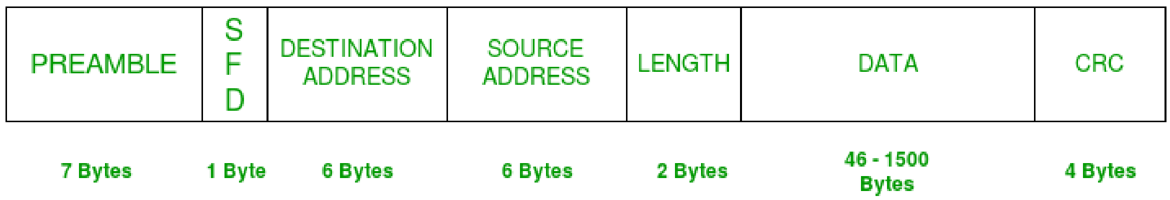
1. Ethernet protocol

1.1. Frame structure (IEEE 802.3):



IEEE 802.3 ETHERNET Frame Format

The basic format for ethernet protocol which is needed for all mac implementation which is an IEEE 802.3 standard.

1.1.1. PREAMBLE:

preamble was introduced to allow for the loss of a few bits due to signal delays. But today's high-speed Ethernet doesn't need Preamble to protect the frame bits.

1.1.2. SFD (Start of frame delimiter):

This is a 1-Byte field that is always set to 10101011. SFD indicates that upcoming bits are starting the frame, Sometimes SFD is considered part of PREAMBLE, this is the reason Preamble is described as 8 Bytes in many places.

1.1.3. Destination address:

This is a 6-Byte field that contains the MAC address of the machine for which data is destined.

1.1.4. Source address:

This is a 6-Byte field that contains the MAC address of the source machine.

1.1.5. Length:

Length is a 2-Byte field, which indicates the length of the entire Ethernet frame.

1.1.6. Data:

This is the place where actual data is inserted, also known as Payload. Both IP header and data will be inserted here if Internet Protocol is used over Ethernet.

1.1.7. CRC (Cyclic Redundancy Check):

CRC is 4 Byte field. This field contains a 32-bits hash code of data, which is generated over the Destination Address, Source Address, Length, and Data field. If the checksum computed by destination is not the same as sent checksum value, data received is corrupted.

1.2. Frame structure (Ethernet II):

DA	SA	Type	DSAP	SSAP	Ctrl	Data	FCS
6 Bytes	6 Bytes	2 Bytes	1 Byte	1 Byte	1 Byte	> 46 Bytes	4 Bytes

Proposed ETHERNET Frame Extension

Unlike basic version of ethernet protocol you can have a larger payload for your data.

1.2.1. DA:
Destination MAC Address, size: 6bytes

1.2.2. SA:
Source MAC Address, size: 6bytes

1.2.3. Type:
Ethertype, size: 2bytes

1.2.4. DSAP:
Destination Service Access Point, size: 1byte

1.2.5. SSAP:
Source Service Access Point, size: 1byte

1.2.6. Ctrl:
Control field, size: 1byte

1.2.7. Data:
Data, size: gt 46bytes

1.2.8. FCS:
Frame checksum, size: 4bytes

1.3. Capturing packets:

Ethernet protocol is observable over any packets sent and received but various of inner protocols:

- This is a random packet for example:

```
529 15.113661   172.24.41.27   35.190.80.1      TCP       54 12533 → 443 [ACK] Seq=270 Ack=4580 Win=
530 15.114938   172.24.41.27   35.190.80.1      TLSv1.3  134 Change Cipher Spec, Application Data
531 15.199285   35.190.80.1    172.24.41.27     TLSv1.3  672 Application Data, Application Data
532 15.199420   172.24.41.27   35.190.80.1      TLSv1.3  715 Application Data, Application Data, App
533 15.203964   172.24.41.27   142.250.186.142  TCP     1374 [TCP Retransmission] 12513 → 443 [ACK]
534 15.314036   172.24.41.27   50.7.127.174     TCP       54 12522 → 443 [FIN, ACK] Seq=518 Ack=1 Wi
```

- In frame sections you can see further details about requests:
- For example

```
Ethernet II, Src: IntelCor_97:8d:f8 (b4:0e:de:97:8d:f8), Dst: Cisco_ed:09:d5 (d0:72:dc:ed:09:d5)
  Destination: Cisco_ed:09:d5 (d0:72:dc:ed:09:d5)
    Address: Cisco_ed:09:d5 (d0:72:dc:ed:09:d5)
    .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  Source: IntelCor_97:8d:f8 (b4:0e:de:97:8d:f8)
    Address: IntelCor_97:8d:f8 (b4:0e:de:97:8d:f8)
    .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  Type: IPv4 (0x0800)
```

- As you can see there is destination, source and type as we saw in ethernet protocol structure
- Destinations field:

```
> Frame 532: 715 bytes on wire (5720 bits), 715 bytes captured (5720 bits) on interface \Device\NPF_{F340    0000  d0 72 dc ed 09 d5 b4 0e  de 97
v Ethernet II, Src: IntelCor_97:8d:f8 (b4:0e:de:97:8d:f8), Dst: Cisco_ed:09:d5 (d0:72:dc:ed:09:d5)         0010  02 bd 71 c8 40 00 80 06  3d 80
  v Destination: Cisco_ed:09:d5 (d0:72:dc:ed:09:d5)                                                         0020  50 01 30 f5 01 bb 1b 25  24 8d
      Address: Cisco_ed:09:d5 (d0:72:dc:ed:09:d5)                                                           0030  02 01 57 45 00 00 17 03  03 00
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)                     0040  1f d6 92 b4 82 66 fd ca  3f 9c
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)                                  0050  a6 bc 77 b8 bc 2c 66 69  76 7c
                                                                                                            0060  d1 83 25 bf 17 03 03 00  39 86
```

6 bytes length for destination as we saw

Source structure is same as destination.

