Seminario de Redes de Computadoras 66.48

DESARROLLO DE APLICATIVOS CON WINSOCKETS

Docentes: Ing. Marcelo Utard
Ing. Pablo Ronco

Alumnos: Baños, Germán
Gámez, Pablo
Rabino, Juan Pablo
Salas, Federico

Introducción

En el siguiente trabajo nos proponemos lograr una introducción completa a los sockets en entorno Windows para lograr luego generar aplicaciones prácticas de lo aprendido a lo largo del curso.

¿Qué es un socket?

Los sockets son una interfaz de entrada-salida que permite la comunicación entre procesos en el mismo equipo o en equipos diferentes.

Los sockets brindan al programa de aplicación la capacidad de acceder a las funciones del sistema operativo, mediante "system calls". Básicamente en el sentido práctico es una forma de independizar al programador de las capas inferiores de comunicación. Simplemente los sockets se encargan de la comunicación fisica, empaquetamiento de los datagramas, handshaking tcp, etc.

Cada socket que se crea queda totalmente especificado mediante una dirección IP, un protocolo de comunicación (i.e. TCP, UDP) y un número de puerto.

Winsock

En la fig 1 se ve claramente el esquema de funcionamiento de la interfaz de sockets.

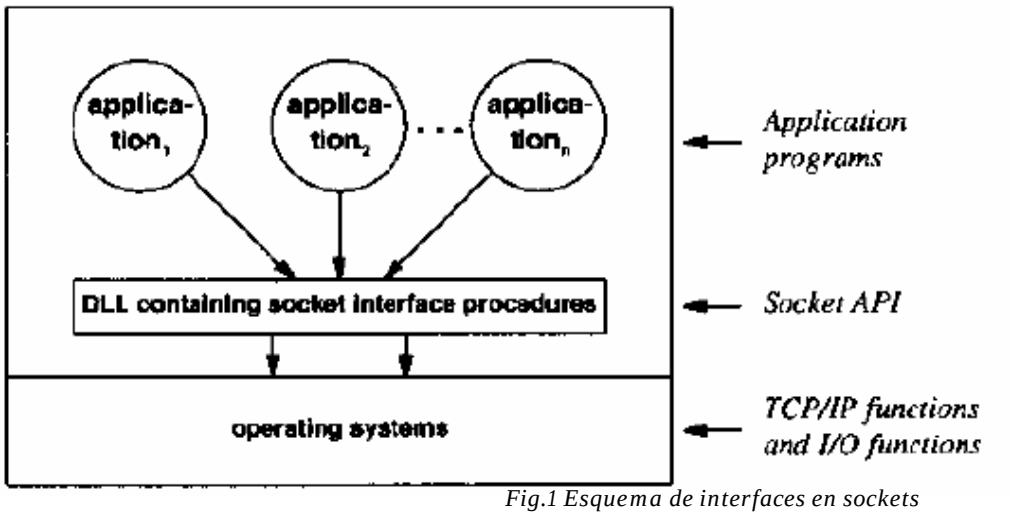


Fig.1 Esquema de interfaces en sockets

En un sentido más práctico cuando se realiza el programa no se incluyen todas las funciones de la API Winsock sino que se incluyen en una biblioteca (DLL) que se carga dinámicamente en tiempo de ejecución. Y varias aplicaciones pueden compartir dichas funciones de biblioteca para no ser cargadas nuevamente con cada programa.

Funciones de la Winsock API

WSAStartup: Inicializa la librería de Windows Sockets. Se produce un enlace a la misma en tiempo de ejecución.

WSACleanup: Se debe ejecutar esta función al terminar el programa para quitar el enlace del proceso aplicativo a la DLL.

Socket: Crea un nuevo socket. Se le pasa el nombre de la familia de protocolos (ej.:TCP/IP) y el tipo de servicio (TCP o UDP). Devuelve un descriptor de archivo.

int socket (int dominio, int tipo, int protocolo);

Connect: Establece una conexión con un servidor remoto. Se le pasa el IP y el puerto de la máquina remota.

int connect (int sock, struct sockaddr *servidor, size_t servidor);

Bind: Asocia al socket una dirección IP y un número de puerto.

int bind (int sock, struct sockaddr * servidor, size_t servidor);

Send: Envía información. Se tiene que especificar descriptor de archivo, dirección de memoria de los datos a enviar, tamaño del buffer y bits para control de transmisión.

int send (int sock, const void *mensaje, size_t mensaje, int flags);

Recv: Recibe información. Es análoga a Send.

int recv (int sock, void *mensaje, size_t mensaje, int flags);

Listen: Acepta nuevas conexiones entrantes al servidor. Necesita como parámetros el tipo de socket (activo=cliente o pasivo=servidor) y tamaño de cola de direcciones entrantes.

int listen (int sock, n_conexiones);

Accept: Acepta una conexión. Se le pasa el descriptor del socket entrante y crea un nuevo socket para cada conexión.

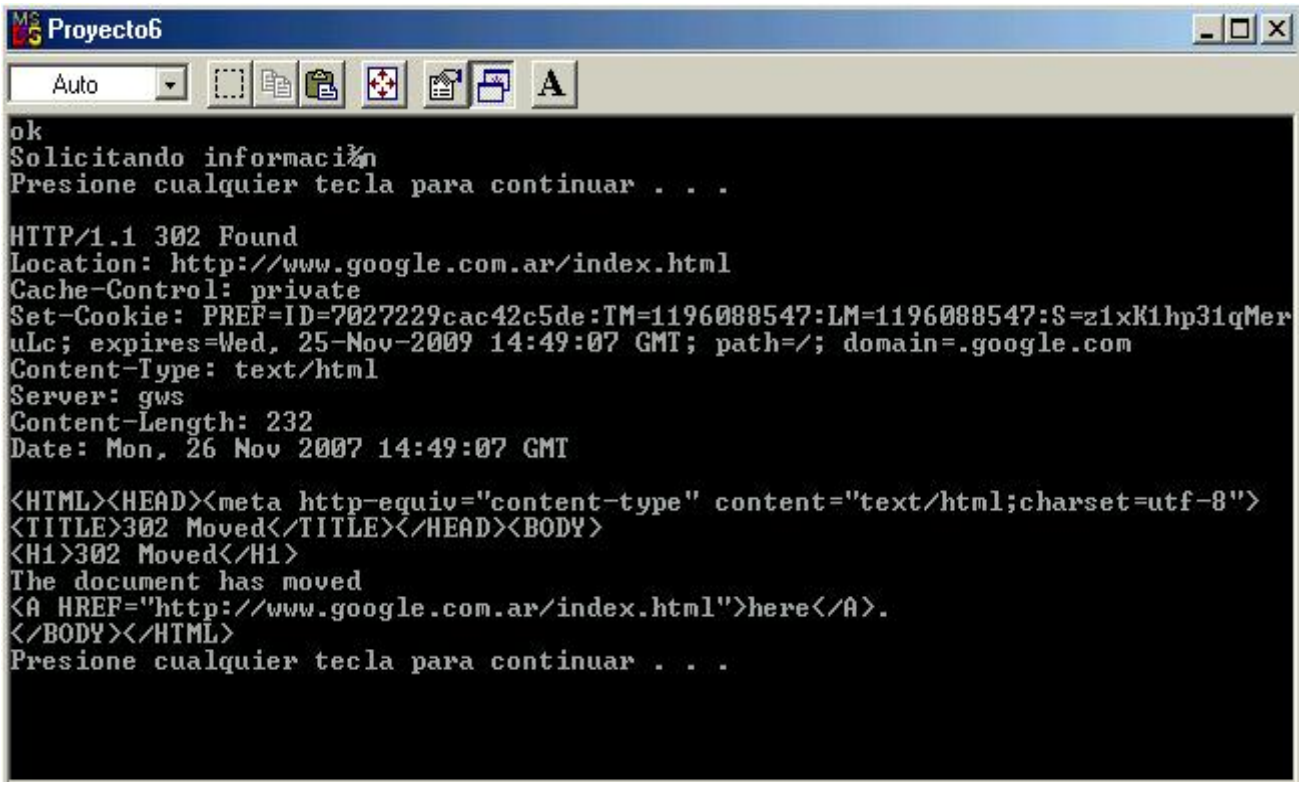
int accept (int sock, struct sockaddr *servidor, inet *tam_sockaddr);

Primera implementación: Cliente HTTP

Se desarrolló una pequeña aplicación capaz de obtener un objeto "index.html" a partir de una dirección IP mediante la conexión con el puerto 80 del Server y el comando GET de HTTP.

