# INTRODUCCION

En la presente monografía se describirá arquitecturas para la realización de servidores Web de alto rendimiento básicamente orientado para el lenguaje de programación JAVA que es software que esta predominando en la actualidad en Internet.

El desafío actual es construir infraestructuras de comunicaciones y herramientas de aplicación capaces de soportar el transporte de información multimedia a través de redes de área amplia, en especial, a través de Internet. Los inconvenientes a solucionar están directamente ligados a la naturaleza de la información multimedial, y a las necesidades de entrega en tiempo real y a cierta frecuencia constante. Estas condiciones requieren de mayor ancho de banda disponible y de mecanismos que permitan manejar los tiempos de retardo y la congestión en las redes. En la actualidad, las redes de datos son compartidas por gran cantidad de usuarios, los cuales tienen ancho de banda limitado, sensible retardo y disponibilidad impredecible (la red opera haciendo su mejor esfuerzo, sin brindar calidad de servicio). Desde principios de los 70 se busco solucionar los inconvenientes anteriormente mencionados creando protocolos como TCP/IP.

TCP/IP es una colección de protocolos estándar de la industria diseñada para intercomunicar grandes redes (WANs = *Wide Area Networks)*. Las siglas TCP/IP provienen de *Transmission Control Protocol / Internet Protocol.* Vamos a intentar dar en esta parte, unos conceptos sobre TPC/IP. El TCP/IP fue originado con los experimentos de intercambio de paquetes dirigido por el *U.S. Department of Defense Advanced Research Projects Agency* (DARPA) durante la década de 1960 a 1970.

Hay varios hitos importantes en la historia del TCP/IP:

1970: Los ordenadores de la *Advanced Research Agency Network* (ARPANET) comienzan a utilizar el NCP (*Network Control Protocol)*.
1972: La primera especificación Telnet. *"Ad hoc Telnet Protocol*" se define como una RFC, la 318.
1973: RFC 454. Se introduce el FTP (*File Transport Protocol)*
1974: El TCP (*Transmission Control Program)* se especifica detalladamente.
1981: El estándar IP se publica en la RFC 791
1982: La '*Defense Communications Agency'* (DCA) y ARPA establecen a la '*Transmision Control Protcolol (TCP)* y al *Internet Protocol (IP)* como la colección de protocolos TCP/IP.
1983: ARPANET cambia de NCP a TCP/IP
1984: Se define el concepto de DNS (*Domanin Name System*)

Con la creación de TCP/IP se estructura un modelo de cuatro capas en el que nos basaremos para el estudio de la presente monografía. Aunque este modelo es general en todas las implementaciones del TCP/IP, a lo largo del presente documento, vamos a ceñirnos a su implementación básicamente en Microsoft Windows.

**Capa de Red**: La base de este modelo de capas, es la capa de *interface* de red. Esta capa es la responsable de enviar y recibir *frames,* los cuales son los paquetes de información que viajan en una red cono una 'unidad simple'. La capa de red, envía *frames* a la red, y recupera *frames* de la red.

**Capa de Internet**: Este protocolo encapsula paquetes en *datagramas* internet y además esta capa ejecuta todos los algoritmos de enrutamiento (*routing*) de paquetes. Los cuatro protocolos Internet son: *Internet Protocol* (IP), *Address Resolution Protocol* (ARP), *Internet Control Message Protocol* (ICMP) y *Internet Group Management Protocol* (IGMP).

IP es el responsable del envío y enrutamiento de paquetes entre maquinas y redes.

- ARP obtiene las direcciones de hardware de las maquinas situadas en la misma red física. Recordemos que la dirección física de cada tarjeta de red es única en el mundo. La dirección "física" ha sido implementada vía hardware por el fabricante de la tarjeta de red, y dicho fabricante, lo selecciona de un rango de direcciones único asignado a él y garantiza la unicidad de dicha tarjeta. Este caso es el más corriente y es el de las tarjetas de Red Ethernet.

- ICMP envía mensajes e informa de errores en el envío de paquetes.

- IGMP se utiliza para la comunicación entre *routers* (enrutadores de Internet).

**Capa de Transporte**: La capa de transporte, da el nivel de "sesión" en la comunicación. Los dos protocolos posibles de transportes son TCP (*Transmission Control Protocol)* y UDP *(User Datagram Protocol)*. Se puede utilizar uno u otro protocolo dependiendo del método preferido de envío de datos.

El TCP nos da un tipo de conectividad "orientada a conexión". Típicamente se utiliza para transferencias de largas cantidades de datos de una sola vez. Se utiliza también en aplicaciones que requieren un "reconocimiento" o validación (ACK : *acknowledgment*) de los datos recibidos.

El UDP proporciona conexión de comunicación y no garantiza la entrega de paquetes. Las aplicaciones que utilicen UDP normalmente envían pequeñas cantidades de datos de una sola vez. La aplicación que lo utilice, es la responsable en este caso de la integridad de los paquetes y debe establecer sus propios mecanismos para pedir repetición de mensaje, seguimiento, etc, no existiendo ni garantía de entrega ni garantía del orden de entrega en la maquina destino.

**Capa de Aplicación**: En la cima de este modelo, está la capa de aplicación. Esta es la capa que las aplicaciones utilizan para acceder a la red. Existen muchas utilidades y servicios en la capa de aplicación, por ejemplo: FTP, Telnet, SNMP y DNS.

Una vez comprendido la arquitectura en la que se basa para la creación de servidores web conviene explicar un poco del lenguaje JAVA.

Java es un *lenguaje orientado a objetos*. Esto significa que posee ciertas características que hoy día se consideran estándares en los lenguajes OO:

- Objetos
- Clases
- Métodos
- Subclases
- Herencia simple
- Enlace dinámico
- Encapsulamiento

Para programar orientado a objetos es necesario primero *diseñar un conjunto de clases*. La claridad, eficiencia del programa resultante dependerá principalmente de la calidad del diseño de clases. Un buen diseño de clases significará una gran economía en tiempo de desarrollo y manutención.

Lamentablemente se necesita mucha habilidad y experiencia para lograr diseños de clases de calidad. Un mal diseño de clases puede llevar a programas OO de peor calidad y de más alto costo que el programa equivalente no OO.

Es válido entonces preguntarse: ¿Por qué programar en un lenguaje OO, si se requiere una experiencia que probablemente uno nunca tendrá el tiempo de práctica para llegar a dominarla?

La respuesta a esta pregunta es la siguiente: Java es un lenguaje multiparadigma no necesita hacer un diseño de clases para programar una aplicación de mil líneas.

¿Entonces por qué no usar otro lenguaje más simple como Visual Basic, si no necesito orientación a objetos?

Porque la ventaja potencial más importante de un lenguaje OO está en las *bibliotecas de clases* que se pueden construir para él. Una biblioteca de clases cumple el mismo objetivo de una biblioteca de procedimientos en lenguaje como C. Sin embargo:

Una biblioteca de clases es mucho más fácil de usar que una biblioteca de procedimientos, incluso para programadores sin experiencia en orientación a objetos. Esto se debe a que las clases ofrecen mecanismos de abstracción más eficaces que los procedimientos.

Por lo tanto podemos distinguir entre varios tipos de programadores en Java:

- El diseñador de clases: Es el encargado de definir qué clases ofrece una biblioteca y cuál es la funcionalidad que se espera de estas clases. Esta persona tiene que ser muy hábil y de mucha experiencia. Un diseño equivocado puede conducir a clases que son incomprensibles para los clientes de la biblioteca.

- El programador de clases de biblioteca: Sólo programa las clases especificadas por el diseñador de clases. Esta persona debe entender orientación a objetos, pero no requiere mayor experiencia en diseño de clases.
- El cliente de bibliotecas: Es el programador de aplicaciones. Él sólo usa las clases que otros han diseñado y programado.

Tanto programadores de clases como clientes de bibliotecas pueden llegar a convertirse en buenos diseñadores de clases en la medida que adquieran experiencia, comparando los diseños de las bibliotecas que utilicen.

Java ha sido diseñado de modo de eliminar las complejidades de otros lenguajes como C y C++.

Si bien Java posee una sintaxis similar a C, con el objeto de facilitar la migración de C hacia a Java, Java es semánticamente muy distinto a C:

- Java no posee aritmética de punteros: La aritmética de punteros es el origen de muchos errores de programación que no se manifiestan durante la depuración y que una vez que el usuario los detecta son difíciles de resolver.
- No se necesita hacer **delete**: Determinar el momento en que se debe liberar el espacio ocupado por un objeto es un problema difícil de resolver correctamente. Esto también es el origen a errores difíciles de detectar y solucionar.
- No hay herencia múltiple: En C++ esta característica da origen a muchas situaciones de borde en donde es difícil predecir cuál será el resultado. Por esta razón en Java se opta por herencia simple que es mucho más simple de aprender y dominar.

Toda implementación de Java debe tener las siguientes bibliotecas de clases:

- Manejo de archivos
- Comunicación de datos
- Acceso a la red internet
- Acceso a bases de datos
- Interfaces gráficas

Los programas en Java pueden ejecutarse en cualquiera de las siguientes plataformas, sin necesidad de hacer cambios:

- Windows/95 y /NT
- Power/Mac
- Unix (Solaris, Silicon Graphics, ...)

La compatibilidad es total:

- A nivel de fuentes: El lenguaje es exactamente el mismo en todas las plataformas.

- A nivel de bibliotecas: En todas las plataformas están presentes las mismas bibliotecas estándares.
- A nivel del código compilado: el código intermedio que genera el compilador es el mismo para todas las plataformas. Lo que cambia es el intérprete del código intermedio.

Por lo tanto Java no es un lenguaje para hacer sistemas operativos o administradores de memoria, pero sí es un excelente lenguaje para programar aplicaciones.

Java combina flexibilidad, robustez y legibilidad gracias a una mezcla de chequeo de tipos durante la compilación y durante la ejecución. En Java se pueden tener punteros a objetos de un tipo específico y también se pueden tener punteros a objetos de cualquier tipo. Estos punteros se pueden convertir a punteros de un tipo específico aplicando un *cast*, en cuyo caso se chequea en tiempo de ejecución de que el objeto sea de un tipo compatible.

En Java los programadores no necesitan preocuparse de liberar un trozo de memoria cuando ya no lo necesitan. Es el recolector de basuras el que determina cuando se puede liberar la memoria ocupada por un objeto.

Un recolector de basuras es un gran aporte a la productividad. Se ha estudiado en casos concretos que los programadores han dedicado un 40% del tiempo de desarrollo a determinar en qué momento se puede liberar un trozo de memoria.

Los contenidos se enfocan desde un punto de vista práctico, mediante la exposición de los conceptos, el estudio de los mecanismos reales de implantación (ilustrados con el uso de un lenguaje de programación orientado específicamente a ello como Java), la descripción de las bibliotecas y herramientas necesarias, y a la realización de ejercicios.

