INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

Computer Networks

"Frame Analyzer"

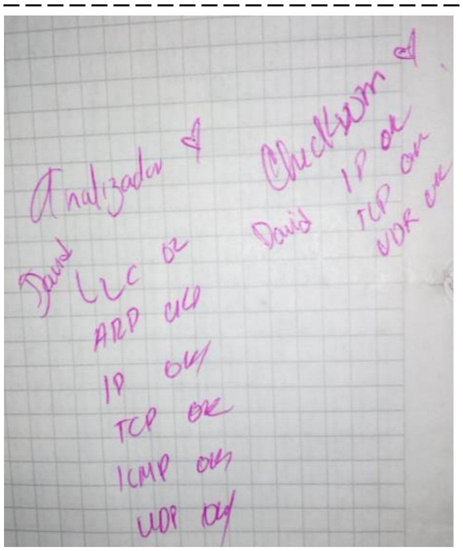
A precise documentation of what we learnt from the Computer Networks course's last unit, up to the implementation of how to analyze LLC, ARP and IP frames.

By:

Portilla Martinez Jose David

Professor:
MSc. NIDIA ASUNCIÓN CORTEZ DUARTE

December 2018



Index

Introduction:

This document provides information about the making of this practice. At its core, this report will only contain a brief description of each header and protocol, the code that I implement according to the professor's specifications and help, and tests of how the program works.

LLC

Acronym and Description

Logical Link Control

The LLC protocol is indirectly derived from the HDLC protocol, from which inherits its control field and standardized by the IEEE 802.2, which defines the same protocol as a data link control layer used on 802.3, 802.5, and other networks. IBM originally designed LLC as a sublayer in the IBM Token Ring architecture.

Header

LLC Header		
Destination SAP	**Source SAP**	**Control Field**
8 bits	8 bits	8/16 bits

Fields

- **Source SAP:** is an 8-bit long field that represents the logical address of the network layer entity that has created the message.
- **Destination SAP:** is an 8-bit long field that represents the logical addresses of the network layer entity intended to receive the message.
- **Control Field:** Following the destination and source SAP fields is a control field. IEEE 802.2 was conceptually derived from HDLC, and has the same three types of PDUs:

  - ✓ Unnumbered format PDUs, or U-format PDUs, with an 8-bit control field, which are intended for connectionless applications;
  - ✓ Information transfer format PDUs, or I-format PDUs, with a 16-bit control and sequence numbering field, which are intended to be used in connection-oriented applications;
  - ✓ Supervisory format PDUs, or S-format PDUs, with a 16-bit control field, which are intended to be used for supervisory functions at the LLC layer.
  - ✓ To carry data in the most-often used unacknowledged connectionless mode the U-format is used. It is identified by the value '11' in lower two bits of the single-byte control field.

Code

```
if (ToT <= 1500)
{
    printf("\n\t|\tTamaño: %4d bytes\t\t\t|", ToT);
    printf("\n\t==================================================\n");
    printf("\n\t\t\t  Protocolo LLC\n");
    printf("\t==================================================\n");
    printf("\t|\tSAP Destino: %.2X\t\t\t\t|\n", frame[14]);
    printf("\t|\tSAP Origen:  %.2X\t\t\t\t|\n", frame[15]);
    printf("\t|\t-----> Campo de Control <-----\t\t|\n");
    switch (frame[16] & 3)
    {
        case 0: case 2: //T-I
            printf("\t|\tT-I, ");
            if (E) //Extendido
                  printf("N(s) = %.3d, N(r) = %.3d\t\t|\n", (frame[16] >> 1), (frame[17] >> 1));
            else
                  printf("N(s) = %3d, N(r) = %d\t\t|\n", (frame[16] >> 1) & 7, frame[16] >> 5);
                 break;
        case 1: //T-S
            printf("\t|\tT-S, ");
            if (E) // Extendido
                printf("%4s, N(r) = %d\t\t\t|\n", SS[(frame[16]>>2) & 3], frame[17] >> 1);
            else
                printf("%4s, N(r) = %d\t\t\t|\n", SS[(frame[16]>>2) & 3], frame[17] >> 5);
                  break;
         case 3: //T-U
             if (frame[16] & 16)
             {
                  M = (((frame[16] >> 2) & 3) | ((frame[16] >> 3) & 28));
                  printf("\t|\t\tT-U, ");
                  if (frame[15] & 1) //Respuesta
                      printf("%5s, f\t\t\t|\n", UR[M]);
                  else //Comando
                  {
                      printf("%5s, p\t\t\t|\n", UC[M]);
                      E = (M ^ 11) ? ((M ^ 15) ? ((M ^ 27) ? 0 : 1) : 1) : 1;
                  }
              }
                  break;
          default:
                  break;
     }
```

Tests

I select 2 random frames for testing.

First Frame (No. 1 in frames given by the professor):

```
0x00 0x02 0xb3 0x9c 0xae 0xba 0x00 0x02 0xb3 0x9c 0xdf 0x1b 0x00 0x03 0xf0 0xf0
0x7f 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x43 0x05 0x90 0x6d
```

At execution, the program shows as follows (Figure 1):

```
              _-_-_-_-_-_-_Trama 1-_-_-_-_-_-_-

                      Protocolo Ethernet
=================================================
|       MAC Destino: 00:02:B3:9C:AE:BA          |
|       MAC Origen:  00:02:B3:9C:DF:1B          |
|       Tamaño:    3 bytes                      |
=================================================

                      Protocolo LLC
=================================================
|       SAP Destino: F0                         |
|       SAP Origen:  F0                         |
|       -----> Campo de Control <-----          |
|               T-U, SABME, p                   |
|       ------------------------------          |
=================================================
```

Figure 1 Execution at Frame 1

Second frame (No. 15 in frames given by the professor):

```
0x00 0x02 0xb3 0x9c 0xdf 0x1b 0x00 0x02 0xb3 0x9c 0xae 0xba 0x00 0x46 0xf0 0xf0
0x04 0x06 0x0e 0x00 0xff 0xef 0x16 0x0c 0x00 0x00 0x28 0x00 0x28 0x00 0x7f 0x23
```