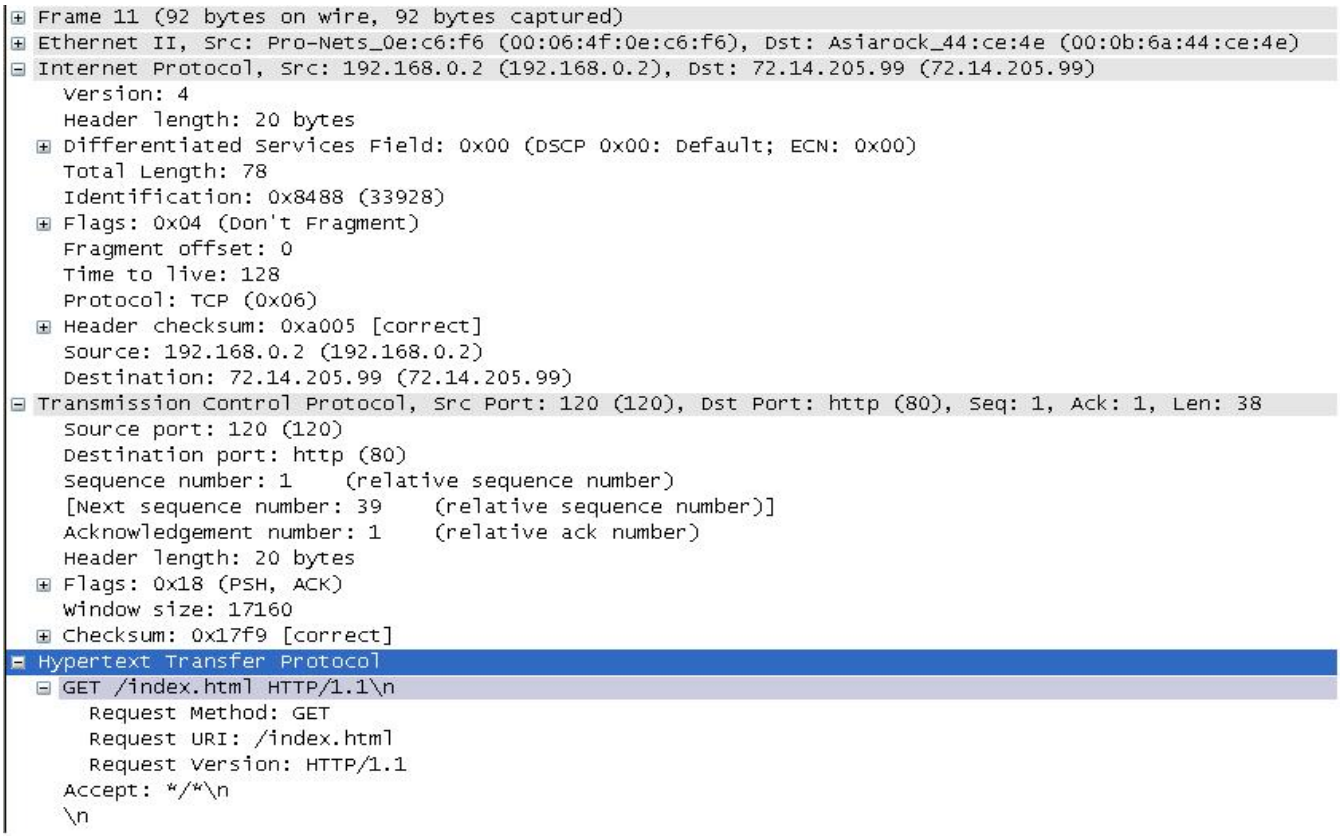
IP: 72.14.205.99 (www.google.com)
Puerto: 80
Comando: GET /index.html HTTP/1.1\nAccept: */*\n\n

Obteniendo lo siguiente:

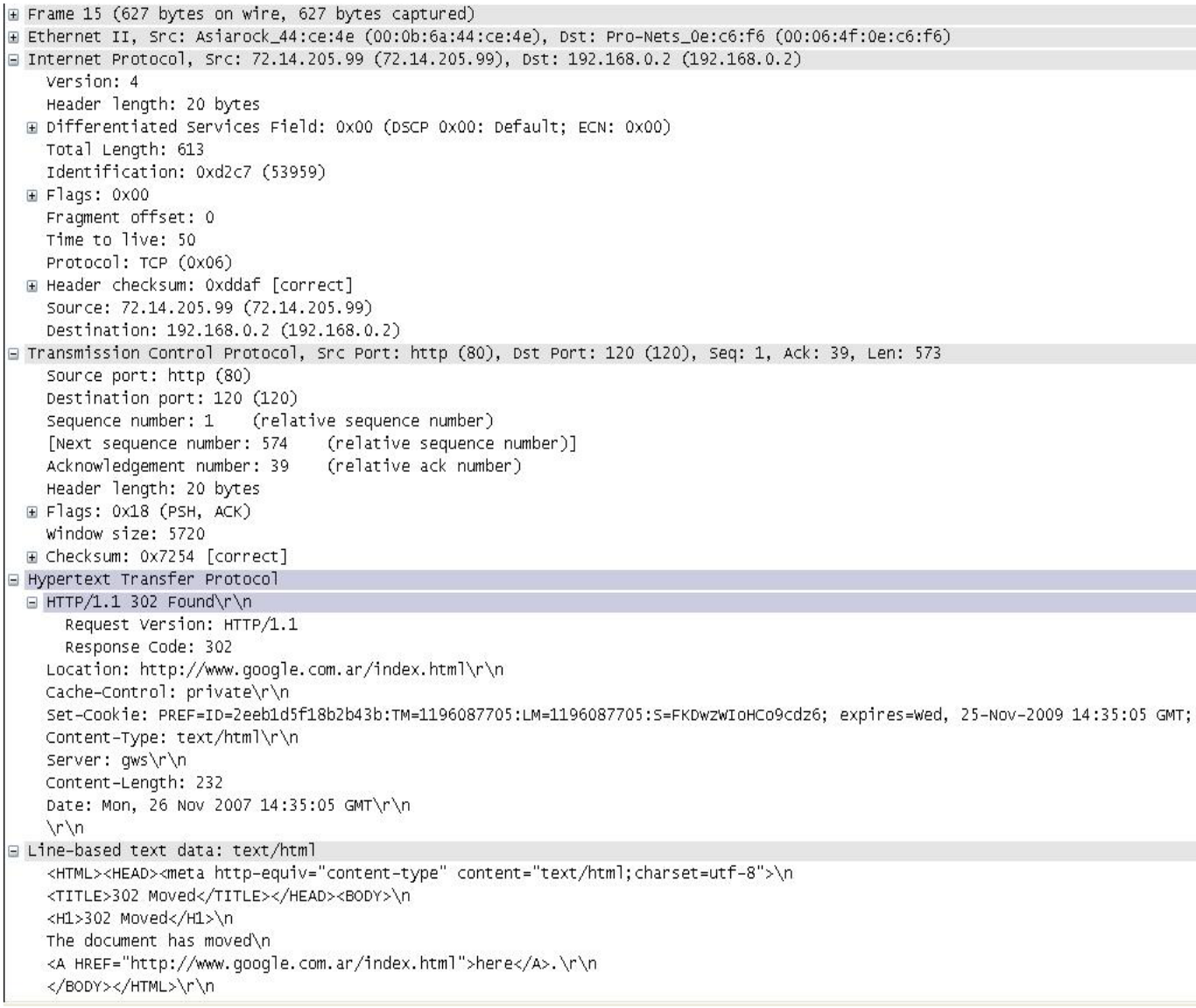


Podemos observar que la pagina nos redirecciona a www.google.com.ar/index.html. Pero de todas maneras la aplicación logró su propósito.

En la siguiente captura podemos ver el datagrama que solicita el objeto index.html



Y luego el datagrama enviado por el servidor redireccionando a www.google.com.ar



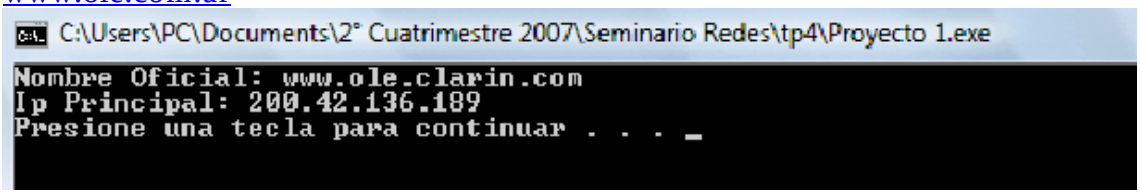
Segunda implementación: DNS

Este programa tomará como argumento por la línea de comandos el nombre de un equipo y nos dirá su nombre oficial y su dirección ip principal, de no encontrarlo nos dará un mensaje de error.

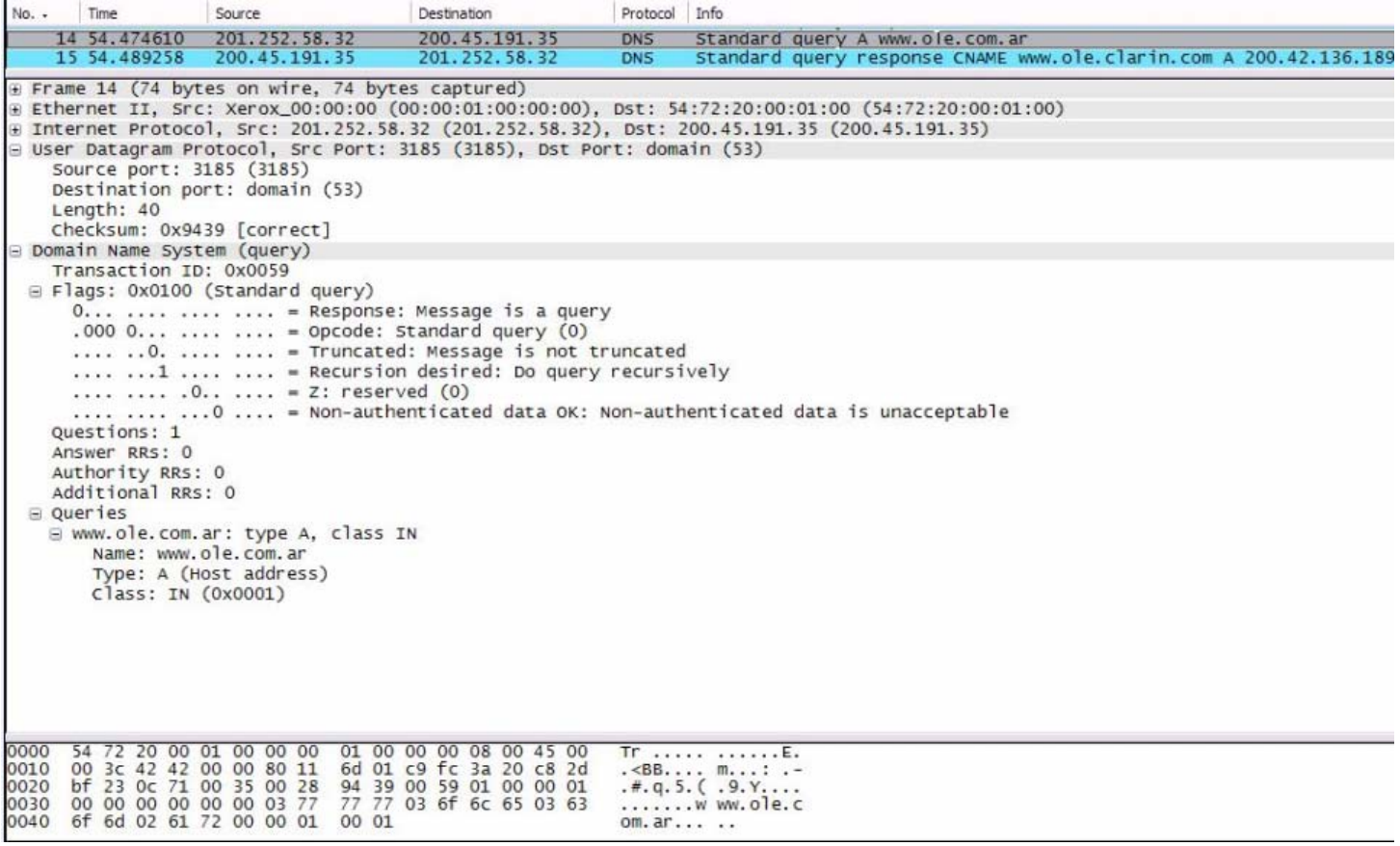
Código fuente:

```c
#include <stdio.h>
#include <stdlib.h>
#include <winsock.h>
#include <windows.h>

int main(int argc, char *argv[])
{
    WSADATA datos;  // datos que usa el winsock
    WORD winsock;  // versión de winsock a utilizar
    struct hostent *info;  // información que obtendremos del equipo

    // Comprobamos que se metan los parametros adecuados
    if(argc!=2)
        printf("Uso: resuelveNombre <nombre_equipo> \n");
    else
    {
        // Usaremos la versión 1.0 de winsock
        winsock=MAKEWORD(1,1);
        // Inicializamos la libreria winsock llamando a WSAStartup
        if((WSAStartup(winsock, &datos))==-1)
            printf("Error: Problemas al inicializar el winsock \n");
        else
    {
        // Obtenemos información del equipo
        info=gethostbyname(argv[1]);
        /* Si el equipo no existe o no se puede resolver la dirección
           lo indicamos, sino visualizamos los datos pedidos: ip + nombre oficial*/
        if(info==NULL)  // es lo mismo que if(!info)
            printf("Error: No se puede resolver \"%s\" \n", argv[1]);
        else
        {
            // Visualizamos los datos almacenados en la estructura info
            printf("Nombre Oficial: %s \n", info->h_name);
            /* La ip está en formato de red, hay que pasarla a formato ASCII con
               la función inet_ntoa */
            printf("Ip Principal: %s \n", inet_ntoa(*((struct in_addr *)info->h_addr)));
        }
        // Cerramos la liberia winsock
        WSACleanup();
    }
}

system("PAUSE");
EXIT_SUCCESS;
```

