ESCOM - IPN

Practica 2:

Implementación de Checksum para el Encabezado IP, TCP y UDP

Redes de Cómputadoras 2CM10

Oscar Andrés Rosas Hernandez Arturo Rivas Rojas

Junio 2018

Índice

1. Protocolo OSI

1.1. Definición

Antes que nada, es un modelo de referencia. Pretende que los sistemas que son diseñados con base en el, se pueden comunicar sin problemas.

1.2. Partes

- **Capa Física:**

  Se encarga de la transmición de datos, de cadenas de bits no estrucurados sobre el medio físico y esta relacionado con:

  - Voltaje necasario para representar cada bit
  - Cuanto dura cada símbolo, es decir el tiempo de trama
  - Si se realiza simultáneamente en ambos sentidos o no
  - Como se establece una transmición y como interrumpirla
  - Especificar como serán los pines del conector de red, para que sirve cada pin pues

- **Capa de Enlace de Red:**

  Trabaja con direcciónes físicas.

  Proporciona el servicio de transferencia de datos (tramas) llevando a cabo la sincronización y correción de datos, así como el control de flujo.

- **Capa de Red:**

  Trabaja con IP (es decir, direcciones lógicas) para poder conectar dos redes. Es responsable del establecimiento, mantenimiento y cierre de conexión.

  También brinda las funciones de direccionamiento lógico y enrutamiento.

  Se direccionan de manera lógica y no física para evitar problemas con el hardware.

- **Capa de Transporte:**

  Hablaremos de si será un archivo orientado a conexión (TCP), o si no esta orientado a conexión (UDP), es decir la importancia que le damos a si queremos los datos integros o si requerimos gran velocidad.

  Proporciona seguridad, transferencia y transporte de datos entre los puntos finales también proporcionan mecanismos de control de flujo y de errores en el origen y destino.

  Esta proporciona el control de la comunicación entre diferentes aplicaciones, establece, gestiona y cierra la comunicación entre aplicaciones

- **Capa de Sesión:**

  Hablaremos los números de puerto, un identificador de programa que nos permite ejecutar varias aplicaciones al mismo tiempo, estas son permite ejecutar unos 65,536 aplicaciones en red TCP y otros 65,536 en UDP. Si, un montón.

- **Presentación:**

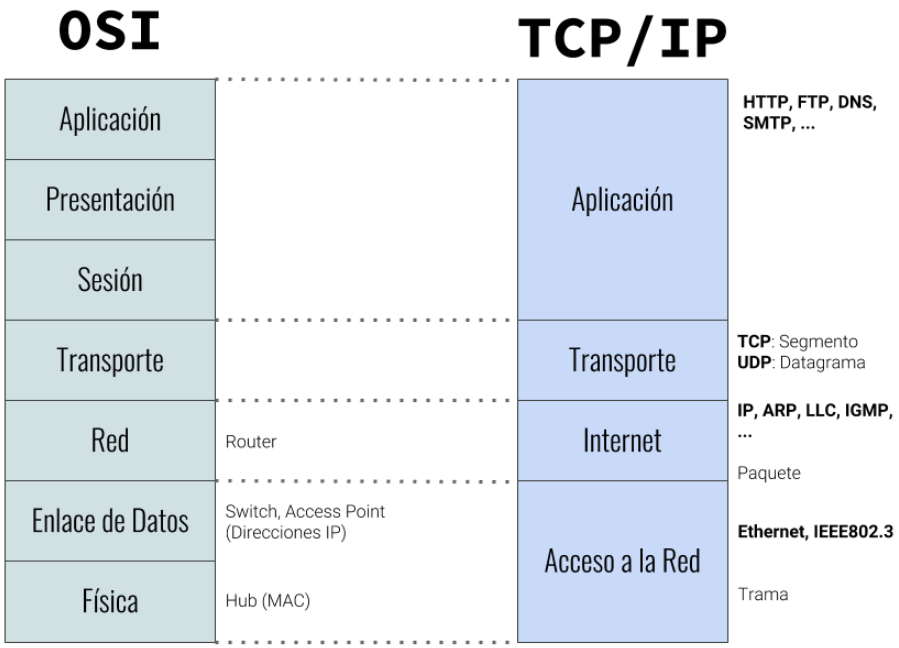
  Poder transmitir los datos de manera transparte y sin importar la arquitectura de las computadoras de origen y destino.

- **Aplicación:**

  Es donde trabajamos a nivel usario y ... poquito más.

  Proporciona un medio a los programas de aplicación para acceder a los servicios de red, contiene funciones de administración de aplicaciones distribuidas.

1.3. Diagrama OSI (Y su comparación vs IP)



2. Protocolo IP

2.1. 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.

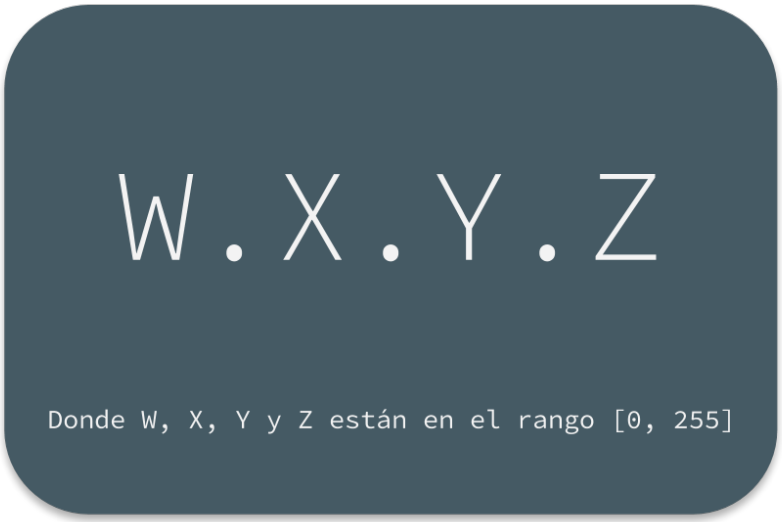
2.1.1. Direcciones IP

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.

