INSTITUTO POLITÉCNICO NACIONAL

Escuela Superior de Computo

Analizador de tramas. Versión IP Completo

Redes de Computadoras

Autores:
Hernández Vergara, Eduardo
Rojas Cruz, José Ángel

Profesora:
M. en C. Nidia Asunción Cortez Duarte

3 de Junio de 2022

Índice

1. Introdución

1.1. El modelo OSI y la arquitectura TCP/IP

El modelo OSI nace de la necesidad de tener un estándar para la comunicación en redes amplias o sea equipos de distintos fabricantes, así como facilitar economías a gran escala y debido a la complejidad que implican estas comunicaciones de proporciones bíblicas, por ello las distintas funcionalidades se dividen en partes mas manejables, por ello el modelo OSI fue muy aceptada para estructurar problemas, y fue adoptada por el ISO. En esta técnica, las funciones de comunicación se distribuyen en un conjunto jerárquico de capas. Cada capa realiza un subconjunto de tareas relacionadas entre sí, de entre las necesarias para llegar a comunicarse con otros sistemas. Por otra parte, cada capa se sustenta en la capa inmediatamente inferior, la cual realizará funciones más primitivas, ocultando los detalles a las capas superiores. Una capa proporciona servicios a la capa inmediatamente superior. Idealmente, las capas deberían estar definidas para que los cambios en una capa no implicaran cambios en las otras capas. De esta forma, el problema se descompone en varios subproblemas más abordables.

La arquitectura de protocolos TCP/IP es resultado de la investigación y desarrollo llevados a cabo en la red experimental de conmutación de paquetes ARPANET, financiada por la Agencia de Proyectos de Investigación Avanzada para la Defensa (DARPA, Defense Advanced Research Projects Agency), y se denomina globalmente como la familia de protocolos TCP/IP. Esta familia consiste en una extensa colección de protocolos que se han especificado como estándares de Internet por parte de IAB (Internet Architecture Board).

Figura 2: Comparación entre las arquitecturas de protocolos TCP/IP y OSI.

1.1.1. Encapsulado del modelo OSI

Veremos cual es el proceso de encapsulamiento del modelo OSI:

CAPA 7. La información comienza con el nombre de datos en la Capa de Aplicación en el lado del emisor

CAPA 6. Conforme los datos se mueven a la Capa de Presentación es codificado o comprimido a un formato estándar a veces encriptado. Después los datos del usuario son convertidos a un formato estándar común

CAPA 5. Después se mueve a la Capa de Sesión. En esta capa, un ID de sesión se agrega a los datos. (Hasta este punto los datos todavía tienen su estructura original)

CAPA 4. Ahora los datos pasan a la Capa de Transporte en la capa de transporte los datos son fraccionados en diferentes y más pequeños bloques o piezas. A cada bloque se le agrega una cabecera, que contiene:

1. Puertos de destino y Origen
2. Números de secuencia y otra información
3. Todos en conjunto crean un nuevo PDU
4. El nuevo PDU se llama Segmento si TCP es el protocolo utilizado, o lo llamaremos Datagrama si se trata de UDP.

CAPA 3. Cuando el segmento viaja a la Capa de Red una nueva cabevera de IP se agrega. Esta cabecera de IP contiene la dirección IP origen y la direccion IP destino y otra información

CAPA 2. Cuando el paquete pasa a Capa de Enlace; se repite el proceso y se agrega una cabecera y un bloque llamadao FCS al final del paquete en esta capa un nuevo PDU llamado frame o trama es creado. La cabecera de la trama contiene la dirección MAC de origen y destino y como otro control de información. FCS marca el final de la trama y también se usa para verificar

CAPA 1. El frame ahora es enviado a la Capa física. En esta capa física solo se consideran los 1 y 0's que representan a la trama es por eso que a este tipo de PDU normalmente se le llaman los Bits. Los 1 y 0's entonces están listos para ser convertidos en cualquier tipo de señal ya sea electrica, ondas de radio, luz, etc.

Ahora realizaremos el proceso de desencapsulamiento:

CAPA 1 y CAPA 2.- La Capa Física recibe bits y los manda a la Capa de Enlace de datos donde son interpretados como una trama y se Verifica su cabercera e información adicional añadida, si la MAC address tiene correspondencia y no se encuentran errores entronces la trama es descartada y el paquete IP extraido y entregado a la capa de red.

CAPA 3.- En la Capa de Red la cabecera IP es verificada y si esta IP concuerda entonces la cabercera IP es eliminada del paquete IP.

CAPA 4.- Ahora el segmento pasa a la Capa de Transporte donde se examina esta información, se busca el numero de puerto.

CAPA 5.- La información es transferida a la Capa de Sesión considerando la aplicación que le corresponde Según el numero de puerto. En este punto un ID de sesión se ocupa.

CAPA 6 Y 7.- Los datos pasan ahora a la Capa de Presentación cualquier encriptado será removido y el datos será recuperado a su forma original que entonces será presentada a la Capa de Aplicación.

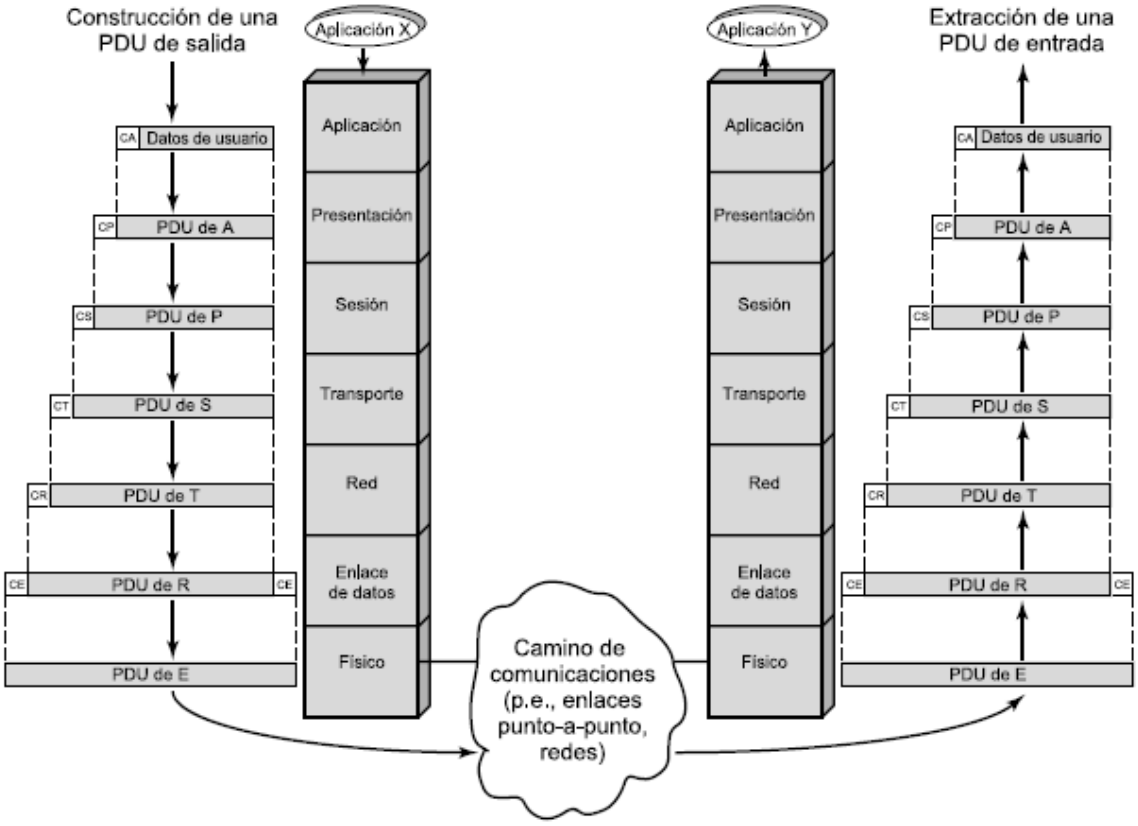

Figura 3: El entorno OSI

1.2. Cabecera Ethernet

Para poder hablar propiamente de la cabecera ethernet, tenemos que hablar de las cabeceras de paquete de interfaz de red.

En esta capa de iterfaz de red, se adjuntan cabeceras de paquete a los datos de salida, los paquetes se tienen que enviar a tráves del adaptador de red a la red apropiada. Los paquetes pasan por muchas pasarelas antes de alcanzar los destinos. En la red de destino, las cabeceras se separan de los paquetes y se envían los datos al sistema principal apropiado.

Pasando al tema principal que sería la cabecera ethernet, esta esta compuesta simplemente por 14 bytes, en donde los primeros 6 bytes es la dirección destino, los siguientes 6 bytes es la dirección origen y los 2 restantes es el tamaño o tipo, si es menor a 1500 es tamaño y nos da el mismo de la cabecera LLC, si es 0800 y 0806 en hexadecimal son IP y ARP respectivamente.

