ESCOM - IPN

# Reporte Practica 1 y 2:

**Tipo de Trama, Dirección MAC y diferentes Checksums**

Redes de Computadora

Oscar Andrés Rosas Hernandez y Arturo Rivas Rojas

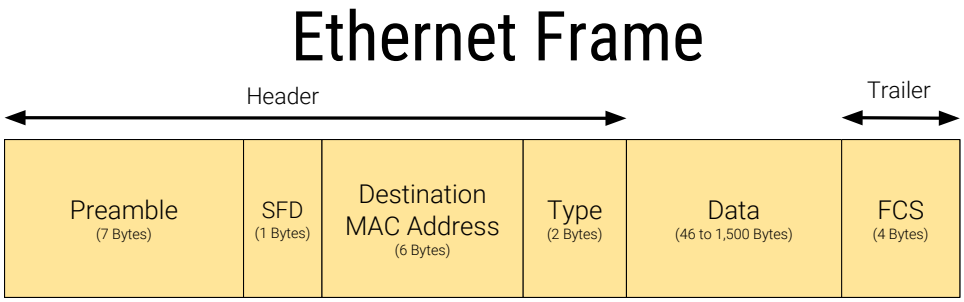
Realizada Marzo 2018
Entregada 23 de Marzo 2018

# Índice general

# Capítulo 1

# Marco Teórico

## 1.1. Protocolo Ethernet




### 1.1.1. Explicación

Los campos que componen una trama ethernet son los siguientes:

- **Preámbulo (Preamble)**

  Campo con una secuencia de bits utilizada para sincronizar y estabilizar el medio físico antes de iniciar la transmisión. Es una secuencia de unos y ceros. conocida que permite a los nodos saber que esta llegando un nuevo frame.

  El patrón es el siguiente:

  10101010101010101010101010101010101010101010101010101010

  *Tamaño: 7 Bytes*

- **SFD (Start Frame Delimiter)**

  Delimitador de inicio de trama. Campo que contiene la secuencia 10101011.

  Indica el inicio de una trama de datos.

  *Tamaño: 1 Byte*

- **Dirección de Destino (Destination Address)**

  Campo que contiene la dirección MAC a la que se envía la trama.

  El bit más a la izquierda del campo indica cuando la dirección es individual (indicado por un 0) o un grupo de direcciones (indicado por un 1). El segundo bit desde la izquierda indica cuando la dirección destino es globalmente administrada (indicado por un 0). La capa de enlace de datos del remitente añade la dirección de destino a la trama. La capa de enlace de datos del destinatario examina la dirección de destino para identificar los mensajes a recibir.

  *Tamaño: 6 Bytes*

- **Dirección de Origen (Source Address)**

  Campo que contiene la dirección MAC del dispositivo que envía la trama. La dirección de origen es siempre una dirección individual y el bit más a la izquierda es siempre 0. Con ella el receptor conoce a quien debe dirigir las respuestas del mensaje.

  *Tamaño: 6 Bytes*

- **Tipo de Protocolo o Longitud**

  Este campo es el que distingue a las tramas IEEE 802.3 de las tramas Ethernet.

  Valores para este campo iguales o menores de x05DC (1500 en decimal) indican que es una trama IEEE 802.3 y el valor representa la longitud del campo de datos.

  Valores para este campo iguales o mayores de x0600 indican que es una trama Ethernet y el valor representa el tipo de protocolo.

  *Tamaño: 2 Bytes*

- **Datos and Pad(Payload)**

  Contiene los datos a transferir entre origen y destino. Si este campo fuera menor de 46 bytes se añade un campo de "relleno", es decir pad para mantener el tamaño mínimo de paquete.

  *Tamaño: 46 a 1,500 Bytes*

- **FCS (Frame Check Sequence)**

  Secuencia de verificación de trama.

  Campo que contiene un valor de para control de errores, CRC (Cyclical Redundancy Check). La verificación de redundancia cíclica (CRC), consiste en un valor calculado por el emisor que resume todos los datos de la trama. El receptor calcula nuevamente el valor y, si coincide con el de la trama, entiende que la trama se ha

  El campo FCS es generado ó calculado sobre los campos dirección de destino, la dirección de origen, el tipo/longitud y datos.

  *Tamaño: 4 Bytes*

## 1.2. Suma de Comprobación: CheckSum

Este algoritmo permite verificar la integridad de la PDU y su calculo es de la siguiente manera:

- Ordena los datos en palabras de 16 bits
- Poner ceros en la posición del checksum y sumar con acarreos
- Suma cualquier acarreo fuera de los 16 bits
- Complementar a uno

## 1.3. Protocolo IP

### 1.3.1. Definiciones

Debido a la cantidad de cables necesarios para conectar cada red con cada otra red del mundo no todas las redes tienen una conexión directa, es decir, no existe un cable entre tu red lócal y los servidores de Facebook por ejemplo.

Por eso existe el Protocolo IP que nos permite comunicarnos entre redes.

En resumén lo que permite es que tu red local solo este conectada a unas pocas redes y a varios routers, estos tienen algo llamado una tabla de direcciones, que les permite navegar entre redes hasta encontrar su destino.

El enrutamiento es parecido a la recursión, en el sentido en que no soluciona tu problema sino que solo te lleva un paso más cerca.

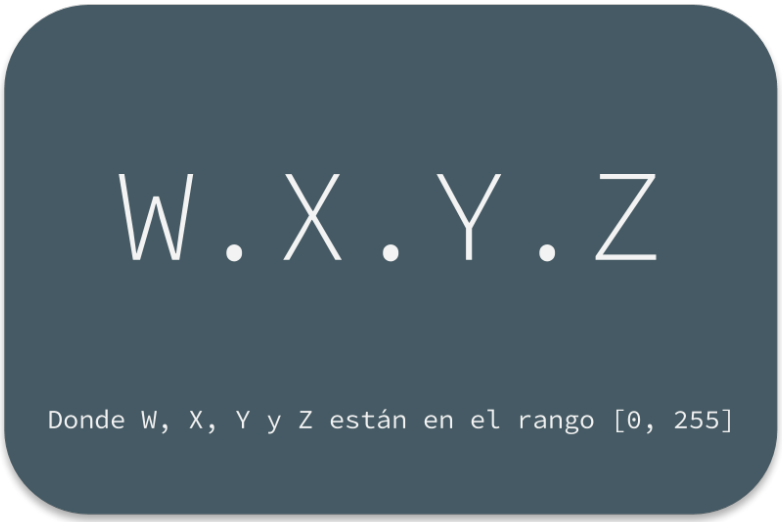
#### Direcciones IP

Es un identificador único (o casi, ya verás después porque). Necesitamos un identificador único porque es lo que nos permite enviar información y que la información que esperamos de regreso sepa a donde llegar.