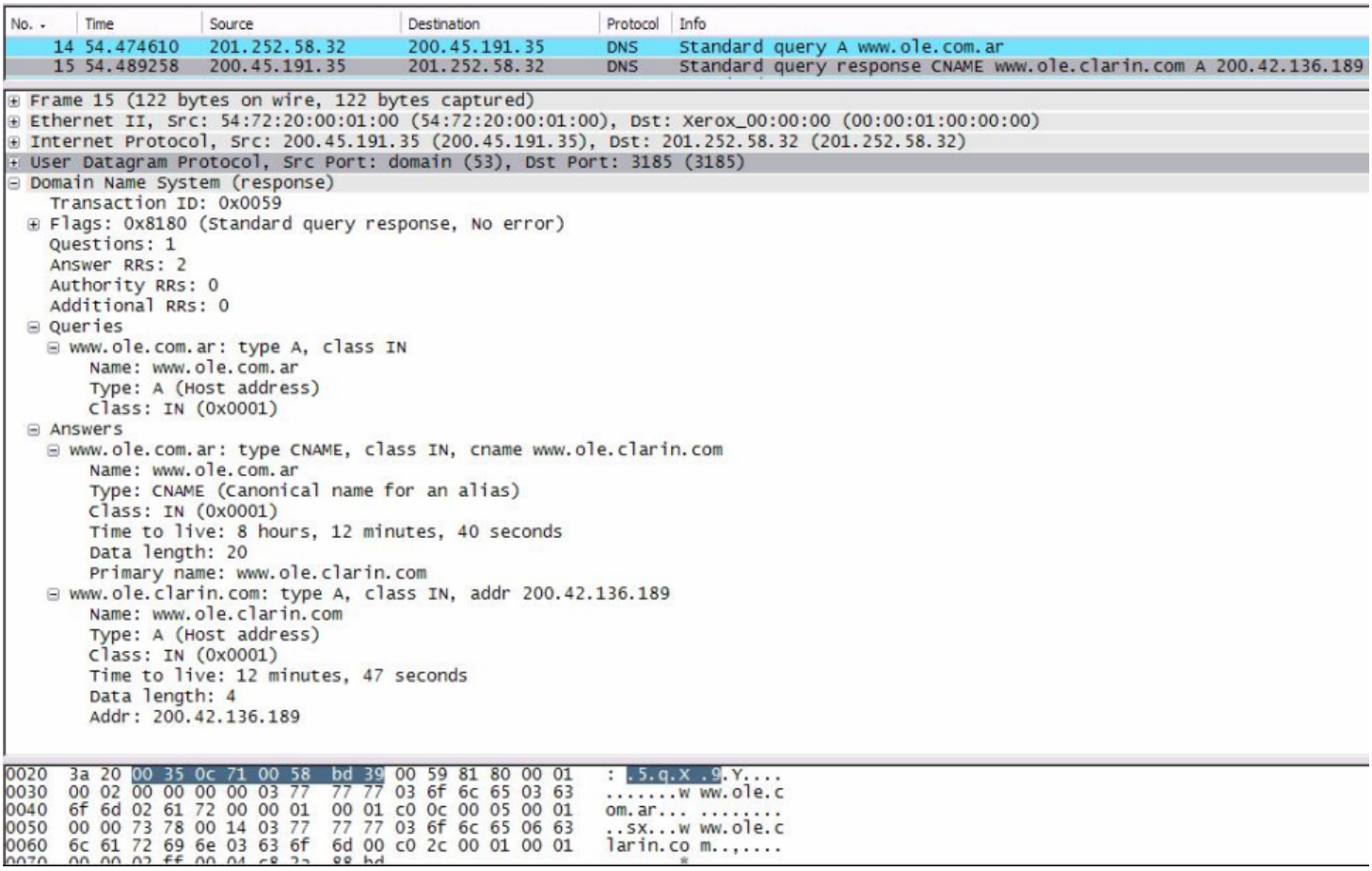
Con lo que obtuvimos el siguiente resultado al ingresar como argumento www.ole.com.ar



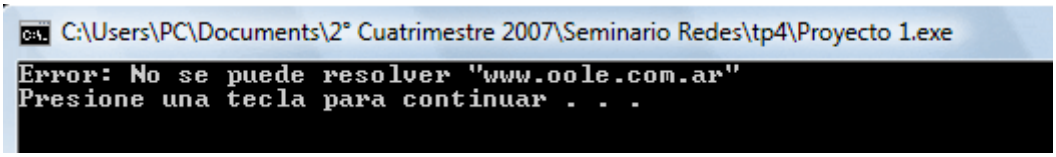
Y en la captura pudimos evidenciar el uso del query DNS



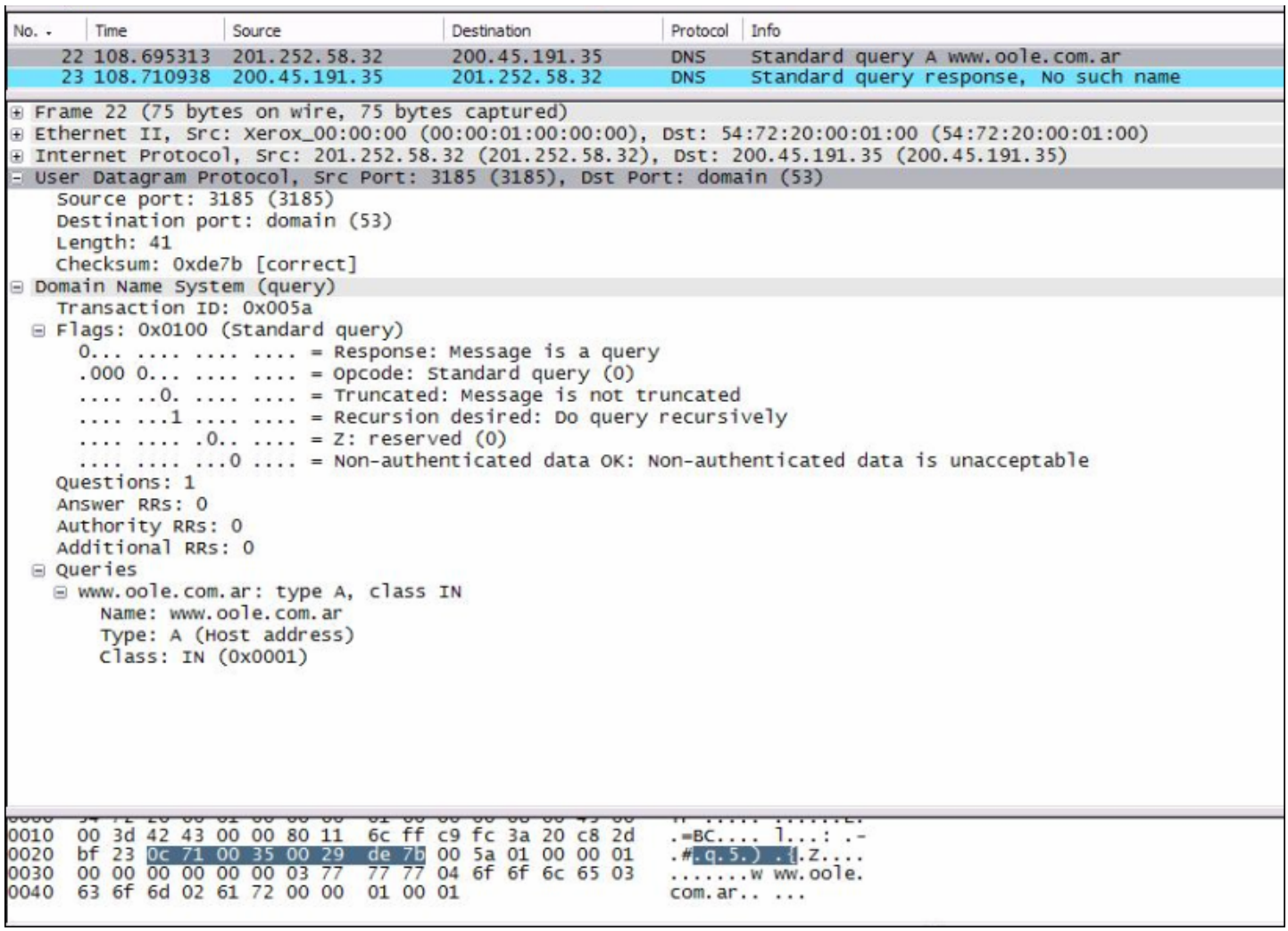
Y la correspondiente respuesta del servidor