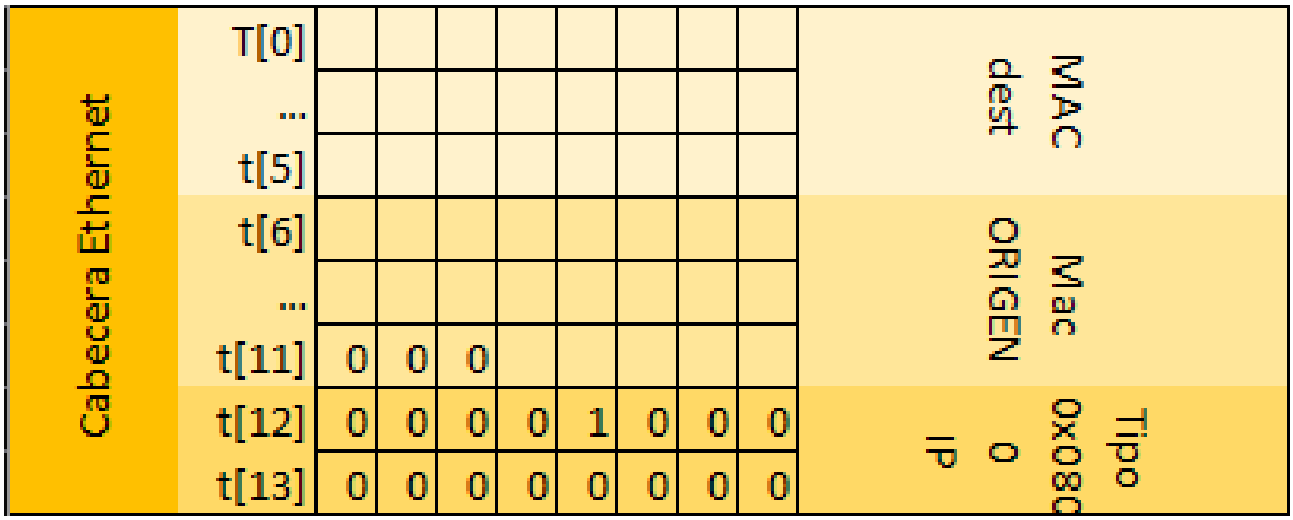

Figura 4: El mapa de memoria para la cabecera Ethernet

Listing 1: Funcion que Analiza Cabecera Ethernet

```
1  void leerTrama(unsigned char * T){
2    printf("\nCabecera ethernet \n");
3    unsigned short tot = T[12] << 8 | T[13];
4    printf("MAC DESTINO %.2x: %.2x: %.2x: %.2x: %.2x: %.2x\n", T[0], T[1],
         T[2], T[3], T[4], T[5]);
5    printf("MAC ORIGEN %.2x: %.2x: %.2x: %.2x: %.2x: %.2x\n", T[6], T[7],
        T[8], T[9], T[10], T[11]);
6    if (tot < 1500){
7        printf("Tamano de la cabecera LLC: %d bytes \n", tot);
8        analizarTrama(T);
9    }else{
10       if (tot == 2048){
11           printf("TIPO IP\n");// analiza IP
12       }else if (tot == 2054){
13           printf("TIPO ARP\n");// analiza ARP
14           analizaARP(T);
15       }else{
16           printf("TIPO: %.2x%.2x", T[12], T[13]);
17       }
18   }
19 }
```

1.3. El Protocolo IP(Internet Protocol)

El protocolo IP es el encargado de transmitir datagramas desde un host a otro, si fuera necesario, via routers intermediarios. IP proporciona un servicio de entrega que se puede describir como no fiable o el mejor posible, best-effort, porque no existe garantia de entrega. Los paquetes se pueden perder, ser duplicados, sufrir retrasos o ser entregados en un orden distinto al original, pero esos errores solo ocurren cuando los buffers en el destino están llenos. La única comprobación de errores realizada por IP es el checksum, de la cabecera, que es asequible de calcular y asegura que no se han detectado alteraciones en los datos bien de direccionamiento o bien de gestión de paquete.

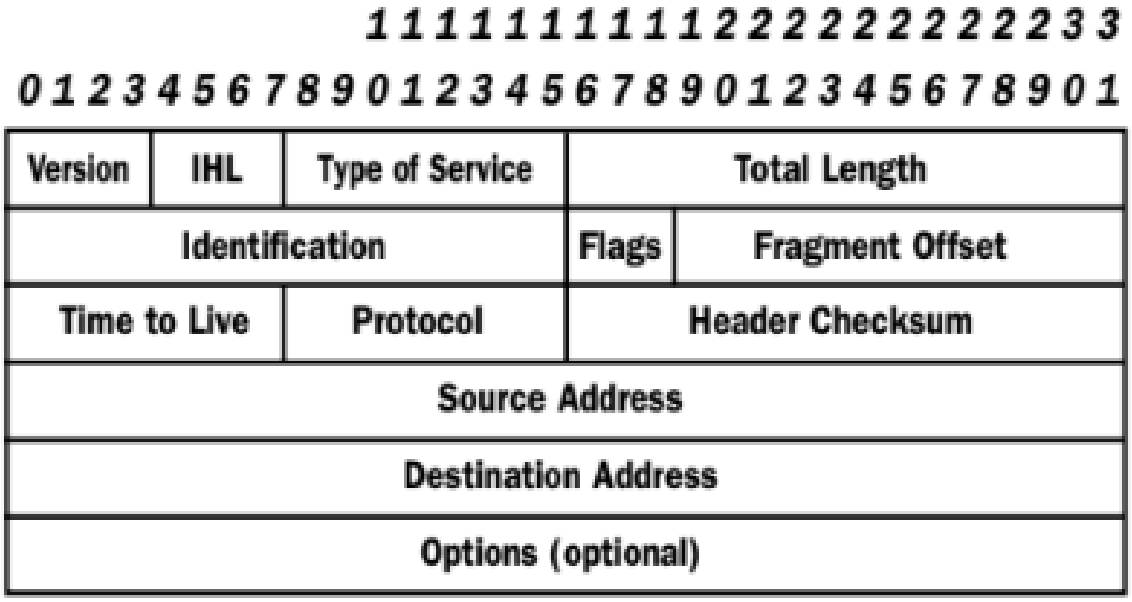

Figura 5: Son los campos de la cabecera IP

En los campos primeramente tenemos al de version, este campo nos sirve para la version del protocolo IP, ya sea IPv4 o IPv6; el segundo campo, el IHL nos sirve para obtener el tamaño de nuestra cabecera IP, que puede ser de 20 a 60 bytes (el tamaño total de mi cabecera IP), basicamente es para el campo llamado opciones si el campo es 5, es que no tiene opciones, el siguiente campo nos sirve para el tipo de servicio que es, ya sea uno de costo minimo, fiabilidad maxima, etc; despues nos da el numero de bytes en el paquete, el tamaño maximo es de 65 535 bytes, despues tenemos al identificador, que nos dice que valor de ID tiene; despues tenemos las banderas, que tiene 3 valores, x(reservado), D = 1(dont fragment), y M = 1(More Fragment), después tenemos el Fragment offset en el que tenemos que posicionarlo en el datagrama original; después tenemos el time to live que nos dice el tiempo de vida de nuestro protocolo; después tenemos lo que seria nuetro protocolo que usaremos despues, que en nuestro caso usaremos solo ICMP, TCP y UDP; ahora tenemos el checksum que mas arriba explicamos para que sirve; y solo nos quedan las direcciones IP de destino y origen junto con las Opciones si es que tiene.

El tamaño maximo de la cabecera IP es de 60 bytes, el tamaño minimo de la cabecera es de 20 bytes, y el tamaño maximo del IHL es de 15, el tamaño minimo del es de 5.

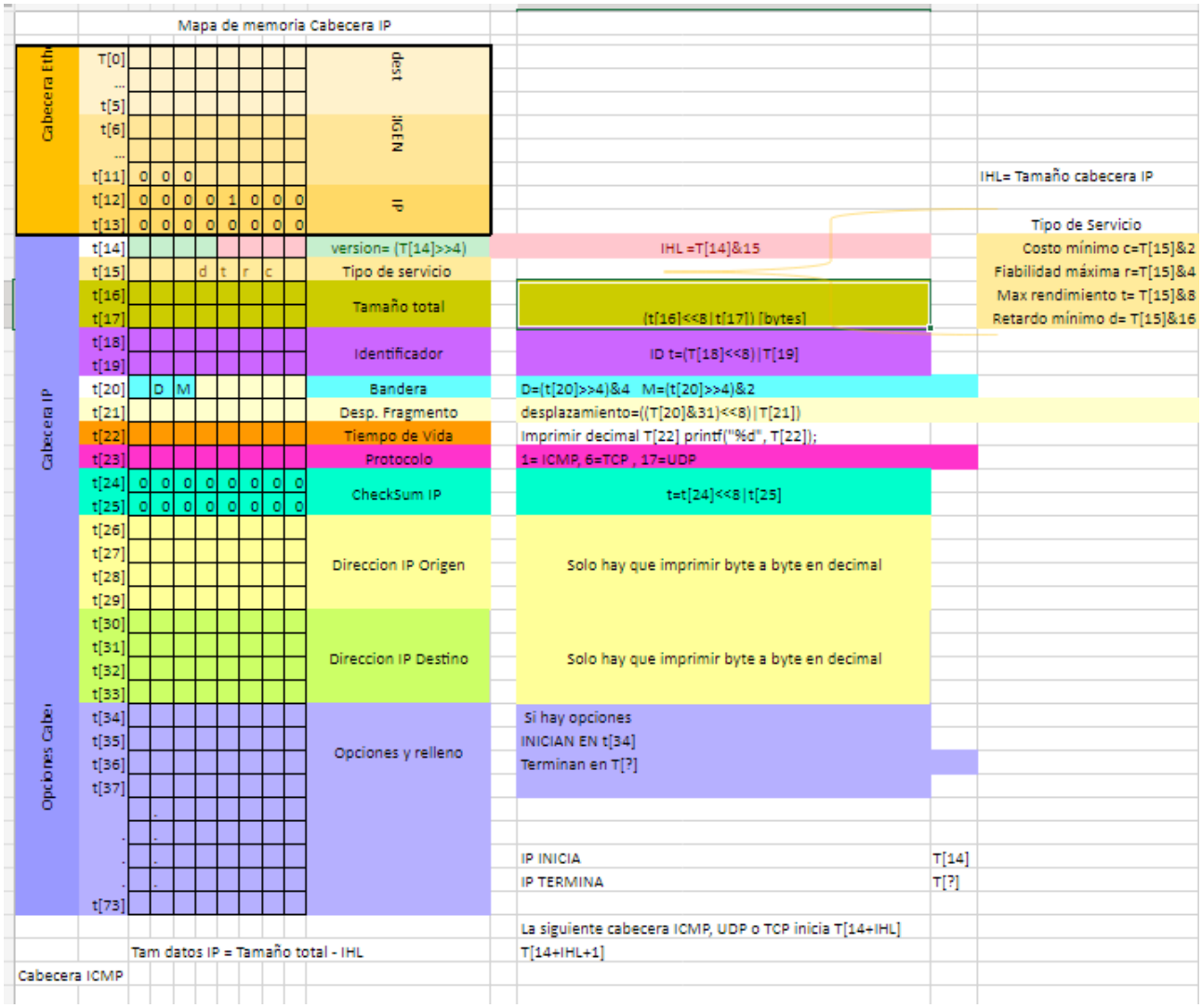

Figura 6: Es el mapa de memoria de la cabecera IP

1.4. El Protocolo ICMP

El estándar IP especifica que una implementación que cumpla las especificaciones del protocolo debe también implementar ICMP. ICMP proporciona un medio para transferir mensajes desde los dispositivos de encaminamiento y otros computadores a un computador. En esencia, ICMP proporciona información de realimentación sobre problemas del entorno de la comunicación. Algunas situaciones donde se utiliza son: cuando un datagrama no puede alcanzar su destino, cuando el dispositivo de encaminamiento no tiene la capacidad de almacenar temporalmente para reenviar el datagrama y cuando el dispositivo de encaminamiento indica a una estación que envíe el tráfico por una ruta más corta. En la mayoria de los casos, el mensaje ICMP se envía en respuesta a un datagrama, bien por un dispositivo de encaminamiento en el camino del datagrama o por el computador destino deseado.

