



# Redes de computadoras

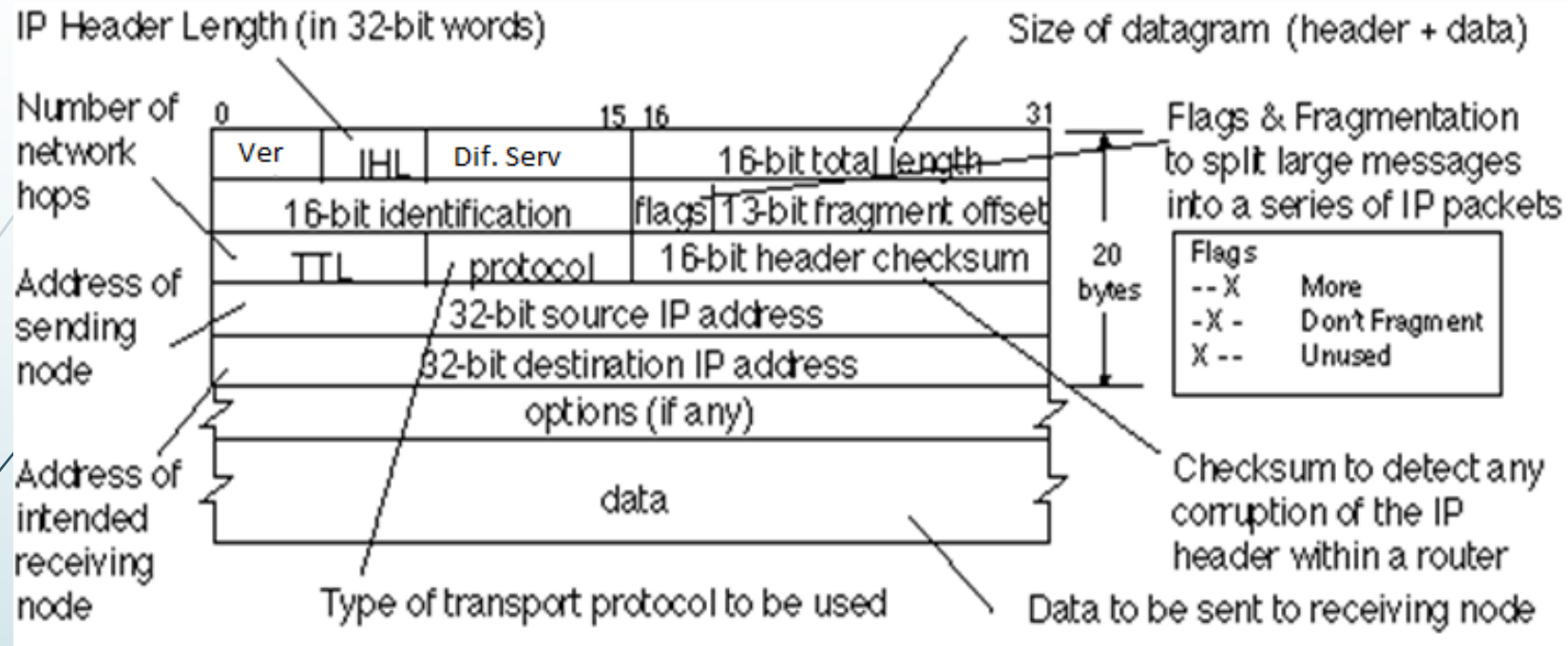
**Protocolos de capa de Red  
y Transporte**



# IP (Protocolo Internet)

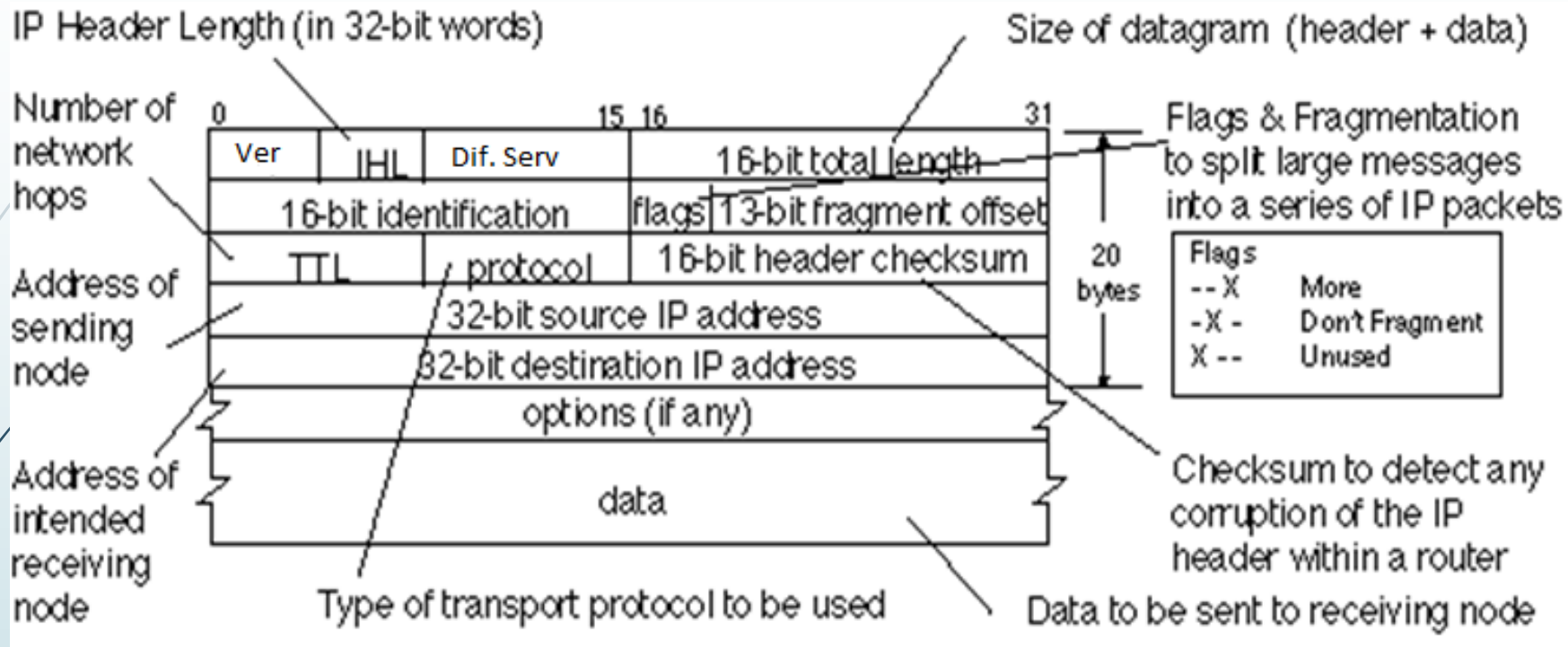
- Es un protocolo de capa de red, encargado de transportar datos a través de la red
- Realiza la entrega de datos basado en la premisa del mejor esfuerzo (no garantiza que los datos lleguen, o lleguen completos, o sin errores, o en orden, o sin duplicados)
- Tiene su especificación en RFC 791
- Para poder realizar la entrega de los datos cuenta con la siguiente información de encaminamiento:

# Protocolo IP



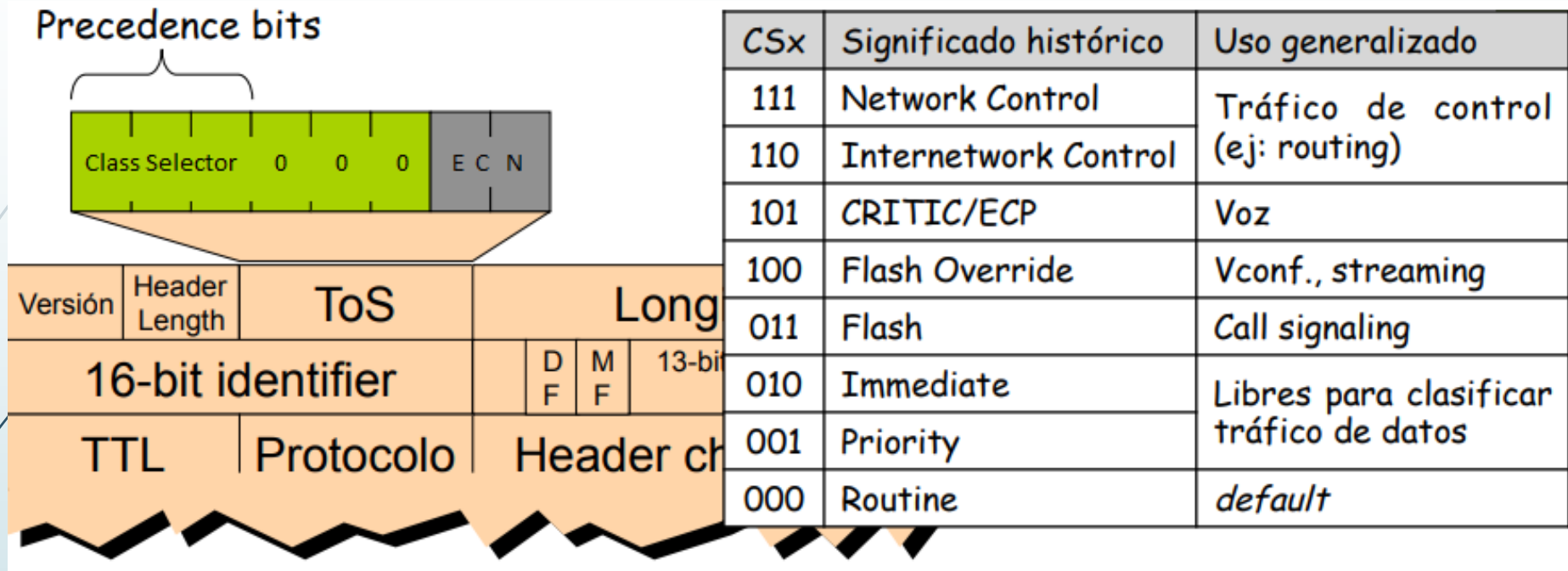
- Versión = 4 para IPv4, 6 para IPv6
- IHL = longitud de encabezado (palabras de 4 bytes)
- Longitud total = longitud del PDU IP (Encabezado + datos)

# Protocolo IP



- Servicios diferenciados = utilizados para intentar garantizar la calidad de servicio

