UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE ELECTRÓNICA

Análisis Experimental de la Transmisión de Datos

ELO322 – Redes de Computadores I

Pablo Ahumada D.

Jorge Cápona G.

Resumen

Se muestra un análisis realizado a una conexión TCP, tratando de identificar los métodos de control de congestión TCP, a través del software Wireshark. También se muestran algunos contenidos vistos en clases, en relación a una comunicación TCP, como por ejemplo cabeceras, métodos de conexión, entre otros.

Introducción

Durante el curso, se ha estudiado las características del protocolo TCP de la capa de transporte. Las características de este protocolo que lo diferencian de UDP son medidas de control para asegurar una conexión que garantice que todos los paquetes enviados lleguen a destino, y para regular el tráfico en la red.

En este trabajo se analizará en la práctica algunas características de TCP. Mediante programación en lenguaje C, se ha trabajado sobre un programa, con la estructura cliente/servidor, donde el cliente envía un string de tamaño variable (Dependiendo del usuario) al host que corre el programa servidor. De esta manera, y utilizando el software Wireshark, se analizarán las características de velocidad, de corrección de paquetes perdidos. También, se utilizará el software para mostrar otras características de una red, como por ejemplo, el tiempo de ronda completa, RTT, diagramas de envío y recepción, 3 way handshake, y fin de una conexión.

El objetivo del presente trabajo es realizar un análisis práctico de los contenidos vistos en clases, explorando las características de una conexión TCP.

Análisis con Wireshark

Los análisis realizados se han hecho utilizando el software Wireshark. Este software permite el análisis de protocolos a través de la red. Posee una interfaz gráfica amigable y fácil de utilizar.

En el caso del protocolo TCP, se utilizó mucho las utilidades que este software provee. Wireshark, en su pantalla principal, muestra todos los mensajes, enviados y recibidos por el computador, y entrega el tiempo respecto al inicio de toma de muestra del evento, la IP de la fuente y de destino, el protocolo utilizado, y otros datos. En este caso se observan los comandos SYN, ACK, Win, Len, MSS. Win corresponde al tamaño de ventana que se informan entre hosts; Len corresponde al tamaño del paquete en bytes; MSS corresponde al tamaño máximo de segmento de datos, que en este caso sólo se informa al establecer la comunicación.

a) b)

Figura 1: Diagrama de flujos de una conexión TCP. a) Establecimiento de conexión. b) Final de conexión.

En la figura 1-a) se observa un diagrama de flujos del comienzo de una conexión TCP. Se puede ver el acuerdo "3 way handshake" en las 3 primeras líneas, para luego dar comienzo a la comunicación y envío de paquetes. Lo mismo se observa en la figura 1-b) para observar cómo termina la comunicación TCP entre 2 hosts.

Una de las características que entrega Wireshark es el Throughput de la conexión. Este término hace referencia a la capacidad de un enlace para transportar información útil.

```
Transmission Control Protocol, Src Port: smartpackets (3218), Dst Port: distinct (9999), Seq: 1687201, Ack: 1, Len: 1406
    Source port: smartpackets (3218)
    Destination port: distinct (9999)
    Sequence number: 1687201    (relative sequence number)
    [Next sequence number: 1688607    (relative sequence number)]
    Acknowledgement number: 1    (relative ack number)
    Header length: 20 bytes
  Flags: 0x10 (ACK)
      0... .... = Congestion Window Reduced (CWR): Not set
      .0.. .... = ECN-Echo: Not set
      ..0. .... = Urgent: Not set
      ...1 .... = Acknowledgment: Set
      .... 0... = Push: Not set
      .... .0.. = Reset: Not set
      .... ..0. = Syn: Not set
      .... ...0 = Fin: Not set
    Window size: 65535
  Checksum: 0x8555 [correct]
  [SEQ/ACK analysis]
Data (1406 bytes)
    Data: 333435363738393031323334353637383930313233343536...
```

Figura 2: Segmento TCP en Wireshark

Wireshark también permite analizar de forma completa los segmentos TCP e IP, entre otros. Se muestra en la figura 2 como Wireshark entrega información sobre un segmento TCP.

Tipos de control de congestión para TCP

TCP implementa varios métodos de control para mejorar el desempeño de TCP y asegurar un traspaso confiable de información.

