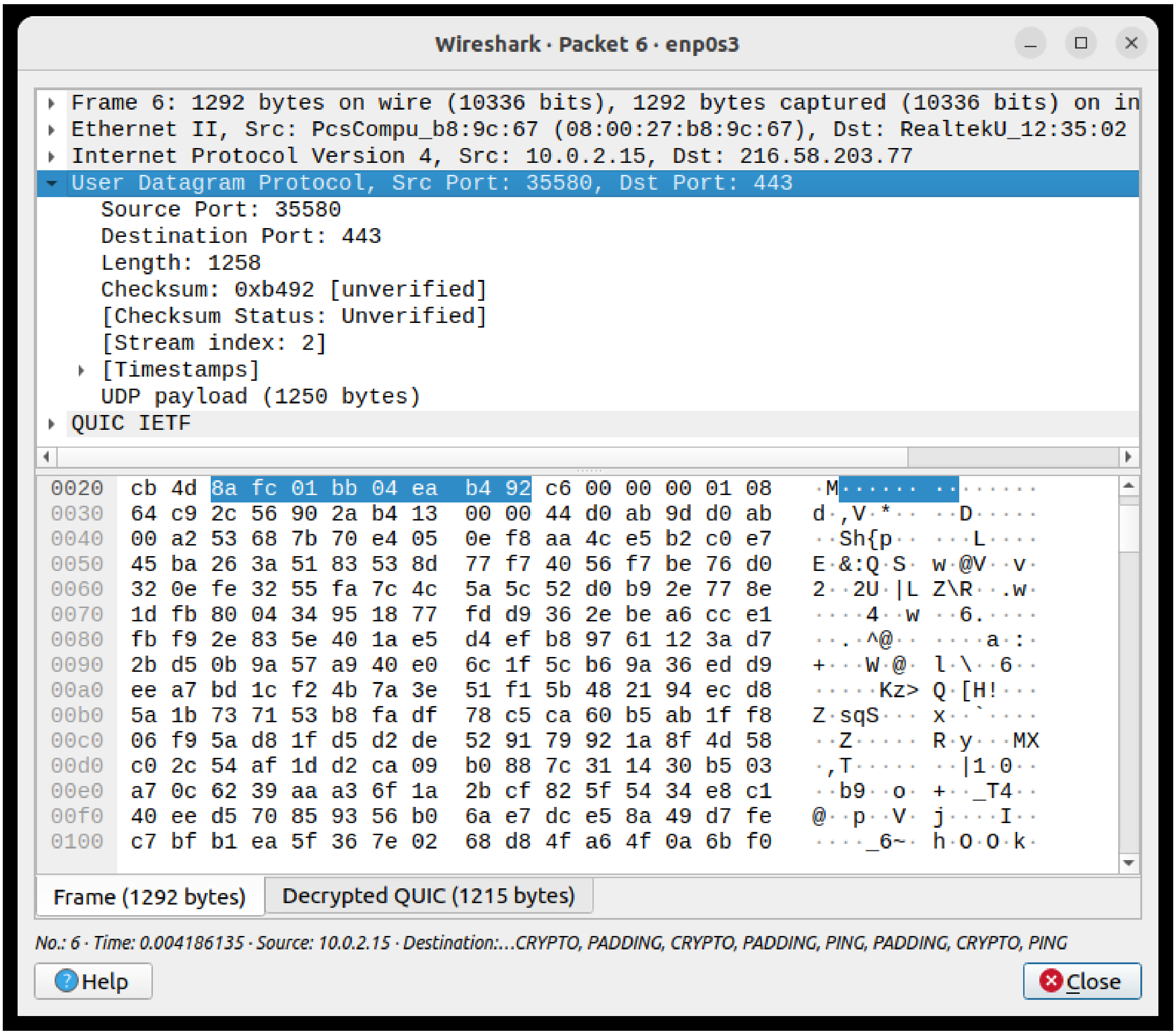
Here, client IP is `10.0.2.15` and server IP is `142.251.43.2`.

The three-way handshake process:

1. **SYN**: The client sends a SYN to server, and set the sequence number to a random value A (0 here)
2. **SYN-ACK**: The server replies the client with SYN-ACK. The acknowledge number is set to a number greater than A (1 here) and sequence number is set to a random number B (0 here)
3. **ACK**: The client send ACK back to the server. The sequence number is set to SYN-ACK's Ack value and Ack number is set to a number greater than B (1 here)

2. UDP

UDP capture by Wireshark



UDP header is 8 bytes here:

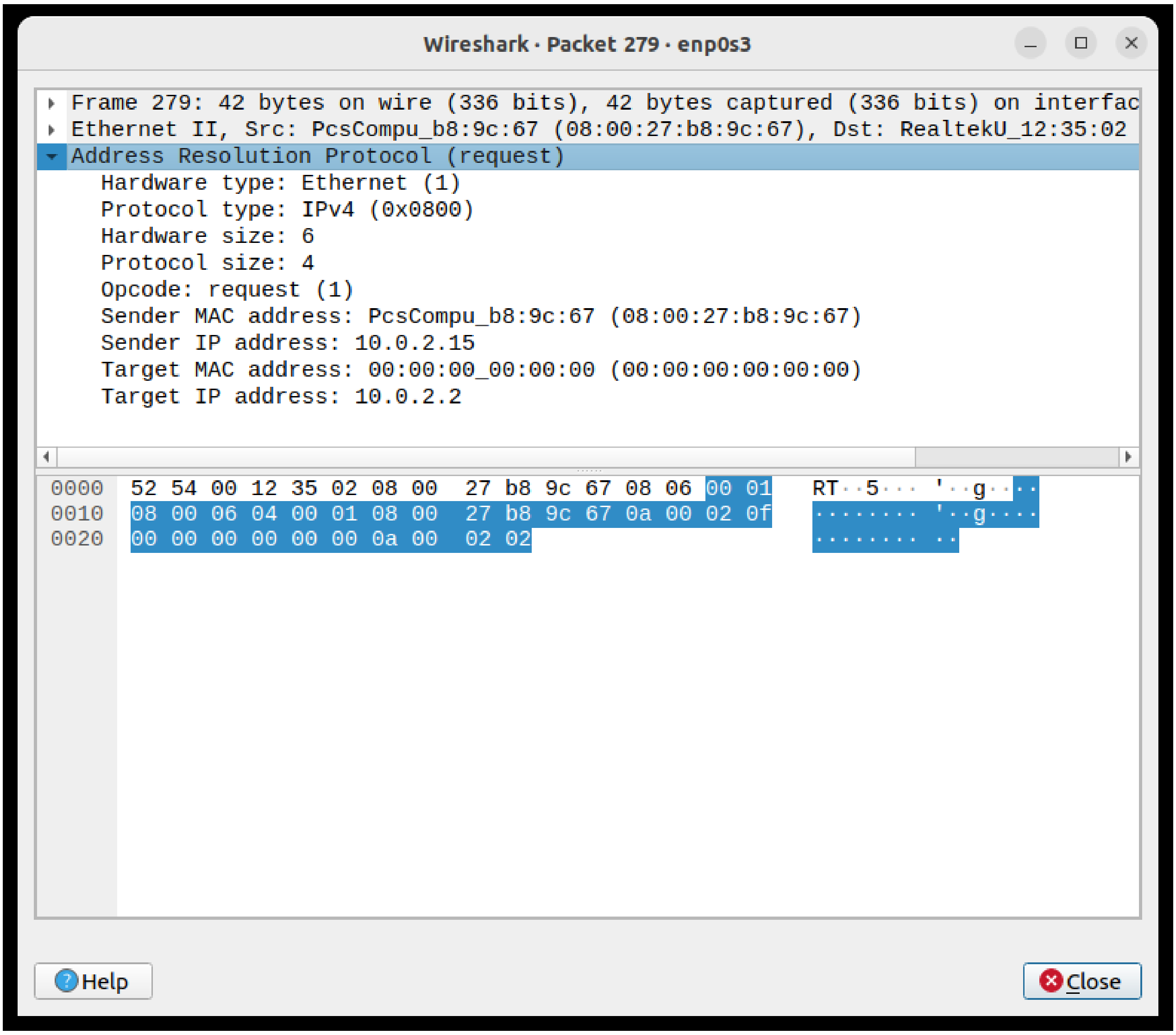
- Source port: 2 bytes, here `0x8afc`
- Destination port: 2 bytes, here `0x01bb`
- Length: 2 bytes, here `0x04ea`
- Checksum: 2 bytes, here `0xb492`

UDP capture by tshark command

```
mumeicc@CC-VM:~$ tshark -i enp0s3 -f "udp"
Capturing on 'enp0s3'
 ** (tshark:4745) 18:51:18.852771 [Main MESSAGE] -- Capture started.
 ** (tshark:4745) 18:51:18.852965 [Main MESSAGE] -- File: "/tmp/wireshark_enp0s3
B9J5W1.pcapng"
    1 0.000000000    10.0.2.15 → 10.20.232.47 DNS 100 Standard query 0xde40 A co
nnectivity-check.ubuntu.com OPT
    2 0.004005616 10.20.232.47 → 10.0.2.15    DNS 244 Standard query response 0x
de40 A connectivity-check.ubuntu.com A 185.125.190.18 A 91.189.91.49 A 35.232.11
1.17 A 185.125.190.17 A 34.122.121.32 A 185.125.190.49 A 35.224.170.84 A 91.189.
91.48 A 185.125.190.48 OPT
    3 20.013638940    10.0.2.15 → 10.20.232.47 DNS 100 Standard query 0xbadc A c
onnectivity-check.ubuntu.com OPT
    4 20.018299743 10.20.232.47 → 10.0.2.15    DNS 244 Standard query response 0
xbadc A connectivity-check.ubuntu.com A 35.224.170.84 A 185.125.190.17 A 185.125
.190.18 A 91.189.91.49 A 185.125.190.48 A 91.189.91.48 A 185.125.190.49 A 34.122
.121.32 A 35.232.111.17 OPT
    5 20.425350271    10.0.2.15 → 10.20.232.47 DNS 100 Standard query 0x281b A l
ocation.services.mozilla.com OPT
    6 20.425452201    10.0.2.15 → 10.20.232.47 DNS 100 Standard query 0xe539 AAA
A location.services.mozilla.com OPT
    7 20.433641579 10.20.232.47 → 10.0.2.15    DNS 248 Standard query response 0
```