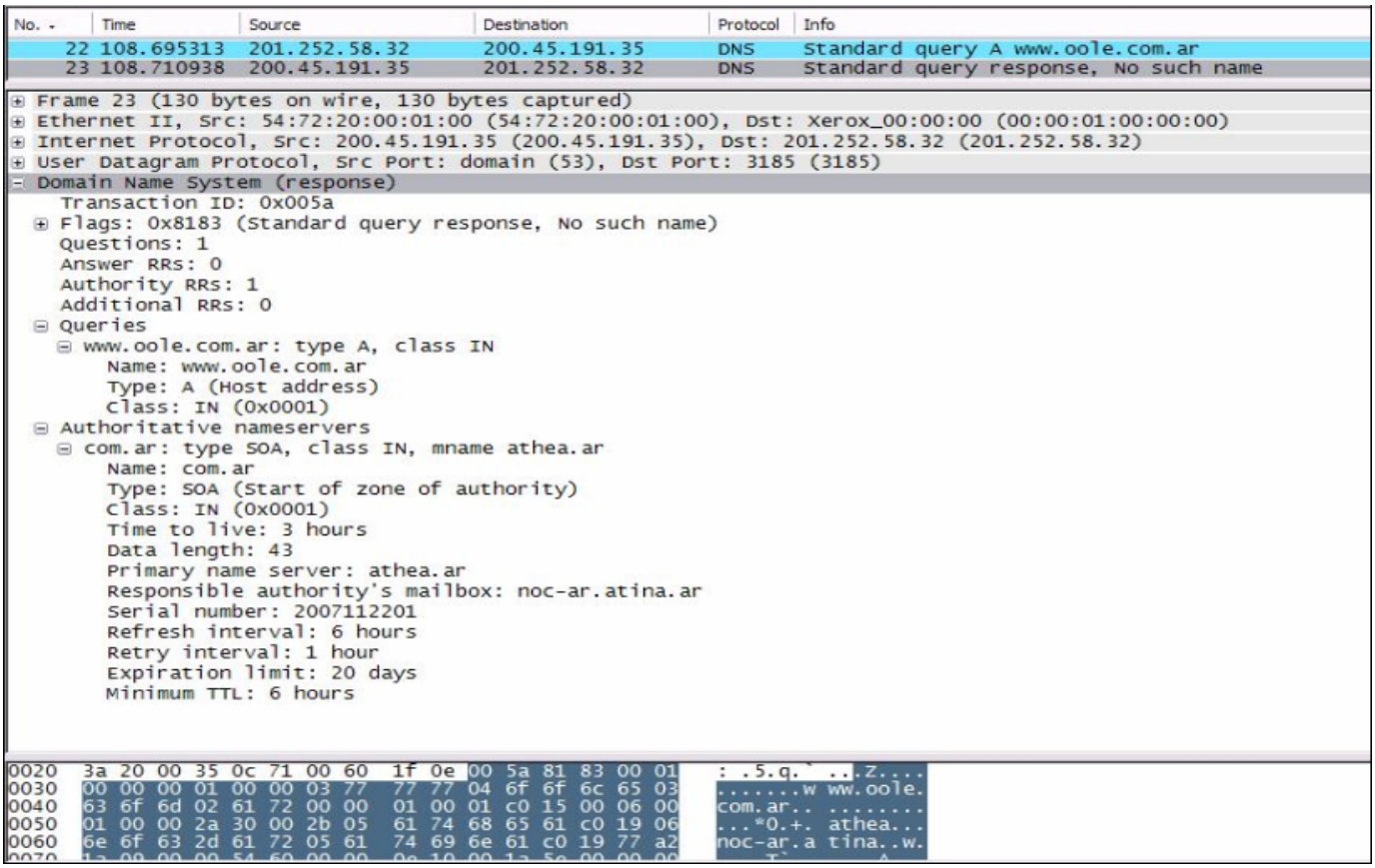


Luego decidimos probar con un caso incorrecto al pasarle el argumento www.oole.coml.ar



Y las capturas





Tercera implementacion: DNS inverso

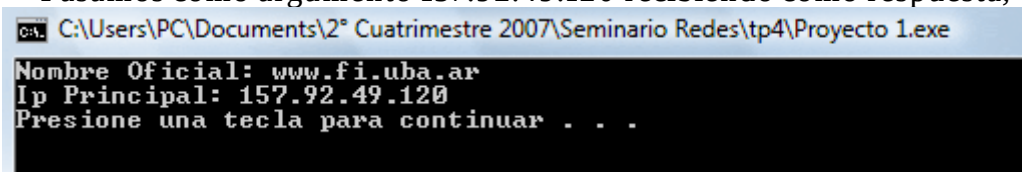
Ahora lo que haremos es la implementación inversa del problema anterior, pasaremos una dirección IP y el programa nos devolverá el nombre del dominio.

Código Fuente:

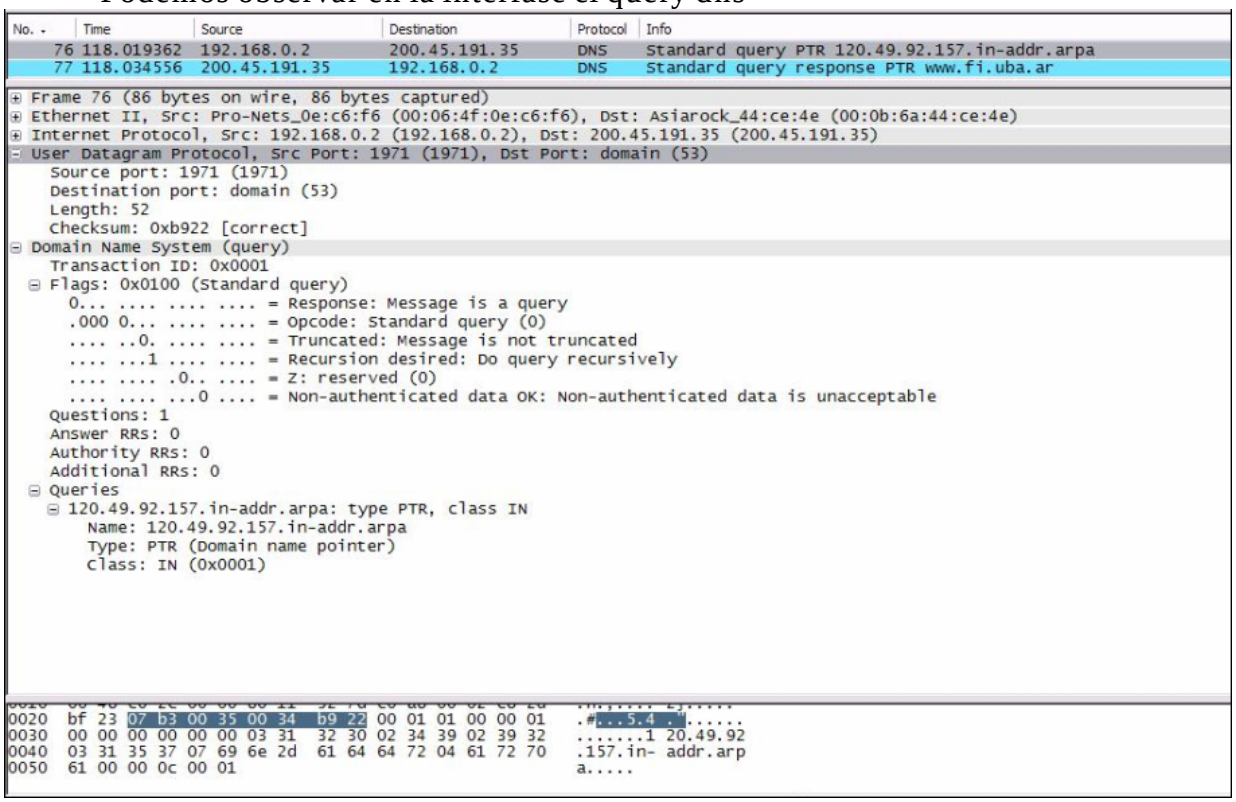
```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <winsock.h>

int main(int argc, char *argv[])
{
    WSADATA datos;  // datos que usa el winsock
    WORD winsock;  // versión de winsock a utilizar
    struct hostent *info;  // información que obtendremos del equipo
    long ip;  // La ip pasada la almacenaremos en formato long

    // Comprobamos que se metan los parametros adecuados
    if(argc!=2)
        printf("Uso: resuelveIp <ip_equipo> \n");
    else
    {
        // Usaremos la versión 1.0 de winsock
        winsock=MAKEWORD(1,1);
        // Inicializamos la libreria winsock llamando a WSAStartup
        if((WSAStartup(winsock, &datos))==-1)
            printf("Error: Problemas al inicializar el winsock \n");
        else
```

```
    {
        /* Pasamos la cadena de caracteres de la ip a formato long. Para
           ello deberemos utilizar la funcion inet_addr */
        ip=inet_addr(argv[1]);
        /* Obtenemos información del equipo con la funcion gethostbyaddr */
        info=gethostbyaddr((char *)&ip, sizeof(ip), AF_INET);

        /* Si el equipo no existe o no se puede resolver la dirección
           lo indicamos, sino visualizamos los datos pedidos: ip + nombre oficial*/
        if(!info)
            printf("Error: No se puede resolver \"%s\" \n", argv[1]);
        else
        {
            // Visualizamos los datos almacenados en la estructura info
            printf("Nombre Oficial: %s \n", info->h_name);
            /* La ip está en formato de red, hay que pasarla a formato ASCII con
               la función inet_ntoa */
            printf("Ip Principal: %s \n", inet_ntoa(*((struct in_addr *)info->h_addr)));
        }
        // Cerramos la liberia winsock
        WSACleanup();
    }
}
```