### 1.3.2. Dirección IPv4

Como fue originalmente desarrollado este esquema podría alocar un identificador de **32 bits** a cada dispositivo que se quisiera conectar a internet. Esto nos daría algo así como 4 mill millones de posibles direcciones IP.

La convención es que estos serían representados como 4 conjuntos de 8 bits representados en decimal (una forma un poquito más amigable al público general), es decir:




Por ejemplo una IP v4 válida podría ser 140.247.220.12.

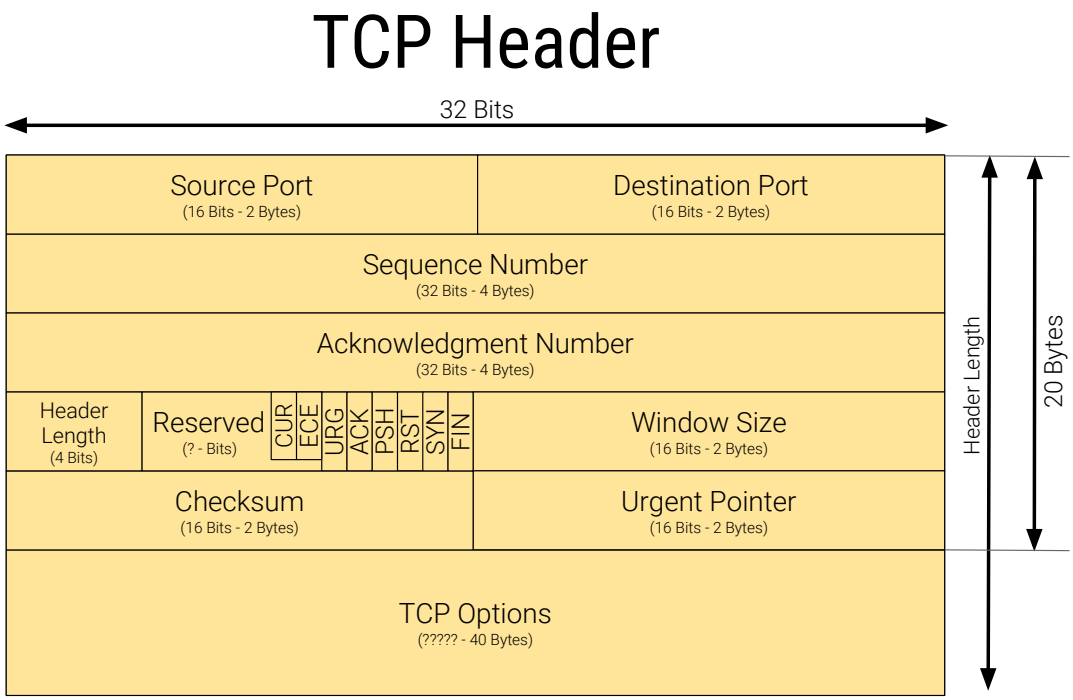
### 1.3.3. Problemas con IPv4

Ahora, recuerda que te dige que IP v4 acepta unos 4 mil millones de direcciones válidas, ahora el problema es que ahora mismo hay vivos mas de 7 mil millones de personas (A principios del siglo XXI) cada una con seguramente más de un dispositivo que quieran conectar a internet.

Por lo tanto tenemos que encontrar una forma de solucionar esto.