- Type field:

```
> Frame 532: 715 bytes on wire (5720 bits), 715 bytes captured (5720 bits) on interface \Device\NPF_{F340   0000  d0 72 dc ed 09 d5 b4 0e  de 97 8d f8 08 00 45
v Ethernet II, Src: IntelCor_97:8d:f8 (b4:0e:de:97:8d:f8), Dst: Cisco_ed:09:d5 (d0:72:dc:ed:09:d5)        0010  02 bd 71 c8 40 00 80 06  3d 80 ac 18 29 1b 23
  v Destination: Cisco_ed:09:d5 (d0:72:dc:ed:09:d5)                                                        0020  50 01 30 f5 01 bb 1b 25  24 8d 46 e9 6f d5 50
      Address: Cisco_ed:09:d5 (d0:72:dc:ed:09:d5)                                                          0030  02 01 57 45 00 00 17 03  03 00 29 41 c5 66 bc
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)                    0040  1f d6 92 b4 82 66 fd ca  3f 9c c4 42 5c c2 8b
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)                                 0050  a6 bc 77 b8 bc 2c 66 69  76 7c 2f 63 fb 4b 83
  v Source: IntelCor_97:8d:f8 (b4:0e:de:97:8d:f8)                                                          0060  d1 83 25 bf 17 03 03 00  39 86 66 a2 1a b9 90
      Address: IntelCor_97:8d:f8 (b4:0e:de:97:8d:f8)                                                       0070  d9 0e d2 a7 8d ad ef f5  71 9b 7d ee 09 b3 8f
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)                    0080  7e f7 93 31 76 b0 13 ff  ff f9 b6 c6 cf 12 9b
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)                                 0090  62 68 f3 6b ad 13 b6 34  af 24 35 d1 89 5d 0c
    Type: IPv4 (0x0800)                                                                                    00a0  9f 0b 17 03 03 00 e8 26  8c b0 a6 25 0e 6f 9e
> Internet Protocol Version 4, Src: 172.24.41.27, Dst: 35.190.80.1                                         00b0  85 c2 a1 6b b6 01 5d 87  a9 00 39 9b 5a f1 2c
                                                                                                           00c0  20 6f 7d 57 8c 70 5a 65  8b da 57 06 e1 5b 52
```

2 bytes for type

Another example:

- A http request:

No.	Time	Source	Destination	Protocol	Length	Info
71	3.494169	172.24.41.27	92.123.106.33	HTTP	165	GET /connecttest.txt HTTP/1.1
76	3.836597	92.123.106.33	172.24.41.27	HTTP	267	HTTP/1.1 200 OK (text/plain)

- Ethernet section

```
> Frame 71: 165 bytes on wire (1320 bits), 165 bytes captured (1320 bits) on interface \Device\NPF_{F340840E-F56F-46C8-A521-9B3D386D82E3}, id 0
v Ethernet II, Src: IntelCor_97:8d:f8 (b4:0e:de:97:8d:f8), Dst: Cisco_ed:09:d5 (d0:72:dc:ed:09:d5)
  > Destination: Cisco_ed:09:d5 (d0:72:dc:ed:09:d5)
  > Source: IntelCor_97:8d:f8 (b4:0e:de:97:8d:f8)
    Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 172.24.41.27, Dst: 92.123.106.33
> Transmission Control Protocol, Src Port: 5447, Dst Port: 80, Seq: 1, Ack: 1, Len: 111
> Hypertext Transfer Protocol
```

```
0000  d0 72 dc ed 09 d5 b4 0e  de 97 8d f8 08 00 45 00   ·r······ ······E·
0010  00 97 fb 74 40 00 80 06  63 1c ac 18 29 1b 5c 7b   ···t@··· c···)·\{
0020  6a 21 15 47 00 50 0d bf  8e ab 05 5b 8b f0 50 18   j!·G·P·· ···[··P·
0030  02 03 0f 0b 00 00 47 45  54 20 2f 63 6f 6e 6e 65   ······GE T /conne
0040  63 74 74 65 73 74 2e 74  78 74 20 48 54 54 50 2f   cttest.t xt HTTP/
0050  31 2e 31 0d 0a 43 6f 6e  6e 65 63 74 69 6f 6e 3a   1.1··Con nection:
0060  20 43 6c 6f 73 65 0d 0a  55 73 65 72 2d 41 67 65    Close·· User-Age
0070  6e 74 3a 20 4d 69 63 72  6f 73 6f 66 74 20 4e 43   nt: Micr osoft NC
0080  53 49 0d 0a 48 6f 73 74  3a 20 77 77 77 2e 6d 73   SI··Host : www.ms
0090  66 74 63 6f 6e 6e 65 63  74 74 65 73 74 2e 63 6f   ftconnec ttest.co
00a0  6d 0d 0a 0d 0a                                     m····
```

There is total of 14 bytes here, first 6 for destination, second 6 for source and last 2 for type.

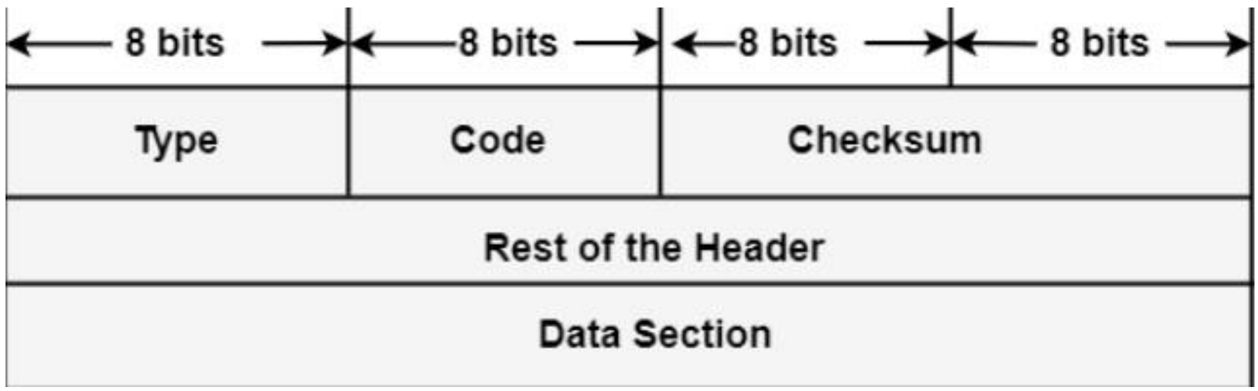
```
Pinging restrictmoderate.youtube.com [10.10.34.36] with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 10.10.34.36:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

There is no reply’s because destination is unreachable:

No.	Time	Source	Destination	Protocol	Length	Info
104	2.299745	172.24.40.86	10.10.34.36	ICMP	74	Echo (ping) request id=0x0001, seq=57337/63967, ttl=128 (no response found!)
345	7.219311	172.24.40.86	10.10.34.36	ICMP	74	Echo (ping) request id=0x0001, seq=57338/64223, ttl=128 (no response found!)
554	12.224617	172.24.40.86	10.10.34.36	ICMP	74	Echo (ping) request id=0x0001, seq=57339/64479, ttl=128 (no response found!)
832	17.221128	172.24.40.86	10.10.34.36	ICMP	74	Echo (ping) request id=0x0001, seq=57340/64735, ttl=128 (no response found!)

2. ICMP:

2.1. Frame structure:



2.1.1. Type: It is an 8-bit field. It represents the ICMP message type. The values area from 0 to 127 are described for ICMPv6, and the values from 128 to 255 are the data messages.

2.1.2. Code: It is an 8-bit field that represents the subtype of the ICMP message.

2.1.3. Checksum: It is a 16-bit field to recognize whether the error exists in the message or not.

2.2. Capture section:

- We can use ping command on CMD to test the ICMP protocol:

```
C:\Users\mohammad>ping 8.8.8.8

Pinging 8.8.8.8 with 32 bytes of data:
Reply from 8.8.8.8: bytes=32 time=73ms TTL=108
Reply from 8.8.8.8: bytes=32 time=63ms TTL=108
Reply from 8.8.8.8: bytes=32 time=61ms TTL=108
Reply from 8.8.8.8: bytes=32 time=57ms TTL=108

Ping statistics for 8.8.8.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 57ms, Maximum = 73ms, Average = 63ms

C:\Users\mohammad>
```

- There are 4 requests for ping to 8.8.8.8 server and if the given IP is a reachable IP there should be 4 replies too.
- Let’s check the wireshark:
- Search for ICMP protocols in wireshark

icmp

No.	Time	Source	Destination	Protocol	Length	Info
754	15.632498	172.24.41.27	8.8.8.8	ICMP	74	Echo (ping) request id=0x0001, seq=17/4352, ttl=128 (reply in 765)
765	15.706100	8.8.8.8	172.24.41.27	ICMP	74	Echo (ping) reply id=0x0001, seq=17/4352, ttl=108 (request in 754)
776	16.649248	172.24.41.27	8.8.8.8	ICMP	74	Echo (ping) request id=0x0001, seq=18/4608, ttl=128 (reply in 777)
777	16.712617	8.8.8.8	172.24.41.27	ICMP	74	Echo (ping) reply id=0x0001, seq=18/4608, ttl=108 (request in 776)
799	17.662456	172.24.41.27	8.8.8.8	ICMP	74	Echo (ping) request id=0x0001, seq=19/4864, ttl=128 (reply in 807)
807	17.723391	8.8.8.8	172.24.41.27	ICMP	74	Echo (ping) reply id=0x0001, seq=19/4864, ttl=108 (request in 799)
834	18.672695	172.24.41.27	8.8.8.8	ICMP	74	Echo (ping) request id=0x0001, seq=20/5120, ttl=128 (reply in 835)
835	18.729677	8.8.8.8	172.24.41.27	ICMP	74	Echo (ping) reply id=0x0001, seq=20/5120, ttl=108 (request in 834)

- There are 4 requests and for replies here
- For the request the source is our device and the destination is the given IP (here 8.8.8.8)
- And for replies the source and destination has been replaced with each other

```
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x4d4a [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence Number (BE): 17 (0x0011)
    Sequence Number (LE): 4352 (0x1100)
    [Response frame: 765]
```

```
> Ethernet II, Src: IntelCor_97:8d:f8 (b4:0e:de:97:8d:f8), Dst: Cisco_ed:09:d5 (d0:72:dc:ed:09:d5)    0010  00 3c 86 05 00 00 80 01
> Internet Protocol Version 4, Src: 172.24.41.27, Dst: 8.8.8.8                                          0020  08 08 08 00 4d 4a 00 01
v Internet Control Message Protocol                                                                     0030  67 68 69 6a 6b 6c 6d 6e
    Type: 8 (Echo (ping) request)                                                                       0040  77 61 62 63 64 65 66 67
    Code: 0
    Checksum: 0x4d4a [correct]
```

Type field: 1byte or 8bits

```
> Internet Protocol Version 4, Src: 172.24.41.27, Dst: 8.8.8.8      0020  08 08 08 00 4d 4a
v Internet Control Message Protocol                                 0030  67 68 69 6a 6b 6c
    Type: 8 (Echo (ping) request)                                   0040  77 61 62 63 64 65
    Code: 0
    Checksum: 0x4d4a [correct]
    [Checksum Status: Good]
```

Code field: same as type field it's 8bits

```
> Frame 754: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{F340840E    0000  d0 72 dc ed 09 d5 b4 0e
> Ethernet II, Src: IntelCor_97:8d:f8 (b4:0e:de:97:8d:f8), Dst: Cisco_ed:09:d5 (d0:72:dc:ed:09:d5)          0010  00 3c 86 05 00 00 80 01
> Internet Protocol Version 4, Src: 172.24.41.27, Dst: 8.8.8.8                                                0020  08 08 08 00 4d 4a 00 01
v Internet Control Message Protocol                                                                           0030  67 68 69 6a 6b 6c 6d 6e
    Type: 8 (Echo (ping) request)                                                                             0040  77 61 62 63 64 65 66 67
    Code: 0
    Checksum: 0x4d4a [correct]
    [Checksum Status: Good]
```

Checksum field: 16bits of data

- In second example we use an unreachable IP:

```
Pinging 192.168.0.0 with 32 bytes of data:
Reply from 172.24.40.1: Destination net unreachable.
Reply from 172.24.40.1: Destination net unreachable.
Reply from 172.24.40.1: Destination net unreachable.
Reply from 172.24.40.1: Destination net unreachable.

Ping statistics for 192.168.0.0:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),