Aunque ICMP está, a todos los efectos, en el mismo nivel que IP en el conjunto de protocolos TCP/IP, es un usuario de IP. Cuando se construye un mensaje ICMP, éste se pasa a IP, que encapsula el mensaje con una cabecera IP y después transmite el datagrama resultante de la forma habitual. Ya que los mensajes ICMP se transmiten en datagramas IP, no se garantiza su entrega y su uso no se puede considerar fiable.

En la figura 7 se nos muestra el formato de varios tipos de mensajes ICMP. Todos los mensajes de ICMP empiezan con una cabecera de 64 bits que consta de los siguientes campos:

- Tipo (1 byte): especifica el tipo de mensaje ICMP.
- Código(1 byte): se usa para especificar parámetros del mensaje que se pueden codificar en uno o unos pocos bits.
- Checksum(2 bytes): suma de comprobación del mensaje ICMP entero. Se utiliza el mismo algoritmo de suma de comprobación que en IP.
- Parámetros(4 bytes): se usa para especificar parámetros más largos.

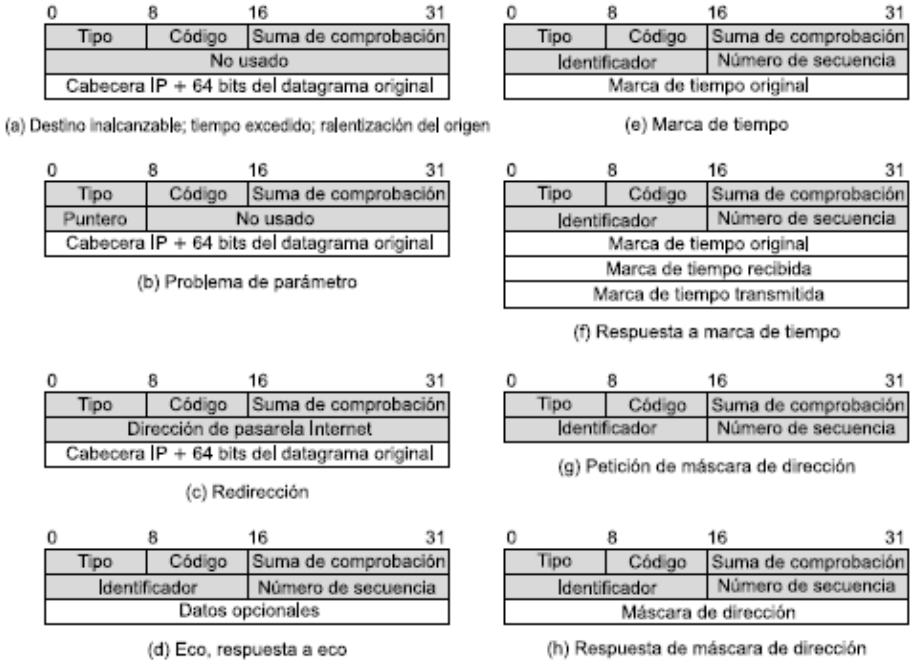

Figura 7: Formatos de mensajes ICMP

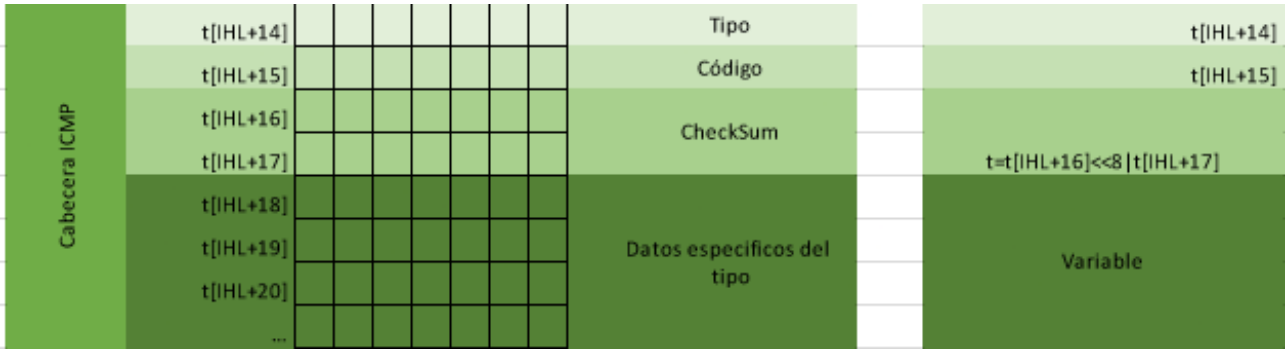

Figura 8: Es el mapa de memoria del ICMP

Figura 9: Un ICMP de tipo solicitud ECO

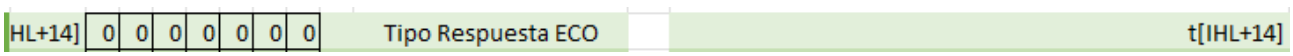

Figura 10: Un ICMP de tipo respuesta ECO

1.5. Capa de Transporte

En una arquitectura de protocolos, el protocolo de transporte se sitúa sobre la capa de red o de interconexión, que proporciona los servicios relacionados con la red, y justo debajo de las capas de aplicación y de otros protocolos de capas superiores. El protocolo de transporte proporciona servicios a los usuarios del servicio de transporte (TS, Transport Service), como FTP, SMTP y TELNET. La entidad local de transporte se comunica con alguna otra entidad de transporte remota utilizando los servicios de alguna capa inferior, como puede ser el protocolo Internet (IP). El servicio general proporcionado por un protocolo de transporte es el transporte de datos extremo a extremo, de forma que aísle al usuario TS de los detalles de los sistemas de comunicaciones subyacentes.

1.5.1. Sistemas Orientados a Conexión

Una red orientada a conexión es aquella en la que inicialmente no existe conexión lógica entre los ETD y la red. Una red orientada a conexión cuida bastante los datos del usuario. El procedimiento exige una confirmación explicita de que se ha establecida la conexión, y si no es así la red informa al ETD solicitante que no ha podido establecer esa conexión. Las redes conectadas a conexión llevan un control permanente de todas las sesiones entre distintos ETD, e intentan asegurar que los datos no se pierdan en la red. Las redes no orientadas a conexión pasan directamente del estado libre al modo de transferencia de datos, finalizado el cual vuelve al estado libre. Además, las redes de este no ofrecen confirmaciones, control de flujo ni recuperación de errores aplicables a toda la red, aunque estas funciones si existen para cada enlace particular, el coste de una red no orientada a conexión es mucho menor. Las redes orientas a conexión suelen compararse conceptualmente con el sistema telefónico. El que llama sabe que se ha establecido una comunicación cuando oye hablar a alguien al otro lado de la línea.