3. ARP

ARP capture by Wireshark



ARP header here is 28 bytes:

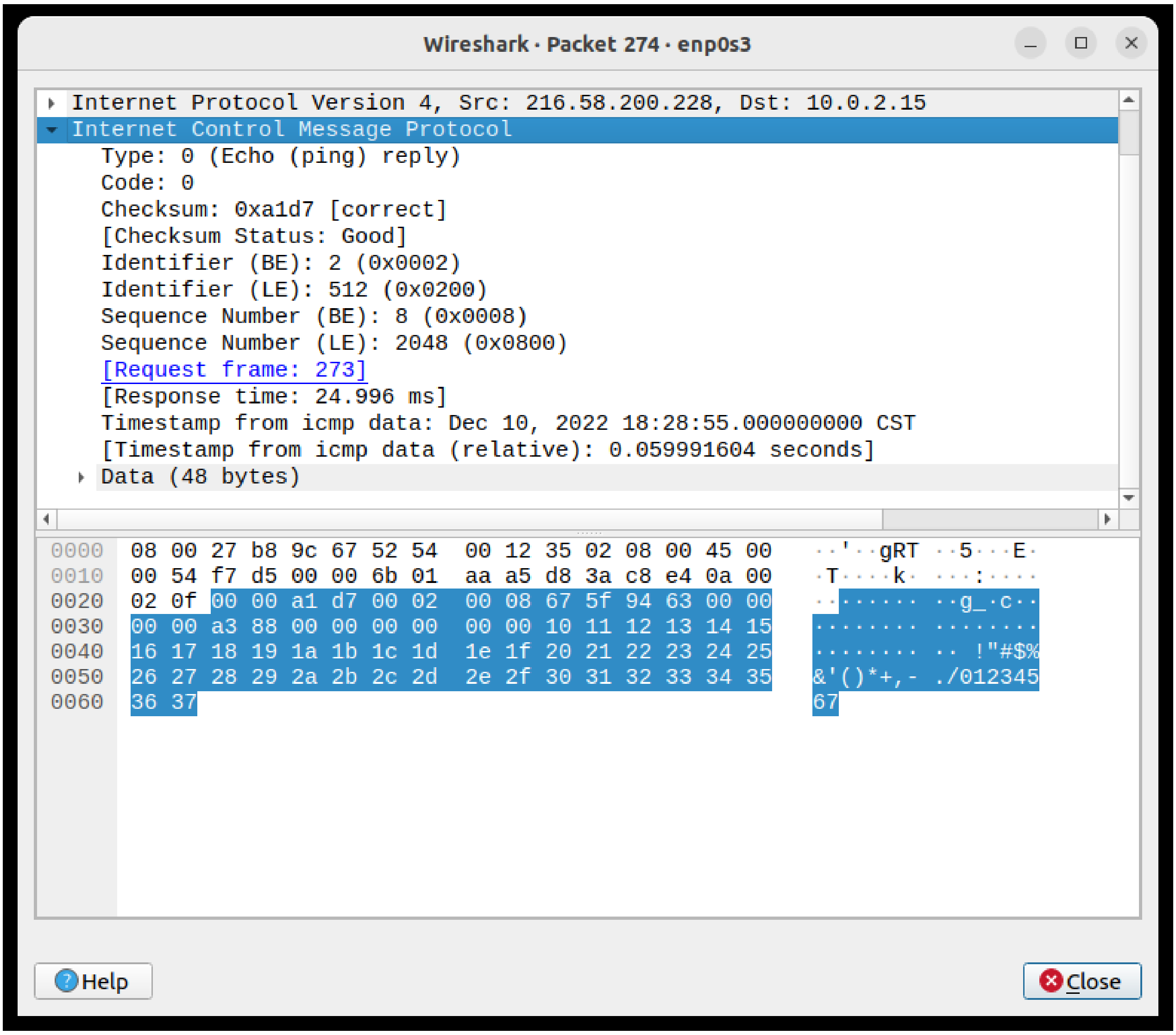
- Hardware type, Protocol type: 2 bytes each
- Hardware size, Protocol size: 1 byte each
- Opcode: 2 bytes
- Sender and target MAC address: 6 bytes each
- Sender and target IP address: 4 bytes each