- **Control de Flujo**: receptor envía a emisor su tamaño de ventana para regular la cantidad de paquetes enviados y no provocar pérdidas de datos por desborde en buffer.
- **Control de Congestión**: tiene que ver, no sólo con el host receptor, sino que con toda la red. Realiza acciones que aseguran no congestionarla:

- Slow Start: inicio lento. Tasa aumenta exponencialmente hasta valor umbral, luego sigue aumentando de forma lineal. Se aplica al comienzo de transmisión y después de una supuesta pérdida de paquete por timeout.
- TCP AIMD: Aumento Aditivo, Decrecimiento Multiplicativo: aumento aditivo se refiere a que la ventana de congestión aumenta en un 1mss cada RTT en ausencia de pérdidas. Decrecimiento multiplicativo hace que la ventana de congestión se reduzca a la mitad después de pérdida.

Pruebas Realizadas

Comportamiento de la red ante uso de TCP.

Se configura el programa para poder enviar una secuencia de caracteres de distinta longitud. Se analizarán las distintas comunicaciones y se tratará de identificar si existen métodos de control de congestión, y otros detalles que pueden ser observados con el software Wireshark.

1. Conexión cerrada súbitamente por receptor.

Se realiza un envío. De forma súbita, el receptor se desconecta de internet. Esta situación es similar a una de congestión extrema, ya que simula la pérdida de ACK. Realizando un análisis, se puede observar que una vez que el receptor se desconecta, siguen llegando ACK, por lo que ventana sigue aumentando, y se siguen enviando paquetes. Una vez que dejan de llegar los ACK, el emisor deja de enviar paquetes, y realiza re-envíos del último paquete recibido.

2. 100 envíos de 200 KB.

Se envían 100 veces una secuencia de caracteres de 200 KB, cada 100 milisegundos. Con esto se busca crear una situación de congestión en la red y en servidor. Se observa que la relación, a simple vista entre el número de secuencia del segmento enviado es lineal con el tiempo.

En la figura 3.d) se observa un acercamiento a la parte inicial del gráfico 3.b). La partida lenta TCP provoca que el gráfico tenga forma exponencial para los primeros paquetes. Esto se puede observar, pero durante un período de tiempo muy pequeño. Luego se ve una zona lineal. También se puede ver que la gráfica no es continua.