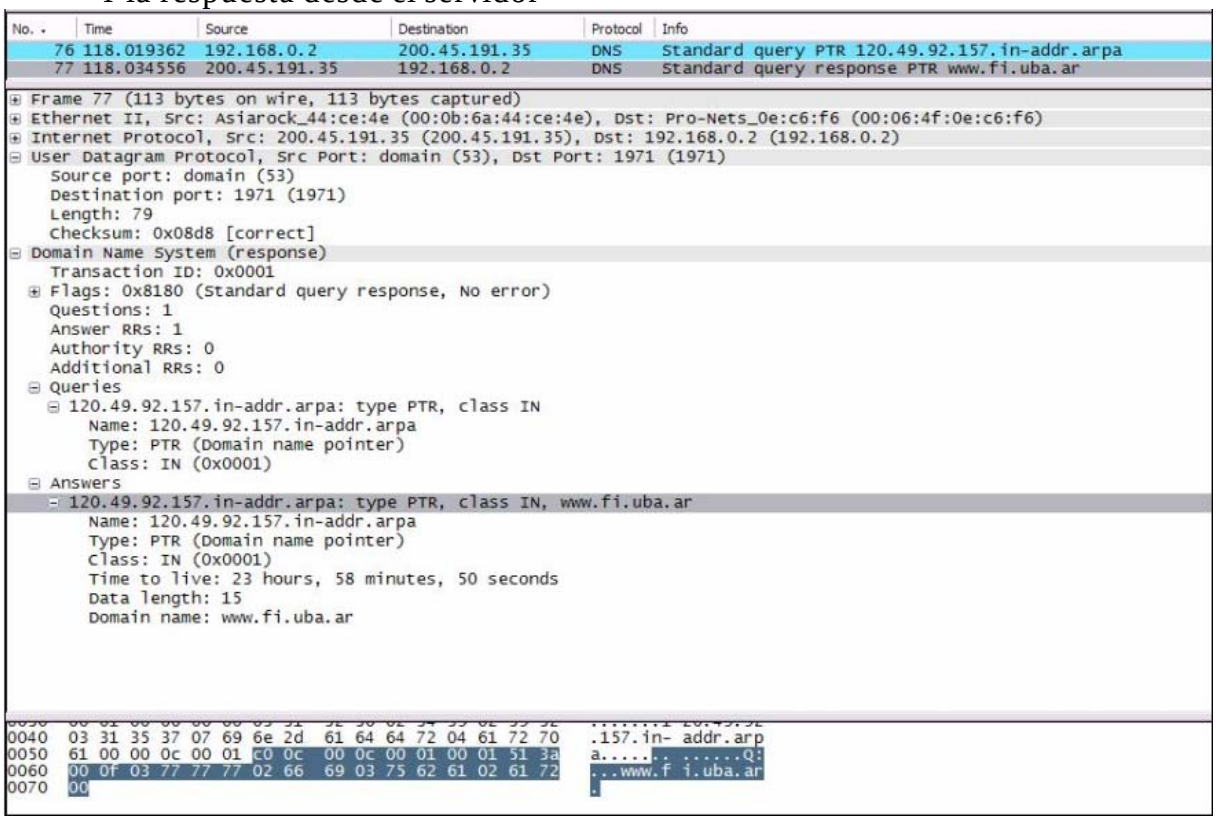
Pasamos como argumento 157.92.49.120 recibiendo como respuesta,



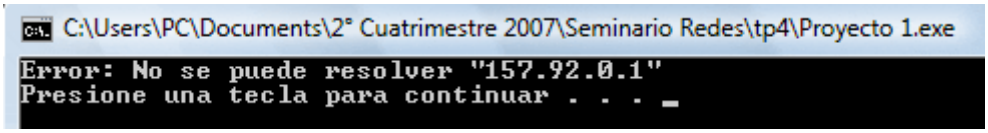
Podemos observar en la interfase el query dns



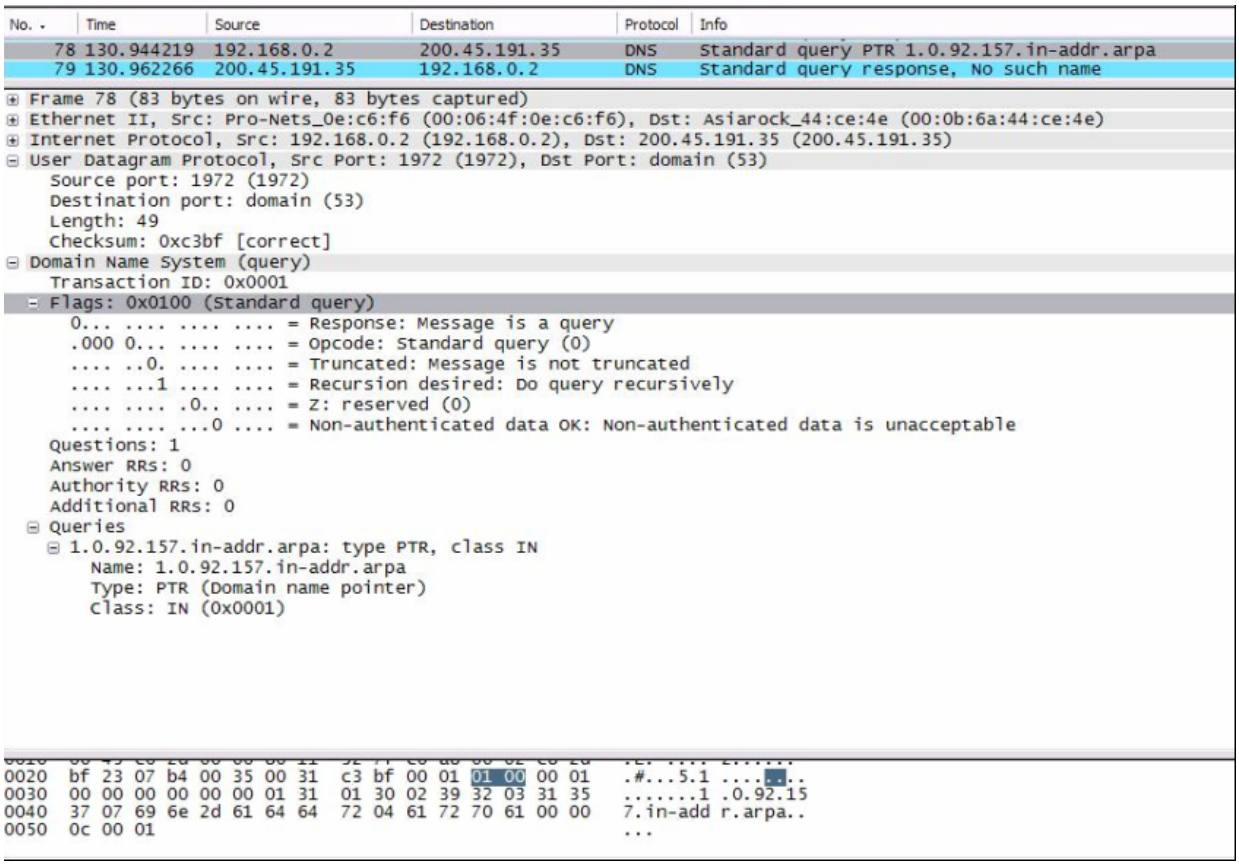
Y la respuesta desde el servidor



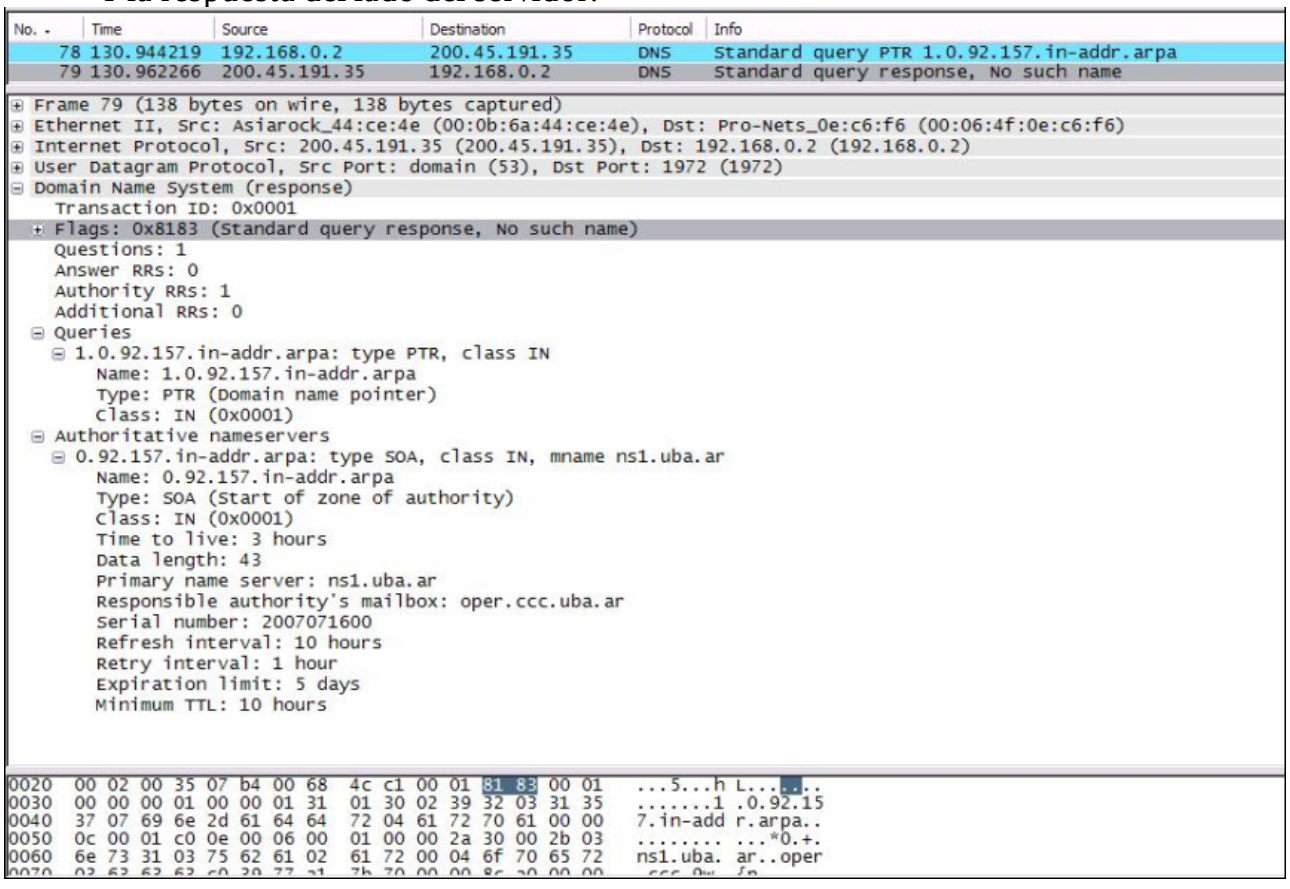
Luego comprobamos el funcionamiento de la aplicación enviando una dirección IP incorrecta.