C:\Users\mohammad>
```

- There is 0 percent loss and there is the unreachable reply for given IP

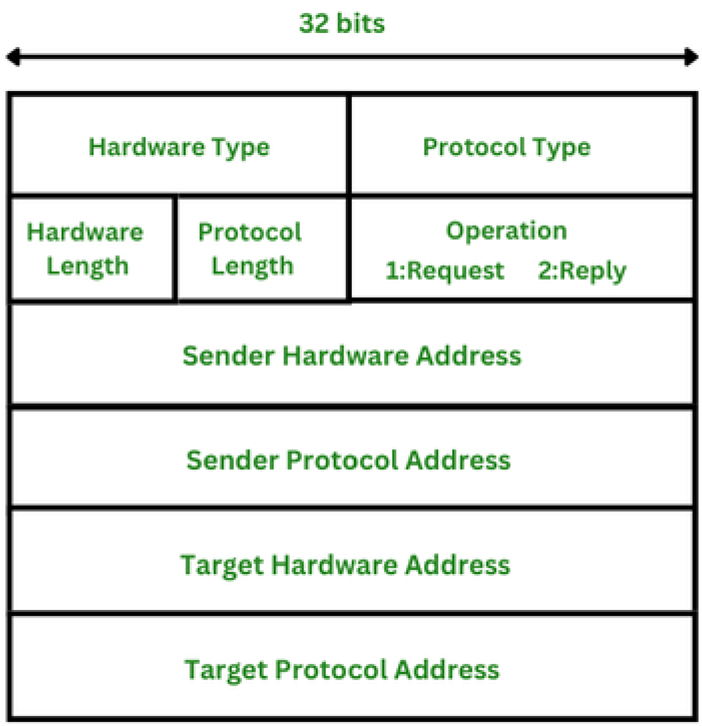
- It means the request and reply was sent and received successfully but the given IP is not usable

No.	Time	Source	Destination	Protocol	Length	Info
167	3.312584	172.24.41.27	192.168.0.0	ICMP	74	Echo (ping) request id=0x0001, seq=40/10240, ttl=128 (no response found!)
169	3.321963	172.24.40.1	172.24.41.27	ICMP	70	Destination unreachable (Communication administratively filtered)
188	4.324580	172.24.41.27	192.168.0.0	ICMP	74	Echo (ping) request id=0x0001, seq=41/10496, ttl=128 (no response found!)
189	4.333649	172.24.40.1	172.24.41.27	ICMP	70	Destination unreachable (Communication administratively filtered)
211	5.334330	172.24.41.27	192.168.0.0	ICMP	74	Echo (ping) request id=0x0001, seq=42/10752, ttl=128 (no response found!)
212	5.409142	172.24.40.1	172.24.41.27	ICMP	70	Destination unreachable (Communication administratively filtered)
227	6.343297	172.24.41.27	192.168.0.0	ICMP	74	Echo (ping) request id=0x0001, seq=43/11008, ttl=128 (no response found!)
228	6.346147	172.24.40.1	172.24.41.27	ICMP	70	Destination unreachable (Communication administratively filtered)

- Requests and replies

3. ARP (Address Resolution Protocol):

3.1. Frame structure:



3.1.1. Hardware type: This is 16 bits field defining the type of the network on which ARP is running. Ethernet is given type 1.

3.1.2. Protocol type: This is 16 bits field defining the protocol. The value of this field for the IPv4 protocol is 0800H.

3.1.3. Hardware length: This is an 8 bits field defining the length of the physical address in bytes. Ethernet is the value 6.

3.1.4. Protocol length: This is an 8 bits field defining the length of the logical address in bytes. For the IPv4 protocol, the value is 4.

3.1.5. Operation (request or reply): This is a 16 bits field defining the type of packet. Packet types are ARP request (1), and ARP reply (2).

3.1.6. Sender hardware address: This is a variable length field defining the physical address of the sender. For example, for Ethernet, this field is 6 bytes long.

3.1.7. Sender protocol address: This is also a variable length field defining the logical address of the sender For the IP protocol, this field is 4 bytes long.

3.1.8. Target hardware address: This is a variable length field defining the physical address of the target. For Ethernet, this field is 6 bytes long. For the ARP request messages, this field is all Os because the sender does not know the physical address of the target.

3.1.9. Target protocol address: This is also a variable length field defining the logical address of the target. For the IPv4 protocol, this field is 4 bytes long.

3.2. Capturing packets:

- At first turn off the wi-fi.
- The start capturing in wireshark over wi-fi
- Then turn on the wifi
- Stop capturing in wireshark
- Search for arp protocols

- Example of sending arp request:

5	0.084676	Cisco_ed:09:d5	IntelCor_97:8d:f8	ARP	60 172.24.40.1 is at d0:72:dc:ed:09:d5
37	0.338107	IntelCor_97:8d:f8	Broadcast	ARP	42 Who has 172.24.40.1? Tell 172.24.40.86
39	0.344715	Cisco_ed:09:d5	IntelCor_97:8d:f8	ARP	60 172.24.40.1 is at d0:72:dc:ed:09:d5

- In the frame sections you will see the frame structure of this protocol:

```
Protocol type: IPv4 (0x0800)
Hardware size: 6
Protocol size: 4
Opcode: request (1)
Sender MAC address: IntelCor_97:8d:f8 (b4:0e:de:97:8d:f8)
Sender IP address: 172.24.40.86
Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
Target IP address: 172.24.40.1
```

```
0000  ff ff ff ff ff ff b4 0e  de 97 8d f8 08 06 00 01
0010  08 00 06 04 00 01 b4 0e  de 97 8d f8 ac 18 28 56
0020  00 00 00 00 00 00 ac 18  28 01
```

As you see the source address is my pc and there is no specific destination for arp protocol, packet travels to all needed destination by itself (broadcast).

- Answer for above request:

37	0.338107	IntelCor_97:8d:f8	Broadcast	ARP	42 Who has 172.24.40.1? Tell 172.24.40.86
39	0.344715	Cisco_ed:09:d5	IntelCor_97:8d:f8	ARP	60 172.24.40.1 is at d0:72:dc:ed:09:d5
43	0.452019	IntelCor_97:8d:f8	Broadcast	ARP	42 Who has 172.24.40.86? (ARP Probe)

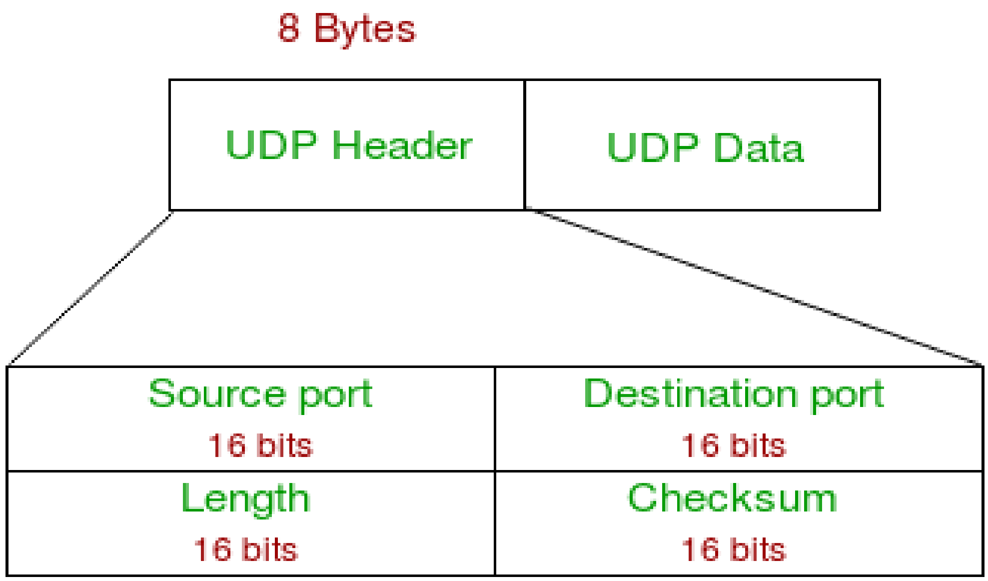
```
Protocol type: IPv4 (0x0800)
Hardware size: 6
Protocol size: 4
Opcode: reply (2)
Sender MAC address: Cisco_ed:09:d5 (d0:72:dc:ed:09:d5)
Sender IP address: 172.24.40.1
Target MAC address: IntelCor_97:8d:f8 (b4:0e:de:97:8d:f8)
Target IP address: 172.24.40.86
```

