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.

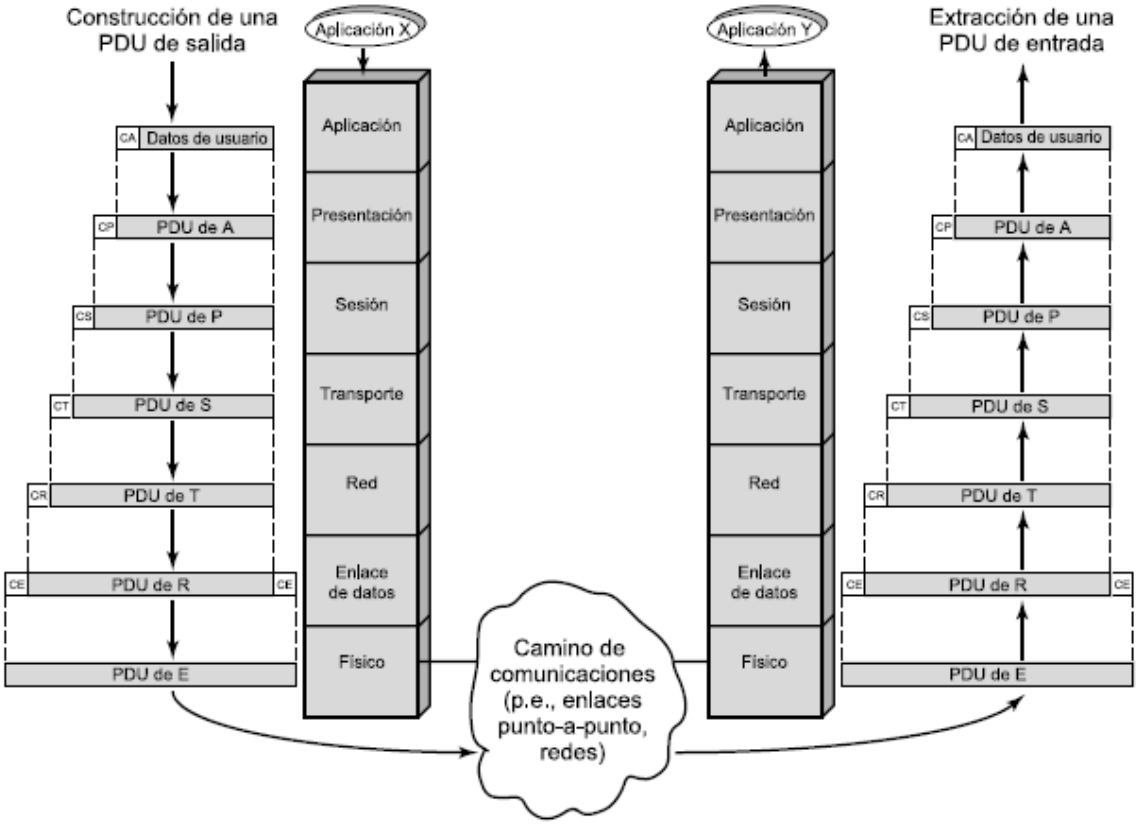

Figura 3: El entorno OSI

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.