Podemos observar también las peticiones realizadas



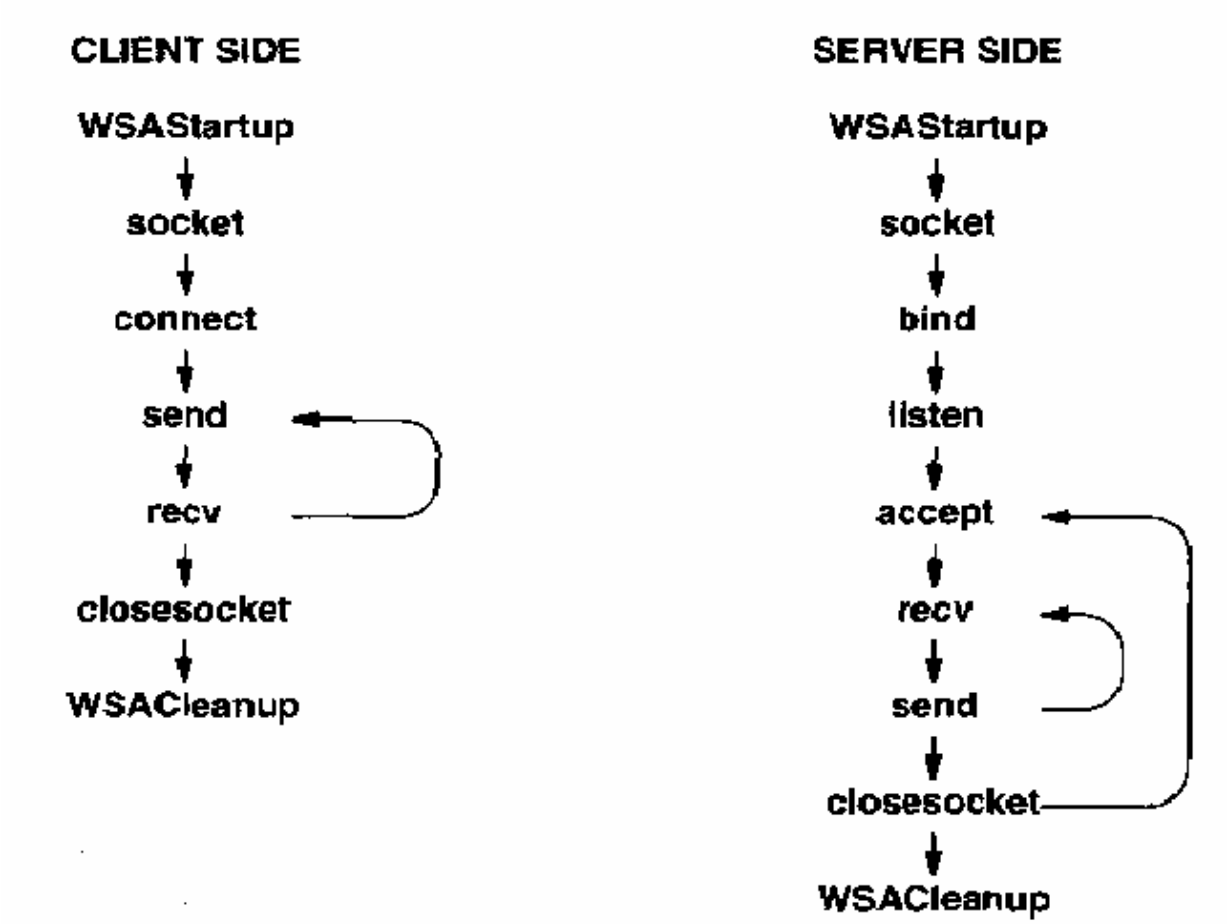
Y la respuesta del lado del servidor.



Cuarta implementación: Aplicación Cliente-Servidor, programa de Chat.

Finalmente nos decidimos a realizar una aplicación cliente y otra servidor para que se comuniquen entre ellas dentro de una LAN para lograr un programa de mensajeo instantáneo.

Primero veamos un esquema del funcionamiento de la parte cliente y servidor.



El servidor realiza entonces las siguientes tareas:

1. Crea un socket de comunicación.
2. Asocia dirección IP y Puerto de conexión al socket.
3. Establece la cola de conexiones entrantes, para atender según orden de llegada.
4. Atiende las conexiones entrantes.
5. Envía y recibe datos.
6. Cierra el socket para terminar la comunicación.

Y el cliente cliente:

1. Crea un socket de comunicación.
2. Se conecta al servidor que esta esperando conexiones entrantes.
3. Envía y recibe datos.
4. Cierra el socket para terminar la comunicación

Las funciones para realizar todas estas tareas fueron especificadas en la introducción a Winsock API.

Código fuente servidor:

servidor.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <winsock.h>
#define PUERTO 6000 // Puerto de conexion
#define IP "192.168.0.2"

int main()
{
  WORD winsock;  // Version del winsock
  WSADATA datos;  // Datos para inicializar winsock
  SOCKET escucha;  // Con este socket aceptamos conexiones entrantes
  SOCKET comunicacion;  // Con este socket enviamos y recibimos datos
  struct sockaddr_in servidor;  // Datos para la conexion
  int tam_sockaddr=sizeof(struct sockaddr);  // Bytes de la estructura sockaddr
  char carac_enviado[50]="";
  char carac_recibido[50]="";

  // Inicializamos winsock
  winsock=MAKEWORD(1,1);
  if(WSAStartup(winsock, &datos))
      printf("Error: Problemas al inicializar winsock \n");
  else
  {
      // Creamos socket.
      escucha=socket(AF_INET, SOCK_STREAM, 0);
      if(escucha==-1)
          printf("Error: Problemas al crear el socket \n");
      else
      {
          // Cubrimos datos estructura sockaddr_in
          servidor.sin_family=AF_INET;
          servidor.sin_port=htons(PUERTO);  // El puerto estará en Formato de Red
          servidor.sin_addr.s_addr=inet_addr(IP);  // La ip estará en formato long

          // Asociamos ip y puerto al socket
          if((bind(escucha, (struct sockaddr *)&servidor, sizeof(servidor)))==-1)
              printf("Erro: Problemas al asociar puerto e ip al socket \n");
          else
          {
              // Establecemos cola de clientes a 1
              listen(escucha, 1);

              // Aceptamos conexiones entrantes
              printf("Esperando conexiones entrantes... \n");
              comunicacion=accept(escucha, (struct sockaddr *)&servidor, &tam_sockaddr);

              // Como no vamos a aceptar más conexiones cerramos el socket escucha
              close(escucha);
```

```
            while(((strcmp("shutdown\n",carac_recibido))&(strcmp("shutdown\n",carac_enviado)))!=0)
            {
             recv (comunicacion, (void *)carac_recibido, 50, 0);
             printf("\nCliente> %s \n", carac_recibido);
             printf("Servidor> ");
             fgets(carac_enviado, 50, stdin);
             send (comunicacion, (const void *)carac_enviado, 50, 0);
            }

            // Cerramos el socket de la comunicacion
            close(comunicacion);
        }
    }
    // Cerramos liberia winsock
    WSACleanup();
}

system("PAUSE");
return (EXIT_SUCCESS);
}
```

Podemos ver como detalle que el servidor necesita abrir 2 sockets, uno para escuchar las conexiones entrantes y el otro para efectuar el envío y la recepción de datos.

Código Fuente Cliente:

[*] cliente.c

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <winsock.h>
#define IP "192.168.0.2"
#define PUERTO 6000     // Puerto de conexión

int main(int argc, char *argv[])
{
  WORD winsock; //Version del winsock
  WSADATA datos;           // Datos Para inicializar el Winsock
  SOCKET comunicacion; // socket de envio y recepcion de datos
  struct sockaddr_in servidor;   //Datos para la conexion
  int estado_conexion;
  int num_enviado=2;
  int num_recibido=4;
  char carac_recibido[50]="";
  char carac_enviado[50]="";

  fflush(stdin);
  // inicializacion del Winsock
  winsock=MAKEWORD(1,1);

  if(WSAStartup(winsock, &datos))
      printf("Error: Problemas al inicializar winsock \n");
  else
  {
      comunicacion=socket(AF_INET, SOCK_STREAM, 0); //Creamos socket de datos
      if(comunicacion==-1)
          printf("Error: Problemas al crear el socket \n");
      else
```