## 1.4. Protocolo TCP

### 1.4.1. Header - Encabezado



# Capítulo 2

# Practica 1

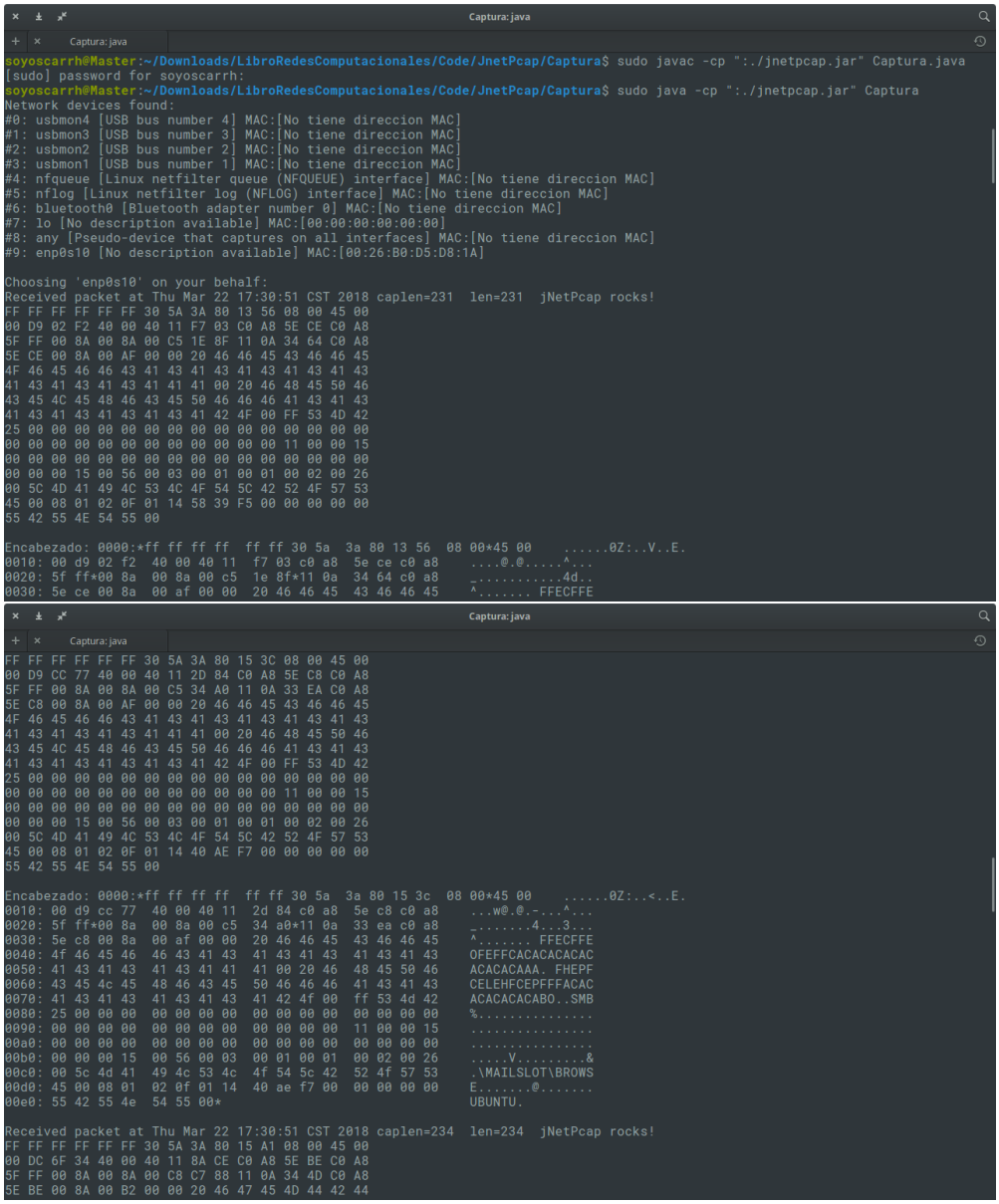
## 2.1. Desarrollo

La práctica consiste en capturar tramas que pasan a través de la tarjeta de red configurada en modo promiscuo. A partir de estas tramas analizamos si se trataba de una trama Ethernet, y posteriormente si el protocolo de Internet era IP. Para ello utilizamos la librería pcap y el código que nos fue proporcionado por el profesor.

Ahora procediamos con la parte inreresante, encontrar las direcciones MAC de origen y destino así como el tipo de red de cada una de las tramas analizadas, para hacerlo usabamos el siguiente algoritmo.

1. Creabamos dos variables temporables que almacenarias las direcciones mac así como una tercera que sería el tipo.
2. Para encontrar la MAC de origen lo que haciamos es que contabamos los primeros 6 bytes de la trama y los añadiamos uno por uno a la variable temporal, de una manera parecida con lo siguientes 6 para la MAC destino.
3. Para encontrar el tipo tendríamos que usar algo de aritmetica de corrimiento para obtener el número formado por los bytes 12, 13
4. Finalmente procediamos a mostrarlo por pantalla los 3 resultados

## 2.2. Capturas


Captura: java
Captura: java
soyoscarrh@Master:~/Downloads/LibroRedesComputacionales/Code/JnetPcap/Captura$ sudo javac -cp ":./jnetpcap.jar" Captura.java
[sudo] password for soyoscarrh:
soyoscarrh@Master:~/Downloads/LibroRedesComputacionales/Code/JnetPcap/Captura$ sudo java -cp ":./jnetpcap.jar" Captura
Network devices found:
#0: usbmon4 [USB bus number 4] MAC:[No tiene direccion MAC]
#1: usbmon3 [USB bus number 3] MAC:[No tiene direccion MAC]
#2: usbmon2 [USB bus number 2] MAC:[No tiene direccion MAC]
#3: usbmon1 [USB bus number 1] MAC:[No tiene direccion MAC]
#4: nfqueue [Linux netfilter queue (NFQUEUE) interface] MAC:[No tiene direccion MAC]
#5: nflog [Linux netfilter log (NFLOG) interface] MAC:[No tiene direccion MAC]
#6: bluetooth0 [Bluetooth adapter number 0] MAC:[No tiene direccion MAC]
#7: lo [No description available] MAC:[00:00:00:00:00:00]
#8: any [Pseudo-device that captures on all interfaces] MAC:[No tiene direccion MAC]
#9: enp0s10 [No description available] MAC:[00:26:B0:D5:D8:1A]
Choosing 'enp0s10' on your behalf:
Received packet at Thu Mar 22 17:30:51 CST 2018 caplen=231 len=231 jNetPcap rocks!
FF FF FF FF FF FF 30 5A 3A 80 13 56 08 00 45 00
00 D9 02 F2 40 00 40 11 F7 03 C0 A8 5E CE C0 A8
5F FF 00 8A 00 8A 00 C5 1E 8F 11 0A 34 64 C0 A8
5E CE 00 8A 00 AF 00 00 20 46 46 45 43 46 46 45
4F 46 45 46 46 43 41 43 41 43 41 43 41 43 41 43
41 43 41 43 41 43 41 41 00 20 46 48 45 50 46
43 45 4C 45 48 46 43 45 50 46 46 46 41 43 41 43
41 43 41 43 41 43 41 43 41 42 4F 00 FF 53 4D 42
25 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 11 00 00 15
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 15 00 56 00 03 00 01 00 01 00 02 00 26
00 5C 4D 41 49 4C 53 4C 4F 54 5C 42 52 4F 57 53
45 00 08 01 02 0F 01 14 58 39 F5 00 00 00 00 00
55 42 55 4E 54 55 00
Encabezado: 0000:*ff ff ff ff ff ff 30 5a 3a 80 13 56 08 00*45 00 ......0Z:..V..E.
0010: 00 d9 02 f2 40 00 40 11 f7 03 c0 a8 5e ce c0 a8 ....@.@.....^...
0020: 5f ff*00 8a 00 8a 00 c5 1e 8f*11 0a 34 64 c0 a8 _...........4d..
0030: 5e ce 00 8a 00 af 00 00 20 46 46 45 43 46 46 45 ^....... FFECFFE
Captura: java
Captura: java
FF FF FF FF FF FF 30 5A 3A 80 15 3C 08 00 45 00
00 D9 CC 77 40 00 40 11 2D 84 C0 A8 5E C8 C0 A8
5F FF 00 8A 00 8A 00 C5 34 A0 11 0A 33 EA C0 A8
5E C8 00 8A 00 AF 00 00 20 46 46 45 43 46 46 45
4F 46 45 46 46 43 41 43 41 43 41 43 41 43 41 43
41 43 41 43 41 43 41 41 00 20 46 48 45 50 46
43 45 4C 45 48 46 43 45 50 46 46 46 41 43 41 43
41 43 41 43 41 43 41 43 41 42 4F 00 FF 53 4D 42
25 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 11 00 00 15
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 15 00 56 00 03 00 01 00 01 00 02 00 26
00 5C 4D 41 49 4C 53 4C 4F 54 5C 42 52 4F 57 53
45 00 08 01 02 0F 01 14 40 AE F7 00 00 00 00 00
55 42 55 4E 54 55 00
Encabezado: 0000:*ff ff ff ff ff ff 30 5a 3a 80 15 3c 08 00*45 00 ......0Z:..<..E.
0010: 00 d9 cc 77 40 00 40 11 2d 84 c0 a8 5e c8 c0 a8 ...w@.@.-...^...
0020: 5f ff*00 8a 00 8a 00 c5 34 a0*11 0a 33 ea c0 a8 _.......4...3...
0030: 5e c8 00 8a 00 af 00 00 20 46 46 45 43 46 46 45 ^....... FFECFFE
0040: 4f 46 45 46 46 43 41 43 41 43 41 43 41 43 41 43 OFEFFCACACACACAC
0050: 41 43 41 43 41 43 41 41 41 00 20 46 48 45 50 46 ACACACAAA. FHEPF
0060: 43 45 4c 45 48 46 43 45 50 46 46 46 41 43 41 43 CELEHFCEPFFFACAC
0070: 41 43 41 43 41 43 41 43 41 42 4f 00 ff 53 4d 42 ACACACACABO..SMB
0080: 25 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 %...............
0090: 00 00 00 00 00 00 00 00 00 00 00 00 11 00 00 15 ................
00a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00b0: 00 00 00 15 00 56 00 03 00 01 00 01 00 02 00 26 .....V.........&
00c0: 00 5c 4d 41 49 4c 53 4c 4f 54 5c 42 52 4f 57 53 .\MAILSLOT\BROWS
00d0: 45 00 08 01 02 0f 01 14 40 ae f7 00 00 00 00 00 E.......@.......
00e0: 55 42 55 4e 54 55 00* UBUNTU.
Received packet at Thu Mar 22 17:30:51 CST 2018 caplen=234 len=234 jNetPcap rocks!
FF FF FF FF FF FF 30 5A 3A 80 15 A1 08 00 45 00
00 DC 6F 34 40 00 40 11 8A CE C0 A8 5E BE C0 A8
5F FF 00 8A 00 8A 00 C8 C7 88 11 0A 34 4D C0 A8
5E BE 00 8A 00 B2 00 00 20 46 47 45 4D 44 42 44