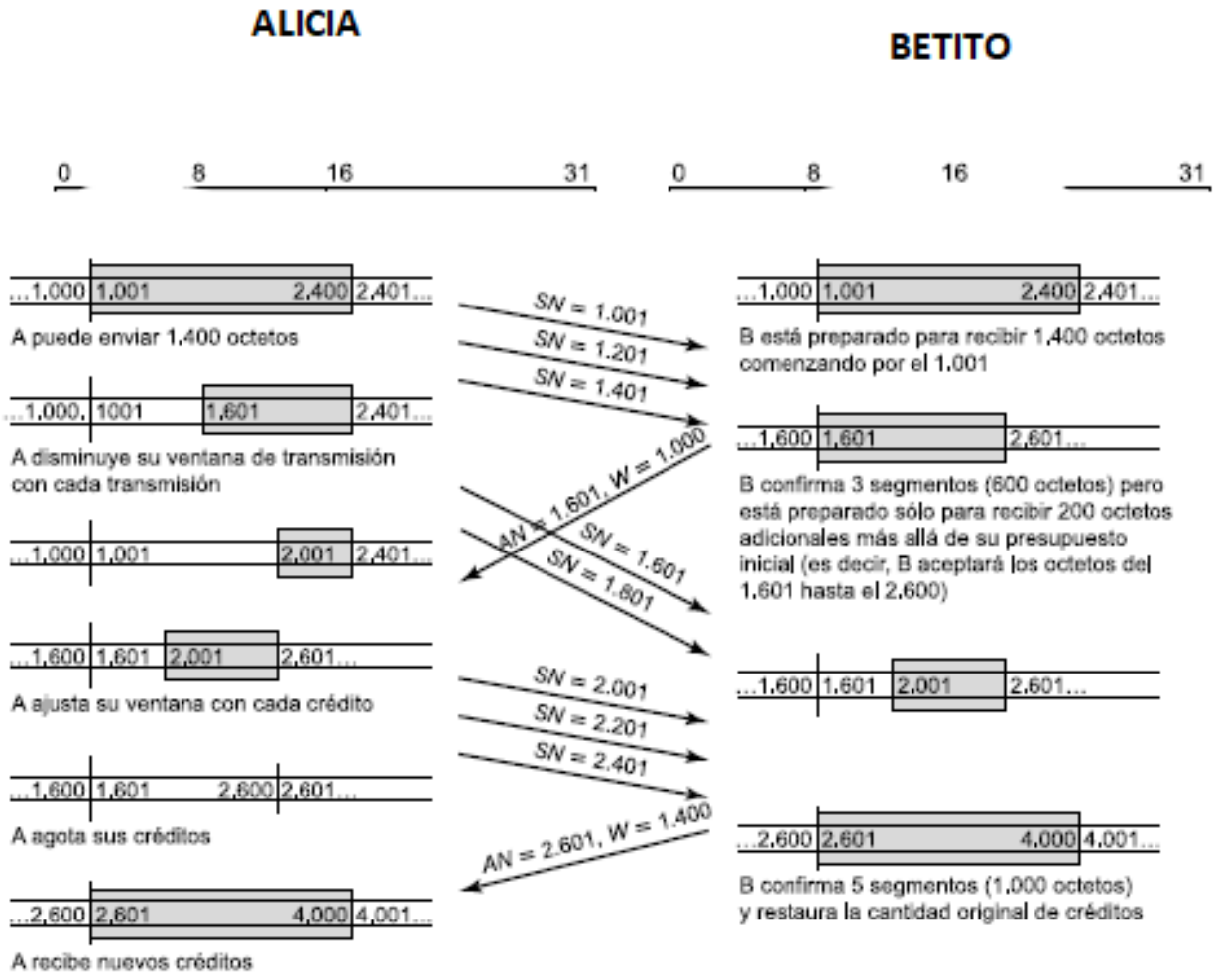

Figura 11: Sistema Orientado a Conexión entre Alicia y Betito

1.6. Protocolo TCP

TCP proporciona un servicio de transporte sofisticado. Proporciona entrega fiable de secuencias de bytes arbitrariamente grandes por vía de la abstracción de la programación basada en streams. La garantía de fiabilidad implica la entrega al proceso receptor de todos los datos confiados al protocolo TCP por el proceso emisor y en el mismo orden TCP está orientado a conexión. Antes de transferir cualquier dato, el proceso emisor y el receptor deben cooperar para establecer un canal de comunicaciones bidireccional. La conexión es simplemente un acuerdo extremo a extremo para realizar una transmisión fiable de datos.

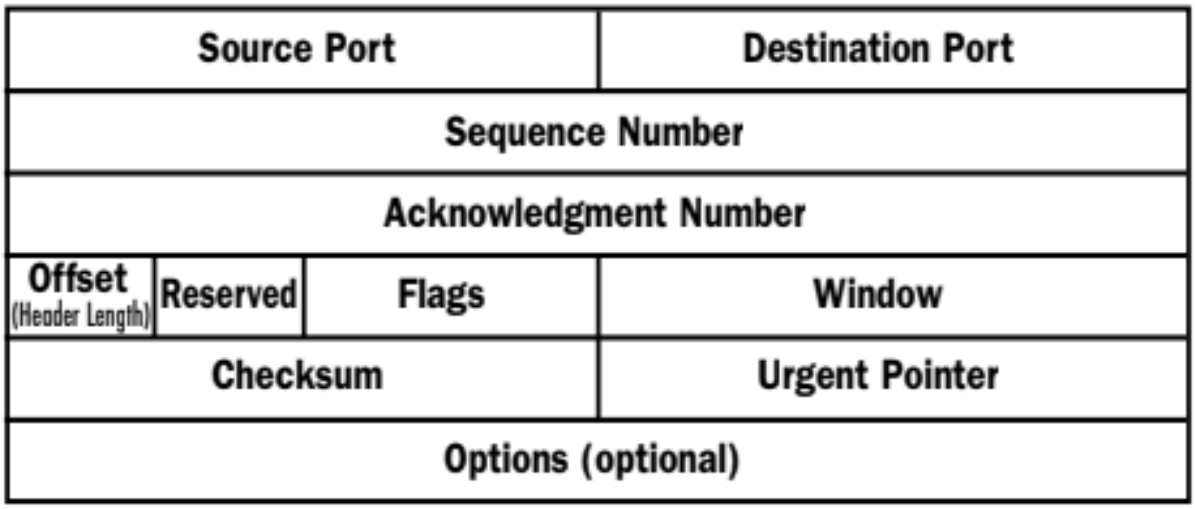

Figura 12: Es la cabecera de TCP

Los primeros 2 campos son el puerto de salida y el puerto destino, ambos con 2 bytes, estos simplemente nos van a decir por que puerto va a salir y por que puerto va a entrar, para este hay varias opciones, el numero de secuencia nos indica su numero, basado en secuencias de bytes de salida del primer byte del segmento; el siguiente nos indica el numero de asentamiento o reconocimiento, que es el numero de secuencia del siguiente byte que el receptor espera recibir; el offset nos indica el numero de palabras de 32 bits de la cabecera TCP, valor minimo = 5, valor maximo = 15; 6 bits reservadis 0's; la bandera nos indica si es urgente, de asentamiento, push, reset, o finalizacion; el siguiente ventana nos indica el numero de bytes que hay disponible en el buffer receptor; despues el cheksum que esta conformado por una pseudo cabecera que tiene elementos de IP y todo el segemento TCP; para al final tener el puntero urgente que nos indica la ubicación de los datos urgentes en el segmento.

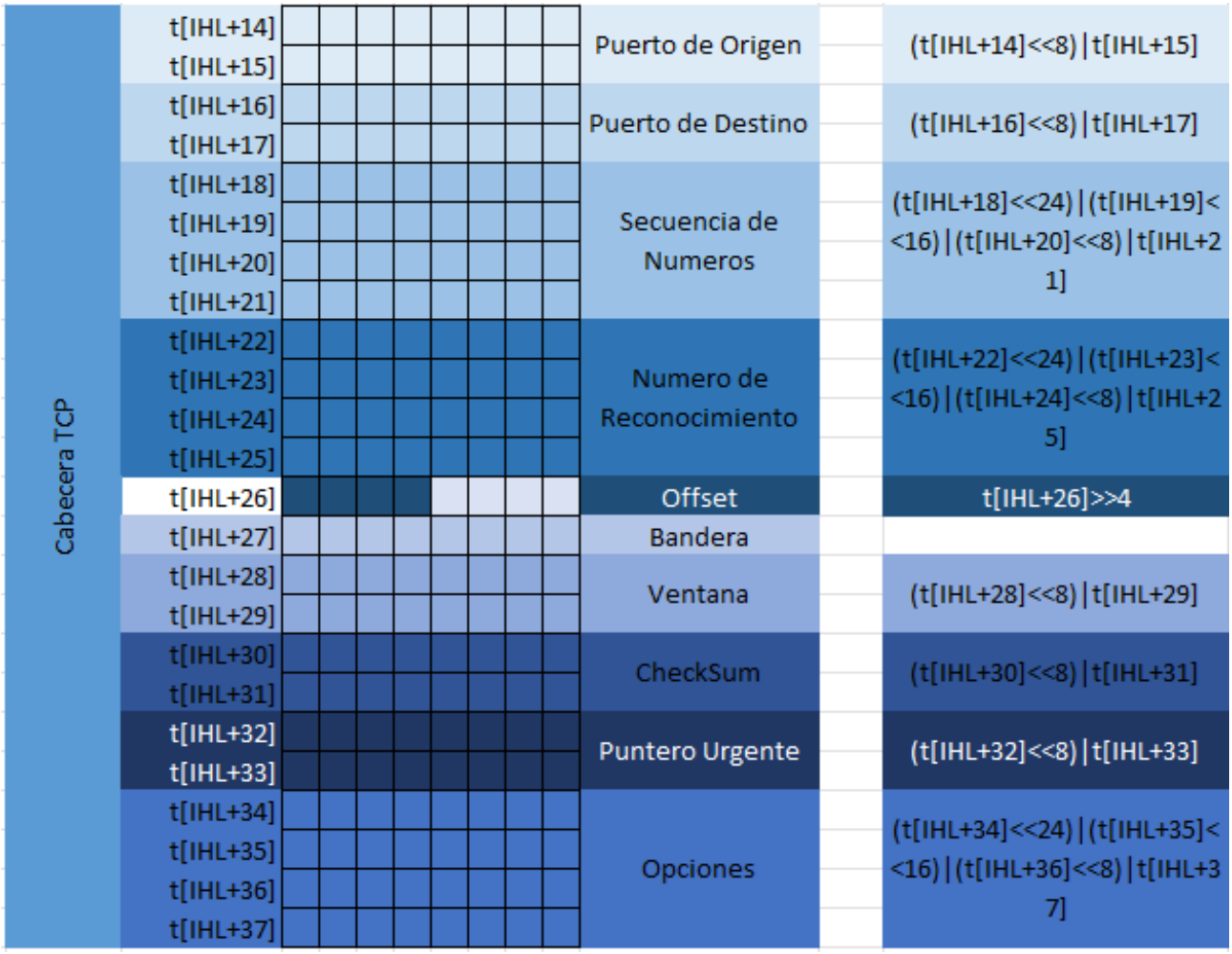

Figura 13: Es el mapa de memoria del Protocolo TCP

Figura 14: Es el mapa de memoria de la pseudocabecera para el checksum del Protocolo TCP