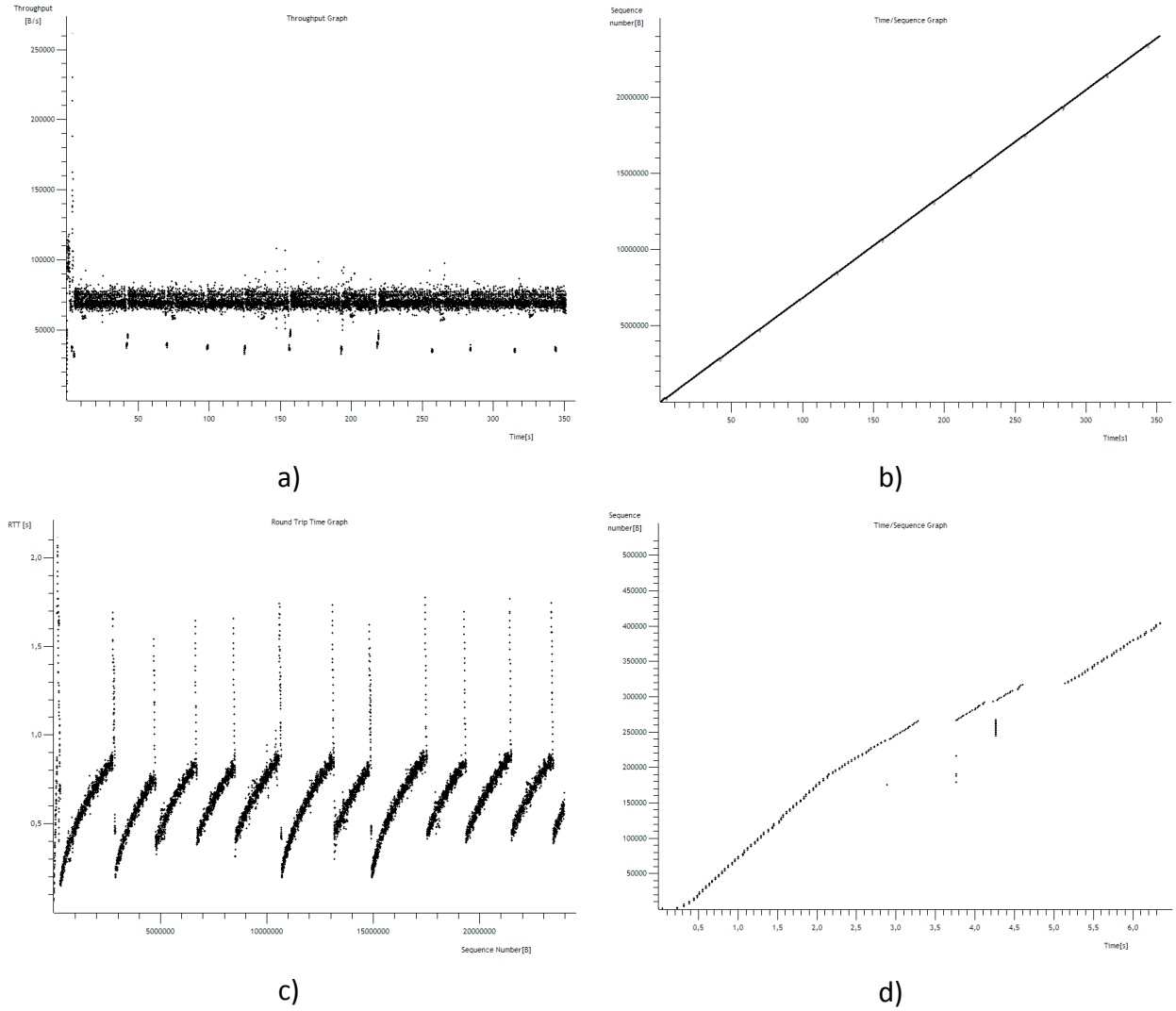


Figura 3: Resultados para 100 envíos de 200 KB. a) Throughput. b) Número de Secuencia vs tiempo. c) RTT vs número de secuencia. d) Acercamiento al comienzo de b).

Existen puntos bajo la recta "principal". Estas discontinuidades se deben a que existen pérdidas en el envío de paquetes. Los puntos solitarios bajo la recta principal se deben a que cierto paquete enviado, no recibió acuse de recibo o pasó el tiempo de espera para éste. Esto implica que se envía nuevamente. El hecho de que pase un largo tiempo entre trazas (como los observados en la figura), implica que la retransmisión del paquete se debió a que se venció el tiempo de espera del paquete.

De la forma que tiene RTT (figura 3-c)) y la forma del Throughput, denominado por la letra B, (figura 3-a)), se buscó una ecuación que determinara la relación entre ambos. Investigando, se encontró que existen varios estudios, alejados del nivel de este trabajo, que buscan identificar esta relación. Esta relación es compleja, e incluye varios parámetros que dependen del canal de comunicación.

Una relación, encontrada en "*Modeling TCP Throughput: A simple model and its Empirical Validation"* (Padhye, J., Firoiu, V., Towsley, D., Kurose, J) propone:

$$B \approx \min\left(\frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 \min\left(1{,}3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)}\right)$$

Donde b representa el número de paquetes reconocidos por cierto ACK, p la probabilidad de que ocurra una pérdida (de que no llegue un reconocimiento), y To corresponde al tiempo de timeout.