```
0000  b4 0e de 97 8d f8 d0 72  dc ed 09 d5 08 06 00 01
0010  08 00 06 04 00 02 d0 72  dc ed 09 d5 ac 18 28 01
0020  b4 0e de 97 8d f8 ac 18  28 56 00 00 00 00 00 00
0030  00 00 00 00 00 00 00 00  00 00 00 00
```

In the answer you see that the destination is changed to my address and the source address is now changed to receiver of last request.

4. UDP (User Datagram Protocol):

4.1. Frame structure:



4.1.1. Source Port: Source Port is a 2 Byte long field used to identify the port number of the source.

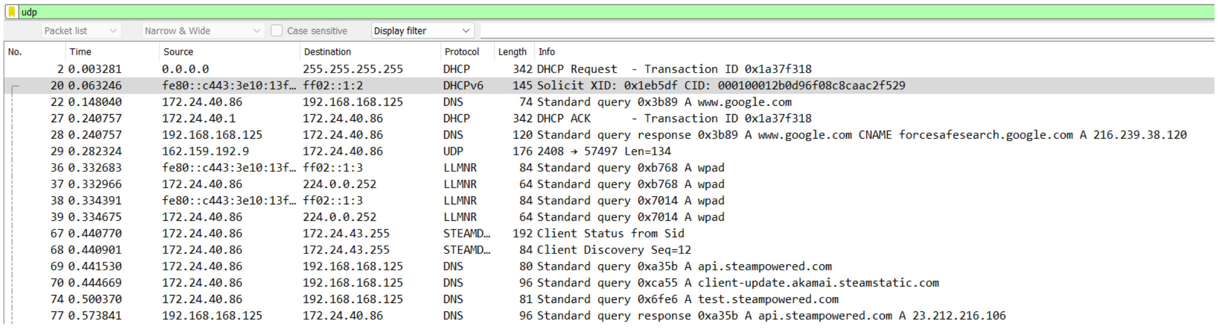
4.1.2. Destination Port: It is a 2 Byte long field, used to identify the port of the destined packet.

4.1.3. Length: Length is the length of UDP including the header and the data. It is a 16-bits field.

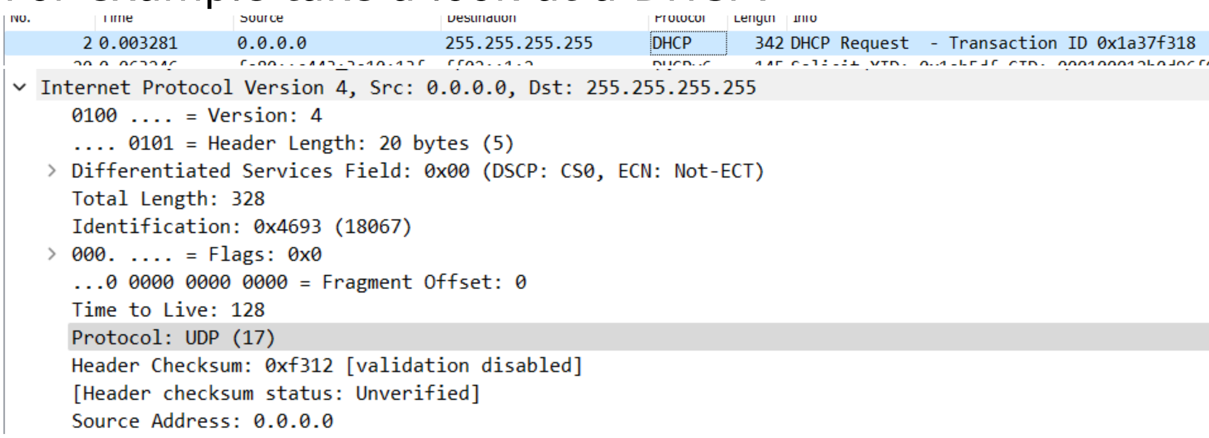
4.1.4. Checksum: Checksum is 2 Bytes long field. It is the 16-bit one's complement of the one's complement sum of the UDP header, the pseudo-header of information from the IP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.

4.2. Capturing packets:

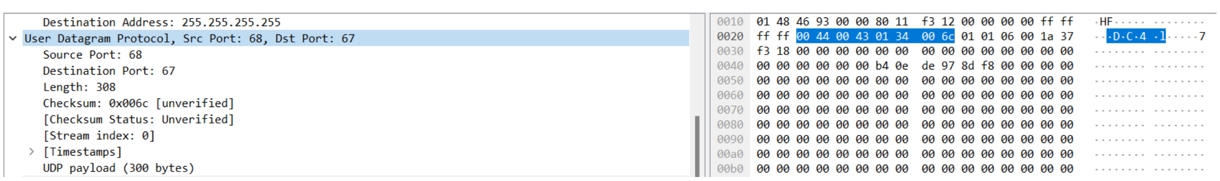
- Turn off the wi-fi
- Start capturing over wi-fi
- Turn on the wi-fi
- Search for udp protocls
- Now basically you can see all frames that are using udp itself or they are running over udp protocol like an inner protocol.



- As you see there are many protocols that are running udp as inner protocol like: DHCP, DHCPV6, DNS , LLMNR, MDNS, …
- For example take a look at a DHCP:

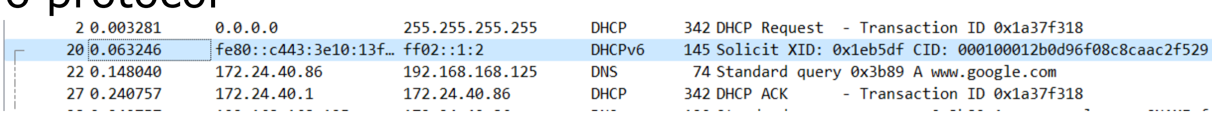


Inner protocol which is udp



All udp sections of frame are shown

- Another example is DHCPV6 which uses version 6 protocol