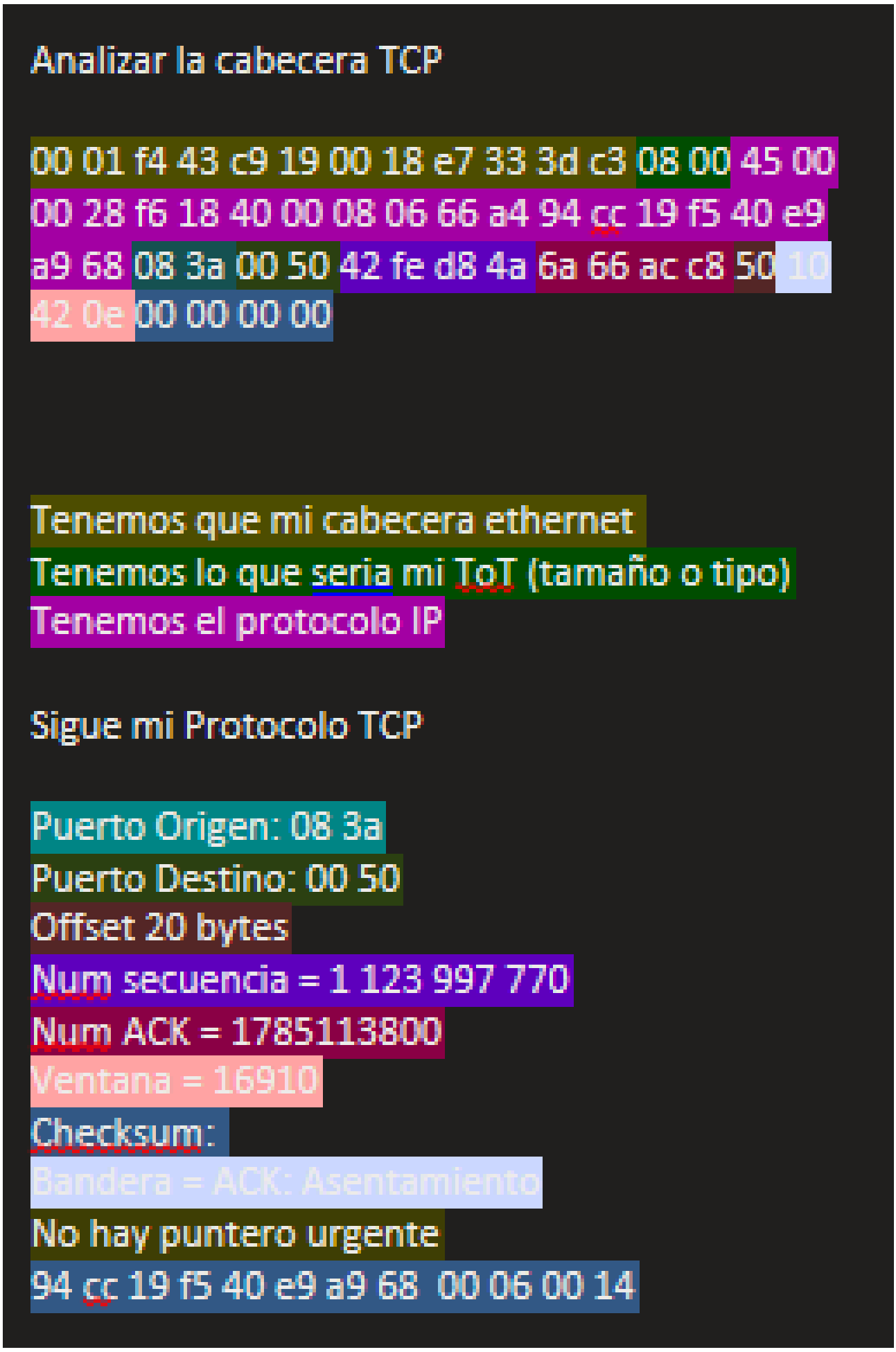

Figura 15: Un ejemplo para la realizacion del cheksum en TCP

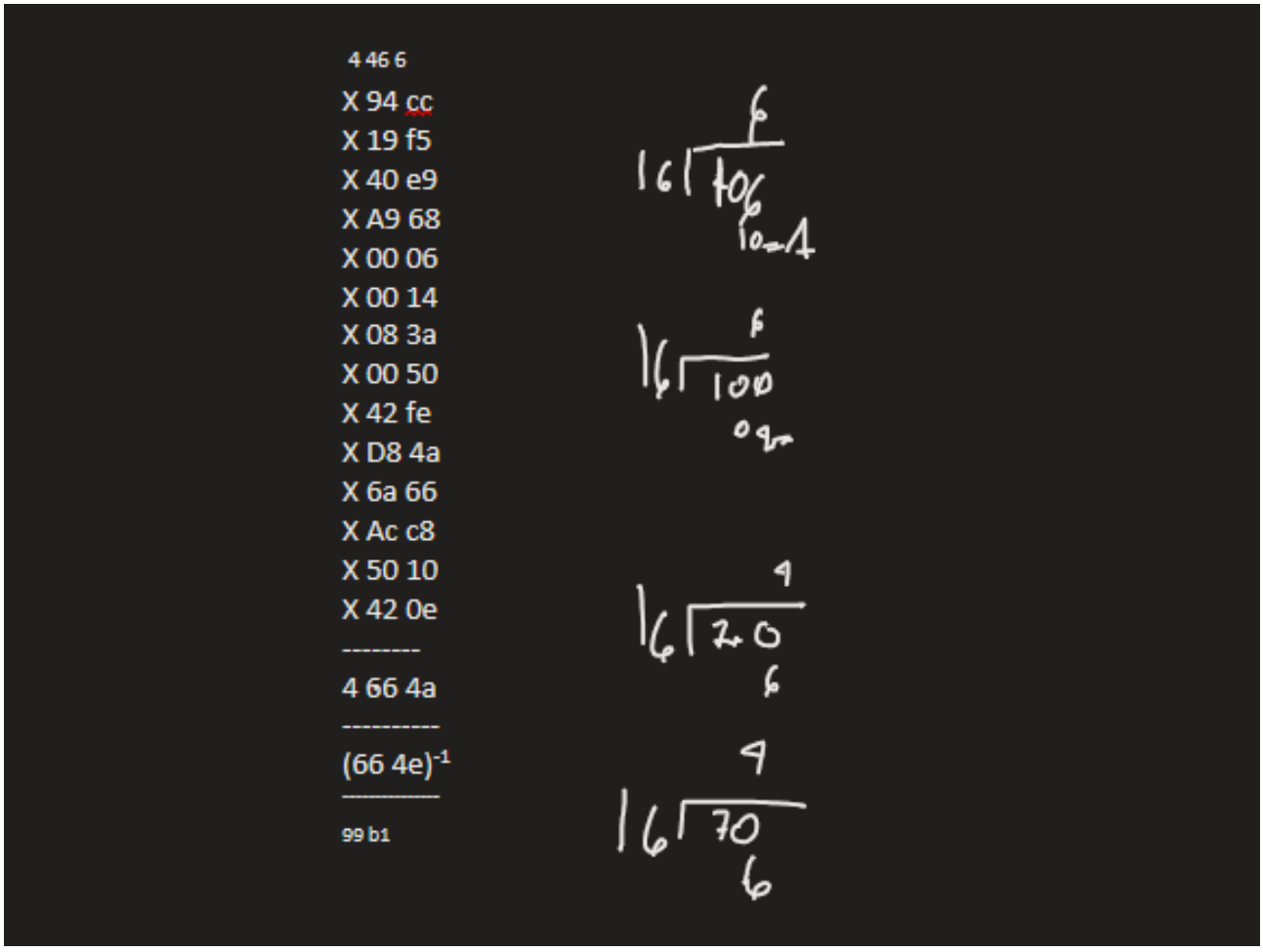

Figura 16: Resolucion de ejemplo para la realizacion del cheksum en TCP

1.7. El Protocolo UDP

UDP es casi una réplica en el nivel de transporte IP. Cada datagrama UDP se encapsula en un paquete IP. Tiene una cabecera corta que incluye los números de puerto origen y destino (las direcciones correspondientes a los hosts están en la cabecera IP), un campo de longitud y otro que es el checksum. UDP no ofrece garantía de entrega. Ya hemos comentado que los datagramas IP pueden desecharse en caso de congestión o error en la red. UDP no añade ningún mecanismo adicional de fiabilidad excepto el checksum, opcional. Si el checksum es distinto de 0, el host receptor calcula el valor de comprobación para el contenido del paquete y lo compara con el valor de comprobación para el contenido del paquete y lo compara con el valor recibido en el paquete, desechándolo en caso de que no coincidan. Su uso está restringido a aquellas aplicaciones y servicios que no requieran una entrega fiable de mensajes simples o múltiples.