At execution, the program shows as follows (Figure 2):

```
              _-_-_-_-_-_-_Trama 15-_-_-_-_-_-_-

                      Protocolo Ethernet
=================================================
|       MAC Destino: 00:02:B3:9C:DF:1B          |
|       MAC Origen:  00:02:B3:9C:AE:BA          |
|       Tamaño:   70 bytes                      |
=================================================

                      Protocolo LLC
=================================================
|       SAP Destino: F0                         |
|       SAP Origen:  F0                         |
|       -----> Campo de Control <-----          |
|       T-I, N(s) = 002, N(r) = 003             |
|       ------------------------------          |
=================================================
```

Figure 2 Execution at Frame 15

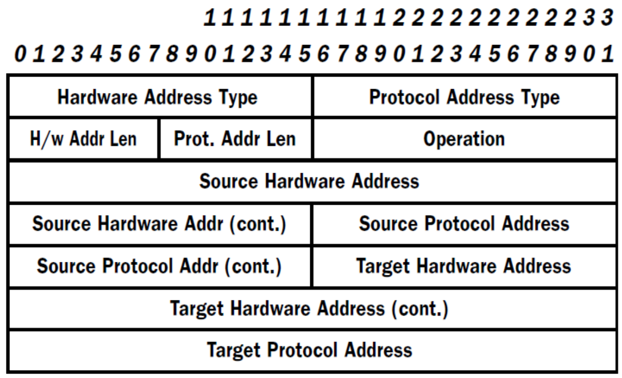
ARP

Acronym and Description

Address Resolution Protocol

ARP is a protocol for mapping an IP address to a physical MAC address that is recognized in the local network. For example, in IPv4, the most common level of IP in use today, an address is 32 bits long. In an Ethernet local area network, however, addresses for attached devices are 48 bits long. A table, usually called the ARP cache, is used to maintain a correlation between each MAC address and its corresponding IP address. ARP provides the protocol rules for making this correlation and providing address conversion in both directions.

Header



Fields

- **Hardware Address Type:** This field specifies the network link protocol type.
- **Protocol Address Type:** This field specifies the internetwork protocol for which the ARP request is intended. For IPv4, this has the value 0x0800. The permitted PTYPE values share a numbering space with those for EtherType.
- **Hardware Address Length:** Length of a hardware address. Ethernet addresses size is 6.
- **Protocol Address Length:** Length of addresses used in the upper layer protocol.
- **Operation:** Specifies the operation that the sender is performing: 1 for request, 2 for reply.
- **Source Hardware Address:** Media address of the sender. In an ARP request this field is used to indicate the address of the host sending the request. In an ARP reply this field is used to indicate the address of the host that the request was looking for.
- **Source Protocol Address:** Internetwork address of the sender.
- **Target Hardware Address:** Media address of the intended receiver. In an ARP request this field is ignored. In an ARP reply this field is used to indicate the address of the host that originated the ARP request.
- **Target Protocol Address:** Internetwork address of the intended receiver.

Code

```
else
{
            printf("\n\t\t\t Protocolo ARP\n");

            printf("\t=================================================\n");

            printf("\t|\tTipo de dir. Hardware: ");

            printf("%s\t\t|\n", (frame[15] ^ 1) ? ((frame[15] ^ 6) ? ((frame[15] ^
15) ? ((frame[15] ^ 16) ? "Otro" : "ATM") : "Frame Relay") : "LAN") : "Ethernet");

            printf("\t|\tTipo de dir. Protocolo: IP\t\t|\n");

            printf("\t|\tTam. dir. Hardware: ");

            printf("%d\t\t\t|\n", frame[18]);

            printf("\t|\tTam. dir. IPv4: ");

            printf("%d\t\t\t|\n", frame[19]);

            printf("\t|\tCódigo de Operación: ");

            printf("%s\t\t|\n", OP[frame[21]]);

            printf("\t|\tMAC Origen: "); printMAC(frame, 22);

            printf("\n\t|\tIP Origen: "); printIP(frame, 28);

            printf("\t|\tMAC Destino: "); printMAC(frame, 32);

            printf("\n\t|\tIP Destino: "); printIP(frame, 38);

}
```

Tests

Again, I choose 2 frames. An important field in this protocol is the operation code, which specifies the operation that the sender is performing, either request or reply. I selected one of each, so we can see the differences.

The first frame for ARP, in this case request:

```
0xff 0xff 0xff 0xff 0xff 0xff 0x00 0x23 0x8b 0x46 0xe9 0xad 0x08 0x06 0x00 0x04
0x08 0x00 0x06 0x04 0x00 0x01 0x00 0x23 0x8b 0x46 0xe9 0xad 0x94 0xcc 0x39 0xcb
0x00 0x00 0x00 0x00 0x00 0x00 0x94 0xcc 0x39 0xfe
```

At execution, the program displayed (Figure 3):

```
           _-_-_-_-_-_-_Trama 34-_-_-_-_-_-_-

                  Protocolo Ethernet
=================================================
|       MAC Destino: FF:FF:FF:FF:FF:FF          |
|       MAC Origen:  00:23:8B:46:E9:AD          |
|       Tipo: ARP                               |
=================================================

                  Protocolo ARP
=================================================
|       Tipo de dir. Hardware: Otro             |
|       Tipo de dir. Protocolo: IP              |
|       Tam. dir. Hardware: 6                   |
|       Tam. dir. IPv4: 4                       |
|       Código de Operación: Solicitud          |
|       MAC Origen: 00:23:8B:46:E9:AD           |
|       IP Origen: 148.204.057.203              |
|       MAC Destino: 00:00:00:00:00:00          |
|       IP Destino: 148.204.057.254             |
=================================================
```