[*] cliente.c

```
        comunicacion=socket(AF_INET, SOCK_STREAM, 0); //Creamos socket de datos
        if(comunicacion==-1)
            printf("Error: Problemas al crear el socket \n");
        else
        {
            // Cubrimos datos de la estructura sockaddr_in
            servidor.sin_family=AF_INET;
            servidor.sin_port=htons(PUERTO); //el puerto estara en formato de Red
            servidor.sin_addr.s_addr=(inet_addr(IP)); // la ip en formato long

            //nos conectamos al servidor

            estado_conexion=connect(comunicacion, (struct sockaddr *)&servidor, sizeof(servidor));
            if(estado_conexion==-1)
                printf("Error: No se puede conectar a \"%s\" \n", IP);

            else
            {
            //inicio de rutina de envio y recepcion de datos

            while(((strcmp("shutdown\n",carac_recibido))&(strcmp("shutdown\n",carac_enviado)))!=0)
                {

                    printf("Cliente> ");
                    fgets(carac_enviado,50,stdin);
                    send(comunicacion, (const void *)carac_enviado, 50, 0);
                    recv(comunicacion, (void *)carac_recibido, 50, 0);
                    printf("Servidor> %s \n", carac_recibido);
                }
```

[*] cliente.c

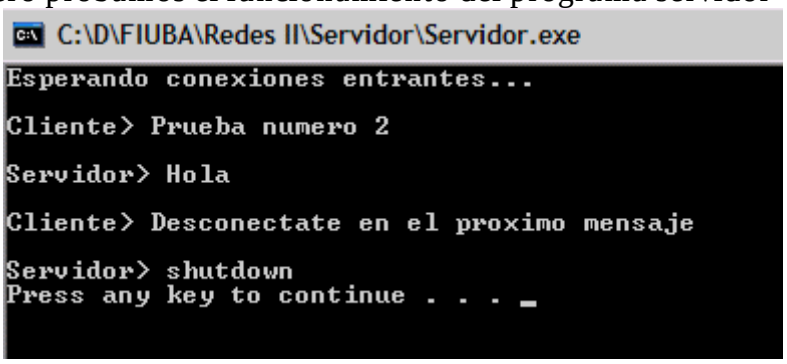
```
            else
            {
            //inicio de rutina de envio y recepcion de datos

            while(((strcmp("shutdown\n",carac_recibido))&(strcmp("shutdown\n",carac_enviado)))!=0)
                {

                    printf("Cliente> ");
                    fgets(carac_enviado,50,stdin);
                    send(comunicacion, (const void *)carac_enviado, 50, 0);
                    recv(comunicacion, (void *)carac_recibido, 50, 0);
                    printf("Servidor> %s \n", carac_recibido);
                }



            }
            close(comunicacion); // cierro el socket


        }

        WSACleanup(); // cierro winsock
    }

    system("PAUSE");
    return (EXIT_SUCCESS);
}
```

La diferencia con el programa servidor, es que el cliente solo necesita un socket mediante el cual enviar y recibir los datos. Pero no necesita un socket para escuchar las conexiones entrantes, ya que no las tiene.

El programa funciona como un “handy” es decir, se envía un mensaje del cliente al servidor y luego del servidor al cliente hasta que uno de los dos corta la comunicación enviando un “shutdown”.

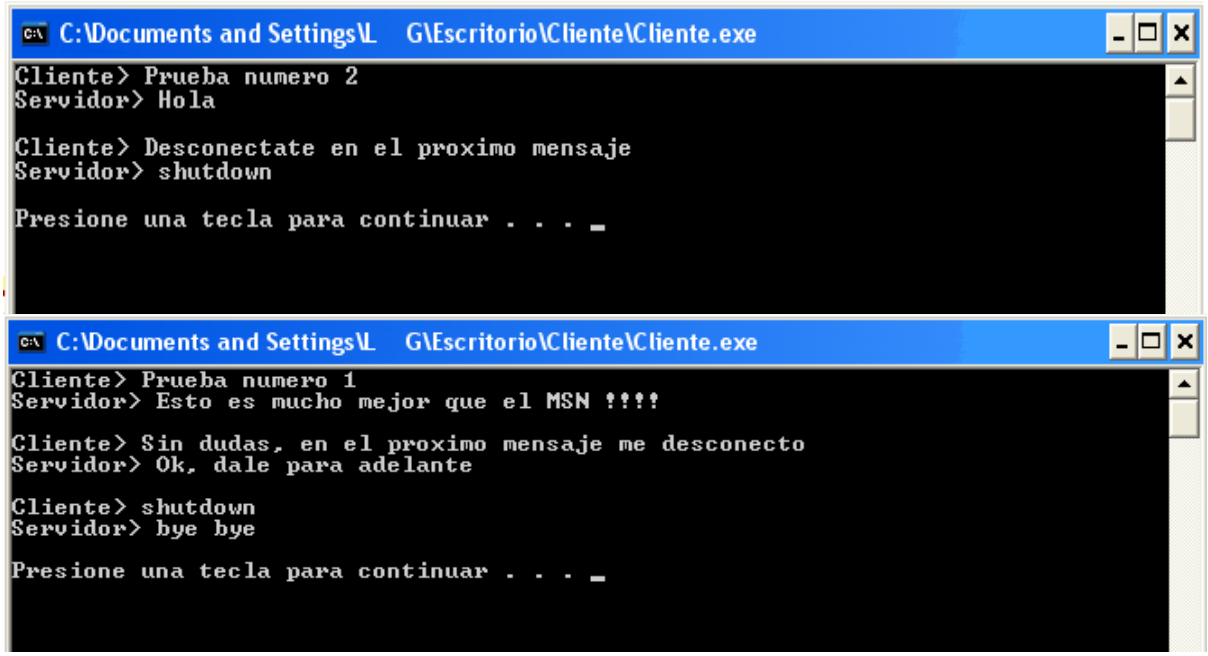
Procedimos luego a comprobar el funcionamiento del aplicativo, realizando las siguientes capturas de pantalla.

Primero probamos el funcionamiento del programa servidor



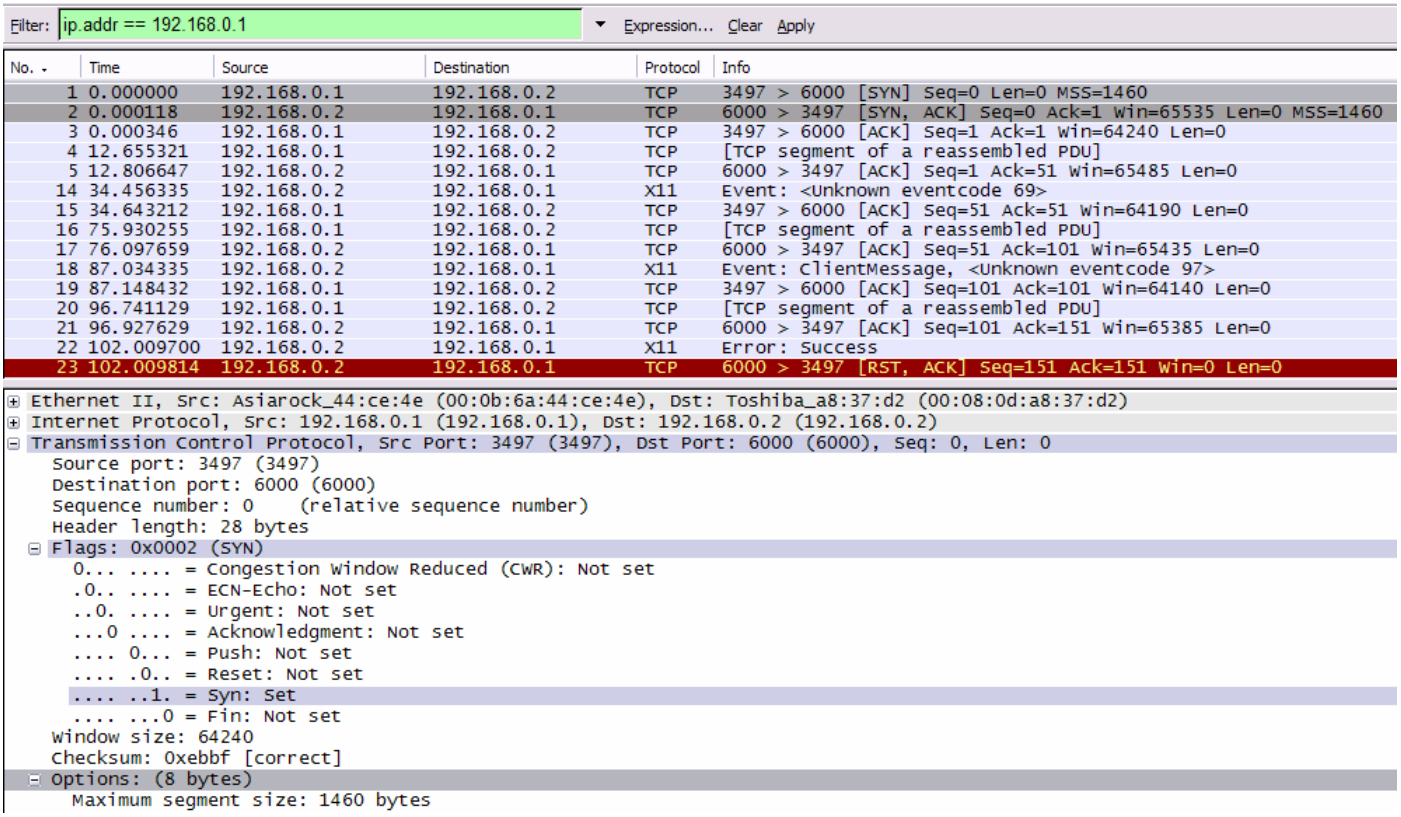
Se ve lo explicado anteriormente, se envía de a 1 mensaje por vez y se termina cuando alguno de los dos escribe “shutdown”.

Ahora veamos el lado cliente



En la primer captura de pantalla podemos ver la desconexión del lado servidor en el cliente y luego una desconexión del cliente del lado del cliente.