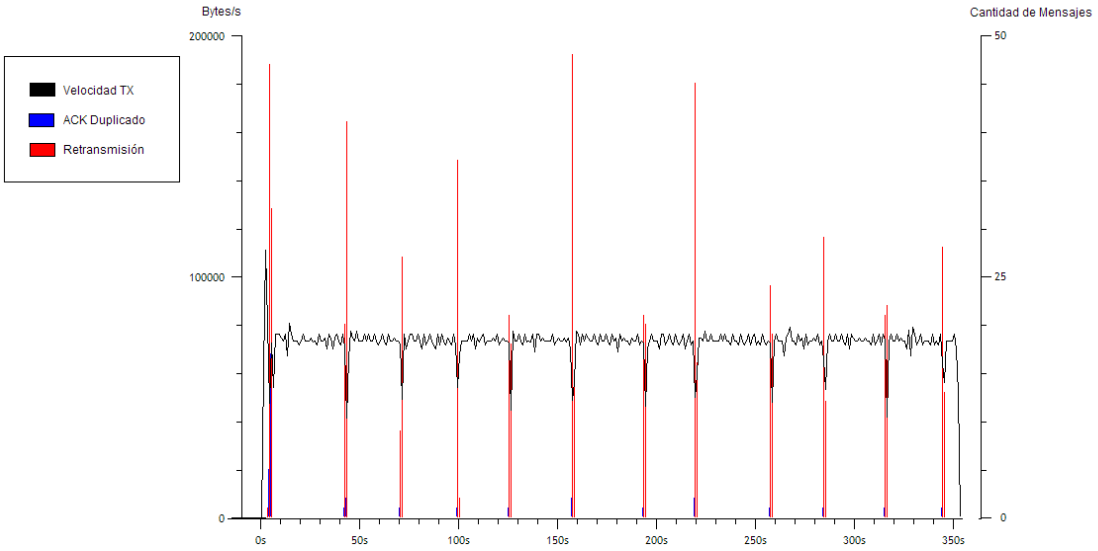


Figura 4: Resultados para 100 envíos de 200 KB. Velocidad de transmisión, ACK duplicados y retransmisiones.

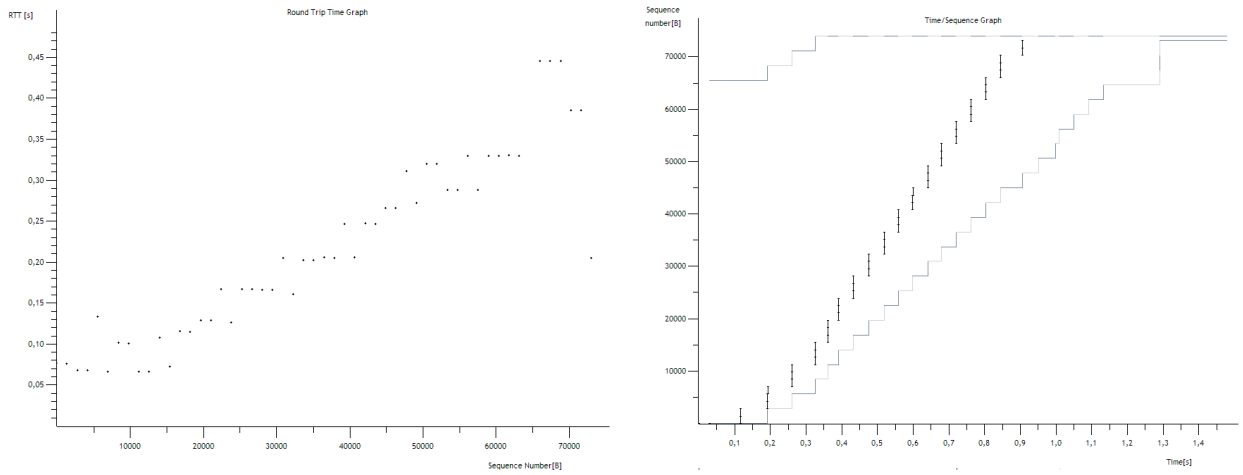
Se puede observar, que la cantidad de datos enviados no fue suficiente para lograr una situación de congestión. Sin embargo, se logró analizar el comportamiento de retransmisiones debido a ACK Duplicados.

3. **Envío continuo de paquetes**

Se envían paquetes de forma continua, con un tiempo muy pequeño. De esta forma, se sigue buscando lograr la congestión de la red o del puerto del host receptor.

Con esta situación se logró congestionar la red. Esto se puede observar en la figura 5-b). Se observa que se reciben ACK, por lo que se pueden seguir enviando paquetes. Pero como aumenta rápidamente, el valor de ventana no aumenta. Es por esto que la conexión se "cae", ya que el valor de ventana es alcanzado. Si bien no se muestra en la

figura, más adelante el envío continúa ya que llegan los ACK correspondientes y se sigue la transmisión.



a) b)

Figura 5: Envío continuo de paquetes. a) RTT. b) Número de secuencia vs Tiempo

Conclusiones

Mediante esta experiencia se buscaba observar en la práctica algunos de los temas vistos en clases. Esto se logró gracias al uso del software Wireshark. Así, los métodos de control de congestión y flujo, en el protocolo TCP. Se estudiaron más casos de los mostrados en este trabajo, pero se mostraron aquellos en que se observa más claramente estos métodos. Además, se pudo observar los parámetros correspondientes al encabezado de un segmento TCP, así como también los métodos mediante los que se inicia y termina una conexión de este tipo.