```
Internet Protocol Version 6, Src: fe80::c443:3e10:13f1:c498, Dst: ff02::1:2
    0110 .... = Version: 6
    .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... 1000 0001 0011 0001 1110 = Flow Label: 0x8131e
    Payload Length: 91
```

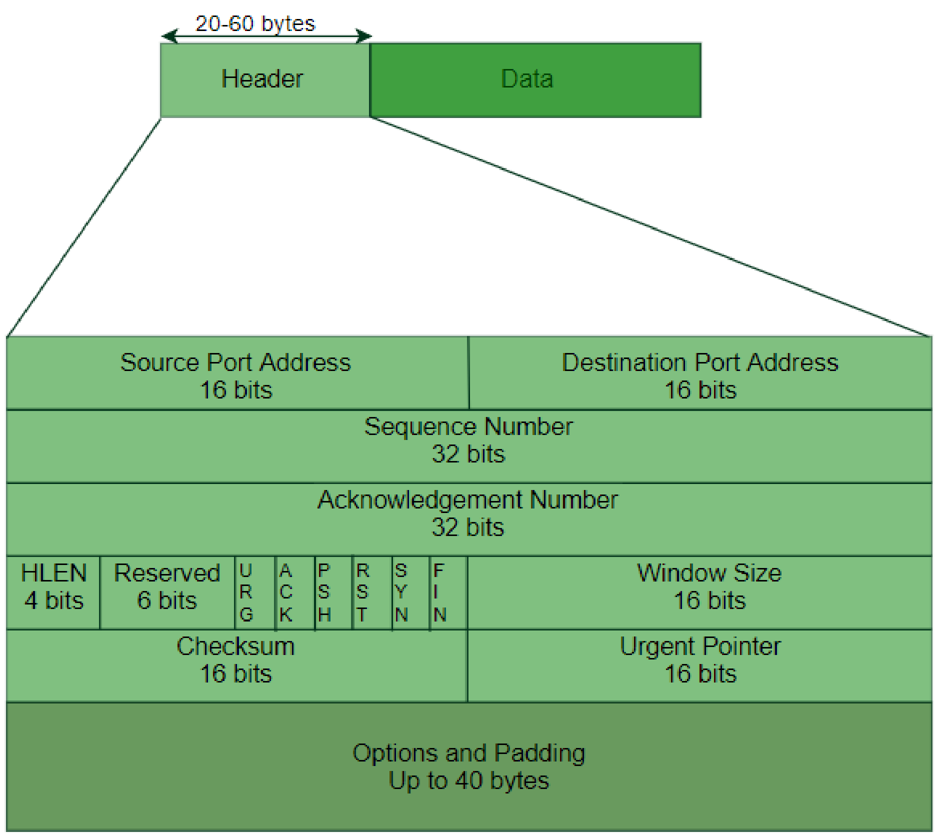
Version 6

```
User Datagram Protocol, Src Port: 546, Dst Port: 547
    Source Port: 546
    Destination Port: 547
    Length: 91
    Checksum: 0xcee1 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 1]
    [Timestamps]
    UDP payload (83 bytes)
```

Same udp frame

5. TCP (Transmission Control Protocol):

5.1. Frame structure:



5.1.1. Source Port Address:

a 16-bit field that holds the port address of the application that is sending the data segment.

5.1.2. Destination Port Address:

a 16-bit field that holds the port address of the application in the host that is receiving the data segment.

5.1.3. Sequence Number:

a 32-bit field that holds the sequence number, i.e, the byte number of the first byte that is sent in that particular segment. It is used to reassemble the message at the receiving end of the segments that are received out of order.

5.1.4. Acknowledgement Number:

a 32-bit field that holds the acknowledgement number, i.e, the byte number that the receiver expects to receive next. It is an acknowledgement for the previous bytes being received successfully.

5.1.5. Header Length (HLEN):

This is a 4-bit field that indicates the length of the TCP header by a number of 4-byte words in the header, i.e if the header is 20 bytes(min length of TCP header), then this field will hold 5 (because 5 x 4 = 20) and the maximum length: 60 bytes, then it'll hold the value 15(because 15 x 4 = 60). Hence, the value of this field is always between 5 and 15.

5.1.6. Control flags:

These are 6 1-bit control bits that control connection establishment, connection termination, connection abortion, flow control, mode of transfer etc.

5.1.7. Window size:

This field tells the window size of the sending TCP in bytes.

5.1.8. Checksum:

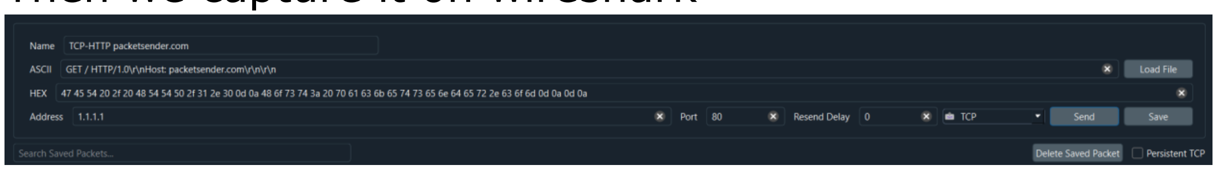
This field holds the checksum for error control. It is mandatory in TCP as opposed to UDP.

5.1.9. Urgent pointer:

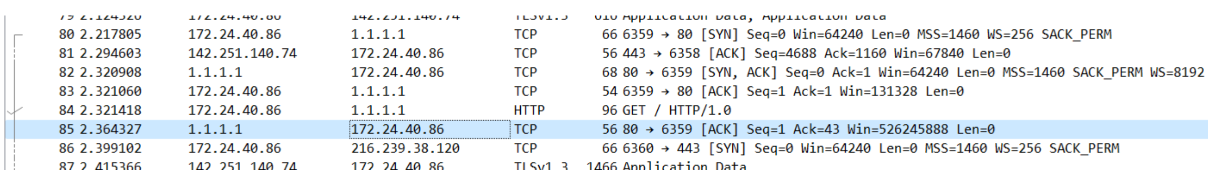
This field (valid only if the URG control flag is set) is used to point to data that is urgently required that needs to reach the receiving process at the earliest. The value of this field is added to the sequence number to get the byte number of the last urgent byte.

5.2. Capturing frames:

- We use packet sender app to send a tcp frame
- Then we capture it on wireshark



Sent segment



Logs for tcp protocol

- At first we see a request from our device to 1.1.1.1 and after that we see the response came back with our device as the destination and 1.1.1.1 as sender

```
Transmission Control Protocol, Src Port: 6359, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 6359
    Destination Port: 80
    [Stream index: 20]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 2406234210
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x002 (SYN)
    Window: 64240
    [Calculated window size: 64240]
    Checksum: 0xbcb9 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), N
  > [Timestamps]
```

Tcp segment of request
As you see the sequence number is set to zero which it means the connection has just started
And we have the syn flag for the Synchronize sequence numbers

```
Transmission Control Protocol, Src Port: 80, Dst Port: 6359, Seq: 0, Ack: 1, Len: 0
    Source Port: 80
    Destination Port: 6359
    [Stream index: 20]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 4108219120
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 2406234211
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x012 (SYN, ACK)
    Window: 64240
    [Calculated window size: 64240]
    Checksum: 0x54d4 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (12 bytes), Maximum segment size, No-Operation (NOP), No-Operation (NOP), SACK permitted,
  > [Timestamps]
```

Here you see the acknowledgment for the our request (ack flag is up too)

Note that the place of source and destination port has changed.

- Now we see the second segment which was sent just after the first segment