Figure 3 Execution at Frame 34

The second frame for ARP, in this case reply:

```
0x00 0x23 0x8b 0x46 0xe9 0xad 0x00 0x1f 0x45 0x9d 0x1e 0xa2 0x08 0x06 0x00 0x01
0x08 0x00 0x06 0x04 0x00 0x02 0x00 0x1f 0x45 0x9d 0x1e 0xa2 0x94 0xcc 0x39 0xfe
0x00 0x23 0x8b 0x46 0xe9 0xad 0x94 0xcc 0x39 0xcb 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

At execution, the program displayed (Figure 4):

```
           _-_-_-_-_-_-_Trama 35-_-_-_-_-_-_-

                  Protocolo Ethernet
=================================================
|       MAC Destino: 00:23:8B:46:E9:AD          |
|       MAC Origen:  00:1F:45:9D:1E:A2          |
|       Tipo: ARP                               |
=================================================

                  Protocolo ARP
=================================================
|       Tipo de dir. Hardware: Ethernet         |
|       Tipo de dir. Protocolo: IP              |
|       Tam. dir. Hardware: 6                   |
|       Tam. dir. IPv4: 4                       |
|       Código de Operación: Respuesta          |
|       MAC Origen: 00:1F:45:9D:1E:A2           |
|       IP Origen: 148.204.057.254              |
|       MAC Destino: 00:23:8B:46:E9:AD          |
|       IP Destino: 148.204.057.203             |
=================================================
```

Figure 4 Execution at Frame 35

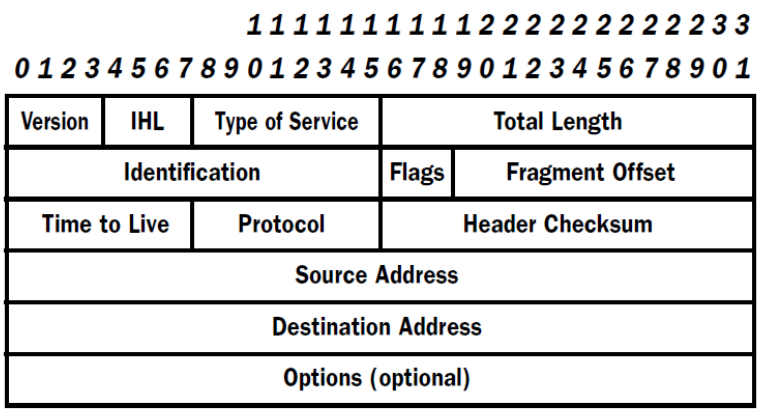
IP

Acronym and Description

Internet Protocol

IP is a connectionless protocol for use on packet-switched networks. It operates on a best effort delivery model, in that it does not guarantee delivery, nor does it assure proper sequencing or avoidance of duplicate delivery. These aspects, including data integrity, are addressed by an upper layer transport protocol, such as the Transmission Control Protocol (TCP).

Header



Fields

- **Version:** The first header field in an IP packet is the four-bit version field. For IPv4, this is always equal to 4.
- **Internet Header Length:** This field has 4 bits, which is the number of 32-bit words. Since an IPv4 header may contain a variable number of options, this field specifies the size of the header. The minimum value for this field is 5, which indicates a length of 5 × 32 bits = 160 bits = 20 bytes. As a 4-bit field, the maximum value is 15 words (15 × 32 bits, or 480 bits = 60 bytes).
- **Type of Service:** This field specifies a datagram's priority and request a route for low-delay, high-throughput, or highly-reliable service. Based on these ToS values, a packet would be placed in a prioritized outgoing queue, or take a route with appropriate latency, throughput, or reliability.
- **Total Length:** This 16-bit field defines the entire packet size in bytes, including header and data. The minimum size is 20 bytes (header without data) and the maximum is 65,535 bytes.
- **Identification:** This field is an identification field and is primarily used for uniquely identifying the group of fragments of a single IP datagram.

- **Flags:** A three-bit field follows and is used to control or identify fragments.
  - ✓ bit 0: Reserved. Must be zero.
  - ✓ bit 1: Don't Fragment
  - ✓ bit 2: More Fragments
- **Fragment Offset:** This field is measured in units of eight-byte blocks. It is 13 bits long and specifies the offset of a fragment relative to the beginning of the original unfragmented IP datagram.
- **Time to Live:** This field limits a datagram's lifetime. It is specified in seconds, but time intervals less than 1 second are rounded up to 1. In practice, the field has become a hop count, when the datagram arrives at a router, the router decrements the TTL field by one.
- **Protocol:** This field defines the protocol used in the data portion of the IP datagram.
- **Header Checksum:** The 16-bit IP header checksum field is used for error-checking of the header.
- **Source Address:** This field is the IP address of the sender of the packet. Note that this address may be changed in transit by a network address translation device.
- **Destination Address:** This field is the IP address of the receiver of the packet. As with the source address, this may be changed in transit by a network address translation device.
- **Options:** The options field is not often used. Note that the value in the IHL field must include enough extra 32-bit words to hold all the options.

Code

```c
printf("\n\t\t\t Protocolo IP\n");

printf("\t==================================================\n");
printf("\t|\tVersion: IPv%d\t\t\t\t|\n", frame[14] >> 4);
IHL = (frame[14] & 15) << 2;
if ((IHL < 20))
{
    printf("\t-------->Parámetros Minimos No Correctos<--------\n");
    return;
}
printf("\t|\tIHL: %d Bytes\t\t\t\t|\n", IHL);
if ((frame[15] >> 1) & 15)
{
    printf("\t|\tTipo de Servicio: \t\t\t|\n");
    if (frame[15] & 16)
        printf("\t|\t\t->Rendimiento Minimo\t\t|\n");
    if (frame[15] & 8)
        printf("\t|\t\t->Retraso Maximo\t\t|\n");
    if (frame[15] & 4)
        printf("\t|\t\t->Cofiabilidad Maxima\t\t|\n");
    if (frame[15] & 2)
        printf("\t|\t\t->Costo Minimo\t\t\t|\n");
}
printf("\t|\tLongitud Total: %d Bytes\t\t|\n", (frame[16] << 8) | frame[17]);
printf("\t|\tIdentificación: %d Bytes\t\t|\n", (frame[18] << 8) | frame[19]);
```