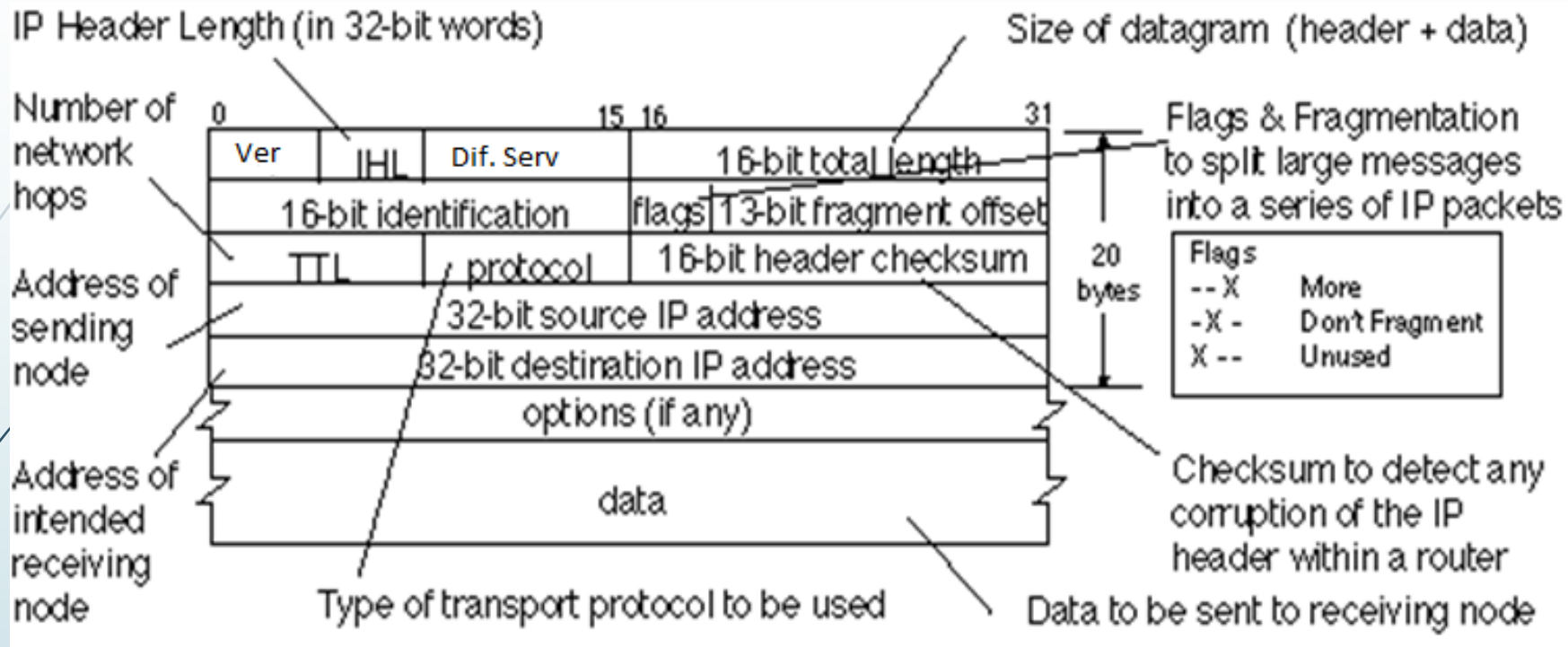
# Servicios Diferenciados



ECN = {

- 00 = Sin capacidad ECN
- 01 = Capacidad de transporte ECN (0)
- 10 = Capacidad de transporte ECN (1)
- 11 = Congestión encontrada

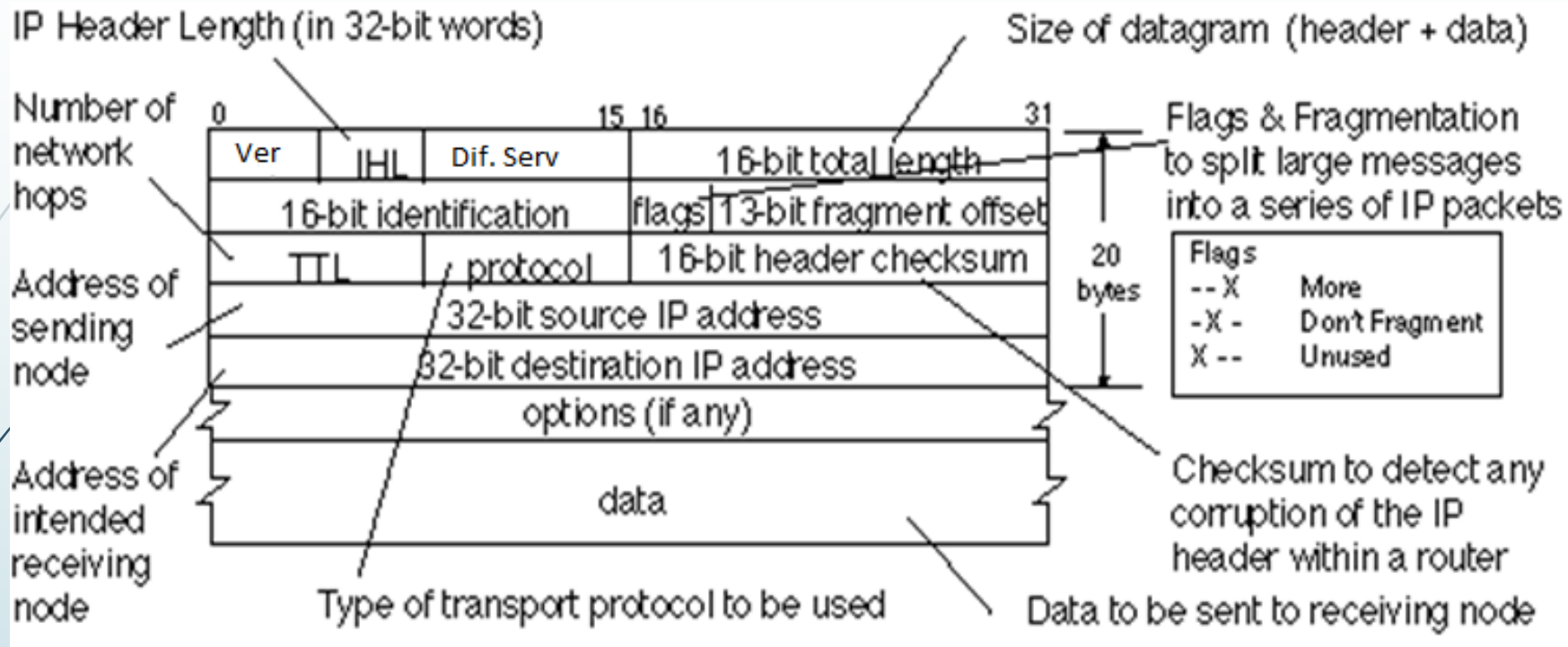
# Protocolo IP



- ID = identificador de paquete (usado en fragmentación)
- Banderas = 0 X Y
- Desplazamiento (offset) = posición de inicio de los datos del paquete
- Tiempo de vida (TTL)= # de enrutadores que pueden ser atravesados antes de descartar paquete

\*Fuente imagen: <https://blake.erg.abdn.ac.uk/users/gorry/course/images/ip-header.gif>

# Protocolo IP

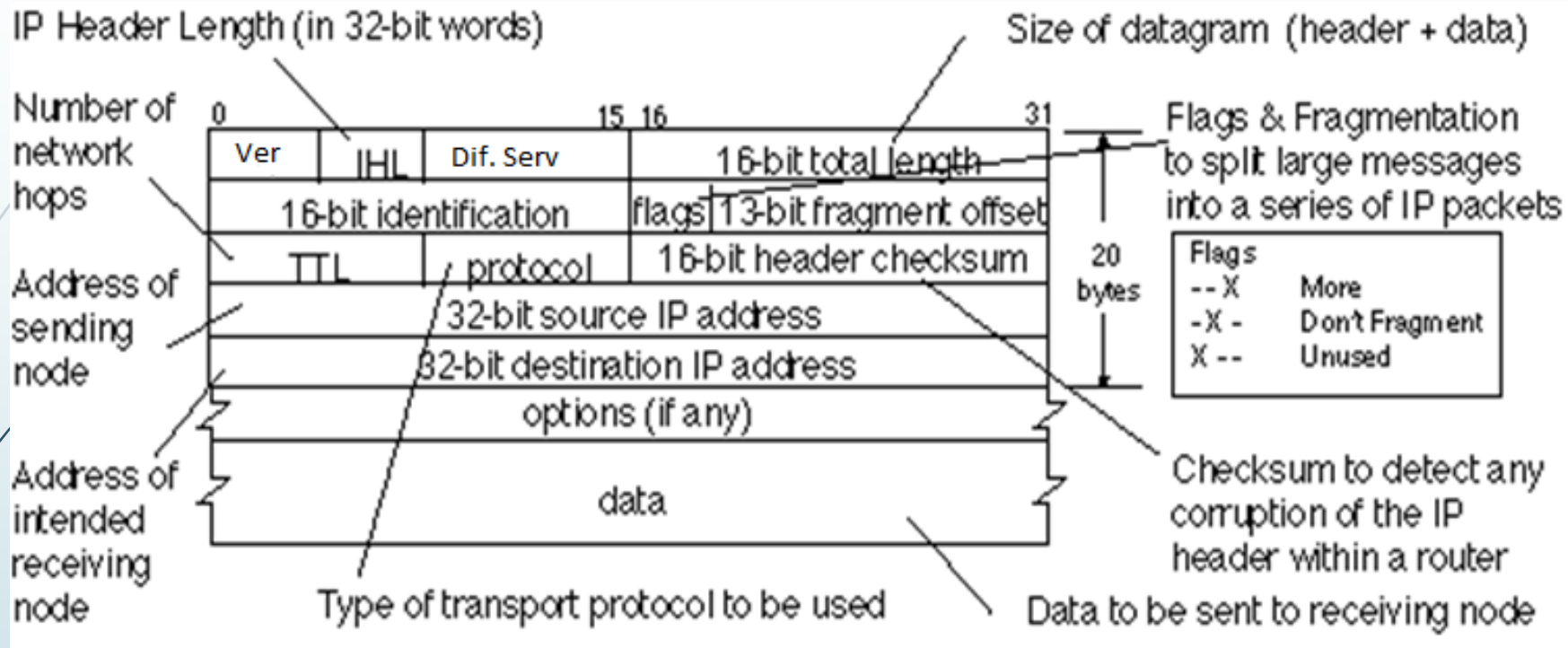


- Protocolo = identificador del protocolo de capa de Transporte/Red encapsulado en el paquete IP
- Checksum = Suma de comprobación del encabezado IP
- Dirección IP origen = Identificador de la máquina emisora del paquete IP
- Dirección IP destino = Identificador de la máquina receptora del paquete IP

\*Fuente imagen: <https://blake.erg.abdn.ac.uk/users/gorry/course/images/ip-header.gif>



# Protocolo IP



- Opciones:
  - Etiquetado de seguridad (uso militar, RFC 1108)
  - Enrutamiento de origen
  - Registro de ruta (ej. Ping -r)



# Análisis de IP en Npcap

## ► Uso de estructuras


```
/* Dirección IP */  
typedef struct dir_IP{  
    u_char byte1;  
    u_char byte2;  
    u_char byte3;  
    u_char byte4;  
}dir_IP;
```

# Análisis de IP en Npcap

► Uso de estructuras

```
/* Encabezado de IPv4 */
typedef struct ip_enc{
    u_char  ver_ihl;           // Version (4 bits) + Longitud encabezado IP (4 bits)
    u_char  tos;               // Tipo de servicio
    u_short tlen;              // Longitud total PDU
    u_short id;                // Identificación
    u_short flags_fo;          // Flags (3 bits) + Fragment offset (13 bits)
    u_char  ttl;               // Tiempo de vida
    u_char  proto;              // Protocolo
    u_short crc;               // Suma de comprobación encabezado IP
    dir_IP  saddr;              // Dirección IP origen
    dir_IP  daddr;              // Dirección IP destino
    u_int   op_pad;             // Opciones + relleno
}ip_enc;
```

\*Fuente imagen: <https://blake.erg.abdn.ac.uk/users/gorry/course/images/ip-header.gif>



```
void packet_handler(u_char *param, const struct pcap_pkthdr *header, const u_char *pkt_data) {  
    //...
```

```
    unsigned short tipo = (pkt_data[12]*256)+pkt_data[13];  
    if (tipo==2048){
```

```
        printf("Paquete IP.\n");
```

```
        ip_enc *ip;
```

```
        u_int ip_longitud;
```

```
        /* obtenemos la posición de inicio de IP */
```

```
        ip = (ip_enc *) (pkt_data + 14); //length of ethernet header
```

```
        printf("%d.%d.%d.%d -> %d.%d.%d.%d\n",
```

```
            ip->saddr.byte1,
```

```
            ip->saddr.byte2,
```

```
            ip->saddr.byte3,
```

```
            ip->saddr.byte4,
```

```
            ip->daddr.byte1,
```

```
            ip->daddr.byte2,
```

```
            ip->daddr.byte3,
```

```
            ip->daddr.byte4);
```

# Análisis de IP en PCAP4J

- Clase `IPv4Packet` (`org.pcap4j.packet.IpV4Packet`) (clase abstracta)

## Métodos

- + `static IpV4Packet newPacket(byte[] rawData, int offset, int length)`
- + `IpV4Packet.IpV4Header getHeader()`

# Análisis de IP en PCAP4J

- Clase `IPv4Header` (`org.pcap4j.packet.IpV4Packet.IpV4Header`) (clase abstracta)