Al utilizar Wireshark, se comprobó la gran utilidad que tiene este software para el análisis de protocolos, ya que tiene una amplia gama de opciones, que permiten realizar análisis profundos y muy específicos.

Referencias

- Sanders, Chris. "*Practical Paquet Análisis*". No Starch Press. 2007.
- Padhye, J., Firoiu, V., Towsley, D., Kurose, J.. "*Modeling TCP Throughput: A simple model and its Empirical Validation*".
- Ahn, J., Danzig, P., Liu, Z., Yau, L.. "*Emulation of TCP Vegas: Emulation and Experiment".*
- Rexford, J. "*Congestion Control",* presentación de asignatura Computer Networks, Princeton University, Primavera de 2006.
- González, A. Documentos de asignatura Redes de Computadores, Universidad Técnica Federico Santa María, 2009.
- *"Wireshark: Display Filter Reference"*. Centro de soporte de Wireshark. Disponible en http://www.wireshark.org/docs/dfref/.

Anexos

Código Utilizado, en lenguaje C, basado en tutorial encontrado en http://www.aztekmindz.org/2008/06/24/tutorial-programacion-de-sockets-en-c-parte-i/

CLIENTE:

```
#include <cstdlib>
#include <stdio.h>
#include <winsock2.h> // Referencia a la libreria
#include <time.h>

using namespace std;

WSADATA wsadata; //Declaramos WSADATA

struct hostent *host;
//Delcaramos estrucutra hostent donde almacenaremos la IP que nos devuelva gethostbyname

SOCKADDR_IN conexrem;
/*Declaramos una estructura SOCKADDR_IN para no tener que definir una IP y un puerto
en cada paquete que enviemos. De esa forma todo viajara encapsulado utilizando la estructura
SOCKADDR_IN.
*/

SOCKET locsock; // Declaramos el descriptor de fichero que nos de el socket

int WSAInicio() { //declaramos procedimiento
        int wasa = WSAStartup(MAKEWORD(2,0),&wsadata); //Indicamos version 2.0 del socket
        if (wasa != 0) { // Si existen errores...
   printf("%s","Error iniciando WSAStartup \n"); //Mostramos un mensaje
                WSACleanup(); //Limpiamos WSADATA
                return 1; // Retornamos 1 dado que la funcion fallo
        }
        return 0; // Si se inicio todo bien retornamos 0
}

int definirsocket() {
        locsock = socket(AF_INET/* IP V4 */, SOCK_STREAM, 0); // Indicamos que usaremos un socket
Stream(TCP)
        if (locsock == INVALID_SOCKET) { // Si existen errores...
        printf("%s","Error definiendo socket \n"); //Mostramos un mensaje
                WSACleanup(); //Limpiamos WSADATA
                return 1; // Retornamos 1 dado que la funcion fallo
        }
        return 0; // Si se inicio todo bien retornamos 0
}

int estructsocket() { // Definimos procedimiento
  host=gethostbyname("190.164.77.54"); //Definimos Host con la IP que devuelva gethostbyname

 /* Resolvemos la direccion IP del Dominio localhost, con esto conseguimos traducir nombres de dominio
        sin utilizar direccion IP directamente.
        Esta sera la IP a donde nuestro Cliente conectara.
        */
        conexrem.sin_port = htons(9999); //Ordenacion de Red.
   /*  Definimos puerto (9999) del socket utilizando un "short de máquina a short de la red" (htons)