2.2. 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.

2.2.1. Problemas con IPv4

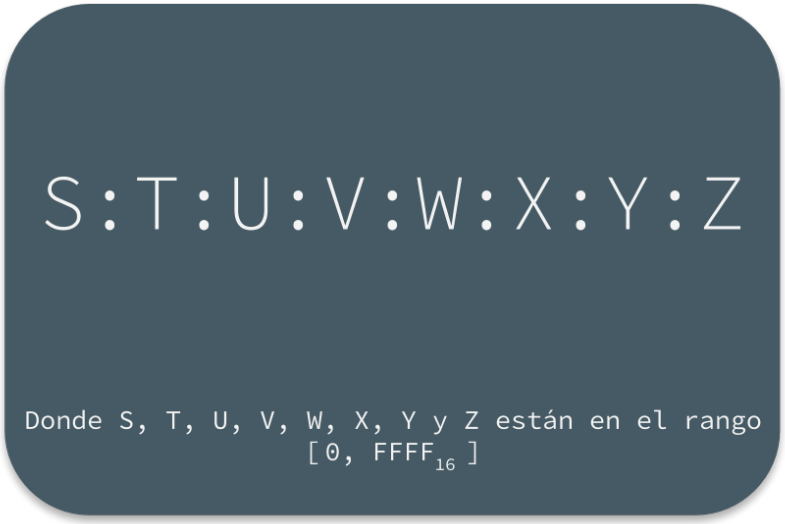
Ahora, recuerda que te dige que IP v4 acepta unos 4 mil millones de direcciones válidas, ahora el problema es que ahora mismo hay vivos mas de 7 mil millones de personas (A principios del siglo XXI) cada una con seguramente más de un dispositivo que quieran conectar a internet.

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

2.3. Dirección IPv6

Como vimos antes, ahora que parece que la cantidad de direcciones IPv4 se nos esta quedando corta, poco a poco estamos pasando de IPv4 a IPv6 que contará con nada menos y nada mas que **128 bits** para una dirección, es decir nos permitirá tener unas más o menos: $340,282,366,920,938,463,463,374,607,431,768,211,456$ posibles direcciones IP. Un chingo.

La convención es que estos serían representados como 8 conjuntos de 65535 bits representados en hexadecimal (porque de otra manera sale un númerote), es decir:



Por ejemplo una IPv6 podría ser $2001:0DB8:0000:0042:0000:8A2E:0370:7334$

2.4. Haciendo un poco más faciles las Direcciones IPv6

Ahora, todo esta mucho mejor que con IPv4, pero tenemos un pequeño problema, sus direcciones son moustrosamente enormes, por lo que tuvimos que hacer algunas simplificaciones para los humanos:

- Ignora los ceros dentro de cada grupo de 4 dígitos hexadecimales:

  Ejemplo:

  De $2001:0DB8:0000:0042:0000:8A2E:0370:7334$
  a $2001:0DB8:0:42:0:8A2E:370:7334$

- Si tienes un montón de ceros pon :: y da por sentado que quien lee esta dirección tiene cerebro y puede entender que ahí van ceros:

  Ejemplo:

  De $2001:0DB8:0000:0042:0000:0000:0000:0000$
  a $2001:0DB8:0000:0042::$

2.5. Direcciones Especiales

- **Ausencia de Dirección:** 0.0.0.0
- **Broadcast:** 255.255.255.255
- **Loopback (LocalHost):** 127.0.0.0

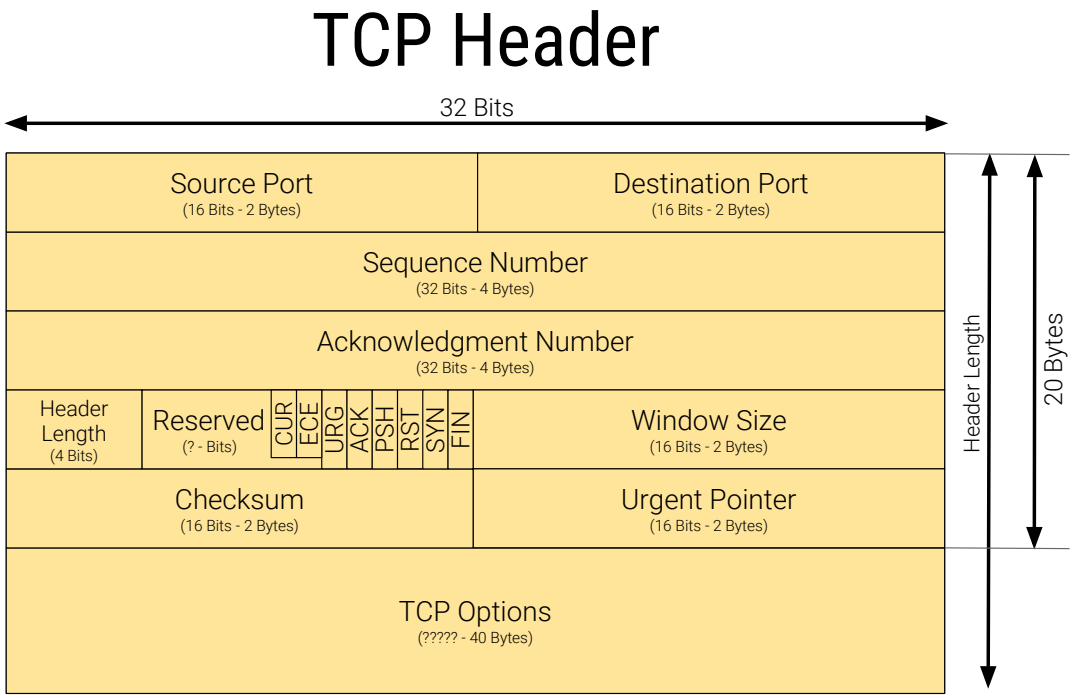
  Esta dirección nunca saldrá de nuestra máquina
- **Direcciones Privadas (RFC 1918):**