## Métodos

+ <code>IpVersion</code>	<code>getVersion()</code> <code>//.valueAsString()</code>
+ <code>int</code>	<code>getIhlAsInt()</code>
+ <code>IpV4Packet.IpV4Tos</code>	<code>getTos()</code> <code>//.toString()</code>
+ <code>int</code>	<code>getTotalLengthAsInt()</code>
+ <code>int</code>	<code>getIdentificationAsInt()</code>
+ <code>boolean</code>	<code>getDontFragmentFlag()</code>
+ <code>boolean</code>	<code>getMoreFragmentFlag()</code>
+ <code>short</code>	<code>getFragmentOffset()</code>
+ <code>int</code>	<code>getTtlAsInt()</code>
+ <code>IpNumber</code>	<code>getProtocol()</code> <code>//.name()</code> <code>//.value()</code>

# Análisis de IP en PCAP4J

- Clase `IPv4Header` (`org.pcap4j.packet.IpV4Packet.IpV4Header`) (clase abstracta)

## Métodos

```
+ short                getHeaderChecksum()
+ Inet4Address          getSrcAddr()    //.getHostAddress()
+ Inet4Address          getDstAddr()    //.getHostAddress()
+ List<IpV4Packet.IpV4Option>  getOptions()
    - IpV4Option (Interfaz)
      + IpV4OptionType  getType()

//
https://www.javadoc.io/doc/org.pcap4j/pcap4j/1.7.1/org/pcap4j/packet/namednumber/IpV4OptionType.html
```

# Análisis de IP en PCAP4J

```
while (true) {  
    ➤ Ej. byte[] packet = handle.getNextRawPacket();  
    if (packet == null) {  
        continue;  
    } else {  
        int tipo_b1 = (packet[12]>=0)?packet[12]*256:(packet[12]+256)*256;  
        int tipo_b2 = (packet[13]>=0)?packet[13]:packet[13]+256;  
        int tipo = tipo_b1+tipo_b2;  
        System.out.println("\nTipo="+tipo);  
        switch(tipo){  
            case(int)2048: {  
                try {  
                    //IP  
                    int ihl = (packet[14]&0x0f)*4;  
                    System.out.println("tam paquete IP:"+ihl+" bytes");  
                }  
            }  
        }  
    }  
}
```

**\*Fuente:** <https://github.com/kaitoy/pcap4j/raw/v1/pcap4j-sample/src/main/java/org/pcap4j/sample/LoopRaw.java>



# Análisis de IP en PCAP4J

► Ej.

```
byte[] tmp_ip = Arrays.copyOfRange(packet, 14, 14+ihl);
IPv4Packet ip =IPv4Packet.newPacket(tmp_ip, 0, tmp_ip.length);
System.out.println("Version: "+ip.getHeader().getVersion().valueAsString());
System.out.println("IHL: "+ip.getHeader().getIhlAsInt());
System.out.println("Serv. Dif: "+ip.getHeader().getTos().toString());

int lt =
(ip.getHeader().getTotalLength()>0)?ip.getHeader().getTotalLength():ip.getHeader().getTotalLength()+65536;
System.out.println("Longitud total: "+lt);

int id=
(ip.getHeader().getIdentification()>0)?ip.getHeader().getIdentification():ip.getHeader().getIdentification(
)+65536;
System.out.println("Id: "+id);
String df=(ip.getHeader().getDontFragmentFlag())?"Encendido":"Apagado";
```

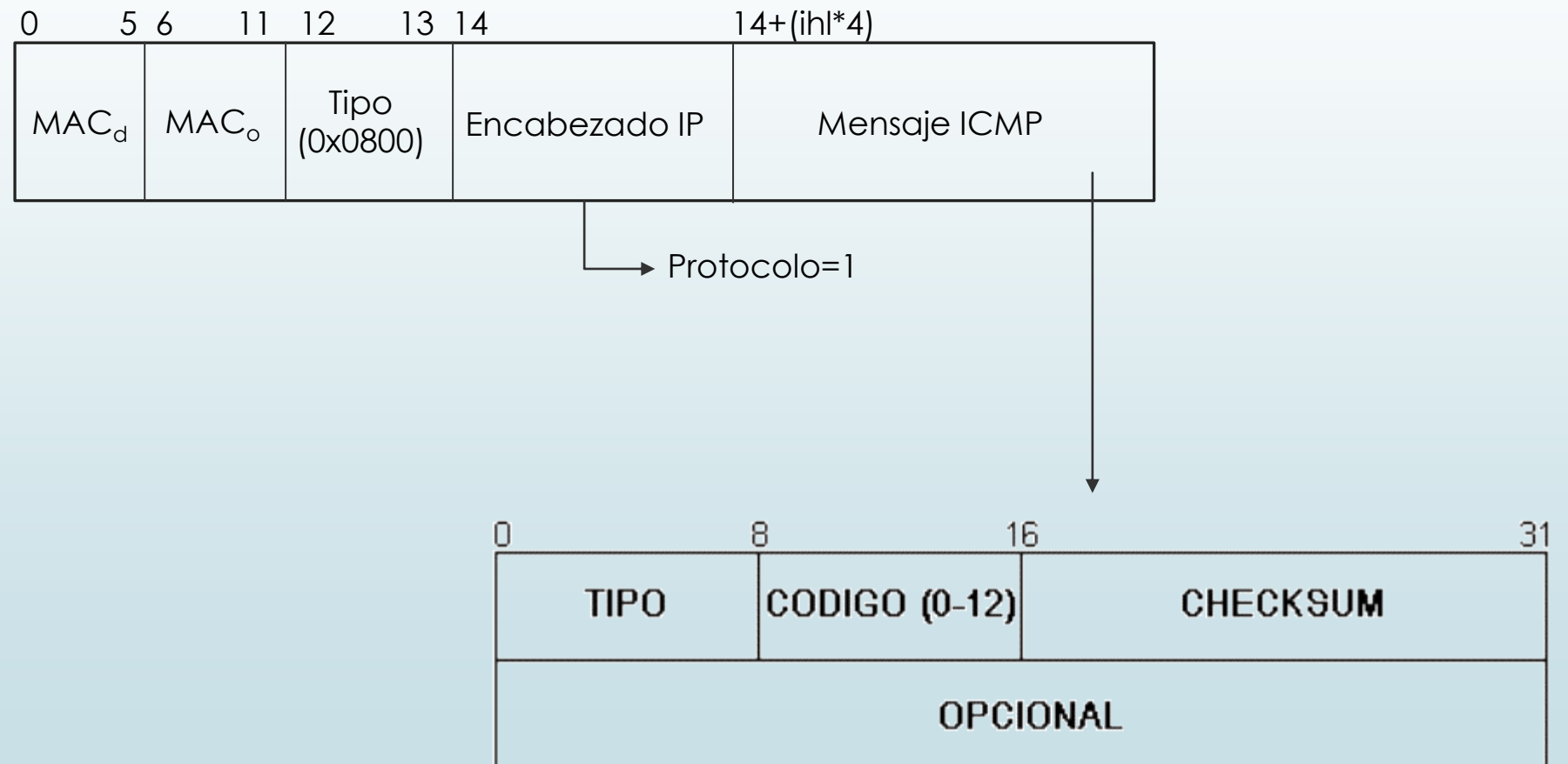
**\*Fuente:** <https://github.com/kaitoy/pcap4j/raw/v1/pcap4j-sample/src/main/java/org/pcap4j/sample/LoopRaw.java>



# ICMP (Protocolo de Mensajes de Control de Internet, *Internet Control Message Protocol*)

- Protocolo utilizado para la transmisión de mensajes de control (eco, máscara de red, marca de tiempo, etc.) y el informe de condiciones de error (ttl excedido en tránsito, embotellamiento, destino inalcanzable, etc.), en una comunicación host a host
- Escrito por Jon Postel (Uno de los fundadores de Internet) en 1981, tiene su especificación en RFC 777, 760, **792**
- Este protocolo forma parte de la pila TCP/IP y opera en la capa de red encapsulado dentro de un paquete IP (**protocolo=1**)

# Formato de mensaje ICMP

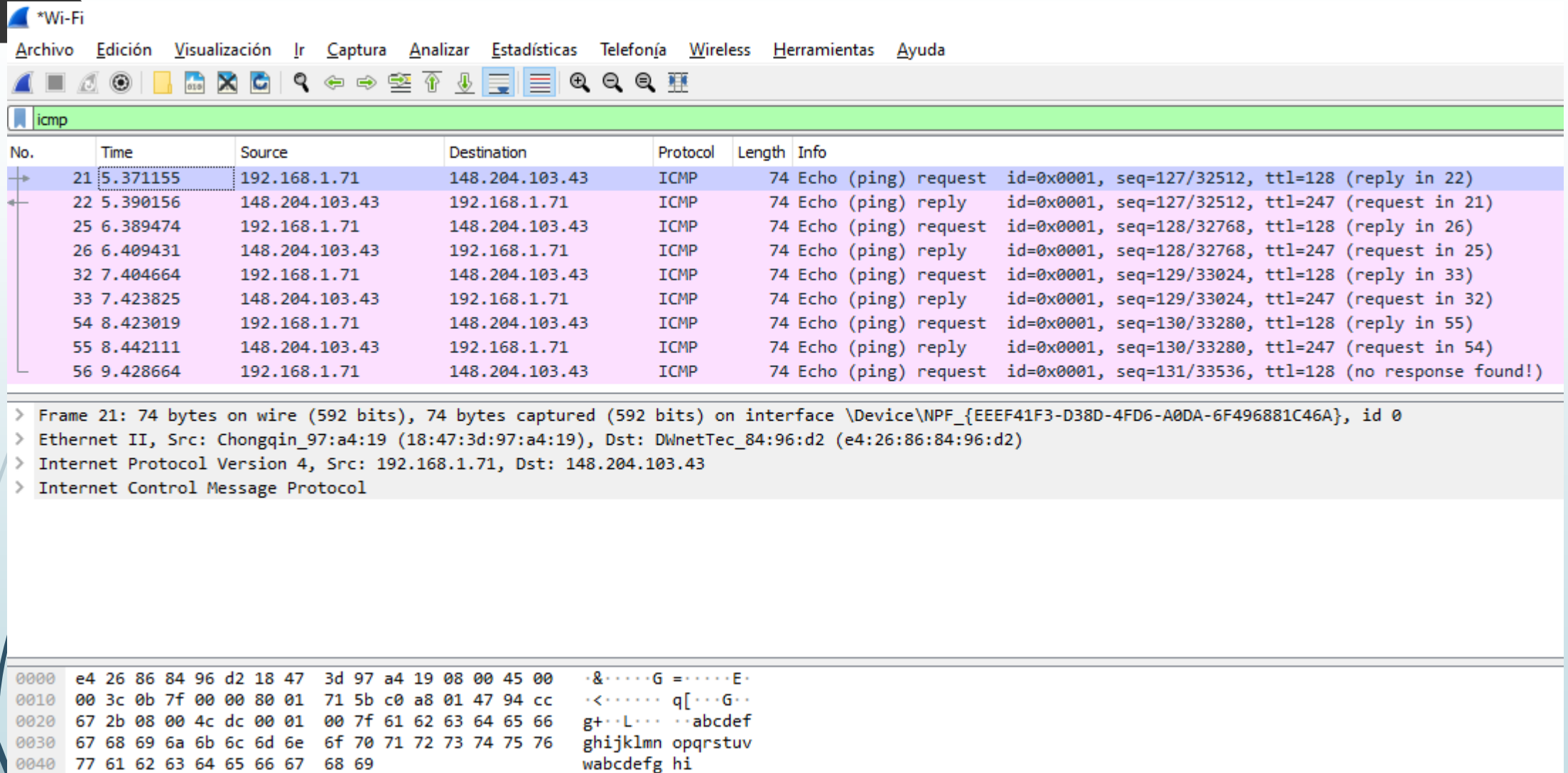


**\*Imagen de creación propia a partir de la información de RFC 777, RFC 760, RFC 792**

Type	Code	Description
0 – Echo Reply	0	Echo reply
3 – Destination Unreachable	0	Destination network unreachable
	1	Destination host unreachable
	2	Destination protocol unreachable
	3	Destination port unreachable
	4	Fragmentation needed and DF flag set
	5	Source route failed
5 – Redirect Message	0	Redirect datagram for the Network
	1	Redirect datagram for the host
	2	Redirect datagram for the Type of Service and Network
	3	Redirect datagram for the Service and Host
8 – Echo Request	0	Echo request
9 – Router Advertisement	0	Use to discover the addresses of operational routers
10 – Router Solicitation	0	
11 – Time Exceeded	0	Time to live exceeded in transit
	1	Fragment reassembly time exceeded
12 – Parameter Problem	0	Pointer indicates error
	1	Missing required option
	2	Bad length
13 – Timestamp	0	Used for time synchronization
14 – Timestamp Reply	0	Reply to Timestamp message