```

```
   Esto lo hacemos para ordenar la forma en la que enviaremos y recibiremos los datos por el puerto del
socket
   */
        conexrem.sin_addr = *((struct in_addr *)host->h_addr);
        /* Definimos la IP a donde conectaremos, como teniamos la direccion IP almacenada en "host"
   y la funcion solo admite "in_addr" para almacenar la IP en "h_addr" tenemos que hacer un Casting de
modo que
   "host" que es ahora un "in_addr" (debido a el casting (struct in_addr *) ) sea un puntero a "h_addr".
        */
        conexrem.sin_family = AF_INET; // Ordenacion de Maquina
        /*
        Definimos la version 4 de IP
        */
        memset(conexrem.sin_zero,0,8); // Ponemos en 0 la cadena sin_zero en sus 8 espacios

        if (connect(locsock, (sockaddr*)&conexrem, sizeof(conexrem)) == SOCKET_ERROR) { // Si existe un
error...
   /* Ahora procedemos a realizar la conexion con la IP remota que definimos en la estructura
"conexrem.sin_addr"
   y el puerto definido en "conexrem.sin_port". Dado que connect solo acepta "sockaddr" enves de nuestra
   estructura "SOCKADDR_IN" tenemos que realizar un casting para convertir "conexrem" en un
"sockaddr".
   Despues medimos la longitud de la estructura con "sizeof" y por ultimo verificamos si se produjo un error
   tratando de conectar ya que por ejemplo la direccion IP remota podria no existir o estar detras de un
FireWall
  */
        printf("%s","Error conectando al servidor remoto \n"); // En caso de error mostramos msj
                WSACleanup(); //Limpiamos WSADATA
                return 1; // Retornamos 1 dado que la funcion fallo
        } else // De lo contrario...
     printf("%s","Coneccion realizada con exito \n"); // Mostramos msj
                return 0; // Si se inicio todo bien retornamos 0
}

void enviarmsj(){
int i = 50;

   while(i>0){
        char msj[] = "MENSAJE";
        printf("\n %s %d \n", "tamaño", sizeof(msj));
        send(locsock,msj,sizeof(msj),0); // Enviamos mensaje
        Sleep(50);
        i--;

   }
   //char a[]="FIN";
   //send(locsock,a,sizeof(a),0); // Enviamos mensaje

   //char msj2[] = "END";
   //send(locsock,msj2,sizeof(msj2),0); // Enviamos mensaje
   /* Utilizamos la funcion "send" para enviar datos atraves del descriptor de fichero "locsock" que nos dio
el socket
   Despues medimos la cantidad de caracteres a enviar en "msj" utilizando "sizeof"
   */
   }

void sockets(){ // Procedimiento que iniciara el socket secuencialmente.
     if((WSAInicio()) == 0) { // Si se inicio WSAInicio sin errores...
```

```
        if((definirsocket()) == 0) { // Si se inicio definirsocket sin errores...
          if((estructsocket()) == 0) { // Si se inicio estructsocket sin errores...
            enviarmsj(); // Iniciamos el procedimiento "enviarmsj"
          } else { // Si no conecto..
              Sleep(500); // Esperamos 500 Milisegundos y...
                          sockets(); // Repetimos proceso
              }
        }
      }
    }

int main(int argc, char *argv[])
{
      sockets(); // Iniciamos el Socket
}
```