  Aquí ya se complican las cosas, estas direcciones estan reservadas:

  - Para la Clase A tenemos el rango 10.0.0.0 a 10.255.255.255
  - Para la Clase B tenemos el rango 172.16.0.0 a 172.31.255.255
  - Para la Clase B tenemos el rango 192.168.0.0 a 192.168.255.255

3. Protocolo TCP

3.1. Header - Encabezado



4. 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

5. Capa de Red

5.1. Definición

En esta capa se lleva a cabo el direccionamiento lógico que tiene carácter jerárquico, se selecciona la mejor ruta hacia el destino mediante el uso de tblas de enrutamiento a través del uso de protocolos de enrutamiento o por direccionamiento estático.

Protocolos de Capa de Red son por ejemplo: IP, IPX, RIP, IGRP, Apple Talk.

5.2. Funciones de la capa de red

La capa de red define cómo transportar el tráfico de datos entre dispositivos que no están conectados localmente en el mismo dominio de difusión, es decir, que pertenecen a diferentes redes.

Para conseguir esta comunicación se necesita conocer las direcciones lógicas asociadas a cada puesto de origen y de destino y una ruta bien definida a través de la red para alcanzar el destino deseado. La capa de red es independiente de la de enlace de datos y, por tanto, puede ser utilizada para conectividad de medios fisicos diferentes.

6. Capa de Transporte

6.1. Definición

Es la encargada de la comunicación confiable entre host, control de flujo y de la corrección de errores entre otras cosas. Los datos son divididos en segmentos identificados con un encabezado con un número de puerto que identifica la aplicación de origen. En esta capa funcionan protocolos como UDP y TCP, siendo este último uno de los más utilizados debido a su estabilidad y confiabilidad.

6.2. Funciones de la capa de red

Para conectar dos dispositivos remotos es necesario establecer una conexión. La capa de transporte establece las relgas para esta interconexión. Permite que las estaciones finales ensamblen y reensamblen múltiples segmentos del mismo flujo de datos. Esto se hace por medio de identificadores que en TCP/IP reciben el nombre de número de puerto. La capa cuatro permite además que las aplicaciones soliciten transporte fiable entre los sistemas. Asegura que los segmentos distribuidos serán confirmados al remitente. Coloca de nuevo los segmentos en su orden correcto en el receptor. Proporciona control de flujo regulando el tráfico de datos.

En la capa de transporte, los datos pueden ser transmitidos de forma fiable o no fiable. Para IP, el protocolo TCP (Protocolo de control de transporte) es fiable u orientado a conexión con un saludo previo de tres vías, mientras que UDP (Protocolo de datagrama de usuario) no es fiable, o no orientado a conexión, donde solo se establece un saludo de dos vías antes de enviar los datos.

TCP utiliza una técnica llamada ventanas, donde se establece la cantidad de envío de paquetes antes de transmitir; mientras que en el windowing o de ventana deslizante, el flujo de envío de datos es negociado dinámicamente entre el emisor y receptor. En las ventanas deslizantes o windowing cada acuse de recibo (ACK) confirma la recepción y el envío siguiente.

7. Proceso de Encapsulación de Datos

El proceso desde que los datos son incorporados al ordenador hasta que se transmiten al medio se llama encapsulación. Estos datos son formateados, segmentados, identificados con el direccionamiento lógico y fisico para finalmente ser enviados al medio. A cada capa del modelo OSI le corresponde una **PDU** (Unidad de datos) siguiendo por lo tanto el siguiente orden de encapsulamiento:

1. Datos
2. Segmentos
3. Paquetes
4. Tramas
5. Bits

Debido a que posiblemente la cantidad de los datos sea deamasiada, la capa de transporte desde el origen se encarga de segmentarlos par así ser empaquetados debidamente, esta misma capa en el destino se encargará de reensamblar los datos y colocarlos en forma secuencial, ya que no siempre llegan a su destino en el orden en que han sido segmentados, así mismo acorde al protocolo que se esté utilizando habrá o no corrección de errores. Estos segmentos son empaquetados (paquetes o datagramas) e identificados en la capa de red con la dirección lógica o IP correspondiente al origen y destino. Ocurre lo mismo con la dirección MAC en la capa de enlace de datos formándose las tramas o frames para ser transmitidos a través de alguna interfaz. Finalmente las tramas son enviadas al medio desde la capa física.

El proceso inverso se realiza en el destino y se llama desencapsulación de datos.

8. Practica en Si

8.1. Ideas

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 verifica posibles errores y dar por buena la trama.