\*Fuente de la imagen: [https://miro.medium.com/v2/resize:fit:720/format:webp/1\\*\\_GK8lxZws3JZNyHtJMEg5A.png](https://miro.medium.com/v2/resize:fit:720/format:webp/1*_GK8lxZws3JZNyHtJMEg5A.png)

# Ejemplo wireshark



\*Wi-Fi

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

icmp

No.	Time	Source	Destination	Protocol	Length	Info
21	5.371155	192.168.1.71	148.204.103.43	ICMP	74	Echo (ping) request id=0x0001, seq=127/32512, ttl=128 (reply in 22)
22	5.390156	148.204.103.43	192.168.1.71	ICMP	74	Echo (ping) reply id=0x0001, seq=127/32512, ttl=247 (request in 21)
25	6.389474	192.168.1.71	148.204.103.43	ICMP	74	Echo (ping) request id=0x0001, seq=128/32768, ttl=128 (reply in 26)
26	6.409431	148.204.103.43	192.168.1.71	ICMP	74	Echo (ping) reply id=0x0001, seq=128/32768, ttl=247 (request in 25)
32	7.404664	192.168.1.71	148.204.103.43	ICMP	74	Echo (ping) request id=0x0001, seq=129/33024, ttl=128 (reply in 33)
33	7.423825	148.204.103.43	192.168.1.71	ICMP	74	Echo (ping) reply id=0x0001, seq=129/33024, ttl=247 (request in 32)
54	8.423019	192.168.1.71	148.204.103.43	ICMP	74	Echo (ping) request id=0x0001, seq=130/33280, ttl=128 (reply in 55)
55	8.442111	148.204.103.43	192.168.1.71	ICMP	74	Echo (ping) reply id=0x0001, seq=130/33280, ttl=247 (request in 54)
56	9.428664	192.168.1.71	148.204.103.43	ICMP	74	Echo (ping) request id=0x0001, seq=131/33536, ttl=128 (no response found!)

> Frame 21: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF\_{EEEE41F3-D38D-4FD6-A0DA-6F496881C46A}, id 0

> Ethernet II, Src: Chongqin\_97:a4:19 (18:47:3d:97:a4:19), Dst: DWnetTec\_84:96:d2 (e4:26:86:84:96:d2)

> Internet Protocol Version 4, Src: 192.168.1.71, Dst: 148.204.103.43

> Internet Control Message Protocol

0000	e4 26 86 84 96 d2 18 47 3d 97 a4 19 08 00 45 00	·&·.....G =.....E·
0010	00 3c 0b 7f 00 00 80 01 71 5b c0 a8 01 47 94 cc	·<..... q[...G·
0020	67 2b 08 00 4c dc 00 01 00 7f 61 62 63 64 65 66	g+...L... ··abcdef
0030	67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76	ghijklmn opqrstuv
0040	77 61 62 63 64 65 66 67 68 69	wabcdefg hi

\* Imagen generada a partir de captura de pantalla de aplicación wireshark en ejecución

# Ejemplo wireshark

> Frame 21: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF\_{EEEF41F3-D38D-4FD6-A0DA-6F496881C46A}, id 0  
> Ethernet II, Src: Chongqin\_97:a4:19 (18:47:3d:97:a4:19), Dst: DWnetTec\_84:96:d2 (e4:26:86:84:96:d2)  
✓ Internet Protocol Version 4, Src: 192.168.1.71, Dst: 148.204.103.43  
    0100 .... = Version: 4  
    .... 0101 = Header Length: 20 bytes (5)  
    > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)  
    Total Length: 60  
    Identification: 0x0b7f (2943)  
    > Flags: 0x00  
    Fragment Offset: 0  
    Time to Live: 128  
    Protocol: ICMP (1)  
    Header Checksum: 0x715b [validation disabled]  
    [Header checksum status: Unverified]  
    Source Address: 192.168.1.71  
    Destination Address: 148.204.103.43

---

0000	e4 26 86 84 96 d2 18 47 3d 97 a4 19 08 00 45 00	·&······G =·····E·
0010	00 3c 0b 7f 00 00 80 01 71 5b c0 a8 01 47 94 cc	·<······q[···G··
0020	67 2b 08 00 4c dc 00 01 00 7f 61 62 63 64 65 66	g+··L··· ··abcdef
0030	67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76	ghijklmn opqrstuv
0040	77 61 62 63 64 65 66 67 68 69	wabcdefg hi

\* Imagen generada a partir de captura de pantalla de aplicación wireshark en ejecución

\*Wi-Fi

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

icmp

No.	Time	Source	Destination	Protocol	Length	Info
21	5.371155	192.168.1.71	148.204.103.43	ICMP	74	Echo (ping) request id=0x0001, seq=127/32512, ttl=128 (reply in 22)
22	5.390156	148.204.103.43	192.168.1.71	ICMP	74	Echo (ping) reply id=0x0001, seq=127/32512, ttl=247 (request in 21)
25	6.389474	192.168.1.71	148.204.103.43	ICMP	74	Echo (ping) request id=0x0001, seq=128/32768, ttl=128 (reply in 26)
26	6.409431	148.204.103.43	192.168.1.71	ICMP	74	Echo (ping) reply id=0x0001, seq=128/32768, ttl=247 (request in 25)
32	7.404664	192.168.1.71	148.204.103.43	ICMP	74	Echo (ping) request id=0x0001, seq=129/33024, ttl=128 (reply in 33)
33	7.423825	148.204.103.43	192.168.1.71	ICMP	74	Echo (ping) reply id=0x0001, seq=129/33024, ttl=247 (request in 32)
54	8.423019	192.168.1.71	148.204.103.43	ICMP	74	Echo (ping) request id=0x0001, seq=130/33280, ttl=128 (reply in 55)
55	8.442111	148.204.103.43	192.168.1.71	ICMP	74	Echo (ping) reply id=0x0001, seq=130/33280, ttl=247 (request in 54)
56	9.428664	192.168.1.71	148.204.103.43	ICMP	74	Echo (ping) request id=0x0001, seq=131/33536, ttl=128 (no response found!)

Protocol: ICMP (1)  
Header Checksum: 0x715a [validation disabled]  
[Header checksum status: Unverified]  
Source Address: 192.168.1.71  
Destination Address: 148.204.103.43

Internet Control Message Protocol

Type: 8 (Echo (ping) request)  
Code: 0  
Checksum: 0x4cdb [correct]  
[Checksum Status: Good]  
Identifier (BE): 1 (0x0001)  
Identifier (LE): 256 (0x0100)  
Sequence Number (BE): 128 (0x0080)  
Sequence Number (LE): 32768 (0x8000)  
[\[Response frame: 26\]](#)

Data (32 bytes)

0000	e4 26 86 84 96 d2 18 47 3d 97 a4 19 08 00 45 00	·&·.....G =.....E·
0010	00 3c 0b 80 00 00 80 01 71 5a c0 a8 01 47 94 cc	·<..... qZ···G··
0020	67 2b 08 00 4c db 00 01 00 80 61 62 63 64 65 66	g+···L·· ··abcdef
0030	67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76	ghijklmn opqrstuv
0040	77 61 62 63 64 65 66 67 68 69	wabcedfg hi

Checksum (icmp.checksum), 2 byte(s)

Paquetes: 56 · Mostrado: 17 (30.4%) · Perdido: 0 (0.0%)

Perfil: Default

07:48 p. m.  
12/01/2021

\* Imagen generada a partir de captura de pantalla de aplicación wireshark en ejecución



# Análisis de ICMP en Npcap

## ► Uso de estructuras

```
/* dirección IPv4 */
```

```
typedef struct dir_IP{  
    u_char byte1;  
    u_char byte2;  
    u_char byte3;  
    u_char byte4;  
}dir_IP;
```

```
/* encabezado IPv4 */
```

```
typedef struct ip_enc{  
    u_char  ver_ihl;           // Version (4 bits) + IP header length (4 bits)  
    u_char  tos;              // Type of service  
    u_short tlen;             // Total length  
    u_short id;               // Identification  
    u_short flags_fo;         // Flags (3 bits) + Fragment offset (13 bits)  
    u_char  ttl;              // Time to live  
    u_char  proto;            // Protocol  
    u_short crc;              // Header checksum  
    ip_address saddr;         // Source address  
    ip_address daddr;         // Destination address  
    u_int    op_pad;          // Option + Padding  
}ip_enc;
```

\*Fuente: <https://npcap.com/guide/npcap-tutorial.html>

# Análisis de ICMP en Npcap

## ► Uso de estructuras

```
/* ICMP header*/  
typedef struct icmp_enc{  
    u_char tipo;           // ICMP tipo  
    u_char codigo;        // ICMP código  
    u_short crc;           // Checksum  
}icmp_enc;
```

\*Fuente: <https://npcap.com/guide/npcap-tutorial.html>

```
void packet_handler(u_char *param, const struct pcap_pkthdr *header, const u_char *pkt_data) {  
//. . .
```

```
    unsigned short tipo = (pkt_data[12]*256)+pkt_data[13];
```

```
    if (tipo==2048){
```

```
        printf("Paquete IP..\n");
```

```
        ip_enc *ip;
```

```
        u_int ip_len;
```

```
        /* retireve the position of the ip header */
```

```
        ip = (ip_enc *) (pkt_data + 14);
```

```
        printf("%d.%d.%d.%d -> %d.%d.%d.%d\n",
```

```
            ip->saddr.byte1,
```

```
            ip->saddr.byte2,
```

```
            ip->saddr.byte3,
```

```
            ip->saddr.byte4,
```

```
            ip->daddr.byte1,
```

```
            ip->daddr.byte2,
```

```
            ip->daddr.byte3,
```

```
            ip->daddr.byte4);
```

```
    If (ip->proto==1){
```

```
        icmp_enc *icmp;
```

```
        u_char ihl = ((ip->ver_ihl)&0x0f)*4;
```

```
        icmp = (icmp_enc *) (pkt_data + 14+(ihl));
```

```
        printf("Tipo: %d Codigo:%d\n",icmp->type, icmp->code);
```

```
    }//if
```

```
}//if
```

**\*Fuente:** <https://npcap.com/guide/npcap-tutorial.html>

# Análisis de ICMP en Pcap4j

- Clase `IcmpV4CommonPacket` (`org.pcap4j.packet.IcmpV4CommonPacket`)

## Métodos

```
+ IcmpV4CommonPacket.Builder  getBuilder() //objeto Builder
+ Packet                      getPayload()
+ static IcmpV4CommonPacket    newPacket(byte[] rawData, int offset, int length)
+ IcmpV4CommonPacket.IcmpV4CommonHeader  getHeader()
```

# Análisis de ICMP en Pcap4j

- Clase `IcmpV4CommonPacket.IcmpV4CommonHeader`  
(`org.pcap4j.packet.IcmpV4CommonPacket.IcmpV4CommonHeader`)

## Métodos

```
+ IcmpV4Type getType() // .valueAsString() ó .name()  
+ IcmpV4Code getCode() // .valueAsString() ó .name()  
+ short getChecksum()
```