```c
if ((frame[20] >> 5) & 3)
{
    printf("\t|\tBanderas: \t\t\t\t|\n");
    if (frame[20] & 64)
        printf("\t|\t\t->No Fragmentar\t\t\t|\n");
    if (frame[20] & 32)
        printf("\t|\t\t->Fragmentar Más\t\t|\n");
}

printf("\t|\tTiempo de Vida: %d Saltos\t\t|\n", frame[22]);
printf("\t|\tProtocolo: ");
printf("%s\t\t\t\t|\n", (frame[23] ^ 17) ? ((frame[23] ^ 6) ? ((frame[23] ^ 1) ?
"Otro" : "ICMP") : "TCP") : "UDP");
printf("\t|\tChecksum: %s\t\t|\n", checksum(frame, 14, 14 + IHL - 1, 0));
printf("\t|\tIP Origen: "); printIP(frame, 26);
printf("\t|\tIP Destino: "); printIP(frame, 30);

if (IHL > 20)
{
    printf("\t|\tOpciones:\t\t\t\t|\n"); //0, 1, 7, 68, 131, 137

    for (i = 0; i < IHL - 20; i++)
    {
        printf(" ->Flag: %s", (frame[34 + i] >> 7) ? "Copiar\t" : "No Copiar");
        printf("\t|\t ->Clase: %s\n", class[(frame[34 + i] >> 5) & 3]);
        printf("\t|\t ->Opción: %s\n", (frame[34 + i]) ? (((frame[34 + i]) ^ 1) ?
(((frame[34 + i]) ^ 7) ? (((frame[34 + i]) ^ 131) ? (((frame[34 + i]) ^ 137) ?
(((frame[34 + i]) ^ 68) ? "Otro\t\t\t\t|" : "Timestamp\t\t\t|") : "Ruta de Origen
Estricta\t|") : "Ruta de Origen Ligera\t|") : "Record de Ruta\t\t|") : "No
Operación\t\t\t|") : "Final Lista de Opciones\t|");
        printf("\t|\t\t\t\t\t\t|\n");
    }
}
```

Tests

For this test I decided not to show the entirety of the frame, because when the header is IP, then we have another header, whether is ICMP, TCP or UDP. Which they'll be describe in later sections, but for now we're only interested in the IP header

As for previous tests, I'm going to choose two frames, so we can analyze when we get to the IP header. The first one will be an IP without options, and the next one will one with options.

The first frame for IP, in this case without options:

```
0x02 0xff 0x53 0xc3 0xe9 0xab 0x00 0xff 0x66 0x7f 0xd4 0x3c 0x08 0x00 0x45 0x00
0x00 0x30 0x2c 0x00 0x40 0x00 0x80 0x01 0x4b 0x79 0xc0 0xa8 0x01 0x02 0xc0 0xa8
0x01 0x01 0x05 0x02 0xfa 0xfd
```

At execution, the program displayed (Figure 5):

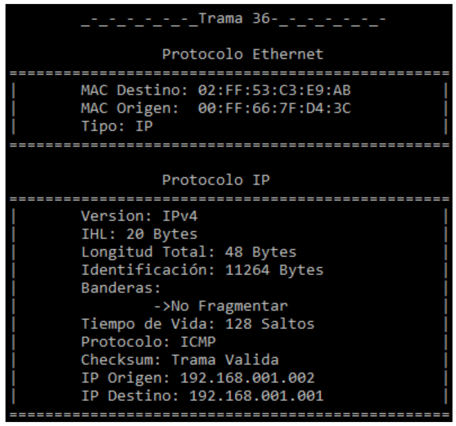


Figure 5 Execution at Frame 36

The second frame for IP, in this case with options:

```
0x00 0x1f 0x45 0x9d 0x1e 0xa2 0x00 0x23 0x8b 0x46 0xe9 0xad 0x08 0x00 0x46 0x00
0x80 0x42 0x04 0x55 0x34 0x11 0x80 0x11 0x2c 0x11 0x94 0xcc 0x39 0xcb 0x94 0xcc
0x67 0x02 0x07 0x44 0x83 0x89 0x04 0x0c 0x00 0x35 0x00 0x2e 0x85 0x7c 0xe2 0x1a
0x01 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x03 0x77 0x77 0x77 0x03 0x69
0x73 0x63 0x05 0x65 0x73 0x63 0x6f 0x6d 0x03 0x69 0x70 0x6e 0x02 0x6d 0x78 0x00
0x00 0x1c 0x00 0x01
```

At execution, the program displayed (Figure 6):

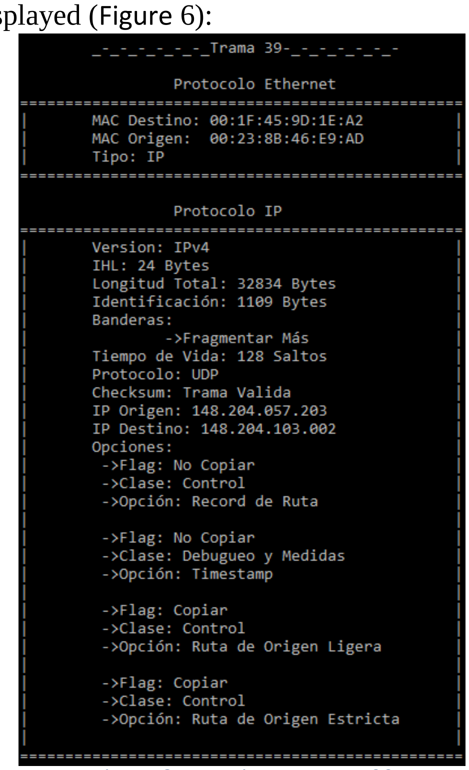


Figure 6 Execution at Frame 39

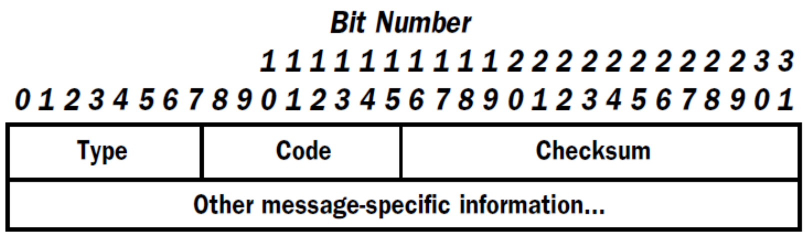
ICMP

Acronym and Description

Internet Control Message Protocol

Is a supporting protocol in the Internet protocol suite. It is used by network devices, including routers, to send error messages and operational information indicating, for example, that a requested service is not available or that a host or router could not be reached.

Header



Fields

Type Code

0 Echo Reply
3 Destination Unreachable
 0 Net Unreachable
 1 Host Unreachable
 2 Protocol Unreachable
 3 Port Unreachable
 4 Fragmentation Needed & DF Set
 5 Source Route Failed
 6 Destination Network Unknown
 7 Destination Host Unknown
 8 Source Host Isolated
 9 Network Administratively Prohibited
 10 Host Administratively Prohibited
 11 Network Unreachable for TOS
 12 Host Unreachable for TOS
 13 Communication Administratively Prohibited
4 Source Quench
5 Redirect
 0 Redirect Datagram for the Network
 1 Redirect Datagram for the Host
 2 Redirect Datagram for the TOS & Network
 3 Redirect Datagram for the TOS & Host
8 Echo
9 Router Advertisement
10 Router Selection

11 Time Exceeded
 0 Time to Live exceeded in Transit
 1 Fragment Reassembly Time Exceeded
12 Parameter Problem
 0 Pointer indicates the error
 1 Missing a Required Option
 2 Bad Length
13 Timestamp
14 Timestamp Reply
15 Information Request
16 Information Reply
17 Address Mask Request
18 Address Mask Reply
30 Traceroute

Checksum: The 16-bit IP header checksum field is used for error-checking of the header.

Code