```
v Transmission Control Protocol, Src Port: 6359, Dst Port: 80, Seq: 1, Ack: 1, Len: 0
    Source Port: 6359
    Destination Port: 80
    [Stream index: 20]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 2406234211
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 4108219121
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window: 513
    [Calculated window size: 131328]
    [Window size scaling factor: 256]
    Checksum: 0x8e9c [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
```

As we can see the sequence number has incremented by one and the syn flag is no longer up (because synchronizes has been done at last segment)
Another thing to mention is that window size has gone up.

Then after this ack which was sent by out device we see another segment:

```
84 2.321418    172.24.40.86    1.1.1.1    HTTP    96 GET / HTTP/1.0
Transmission Control Protocol, Src Port: 6359, Dst Port: 80, Seq: 1, Ack: 1, Len: 42
    Source Port: 6359
    Destination Port: 80
    [Stream index: 20]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 42]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 2406234211
    [Next Sequence Number: 43    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 4108219121
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window: 513
    [Calculated window size: 131328]
    [Window size scaling factor: 256]
    Checksum: 0x8e6e [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
```

As you see the psh(request for push) flag has been added is to send data

After all communication between us and server The server sent this segment to terminate the connection:

```
92 2.435025    1.1.1.1         172.24.40.86    HTTP    461 HTTP/1.1 403 Forbidden  (text/plain)
93 2.444405    1.1.1.1         172.24.40.86    TCP     56 80 → 6359 [FIN, ACK] Seq=408 Ack=43 Win=65536 Len=0
94 2.444485    172.24.40.86    1.1.1.1         TCP     54 6359 → 80 [ACK] Seq=43 Ack=409 Win=130816 Len=0
Transmission Control Protocol, Src Port: 80, Dst Port: 6359, Seq: 408, Ack: 43, Len: 0
    Source Port: 80
    Destination Port: 6359
    [Stream index: 20]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 408    (relative sequence number)
    Sequence Number (raw): 4108219528
    [Next Sequence Number: 409    (relative sequence number)]
    Acknowledgment Number: 43    (relative ack number)
    Acknowledgment number (raw): 2406234253
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x011 (FIN, ACK)
    Window: 8
    [Calculated window size: 65536]
    [Window size scaling factor: 8192]
    Checksum: 0x8ed3 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
```

You can see that the FIN flag has been added which means termination of connection

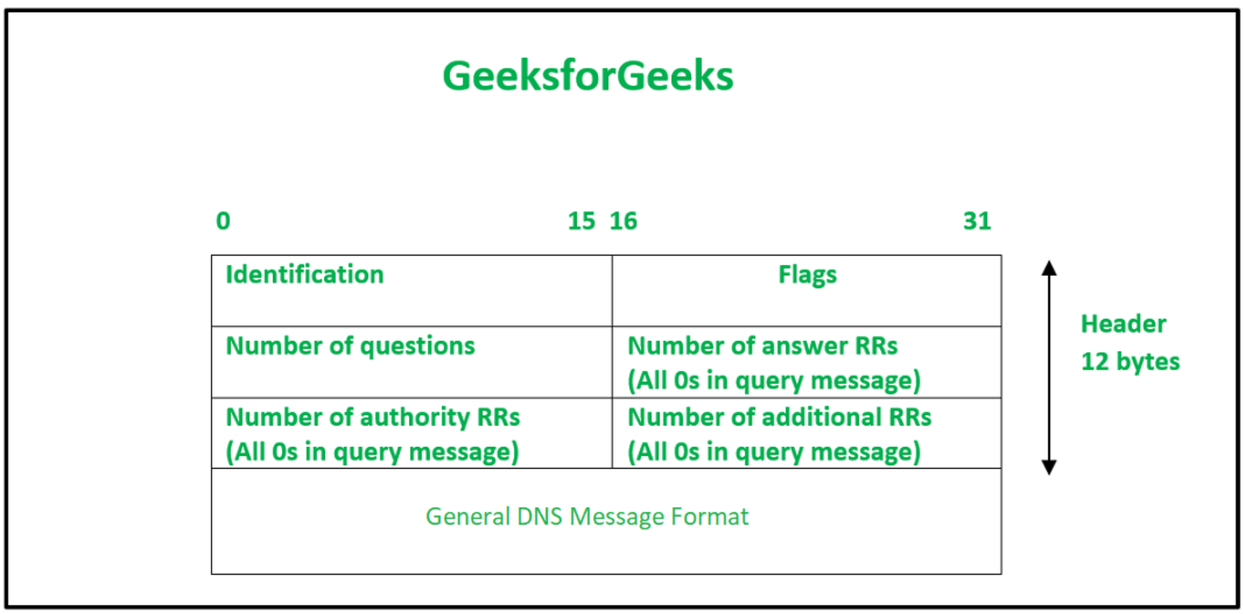
6. DNS (Domain Name System):

6.1. DNS structure:

DNS allows you to interact with devices on the Internet without having to remember long strings of numbers. Changing of information between client and server is carried out by two types of DNS messages:
Query message - Response message.

Query is sent by us and response is received by us intervally.

Query and response (almost) share the same structure:



6.1.1. Identification: The identification field is made up of 16 bits which are used to match the response with the request sent from the client-side. The matching is carried out by this field as the server copies the 16-bit value of identification in the response message so the client device can match the queries with the corresponding response received from the server-side.

6.1.2. Number of Questions- It is a 16-bit field to specify the count of questions in the Question Section of the message. It is present in both query and response messages.

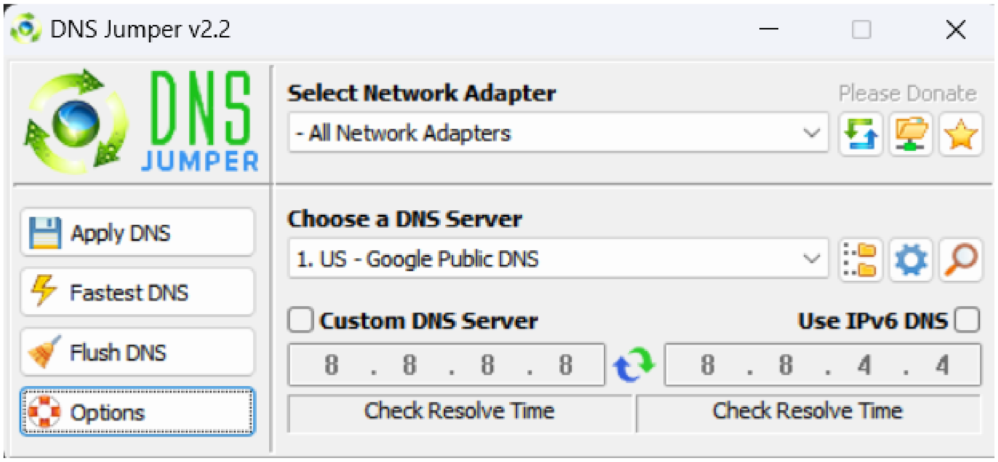
6.1.3. A number of answer RRs- It is a 16-bit field that specifies the count of answer records in the Answer section of the message. This section has a value of 0 in query messages. The server answers the query received from the client. It is available only in response messages.

6.1.4. A number of authority RRs- It is a 16-bit field that gives the count of the resource records in the Authoritative section of the message. This section has a value of 0 in query messages. It is available only in response messages. It gives information that comprises domain names about one or more authoritative servers.

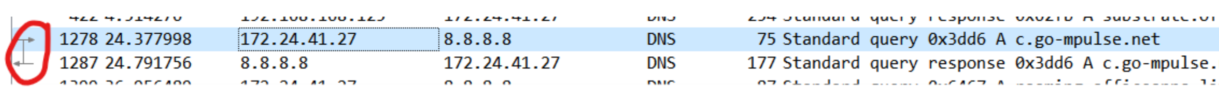
6.1.5. A number of additional RRs– It is a 16-bit field that holds additional records to keep additional information to help the resolver. This section has a value of 0 in query messages. It is available only in response messages.

6.2. Capture section:

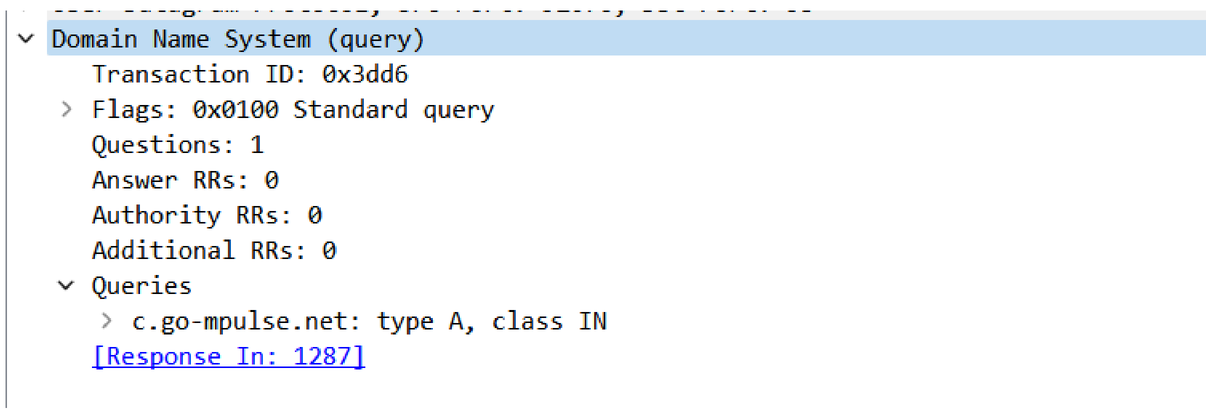
- On this section to check some DNS requests we try to change our DNS.
- You can change your DNS manually or you can change it using third-party softwars.
- I will be using DNS-jumper here:



- Select a server and then apply.



- These two sections shows the query and response message.
- For the query message the source is our IP and the destination is the IP of the selected server (8.8.8.8 or google.com).
- And for the response message the destination is our IP.



- Query:
- All sections match the known headers.
- There is a flag section that has multiple flags.
- For the query all you need is the standard query flag.
- As you can see you can directly reach to the response message which was below the query message.

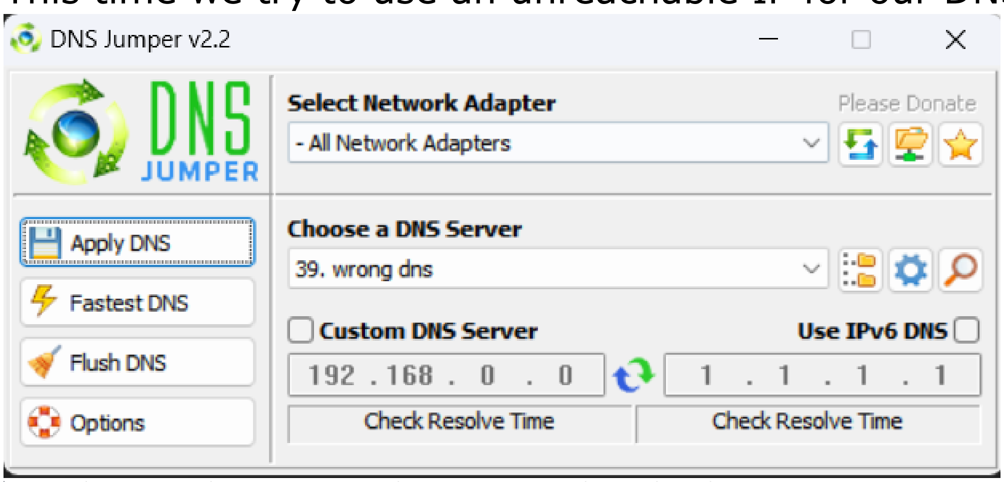
```
Domain Name System (response)
    Transaction ID: 0x3dd6
  > Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 3
    Authority RRs: 0
    Additional RRs: 0
  > Queries
  > Answers
    [Request In: 1278]
    [Time: 0.413758000 seconds]
```

- This is the response message.
- As you can see there is extra flags that determines the type of the message (for example you know that this one is a response and It has no errors)