# Análisis de ICMP en Jnetpcap

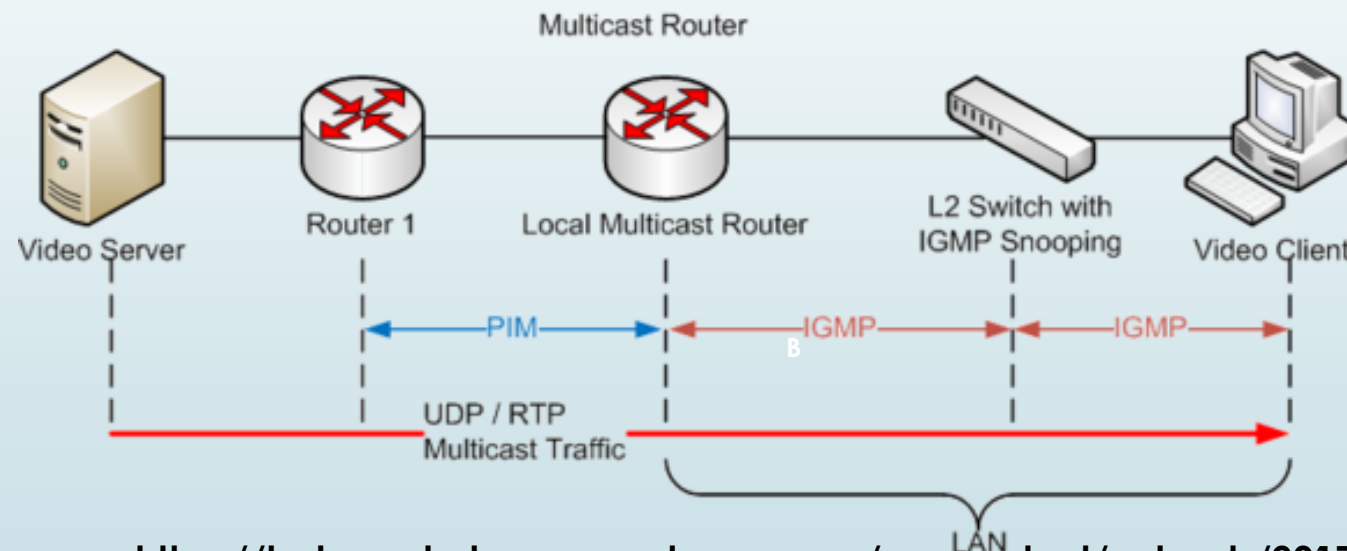
► Ej.

```
int tipo_b1 = (packet[12]>=0)?packet[12]*256:(packet[12]+256)*256;
int tipo_b2 = (packet[13]>=0)?packet[13]:packet[13]+256;
int tipo = tipo_b1+tipo_b2;
switch(tipo){
    case(int)2048: {
        byte[] tmp_ip = Arrays.copyOfRange(packet, 14, 14+ihl);
        IPv4Packet ip =IPv4Packet.newPacket(tmp_ip, 0, tmp_ip.length);
        int proto=ip.getHeader().getProtocol().value().intValue();
        switch(proto){
            case (int)1:
                byte[] tmp_icmp = Arrays.copyOfRange(packet, 14+ihl, 14+ihl+lt_PDU_Transp);
                IcmpV4CommonPacket icmp =IcmpV4CommonPacket.newPacket(tmp_icmp, 0, tmp_icmp.length);
                System.out.println("Tipo:
"+icmp.getHeader().getType().valueAsString()+" (" +icmp.getHeader().getType().name()+")");
```

**\*Fuente:** <https://github.com/kaitoy/pcap4j/raw/v1/pcap4j-sample/src/main/java/org/pcap4j/sample/LoopRaw.java>

# IGMP (Protocolo de Gestión de Grupos de Internet, *Internet Group Management Protocol*)

- Protocolo IGMP es utilizado para gestionar la pertenencia a grupos de multidifusión en redes LAN
- Tiene su especificación en RFC 1112(IGMPv1), 2236(IGMPv2), 3376(IGMPv3)
- Este protocolo forma parte de la pila TCP/IP y opera en la capa de red encapsulado dentro de un paquete IP (**protocolo=2**)



\*Fuente de la imagen: <https://todopacketracer.wordpress.com/wp-content/uploads/2017/08/igmp.jpg?w=300>



# Versiones de IGMP

- **IGMPv1 (RFC 1112):** Los Host pueden unirse a grupos de Multicast. **No hay mensajes de abandono** del grupo. Los enrutadores procesan las bajas del grupo usando el mecanismo **Time-out** (260 seg.) para descubrir los host que ya no están interesados en ser miembros.
- **IGMPv2 (RFC2236):** Añade la **capacidad de abandonar un grupo** al protocolo, permitiendo a los miembros del grupo abandonar activamente un grupo Multicast.
- **IGMPv3 (RFC 3376):** introduce la **seguridad gracias a las fuentes de multidifusión seleccionables**. Una revisión mayor del protocolo, que permite a los host especificar el origen deseado de tráfico Multicast. El tráfico que viene de otros host es bloqueado. Esto permite a los host bloquear paquetes que vienen desde fuentes que envían tráfico indeseado.

\*Fuente: <https://netseccloud.com/igmp-v2-vs-v3>

# Funcionamiento IGMP

- Cuando una aplicación de la capa de aplicación decide transmitir vía multicast (UDP, direcciones clase D):
  - La dirección IP multicast se mapea en una dirección MAC multicast y la interfaz de red comienza a escuchar dicha dirección además de su propia dirección MAC

Ej. ( 224.1.1.1) →(01:00:5e:01:01:01)

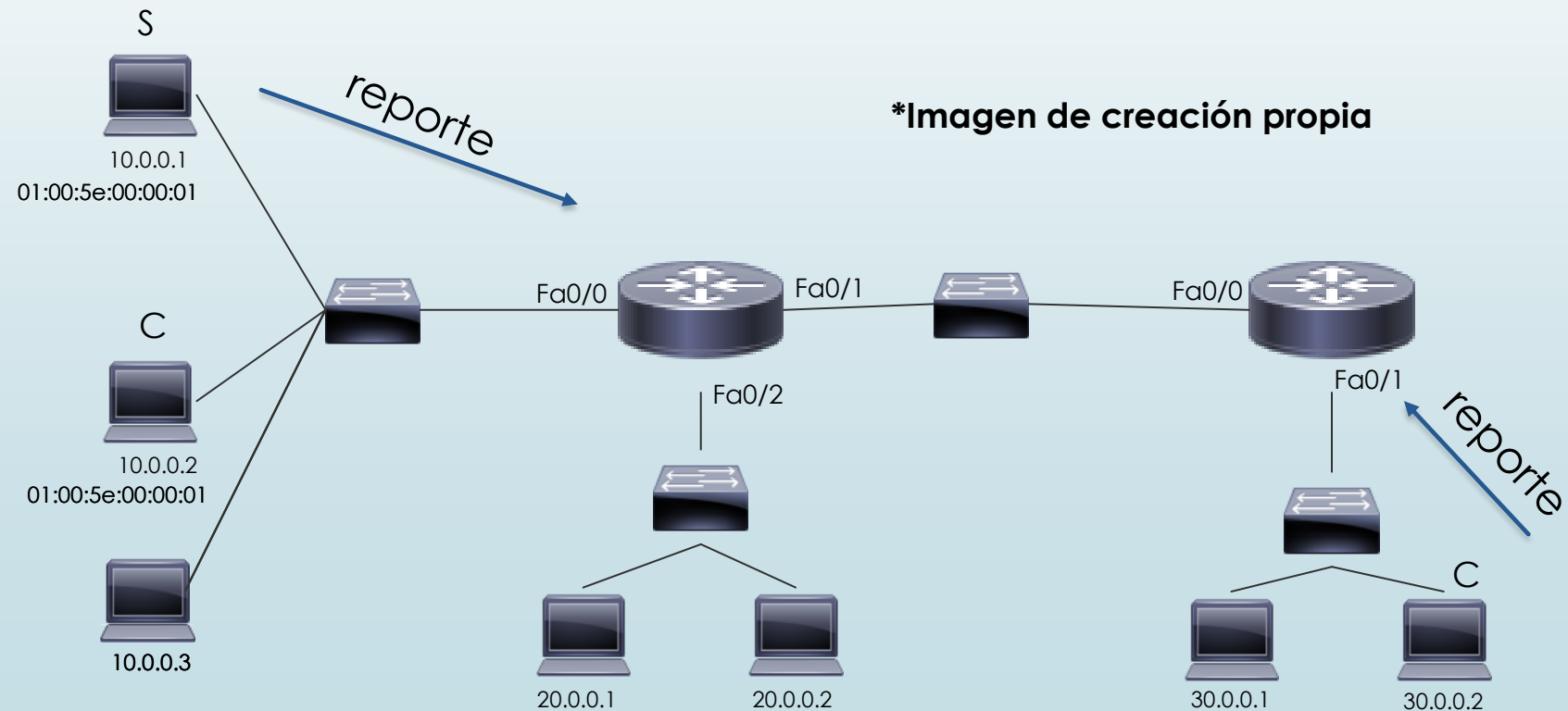
\*Distintas direcciones IP multicast pueden producir la misma dirección MAC multicast

Ej. (230.129.10.10) →(01:00:5e:01:0A:0A)

(225.1.10.10) →(01:00:5e:01:0A:0A)

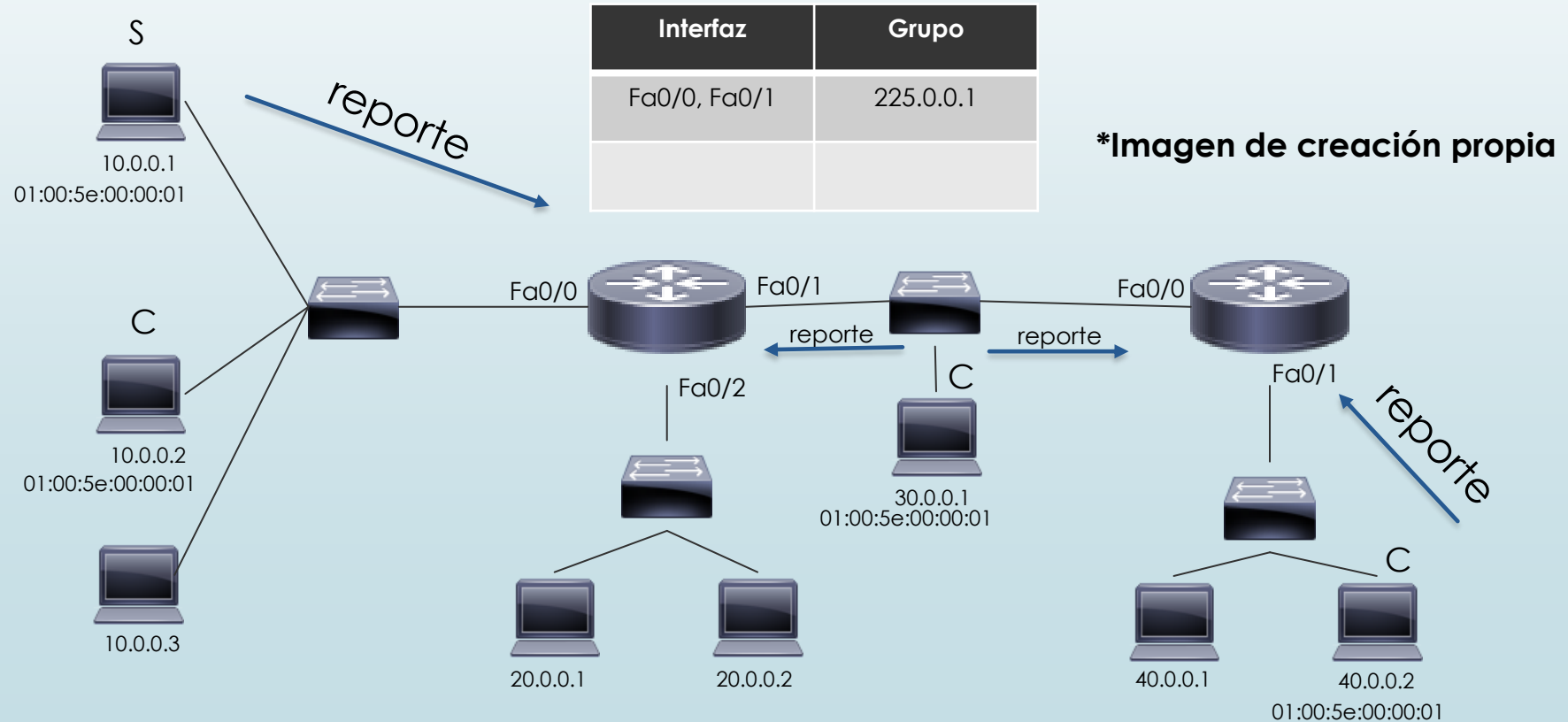
# Funcionamiento IGMP

- El host transmite un mensaje de reporte a los enrutadores IGMP cercanos avisando que escuchará la dirección de grupo