# 1. Arquitectura en niveles y cliente servidor.

## *1.1 Introducción a las redes de computadores.*

Una red de computadores es un conjunto de computadores conectados entre sí que emplean un mismo leguaje o **protocolo.** Estos computadores se interconectan mediante el teléfono, las microondas o los satélites. Para que se conecten unas redes con otras se establecen una serie de ordenadores denominados **routers**, que se encargar de gestionar las comunicaciones. Las redes permiten que múltiples computadores se puedan comunicarse entre ellos y compartir datos y recursos, esto es programas.

La aparición de las redes de computadores surgió en los años setenta, en los Estados Unidos con una red llamada Arpanet resistente a fallos entre nodos e independientes entre si, en su origen fue un proyecto militar conformada por cuatro universidades, el cual se lo puede denominar que fue la semilla del Internet.

Luego aparecieron en el mercado empresas que proveían equipos tales como IBM (Mainframes SNA), DIGITAL (DECNET), NOVELL Netware, para crear un estándar se creo el modelo de referencia ISO-OSI.

La *Organization for International Standardization* (ISO) la que dedicó varios años a establecer unas normas universales que homogeneizasen las comunicaciones de las redes de computadores. Estos estándares permitieron que se instalase el software IP en numerosos ordenadores para que se entendieran entre ellos. Pronto el gobierno y las universidades norteamericanas se dedicaron a interconectarse utilizando dicho protocolo ya estandarizado, buscando unos fines de investigación y académicos bien distintos de los militares.

En efecto, en el mundo académico, la Universidad de Berkley comenzó a utilizar el protocolo de comunicación IP en sus redes de ordenadores UNIX hacia 1982, lo que les permitió acceder a ARPAnet. Se trataba de que la comunidad académica pudiera intercambiar sus experimentaciones y conocimientos.

Por su parte, a fines de los 80, una agencia del gobierno norteamericano llamada la *National Science Foundation* (NSF) desarrolló cinco centros de proceso de datos en distintos lugares. Para comunicarlos entre sí se pretendió utilizar la infraestructura de ARPAnet, pero la burocracia abortó el intento.

La NSF tuvo que desarrollar su propia red de ordenadores, y para ello empleó la tecnología IP de ARPAnet. Y lo que es más importante, permitió que todas las universidades que lo deseasen se

interconectases a través de su red. La NSF pretendía promover el acceso universal a la educación facilitando dinero para dichos propósitos.

Durante los comienzos de los años 90, e incluso un poco antes, Internet desbordó las fronteras norteamericanas. Ya no solo de comunicaban las bases militares norteamericanas fuera de EE.UU., sino que esos países aliados comenzaban a conectarse también.

Este fue el origen de Internet, inicialmente restringida a los organismos científicos, gubernamentales y educativos; pero el gran salto hacia delante lo dieron las empresas privadas cuando decidieron entrar en Internet en los años 90. Eso ocurrió cuando se popularizó Internet gracias a la difusión del *World Wide Web* a partir de 1994. Desde entonces, el crecimiento de la Red se experimenta principalmente en el campo de las empresas, que han visto una fuente de negocios sin igual en Internet, así como en los propios hogares. Hoy se puede afirmar que el crecimiento de Internet es exponencial.

## *1.2 Servicios de Internet.*

A medida que fueron creciendo las redes de computadores e interconectándose entres si, como ya se menciono se creo una red mundial denominada Internet, que es un conjunto de protocolos (IP, TCP, UDP, DNS) que incorpora una serie de servicios distribuidos (ftp, telnet, dns, e-mail, www). Internet es, o pretende ser, la **red de redes** que abarca todas las redes de computadores que hay en el mundo, o al menos esa es su "vocación universalista". Vocación que se plasma hablando un mismo "esperanto"; en efecto existe ese lenguaje común que emplea Internet es el protocolo **TCP/IP** (Transmission Control Protocol/ Internet Protocol). Eso sí, dentro del conjunto de Internet cada red que la compone funciona independientemente y se "sumergen" en la Red en la medida que deseen.

Para la interconexión de los computadores se baso en la conmutación de paquetes (almacenamiento y reenvío). Inicialmente emulan servicios del Sistema Operativo tales como:

- Correo Electrónico
- Terminal Virtual
- Transferencia de archivos.

En definitiva, Internet es una red de redes a través de la cual todo el planeta puede comunicarse, así como intercambiar informaciones y programas electrónicos.

Internet es un sistema en el que participan dos partes: por un lado está el ordenador desde el que accedemos a la Red, **ordenador local**, que funciona gracias a un **programa cliente**, y en el otro extremo se encuentra el ordenador al que accedemos, **ordenador remoto**, que nos facilita lo que necesitamos gracias a un **programa servidor**.

El programa cliente gestiona la comunicación con el servidor y ofrece las herramientas necesarias para poder trabajar con dicho servidor.

El programa servidor se encarga de transmitir la información en la forma más adecuada para el usuario o usuarios, ya que un servidor admite múltiples accesos simultáneos.

Los programas cliente y servidor pueden ser muy variados y funcionar sobre sistemas operativos diversos (UNIX, Windows NT, MS-DOS, OS/2, etc.). Eso sí tanto el cliente como el servidor deben utilizar el protocolo de comunicaciones TCP/IP, como hemos visto anteriormente.

El protocolo TCP controla la transmisión de ficheros electrónicos; en efecto, divide la información en porciones apropiadas numeradas para que puedan volver a unirse en su totalidad, o si hubiera algún error en la transmisión le permite identificar al instante. Esa labor de recomposición también la realiza el propio protocolo TCP.

Por su lado, el **protocolo IP** asigna a cada paquete de información electrónica que fluye por la Red la dirección apropiada en Internet, ***IP address***. Dirección que identifica e individualiza a todos y cada uno de los ordenadores conectados a Internet, que reciben el nombre de **anfitrión** u ***host***. Esta dirección o número IP se compone por cuatro grupos sucesivos de cifras, con un valor comprendido entre 0 y 255, que están separados por puntos. Por ejemplo una dirección IP puede tomar aparecer así: 188.132.56.23., aunque también existe una forma alternativa de tipo alfabético. Las direcciones IP al ser numéricas resultan difíciles de recordar y ya desde los comienzos de Internet se buscaba un sistema de nombres para identificar dichas direcciones IP. Así se asignan unos nombres que se denominan **dominios**. Al principio, el *Network Information Center* (NIC), ahora InterNIC, registraba en un fichero electrónico todos los dominios. Pero, a medida que fue creciendo Internet, el NIC no daba a basto para afrontar por sí solo todas las solicitudes de dominio, además, el enorme fichero se distribuía cada vez más lentamente en la Red, pues debí llegar a todas las máquinas que aparecían en él. Como consecuencia de esa situación tuvo que desarrollarse el denominado Domain Name System o **DNS**.

El DNS es un método jerárquico que sirve para establecer nombres de dominio por medio de la asignación de diversos grupos de subsistemas

de los nombres. Cada nivel del sistema también se denomina dominio y se separa por puntos, con una estructura que sería algo así como:

*nombre_host.subsubdominio.subdominio.dominio_principal*

Puede haber un número variable de grupos de dominio, normalmente menos de cinco, que va del más específico al más general. Analicemos por ejemplo el nombre pardoo.cs.umass.edu; el dominio "pardoo" identifica un *host*, esto es un ordenador con una dirección IP; el nombre del ordenador ha sido asignado arbitrariamente por el departamento que lo mantiene, el "cs"; dicho departamento se encuentra en la Universidad de Massachusetts, "umass"; la cual pertenece al dominio "edu" que identifica a las universidades y centros educativos norteamericanos.

Precisamente el dominio **edu** es uno de los seis **dominios organizativos** que existen en Estados Unidos:

- **com** empresas y organizaciones comerciales.
- **edu** entidades educativas: universidades, escuelas secundarias, etc.
- **gov** organismos del gobierno norteamericano, no militares.
- **mil** organismos del los tres ejércitos norteamericanos.
- **org** organizaciones como fundaciones, asociaciones, partidos polísticos, etc.
- **net** recursos de la Red como, por ejemplo, suministradores de acceso a Internet.

Junto a estos dominios organizativos exclusivos de EE.UU. Existen unos **dominios geográficos** que identifican a todos y cada uno de los países del mundo. Así todos los dominios localizados en Ecuador poseen el dominio **ec**, los franceses **fr**, los ingleses **uk**, etc. Puede consultarse una lista completa de los códigos de todos los países en la Universidad de California.

Para acceder a un computador conectado a Internet se puede utilizar indistintamente la dirección IP o el nombre de dominio, aunque no siempre, ya que algunos *hosts* no tienen DNS. Eso sí, siempre se traducen los nombres de dominio a direcciones IP que son las que pueden entender las computadoras, misión que realiza un computador denominado **servidos de nombres de dominio**. En el caso de aquellos que sí la tienen el computador puede facilitar una serie de espacios a diversos usuarios de la organización que los mantiene, este espacio se llama **cuenta**. Dicha cuenta ha de contar con un *user name, user id, login,* etc.

Estas cuentas identificadas por todos esos parámetros son necesarias para distribuir ordenadamente los ficheros en directorios separados para

cada individuo, y para asignar direcciones de correo electrónico. En efecto la forma de una dirección de correo adopta el formato:

*nombre_usuario@nombre_host.subdominio.dominio_principal*

Como se ve se utiliza un nombre que designe al individuo particular, seguido por @ **(arroba)** y por el dominio del host. Por ejemplo:

ventas@islogica.com, donde ventas es el usuario particular con una cuenta en el ordenador con dominio islogica, el cual se encuentra en el dominio principal com.

### *1.3 Resumen de protocolos.*

Los protocolos que vamos analizar brevemente a continuación los dividiremos en dos tipos:

**De Internet.**
**De Control.**

Los primeros para su mejor comprensión se subdividirá en un *nivel de aplicación* en el cual se utiliza un protocolo denominado http () el cual es utilizado por los navegadores webs, debajo de este nivel encontramos *nivel de transporte* con los protocolos:

TCP que es fiable, utiliza conexiones y es bidireccional, es lento y su velocidad es variable, es diseñado para baja tasa binaria y malas condiciones del canal y realiza la retransmisión.

UDP no orientado a conexión, utiliza paquetes datagramas y no es fiable ni seguro y la responsabilidad esta en las capas superiores, soporta comunicaciones en grupo.

En el capítulo tres se revisará más a detalle algunos protocolos explicados anteriormente.

El siguiente nivel es el *nivel de red* el cual utiliza el protocolo IP, y por último esta el *nivel de enlace* en el cual se encuentran todos los drivers.

En los segundos los podemos dividir en:

- Internet Control Message Protocol.- utilizado para realizar las comprobaciones y detectar los errores.

- Address Resolution Protocol.- resuelve las direcciones entre la dirección IP del equipo con la dirección MAC de la tarjeta de Red.

- Reserve Address Resolution Protocol.- cumple funciones inversas a ARP.