## 2.3. Código

```
/*=============================================================
==========                  MAIN                 ==========
=============================================================*/
public static void main (String[] Args) {

    StringBuilder ErrorData = new StringBuilder();
    PcapIf Device = SelectDevicesByConsole();

    if (Device == null) return;

    /*=========================================
    ==========    START THE PCAP    ==========
    =========================================
    Remember:
        - SnapshotLength
            Refers to the amount of actual data captured
            from each packet passing through the specified network interface.
            64*1024 = 65536 bytes
        */

    int SnapshotLength  = 64 * 1024;               // Capture all packets, no trucation
    int Flags           = Pcap.MODE_PROMISCUOUS;   // capture all packets
    int Timeout         = 10 * 1000;               // 10 seconds in millis

    Pcap PcapInstance = Pcap.openLive(
        Device.getName(),
        SnapshotLength,
        Flags,
        Timeout,
        ErrorData
    );

    if (PcapInstance == null) {
        System.err.printf("Error while opening device: " + ErrorData.toString());
        return;
    }


    /*=========================================
    ==========        FILTER       ==========
    =========================================*/
    PcapBpfProgram Filter = new PcapBpfProgram();

    String Expression = "";             // "port 80";
    int Optimize      = 0;              // 1 means true, 0 means false
    int Netmask       = 0;              // Netmask value

    int Result = PcapInstance.compile(
        Filter,
        Expression,
        Optimize,
        Netmask
    );

    if (Result != Pcap.OK)
        System.out.println("Filter error: " + PcapInstance.getErr());

    PcapInstance.setFilter(Filter);



    /*=========================================
    ======    CREATE PACKET HANDLER    ======
    =========================================*/
    PcapPacketHandler<String> JPacketHandler = new PcapPacketHandler<String>() {

        public void nextPacket(PcapPacket Packet, String User) {

            /*=====================================
            ======          SHOW INFO          ======
            =====================================*/

            System.out.println("=============================================================");
            System.out.println("==========    PACKET    ==========");
            System.out.println("=============================================================\n\n");

            System.out.printf("\nReceived at %s",new
Date(Packet.getCaptureHeader().timestampInMillis()));
            System.out.printf("\nCapture Length = %-4d", Packet.getCaptureHeader().caplen());
            System.out.printf("\nOriginal Sizeh = %-4d", Packet.getCaptureHeader().wirelen());
            System.out.printf("\nUSer           = %s\n\n\n", User);

            String MACAddressOrigin  = "";
            String MACAddressDestiny = "";
```

```
84
85                  /*=============================================
86                  ====    SHOW RAW DATA & GET MAC'S    ====
87                  =============================================*/
88                  for (int i = 0; i < Packet.size(); i++) {
89
90                      System.out.printf("%02X ", Packet.getUByte(i));
91
92                      if (i %16 == 15) System.out.println("");
93
94                      if (i < 6)
95                          MACAddressOrigin += String.format("%02X ", Packet.getUByte(i));
96                      else if (i < 12)
97                          MACAddressDestiny += String.format("%02X ", Packet.getUByte(i));
98
99                  }
100                 System.out.println("\n\n\n\n");
101
102                 /*=============================================
103                 ====         SHOW MAC'S & TYPE           ====
104                 =============================================*/
105                 int Type = Packet.getUByte(12) * 256 + Packet.getUByte(13);
106
107                 System.out.println("MAC Origin  = " + MACAddressOrigin);
108                 System.out.println("MAC Destiny = " + MACAddressDestiny);
109                 System.out.printf("Type        = %04X\n", Type);
110
111
112                 /*=============================================
113                 ====          IS AN IP PACKET?           ====
114                 =============================================*/
115                 if ((Type & 0xFFFF) == 0x0800) {
116
117                     System.out.println("\nIs an IPv4 Packet!");
118
119                     byte[] PacketAsByteArray = Packet.getByteArray(0, Packet.size());
120
121                     int IPPacketSize = (PacketAsByteArray[14] & 0x0F) * 4;
122                     byte[] IPHeader = new byte[IPPacketSize];
123                     System.arraycopy(PacketAsByteArray, 14, IPHeader, 0, IPPacketSize);
124
125                     //IPHeader[10] = 0x00;
126                     //IPHeader[11] = 0x00;
127
128                     int IPacketSize = (((IPHeader[2] << 8) & 0xFF00) | ((IPHeader[3]) & 0xFF));
129
130                     System.out.printf("Complemnt to 1 of Checksum IPv4: ");
131                     System.out.printf("%04X\n", CalculateChecksum(IPHeader));
132
133
134                     if (Packet.size() > (13 + IPPacketSize)) {
135
136                         /*=============================================
137                         ====          IS AN TCP PACKET?          ====
138                         =============================================*/
139                         if (IPHeader[9] == 0x06) {
140
141                             System.out.println("\n\nIs an TCP Packet!");
142
143                             byte[] TCPHeader = new byte[12];
144
145                             for (int i = 0; i < 8; i++)
146                                 TCPHeader[i] = IPHeader[IPPacketSize - 8 + i];
147
148                             int TCPPacketSize = IPacketSize - IPPacketSize;
149
150                             TCPHeader[8]  = 0x00;
151                             TCPHeader[9]  = 0x06;
152                             TCPHeader[10] = (byte)(TCPPacketSize & 0x0000FF00);
153                             TCPHeader[11] = (byte)(TCPPacketSize & 0x000000FF);
154
155                             byte[] TCPPacket = new byte[TCPPacketSize + 12];
156                             System.arraycopy(TCPHeader, 0, TCPPacket, 0, 12);
157                             System.arraycopy(PacketAsByteArray, IPPacketSize + 14, TCPPacket, 12,
    TCPPacketSize);
158
159                             //TCPPacket[28] = 0x00;
160                             //TCPPacket[29] = 0x00;
161
162                             System.out.printf("Complemnt to 1 of Checksum TCP: ");
163                             System.out.printf("%04X\n", CalculateChecksum(TCPPacket));
164                         }
165
166                         /*=============================================
167                         ====          IS AN UDP PACKET?          ====
168                         =============================================*/
169                         if (IPHeader[9] == 0x11) {
170
```

```
171                             System.out.println("\n\nIs an UDP Packet!");
172
173                             byte[] UDPHeader = new byte[12];
174
175                             for (int i = 0; i < 8; i++)
176                                 UDPHeader[i] = IPHeader[IPPacketSize - 8 + i];
177
178                             int UDPPacketSize = IPacketSize - IPPacketSize;
179
180                             UDPHeader[8]  = 0x00;
181                             UDPHeader[9]  = 0x11;
182                             UDPHeader[10] = (byte)(UDPPacketSize & 0x0000FF00);
183                             UDPHeader[11] = (byte)(UDPPacketSize & 0x000000FF);
184
185                             byte[] UDPPacket = new byte[UDPPacketSize + 12];
186                             System.arraycopy(UDPHeader, 0, UDPPacket, 0, 12);
187                             System.arraycopy(PacketAsByteArray, IPPacketSize + 14, UDPPacket, 12,
    UDPPacketSize);
188
189                             //UDPPacket[18] = 0x00;
190                             //UDPPacket[19] = 0x00;
191
192                             System.out.printf("Complemnt to 1 of Checksum UDP: ");
193                             System.out.printf("%04X\n", CalculateChecksum(UDPPacket));
194                         }
195                     }
196                 }
197                 else
198                     System.out.println("\nNot an IPv4 Packet");
199
200                 System.out.println ("\n\nRaw Data: \n" + Packet.toHexdump ());
201             }
202
203         };
204
205
206
207         /*=============================================
208         =====            DO A LOOP              =====
209         =============================================
210
211             Remember:
212                 Fourth we enter the loop and tell it to capture 5 packets. The loop
213                 method does a mapping of pcap.datalink() DLT value to JProtocol ID, which
214                 is needed by JScanner. The scanner scans the packet Datafer and decodes
215                 the headers. The mapping is done automatically, although a variation on
216                 the loop method exists that allows the programmer to sepecify exactly
217                 which protocol ID to use as the data link type for this pcap interface.
218         */
219         PcapInstance.loop(5, JPacketHandler, "In a Loop");
220
221         PcapInstance.close();
222     }
223
224 }
```