El procedimiento que seguimos fue el siguiente:

- Calcular la longitud del encabezado IP.
- Crear un nuevo arreglo de bytes para almacenar el encabezado.
- Identificar IP de origen e IP destino.
- 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:

- Calculamos la longitud del encabezado TCP.
- Creamos un nuevo arreglo que almacenaba el encabezado TCP.
- Creamos un arreglo que iba a contener la información correspondiente al pseudo-header.

- Agregamos la IP de origen y destino al pseudo-header, en ese orden.
- 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.
- Calculamos la longitud del payload utilizando la longitud total menos la longitud del encabezado IP menos 14 que es la longitud del encabezado TCP.
- Agregamos la información del payload a los bytes 11 y 12.
- Creamos un nuevo arreglo que contendrá el payload de TCP y copiamos la información.
- Creamos un arreglo auxiliar que contendrá: el pseudo-header, el encabezado TCP y el payload.
- Calculamos el checksum utilizando el arreglo auxiliar.

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

- Creamos un arreglo que contendrá el encabezado UDP y copiamos la información.
- Creamos un arreglo de 12 bytes que nos servirá para crear el pseudo-header.
- Copiamos el IP origen y el IP destino al pseudo-header.
- 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.
- Copiamos el encabezado UDP al pseudo-header.
- Calculamos la longitud del payload: longitud de la trama menos longitud de IP menos longitud de UDP menos 14.
- Creamos un arreglo de bytes que contendrá el payload y copiamos la información de la trama.
- Creamos un arreglo temporal de bytes para realizar el cálculo del checksum. Agregamos: pseudo-header, encabezado UDP y el payload, en ese orden.
- Realizamos el cálculo del checksum.