ARP capture by tshark command

```
mumeicc@CC-VM:~$ tshark -i enp0s3 -f "arp"
Capturing on 'enp0s3'
 ** (tshark:6099) 01:02:54.686214 [Main MESSAGE] -- Capture started.
 ** (tshark:6099) 01:02:54.686315 [Main MESSAGE] -- File: "/tmp/wireshark_enp0s3
AJ3CX1.pcapng"
    1 0.000000000 PcsCompu_b8:9c:67 → RealtekU_12:35:02 ARP 42 Who has 10.0.2.2?
 Tell 10.0.2.15
    2 0.000057919 RealtekU_12:35:02 → PcsCompu_b8:9c:67 ARP 60 10.0.2.2 is at 52
:54:00:12:35:02
```

4. ICMP

ICMP capture by Wireshark

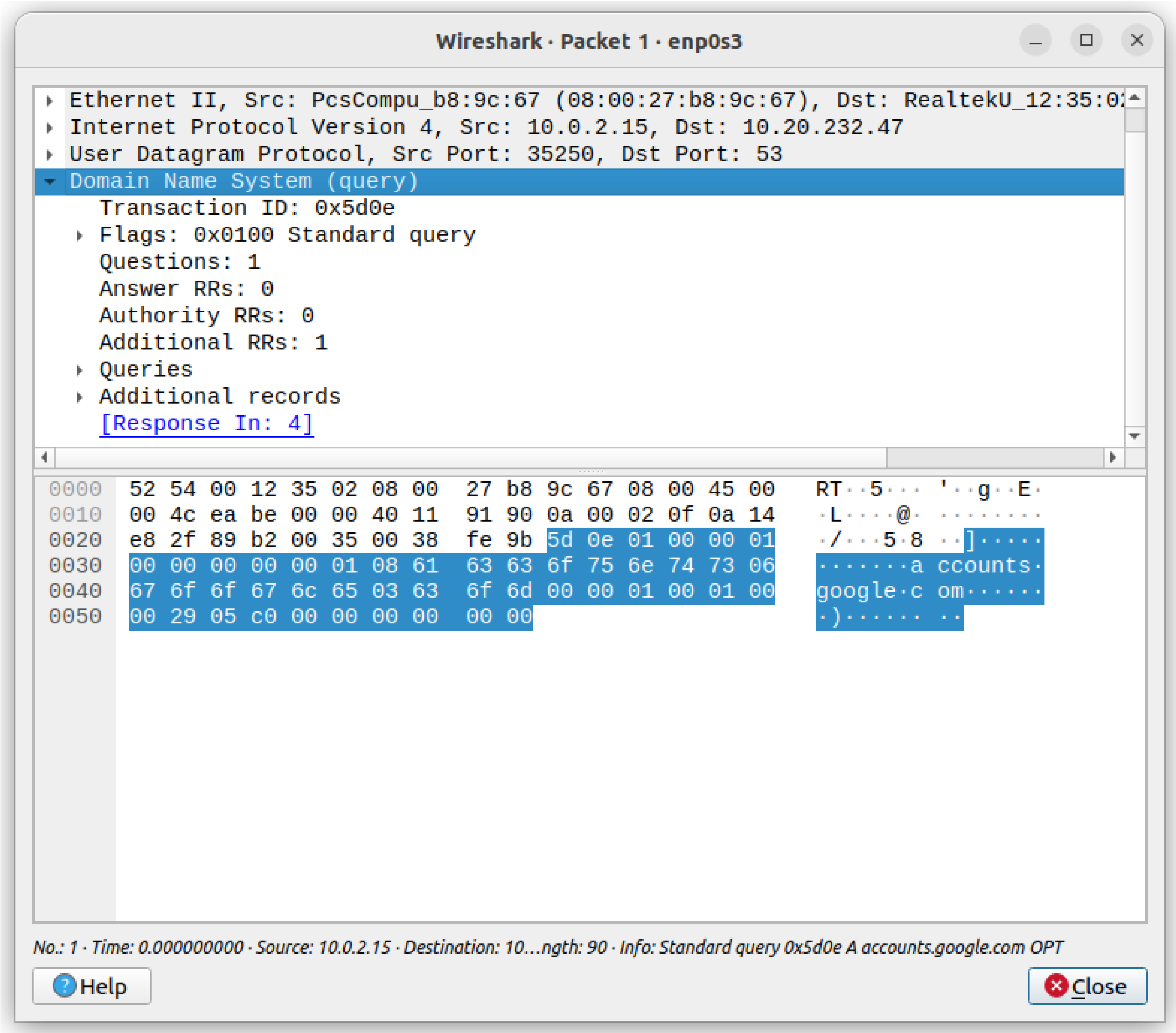


ICMP record here is 64 bytes, including first 8 bytes header:

- Type, Code: 1 bytes each
- Checksum: 2 bytes
- Identifier (BE and LE): 2 bytes
- Sequence number (BE and LE): 2 bytes
- Timestamp: 8 bytes
- Data: 48 bytes