```
printf("\t=================================================\n");
printf("\n\t\t\t Protocolo ICMP\n");
printf("\t=================================================\n");
if (strlen(type[frame[14 + IHL]]) < 18)
     printf("\t|\tTipo: %17s\t\t\t|\n", type[frame[14 + IHL]]);
else
     printf("\t|\tTipo: %s\t|\n", type[frame[14 + IHL]]);
if (frame[14 + IHL] ^ 3)
{
     if (frame[14 + IHL] ^ 5)
     {
         if (frame[14 + IHL] ^ 11)
         {
              if (frame[14 + IHL] ^ 12){}
              else
                  printf("\t|\tCódigo: %s\t|\n", PP[frame[15 + IHL]]);
          }
          else
              printf("\t|\tCódigo: %s\t|\n", TE[frame[15 + IHL]]);
      }
      else
          printf("\t|\tCódigo: %s\t|\n", RED[frame[15 + IHL]]);
}
else
    printf("\t|\tCódigo: %s\t|\n", DI[frame[15 + IHL]]);

printf("\t|\tChecksum: %s\t\t|\n", checksum(frame, 14 + IHL, 17 + IHL, 1));
```

Tests

Here we only made one test, as the ICMP header isn't that complicated to understad.

First and only frame for ICMP (figure 7):

```
0x02 0xff 0x53 0xc3 0xe9 0xab 0x00 0xff 0x66 0x7f 0xd4 0x3c 0x08 0x00 0x45 0x00
0x00 0x30 0x2c 0x00 0x40 0x00 0x80 0x01 0x4b 0x79 0xc0 0xa8 0x01 0x02 0xc0 0xa8
0x01 0x01 0x05 0x02 0xfa 0xfd
```

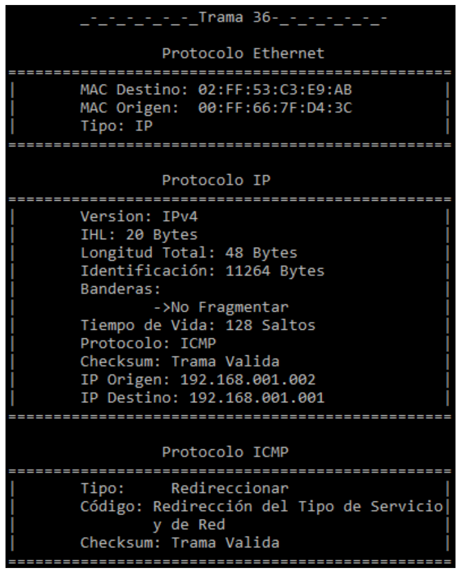


Figure 7 Execution at Frame36

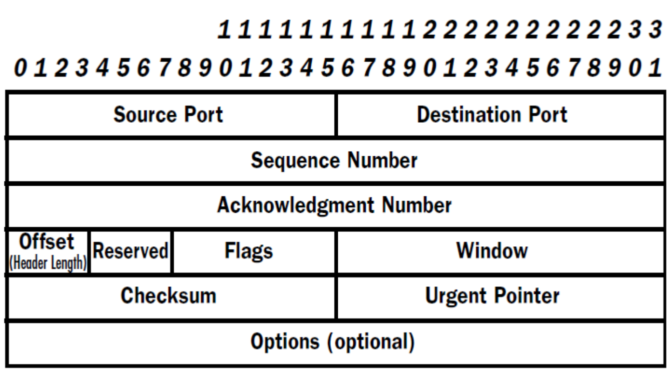
TCP

Acronym and Description

Transmission Control Protocol

The Transmission Control Protocol is one of the main protocols of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of bytes between applications running on hosts communicating via an IP network.

Header



Fields

- **Source Port:** Identifies the sending port.
- **Destination Port:** Identifies the receiving port.
- **Sequence Number:** Has a dual role:
  - ✓ If the S flag is set (1), then this is the initial sequence number. The sequence number of the actual first data byte and the acknowledged number in the corresponding ACK are then this sequence number plus 1.
  - ✓ If the S flag is clear (0), then this is the accumulated sequence number of the first data byte of this segment for the current session.
- **Acknowledgment Number:** If the ACK flag is set then the value of this field is the next sequence number that the sender of the ACK is expecting. This acknowledges receipt of all prior bytes (if any). The first ACK sent by each end acknowledges the other end's initial sequence number itself, but no data**.**
- **Offset:** Specifies the size of the TCP header in 32-bit words.
- **Reserved:** For future use and should be set to zero.
- **Flags:**
  - ✓ U (1 = Consult urgent pointer, notify server application of urgent data)
  - ✓ A (1 = Consult acknowledgement field)
  - ✓ P (1 = Push data)
  - ✓ R (1 = Reset connection)
  - ✓ S (1 = Synchronize sequence numbers)
  - ✓ F (1 = no more data; Finish connection)
- **Window:** The size of the receive window, which specifies the number of window size units
- **Checksum:** The 16-bit checksum field is used for error-checking of the header, the Payload and a Pseudo-Header.
- **Urgent Pointer:** If the U flag is set, then this 16-bit field is an offset from the sequence number indicating the last urgent data byte.
- **Options:** The length of this field is determined by the data offset field.
  - ✓ 0 End of Options list
  - ✓ 1 No operation (pad)
  - ✓ 2 Maximum segment size
  - ✓ 3 Window scale
  - ✓ 4 Selective ACK ok
  - ✓ 8 Timestamp

Code