SERVIDOR:

```
#include <cstdlib>
#include <stdio.h>
#include <winsock2.h> // Referencia a la libreria

using namespace std;

WSADATA wsadata; //Declaramos WSADATA

struct hostent *host;
//Delcaramos estrucutra hostent donde almacenaremos la IP que nos devuelva gethostbyname

SOCKADDR_IN conexloc;
/*Declaramos una estructura SOCKADDR_IN para no tener que definir una IP y un puerto
en cada paquete que enviemos. De esa forma todo viajara encapsulado utilizando la estructura
SOCKADDR_IN.
*/

SOCKET locsock; // Declaramos el descriptor de fichero que nos de el socket

char Buffer[1023]; // Declaramos el tamaño del Buffer

int WSAInicio() { //declaramos procedimiento
          int wasa = WSAStartup(MAKEWORD(2,0),&wsadata); //Indicamos version 2.0 del socket
          if (wasa != 0) { // Si existen errores...
    printf("%s","Error iniciando WSAStartup \n"); //Mostramos un mensaje
                    WSACleanup(); //Limpiamos WSADATA
                    return 1; // Retornamos 1 dado que la funcion fallo
          }
          return 0; // Si se inicio todo bien retornamos 0
}

int definirsocket() {
          locsock = socket(AF_INET/* IP V4 */, SOCK_STREAM, 0); // Indicamos que usaremos un socket
Stream(TCP)
          if (locsock == INVALID_SOCKET) { // Si existen errores...
          printf("%s","Error definiendo socket \n"); //Mostramos un mensaje
                    WSACleanup(); //Limpiamos WSADATA
                    return 1; // Retornamos 1 dado que la funcion fallo
          }
          return 0; // Si se inicio todo bien retornamos 0
}
```

```
int estructsocket() { // Definimos procedimiento
    conexloc.sin_family = AF_INET; // Ordenacion de Maquina
     /*
       Definimos la version 4 de IP
     */
    conexloc.sin_addr.s_addr = INADDR_ANY;
    /*
       Definimos IP local
    */
    conexloc.sin_port = htons(9999);
    /* Definimos puerto (9999) por el que escuchara el socket utilizando un “short de máquina a short de la red” (htons)
       Esto lo hacemos para ordenar la forma en la que enviaremos y recibiremos los datos por el puerto del socket, mas información buscar en google “Big-Endian”.
    */

            if (bind(locsock, (sockaddr*)&conexloc, sizeof(conexloc)) == SOCKET_ERROR) { // Si existen errores...
            /* Una ves creado el socket asociamos el descriptor de ficher "locsock" a un puerto, para eso utilizamos la funcion "bind"
               Despues asignamos la direccion IP pasandola como puntero a "sockaddr".
            */
                    printf("%s","Error definiendo socket \n"); //Mostramos un mensaje
                    WSACleanup(); //Limpiamos WSADATA
                    return 1; // Retornamos 1 dado que la funcion fallo
            } else {
          if (listen(locsock, 1) == SOCKET_ERROR) { // Si existen errores...
          /* La funcion listen sirve para poner el socket en escucha por un puerto determinado, en nuestro caso el puerto 9999
             que fue definido al socket en la funcion "bind", utilizando el descriptor de fichero "locsock" que "socket()" nos dio
             El ultimo numero es la cantidad de conexiones que podemos tener como maximo en la cola de espera.
          */
                    printf("%s","Error Al ponerse en escucha \n"); //Mostramos un mensaje
              WSACleanup(); //Limpiamos WSADATA
                        return 1; // Retornamos 1 dado que la funcion fallo
                    } else {
                printf("%s","Esperando conexiones por puerto 9999 \n"); //Mostramos un mensaje
                return 0; // Si se inicio todo bien retornamos 0
               }

    }
}

void conexion(){

    int conm; //Declaramos variable para definir longitud de la estructura "sockaddr"
    conm=sizeof(struct sockaddr); // definimos longitud de "sockaddr"
    locsock=accept(locsock,(sockaddr*)&conexloc,&conm); // Conexion establecida
    /* Aceptamos la conexión con la funcion "accept" utilizando el descriptor de fichero "locsock" para transmitir los datos entre ambas computadoras
       mediante nuestra estructura "SOCKADDR_IN" definida anteriormente en "conexloc"
    */
    printf("%s","Conexion establecida \n"); //Mostramos un mensaje

          while (conm!=0){ //mientras estemos conectados..
```

```
            conm=recv(locsock,Buffer,sizeof(Buffer),0); //recibimos los datos que envie
            /* La funcion "recv" se encarga de recibir los datos atraves del descriptor de fichero "locsock"
              Buffer es donde se va almacenar la informacion recibida y posteriormente con "sizeof" medimos la longitud total del buffer recibido
              Dado que "recv" devuelve un valor de cero en caso de existir un error se utiliza un bucle while para verificar
              que seguimos conectados con el equipo remoto
            */
            if (conm>0){  //si seguimos conectados al cliente
            printf("Datos recibidos:%s \n Tamaño: %d \n",Buffer, sizeof(Buffer)); //imprimimos los datos recibidos

            }
         }
    }

void sockets(){ // Procedimiento que iniciara el socket secuencialmente.
      if((WSAInicio()) == 0) { // Si se inicio WSAInicio sin errores...
        if((definirsocket()) == 0) { // Si se inicio definirsocket sin errores...
          if((estructsocket()) == 0) { // Si se inicio estructsocket sin errores...
            conexion(); // Iniciamos el procedimiento "conexion"
          }
        }
      }
    }

int main(int argc, char *argv[])
{
   sockets(); // Iniciamos el Socket
}
```