5. DNS

DNS capture by Wireshark



DNS record here is 48 bytes:

- Header 4 bytes, including
  - Transaction ID: 2 bytes, here `0x5d0e`
  - Flags: 2 bytes. We have a flags example:

```
▾ Flags: 0x8180 Standard query response, No error
    1... .... .... .... = Response: Message is a response
    .000 0... .... .... = Opcode: Standard query (0)
    .... .0.. .... .... = Authoritative: Server is not an authority for domain
    .... ..0. .... .... = Truncated: Message is not truncated
    .... ...1 .... .... = Recursion desired: Do query recursively
    .... .... 1... .... = Recursion available: Server can do recursive queries
    .... .... .0.. .... = Z: reserved (0)
    .... .... ..0. .... = Answer authenticated: Answer/authority portion was n
    .... .... ...0 .... = Non-authenticated data: Unacceptable
    .... .... .... 0000 = Reply code: No error (0)
```

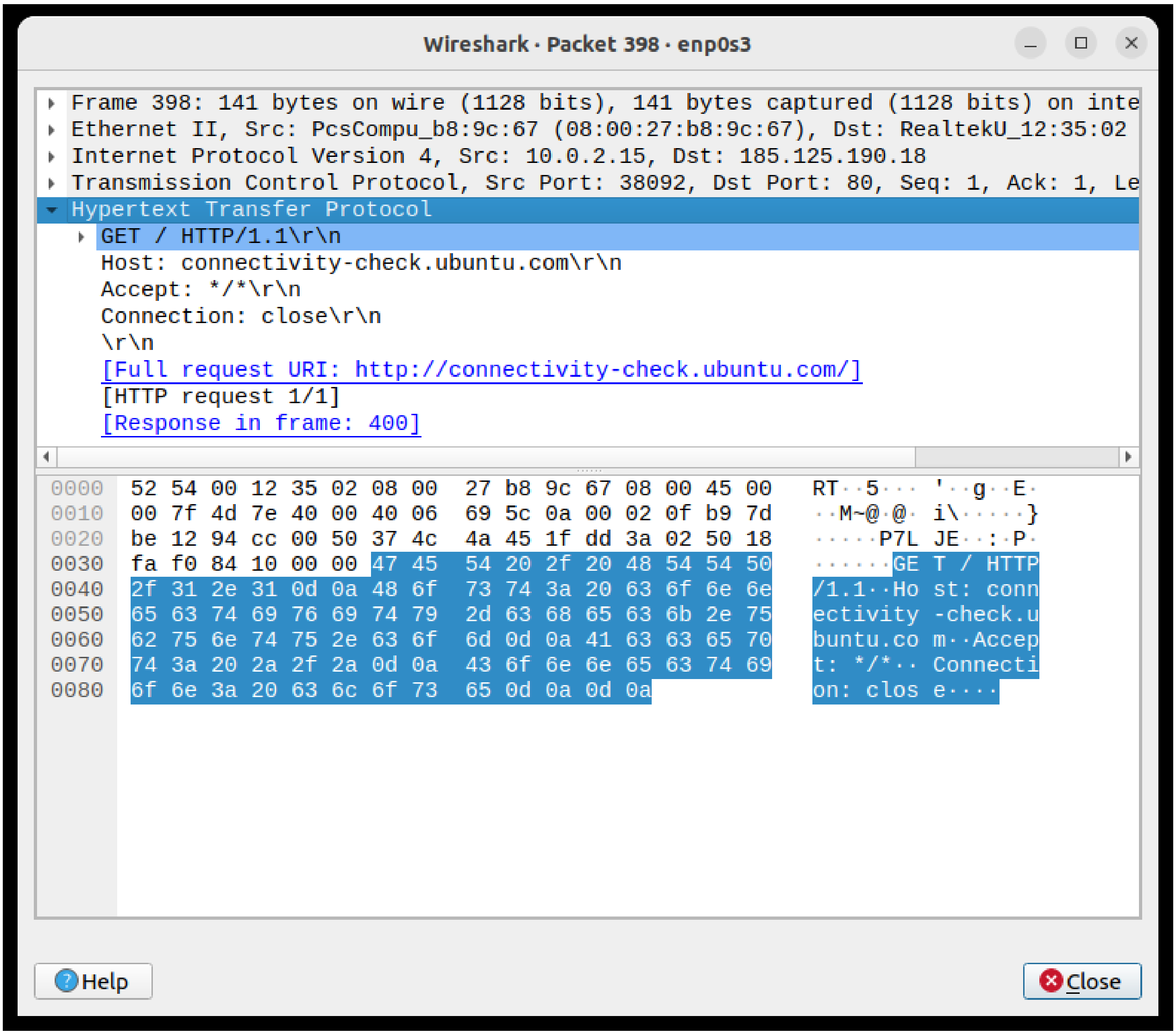
    The header flags format of DNS message in Wikipedia shows in here:

Field	Description	Length (bits)
QR	Indicates if the message is a query (0) or a reply (1)	1
OPCODE	The type can be QUERY (standard query, 0), IQUERY (inverse query, 1), or STATUS (server status request, 2)	4
AA	Authoritative Answer, in a response, indicates if the DNS server is authoritative for the queried hostname	1
TC	TrunCation, indicates that this message was truncated due to excessive length	1
RD	Recursion Desired, indicates if the client means a recursive query	1
RA	Recursion Available, in a response, indicates if the replying DNS server supports recursion	1
Z	Zero, reserved for future use	3
RCODE	Response code, can be NOERROR (0), FORMERR (1, Format error), SERVFAIL (2), NXDOMAIN (3, Nonexistent domain), etc.[37]	4

- Questions, Answer RRs, Authority RRs and Additional RRs: 2 bytes each
- Query and Additional records

6. HTTP

HTTP message capture by Wireshark

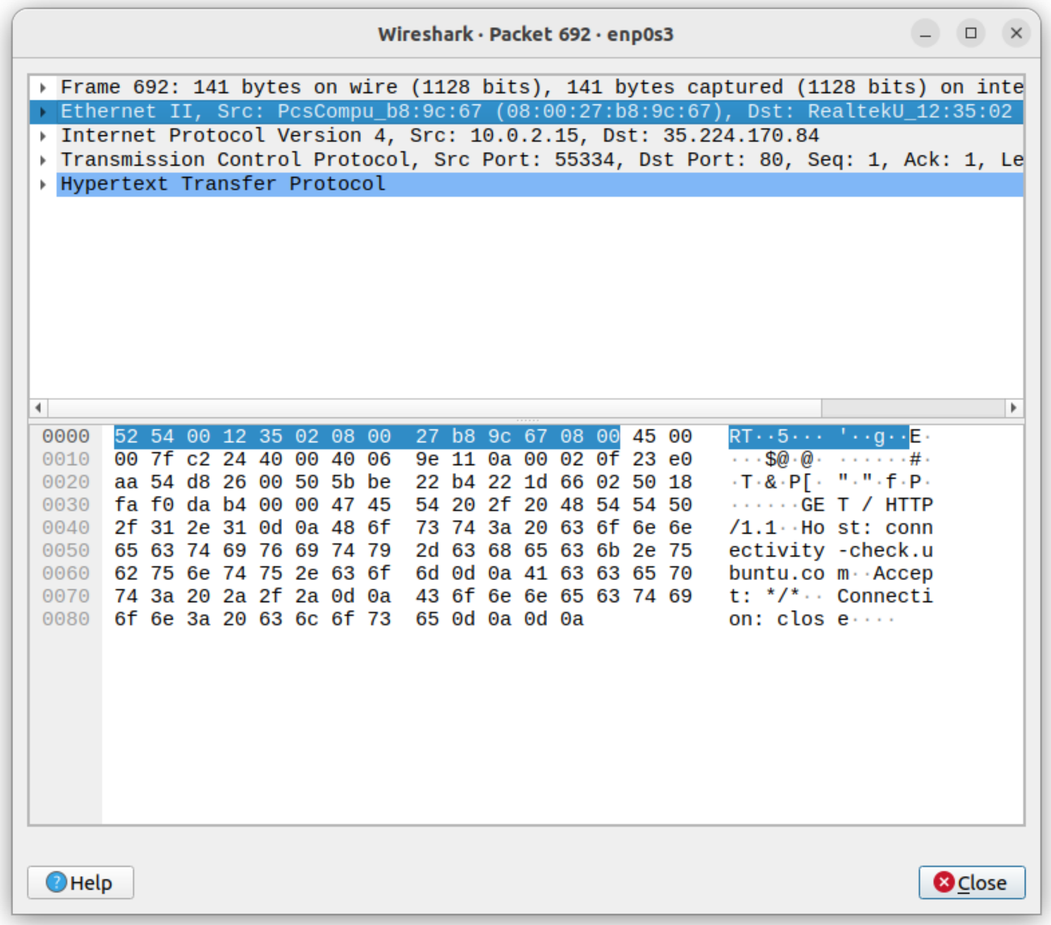


The message contains:

- Request method **GET** with
    - Request URL, `/` here
    - Request Version, `HTTP/1.1` here
- Host, `connectivity -check.ubuntu.com` here
- Accept, `*/*` here
- Connection, `close` here