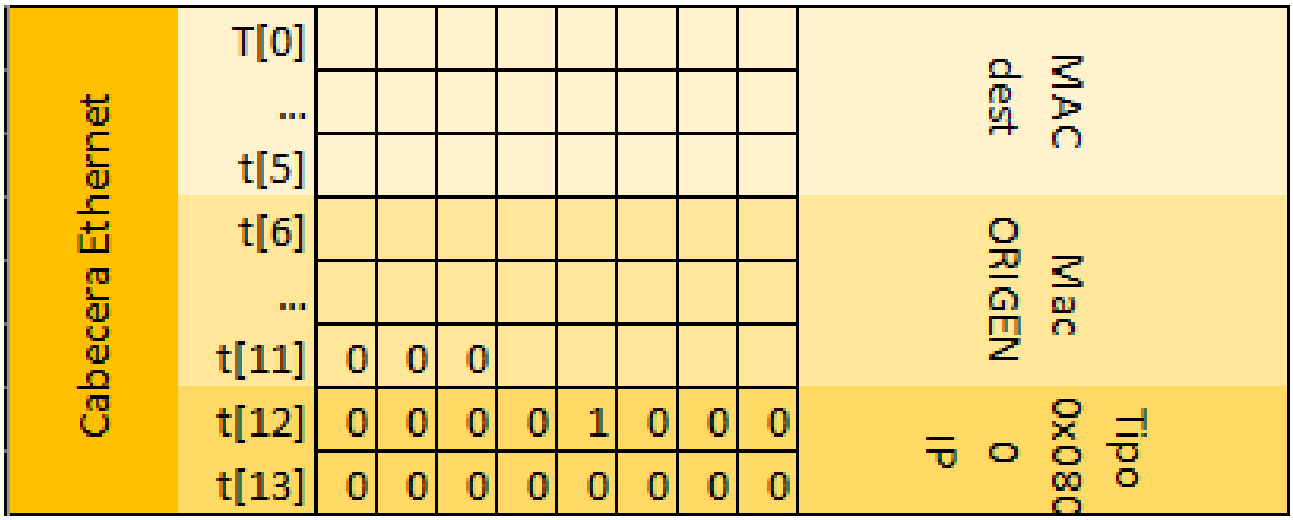

Figura 4: El mapa de memoria para la cabecera Ethernet

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.

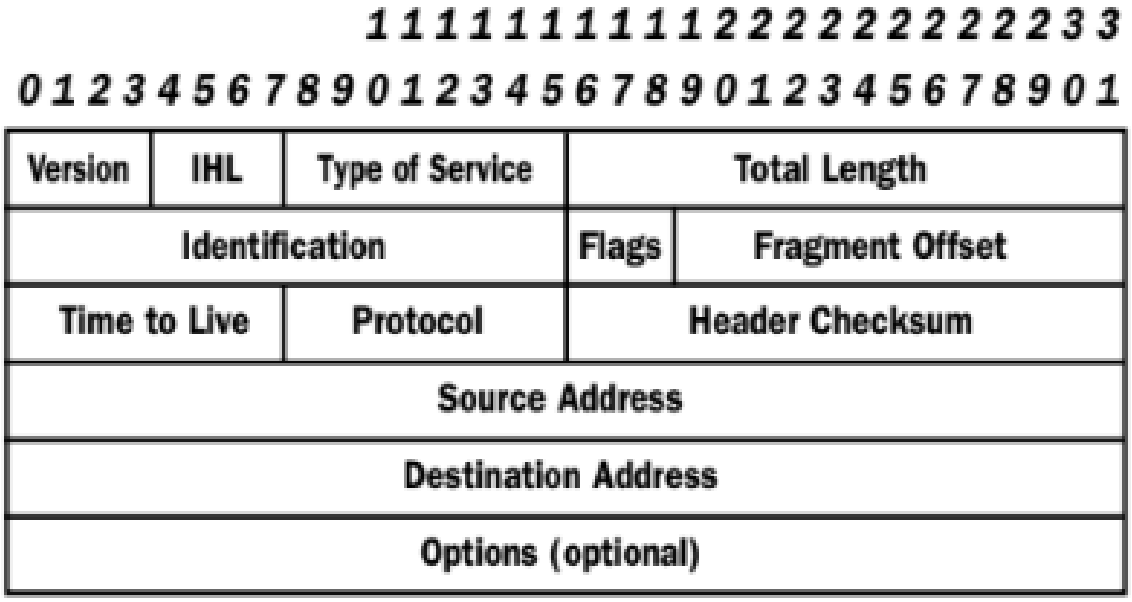

Figura 5: Son los campos de la cabecera IP

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.