8.2. Evidencias

Primeramente el código:

```
1  public class Checksum {
2
3    /**
4     * Calculate the Internet Checksum of a buffer (RFC 1071 - http://www.faqs.org/rfcs/rfc1071.html)
5     * Algorithm is
6     * 1) apply a 16-bit 1's complement sum over all octets (adjacent 8-bit pairs [A,B], final odd
        length is [A,0])
7     * 2) apply 1's complement to this final sum
8     *
9     * Notes:
10    * 1's complement is bitwise NOT of positive value.
11    * Ensure that any carry bits are added back to avoid off-by-one errors
12    *
13    *
14    * @param buf The message
15    * @return The checksum
16    */
17   public static long calculateChecksum(byte[] buf) {
18     int length = buf.length;
19     int i = 0;
20
21     long sum = 0;
22     long data;
23
24     // Handle all pairs
25     while (length > 1) {
26       // Corrected to include @Andy's edits and various comments on Stack Overflow
27       data = (((buf[i] << 8) & 0xFF00) | ((buf[i + 1]) & 0xFF));
28       sum += data;
29       // 1's complement carry bit correction in 16-bits (detecting sign extension)
30       if ((sum & 0xFFFF0000) > 0) {
31         sum = sum & 0xFFFF;
32         sum += 1;
33       }
34
35       i += 2;
36       length -= 2;
37     }
38
39     // Handle remaining byte in odd length buffers
40     if (length > 0) {
41       // Corrected to include @Andy's edits and various comments on Stack Overflow
42       sum += (buf[i] << 8 & 0xFF00);
43       // 1's complement carry bit correction in 16-bits (detecting sign extension)
44       if ((sum & 0xFFFF0000) > 0) {
45         sum = sum & 0xFFFF;
46         sum += 1;
47       }
48     }
49
50     // Final 1's complement value correction to 16-bits
51     sum = ~sum;
52     sum = sum & 0xFFFF;
53     return sum;
54
55   }
56
57     public static void main(String[] args){
58         byte[] buf = {
59                         (byte) 0x45,
60                         (byte) 0x00,
61                         (byte) 0x00,
62                         (byte) 0x3c,
63                         (byte) 0x0a,
64                         (byte) 0x1c,
65                         (byte) 0x40,
66                         (byte) 0x00,
67                         (byte) 0xff,
68                         (byte) 0x06,
69                         (byte) 0x00,
70                         (byte) 0x00,
71                         (byte) 0xa8,
72                         (byte) 0xb0,
73                         (byte) 0x03,
74                         (byte) 0x19,
75                         (byte) 0xa8,
76                         (byte) 0xb0,
77                         (byte) 0x03,
78                         (byte) 0x6c
79                     };
80
81         long resultado = Checksum.calculateChecksum(buf);
```

```
        System.out.printf("Valos del check: %02X\n",resultado);
    }//main

}
```

```


/*=============================================================
=================          IMPORTS          ===================
=============================================================*/

import java.util.*;
import java.io.*;
import org.jnetpcap.Pcap;
import org.jnetpcap.PcapIf;
import org.jnetpcap.packet.PcapPacket;
import org.jnetpcap.packet.PcapPacketHandler;
import org.jnetpcap.PcapBpfProgram;
import org.jnetpcap.protocol.lan.Ethernet;
import org.jnetpcap.protocol.tcpip.*;
import org.jnetpcap.protocol.network.*;
import org.jnetpcap.nio.JBuffer;
import org.jnetpcap.packet.Payload;
import org.jnetpcap.protocol.network.Arp;
import org.jnetpcap.protocol.lan.IEEE802dot2;
import org.jnetpcap.protocol.lan.IEEE802dot3;



/*=============================================================
=================          CHECK SUM          ===================
=============================================================*/
public class Checksumv2 {


    /*=========================================================
    ==============    CALCULATE CHECK SUM    ==================
    =========================================================*/
    public static long CalculateChecksum (byte[] Data) {

        int length = Data.length;
        int i = 0;

        long sum = 0;
        long data;

        while (length > 1) {

            data = (((Data[i] << 8) & 0xFF00) | ((Data[i + 1]) & 0xFF));
            sum += data;

            if ((sum & 0xFFFF0000) > 0) {
                sum = sum & 0xFFFF;
                sum += 1;
            }

            i += 2;
            length -= 2;
        }

        if (length > 0) {
            sum += (Data[i] << 8 & 0xFF00);

            if ((sum & 0xFFFF0000) > 0) {
                sum = sum & 0xFFFF;
                sum += 1;
            }
        }

        sum = ~sum;
        sum = sum & 0xFFFF;

        return sum;

    }


    /*=========================================================
    ==========    FROM BYTE[] => MAC STRING    ================
    =========================================================*/
    private static String MACAsString (final byte[] MAcAddress) {

        final StringBuilder Data = new StringBuilder();

        for (byte Byte: MAcAddress) {

            if (Data.length() != 0) Data.append (':');
```

```
82
83              if (Byte >= 0 && Byte < 16) Data.append ('0');
84
85              Data.append (Integer.toHexString((Byte < 0) ? Byte + 256 : Byte).toUpperCase());
86          }
87
88          return Data.toString();
89
90      }
91
92
93      /*================================================
94      ==========    SELECT DEVICE FROM CONSOLE    ==========
95      ================================================*/
96      static PcapIf SelectDevicesByConsole() {
97
98          int DEFAULT_DEVICE = 10;
99
100         List <PcapIf> AllDevs = new ArrayList <PcapIf>();
101         StringBuilder ErrorData = new StringBuilder();
102         PcapIf SelectedDevice;
103
104         int Result = Pcap.findAllDevs(AllDevs, ErrorData);
105
106         if (Result == Pcap.NOT_OK || AllDevs.isEmpty()) {
107
108             System.err.printf ("Can't read list of Devices, error is %s",