```
printf("\t==================================================\n");
printf("\n\t\t\t Protocolo TCP\n");
printf("\t==================================================\n");
aux = ((frame[14 + IHL] << 8) | frame[15 + IHL]);
printf("\t|\tPuerto de Origen: %s\t\t|\n", (aux ^ 7) ? (((aux ^ 20)) ? ((aux ^ 21) ?
((aux ^ 22) ? ((aux ^ 23) ? ((aux ^ 25) ? ((aux ^ 80) ? ((aux ^ 110) ? ((aux ^ 179) ?
((aux ^ 443) ? "Otro\t" : "HTTPS\t") : "BGP\t") : "POP3\t") : "HTTP\t") : "SMTP\t") :
"Telnet") : "SSH\t") : "FTP-Control"): "FTP-Data"): "Echo\t");
aux = ((frame[16 + IHL] << 8) | frame[17 + IHL]);
printf("\t|\tPuerto de Destino: %s\t\t|\n", (aux ^ 7) ? (((aux ^ 20)) ? ((aux ^ 21) ?
((aux ^ 22) ? ((aux ^ 23) ? ((aux ^ 25) ? ((aux ^ 80) ? ((aux ^ 110) ? ((aux ^ 179) ?
((aux ^ 443) ? "Otro\t" : "HTTPS\t") : "BGP\t") : "POP3\t") : "HTTP\t") : "SMTP\t") :
"Telnet") : "SSH\t") : "FTP-Control"): "FTP-Data"): "Echo\t");
printf("\t|\tNúmero de Secuencia: %.10u\t\t|\n",  (unsigned int)(frame[18 + IHL] <<
24) |  (unsigned int)(frame[19 + IHL] << 16) | (unsigned int)(frame[20 + IHL] << 8) |
(unsigned int)(frame[21 + IHL]));
printf("\t|\tNúmero de Conocimiento: %.10u\t|\n", (unsigned int)(frame[22 + IHL] <<
24) |  (unsigned int)(frame[23 + IHL] << 16) | (unsigned int)(frame[24 + IHL] << 8) |
(unsigned int)(frame[25 + IHL]));
aux = (frame[26 + IHL] >> 4) << 2;
if ((aux < 20))
{
    printf("\t-------->Parámetros Minimos No Correctos<--------\n");
    return;
}
printf("\t|\tTamaño: %.2d\t\t\t\t|\n", aux);
printf("\t|\tBanderas: \t\t\t\t|\n\t|\t\t");
for (j = 1, i = 0; j <= 128, i < 8; j = j << 1, i++)
{
    if (frame[27 + IHL] & j)
        printf("%s/", flagsTCP[i]);
    else
        printf("-/");
}
printf("\t\t|\n\t|\tVentana: %.10d\t\t\t|", (frame[28 + IHL] << 8) | frame[29 + IHL]);
printf("\n\t|\tChecksum: %s\t\t|", checksum(frame, 14 + IHL,  13 + aux + IHL, 1));
printf("\n\t|\tPuntero: %.10d\t\t\t|\n", (frame[32 + IHL] << 8) | frame[33 + IHL]);
if (aux > 20)
{
    printf("\t|\tOpciones:\t\t\t\t|\n"); //0, 1, 3, 4, 8
    for (i = 0; i < aux - 20; i++)
    {
        printf("\t|\t -> %s\n", optionsTCP[(frame[34 + IHL + i] > 8) ? 6 : frame[34 +
IHL + i]]);
    }
}
```

Tests

Like in the IP protocol, TCP may o may not have options at the end of its header, when the header is bigger than 20 bytes, or the Offset is bigger than 5, we could know that the frame has options.

The first frame for TCP, without options:

```
0x02 0xff 0x53 0xc3 0xe9 0xab 0x00 0xff 0x66 0x7f 0xd4 0x3c 0x08 0x00 0x45 0x00
0x00 0x28 0x2d 0x00 0x40 0x00 0x80 0x06 0x4a 0x7c 0xc0 0xa8 0x01 0x02 0xc0 0xa8
0x01 0x01 0x04 0x03 0x00 0x15 0x00 0x3b 0xcf 0x45 0x21 0x5d 0x3a 0x45 0x50 0x10
0x22 0x38 0xdb 0x0d 0x00 0x00
```

The execution of the program at that point (figure 8):

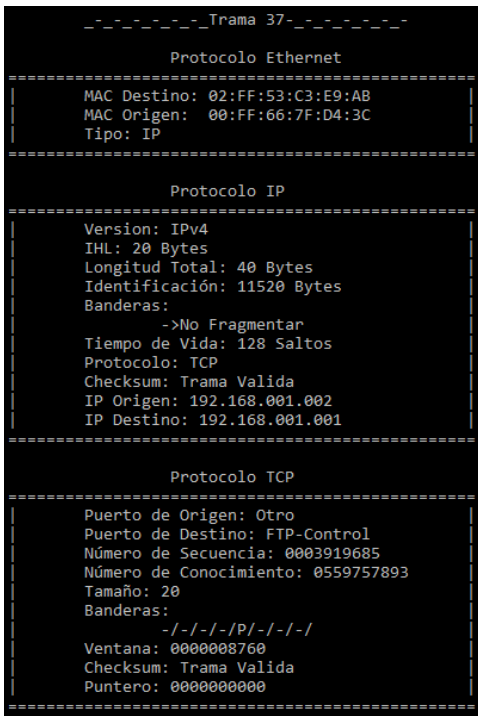
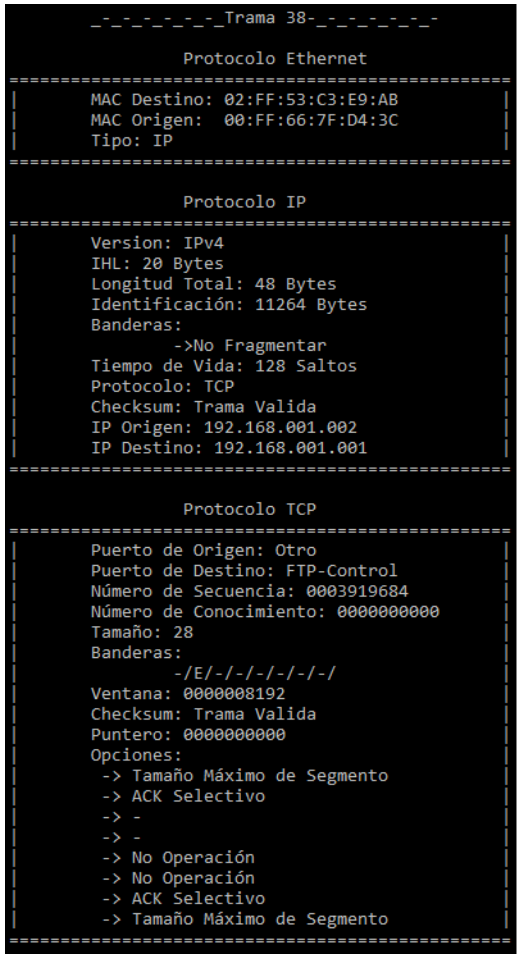


Figure 8 Execution at Frame 37

The second frame for TCP, with options:

```
0x02 0xff 0x53 0xc3 0xe9 0xab 0x00 0xff 0x66 0x7f 0xd4 0x3c 0x08 0x00 0x45 0x00
0x00 0x30 0x2c 0x00 0x40 0x00 0x80 0x06 0x4b 0x74 0xc0 0xa8 0x01 0x02 0xc0 0xa8
0x01 0x01 0x04 0x03 0x00 0x15 0x00 0x3b 0xcf 0x44 0x00 0x00 0x00 0x00 0x70 0x02
0x20 0x00 0x0c 0x34 0x00 0x00 0x02 0x04 0x05 0xb4 0x01 0x01 0x04 0x02
```

The execution of the program at that point (figure 9):



```
_-_-_-_-_-_-_Trama 38-_-_-_-_-_-_-

              Protocolo Ethernet
================================================
|       MAC Destino: 02:FF:53:C3:E9:AB         |
|       MAC Origen:  00:FF:66:7F:D4:3C         |
|       Tipo: IP                               |
================================================

              Protocolo IP
================================================
|       Version: IPv4                          |
|       IHL: 20 Bytes                          |
|       Longitud Total: 48 Bytes               |
|       Identificación: 11264 Bytes            |
|       Banderas:                              |
|               ->No Fragmentar                |
|       Tiempo de Vida: 128 Saltos             |
|       Protocolo: TCP                         |
|       Checksum: Trama Valida                 |
|       IP Origen: 192.168.001.002             |
|       IP Destino: 192.168.001.001            |
================================================

              Protocolo TCP
================================================
|       Puerto de Origen: Otro                 |
|       Puerto de Destino: FTP-Control         |
|       Número de Secuencia: 0003919684        |
|       Número de Conocimiento: 0000000000     |
|       Tamaño: 28                             |
|       Banderas:                              |
|               -/E/-/-/-/-/-/-/               |
|       Ventana: 0000008192                    |
|       Checksum: Trama Valida                 |
|       Puntero: 0000000000                    |
|       Opciones:                              |
|        -> Tamaño Máximo de Segmento          |
|        -> ACK Selectivo                      |
|        -> -                                  |
|        -> -                                  |
|        -> No Operación                       |
|        -> No Operación                       |
|        -> ACK Selectivo                      |
|        -> Tamaño Máximo de Segmento          |
================================================
```

Figure 9 Execution at Frame 38

UDP

Acronym and Description

User Datagram Protocol

With the User Datagram Protocol, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network. Prior communications are not required to set up communication channels or data paths.

Header

```
          1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

Source Port	Destination Port
Length	Checksum

Fields

- **Source Port:** This field identifies the sender's port when meaningful and should be assumed to be the port to reply to if needed. If not used, then it should be zero.
- **Destination Port:** This field identifies the receiver's port and is required.
- **Length:** A field that specifies the length in bytes of the UDP header and UDP data. The minimum length is 8 bytes because that is the length of the header.
- **Checksum:** The checksum field may be used for error-checking of the header and data. This field is optional in IPv4, and mandatory in IPv6.