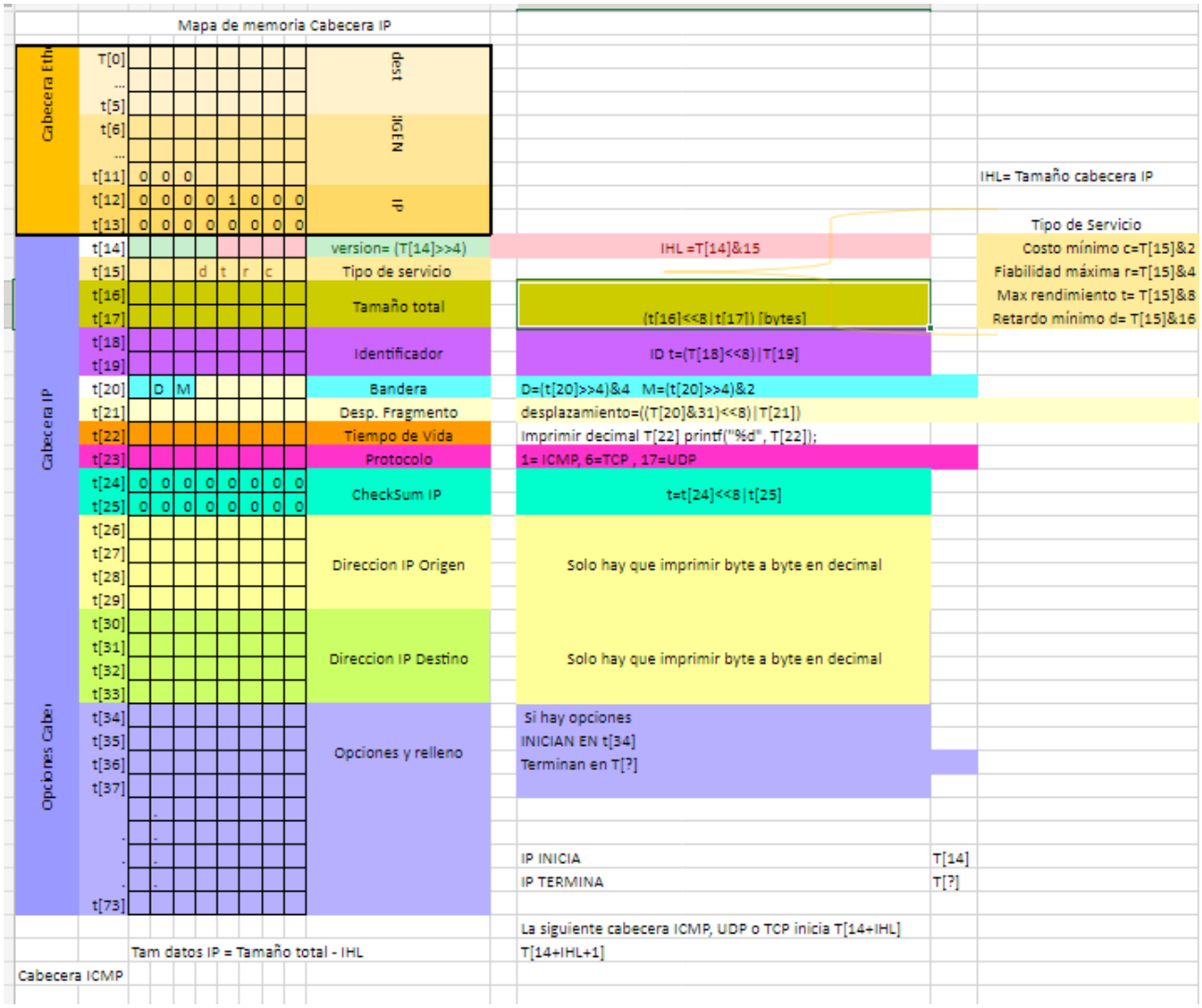

Figura 6: Es el mapa de memoria de la cabecera IP

2. Solución

2.1. Mapa de memoria

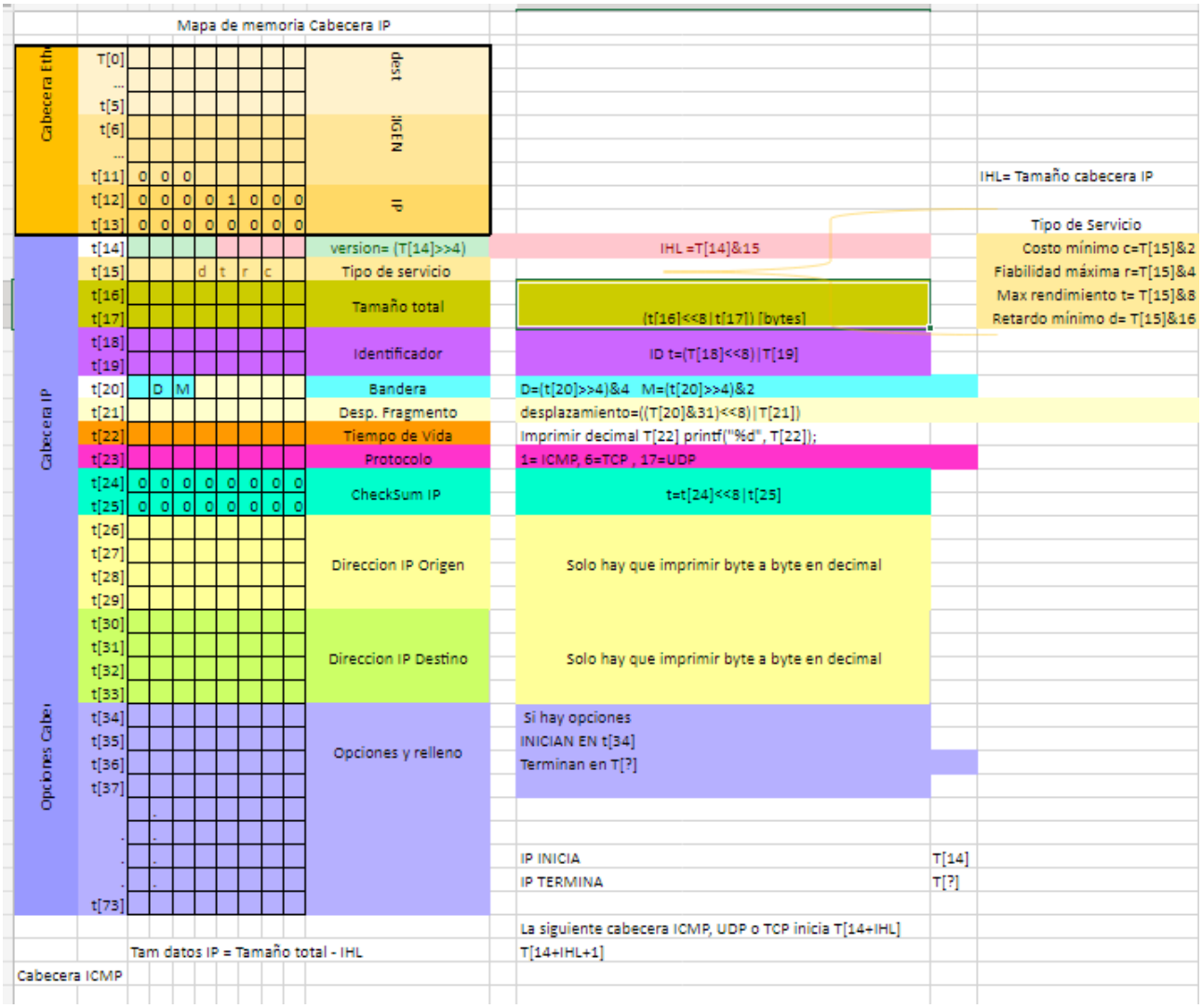

Figura 7: Mapa de memoria