Source Port	Destination Port
Length	Checksum

Figura 17: Es la cabecera de UDP

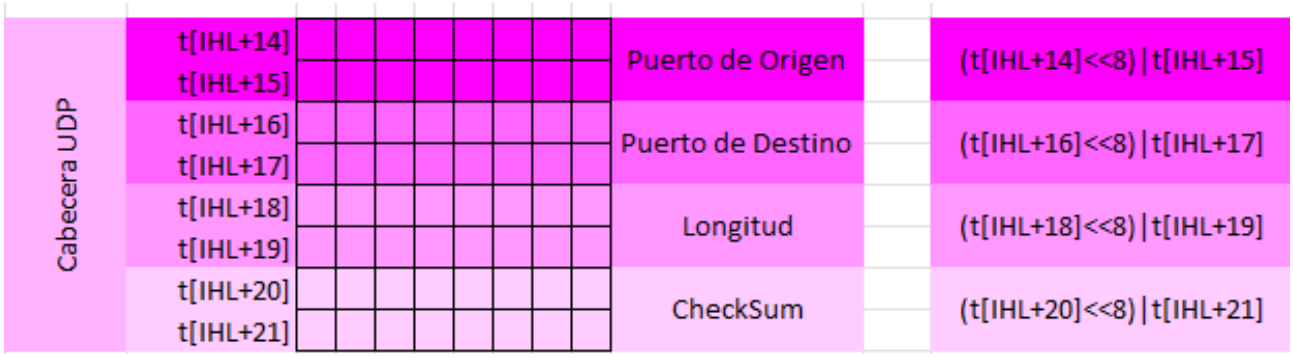

Figura 18: Es el mapa de memoria del Protocolo UDP

2. Código

Listing 2: Analizador de tramas en C

```c
#include<stdio.h>
#include <stdlib.h>
void analizarTrama(unsigned char *T);
void leerTrama(unsigned char *);
void analizaARP(unsigned char *);
void analizaIp(unsigned char *);
void decimal(char c, unsigned char *);
void ICMP(unsigned char *, int);
void TCP(unsigned char *, int);
void checksumTCP(unsigned char *, int, int);
void UDP(unsigned char *, int);
int main(){
  printf("Integrantes: \n");
  printf("Hernandez Vergara Eduardo\nRojas Cruz Jose Angel");
    FILE * ptr;
    char c;
  unsigned char hex = 0, ct = 0, p = 0, n = 0, i = 0;
  unsigned char nbytet[36];
  ptr = fopen("tramasIP.txt", "r");
    while(c != EOF){
    c = fgetc(ptr);
    if(c == '}'){
      nbytet[p] = ct;
      ct = 0;
      p++;
    }
    if(hex == 2){
      hex = 0;
      ct++;//Contar cantidad de bytes
    }
    if(hex == 1){
      hex++;
    }

    if(c == 'x')
      hex = 1;
  }
  fclose(ptr);
  p = 0;
  c = 0;
  ptr = fopen("tramasIP.txt", "r");
    unsigned char *trama = (unsigned char*)(malloc(sizeof(char)*nbytet[0])
        );
    while(c != EOF){
      c = fgetc(ptr);
      if(c == '}'){
                leerTrama(trama);
                free(trama);
                i++;
                trama = (unsigned char*)(malloc(sizeof(char)*nbytet[i]));
        p = 0;
      }
      if(hex == 2){
        decimal(c, &n);
        trama[p] = n;
        n = 0;
```

```
56          hex = 0;
57          p++;
58        }
59        if(hex == 1){
60          decimal(c, &n);
61          n *= 16;
62          hex++;
63        }
64
65        if(c == 'x')
66          hex = 1;
67      }
68    fclose(ptr);
69
70      return 0;
71  }
72
73  void leerTrama(unsigned char * T){
74      printf("\nCabecera ethernet \n");
75      unsigned short tot = T[12] << 8 | T[13];
76      printf("MAC DESTINO %.2x: %.2x: %.2x: %.2x: %.2x: %.2x\n", T[0], T[1],
            T[2], T[3], T[4], T[5]);
77      printf("MAC ORIGEN %.2x: %.2x: %.2x: %.2x: %.2x: %.2x\n", T[6], T[7],
           T[8], T[9], T[10], T[11]);
78      if (tot < 1500){
79          printf("Tamano de la cabecera LLC: %d bytes \n", tot);
80          analizarTrama(T);
81      }else{
82          if (tot == 2048){
83              printf("TIPO IP\n");// analiza IP
84              analizaIp(T);
85          }else if (tot == 2054){
86              printf("TIPO ARP\n");// analiza ARP
87              analizaARP(T);
88          }else{
89              printf("TIPO: %.2x%.2x", T[12], T[13]);
90          }
91      }
92  }
93
94  void analizarTrama(unsigned char *T){
95      char ss[][5] = {"RR", "RNR", "REJ", "SREJ"};
96      char uc[][5] = {"UI", "SIM", "-", "SARM", "UP", "-", "-", "SABM", "
            DISC",
97      "-", "-", "SARME", "--", "-", "-", "SABME", "SNRM", "--", "-", "RSET",
            "-",
98      "-", "-", "XID", "-", "-", "-", "SNRME"};
99      char ur[][5] = {"UI", "RIM", "-", "DM", "-", "-", "-", "-", "RD",
100     "-", "-", "--", "UA", "-", "-", "--", "--", "FRMR", "-", "--", "-",
101     "-", "-", "XID", "-", "-", "-", "--"};
102     printf("TIPO: %.2x %.2x\n", T[16], T[17]);
103     switch (T[16]&3){
104     case 0:
105         if (T[17]&1){
106             if (T[15]&1){
107                 printf("TIPO: T-I. N(s) = %d, N(r)=%d 1-f\n", T[16]>>1, T
                        [17]>>1);
108             }else{
109                 printf("TIPO: T-I. N(s) = %d, N(r)=%d 1-p\n", T[16]>>1, T
                        [17]>>1);
```

```
110             }
111         }else{
112             if (T[15]&1){
113                 printf("TIPO: T-I. N(s) = %d, N(r)=%d 0-f\n", T[16]>>1, T
                        [17]>>1);
114             }else{
115                 printf("TIPO: T-I. N(s) = %d, N(r)=%d 0-p\n", T[16]>>1, T
                        [17]>>1);
116             }
117         }
118         break;
119     case 1:
120         printf("t-S, S = %s\n", ss[(T[16]>>2)&3]);
121         if (T[17]&1){
122             if (T[15]&1){
123                 printf("TIPO: T-S. N(s) = -, N(r)=%d 1-f\n", T[17]>>1);
124             }
125             else{
126                 printf("TIPO: T-S. N(s) = -, N(r)=%d 1-p\n", T[17]>>1);
127             }
128         }else{
129             if (T[15]&1){
130                 printf("TIPO: T-S. N(s) = -, N(r)=%d 0-f\n", T[17]>>1);
131             }
132             else{
133                 printf("TIPO: T-S. N(s) = -, N(r)=%d 0-p\n", T[17]>>1);
134             }
135         }
136         break;
137     case 2:
138         if (T[17]&1){
139             if (T[15]&1){
140                 printf("TIPO: T-I. N(s) = %d, N(r)=%d 1-f\n", T[16]>>1, T
                        [17]>>1);
141             }else{
142                 printf("TIPO: T-I. N(s) = %d, N(r)=%d 1-p\n", T[16]>>1, T
                        [17]>>1);
143             }
144         }else{
145             if (T[15]&1){
146                 printf("TIPO: T-I. N(s) = %d, N(r)=%d 0-f\n", T[16]>>1, T
                        [17]>>1);
147             }else{
148                 printf("TIPO: T-I. N(s) = %d, N(r)=%d 0-p\n", T[16]>>1, T
                        [17]>>1);
149             }
150         }
151         break;
152     case 3:
153         if (T[16]&16){
154             if (T[15]&1){
155                 printf("T-U %s 1-f\n", ur[(T[16]>>2&3)|(T[16]>>3&28)]);
156             }else{
157                 printf("T-U %s 1-p\n", uc[(T[16]>>2&3)|(T[16]>>3&28)]);
158             }
159         }else{
160             if (T[15]&1){
161                 printf("T-U %s 0-f\n", ur[(T[16]>>2&3)|(T[16]>>3&28)]);
162             }else{
163                 printf("T-U %s 0-p\n", uc[(T[16]>>2&3)|(T[16]>>3&28)]);
```

```
164             }
165         }
166         break;
167     }
168 }
169 void analizaARP(unsigned char *T){
170     if(T[14]<<8 | (T[15] == 1)){
171         printf("TIPO: ARP\n");
172     }else if(T[14]<<8 | (T[15] == 6)){
173         printf("IEEE 80.2 LAN\n");
174     }else{
175         printf("Otro: %d\n", (T[14]<<8 |T[15]));
176     }
177     // Tipo de direccion de Protocolo
178     if (T[16]<<8 | (T[17] == 0x0806)){
179         printf("TIPO: iPv4\n");
180     }else{
181         printf("TIPO: %.2x, %.2x\n", T[16], T[17]);
182     }
183     //Tama\~no de la MAC
184     printf("Tama\~no MAC: %d bytes\n", T[18]);
185     //Tama\~no de la Direccion IP
186     printf("Tama\~no IP: %d bytes\n", T[19]);
187     //Op Code
188     if (T[20]<<8 | (T[21] == 1)){
189         printf("Op Code: ARP Request\n");
190     }else if (T[20]<<8 | (T[21] == 2)){
191         printf("Op Code: ARP Reply\n");
192     }else{
193         printf("Otro: %d\n", (T[20]<<8 | T[21]));
194     }
195     printf("Direccion MAC origen: %.2x:%.2x:%.2x:%.2x:%.2x:%.2x\n", T[22],
            T[23], T[24], T[25], T[26], T[27]);
196     printf("Direccion IP origen: %d.%d.%d.%d\n", T[28], T[29], T[30], T
        [31]);
197     printf("Direccion MAC destino: %.2x:%.2x:%.2x:%.2x:%.2x:%.2x\n", T
        [32], T[33], T[34], T[35], T[36], T[37]);
198     printf("Direccion IP destino: %d.%d.%d.%d\n", T[38], T[39], T[40], T
        [41]);
199
200 }
201 void analizaIp(unsigned char *T){
202     int IHL = (T[14]&15) * 4;
203   printf("Es de tipo IPv%d\n", T[14]>>4);
204   if(T[15]&2){
205     printf("Es un ts Costo minimo\n");
206   }else if(T[15]&4){
207     printf("Es un ts fiabilidad \n");
208   }else if(T[15]&8){
209     printf("Es un ts maximo rendimiento\n");
210   }else if(T[15]&16){
211     printf("Es un ts retardo minimo\n");
212   }
213     else{
214         printf("Es un ts normal\n");
215     }
216     //Internet Header Length
217     printf("Tama\~no de cabecera: %d bytes\n", (T[14]&15) * 4);
218     //Tipo de servicio
219     printf("Tipo de servicio: %d\n", T[17]);
```

```
220   printf("El tama\~no total es de %d\n", T[16]<<8 | T[17]);
221   printf("El id es de %d\n", T[18]<<8 | T[19]);
222   if(T[20]&64){
223     printf("Dont fragment\n");
224   }else if(T[20]&32){
225     printf("More Fragment\n");
226   }
227   printf("El desplazamiento del fragmento es de %d\n", (T[20]&31)<<8 | T
        [21]);
228   printf("El tiempo de vida es %d\n", T[22]);
229   if(T[23] == 1){
230     printf("Es ICMP\n");
231         ICMP(T, IHL);
232   }else if(T[23] == 6){
233         printf("Es TCP\n");
234         TCP(T, IHL);
235     }else if(T[23] == 17){
236         printf("Es UDP\n");
237         UDP(T, IHL);
238     }else{
239         printf("Es otro\n");
240     }
241     //Calcula y verifica ChecksumIP
242     printf("--Checksum IP.--\n");
243     unsigned char i = 0;
244     unsigned short int sum = 0, chucksum = 0, aux;
245     for(; i <= IHL +14-1; i+=2){
246         sum += T[i]<<8 | T[i+1];
247     }
248     if((~((sum >> 16) + (sum & 0xFFFF)) & 0xFFFF) == 0x0000){
249         printf("Checksum: Correcto\n");
250         printf(":)\n");
251     }else{
252         printf("Checksum: Incorrecto\n");
253         printf(":(\n");
254         T[24] = 0x00;
255         T[25] = 0x00;
256         i = 0;
257         sum = 0;
258         for(; i <= T[14+IHL-1]; i+=2){
259             sum += T[i]<<8 | T[i+1];
260         }
261     }
262     printf("Operacion Resultado: %.2x\n", sum);
263     chucksum = (sum >> 16) + (sum & 0xFFFF);
264     chucksum = 0xFFFF - chucksum;
265     aux = chucksum;
266     T[24] = aux/256;
267     T[25] = aux%256;
268     printf("Este es el chucksum correcto: %.2x\n", chucksum);
269     printf(":)\n");
270     chucksum = ~(chucksum + sum);
271     if(chucksum == 0x0000){
272         printf("Checksum: Correcto\n");
273     }
274     //datos
275     printf("El checksum es: %2x\n", T[24]<<8 | T[25]);
276   printf("Direccion IP origen: %d.%d.%d.%d\n", T[26], T[27], T[28], T[29])
      ;
```

```
277   printf("Direccion IP destino: %d.%d.%d.%d\n", T[30], T[31], T[32], T
          [33]);
278     //opciones IHL
279     printf("Opciones: ");
280     if (((T[14]&15) * 4 > 20) && ((T[14]&15) * 4 < 60) ){
281         int l;
282         for (l = 34; l < ((T[14]&15)*4) + 14; l++){
283             printf("%.2X ", T[l]);
284         }
285     }else{
286         printf("No hay opciones\n");
287     }
288
289 }
290 void ICMP(unsigned char *T, int IHL){
291     printf("ICMP: ");
292     printf("Tipo de respuesta ECO : %.2x\n", T[IHL+14]);
293     if (T[IHL+14] == 0){
294         printf("Tipo de respuesta: Echo Reply\n");
295         //Datos echo reply
296     }else if (T[IHL+14] == 8){
297         printf("Tipo de respuesta: Echo Request\n");
298     }
299     printf("Mi codigo es: %.2x\n", T[IHL+15]);
300     printf("Mi checksum es: %.2x\n", T[IHL+16] << 8 | T[IHL+17]);
301     printf("El id es: %.2x\n", T[IHL+18] << 8 | T[IHL+19]);
302     printf("El numero de secuencia es: %.2x\n", T[IHL+20] << 8 | T[IHL
          +21]);
303 }
304 void TCP(unsigned char *T, int IHL){
305     printf("TCP: ");
306     printf("Puerto origen: %d\n", T[IHL+14]<<8 | T[IHL+15]);
307     printf("Puerto destino: %d\n", T[IHL+16]<<8 | T[IHL+17]);
308     printf("Numero de secuencia: %d\n", T[IHL+18]<<24 | T[IHL+19]<<16 | T[
          IHL+20]<<8 | T[IHL+21]);
309     printf("Numero de ack: %d\n", T[IHL+22]<<24 | T[IHL+23]<<16 | T[IHL
          +24]<<8 | T[IHL+25]);
310     printf("Tama\~no de la cabecera: %d\n", (T[IHL+26]>>4) * 4);
311     int offset = (T[IHL+26]&15) * 4;
312     //Banderas
313     printf("Banderas: ");
314     if(T[IHL+27]&1){
315         printf("FIN\n");
316     }
317     if(T[IHL+27]&2){
318         printf("SYN\n");
319     }
320     if(T[IHL+27]&4){
321         printf("RST\n");
322     }
323     if(T[IHL+27]&8){
324         printf("PSH\n");
325     }
326     if(T[IHL+27]&16){
327         printf("ACK\n");
328     }
329     if(T[IHL+27]&32){
330         printf("URG\n");
331     }
332     if(T[IHL+27]&64){
```

```
333         printf("ECE\n");
334     }
335     if(T[IHL+27]&128){
336         printf("CWR\n");
337     }
338     //Ventana
339     printf("Ventana: %d\n", T[IHL+28]<<8 | T[IHL+29]);
340     //Checksum
341     checksumTCP(T, offset, IHL);
342     //Puntero Urgente
343     printf("Puntero Urgente: %d\n", T[IHL+32]<<8 | T[IHL+33]);
344     //Opciones
345     printf("Opciones: ");
346     if ((T[IHL+26]&15) * 4 > 20 && (T[IHL+26]&15) * 4 < 60 ){
347         int l;
348         for (l = 34; l < (T[IHL+26]&15) + 34; l++){
349             printf("%.2X ", T[l]);
350         }
351     }else{
352         printf("No hay opciones\n");
353     }
354 }
355 void checksumTCP(unsigned char *T, int offset, int IHL){
356     unsigned char i = 0;
357     unsigned short int checksum = 0, aux = 0;
358     int sum = 0;
359     for(; i <= IHL+14+offset; i+=2){
360         sum += T[IHL+14+i]<<8 | T[IHL+14+i+1];
361     }
362     sum += T[26]<<8 | T[27];
363     sum += T[28]<<8 | T[29];
364     sum += T[30]<<8 | T[31];
365     sum += T[32]<<8 | T[33];
366     sum += T[13]<<8 | T[23];
367     sum += 0x00| ((T[IHL+26]<<4)*4)>>8 ;
368     //printf("Operacion Resultado: %x\n", sum);
369     aux = (sum >> 16) + (sum & 0xFFFF);
370     aux = 0xFFFF - aux;
371     checksum = (T[IHL+30] << 8 | T[IHL+31]);
372     printf("Checksum: %.2x\n", checksum);
373 
374     if(aux == 0){
375         printf("Checksum: Correcto\n");
376     }else{
377         printf("Checksum: Incorrecto\n");
378         T[IHL+30] = 0x00;
379         T[IHL+31] = 0x00;
380         i = 0;
381         sum = 0;
382         for(; i <= IHL+14+offset; i+=2){
383             sum += T[i]<<8 | T[i+1];
384         }
385         sum += T[26]<<8 | T[27];
386         sum += T[28]<<8 | T[29];
387         sum += T[30]<<8 | T[31];
388         sum += T[32]<<8 | T[33];
389         sum += T[13]<<8 | T[23];
390         sum += T[IHL+26]<<8 | 0x00;
391 
392         //printf("Operacion Resultado: %.2x\n", sum);
```

```
393         checksum = (~((sum >> 16) + (sum & 0xFFFF)) & 0xFFFF);
394         if(checksum == 0x0000){
395             printf("Checksum: Correcto\n");
396         }
397     }
398 }
399 void UDP(unsigned char *T, int IHL){
400     printf("UDP: \n");
401     printf("Puerto origen: %d\n", T[IHL+14]<<8 | T[IHL+15]);
402     printf("Puerto destino: %d\n", T[IHL+16]<<8 | T[IHL+17]);
403     printf("Tama\~no de la cabecera: %d\n", (T[IHL+18]>>4) * 4);
404     printf("Checksum: %.2x %.2x\n", T[IHL+19]<<8,  T[IHL+20]);
405     //Opciones
406     printf("Opciones: ");
407     if ((T[IHL+18]&15) * 4 > 20 && (T[IHL+18]&15) * 4 < 60 ){
408         int l;
409         for (l = 20; l < (T[IHL+18]&15) + 20; l++){
410             printf("%.2X ", T[l]);
411         }
412         printf("\n");
413     }else{
414         printf("No hay opciones\n");
415     }
416 }
417 void decimal(char c, unsigned char *n){
418   switch(c){
419     case '1':
420       *n +=1;
421     break;
422     case '2':
423       *n +=2;
424     break;
425     case '3':
426       *n +=3;
427     break;
428     case '4':
429       *n +=4;
430     break;
431     case '5':
432       *n +=5;
433     break;
434     case '6':
435       *n +=6;
436     break;
437     case '7':
438       *n +=7;
439     break;
440     case '8':
441       *n +=8;
442     break;
443     case '9':
444       *n +=9;
445     break;
446     case 'a':
447         case 'A':
448       *n +=10;
449     break;
450     case 'b':
451         case 'B':
452       *n +=11;
```

```
453       break;
454       case 'c':
455           case 'C':
456         *n +=12;
457       break;
458       case 'd':
459           case 'D':
460         *n +=13;
461       break;
462       case 'e':
463           case 'E':
464         *n +=14;
465       break;
466       case 'f':
467           case 'F':
468         *n +=15;
469       break;
470       default:
471         *n += 0;
472    }
473 }
```