Code

```
printf("\t==================================================\n");
printf("\n\t\t\t Protocolo UDP\n");
printf("\t==================================================\n");
aux = ((frame[14 + IHL] << 8) | frame[15 + IHL]);
printf("\t|\tPuerto de Origen: %s\t\t|\n", (aux ^ 7) ? ((aux ^ 19) ? ((aux ^ 37) ?
((aux ^ 53) ? ((aux ^ 67) ? ((aux ^ 68) ? ((aux ^ 69) ? ((aux ^ 137) ? ((aux ^ 138)
? ((aux ^ 161) ? ((aux ^ 162) ? ((aux ^ 500) ? ((aux ^ 514) ? ((aux ^ 520) ? ((aux
^ 33434) ? "Otro\t" : "Traceroute") : "RIP\t") : "SYSLOG") : "ISAKMP") : "SNMP-
Trap") : "SNMP\t") : "Netbios-DGM") : "Netbios-NS") : "TFTP\t") : "BOOTPC") :
"BOOTPS") : "Domain") : "Time\t") : "Chargen") :"Echo\t");
aux = ((frame[16 + IHL] << 8) | frame[17 + IHL]);
printf("\t|\tPuerto de Destino: %s\t\t|\n", (aux ^ 7) ? ((aux ^ 19) ? ((aux ^ 37) ?
((aux ^ 53) ? ((aux ^ 67) ? ((aux ^ 68) ? ((aux ^ 69) ? ((aux ^ 137) ? ((aux ^ 138)
? ((aux ^ 161) ? ((aux ^ 162) ? ((aux ^ 500) ? ((aux ^ 514) ? ((aux ^ 520) ? ((aux
^ 33434) ? "Otro\t" : "Traceroute") : "RIP\t") : "SYSLOG") : "ISAKMP") : "SNMP-
Trap") : "SNMP\t") : "Netbios-DGM") : "Netbios-NS") : "TFTP\t") : "BOOTPC") :
"BOOTPS") : "Domain") : "Time\t") : "Chargen") :"Echo\t");
aux = ((frame[18 + IHL] << 8) | frame[19 + IHL]);
if ((aux < 8))
{
    printf("\t-------->Parámetros Minimos No Correctos<--------\n");
    return;
}
printf("\t|\tTamaño: %.3d Bytes\t\t\t|\n", aux);
printf("\t|\tChecksum: %s\t\t|\n", checksum(frame, 14 + IHL, 13 + IHL + aux, 1));
```

Tests

For the UDP header I only test one frame. The length of this header is considerably small, so it wouldn’t made a difference if I had analyze a large number of frame.

The frame use it's the same as the one that was used for IP with options:

```
0x00 0x1f 0x45 0x9d 0x1e 0xa2 0x00 0x23 0x8b 0x46 0xe9 0xad 0x08 0x00 0x46 0x00
0x80 0x42 0x04 0x55 0x34 0x11 0x80 0x11 0x2c 0x11 0x94 0xcc 0x39 0xcb 0x94 0xcc
0x67 0x02 0x07 0x44 0x83 0x89 0x04 0x0c 0x00 0x35 0x00 0x2e 0x85 0x7c 0xe2 0x1a
0x01 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x03 0x77 0x77 0x77 0x03 0x69
0x73 0x63 0x05 0x65 0x73 0x63 0x6f 0x6d 0x03 0x69 0x70 0x6e 0x02 0x6d 0x78 0x00
0x00 0x1c 0x00 0x01
```

The execution of the program at that point (figure 10):

```
        _-_-_-_-_-_-_Trama 39-_-_-_-_-_-_-

                 Protocolo Ethernet
=================================================
|       MAC Destino: 00:1F:45:9D:1E:A2          |
|       MAC Origen:  00:23:8B:46:E9:AD          |
|       Tipo: IP                                |
=================================================

                 Protocolo IP
=================================================
|       Version: IPv4                           |
|       IHL: 24 Bytes                           |
|       Longitud Total: 32834 Bytes             |
|       Identificación: 1109 Bytes              |
|       Banderas:                               |
|               ->Fragmentar Más                |
|       Tiempo de Vida: 128 Saltos              |
|       Protocolo: UDP                          |
|       Checksum: Trama Valida                  |
|       IP Origen: 148.204.057.203              |
|       IP Destino: 148.204.103.002             |
|       Opciones:                               |
|        ->Flag: No Copiar                      |
|        ->Clase: Control                       |
|        ->Opción: Record de Ruta               |
|                                               |
|        ->Flag: No Copiar                      |
|        ->Clase: Debugueo y Medidas            |
|        ->Opción: Timestamp                    |
|                                               |
|        ->Flag: Copiar                         |
|        ->Clase: Control                       |
|        ->Opción: Ruta de Origen Ligera        |
|                                               |
|        ->Flag: Copiar                         |
|        ->Clase: Control                       |
|        ->Opción: Ruta de Origen Estricta      |
|                                               |
=================================================

                 Protocolo UDP
=================================================
|       Puerto de Origen: Otro                  |
|       Puerto de Destino: Domain               |
|       Tamaño: 046 Bytes                       |
|       Checksum: Trama Valida                  |
=================================================
```

Figure 10 Execution at Frame 39

Conclusions

This might be the longest we spent developing an implementation of what we saw in class, and although we didn't see everything about it, I think we saw everything we could. Not only did we now understand what a frame is, or why is so important that we, as futures engineers knowing how certain protocols work is essential. As well, for future subjects it later semesters as we continue to built our knowledge, I think this practice really make us gain a solid background as how information is sent throughout the internet and why that kinds of process has worked or why it's the more efficient.

References

- Www4.ncsu.edu. (2018). *UDP Checksum HowTo*. [online] Available at: http://www4.ncsu.edu/~mlsichit/Teaching/407/Resources/udpChecksum.html [Accessed 6 Dec. 2018].
- Eit.lth.se. (2018). *IP Protocol Header*. [online] Available at: https://www.eit.lth.se/ppplab/IPHeader.htm [Accessed 6 Dec. 2018].
- Telescript.denayer.wenk.be. (2018). *UDP Header*. [online] Available at: http://telescript.denayer.wenk.be/~hcr/cn/idoceo/udp_header.html [Accessed 6 Dec. 2018].
- Networksorcery.com. (2018). *TCP, Transmission Control Protocol*. [online] Available at: http://www.networksorcery.com/enp/protocol/tcp.htm [Accessed 6 Dec. 2018].