2.2. Correr Programa

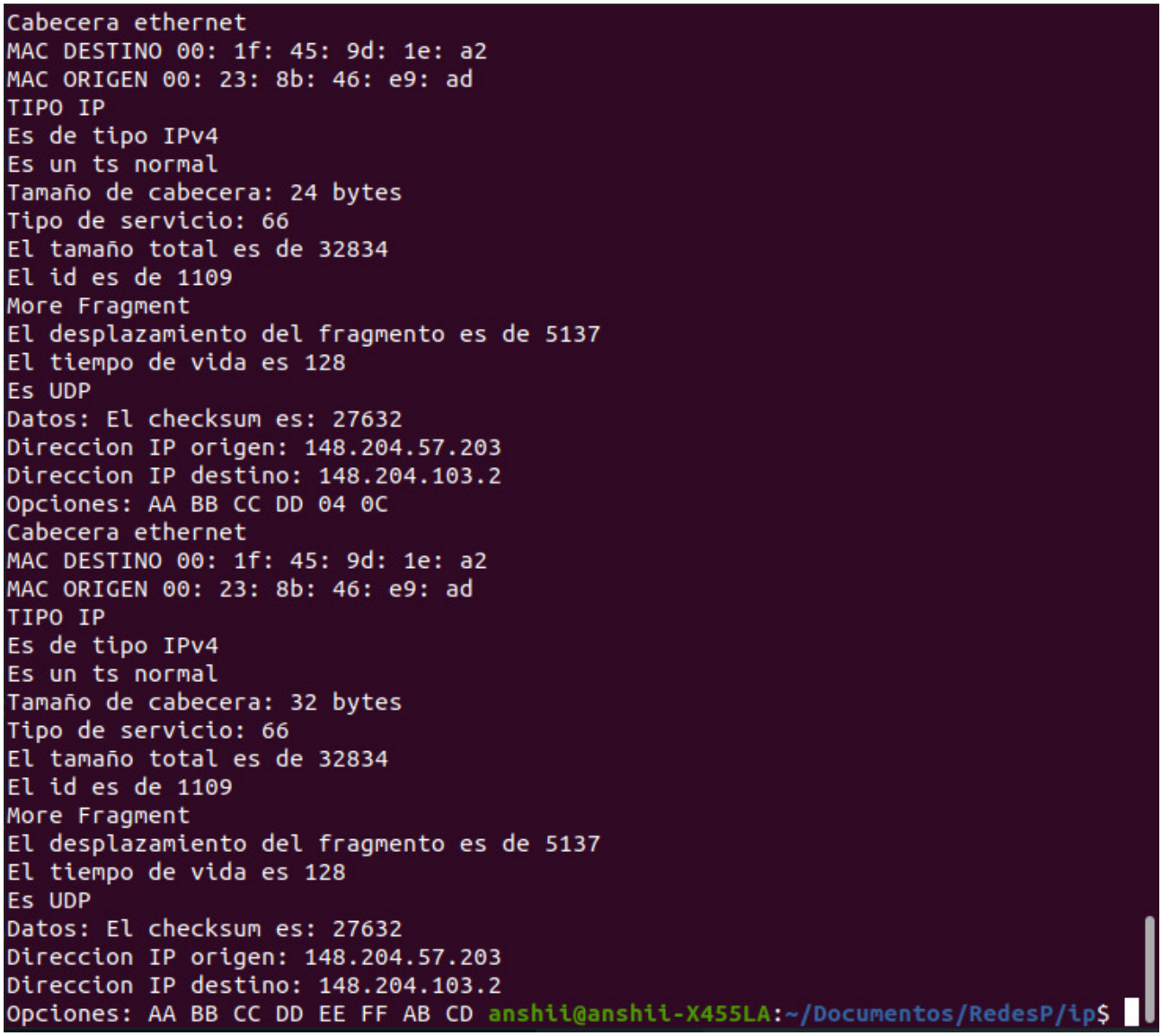

Figura 8: Imprimir las Tramas

3. 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 *);
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;
        hex = 0;
        p++;
      }
      if(hex == 1){
```

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

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