# Funcionamiento IGMP

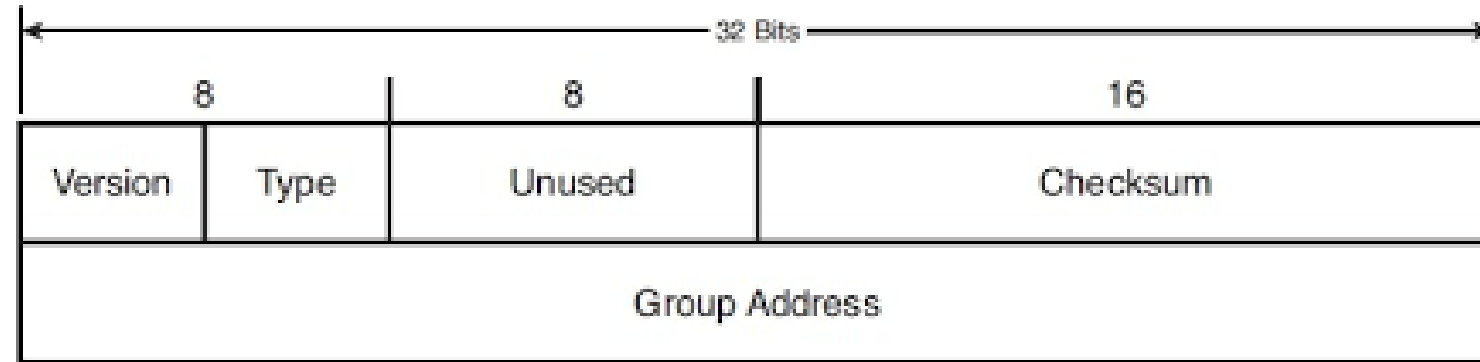
- El enrutador recibe el reporte y actualiza su tabla IGMP agregando ya sea una nueva entrada con la dirección de grupo y la interfaz por donde se recibió el reporte, o solo añadiendo la nueva interfaz por donde hay que transmitir copias de dicho grupo



# Formato de mensaje IGMP

\*Fuente de la imagen: <https://www.researchgate.net/profile/Loye-Sebastien-2/publication/233341383/figure/fig3/AS:11431281185954420@1693810885236/IGMPv1-packet-format.png>

## ► IGMPv1



## IGMP message type values

Message	Type value
Membership Query	0x11
IGMPv1 Membership Report	0x12
IGMPv2 Membership Report	0x16
IGMPv3 Membership Report	0x22
Leave Group	0x17

Tipo=  $\begin{cases} 1 & \text{enviado por el enrutador} \\ 2 & \text{enviado por el host} \end{cases}$

\*Fuente de la imagen: <https://i.imgur.com/3BQlxhe.png>

# Formato de mensaje IGMP

\*Fuente de la imagen: <https://i.imgur.com/yODPMs5.png>

## ➡ IGMPv2

0-7	8-15	16-31
Type	Max Resp Time	Checksum
Group Address		

**IGMP message type values**

Message	Type value
Membership Query	0x11
IGMPv1 Membership Report	0x12
IGMPv2 Membership Report	0x16
IGMPv3 Membership Report	0x22
Leave Group	0x17

\*Fuente de la imagen: <https://i.imgur.com/3BQlxhe.png>

# Formato de mensaje IGMP

➡ IGMPv3

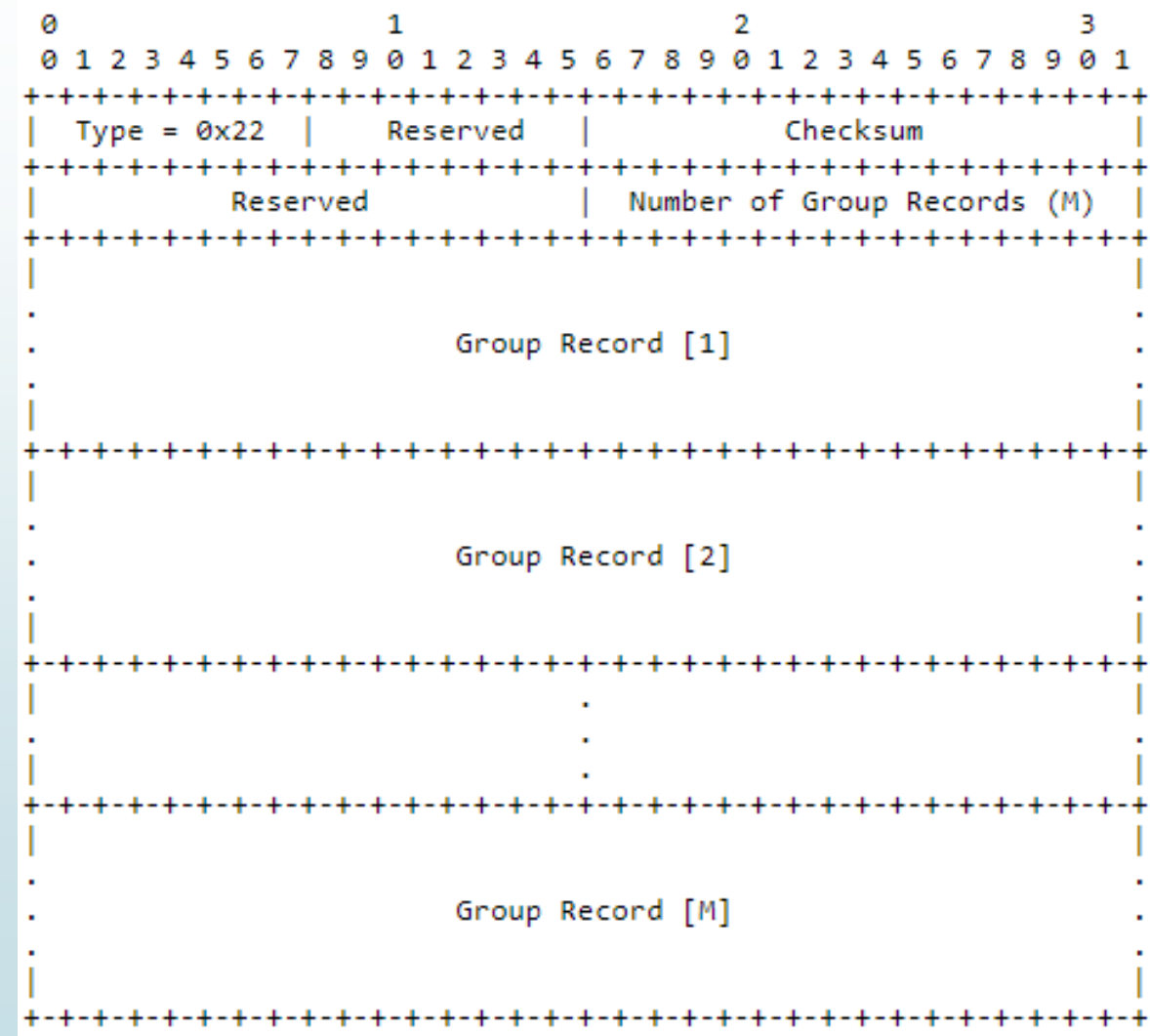
\*Fuente de la imagen:

[https://i-blog.csdnimg.cn/blog\\_migrate/9ea536cfeed0055237a8fefecf86b67b.png](https://i-blog.csdnimg.cn/blog_migrate/9ea536cfeed0055237a8fefecf86b67b.png)

IGMP message type values

Message	Type value
Membership Query	0x11
IGMPv1 Membership Report	0x12
IGMPv2 Membership Report	0x16
IGMPv3 Membership Report	0x22
Leave Group	0x17

\*Fuente de la imagen: <https://i.imgur.com/3BQlxe.png>



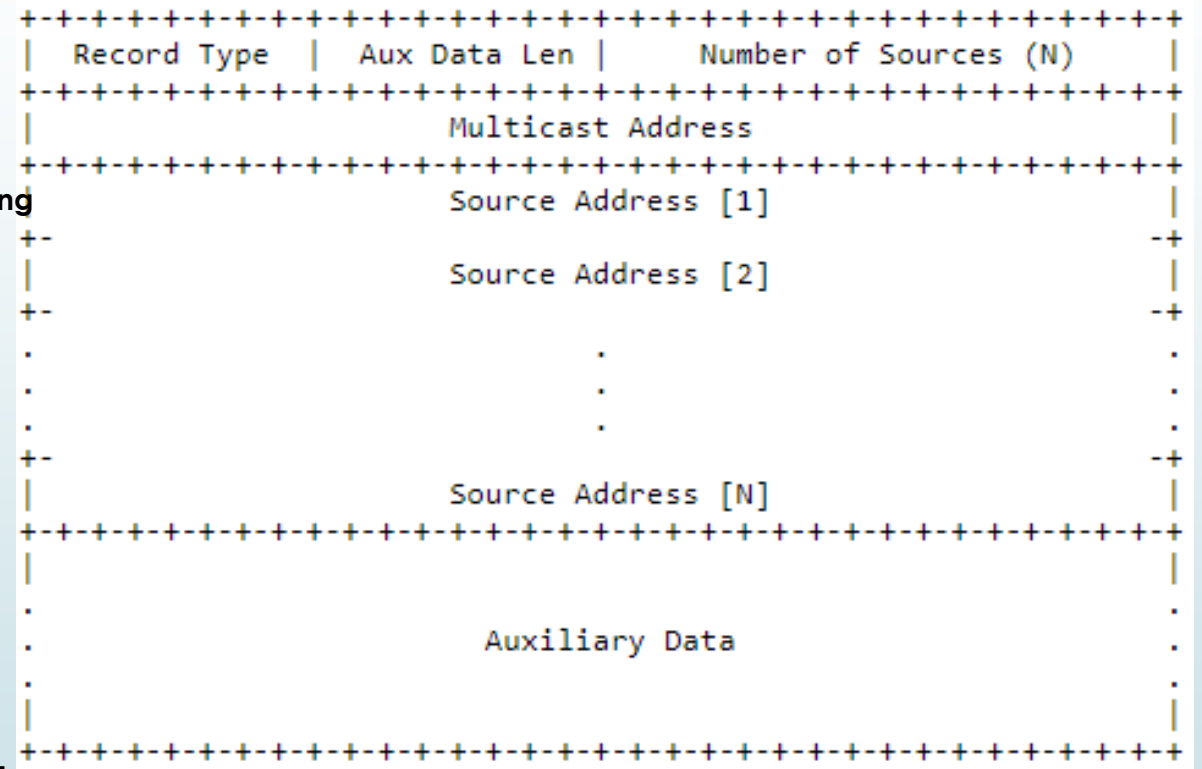


# Formato de mensaje IGMP

## ➡ IGMPv3

\*Fuente de la imagen:

[https://i-blog.csdnimg.cn/blog\\_migrate/6d4aab6173bdcfce9f3e61325e7c9089.png](https://i-blog.csdnimg.cn/blog_migrate/6d4aab6173bdcfce9f3e61325e7c9089.png)