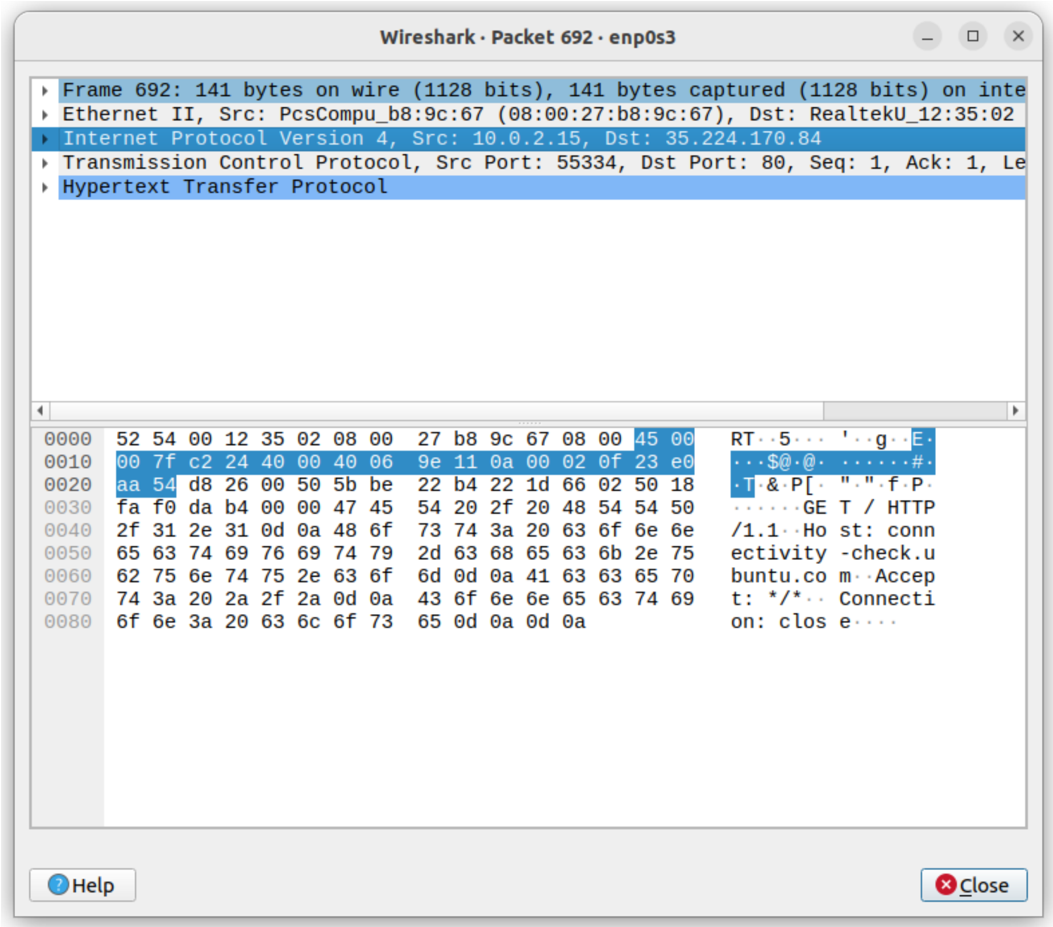
Encapsulation and decapsulation process

We take the HTTP packet as an example.

1. 

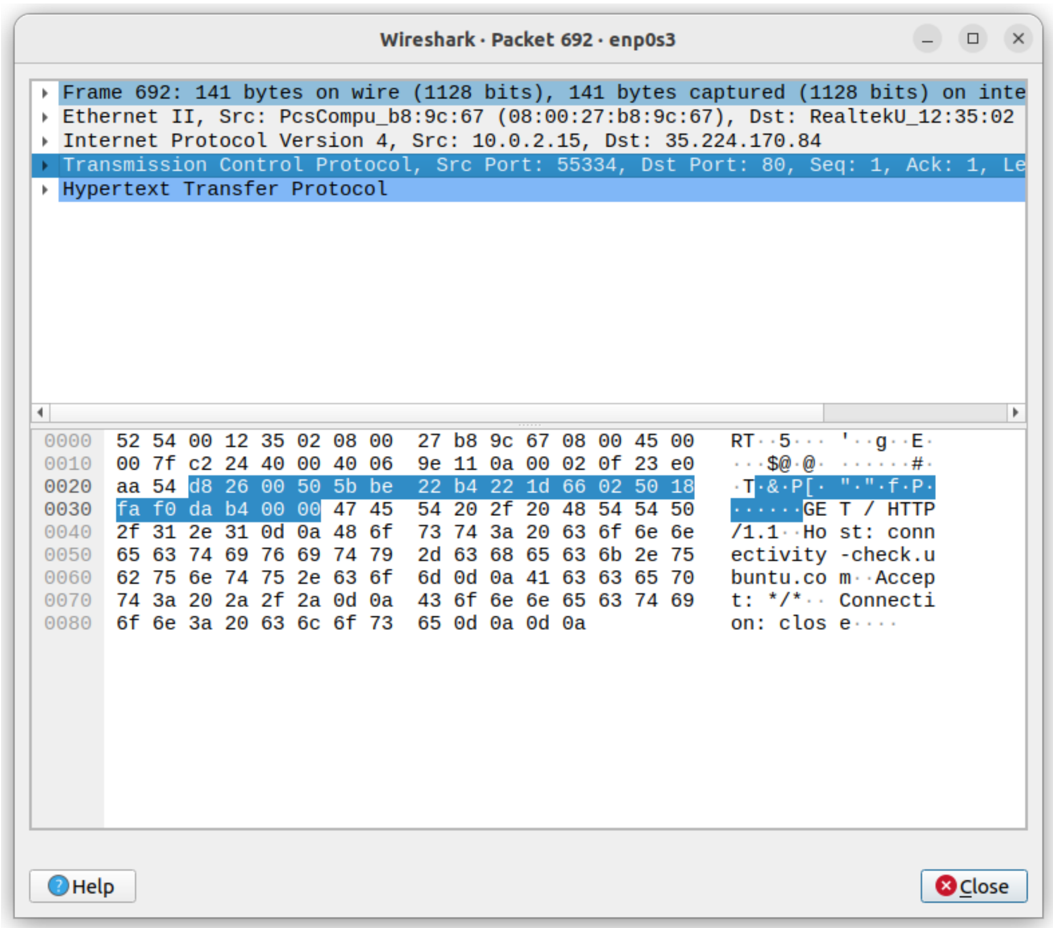
Ethernet II (**Data link layer**)