## 2.4. Conclusiones

**Arturo Rivas Rojas**:

La práctica fue realmente enriquecedora en cuanto a los temas del curso, ya que me ayudo a entender las bases de como van a ser las practicas posteriores, sobretodo en el uso de las biblioteca PCAP que fueron el 90 % de los problemas de esta practica, lograr que todo el sistema funcionara a la perfección.

También por otro lado, me proporciono claridad sobre como es que podemos pasar de lo que vemos teoricamente a un codigo real y a aplicaciones de la vida real.

**Rosas Hernandez Oscar Andrés**:

Es esta práctica implementamos, gracias a la biblioteca PCAP como es que podemos obtener y mostrar todo la información cruda que obtuvimos al hacer que nuestra tarjeta se ponga a recibir tramas en un modo promiscuo.

Vimos ademas como es que podemos obtener 3 elementos clave de dichas trama, primeramente las dirrecciones MACa, despues como es que tenemos que hacer algo de aritemetica con los corrientos para poder obtner e tipo, lo cual resultará un campo super importante en futuras practicas.

# Capítulo 3

# Practica 2

## 3.1. Desarrollo

La práctica consiste en capturar tramas que pasan a través de la tarjeta de red configurada en modo promiscuo. A partir de estas tramas analizamos si se trataba de una trama Ethernet, y posteriormente si el protocolo de Internet era IP. Para ello utilizamos la librería pcap y el código que nos fue proporcionado por el profesor.

Para esta práctica supusimos que la trama con la que se trataba era Ethernet, por lo cual no se revisó que el campo tipo/longitud fuera mayor o igual a 1500 bytes.

Posteriormente, revisamos el byte 13 y 14 de la trama para verificar que el protocolo que se iba a utilizar era IPv4, es decir, revisamos que el valor fuera igual a 0x08 en el byte 13 y 0x00en el byte 14.

Una vez validado el tipo de protocolo como IPv4 continuamos a calcular el checksum para verificar posibles errores y dar por buena la trama.

El procedimiento que seguimos fue el siguiente:

1. Calcular la longitud del encabezado IP.
2. Crear un nuevo arreglo de bytes para almacenar el encabezado.
3. Identificar IP de origen e IP destino.
4. Calcular el checksum utilizando el método proporcionado por el profesor.

Finalmente, procedimos a identificar el protocolo que se utilizó en la capa de transporte Para esto revisamos el valor del byte 24 de la trama. Si el valor resultaba ser 0x06, sabíamos que se trataba de TCP. Por otro lado, si el valor era 0x11, el protocolo era UDP.