3. Solución

3.1. Protocolo IP

```
Integrantes:
Hernandez Vergara Eduardo
Rojas Cruz Jose Angel
Cabecera ethernet
MAC DESTINO aa: aa: aa: aa: aa: aa
MAC ORIGEN bb: bb: bb: bb: bb: bb
TIPO IP
Es de tipo IPv4
Es un ts normal
Tamaño de cabecera: 24 bytes
Tipo de servicio: 56
El tamaño total es de 56
El id es de 9773
El desplazamiento del fragmento es de 0
El tiempo de vida es 64
Es ICMP
ICMP: Tipo de respuesta ECO : 00
Tipo de respuesta: Echo Reply
Mi codigo es: 00
Mi checksum es: abcd
El id es: 1010
El numero de secuencia es: 00
Operacion Resultado: 5b79
Checksum: Incorrecto
:(
Operacion Resultado: abb2
Este es el chucksum correcto: 544d
:)
Checksum: Correcto
El checksum es: 544d
Direccion IP origen: 192.172.1.4
Direccion IP destino: 192.172.241.84
Opciones: 48 4F 4C 41
```

Figura 19: Una trama IP con opciones -ICMP- se imprimen las opciones en hexadecimal

```
Cabecera ethernet
MAC DESTINO aa: aa: aa: aa: aa: aa
MAC ORIGEN cc: cc: cc: cc: cc: cc
TIPO IP
Es de tipo IPv4
Es un ts Costo minimo
Tamaño de cabecera: 20 bytes
Tipo de servicio: 56
El tamaño total es de 56
El id es de 9773
El desplazamiento del fragmento es de 0
El tiempo de vida es 64
Es ICMP
ICMP: Tipo de respuesta ECO : 00
Tipo de respuesta: Echo Reply
Mi codigo es: 00
Mi checksum es: abcd
El id es: 1010
El numero de secuencia es: 00
Operacion Resultado: f91e
Checksum: Incorrecto
:(
Operacion Resultado: 5557
Este es el chucksum correcto: aaa8
:)
Checksum: Correcto
El checksum es: aaa8
Direccion IP origen: 192.172.1.4
Direccion IP destino: 192.172.241.84
Opciones: No hay opciones
```

Figura 20: Una trama IP de costo mÃnimo y se imprime TTL

```
Cabecera ethernet
MAC DESTINO aa: aa: aa: aa: aa: aa
MAC ORIGEN cc: cc: cc: cc: cc: cc
TIPO IP
Es de tipo IPv4
Es un ts Costo minimo
Tamaño de cabecera: 20 bytes
Tipo de servicio: 56
El tamaño total es de 56
El id es de 9773
El desplazamiento del fragmento es de 161
El tiempo de vida es 64
Es UDP
UDP: Puerto origen: 1845
Puerto destino: 2052
Tamaño de la cabecera: 0
Checksum: 00 00
Opciones: No hay opciones
Operacion Resultado: f9cf
Checksum: Incorrecto
:(
Operacion Resultado: 9d64
Este es el chucksum correcto: 629b
:)
Checksum: Correcto
El checksum es: 629b
Direccion IP origen: 192.172.1.4
Direccion IP destino: 192.172.241.84
Opciones: No hay opciones
```