- Open Shortest Path First.- realiza en encaminamientos entre los routers de la misma zona.

- Border Gateway Protocolo. Realiza en encaminamiento entre routers de zonas diferentes.

El orden en que el protocolo ejecuta las rutinas se encuentra en el nivel de aplicación.

### *1.4 Evolución de las aplicaciones de Internet.*

Cada día que pasa existe la evolución de las aplicaciones en Internet, por lo cual se esta revisando aspectos tales como legales y administrativos para que exista una regulación y legislación en el futuro del Internet y no sea para usos fraudulentos, y haya confidencialidad con algunas aplicaciones. Otro aspecto muy importante que se toma en cuenta es en futuro de los negocios con aplicaciones de comercio electrónico, dar un alto nivel de confianza a los usuarios para así tener un despliegue de la red con acceso universal con tarifas reducidas-planas.

Y la evolución más notable será las denominadas “Killer application” con video y VoIP.

Para que se este dando esta evolución acelerada de las aplicaciones de Internet, también se esta dando en lo que se refiere a la Tecnología con redes más rápidas, integración con otras redes (celulares, telefónicas, domesticas, wless), software más rápido, los terminales (equipos celulares, PDA, sistemas empotrados).

En cuanto a lo que es arquitecturas de software se este utilizando el modelo cliente-servidor (www), sistemas distribuidos y modelos p2p, software más fácil de realizar, principalmente JAVA.

## 2 Bibliotecas y principios de Diseño.

### *2.1 Lenguaje JAVA.*

Las características principales del lenguaje JAVA las podemos mencionar como:

Un lenguaje de alto nivel, no gestiona memoria, no hay punteos ni destructores, no primitivas del sistema operativo, realiza llamadas a métodos nativos, y existe concurrencia en el lenguaje.

Es un lenguaje orientado a objetos con un único ámbito permitido que es la clase, no hay funciones ni variables globales, existe genericidad en el polimorfismo y utiliza empaquetamiento.

## *2.2 Encapsulación y ocultación.*

Para la comprensión de Encapsulación definiremos un elemento fundamental, la **clase es la representación de una entidad reconocible, unidad de Encapsulación de datos y operaciones o métodos,** esta puede ser por ejemplo persona, conjunto, conexión con datos o atributos como estado, dirección IP, con operaciones o rutinas como conectar, enviar, las operaciones acceden a los atributos. Cada objeto tiene su copia de datos y funciones.

Las clases son lo más simple de Java. Todo en Java forma parte de una clase, es una clase o describe como funciona una clase. El conocimiento de las clases es fundamental para poder entender los programas Java.

Todas las acciones de los programas Java se colocan dentro del bloque de una clase o un *objeto*. Todos los métodos se definen dentro del bloque de la clase, Java no soporta funciones o variables globales. Esto puede despistar a los programadores de C++, que pueden definir métodos fuera del bloque de la clase, pero esta posibilidad es más un intento de no separarse mucho y ser compatible con C, que un buen diseño orientado a objetos. Así pues, el esqueleto de cualquier aplicación Java se basa en la definición de una clase.

Todos los datos básicos, como los *enteros*, se deben declarar en las clases antes de hacer uso de ellos. En C la unidad fundamental son los ficheros con código fuente, en Java son las clases. De hecho son pocas las sentencias que se pueden colocar fuera del bloque de una clase. La palabra clave import (equivalente al #include) puede colocarse al principio de un fichero, fuera del bloque de la clase. Sin embargo, el compilador reemplazará esa sentencia con el contenido del fichero que se indique, que consistirá, como es de suponer, en más clases.

### *Tipos de Clases*

Los tipos de clases que podemos definir son:

**abstract**

Una clase *abstract* tiene al menos un método abstracto. Una clase abstracta no se instancia, sino que se utiliza como clase base para la herencia.

**final**

Una clase *final* se declara como la clase que termina una cadena de herencia. No se puede heredar de una clase final. Por ejemplo, la clase **Math** es una clase final.

**public**

Las clases *public* son accesibles desde otras clases, bien sea directamente o por herencia. Son accesibles dentro del mismo paquete en el que se han declarado. Para acceder desde otros paquetes, primero tienen que ser importadas.

**synchronizable**

Este modificador especifica que todos los métodos definidos en la clase son sincronizados, es decir, que no se puede acceder al mismo tiempo a ellos desde distintos threads; el sistema se encarga de colocar los flags necesarios para evitarlo. Este mecanismo hace que desde threads diferentes se puedan modificar las mismas variables sin que haya problemas de que se sobrescriban.

A continuación veremos algunos tipos de atributos y operaciones utilizados en JAVA

**Atributos Simples**

| Tipo | Contiene | Inicial | Tamaño | Mínimo | Máximo |
|---|---|---|---|---|---|
| Bolean | Trae/false | False | 1 bit | | |
| Char | Unicote | \U0000 | 16 bits | \U0000 | \UFFFF |
| Byte | Entero | 0 | 8 bits | -128 | 127 |
| Short | Entero | 0 | 16 bits | -32768 | 32767 |
| Int | Entero | 0 | 32 bits | -2147483648 | 2147483647 |
| Long | Entero | 0 | 64 bits | -9E18 | 9E18 |
| float | IEEE 754 real | 0.0 | 32 bits | -3.4E38 | 3.4E38 |
| double | IEEE 754 real | 0.0 | 64 bits | -1.7E308 | 1.7E308 |

**Operaciones simples**

- Aritméticas
    - Son parámetros aritméticos que devuelven un valor entero o real ( + - * / %).
- Relaciónales.

- Parámetros aritméticos que devuelven un valor lógico (> < >= <= ¡= = =).

- Lógicos
  - Parámetros lógicos que devuelven lógico (¡ && ||)

- De bits
  - Parámetros enteros y devuelven enteros

  **~** negar (0 y 1)
  **&** not (bit)
  **|** and
  **^** or
  << >> desplazamiento
  **>>>** desplazamiento a 0.

- Asignaciones
  - Parámetros: variable y expresión

    a = b; asignación.
    a ++; incremento y asignación.
    a --; decremento y asignación.

- Encapsulados en los objetos
  - System.out.print()
  - System.out.println()

**Atributos referenciados.**

- Otros tipos de atributos que representan agrupaciones de datos:
  - Vectores o arrays
  - Clases

- Las variables de estos tipos se manejan mediante un nombre
  - Referencia

    Es un manejador de objetos que ya están creados o ninguno (null), tiene un parecido a un puntero de C ó C++.
    Realiza comparaciones (mira a ver si se refiere al mismo objeto), y copia (la referencia destino se refiere al mismo objeto que al origen).

- Nombre no usado (**null**)

### *2.3 Sentencias de Control.*

Muchas de las sentencias de control del flujo del programa se han tomado del C:

**Sentencias de Salto**

**if/else**

```
if( Boolean ) {
   sentencias;
   }
else {
   sentencias;
}
```

**switch**

```
switch( expr1 ) {
   case expr2:
      sentencias;
      break;
   case expr3:
      sentencias;
      break;
   default:
      sentencias;
      break;
   }
```

***Sentencias de Bucle***

**Bucles for**

```
for( expr1 inicio; expr2 test; expr3 incremento ) {
   sentencias;
   }
```

El ejemplo siguiente dibuja varias líneas en pantalla alternando sus colores entre rojo, azul y verde. Este fragmento sería parte de una función Java (método):

```
int contador;
for( contador=1; contador <= 12; contador++ ) {
   switch( contador % 3 ) {
      case 0:
         setColor( Color.red );
         break;
      case 1:
         setColor( Color.blue );
         break;
      case 2:
         setColor( Color.green );
         break;
      }
   g.drawLine( 10,contador*10,80,contador*10 );
   }
```

También se soporta el operador *coma* (,) en los bucles for

```
for( a=0,b=0; a < 7; a++,b+=2 )
```

**Bucles while**

```
while( Boolean ) {
   sentencias;
   }
```

**Bucles do/while**

```
do {
   sentencias;
   }while( Boolean );
```

***Excepciones***

**try-catch-throw**

```
try {
   sentencias;
} catch( Exception ) {
   sentencias;
}
```

Java implementa excepciones para facilitar la construcción de código robusto. Cuando ocurre un error en un programa, el código que encuentra el error lanza una excepción, que se puede capturar y recuperarse de ella. Java proporciona muchas excepciones predefinidas.

***Control General del Flujo***

```
break [etiqueta]
continue [etiqueta]
return expr;
etiqueta: sentencia;
```

En caso de que nos encontremos con bucles anidados, se permite el uso de etiquetas para poder salirse de ellos, por ejemplo:

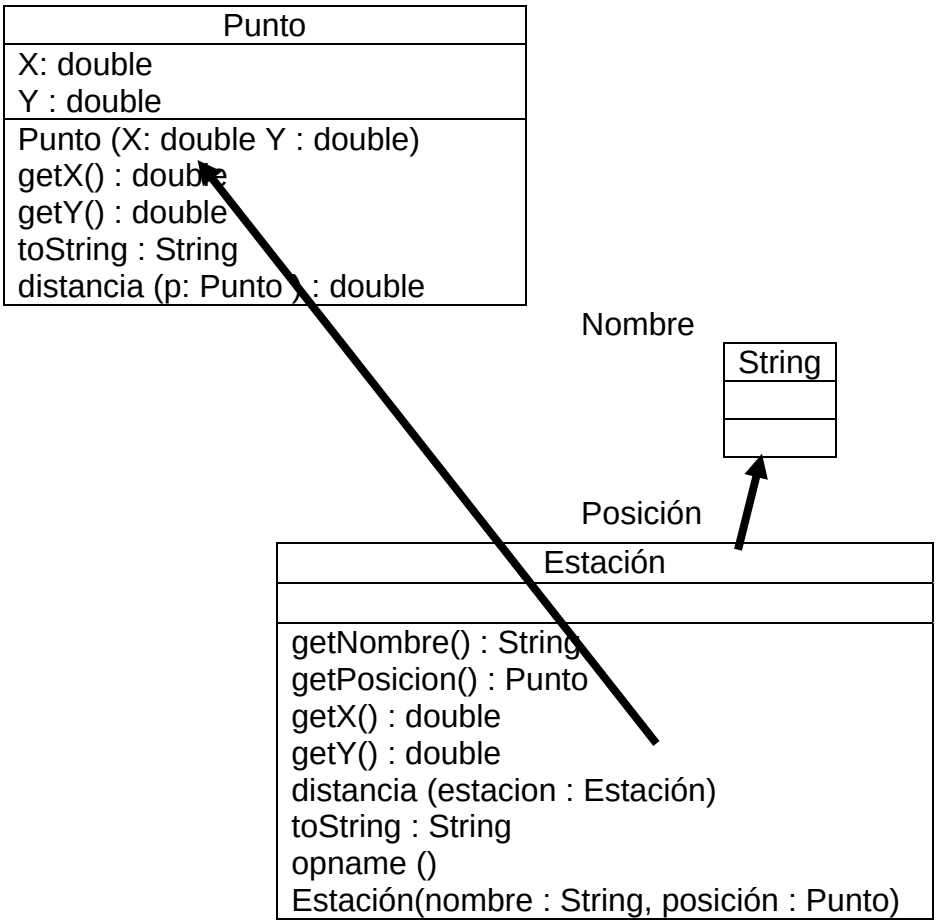
```
uno: for( )
        {
    dos: for( )
            {
            continue;           // seguiría en el bucle interno
            continue uno;       // seguiría en el bucle principal
            break uno;          // se saldría del bucle principal
            }
        }
```

En el código de una función siempre hay que ser consecuentes con la declaración que se haya hecho de ella. Por ejemplo, si se declara una función para que devuelva un entero, es imprescindible que se coloque un *return* final para salir de esa función, independientemente de que haya otros en medio del código que también provoquen la salida de la función. En caso de no hacerlo se generará un *Warning*, y el código Java no se puede compilar con Warnings.

```
int func()
{
if( a == 0 )
   return 1;
return 0;    // es imprescindible porque se retorna un entero
```

## *2.4 Composición y extensión de clases, interfaces.*

En el siguiente gráfico podemos ilustrar una relación de composición que quiere decir hacer los programas más pequeños.




```
Class estacion {
        Private String nombre;
        Private Punto posición;
}
```

**Métodos**

Una operación que realiza acceso a los datos. Podemos definir método como un programa procedimental o procedural escrito en cualquier lenguaje, que está asociado a un objeto determinado y cuya ejecución sólo puede desencadenarse a través de un mensaje recibido por éste o por sus descendientes.