Para ambos casos se requería calcular el valor del checksum. Sin embargo, para ambos había que calcular previamente un pseudo-header que se utiliza en el cálculo del checksum.

En cuanto al checksum de TCP, lo calculamos de la manera siguiente:

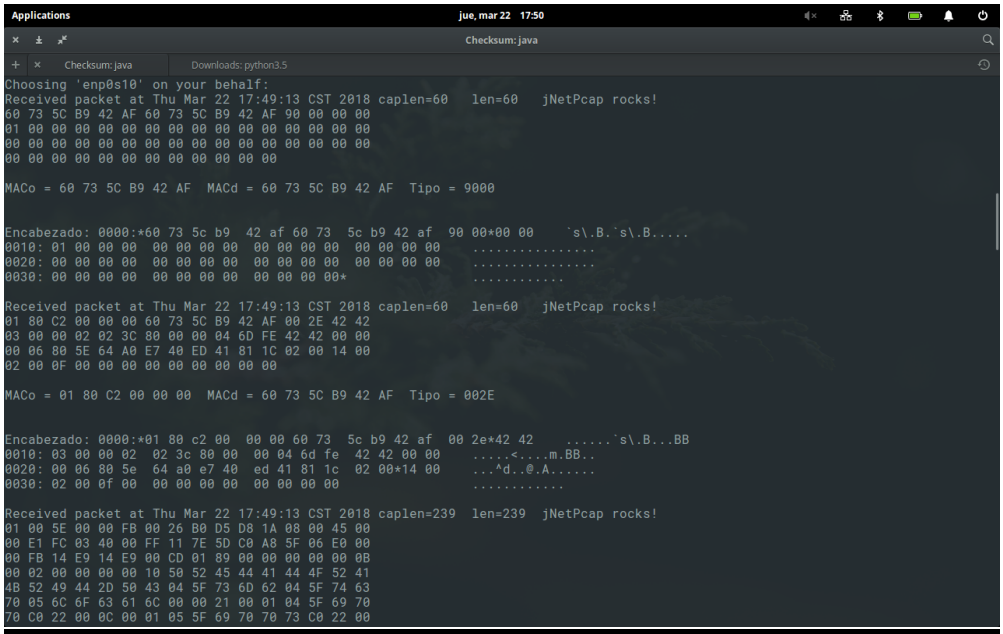
1. Calculamos la longitud del encabezado TCP.
2. Creamos un nuevo arreglo que almacenaba el encabezado TCP.
3. Creamos un arreglo que iba a contener la información correspondiente al pseudo-header.
4. Agregamos la IP de origen y destino al pseudo-header, en ese orden.

5. Agregamos la información correspondiente a los bytes 9 y 10. En el byte 9 se asigna el valor de 0 y en el byte 10 el valor de 0x06 correspondiente a que se trata de un protocolo TCP.
6. Calculamos la longitud del payload utilizando la longitud total menos la longitud del encabezado IP menos 14 que es la longitud del encabezado TCP.
7. Agregamos la información del payload a los bytes 11 y 12.
8. Creamos un nuevo arreglo que contendrá el payload de TCP y copiamos la información.
9. Creamos un arreglo auxiliar que contendrá: el pseudo-header, el encabezado TCP y el payload.
10. Calculamos el checksum utilizando el arreglo auxiliar.

Por otra parte el checksum de UPD, lo calculamos de la manera siguiente:

1. Creamos un arreglo que contendrá el encabezado UDP y copiamos la información.
2. Creamos un arreglo de 12 bytes que nos servirá para crear el pseudo-header.
3. Copiamos el IP origen y el IP destino al pseudo-header.
4. Inicializamos los bytes 9 y 10. El 9 se inicializa a 0x00 y el 10 a 0x11 debido a que se trata del protocolo UDP.
5. Copiamos el encabezado UDP al pseudo-header.
6. Calculamos la longitud del payload: longitud de la trama menos longitud de IP menos longitud de UDP menos 14.
7. Creamos un arreglo de bytes que contendrá el payload y copiamos la información de la trama.
8. Creamos un arreglo temporal de bytes para realizar el cálculo del checksum. Agregamos: pseudo-header, encabezado UDP y el payload, en ese orden.
9. Realizamos el cálculo del checksum.