Figura 21: Una trama UDP cuyo encapsulado IP no tiene opciones y se devuelve el valor del offset en decimal

```
Integrantes:
Hernandez Vergara Eduardo
Rojas Cruz Jose Angel
Cabecera ethernet
MAC DESTINO aa: aa: aa: aa: aa: aa
MAC ORIGEN bb: bb: bb: bb: bb: bb
TIPO IP
Es de tipo IPv4
Es un ts normal
Tamaño de cabecera: 24 bytes
Tipo de servicio: 56
El tamaño total es de 56
El id es de 9773
El desplazamiento del fragmento es de 0
El tiempo de vida es 64
Es ICMP
ICMP: Tipo de respuesta ECO : 00
Tipo de respuesta: Echo Reply
Mi codigo es: 00
Mi checksum es: abcd
El id es: 1010
El numero de secuencia es: 00
Operacion Resultado: 5b79
Checksum: Incorrecto
:(
Operacion Resultado: abb2
Este es el chucksum correcto: 544d
:)
Checksum: Correcto
El checksum es: 544d
Direccion IP origen: 192.172.1.4
Direccion IP destino: 192.172.241.84
Opciones: 48 4F 4C 41
```

Figura 22: Verificar el checksum de las tramas IP, en caso de que este correcto imprimir :) en caso de que sea incorrecto :(e imprimir el checksum correcto.

3.2. Protocolo TCP

Colocar el código correspondiente para la construccion de la pseudocabecera TCP. Y resultados de su ejecución para una trama con el Checksum correcto y otra con el checksum incorrecto.

```
Cabecera ethernet
MAC DESTINO 00: 01: f4: 43: c9: 19
MAC ORIGEN 00: 18: e7: 33: 3d: c3
TIPO IP
Es de tipo IPv4
Es un ts normal
Tamaño de cabecera: 20 bytes
Tipo de servicio: 40
El tamaño total es de 40
El id es de 63000
Dont fragment
El desplazamiento del fragmento es de 0
El tiempo de vida es 128
Es TCP
TCP: Puerto origen: 2106
Puerto destino: 80
Numero de secuencia: 1123997770
Numero de ack: 1785113800
Tamaño de la cabecera: 20
Banderas: ACK
Ventana: 16910
Checksum: 00
Checksum: Incorrecto
Puntero Urgente: 0
Opciones: No hay opciones
--Checksum IP.--
Checksum: Incorrecto
:(
Operacion Resultado: 5901
Este es el chucksum correcto: a6fe
:)
Checksum: Correcto
El checksum es: a6fe
Direccion IP origen: 148.204.25.245
Direccion IP destino: 64.233.169.104
Opciones: No hay opciones
```

Figura 23: Ejemplo 1

```
Cabecera ethernet
MAC DESTINO 00: 01: f4: 43: c9: 19
MAC ORIGEN 00: 18: e7: 33: 3d: c3
TIPO IP
Es de tipo IPv4
Es un ts normal
Tamaño de cabecera: 20 bytes
Tipo de servicio: 40
El tamaño total es de 40
El id es de 63000
Dont fragment
El desplazamiento del fragmento es de 0
El tiempo de vida es 128
Es TCP
TCP: Puerto origen: 2106
Puerto destino: 80
Numero de secuencia: 1123997770
Numero de ack: 1785113800
Tamaño de la cabecera: 20
Banderas: ACK
Ventana: 16910
Checksum: 99b1
Checksum: Correcto
Puntero Urgente: 0
Opciones: No hay opciones
--Checksum IP.--
Checksum: Incorrecto
:(
Operacion Resultado: e592
Este es el chucksum correcto: 1a6d
:)
Checksum: Correcto
El checksum es: 1a6d
Direccion IP origen: 148.204.25.245
Direccion IP destino: 64.233.169.104
Opciones: No hay opciones
```

Figura 24: Ejemplo 2

3.3. Protocolo UDP

Para la implementación de este análizador se imprimen solamente los campos Poner el código correspondiente y la sálida para dos tramas ejemplo.

```
Cabecera ethernet
MAC DESTINO aa: aa: aa: aa: aa: aa
MAC ORIGEN cc: cc: cc: cc: cc: cc
TIPO IP
Es de tipo IPv4
Es un ts Costo minimo
Tamaño de cabecera: 20 bytes
Tipo de servicio: 56
El tamaño total es de 56
El id es de 9773
El desplazamiento del fragmento es de 161
El tiempo de vida es 64
Es UDP
UDP:
Puerto origen: 1845
Puerto destino: 2052
Tamaño de la cabecera: 16
Checksum: 00 00
Opciones: No hay opciones
--Checksum IP.--
Checksum: Incorrecto
:(
Operacion Resultado: dd64
Este es el chucksum correcto: 229b
:)
Checksum: Correcto
El checksum es: 229b
Direccion IP origen: 192.172.1.4
Direccion IP destino: 192.172.241.84
Opciones: No hay opciones
```

Figura 25: Ejemplo 1

```
Cabecera ethernet
MAC DESTINO aa: aa: aa: aa: aa: aa
MAC ORIGEN cc: cc: cc: cc: cc: cc
TIPO IP
Es de tipo IPv4
Es un ts Costo minimo
Tamaño de cabecera: 20 bytes
Tipo de servicio: 56
El tamaño total es de 56
El id es de 9773
El desplazamiento del fragmento es de 161
El tiempo de vida es 64
Es UDP
UDP:
Puerto origen: 22
Puerto destino: 0
Tamaño de la cabecera: 48
Checksum: 4400 01
Opciones: 00 A1 40 11 6B A4 C0 AC
--Checksum IP.--
Checksum: Incorrecto
:(
Operacion Resultado: 69eb
Este es el chucksum correcto: 9614
:)
Checksum: Correcto
El checksum es: 9614
Direccion IP origen: 192.172.1.4
Direccion IP destino: 192.172.241.84
Opciones: No hay opciones
```

Figura 26: Ejemplo 2

4. Conclusiones

A lo largo del desarrollo de esta práctica que fue a lo largo del semestre, pudimos ver la importancia y diferencia de los protocolos orientados a bits y los orientados a bytes, ya que un protocolo orientado a bits es mas eficiente ya que utilizamos y aprovechamos cada uno de los bits de los mensajes sin que haya desperdicio, claro que esto es más dificil sin embargo aprovechas mas los recursos, en cambio uno orientado a bytes es mas sencillo de utilizar, analizar y diseñar, sin embargo hay desperdicio de bits, y no uno pequeño este suele ser muy grande; ejemplos de esto esta el protocolo LLC que es orientado a bits y el protocolo IP que es orientado a bytes.

Antes de este curso de redes tenía una idea de que era el encapsulamiento y desencapsulamiento, sin embargo no sabía como funcionaba, antes solo decia que se tenian que encapsular los mensajes para que lleguen al remitente y desencapsularlos para que el remitente pueda leer esos datos, sin embargo, no es suficiente, por ello en esta materia vimos como se encapsulaban y transportaban por medio de los protocolos, lo cual me parecio correcto para que seamos capaces de observar de mejor manera el proceso que siguen los protocolos.

Alguna vez te habías puesto a pensar que cuando mandas un whatsapp con un *ola k ace* ocurría todo este proceso entre tu host y el host receptor? Y si después mandas *?* y despues otro con un emoji *:)*, es decir, 3 diferentes mensajes a cada uno de esos les pasa el encapsulamiento. Si consideraramos de manera general se transmiten por TCP y el encapsulado desde capa fisica hasta transporte, llena la siguiente tabla y una vez llena, que puedes decir como reflexión?

Mensaje	Cantidad de caractés	Bytes agregados en las cabeceras	Total de Bytes para el Mensaje
Ola k ace	9	9	78
?	1	1	74
:)	1	1	74

Cuadro 1: Mensaje y cabeceras

En la siguiente tabla podemos ver que la cantidad de carácteres es proporcional al numero de bytes a la cantidad de caracteres, ya que a cada caracter le corresponde un byte gracias a ASCII, en donde primero se convierte a un hexadecimal que es el del código que le corresponde y en el total de bytes para el mensaje tenemos los que fueron añadidos en opciones. Por ello somos capaces de observar que es importante esta parte de la encapsulacion.

El operador $*$ es un operador aritmetico en c, el cual nos multiplica los valores que le digamos, mientras que el $<<$ es un operador a nivel de bit en especifico este hace un corrimiento a la izquierda, sin embargo si nosotros realizamos las operaciones T[12]*256 + T[13] y T[12]<<8 | T[13], tenemos el mismo resultado, esto se debe a que multiplicamos el valor de nuestro unsigned char por 256 que nos moveria 8 bits a la izquierda y le sumamos el valor de T[13], mientras que el operador $<<$ directamente nos da el recorrimiento a la izquierda y el operador | le aplica un or logico a nuestra opercion, obvio en mas eficiente en este caso los operadores a nivel de bit q los operadores aritmeticos ya que es menos trabajo para el procesador

Si Betito se quiere comunicar con Alicia a traves de la red mas grande del mundo, que es la red de redes, se clasificaria como WAN, seria de tipo mixto, y seria por fibra optica, para comunicarse a grandes distancias, pero no es tan sencillo ya que es una gran red compuesta de redes mas pequeñas por ejemplo podriamos decir que estan en diferente subred y necesitan de un router con el DG para comunicarse, simplemente las variables son demasiadas para poder

llegar a una conclusion definitiva asi que concluimos que sin mas informacion no podemos decir exactamente todo el proceso que va a hacer.

5. Referencias

Referencias

[1] Couloris G., Dollimore, J., Kindberg, T. (2001). *Sistemas Distribuidos. Conceptos y diseño.* Madrid: Pearson.

[2] Tanenbaum, S. & Wetherall, J. (2012). *Redes de Computadoras.* México: Pearson.

[3] Stallings, W. (2004). *Comunicaciones y Redes de Computadores.* Madrid: Pearson.