\*Fuente: <https://blog.csdn.net/chen1415886044/article/details/112383815>

TipoRegistro=

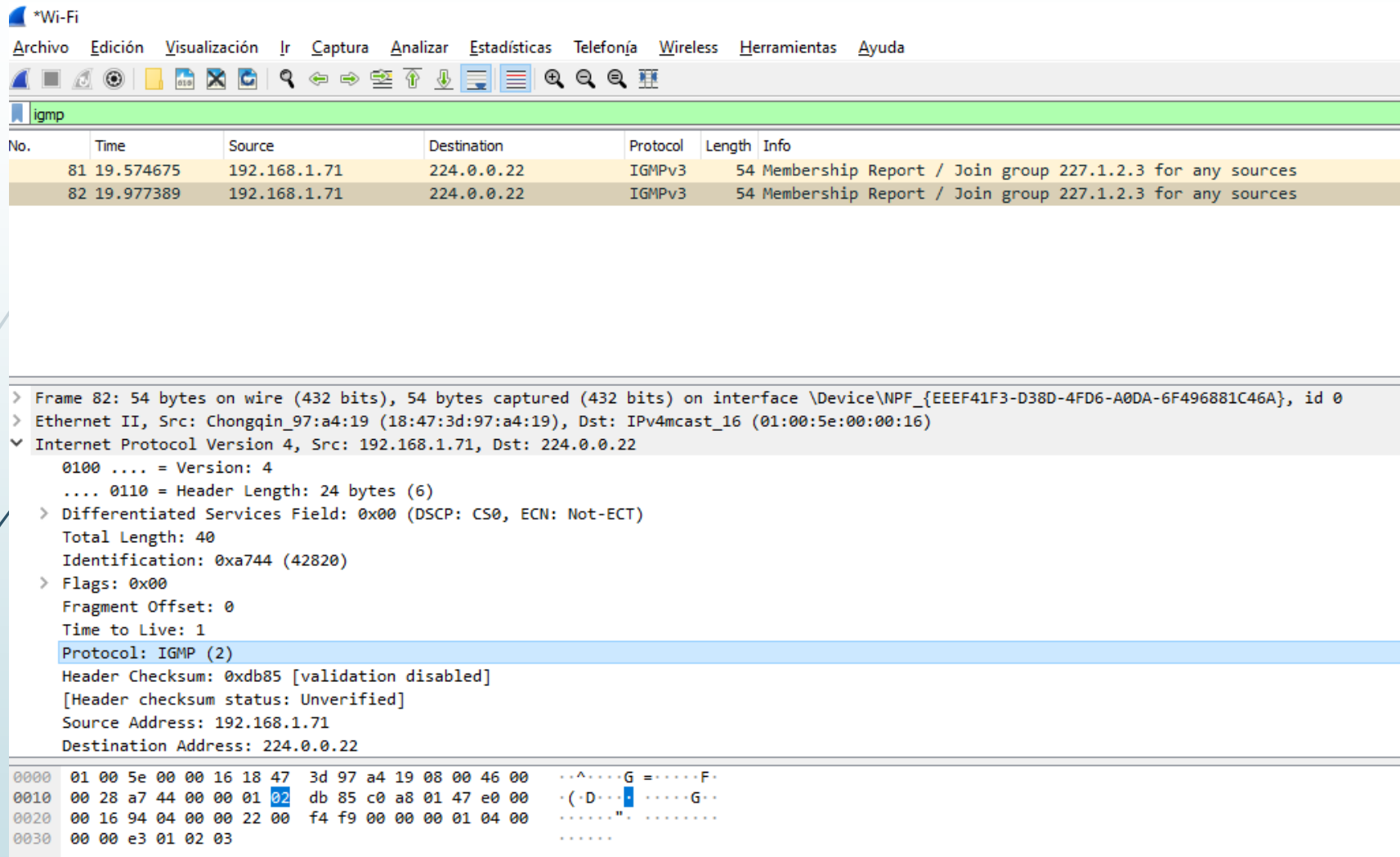
**Como respuesta a una consulta recibida en una interfaz**

- 1: estado actual interfaz=MODE\_IS\_INCLUDE para la dir multicast especificada y los campos de dirección origen
- 2: estado actual interfaz=MODE\_IS\_EXCLUDE para la dir multicast especificada y los campos de dirección origen

**Cuando hay una invocación de cambio de modo en la interfaz de red**

- 3: Change\_TO\_INCLUDE\_MODE la interfaz cambia a modo INCLUDE para la dir multicast y las direcciones origen
- 4: Change\_TO\_EXCLUDE\_MODE la interfaz cambia a modo EXCLUDE para la dir multicast y las direcciones origen

# Ej Wireshark



The screenshot displays the Wireshark network protocol analyzer interface. The title bar indicates the capture is on a Wi-Fi interface. The menu bar includes options like Archivo, Edición, Visualización, Ir, Captura, Analizar, Estadísticas, Telefonía, Wireless, Herramientas, and Ayuda. The toolbar contains various icons for file operations, capture control, and analysis. The packet list pane shows two captured packets, both IGMPv3 Membership Reports, with packet 82 selected. The packet details pane for packet 82 shows the following structure:

- Frame 82: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF\_{EEE41F3-D38D-4FD6-A0DA-6F496881C46A}, id 0
- Ethernet II, Src: Chongqin\_97:a4:19 (18:47:3d:97:a4:19), Dst: IPv4mcast\_16 (01:00:5e:00:00:16)
- Internet Protocol Version 4, Src: 192.168.1.71, Dst: 224.0.0.22
  - 0100 .... = Version: 4
  - .... 0110 = Header Length: 24 bytes (6)
  - Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  - Total Length: 40
  - Identification: 0xa744 (42820)
  - Flags: 0x00
  - Fragment Offset: 0
  - Time to Live: 1
  - Protocol: IGMP (2)
  - Header Checksum: 0xdb85 [validation disabled]
  - [Header checksum status: Unverified]
  - Source Address: 192.168.1.71
  - Destination Address: 224.0.0.22

The packet bytes pane at the bottom shows the raw data in hexadecimal and ASCII. The first few bytes are 01 00 5e 00 00 16 18 47 3d 97 a4 19 08 00 46 00, which correspond to the Ethernet II header and the destination MAC address for IPv4 multicast.

\* Imagen generada a partir de captura de pantalla de aplicación wireshark en ejecución

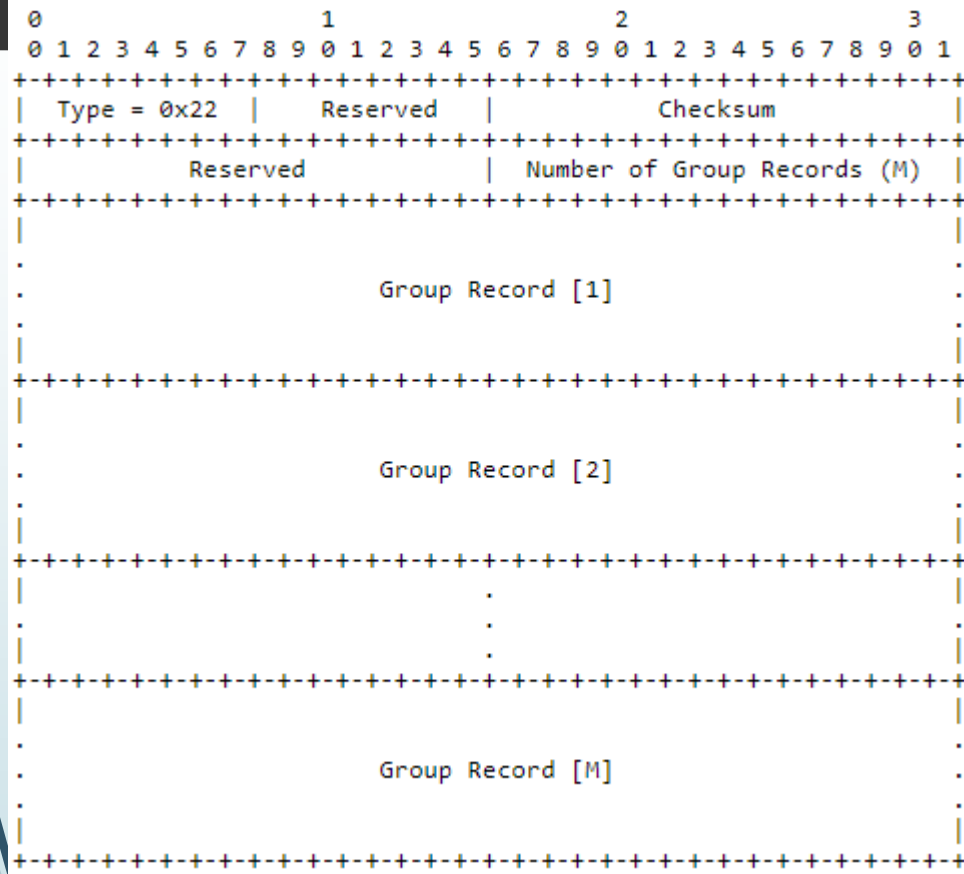
# Ej Wireshark

```
Time to Live: 1
Protocol: IGMP (2)
Header Checksum: 0xdb85 [validation disabled]
[Header checksum status: Unverified]
Source Address: 192.168.1.71
Destination Address: 224.0.0.22
> Options: (4 bytes), Router Alert
Internet Group Management Protocol
[IGMP Version: 3]
Type: Membership Report (0x22)
Reserved: 00
Checksum: 0xf4f9 [correct]
[Checksum Status: Good]
Reserved: 0000
Num Group Records: 1
> Group Record : 227.1.2.3 Change To Exclude Mode
```

000	01 00 5e 00 00 16 18 47 3d 97 a4 19 08 00 46 00	..^....G = .....F.
010	00 28 a7 44 00 00 01 02 db 85 c0 a8 01 47 e0 00	.(.D....G..
020	00 16 94 04 00 00 22 00 f4 f9 00 00 00 01 04 00	.....". .....
030	00 00 e3 01 02 03	.....

\* Imagen generada a partir de captura de pantalla de aplicación wireshark en ejecución

# Ej Wireshark



Time to Live: 1  
Protocol: IGMP (2)  
Header Checksum: 0xdb85 [validation disabled]  
[Header checksum status: Unverified]  
Source Address: 192.168.1.71  
Destination Address: 224.0.0.22  
> Options: (4 bytes), Router Alert  
▼ Internet Group Management Protocol  
[IGMP Version: 3]  
Type: Membership Report (0x22)  
Reserved: 00  
Checksum: 0xf4f9 [correct]  
[Checksum Status: Good]  
Reserved: 0000  
Num Group Records: 1  
▼ Group Record : 227.1.2.3 Change To Exclude Mode  
Record Type: Change To Exclude Mode (4)  
Aux Data Len: 0  
Num Src: 0  
Multicast Address: 227.1.2.3

0000	01 00 5e 00 00 16 18 47 3d 97 a4 19 08 00 46 00	..^....G = ....F.
0010	00 28 a7 44 00 00 01 02 db 85 c0 a8 01 47 e0 00	.(.D....G..
0020	00 16 94 04 00 00 22 00 f4 f9 00 00 00 01 04 00	.....". .....
0030	00 00 e3 01 02 03	.....