```
  Queries
    > c.go-mpulse.net: type A, class IN
  Answers
    > c.go-mpulse.net: type CNAME, class IN, cname wildcard46.go-mpulse.net.edgekey.net
    > wildcard46.go-mpulse.net.edgekey.net: type CNAME, class IN, cname e4518.dscapi7.akamaiedge.net
    > e4518.dscapi7.akamaiedge.net: type A, class IN, addr 104.91.48.142
```

- The query section is the same query as the query message.
- And answer sections is filled too.

- Another example for DNS:
- This time we try to use an unreachable IP for our DNS:



579	14.492981	172.24.41.27	192.168.0.0	DNS	77	Standard query 0xd15f A k-ring.msedge.net
681	15.498102	172.24.41.27	1.1.1.1	DNS	77	Standard query 0xd15f A k-ring.msedge.net
683	15.600623	1.1.1.1	172.24.41.27	DNS	144	Standard query response 0xd15f A k-ring.msedge.net CNAI

- Our devices tries to reach to first IP

- But it was unreachable and there is no answer like the last example.
- After not reaching the first IP it tries the second IP.
- Which this time is reachable

681	15.498102	172.24.41.27	1.1.1.1	DNS	77	Standard query 0xd15f A k-ring.msedge.net
683	15.600623	1.1.1.1	172.24.41.27	DNS	144	Standard query response 0xd15f A k-ring.msedg
726	16.411254	172.24.41.27	1.1.1.1	DNS	88	Standard query 0x6758 A fp-afd-nocache.azuree