Son sinónimos de 'método' todos aquellos términos que se han aplicado tradicionalmente a los programas, como procedimiento, función, rutina, etc. Sin embargo, es conveniente utilizar el término 'método' para que se distingan claramente las propiedades especiales que adquiere un programa en el entorno OOP, que afectan fundamentalmente a la forma de invocarlo (únicamente a través de un mensaje) y a su campo de

acción, limitado a un objeto y a sus descendientes, aunque posiblemente no a todos.

Si los métodos son programas, se deduce que podrían tener argumentos, o parámetros. Puesto que los métodos pueden heredarse de unos objetos a otros, un objeto puede disponer de un método de dos maneras diferentes:

-*Métodos propios.* Están incluidos **dentro** de la cápsula del objeto.

-*Métodos heredados.* Están definidos en un objeto diferente, antepasado de éste (padre,"abuelo", etc.). A veces estos métodos se llaman **método miembro** porque el objeto los posee por el mero hecho de ser miembro de una clase.

Polimorfismo

Una de las características fundamentales de la OOP es el polimorfismo, que no es otra cosa que la posibilidad de construir varios métodos con el mismo nombre, pero con relación a la clase a la que pertenece cada uno, con comportamientos diferentes. Esto conlleva la habilidad de enviar un mismo mensaje a objetos de clases diferentes. Estos objetos recibirían el mismo mensaje global pero responderían a él de formas diferentes; por ejemplo, un mensaje "+" a un objeto ENTERO significaría suma, mientras que para un objeto STRING significaría concatenación ("pegar" strings uno seguido al otro)

Demonios

Es un tipo especial de métodos, relativamente poco frecuente en los sistemas de OOP, que se activa automáticamente cuando sucede algo especial. Es decir, es un programa, como los métodos ordinarios, pero se diferencia de estos porque su ejecución no se activa con un mensaje, sino que se desencadena automáticamente cuando ocurre un suceso determinado: la asignación de un valor a una propiedad de un objeto, la lectura de un valor determinado, etc.

Los demonios, cuando existen, se diferencian de otros métodos por que no son heredables y porque a veces están ligados a una de las propiedades de un objeto, mas que al objeto entero.

Los métodos son funciones de la clase, maneja los campos de un objeto, realiza llamadas (nombre del objeto.nombre del método), tiene parámetros (valores auxiliares necesarios) y por último devuelven valores primitivos, objetos, void.

Existen tres tipos fundamentales de métodos:

a) ***Constructores.-*** tienen el mismo nombre de la clase y sirven para crear el objeto pidiendo memoria a la Memoria Virtual (new) y colocando el estado inicial constructor. Ejm:

```
Public Conjunto (){

    Numeros = new bolean [MaxElems];
    IdConjunto = nConjs;
    nConjs ++;

}
```

b) ***Main.-*** Sirven para ejecutar una aplicación JAVA, se da el nombre de la clase que se dirige. Busca un método llamado main, debe ser estático. Ejm:

```
Class Echo {

    Public static void main (String[] args){
            for (int i = 0; i < args.length; i ++)
                    System.out.print(args[i] + ¨ ¨ );

    }
}
```

c) ***Estáticos.-*** Utiliza memoria estática. Ejm:

```
Class Punto {

        Private static long numPuntos = 0
        Private static long totalPuntos  ( ){
                Return numPuntos;
        }

        private double x
        private double y;

        Public Punto (double x, double y){
                This.x;
                This.y;
                numPuntos++;
}
        }.
```

## *2.5 Bibliotecas y módulos.*

A continuación revisaremos  los que es un módulo

El módulo lo podemos definir igual que una carpeta o directorio en el cual puede tener subdirectorios:

- Al principio de cada fichero del módulo:

  Package modulo;

- Si dentro del modulo hay nuevos módulos:

  Package modulo.submodulos;

- Variable de entorno CLASSPATH

  Directorio donde buscar los módulos

- Compilar todo el módulo junto

  Javac *.java

- Ejecución

  Java –classpath  $ CLASSPATH

  modulo.clase

En el uso de los módulos el nombre se usa como prefijo de los elementos que contiene ejm:

// importa todo, poner después de la declaración de módulo

import modulo

// importación explícita, después de declaración de módulo

import modulo.clase1;

// uso

modulo.clase1 a = neww modulo.clase1 ();

Es muy importante evitar las importaciones circulares e importar explícitamente.

En la sección de anexos ilustraremos algunas bibliotecas más utilizadas con algunos ejemplos.

# 3 Acceso a facilidades de protocolo TCP, UDP, URL, HTTP, RTP.

## *3.1 Acceso al Servicio DNS en aplicaciones.*

El servicio DNS (Domain Name Server) traduce las direcciones IP a nombres de dominio internet. Este servicio a veces resulta necesario para poder acceder a ciertos recursos de la red (bases de datos, ftp anonymous, aplicaciones).

Para darse de alta en este servicio es imprescindible obtener la dirección IP del equipo. Que según el sistema operativo se hace de la siguiente forma:

Cualquier ordenador con una pegatina. En ella tiene apuntado la dirección IP.

Windows 95/98: Menú Inicio / Ejecutar / winipcfg

Windows NT 4.0: Menú Inicio / Configuración / Panel de control / Red / Protocolos

Seleccionar el Protocolo TCP/IP y pulsar Propiedades.

Windows 3.1: Abrir el trumpet y mirar en las frases escritas al principio la dirección IP.

MS-DOS: Ejecutamos lo siguiente:

- cd \ncsa
- edit config.tel
- mirar la dirección IP que viene al principio del fichero.

**Descripción del servicio**

El DNS (Domain Name System) es el sistema empleado en Internet para poder asignar y usar universalmente nombres unívocos para referirse a los equipos conectados a la red. De esta forma, tanto los usuarios como las aplicaciones pueden emplear nombres de DNS en lugar de direcciones numéricas de red IP. Esto presenta grandes ventajas, entre ellas el hecho de que, para una persona, es más cómodo y nemotécnico el uso de nombres frente al uso de números y por otra parte, permite a una organización independizar el nombre de máquinas, servicios, direcciones de correo electrónico, etc. de las direcciones numéricas concretas que en un determinado momento puedan tener sus equipos

en función de aspectos cambiantes tales como la topología de la red y el proveedor de acceso a Internet.

Técnicamente el DNS es una inmensa base de datos distribuida jerárquicamente por toda la Internet; existen infinidad de servidores que interactúan entre sí para encontrar y facilitar a las aplicaciones clientes que los consultan la traducción de un nombre a su dirección de red IP asociada con la que poder efectuar la conexión deseada. Cada parte de la base de datos está replicada en al menos dos servidores, lo que asegura una debida redundancia.

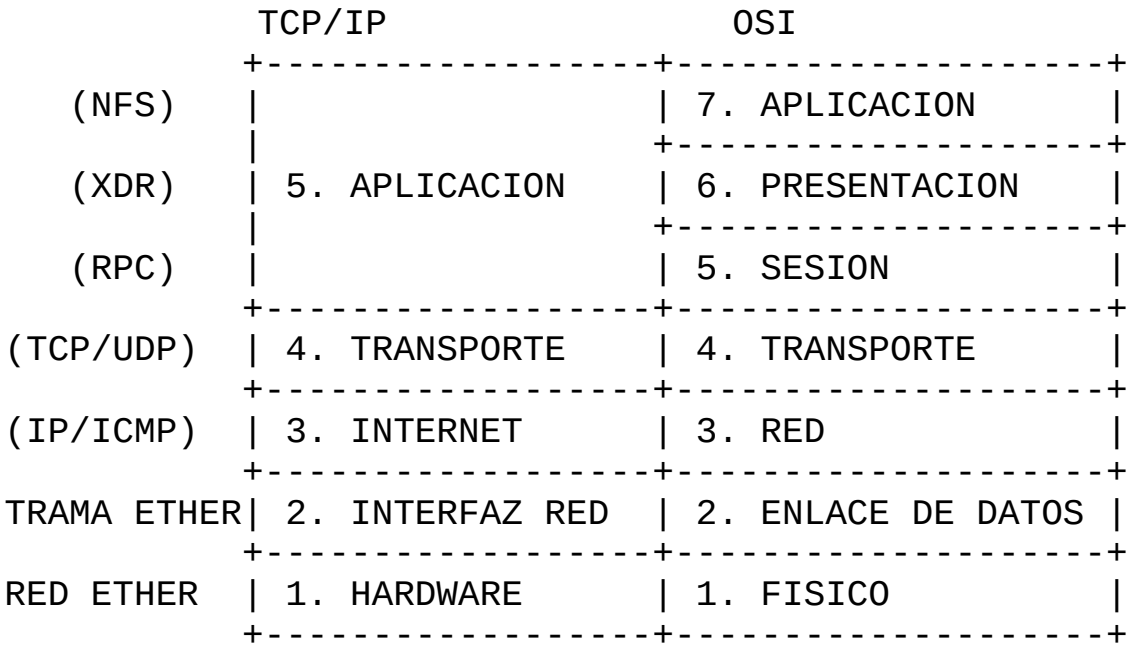
La razón que motivó el desarrollo e implantación del DNS en Internet fue el gran crecimiento en el número de máquinas conectadas. Anteriormente, la asociación entre nombres y direcciones IP se hacía por medio de un listado mantenido centralmente en un único fichero que debía ser constantemente actualizado con cada nuevo equipo conectado y que debía residir en todos y cada uno de los ordenadores conectados a Internet. El mantenimiento de este sistema se hizo inviable en cuanto el número de equipos conectados llegó a unos pocos miles a mediados de los años 80.