109                 ErrorData.toString());
110
111             return null;
112         }
113
114         System.out.println("================================");
115         System.out.println("==  Network Devices found ==");
116         System.out.println("================================");
117
118         try {
119
120             int i = 0;
121             for (PcapIf Device: AllDevs) {
122
123                 String Description = "No Description available";
124                 if (Device.getDescription() != null)
125                     Description = Device.getDescription();
126                 Description = "";
127
128                 final byte[] MAcAddress = Device.getHardwareAddress();
129
130                 String StrMAcAddress = (MAcAddress != null)?
131                     MACAsString(MAcAddress) : "No MAC Address";
132
133                 System.out.printf("  #%d: Name [%s] ", i, Device.getName());
134                 System.out.printf("MAC:[%s]\n", StrMAcAddress);
135
136                 i++;
137
138             }
139
140             SelectedDevice = AllDevs.get(DEFAULT_DEVICE);
141
142             String InfoSelected = (SelectedDevice.getDescription() != null)?
143                 SelectedDevice.getDescription() : SelectedDevice.getName();
144
145             System.out.printf("\n\nChoosing '%s':\n\n\n", InfoSelected);
146
147         }
148         catch (IOException e) {
149             e.printStackTrace ();
150         }
151
152         SelectedDevice = AllDevs.get(DEFAULT_DEVICE);
153
154
155         return SelectedDevice;
156
157     }
158
159
160
161
162     /*================================================================
163     ==========                  MAIN                  ==========
164     ================================================================*/
165     public static void main (String[] Args) {
166
167         StringBuilder ErrorData = new StringBuilder();
168         PcapIf Device = SelectDevicesByConsole();
169
```

```
170         if (Device == null) return;
171 
172         /*=============================================
173         ==========    START THE PCAP    ==========
174         =============================================
175         Remember:
176             - SnapshotLength
177                 Refers to the amount of actual data captured
178                 from each packet passing through the specified network interface.
179                 64*1024 = 65536 bytes
180             */
181 
182         int SnapshotLength  = 64 * 1024;               // Capture all packets, no trucation
183         int Flags           = Pcap.MODE_PROMISCUOUS;   // capture all packets
184         int Timeout         = 10 * 1000;               // 10 seconds in millis
185 
186         Pcap PcapInstance = Pcap.openLive(
187             Device.getName(),
188             SnapshotLength,
189             Flags,
190             Timeout,
191             ErrorData
192         );
193 
194         if (PcapInstance == null) {
195             System.err.printf("Error while opening device: " + ErrorData.toString());
196             return;
197         }
198 
199 
200         /*=============================================
201         ==========        FILTER        ==========
202         =============================================*/
203         PcapBpfProgram Filter = new PcapBpfProgram();
204 
205         String Expression = "";             // "port 80";
206         int Optimize      = 0;              // 1 means true, 0 means false
207         int Netmask       = 0;              // Netmask value
208 
209         int Result = PcapInstance.compile(
210             Filter,
211             Expression,
212             Optimize,
213             Netmask
214         );
215 
216         if (Result != Pcap.OK)
217             System.out.println("Filter error: " + PcapInstance.getErr());
218 
219         PcapInstance.setFilter(Filter);
220 
221 
222 
223         /*=============================================
224         ======    CREATE PACKET HANDLER    ======
225         =============================================*/
226         PcapPacketHandler<String> JPacketHandler = new PcapPacketHandler<String>() {
227 
228             public void nextPacket(PcapPacket Packet, String User) {
229 
230                 /*=============================================
231                 ======          SHOW INFO          ======
232                 =============================================*/
233 
234                 System.out.println("=============================================================");
235                 System.out.println("==============    PACKET    ==============================");
236                 System.out.println("=============================================================\n\n");
237 
238                 System.out.printf("\nReceived at %s",new
    Date(Packet.getCaptureHeader().timestampInMillis()));
239                 System.out.printf("\nCapture Length = %-4d", Packet.getCaptureHeader().caplen());
240                 System.out.printf("\nOriginal Sizeh = %-4d", Packet.getCaptureHeader().wirelen());
241                 System.out.printf("\nUSer           = %s\n\n\n", User);
242 
243                 String MACAddressOrigin  = "";
244                 String MACAddressDestiny = "";
245 
246                 /*=============================================
247                 ====    SHOW RAW DATA & GET MAC'S    ====
248                 =============================================*/
249                 for (int i = 0; i < Packet.size(); i++) {
250 
251                     System.out.printf("%02X ", Packet.getUByte(i));
252 
253                     if (i % 16 == 15) System.out.println("");
254 
255                     if (i < 6)
256                         MACAddressOrigin += String.format("%02X ", Packet.getUByte(i));
```

```
257                     else if (i < 12)
258                         MACAddressDestiny += String.format("%02X ", Packet.getUByte(i));
259 