## 3.2. Capturas


Applications
jue, mar 22 17:50
Checksum: java
Checksum: java
Downloads: python3.5
Choosing 'enp0s10' on your behalf:
Received packet at Thu Mar 22 17:49:13 CST 2018 caplen=60 len=60 jNetPcap rocks!
60 73 5C B9 42 AF 60 73 5C B9 42 AF 90 00 00 00
01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
MACo = 60 73 5C B9 42 AF MACd = 60 73 5C B9 42 AF Tipo = 9000
Encabezado: 0000:*60 73 5c b9 42 af 60 73 5c b9 42 af 90 00*00 00 `s\.B.`s\.B.....
0010: 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0030: 00 00 00 00 00 00 00 00 00 00 00 00 ............
Received packet at Thu Mar 22 17:49:13 CST 2018 caplen=60 len=60 jNetPcap rocks!
01 80 C2 00 00 00 60 73 5C B9 42 AF 00 2E 42 42
03 00 00 02 02 3C 80 00 00 04 6D FE 42 42 00 00
00 06 80 5E 64 A0 E7 40 ED 41 81 1C 02 00 14 00
02 00 0F 00 00 00 00 00 00 00 00 00
MACo = 01 80 C2 00 00 00 MACd = 60 73 5C B9 42 AF Tipo = 002E
Encabezado: 0000:*01 80 c2 00 00 00 60 73 5c b9 42 af 00 2e*42 42 ......`s\.B...BB
0010: 03 00 00 02 02 3c 80 00 00 04 6d fe 42 42 00 00 .....<....m.BB..
0020: 00 06 80 5e 64 a0 e7 40 ed 41 81 1c 02 00*14 00 ...^d..@.A.....
0030: 02 00 0f 00 00 00 00 00 00 00 00 00 ............
Received packet at Thu Mar 22 17:49:13 CST 2018 caplen=239 len=239 jNetPcap rocks!
01 00 5E 00 00 FB 00 26 B0 D5 D8 1A 08 00 45 00
00 E1 FC 03 40 00 FF 11 7E 5D C0 A8 5F 06 E0 00
00 FB 14 E9 14 E9 00 CD 01 89 00 00 00 00 00 0B
00 02 00 00 00 00 10 50 52 45 44 41 44 4F 52 41
4B 52 49 44 2D 50 43 04 5F 73 6D 62 04 5F 74 63
70 05 6C 6F 63 61 6C 00 00 21 00 01 04 5F 69 70
70 C0 22 00 0C 00 01 05 5F 69 70 70 73 C0 22 00
Applications
jue, mar 22 17:50
Checksum: java
Checksum: java
Downloads: python3.5
MACo = 33 33 00 00 00 FB MACd = 00 26 B0 D5 D8 1A Tipo = 86DD
Encabezado: 0000:*33 33 00 00 00 fb 00 26 b0 d5 d8 1a 86 dd*60 07 33.....&......`.
0010: 72 c3 00 52 11 ff fe 80 00 00 00 00 00 00 0f 50 r..R...........P
0020: cc fc ca 42 07 c3 ff 02 00 00 00 00 00 00 00 00 ...B............
0030: 00 00 00 00 00 fb*14 e9 14 e9 00 52 a3 39*00 00 ...........R.9..
0040: 00 00 00 02 00 01 00 00 00 00 10 70 72 65 64 61 ...........preda
0050: 64 6f 72 61 6b 72 69 64 2d 50 43 05 6c 6f 63 61 dorakrid-PC.loca
0060: 6c 00 00 01 00 01 c0 0c 00 1c 00 01 c0 0c 00 1c l...............
0070: 00 01 00 00 00 78 00 10 fe 80 00 00 00 00 00 00 .....x..........
0080: 3a d3 bf ab 16 a6 b5 e1* :.......
Received packet at Thu Mar 22 17:49:17 CST 2018 caplen=213 len=213 jNetPcap rocks!
01 00 5E 7F FF FA 1C 39 47 DC 10 CA 08 00 45 00
00 C7 C1 49 40 00 01 11 A8 A9 C0 A8 5E 90 EF FF
FF FA C4 55 07 6C 00 B3 63 B8 4D 2D 53 45 41 52
43 48 20 2A 20 48 54 54 50 2F 31 2E 31 0D 0A 48
4F 53 54 3A 20 32 33 39 2E 32 35 35 2E 32 35 35
2E 32 35 30 3A 31 39 30 30 0D 0A 4D 41 4E 3A 20
22 73 73 64 70 3A 64 69 73 63 6F 76 65 72 22 0D
0A 4D 58 3A 20 31 0D 0A 53 54 3A 20 75 72 6E 3A
64 69 61 6C 2D 6D 75 6C 74 69 73 63 72 65 65 6E
2D 6F 72 67 3A 73 65 72 76 69 63 65 3A 64 69 61
6C 3A 31 0D 0A 55 53 45 52 2D 41 47 45 4E 54 3A
20 47 6F 6F 67 6C 65 20 43 68 72 6F 6D 65 2F 36
32 2E 30 2E 33 32 30 32 2E 36 32 20 4C 69 6E 75
78 0D 0A 0D 0A
MACo = 01 00 5E 7F FF FA MACd = 1C 39 47 DC 10 CA Tipo = 0800
IPv4
El complemento a uno de la suma de IPv4: 0000
UDP
El complemento a uno de la suma de UDP: 0000