```
164 }
165 void analizaARP(unsigned char *T){
166     if(T[14]<<8 | (T[15] == 1)){
167         printf("TIPO: ARP\n");
168     }else if(T[14]<<8 | (T[15] == 6)){
169         printf("IEEE 80.2 LAN\n");
170     }else{
171         printf("Otro: %d\n", (T[14]<<8 |T[15]));
172     }
173     // Tipo de direccion de Protocolo
174     if (T[16]<<8 | (T[17] == 0x0806)){
175         printf("TIPO: iPv4\n");
176     }else{
177         printf("TIPO: %.2x, %.2x\n", T[16], T[17]);
178     }
179     //Tamano de la MAC
180     printf("Tamano MAC: %d bytes\n", T[18]);
181     //Tamano de la Direccion IP
182     printf("Tamano IP: %d bytes\n", T[19]);
183     //Op Code
184     if (T[20]<<8 | (T[21] == 1)){
185         printf("Op Code: ARP Request\n");
186     }else if (T[20]<<8 | (T[21] == 2)){
187         printf("Op Code: ARP Reply\n");
188     }else{
189         printf("Otro: %d\n", (T[20]<<8 | T[21]));
190     }
191     printf("Direccion MAC origen: %.2x:%.2x:%.2x:%.2x:%.2x:%.2x\n", T[22],
          T[23], T[24], T[25], T[26], T[27]);
192     printf("Direccion IP origen: %d.%d.%d.%d\n", T[28], T[29], T[30], T
        [31]);
193     printf("Direccion MAC destino: %.2x:%.2x:%.2x:%.2x:%.2x:%.2x\n", T
        [32], T[33], T[34], T[35], T[36], T[37]);
194     printf("Direccion IP destino: %d.%d.%d.%d\n", T[38], T[39], T[40], T
        [41]);
195
196 }
197 void analizaIp(unsigned char *T){
198   printf("Es de tipo IPv%d\n", T[14]>>4);
199   if(T[15]&2){
200     printf("Es un ts Costo minimo\n");
201   }else if(T[15]&4){
202     printf("Es un ts fiabilidad \n");
203   }else if(T[15]&8){
204     printf("Es un ts maximo rendimiento\n");
205   }else if(T[15]&16){
206     printf("Es un ts retardo minimo\n");
207   }
208     else{
209         printf("Es un ts normal\n");
210     }
211     //Internet Header Length
212     printf("Tamano de cabecera: %d bytes\n", (T[14]&15) * 4);
213     //Tipo de servicio
214     printf("Tipo de servicio: %d\n", T[17]);
215   printf("El tamano total es de %d\n", T[16]<<8 | T[17]);
216   printf("El id es de %d\n", T[18]<<8 | T[19]);
217   if(T[20]&64){
218     printf("Dont fragment\n");
219   }else if(T[20]&32){
```

```
220     printf("More Fragment\n");
221   }
222   printf("El desplazamiento del fragmento es de %d\n", (T[20]&31)<<8 | T
        [21]);
223   printf("El tiempo de vida es %d\n", T[22]);
224   if(T[23] == 1){
225     printf("Es ICMP\n");
226         //ICMP(T);
227   }else if(T[23] == 6){
228         printf("Es TCP\n");
229         //TCP
230     }else if(T[23] == 17){
231         printf("Es UDP\n");
232     }else{
233         printf("Es otro\n");
234     }
235     //datos
236     printf("Datos: ");
237     printf("El checksum es: %d\n", T[24]<<8 | T[25]);
238   printf("Direccion IP origen: %d.%d.%d.%d\n", T[26], T[27], T[28], T[29])
        ;
239   printf("Direccion IP destino: %d.%d.%d.%d\n", T[30], T[31], T[32], T
        [33]);
240     //opciones IHL
241     printf("Opciones: ");
242     if ((T[14]&15) * 4 > 20 && (T[14]&15) * 4 < 60 ){
243         int l;
244         for (l = 34; l < (T[14]&15) + 34; l++){
245             printf("%.2X ", T[l]);
246         }
247     }else{
248         printf("No hay opciones\n");
249     }
250
251 }
252 void decimal(char c, unsigned char *n){
253   switch(c){
254     case '1':
255       *n +=1;
256     break;
257     case '2':
258       *n +=2;
259     break;
260     case '3':
261       *n +=3;
262     break;
263     case '4':
264       *n +=4;
265     break;
266     case '5':
267       *n +=5;
268     break;
269     case '6':
270       *n +=6;
271     break;
272     case '7':
273       *n +=7;
274     break;
275     case '8':
276       *n +=8;
```

```
277       break;
278       case '9':
279         *n +=9;
280       break;
281       case 'a':
282         *n +=10;
283       break;
284       case 'b':
285         *n +=11;
286       break;
287       case 'c':
288         *n +=12;
289       break;
290       case 'd':
291         *n +=13;
292       break;
293       case 'e':
294         *n +=14;
295       break;
296       case 'f':
297         *n +=15;
298       break;
299       default:
300         *n += 0;
301     }
302 }
```