260                 }
261                 System.out.println("\n\n\n\n");
262 
263                 /*============================================
264                 ====         SHOW MAC'S & TYPE            ====
265                 ============================================*/
266                 int Type = Packet.getUByte(12) * 256 + Packet.getUByte(13);
267 
268                 System.out.println("MAC Origin  = " + MACAddressOrigin);
269                 System.out.println("MAC Destiny = " + MACAddressDestiny);
270                 System.out.printf("Type        = %04X\n", Type);
271 
272 
273                 /*============================================
274                 ====         IS AN IP PACKET?             ====
275                 ============================================*/
276                 if ((Type & 0xFFFF) == 0x0800) {
277 
278                     System.out.println("\nIs an IPv4 Packet!");
279 
280                     byte[] PacketAsByteArray = Packet.getByteArray(0, Packet.size());
281 
282                     int IPPacketSize = (PacketAsByteArray[14] & 0x0F) * 4;
283                     byte[] IPHeader = new byte[IPPacketSize];
284                     System.arraycopy(PacketAsByteArray, 14, IPHeader, 0, IPPacketSize);
285 
286                     //IPHeader[10] = 0x00;
287                     //IPHeader[11] = 0x00;
288 
289                     int IPacketSize = (((IPHeader[2] << 8) & 0xFF00) | ((IPHeader[3]) & 0xFF));
290 
291                     System.out.printf("Complemnt to 1 of Checksum IPv4: ");
292                     System.out.printf("%04X\n", CalculateChecksum(IPHeader));
293 
294 
295                     if (Packet.size() > (13 + IPPacketSize)) {
296 
297                         /*============================================
298                         ====         IS AN TCP PACKET?            ====
299                         ============================================*/
300                         if (IPHeader[9] == 0x06) {
301 
302                             System.out.println("\n\nIs an TCP Packet!");
303 
304                             byte[] TCPHeader = new byte[12];
305 
306                             for (int i = 0; i < 8; i++)
307                                 TCPHeader[i] = IPHeader[IPPacketSize - 8 + i];
308 
309                             int TCPPacketSize = IPacketSize - IPPacketSize;
310 
311                             TCPHeader[8]  = 0x00;
312                             TCPHeader[9]  = 0x06;
313                             TCPHeader[10] = (byte)(TCPPacketSize & 0x0000FF00);
314                             TCPHeader[11] = (byte)(TCPPacketSize & 0x000000FF);
315 
316                             byte[] TCPPacket = new byte[TCPPacketSize + 12];
317                             System.arraycopy(TCPHeader, 0, TCPPacket, 0, 12);
318                             System.arraycopy(PacketAsByteArray, IPPacketSize + 14, TCPPacket, 12,
    TCPPacketSize);
319 
320                             //TCPPacket[28] = 0x00;
321                             //TCPPacket[29] = 0x00;
322 
323                             System.out.printf("Complemnt to 1 of Checksum TCP: ");
324                             System.out.printf("%04X\n", CalculateChecksum(TCPPacket));
325                         }
326 
327                         /*============================================
328                         ====         IS AN UDP PACKET?            ====
329                         ============================================*/
330                         if (IPHeader[9] == 0x11) {
331 
332                             System.out.println("\n\nIs an UDP Packet!");
333 
334                             byte[] UDPHeader = new byte[12];
335 
336                             for (int i = 0; i < 8; i++)
337                                 UDPHeader[i] = IPHeader[IPPacketSize - 8 + i];
338 
339                             int UDPPacketSize = IPacketSize - IPPacketSize;
340 
341                             UDPHeader[8]  = 0x00;
342                             UDPHeader[9]  = 0x11;
343                             UDPHeader[10] = (byte)(UDPPacketSize & 0x0000FF00);
```

```
344                             UDPHeader[11] = (byte)(UDPPacketSize & 0x000000FF);
345
346                             byte[] UDPPacket = new byte[UDPPacketSize + 12];
347                             System.arraycopy(UDPHeader, 0, UDPPacket, 0, 12);
348                             System.arraycopy(PacketAsByteArray, IPPacketSize + 14, UDPPacket, 12,
    UDPPacketSize);
349
350                             //UDPPacket[18] = 0x00;
351                             //UDPPacket[19] = 0x00;
352
353                             System.out.printf("Complemnt to 1 of Checksum UDP: ");
354                             System.out.printf("%04X\n", CalculateChecksum(UDPPacket));
355                         }
356                     }
357                 }
358                 else
359                     System.out.println("\nNot an IPv4 Packet");
360
361                 System.out.println ("\n\nRaw Data: \n" + Packet.toHexdump ());
362             }
363
364         };
365
366
367
368         /*=============================================
369         =====            DO A LOOP              =====
370         =============================================
371
372             Remember:
373                 Fourth we enter the loop and tell it to capture 5 packets. The loop
374                 method does a mapping of pcap.datalink() DLT value to JProtocol ID, which
375                 is needed by JScanner. The scanner scans the packet Datafer and decodes
376                 the headers. The mapping is done automatically, although a variation on
377                 the loop method exists that allows the programmer to sepecify exactly
378                 which protocol ID to use as the data link type for this pcap interface.
379         */
380         PcapInstance.loop(5, JPacketHandler, "In a Loop");
381
382         PcapInstance.close();
383     }
384
385 }
```