## 3.3. Código

```
/*=============================================================
==========                  MAIN                 ==========
=============================================================*/
public static void main (String[] Args) {

    StringBuilder ErrorData = new StringBuilder();
    PcapIf Device = SelectDevicesByConsole();

    if (Device == null) return;

    /*=========================================
    ==========    START THE PCAP    ==========
    =========================================
    Remember:
        - SnapshotLength
            Refers to the amount of actual data captured
            from each packet passing through the specified network interface.
            64*1024 = 65536 bytes
        */

    int SnapshotLength  = 64 * 1024;               // Capture all packets, no trucation
    int Flags           = Pcap.MODE_PROMISCUOUS;   // capture all packets
    int Timeout         = 10 * 1000;               // 10 seconds in millis

    Pcap PcapInstance = Pcap.openLive(
        Device.getName(),
        SnapshotLength,
        Flags,
        Timeout,
        ErrorData
    );

    if (PcapInstance == null) {
        System.err.printf("Error while opening device: " + ErrorData.toString());
        return;
    }


    /*=========================================
    ==========        FILTER       ==========
    =========================================*/
    PcapBpfProgram Filter = new PcapBpfProgram();

    String Expression = "";             // "port 80";
    int Optimize      = 0;              // 1 means true, 0 means false
    int Netmask       = 0;              // Netmask value

    int Result = PcapInstance.compile(
        Filter,
        Expression,
        Optimize,
        Netmask
    );

    if (Result != Pcap.OK)
        System.out.println("Filter error: " + PcapInstance.getErr());

    PcapInstance.setFilter(Filter);



    /*=========================================
    ======    CREATE PACKET HANDLER    ======
    =========================================*/
    PcapPacketHandler<String> JPacketHandler = new PcapPacketHandler<String>() {

        public void nextPacket(PcapPacket Packet, String User) {

            /*=====================================
            ======          SHOW INFO          ======
            =====================================*/

            System.out.println("=============================================================");
            System.out.println("==========    PACKET    ==========");
            System.out.println("=============================================================\n\n");

            System.out.printf("\nReceived at %s",new
Date(Packet.getCaptureHeader().timestampInMillis()));
            System.out.printf("\nCapture Length = %-4d", Packet.getCaptureHeader().caplen());
            System.out.printf("\nOriginal Sizeh = %-4d", Packet.getCaptureHeader().wirelen());
            System.out.printf("\nUSer           = %s\n\n\n", User);

            String MACAddressOrigin  = "";
            String MACAddressDestiny = "";
```

```
84
85                  /*=============================================
86                  ====    SHOW RAW DATA & GET MAC'S    ====
87                  =============================================*/
88                  for (int i = 0; i < Packet.size(); i++) {
89
90                      System.out.printf("%02X ", Packet.getUByte(i));
91
92                      if (i %16 == 15) System.out.println("");
93
94                      if (i < 6)
95                          MACAddressOrigin += String.format("%02X ", Packet.getUByte(i));
96                      else if (i < 12)
97                          MACAddressDestiny += String.format("%02X ", Packet.getUByte(i));
98
99                  }
100                 System.out.println("\n\n\n\n");
101
102                 /*=============================================
103                 ====        SHOW MAC'S & TYPE            ====
104                 =============================================*/
105                 int Type = Packet.getUByte(12) * 256 + Packet.getUByte(13);
106
107                 System.out.println("MAC Origin  = " + MACAddressOrigin);
108                 System.out.println("MAC Destiny = " + MACAddressDestiny);
109                 System.out.printf("Type        = %04X\n", Type);
110
111
112                 /*=============================================
113                 ====         IS AN IP PACKET?            ====
114                 =============================================*/
115                 if ((Type & 0xFFFF) == 0x0800) {
116
117                     System.out.println("\nIs an IPv4 Packet!");
118
119                     byte[] PacketAsByteArray = Packet.getByteArray(0, Packet.size());
120
121                     int IPPacketSize = (PacketAsByteArray[14] & 0x0F) * 4;
122                     byte[] IPHeader = new byte[IPPacketSize];
123                     System.arraycopy(PacketAsByteArray, 14, IPHeader, 0, IPPacketSize);
124
125                     //IPHeader[10] = 0x00;
126                     //IPHeader[11] = 0x00;
127
128                     int IPacketSize = (((IPHeader[2] << 8) & 0xFF00) | ((IPHeader[3]) & 0xFF));
129
130                     System.out.printf("Complemnt to 1 of Checksum IPv4: ");
131                     System.out.printf("%04X\n", CalculateChecksum(IPHeader));
132
133
134                     if (Packet.size() > (13 + IPPacketSize)) {
135
136                         /*=============================================
137                         ====         IS AN TCP PACKET?           ====
138                         =============================================*/
139                         if (IPHeader[9] == 0x06) {
140
141                             System.out.println("\n\nIs an TCP Packet!");
142
143                             byte[] TCPHeader = new byte[12];
144
145                             for (int i = 0; i < 8; i++)
146                                 TCPHeader[i] = IPHeader[IPPacketSize - 8 + i];
147
148                             int TCPPacketSize = IPacketSize - IPPacketSize;
149
150                             TCPHeader[8]  = 0x00;
151                             TCPHeader[9]  = 0x06;
152                             TCPHeader[10] = (byte)(TCPPacketSize & 0x0000FF00);
153                             TCPHeader[11] = (byte)(TCPPacketSize & 0x000000FF);
154
155                             byte[] TCPPacket = new byte[TCPPacketSize + 12];
156                             System.arraycopy(TCPHeader, 0, TCPPacket, 0, 12);
157                             System.arraycopy(PacketAsByteArray, IPPacketSize + 14, TCPPacket, 12,
    TCPPacketSize);
158
159                             //TCPPacket[28] = 0x00;
160                             //TCPPacket[29] = 0x00;
161
162                             System.out.printf("Complemnt to 1 of Checksum TCP: ");
163                             System.out.printf("%04X\n", CalculateChecksum(TCPPacket));
164                         }
165
166                         /*=============================================
167                         ====         IS AN UDP PACKET?           ====
168                         =============================================*/
169                         if (IPHeader[9] == 0x11) {
170
```

```
171                             System.out.println("\n\nIs an UDP Packet!");
172
173                             byte[] UDPHeader = new byte[12];
174
175                             for (int i = 0; i < 8; i++)
176                                 UDPHeader[i] = IPHeader[IPPacketSize - 8 + i];
177
178                             int UDPPacketSize = IPacketSize - IPPacketSize;
179
180                             UDPHeader[8]  = 0x00;
181                             UDPHeader[9]  = 0x11;
182                             UDPHeader[10] = (byte)(UDPPacketSize & 0x0000FF00);
183                             UDPHeader[11] = (byte)(UDPPacketSize & 0x000000FF);
184
185                             byte[] UDPPacket = new byte[UDPPacketSize + 12];
186                             System.arraycopy(UDPHeader, 0, UDPPacket, 0, 12);
187                             System.arraycopy(PacketAsByteArray, IPPacketSize + 14, UDPPacket, 12,
    UDPPacketSize);
188
189                             //UDPPacket[18] = 0x00;
190                             //UDPPacket[19] = 0x00;
191
192                             System.out.printf("Complemnt to 1 of Checksum UDP: ");
193                             System.out.printf("%04X\n", CalculateChecksum(UDPPacket));
194                         }
195                     }
196                 }
197                 else
198                     System.out.println("\nNot an IPv4 Packet");
199
200                 System.out.println ("\n\nRaw Data: \n" + Packet.toHexdump ());
201             }
202
203         };
204
205
206
207         /*=============================================
208         =====            DO A LOOP              =====
209         =============================================
210
211             Remember:
212                 Fourth we enter the loop and tell it to capture 5 packets. The loop
213                 method does a mapping of pcap.datalink() DLT value to JProtocol ID, which
214                 is needed by JScanner. The scanner scans the packet Datafer and decodes
215                 the headers. The mapping is done automatically, although a variation on
216                 the loop method exists that allows the programmer to sepecify exactly
217                 which protocol ID to use as the data link type for this pcap interface.
218         */
219         PcapInstance.loop(5, JPacketHandler, "In a Loop");
220
221         PcapInstance.close();
222     }
223
224 }
```

## 3.4. Conclusiones

**Arturo Rivas Rojas**:

La práctica fue realmente enriquecedora en cuanto a los temas del curso, ya que me ayudo a entender mejor sobre la estructura de las tramas Ethernet.

También por otro lado, me proporciono claridad sobre el principio de encapsulación que se debe de cumplir entre cada una de las capas del modelo.

Por último, fue interesante el tema de manipulación de la trama como tal para aislar el encabezado de los datos, así como para generar el pseudo-header necesario para la validación del checksum en TCP y UDP.

**Rosas Hernandez Oscar Andrés**:

Es esta práctica implementamos el algoritmo de Checksum aplicado a verificar tramas Ethernet. Comprobamos que dichas tramas fueran IPv4 verificando que los bytes 13 y 14 sean iguales a 0x0800, para posteriormente decidir si el protocolo de la capa de transporte era TCP o UDP. Finalmente, comprobamos el campo checksum de estos protocolos.

Al correr el programa, verificamos que varias tramas de TCP llegaban incompletas, por lo que el checksum era diferente de cero; mientras que las tramas UDP tendían a llegar sin errores la mayoría del tiempo.

Lo más importante de esta práctica, es la utilización del algoritmo de checksum para identificar los errores en una trama, la cual, asumimos inicialmente, que es una trama Ethernet.

El checksum se validó con base en el protocolo IPv4 y una vez que se ha dado por buena una trama, posteriormente verificamos si el protocolo para el transporte es TCP o UDP, en lo que se utilizó el valor calculado del checksum, en el que intervenía un pseudo-header.