La finalidad del DNS es la de permitir el escalado, tanto desde un punto de vista administrativo como desde un punto de vista técnico, del sistema de nombres de Internet, por medio de una distribución jerárquica de dominios delegados. Los dominios son entidades administrativas cuyo propósito es subdividir la carga de gestión de un administrador central repartiéndola entre distintos subadministradores. Estos, a su vez, pueden repetir el proceso si el tamaño del dominio a administrar así lo aconseja. De esta forma, se pueden crear distintos niveles de dominios delegados, donde cada administrador asigna nombres unívocos a su nivel, garantizando así la univocidad de cualquier nombre del DNS que se forma por yuxtaposición (separada por puntos ".") de los distintos nombres de dominio de abajo a arriba en la jerarquía, hasta llegar al ultimo (denominado raíz del DNS o "."); por ejemplo: maquina.nivel3.nivel2.nivel1.

## *3.2 Uso de protocolos TCP y UDP, Sockets y Datagramas.*

### El modelo TCP/IP.

Los cinco niveles del modelo TCP/IP son:




- **Nivel Interfaz de Red y Hardware:**

Agrupa los bits en tramas para el manejo de la información y se ocupa de las características técnicas de la red ( voltajes, pines ).

- **Nivel Internet:**

Se corresponde con el nivel de Red OSI y controla el direccionamiento de la información. El **IP** se trata de un protocolo para el **intercambio de datagramas en modo no conectado**. Esto no garantiza la llegada de mensajes (cosa que se hará con el TCP). El algoritmo de direccionamiento de Internet se basa en **tablas de direccionamiento** de los datagramas difundidos por los gateways.

- **Nivel de Transporte**:

Se corresponde con el de Transporte OSI, garantizando la seguridad de la conexión y el control del flujo. Incluye el *Protocolo de Control de Transmisión* ( **TCP** ) y el *Protocolo de Datagrama de Usuario* ( **UDP** ).

* El **TCP** es un protocolo **orientado a conexión** que transporta de forma segura grupos de octetos ( segmentos ) **modo duplex** (en los dos

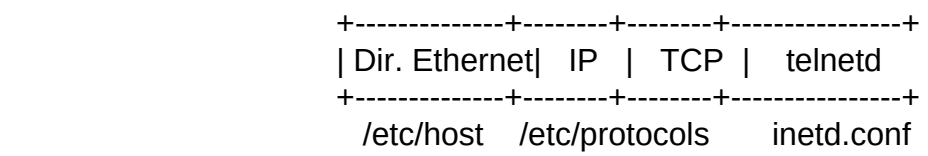
sentidos).

Utiliza el mecanismo de **puerto** ( al igual que el protocolo de transporte **UDP**, pero que actúa en modo datagrama no conectado ). Este mecanismo se basa en la asignación para cada uno de los protocolos del nivel de transporte (TCP o UDP) de un conjunto de puertos de E/S identificados mediante un número entero. Así TCP y UDP **multiplexarán** las conexiones por medio de los números de los puertos. Existen una serie de **puertos reservados** a aplicaciones estándares Internet (ECHO - puerto 7, FTP - puerto 21, TELNET - puerto 23). El archivo **/etc/services** contiene la lista de los puertos estándar. En UNIX están reservados los números de puerto inferiores a 1024. El resto pueden ser utilizados, cualidad fundamental que aprovechan los programas definidos por el usuario para el establecimiento de comunicaciones entre hosts.

- **Nivel de Aplicación**:

Incluye los niveles OSI de sesión, presentación y aplicación. Ejemplos de estos niveles son el *telnet, ftp* o el sistema de ficheros de red *NFS* para el nivel de aplicación, el lenguaje de descripción de información *XDR* para el nivel de presentación o la interfaz de llamada a procedimientos remotos *RPC*.

-> Por ejemplo, el formato de una trama telnet sería:




**Arquitectura y direccionamiento.**

Con respecto a la arquitectura de Internet, la unión de las redes se basa en la existencia de gateways entre ellas. Pero lo más importante dentro de esta incompatibilidad de redes es otorgar a los hosts implicados una **dirección lógica** que se componga de:

- una **dirección de red**

- una **dirección de la máquina dentro de la red**

La dirección completa ocupa 32 bits, dándose en forma de 4 octetos:

$$n_1.n_2.n_3.n_4$$

Cada campo es un número decimal entre 0 y 255.

La dirección nula se refiere a la red.
Para direccional un mensaje a todas las máquinas ( *broadcast* ), se usa una dirección dentro de la red en la cual todos sus bits son iguales a 1.

El valor 127 en el primer campo se llama *loopback* y se refiere a una interfaz que permite al host enviarse paquetes a sí mismo.

Existen varias clase de redes:

**Redes clase A**: gran tamaño. El primer bit es 0, [1-127], con lo que su dirección son los 7 siguientes bits de la dirección de 32 bits (permite 127 redes de clase A). Los 24 bits restantes se usan para direccionar sus hosts locales.

**Redes clase B**: tamaño medio. Los dos primeros bits son 10, [128-191], y su dirección la componen los siguientes 14 bits de la dirección de 32 bits ( 16384 redes de clase B ). Los 16 bits restantes se usan para direccionar sus hosts locales.

**Redes clase C**: pequeño tamaño. Su dirección la componen los siguientes 21 bits a los tres primeros, 110, característicos [192-223] ( 2097152 redes de clase C ). Los 8 bits restantes se usan para direccionar sus hosts locales ( 256 hosts: del 0 al 255 ).

**Redes clase D: Dirección multicast:**

Sus primeros 4 bits son 1110 [224-239], indicando que es entrada de una dirección multicast. Los restantes 28 bits comprenden un grupo específico multicast. Esta dirección es una dirección destino para una o varias máquinas ( a diferencia de las clases anteriores, que se referían sólo a una máquina )

Para simplificar el direccionamiento, se utilizan **direcciones simbólicas** del tipo
**host.organización** ( dominios )

El fichero **/etc/hosts** muestra correspondencia entre las direcciones IP y los nombres de los hosts definidos por defecto.

## UDP

UDP es un protocolo que se basa en el intercambio de datagramas. No permite la numeración de los datagramas, este protocolo se utiliza principalmente cuando el orden en que se reciben los mismos no es un factor fundamental.

Permite que el envío de datagramas a través de la red sea sin que se haya establecido previamente una conexión.

Representa el nivel de servicio mínimo que utilizan muchos sistemas de aplicación basados en transacciones. Es muy útil en los casos en los que no son necesarios los servicios de TCP.

El módulo UDP (*User Datagram Protocol*) [RFC768] es un fino recubrimiento a la capa IP que cumple las funciones equivalentes al nivel OSI de Transporte de datagramas. Proporciona un servicio sin conexión y, por ello, de baja fiabilidad. No se garantiza la entrega de los datagramas, ni su llegada en orden o no duplicación. Los servicios que requieran un sistema de transporte fiable deben emplear el TCP (*Transport Control Protocol*) o algún esquema similar, como la familia de protocolos TPx de OSI.

UDP (User Datagram Protocol) es un protocolo que se basa en el intercambio de datagramas. UDP permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión, ya que el propio datagrama incorpora suficiente información de direccionamiento en su cabecera.

El inconveniente de esta forma de actuación es que no hay confirmación de recepción ni de haber recibido los datagramas en el orden adecuado, debiendo ser la aplicación la que se encargue de controlarlo.

Al igual que los protocolos TCP, utiliza puertos y sockets y, también permite la multiplexación.

El protocolo de datagramas de usuario (UDP) puede ser la alternativa al TCP en algunos casos en los que no sea necesario el gran nivel de complejidad proporcionado por el TCP. Puesto que UDP no admite numeración de los datagramas, éste protocolo se utiliza principalmente cuando el orden en que se reciben los mismos no es un factor fundamental, o también cuando se quiere enviar información de poco tamaño que cabe en un único datagrama.

Cuando se utiliza UDP la garantía de que un paquete llegue a su destino es mucho menor que con TCP debido a que no se utilizan las señales de confirmación. Por todas estas características la cabecera del UDP es bastante menor en tamaño que la de TCP. Esta simplificación resulta en una mayor eficiencia en determinadas ocasiones.

Un ejemplo típico de una situación en la que se utiliza el UDP es cuando se pretende conectar con un ordenador de la red, utilizando para ello el nombre del sistema. Este nombre tendrá que ser convertido a la dirección IP que le corresponde y, por tanto, tendrá que ser enviado a

algún servidor que posea la base de datos necesaria para efectuar la conversión. En este caso es mucho más conveniente el uso de UDP.

A pesar de todo existen servicios perfectamente válidos en un entorno de funcionamiento poco fiable y, por otra parte, en una red local -por ejemplo- la sobrecarga asociada a un protocolo más seguro puede resultar inaceptable. La calidad de servicio intrínseca es ya lo bastante elevada.

Las características de este módulo son:

Transmisión y recepción de datagramas aislados, sin el concepto de conexión

Servicio no fiable: posibles pérdidas, duplicaciones y llegada desordenada de datagramas

Posibilidad de esperar la recepción de datagramas dirigidos a ciertos canales específicos, denominados *"puertos"*

En cuanto a los servicios que utilizan este módulo, destacan:

*Servidores de Dominios de Nombres (DNS) [RFC1034] [RFC1035]*.

Estos sistemas proporcionan una funcionalidad básica en el mundo Internet. Su principal tarea consiste en efectuar la traducción entre una dirección simbólica y su dirección IP correspondiente. También son utilizados masivamente para el encaminamiento del correo electrónico.

- *Protocolos ECHO y DISCARD [RFC862] [RFC863].*

El fin primordial de estos protocolos es el chequeo y diagnóstico de sistemas.

- *Llamadas a procedimientos remotos (RPC) [RFC1057] [RFC1831]*.

Se trata de un protocolo diseñado para la implementación de arquitecturas cliente-servidor y la programación de sistemas distribuidos. Constituye la base de multitud de otros protocolos, como el NFS.

- *Sistema de ficheros en red (NFS) [RFC1094] [RFC1813]*.

Gracias a este protocolo se implementa una arquitectura de ficheros estándar, accesible a través de una red telemática. Tiene un uso masivo en las redes locales, a través de los "servidores de ficheros".

- *Distribución de las tablas de enrutado.*

Para que la red en su conjunto ofrezca una imagen homogénea y coherente es necesario que las diferentes pasarelas dispongan de información actualizada sobre su topología. Esto se consigue mediante el intercambio de información de encaminamiento entre los diferentes sistemas, vía los servicios UDP y TCP.

Interfaz

La interfaz de este módulo está constituida por procesos y por subrutinas ejecutadas en el contexto del llamante. Su utilización es muy simple.