\*Fuente de la imagen:  
[https://i-blog.csdnimg.cn/blog\\_migrate/9ea536cfeed0055237a8fefecf86b67b.png](https://i-blog.csdnimg.cn/blog_migrate/9ea536cfeed0055237a8fefecf86b67b.png)

\* Imagen generada a partir de captura de pantalla de aplicación wireshark en ejecución

# Ej Wireshark

\* Imagen generada a partir de captura de pantalla de aplicación wireshark en ejecución

0	7	8	15	16	31
+-----+-----+-----+-----+-----+-----+					
Record Type		Aux Data Len		Number of Sources (N)	
+-----+-----+-----+-----+-----+-----+					
Multicast Address					
+-----+-----+-----+-----+-----+-----+					
Source Address [1]					
+-----+-----+-----+-----+-----+-----+					
Source Address [2]					
+-----+-----+-----+-----+-----+-----+					
.					
.					
.					
+-----+-----+-----+-----+-----+-----+					
Source Address [N]					
+-----+-----+-----+-----+-----+-----+					
.					
.					
.					
+-----+-----+-----+-----+-----+-----+					
Auxiliary Data					
+-----+-----+-----+-----+-----+-----+					

\*Fuente de la imagen:

[https://i-blog.csdnimg.cn/blog\\_migrate/6d4aab6173bdcfce9f3e61325e7c9089.png](https://i-blog.csdnimg.cn/blog_migrate/6d4aab6173bdcfce9f3e61325e7c9089.png)

**Como respuesta a una consulta recibida en una interfaz**

1:estado actual interfaz=MODE\_IS\_INCLUDE

2:estado actual interfaz=MODE\_IS\_EXCLUDE

**Cuando hay una invocación de cambio de modo en la interfaz de red**

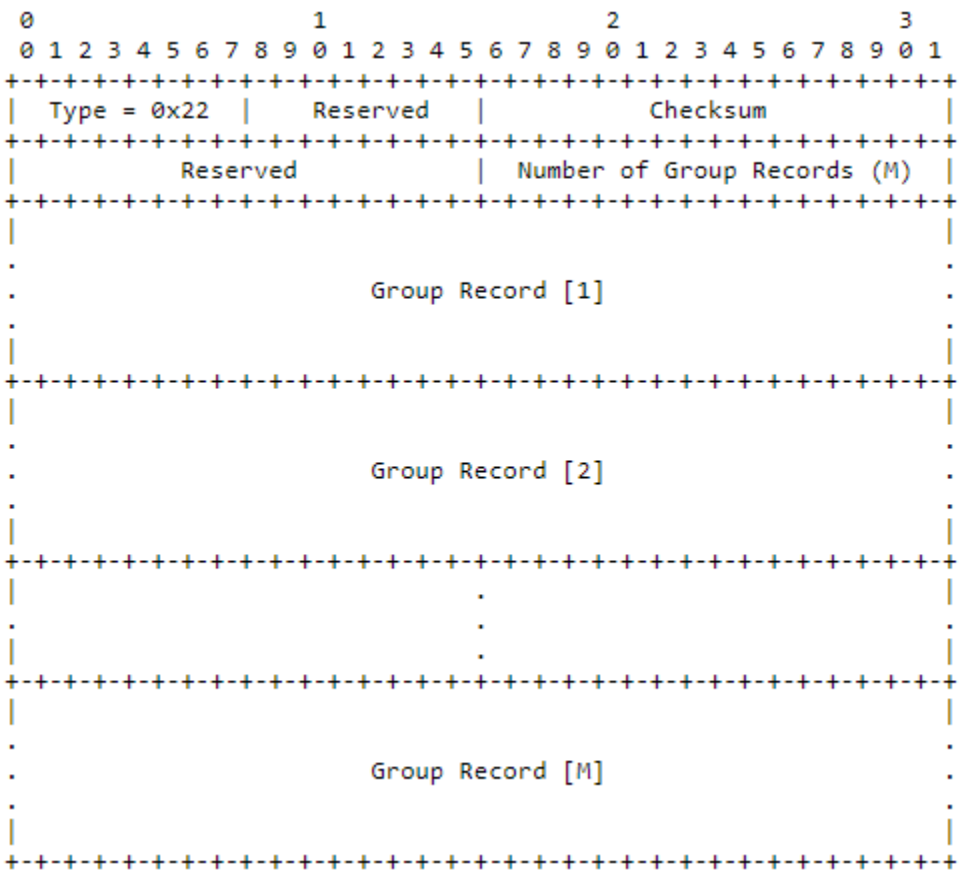
3: Change\_TO\_INCLUDE\_MODE interfaz cambia a modo INCLUDE

4: Change\_TO\_EXCLUDE\_MODE la interfaz cambia a modo EXCLUDE

```
Time to Live: 1
Protocol: IGMP (2)
Header Checksum: 0xdb85 [validation disabled]
[Header checksum status: Unverified]
Source Address: 192.168.1.71
Destination Address: 224.0.0.22
> Options: (4 bytes), Router Alert
Internet Group Management Protocol
  [IGMP Version: 3]
  Type: Membership Report (0x22)
  Reserved: 00
  Checksum: 0xf4f9 [correct]
  [Checksum Status: Good]
  Reserved: 0000
  Num Group Records: 1
  Group Record : 227.1.2.3 Change To Exclude Mode
    Record Type: Change To Exclude Mode (4)
    Aux Data Len: 0
    Num Src: 0
    Multicast Address: 227.1.2.3
```

0000	01 00 5e 00 00 16 18 47 3d 97 a4 19 08 00 46 00	..^....G = ....F.
0010	00 28 a7 44 00 00 01 02 db 85 c0 a8 01 47 e0 00	.(.D....G..
0020	00 16 94 04 00 00 22 00 f4 f9 00 00 00 01 04 00	.....". ....
0030	00 00 e3 01 02 03	.....

# Ejercicio Npcap



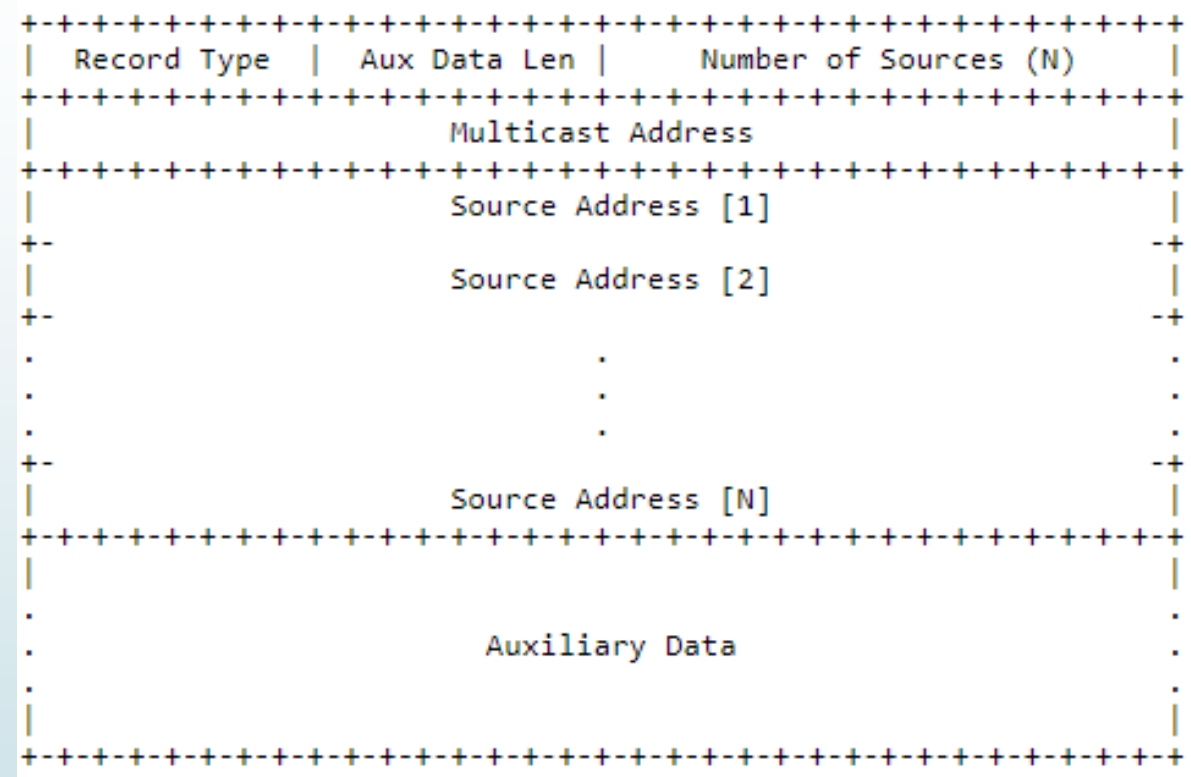
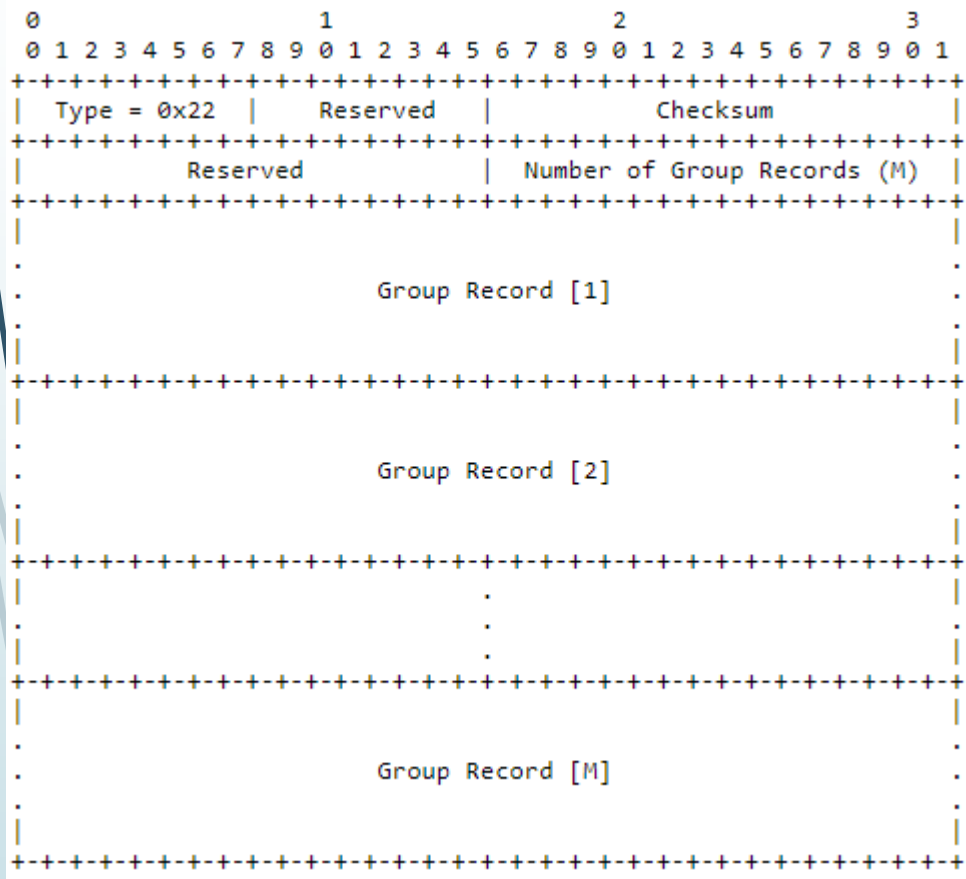
```
/* IGMPv3 header*/
typedef struct igmp_enc{
    u_char type;           // IGMP type
    u_char rsv1;           // reserved
    u_short crc;           // Checksum
    u_short rsv2;          // reserved
    u_short ngr;           // #group records (al menos 1)
}igmp_header;
```

\*Fuente: <https://npcap.com/guide/npcap-tutorial.html>

\*Fuente de la imagen:

[https://i-blog.csdnimg.cn/blog\\_migrate/9ea536cfeed0055237a8fefecf86b67b.png](https://i-blog.csdnimg.cn/blog_migrate/9ea536cfeed0055237a8fefecf86b67b.png)

# Sugerencia para analizar IGMPv3 en Npcap



\*Obtener el # de fuentes (2 bytes)

\*Revisar byte  $(14 + (ihl * 4) + 6)$  y obtener el # de registros de grupo (2 bytes)

\*Fuente de la imagen:

[https://i-blog.csdnimg.cn/blog\\_migrate/6d4aab6173bdcfce9f3e61325e7c9089.png](https://i-blog.csdnimg.cn/blog_migrate/6d4aab6173bdcfce9f3e61325e7c9089.png)

\*Fuente de la imagen:

[https://i-blog.csdnimg.cn/blog\\_migrate/9ea536cfeed0055237a8fefecf86b67b.png](https://i-blog.csdnimg.cn/blog_migrate/9ea536cfeed0055237a8fefecf86b67b.png)