Podemos ver un sniffing de la interfase de conexión donde se aprecia el 3-way handshaking asi como la desconexión de parte del cliente.

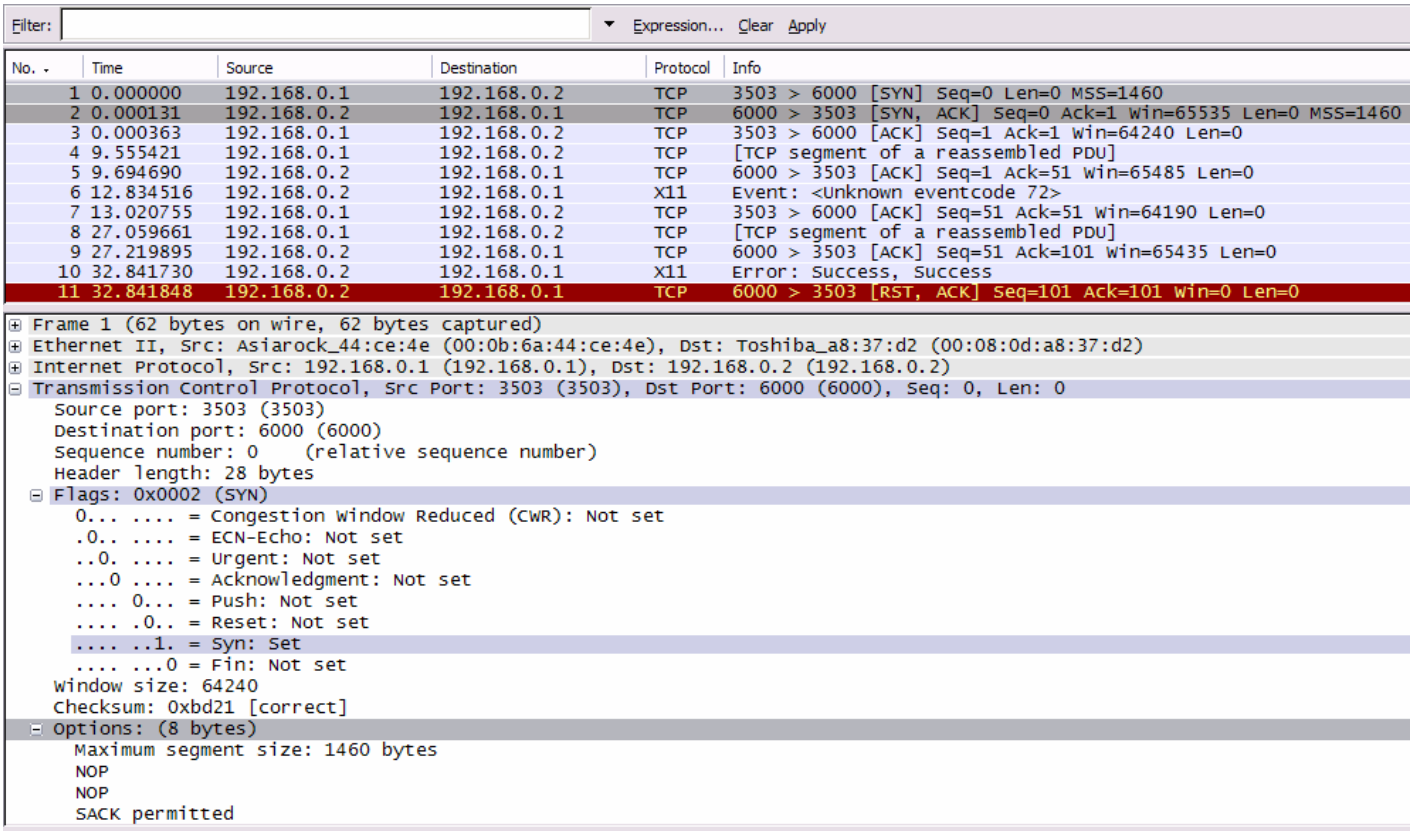


Filter: ip.addr == 192.168.0.1 — Expression… Clear Apply

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.0.1	192.168.0.2	TCP	3497 > 6000 [SYN] Seq=0 Len=0 MSS=1460
2	0.000118	192.168.0.2	192.168.0.1	TCP	6000 > 3497 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
3	0.000346	192.168.0.1	192.168.0.2	TCP	3497 > 6000 [ACK] Seq=1 Ack=1 Win=64240 Len=0
4	12.655321	192.168.0.1	192.168.0.2	TCP	[TCP segment of a reassembled PDU]
5	12.806647	192.168.0.2	192.168.0.1	TCP	6000 > 3497 [ACK] Seq=1 Ack=51 Win=65485 Len=0
14	34.456335	192.168.0.2	192.168.0.1	X11	Event: <Unknown eventcode 69>
15	34.643212	192.168.0.1	192.168.0.2	TCP	3497 > 6000 [ACK] Seq=51 Ack=51 Win=64190 Len=0
16	75.930255	192.168.0.1	192.168.0.2	TCP	[TCP segment of a reassembled PDU]
17	76.097659	192.168.0.2	192.168.0.1	TCP	6000 > 3497 [ACK] Seq=51 Ack=101 Win=65435 Len=0
18	87.034335	192.168.0.2	192.168.0.1	X11	Event: ClientMessage, <Unknown eventcode 97>
19	87.148432	192.168.0.1	192.168.0.2	TCP	3497 > 6000 [ACK] Seq=101 Ack=101 Win=64140 Len=0
20	96.741129	192.168.0.1	192.168.0.2	TCP	[TCP segment of a reassembled PDU]
21	96.927629	192.168.0.2	192.168.0.1	TCP	6000 > 3497 [ACK] Seq=101 Ack=151 Win=65385 Len=0
22	102.009700	192.168.0.2	192.168.0.1	X11	Error: Success
23	102.009814	192.168.0.2	192.168.0.1	TCP	6000 > 3497 [RST, ACK] Seq=151 Ack=151 Win=0 Len=0

```
⊞ Ethernet II, Src: Asiarock_44:ce:4e (00:0b:6a:44:ce:4e), Dst: Toshiba_a8:37:d2 (00:08:0d:a8:37:d2)
⊞ Internet Protocol, Src: 192.168.0.1 (192.168.0.1), Dst: 192.168.0.2 (192.168.0.2)
⊟ Transmission Control Protocol, Src Port: 3497 (3497), Dst Port: 6000 (6000), Seq: 0, Len: 0
    Source port: 3497 (3497)
    Destination port: 6000 (6000)
    Sequence number: 0    (relative sequence number)
    Header length: 28 bytes
  ⊟ Flags: 0x0002 (SYN)
      0... .... = Congestion Window Reduced (CWR): Not set
      .0.. .... = ECN-Echo: Not set
      ..0. .... = Urgent: Not set
      ...0 .... = Acknowledgment: Not set
      .... 0... = Push: Not set
      .... .0.. = Reset: Not set
      .... ..1. = Syn: Set
      .... ...0 = Fin: Not set
    Window size: 64240
    Checksum: 0xebbf [correct]
  ⊟ Options: (8 bytes)
      Maximum segment size: 1460 bytes
```

Y ahora con una prueba similar desde el lado servidor



Filter: — Expression… Clear Apply

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.0.1	192.168.0.2	TCP	3503 > 6000 [SYN] Seq=0 Len=0 MSS=1460
2	0.000131	192.168.0.2	192.168.0.1	TCP	6000 > 3503 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
3	0.000363	192.168.0.1	192.168.0.2	TCP	3503 > 6000 [ACK] Seq=1 Ack=1 Win=64240 Len=0
4	9.555421	192.168.0.1	192.168.0.2	TCP	[TCP segment of a reassembled PDU]
5	9.694690	192.168.0.2	192.168.0.1	TCP	6000 > 3503 [ACK] Seq=1 Ack=51 Win=65485 Len=0
6	12.834516	192.168.0.2	192.168.0.1	X11	Event: <Unknown eventcode 72>
7	13.020755	192.168.0.1	192.168.0.2	TCP	3503 > 6000 [ACK] Seq=51 Ack=51 Win=64190 Len=0
8	27.059661	192.168.0.1	192.168.0.2	TCP	[TCP segment of a reassembled PDU]
9	27.219895	192.168.0.2	192.168.0.1	TCP	6000 > 3503 [ACK] Seq=51 Ack=101 Win=65435 Len=0
10	32.841730	192.168.0.2	192.168.0.1	X11	Error: Success, Success
11	32.841848	192.168.0.2	192.168.0.1	TCP	6000 > 3503 [RST, ACK] Seq=101 Ack=101 Win=0 Len=0

```
⊞ Frame 1 (62 bytes on wire, 62 bytes captured)
⊞ Ethernet II, Src: Asiarock_44:ce:4e (00:0b:6a:44:ce:4e), Dst: Toshiba_a8:37:d2 (00:08:0d:a8:37:d2)
⊞ Internet Protocol, Src: 192.168.0.1 (192.168.0.1), Dst: 192.168.0.2 (192.168.0.2)
⊟ Transmission Control Protocol, Src Port: 3503 (3503), Dst Port: 6000 (6000), Seq: 0, Len: 0
    Source port: 3503 (3503)
    Destination port: 6000 (6000)
    Sequence number: 0    (relative sequence number)
    Header length: 28 bytes
  ⊟ Flags: 0x0002 (SYN)
      0... .... = Congestion Window Reduced (CWR): Not set
      .0.. .... = ECN-Echo: Not set
      ..0. .... = Urgent: Not set
      ...0 .... = Acknowledgment: Not set
      .... 0... = Push: Not set
      .... .0.. = Reset: Not set
      .... ..1. = Syn: Set
      .... ...0 = Fin: Not set
    Window size: 64240
    Checksum: 0xbd21 [correct]
  ⊟ Options: (8 bytes)
      Maximum segment size: 1460 bytes
      NOP
      NOP
      SACK permitted
```

Finalmente como corolario podemos ver un paquete de texto enviado desde el cliente al servidor, mostrando que los mensajes iban encapsulados en modo texto.