2.

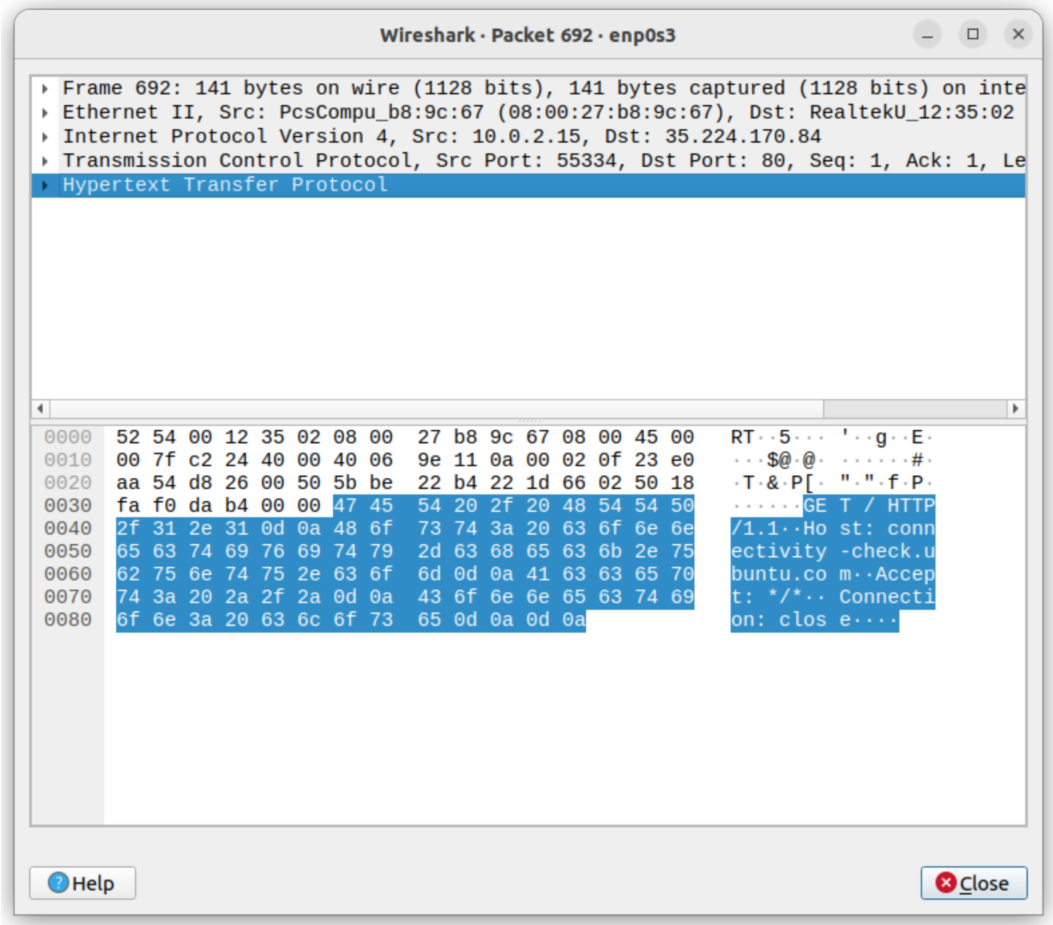


IPv4 (**Network layer**)

3.



TCP (**Transport layer**)

4. 

HTTP (**Application layer**)

Therefore, the encapsulation process are: The data (contents of webpage here) are encapsulated with an HTTP header, then TCP header, IP header and Ethernet header. Then, it is encapsulated to a frame with frame header and tailer. The decapsulation process is the inverse of the encapsulation process, that is, remove the header and tailer one by one from frame to HTTP header.