Luego ejecutarlo nos dará lo siguiente:

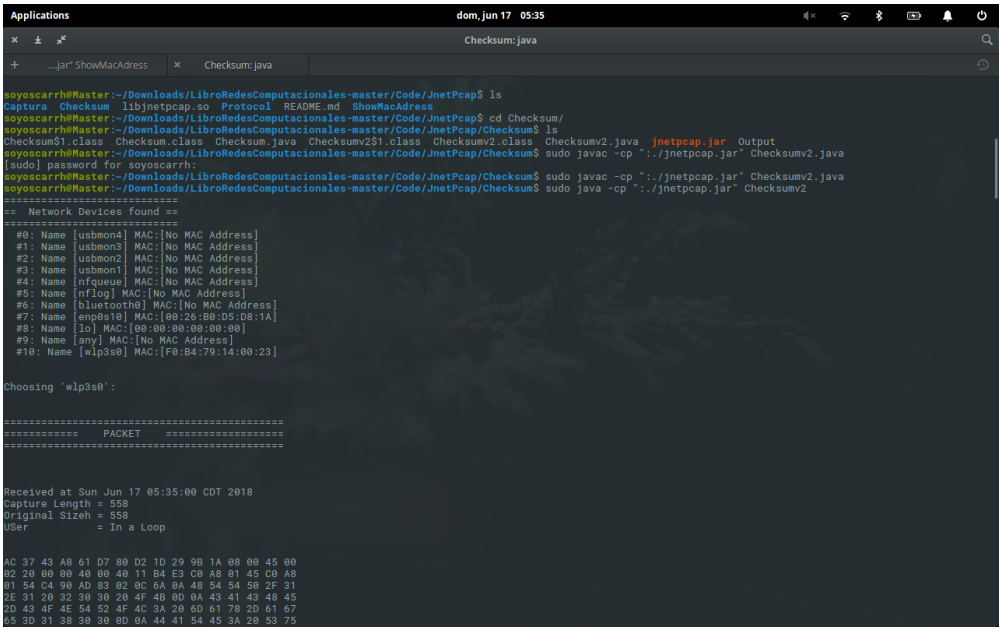


Figura 1: Ejemplo

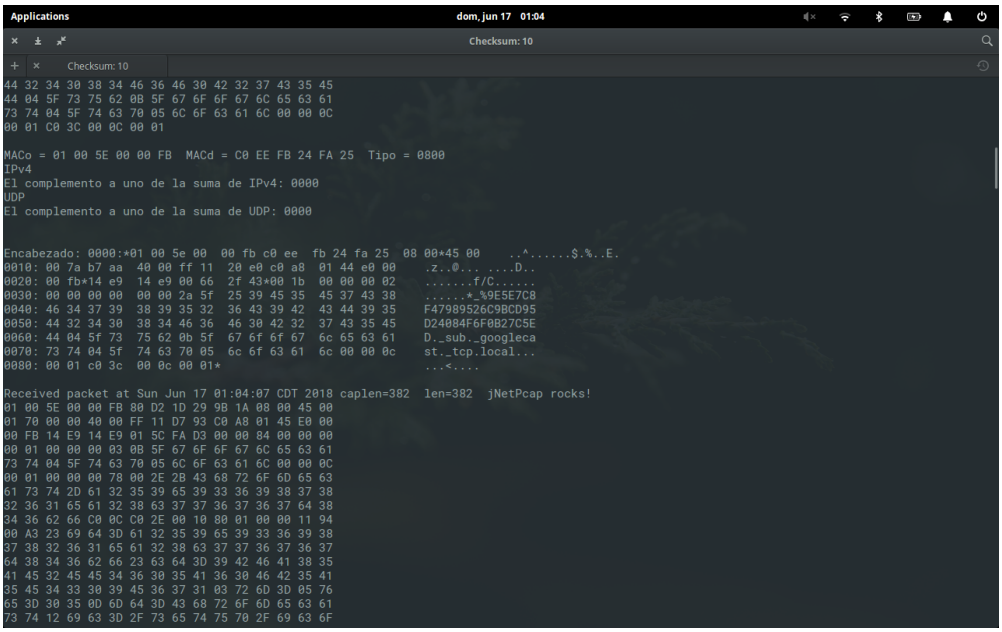


Figura 2: Ejemplo

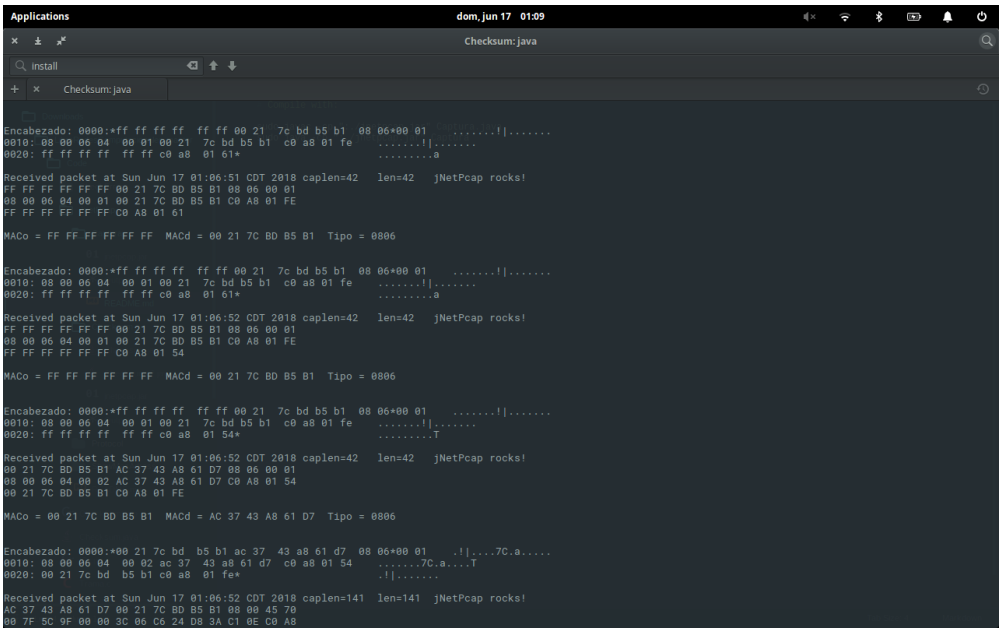


Figura 3: Ejemplo

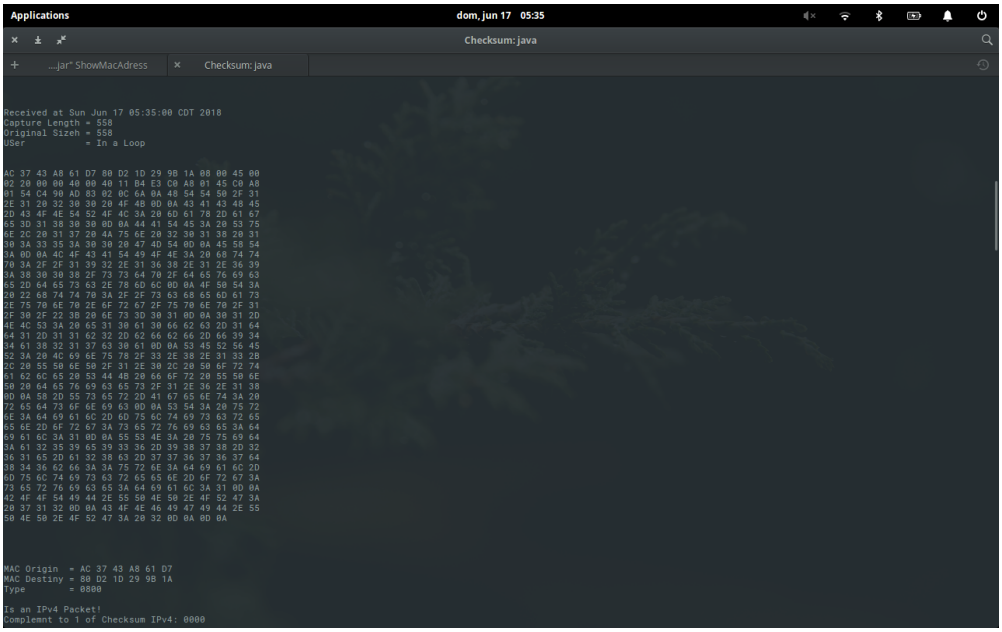


Figura 4: Ejemplo

9. Conclusiones

- Oscar Andrés Rosas Hernandez

  Al finalizar está práctica pude ver como funciona el algoritmo de Checksum y su implementación a un lenguaje de programación

  Su funcionamiento ayuda a ver los errores que pueden tener las tramas. También pude ver como trabajan las capas de red y de transporte.

  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.

- Arturo Rivas Rojas

  La idea en la que se basa la suma de chequeo de Internet es muy sencilla: se suman todas las palabras de 16 bits que conforman el mensaje y se transmite, junto con el mensaje, el resultado de dicha suma (este resultado recibe el nombre de checksum). Al llegar el mensaje a su destino, el receptor realiza el mismo cálculo sobre los datos recibidos y compara el resultado con el checksum recibido. Si cualquiera de los datos transmitidos, incluyendo el mismo checksum, esta corrupto, el resultado no concordará y el receptor sabrá que ha ocurrido un error.

  El checksum se realiza de la siguiente manera: los datos que serán procesados (el mensaje) son acomodados como una secuencias de enteros de 16 bits. Estos enteros se suman utilizando aritmética complemento a uno para 16 bits y, para generar el checksum, se toma el complemento a uno para 16 bits del resultado.

  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.

Referencias

[1] E. Ariganello, Redes Cisco. Guia de estudio para la certificacion CCNA 640-802, 2da Edicion, 201