PROC_UDP_BC

Este proceso es el encargado de inicializar este módulo. Debe ser invocado con un mensaje "MSG_INIT" o "MSG_QUIT". La inicialización de este módulo debe ser posterior a la del módulo IP.

PROC_UDP_SUP

Este proceso se hace cargo de la transmisión de datagramas. Espera mensajes "MSG_MBUF". **campo1** contiene una cadena de dos o más MBUFs, el primero de ellos conteniendo una cabecera UDP interfaz y el resto los datos en sí. La cabecera UDP interfaz se define como:

```
typedef struct {
  uint16  puerto_fuente;
  uint16  puerto_destino;
  uint16  dont_fragment;
  uint16  dummy;
} udp_header;
```

Los campos *"puerto_fuente"* y *"puerto_destino"* identifican el puerto UDP al cual va dirigido el datagrama dentro de la máquina destino, y el puerto al cual tendrá que dirigir su posible respuesta. Si *"dont_fragment"* vale cero se admite la fragmentación del datagrama, mientras que si vale uno será destruido si excede el MTU de alguna de las redes intermedias. El valor de *"dummy"* es arbitrario.

**campo2** contiene la dirección IP fuente (una de las nuestras), mientras que en **campo3** se envía la dirección IP del destino. Ambos valores deben estar en formato HOST.

PROC_UDP_INF

Este es el proceso que procesa los datagramas UDP recibidos por la capa IP, y gestiona los mensajes de control provenientes de ICMP.

MSG_MBUF

Este es el tipo de mensaje más normal que este proceso espera recibir. Contiene un datagrama UDP según el formato definido por la capa IP.
Este proceso también envía mensajes "MSG_MBUF" hacia arriba. En ese caso **campo1** contiene una cabecera UDP interfaz y el datagrama en sí (en dos MBUFs, respectivamente). **campo2** contiene la dirección origen y **campo3** la dirección destino del datagrama. Los campos *"dont_fragment"* y *"dummy"* de la cabecera UDP interfaz contendrán valores indefinidos.

MSG_ICMP_SOURCE_QUENCH

El proceso recibe este mensaje de la capa ICMP. El formato es el definido en el capítulo dedicado a ICMP. No se informa a la aplicación superior.

MSG_ICMP_DEST_UNREACHABLE

Si el proceso recibe este mensaje proveniente de la capa ICMP, envía hacia arriba, al proceso que gestiona el puerto fuente original, otro mensaje "MSG_ICMP_DEST_UNREACHABLE" con **campo1** obedeciendo al formato UDP interfaz, mientras que **campo2** proporciona información adicional (ver el capítulo ICMP) y **campo3** contiene la dirección IP de la máquina a la que no se puede llegar (formato HOST).

En cuanto a rutinas, tenemos:

estado udp_alta_puerto(uint16 puerto,proc_id proc_retorno);

Esta rutina da de alta un puerto UDP para escucha. Cualquier datagrama que llegue al puerto *"puerto"* será enviado al proceso cuyo identificador sea *"proc_retorno"*. Esta rutina retorno "OK" si todo fue bien, "ERR_EN_USO" si el puerto especificado ya ha sido declarado con anterioridad y "ERR_OVERFLOW" si hay demasiado puertos declarados.

uint16 udp_alta_puerto_arbitrario(proc_id proc_retorno);

Esta rutina es idéntica a la anterior pero asigna un puerto cualquiera que esté libre. Por compatibilidad con posibles servicios superiores, el puerto asignado será siempre mayor o igual que 1024. Retorna el valor del puerto asignado, o cero si hay demasiados puertos declarados.

void udp_baja_puerto(uint16 puerto);

Esta rutina da de baja un puerto previamente declarado. Cualquier datagrama UDP que se reciba hacia ese puerto será rechazado con un error ICMP.
estado udp_flujo(uint32 ip,uint16 tamanho);

Esta rutina proporciona un mecanismo de control de flujo UDP. Cuando algún proceso desea enviar un datagrama utilizando este protocolo debe pedir permiso antes mediante el empleo de esta llamada. *"ip"* contiene una de nuestras direcciones origen y *"tamanho"* es el número de bytes que deseamos transmitir. Retorna "OK" si tenemos el flujo abierto, "FLUJO_LLENO" si nos prohíbe la transmisión y "ERR_INVALID" si la dirección especificada no es nuestra. Si el control de flujo está cerrado podemos reintentarlo otra vez tras un tiempo prudencial (una décima de segundo, por ejemplo).

Aún cuando el control de flujo esté cerrado, la capa UDP se hará cargo de cualquier datagrama que se envíe. Esto es así para simplificar ciertos protocolos, pero no debería abusarse de esta característica si no se está completamente seguro de sus implicaciones.

Implementación

Este módulo sigue escrupulosamente todas las guías definidas en [RFC768], implantándolo completamente. Aunque el protocolo original tiene la posibilidad de no emplear suma de control en los datagramas, hemos decidido implementarla debido a que ello no supone una pérdida de eficiencia detectable.

Todos los datagramas salientes, por tanto, incluyen una suma de control. No obstante, por compatibilidad, se admiten datagramas UDP entrantes tanto con suma de control como sin ella.

En la implementación actual los mensajes de control de flujo, "MSG_ICMP_SOURCE_QUENCH", son ignorados. La razón de ello es que no resulta sencillo definir un mecanismo de control de flujo y congestión en un sistema de datagramas. Por otra parte el protocolo UDP no suele suponer una carga apreciable y los esquemas de temporización y reintentos de las capas superiores suelen ser suficientes para reducir este problema.

Cualquier datagrama UDP que se reciba hacia un puerto no declarado será destruido y se generará un mensaje "MSG_ICMP_DEST_UNREACHABLE" con el código "ICMP_NO_PUERTO" para informar a la máquina fuente de que el puerto especificado no existe. Si lo que llega es un datagrama con una suma de control errónea (si tiene suma de control), lo eliminamos sin ninguna acción adicional. No podemos arriesgarnos a generar un mensaje ICMP porque el datagrama estaba corrupto y no podemos contar con que la información de encaminamiento recibida sea correcta.

Encabezado de UDP

El encabezado de UDP proporciona:

Puerto de origen: número de puerto de 16 bits

Puerto de destino: número de puerto de 16 bits

Longitud (del encabezado de UDP + los datos): octetos de 16 bits en el datagrama de UDP

Checksum de UDP (16 campos de 16 bits); si es 0, no hay ningún checksum, sino hay un checksum sobre el pseudo-encabezado + área de datos de UDP

El pseudo encabezado incluye las direcciones IP de origen y destino (garantizando así que el mensaje ha llegado a la máquina y puerto correcto).

Note la violación estricta de capa (UDP es un protocolo de capa 4); usa información de la capa 3.

A pesar de su sencillez y de haber sido opacado por TCP, poderosas y muy utilizadas aplicaciones se basan en UDP. Entre ellas están:

NFS (*Network File System*): permite utilizar discos de estaciones remotas como si fueran propios.

DNS (*Domain Name Server*): servicio de nombres

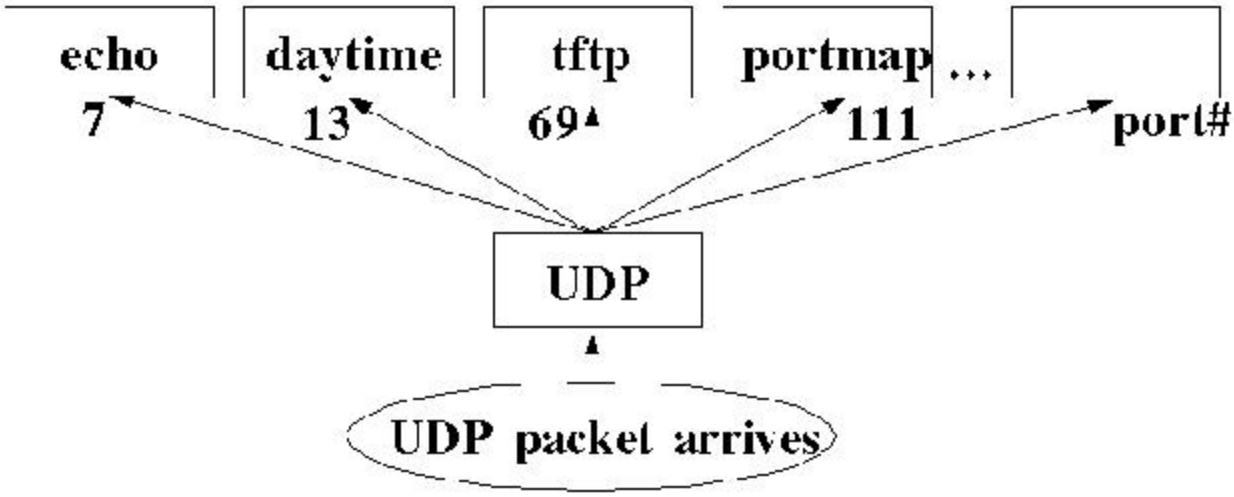
SNMP (*Simple Network ManagementProtocol*)

A continuación daré una reseña para saber a que puerto se debe enviar.

Existen números de puerto UDP Reservados y Disponibles. Asignación universal (asignaciones del puerto "bien conocido") vs. Atado Dinámico Internet usa una mezcla de estos dos con 255 puertos reservados:

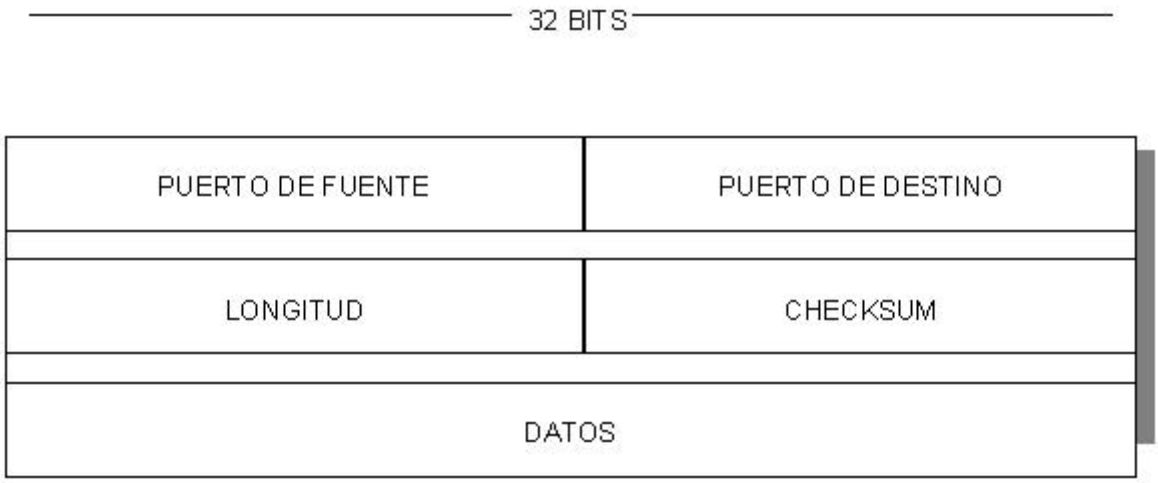
| **nombre oficial del servicio** | **número de port/nombre de protocolo** | **alias** | |
|---|---|---|---|
| Echo | 7/udp | | # |
| Discard | 9/udp | sink null | # |
| Daytime | 13/udp | | # |
| Charlen | 19/udp | ttytst source | # |
| Time | 37/udp | timeserver | # |
| Rlp | 39/udp | resource | # Resource Location Protocol |
| Domain | 53/udp | nameserver | # |
| bootps | 67/udp | | # Bootstrap Protocol Server |
| bootpc | 68/udp | | # Bootstrap Protocol Client |
| tftp | 69/udp | | # Trivial File Transfer Protocol |
| portmap | 111/udp | sunrpc | # |
| ntp | 123/udp | | # Network Time Protocol |
| netbios_ns | 137/udp | | # |
| netbios_dgm | 138/udp | | # |
| netbios_ssn | 139/udp | | # |
| snmp | 161/udp | snmpd | # Simple Network Management Protocol Agent |
| snmp-trap | 162/udp | trapd | # Simple Network Management Protocol Traps |
| # | | | |
| biff | 512/udp | comsat | # mail notification |
| who | 513/udp | whod | # remote who and uptime |
| syslog | 514/udp | | # remote system logging |
| talk | 517/udp | | # conversation |
| ntalk | 518/udp | | # new talk, conversation |
| route | 520/udp | router routed | # routing information protocol |
| timed | 525/udp | timeserver | # remote clock synchronization |
| nfsd | 2049/udp | | # NFS remote file system |
| kerberos | 750/udp | kdc | # Kerberos (server) udp -kfall |

Demultiplexado basado en los puertos de protocolo (o simplemente "puertos") por UDP:




Formato del mensaje UDP

La mejor forma de explicar este protocolo es examinar el mensaje y los campos que lo componen. Como muestra la figura, el formato es muy simple e incluye los siguientes campos:



***Puerto de fuente:*** Este valor identifica el puerto del proceso de aplicación remitente. Este campo es opcional. Si no se utiliza. Se pone a 0.

***Puerto de destino:*** Este valor identifica el proceso de recepción en el computador de destino.

***Longitud:*** Este valor indica la longitud del datagrama de usuario. Incluyendo la cabecera y los datos. La longitud mínima es de 8 octetos.

***Checksum:*** Este valor contiene el valor del complemento a 1 en 16 bits del complemento a 1 de la suma de la seudocabecera de IP, la cabecera de UDP y los datos. Se realiza también el checksum de los campos de relleno ( si es necesario que el mensaje contenga un número de octetos que sea un múltiplo de dos).

Poco más se puede decir de UDP. Representa el nivel de servicio mínimo que utilizan muchos sistemas de aplicación basados en transacciones. Es, sin embargo, muy útil en los casos en los que no son necesarios los servicios de TCP.

Comparativa TCP / UDP

Para calcular los tiempos de ida y vuelta de paquetes en TCP y UDP se han realizado 1000 envíos por tamaño de paquete en cada uno de los protocolos, utilizando para la comparación los tiempos medios obtenidos en cada lote de 1000 envíos.

La aplicación cliente se ha ejecutado en un ordenador de la red 176 de la UVA, y la aplicación servidor en el servidor Duero (red 125).

Para las pruebas en TCP se ha abierto una sola sesión para cada lote de 1000 paquetes.

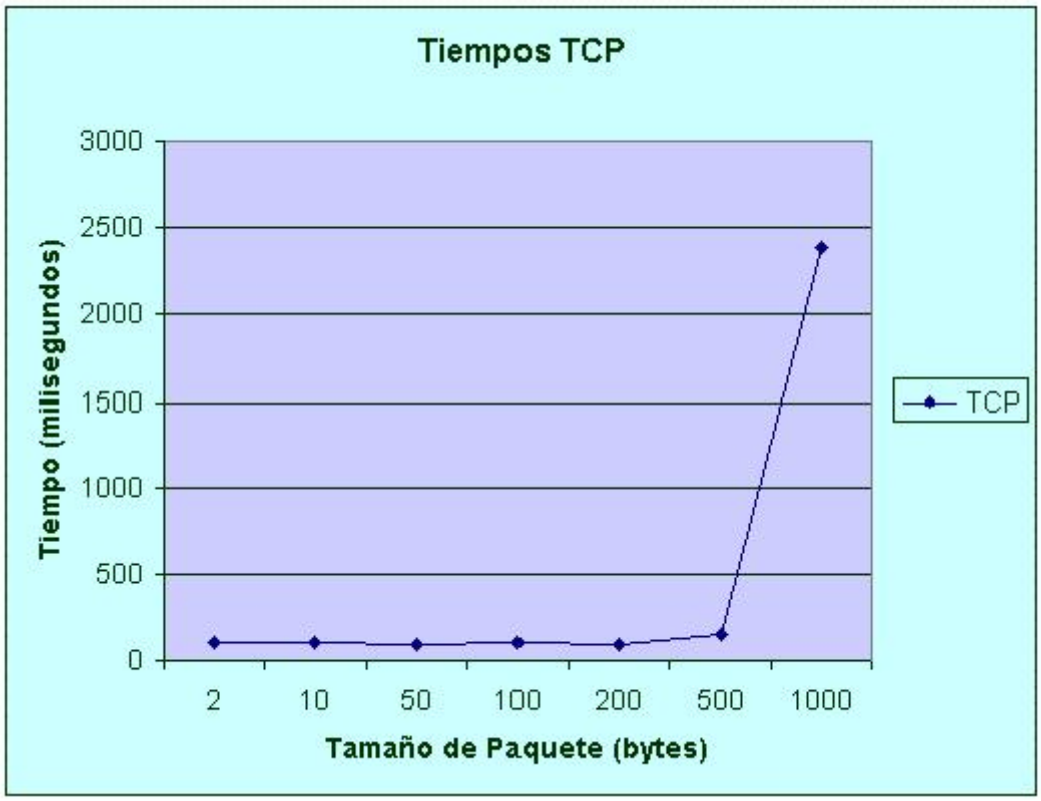
Así mismo se han comprobado los tiempos para 7 tamaños de paquete: 2, 10, 50, 100, 200, 500 y 1000 bytes respectivamente.

Para cada uno de esos tamaños se han obtenido los siguientes tiempos medios (expresados en milisegundos):

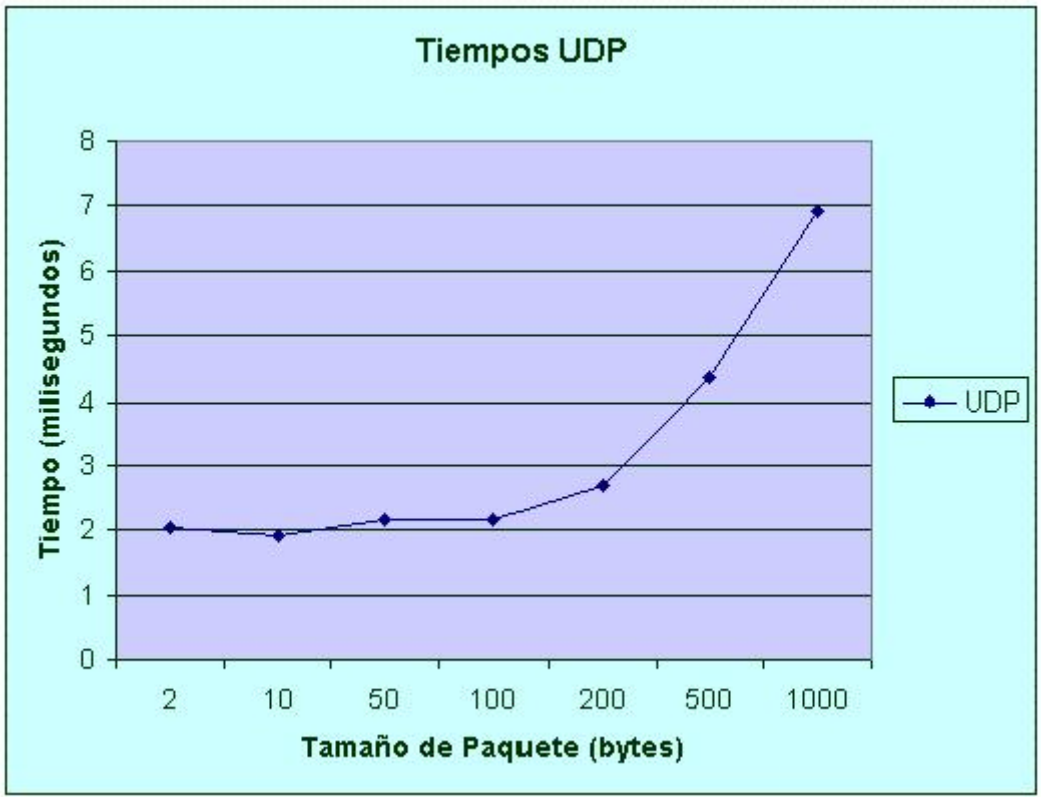
| Tamaño de paquete | Tiempo TCP | Tiempo UDP |
|---|---|---|
| 2 | 99.63 | 2.03 |
| 10 | 100.22 | 1.91 |
| 50 | 97.32 | 2.17 |
| 100 | 103.17 | 2.15 |
| 200 | 96 | 2.69 |
| 500 | 155.99 | 4.36 |
| 1000 | 2381.88 | 6.91 |

A partir de dichos datos se obtienen las siguientes gráficas comparativas:

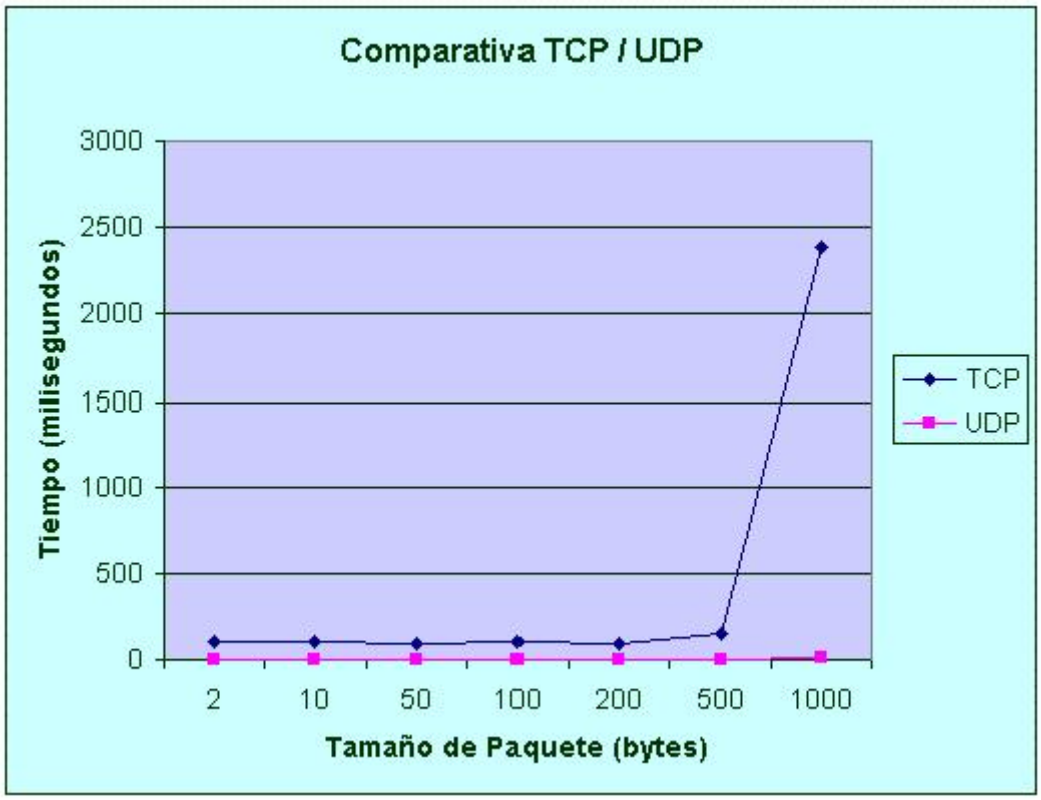
- Tiempos TCP



- Tiempos UDP




- Comparativa TCP/UDP



Como se puede observar, hasta 200 bytes los tiempos son muy similares para todos los tamaños, pudiéndose deber las pequeñas fluctuaciones al estado de la red en cada momento. Mientras que a partir de 500 bytes el tiempo de ida y vuelta de los mensajes aumenta significativamente.

Así mismo se observa que el servicio UDP es mucho más rápido, y cuanto mayor es el tamaño de los paquetes más se nota la diferencia de velocidad entre ambos protocolos.

Con esta investigación se puede llegar a la conclusión de que el protocolo de datagramas de usuario (UDP) proporciona un servicio sin conexión y por ello, baja su fiabilidad.

El inconveniente de usar datagramas sin establecer una conexión previamente es que no existe una confirmación de recepción ni de haber recibido los datagramas en el orden adecuado, debiendo ser la aplicación la que se encargue de controlarlo.

Al igual que los protocolos TCP, utilizan puertos sockets y, también permiten la multiplexación.

**LOS SOCKETS.**

Los *sockets* no son más que puntos o mecanismos de comunicación entre procesos que permiten que un proceso hable ( emita o reciba información ) con otro proceso incluso estando estos procesos en distintas máquinas. Esta característica de interconectividad entre máquinas hace que el concepto de socket nos sirva de gran utilidad. Esta interfaz de comunicaciones es una de las distribuciones de Berkeley al sistema UNIX, implementándose las utilidades de interconectividad de este Sistema Operativo ( *rlogin, telnet, ftp, ...* ) usando sockets.

Un **socket** es al sistema de comunicación entre ordenadores lo que un buzón o un teléfono es al sistema de comunicación entre personas: un punto de comunicación entre dos agentes ( procesos o personas respectivamente ) por el cual se puede emitir o recibir información. La forma de referenciar un socket por los procesos implicados es mediante un **descriptor** del mismo tipo que el utilizado para referenciar ficheros. Debido a esta característica, se podrá realizar redirecciones de los archivos de E/S estándar (descriptores 0,1 y 2) a los sockets y así combinar entre ellos aplicaciones de la red. Todo nuevo proceso creado heredará, por tanto, los descriptores de sockets de su padre.

La comunicación entre procesos a través de sockets se basa en la filosofía **CLIENTE-SERVIDOR**: un proceso en esta comunicación actuará de **proceso servidor** creando un socket cuyo nombre conocerá

el **proceso cliente**, el cual podrá "hablar" con el proceso servidor a través de la conexión con dicho **socket nombrado**.

El proceso crea un socket sin nombre cuyo valor de vuelta es un descriptor sobre el que se leerá o escribirá, permitiéndose una comunicación **bidireccional**, característica propia de los sockets y que los diferencia de los **pipes**, o canales de comunicación unidireccional entre procesos de una misma máquina. El mecanismo de comunicación vía sockets tiene los siguientes pasos:

1º) El proceso servidor crea un socket con nombre y espera la conexión.

2º) El proceso cliente crea un socket sin nombre.

3º) El proceso cliente realiza una petición de conexión al socket servidor.

4º) El cliente realiza la conexión a través de su socket mientras el proceso servidor mantiene el socket servidor original con nombre.

Es muy común en este tipo de comunicación lanzar un proceso hijo, una vez realizada la conexión, que se ocupe del intercambio de información con el proceso cliente mientras el proceso padre servidor sigue aceptando conexiones. Para eliminar esta característica se cerrará el descriptor del socket servidor con nombre en cuanto realice una conexión con un proceso socket cliente.

Todo socket viene definido por dos características fundamentales:

- El **tipo** del socket, que indica la naturaleza del mismo, el tipo de comunicación que puede generarse entre los sockets.

- El **dominio** del socket especifica el conjunto de sockets que pueden establecer una comunicación con el mismo.

Vamos a estudiar con más detalle estos dos aspectos:

**Tipos de sockets.**
Define las **propiedades** de las comunicaciones en las que se ve envuelto un socket, esto es, el tipo de comunicación que se puede dar entre cliente y servidor. Estas pueden ser:

Fiabilidad de transmisión.

Mantenimiento del orden de los datos.

No duplicación de los datos.

El "Modo Conectado" en la comunicación.

Envío de mensajes urgentes.

Los tipos disponibles son los siguientes:

* Tipo **SOCK_DGRAM**: sockets para comunicaciones en **modo no conectado**, con envío de **datagramas** de tamaño **limitado** ( tipo telegrama ). En dominios Internet como la que nos ocupa el protocolo del nivel de transporte sobre el que se basa es el **UDP**.

* Tipo **SOCK_STREAM**: para comunicaciones **fiables en modo conectado**, de **dos vías** y con tamaño **variable** de los mensajes de datos. Por debajo, en dominios Internet, subyace el protocolo **TCP**.

* Tipo **SOCK_RAW**: permite el acceso a protocolos de más bajo nivel como el **IP** ( nivel de red )

* Tipo **SOCK_SEQPACKET**: tiene las características del SOCK_STREAM pero además el tamaño de los mensajes es fijo.

**El dominio de un socket.**

Indica el formato de las direcciones que podrán tomar los sockets y los protocolos que soportarán dichos sockets.

La estructura genérica es:

```
struct sockaddr {

        u__short    sa__family;        /* familia */
        char        sa__data[14];      /* dirección */

    };
```

Pueden ser:

* Dominio **AF_UNIX** ( Address Family UNIX ):

El cliente y el servidor deben estar en la misma máquina. Debe incluirse el fichero cabecera **/usr/include/sys/un.h**. La estructura de una dirección en este dominio es:

```
struct sockaddr__un {

short        sun__family;  /* en este caso AF_UNIX */
char         sun__data[108]; /* dirección */
   };
```

* Dominio **AF_INET** ( Address Family INET ):

El cliente y el servidor pueden estar en cualquier máquina de la red Internet. Deben incluirse los ficheros cabecera **/usr/include/netinet/in.h, /usr/include/arpa/inet.h, /usr/include/netdb.h**. La estructura de una dirección en este dominio es:

```
struct in__addr {
   u__long     s__addr;
};

struct sockaddr__in {
short         sin_family;  /* en este caso AF_INET */
u__short     sin_port;   /* numero del puerto */
struct in__addr         sin__addr; /* direcc Internet */
char         sin_zero[8];    /* campo de 8 ceros */

   };
```

Estos dominios van a ser los utilizados en xshine. Pero existen otros como:

* Dominio **AF_NS**:

Servidor y cliente deben estar en una red XEROX.

* Dominio **AF_CCITT**:

Para protocolos CCITT, protocolos X25

**FILOSOFIA CLIENTE-SERVIDOR: el Servidor.**

Vamos a explicar el proceso de comunicación servidor-cliente en **modo conectado**, modo utilizado por las aplicaciones estándar de Internet (telnet, ftp). El servidor es el proceso que crea el socket no nombrado y acepta las conexiones a él. El orden de las llamadas al sistema para la realización de esta función es:

1º) int **socket** ( int *dominio*, int *tipo*, int *protocolo* )
crea un socket sin nombre de un dominio, tipo y protocolo específico

*dominio* : AF_INET, AF_UNIX

*tipo* : SOCK__DGRAM, SOCK__STREAM
*protocolo* : 0 ( protocolo por defecto )

2º) int **bind** ( int *dfServer*, struct sockaddr* *direccServer*, int *longDirecc* )

Nombra un socket: asocia el socket no nombrado de descriptor *dfServer* con la dirección del socket almacenado en *direccServer*. La dirección depende de si estamos en un dominio AF_UNIX o AF_INET.

3º) int **listen** ( int *dfServer*, int *longCola* )

Especifica el máximo número de peticiones de conexión pendientes.

4º) int **accept** ( int *dfServer*, struct sockaddr* *direccCliente*, int* *longDireccCli*)

Escucha al socket nombrado servidor *dfServer* y espera hasta que se reciba la petición de la conexión de un cliente. Al ocurrir esta incidencia, crea un socket sin nombre con las mismas características que el socket servidor original, lo conecta al socket cliente y devuelve un descriptor de fichero que puede ser utilizado para la comunicación con el cliente.

**El Cliente.**

Es el proceso encargado de crear un socket sin nombre y posteriormente enlazarlo con el socker servidor nombrado. O sea, es el proceso que demanda una conexión al servidor. La secuencia de llamadas al sistema es:

1º) int **socket** ( int *dominio*, int *tipo*, int *protocolo* )

Crea un socket sin nombre de un dominio, tipo y protocolo específico

*dominio* : AF_INET, AF_UNIX
*tipo* : SOCK__DGRAM, SOCK__STREAM
*protocolo* : 0 ( protocolo por defecto )

2º) int **connect** ( int *dfCliente*, struct sockaddr* *direccServer*, int *longDirecc* )

Intenta conectar con un socket servidor cuya dirección se encuentra incluida en la estructura apuntada por *direccServer*. El descriptor *dfCliente* se utilizará para comunicar con el socket servidor. El tipo de estructura dependerá del dominio en que nos encontremos.

Una vez establecida la comunicación, los descriptores de ficheros serán utilizados para almacenar la información a leer o escribir.

| SERVIDOR | CLIENTE |
|---|---|
| descrServer = **socket** ( dominio, SOCK_STREAM,PROTOCOLO) | descrClient = **socket** (dominio, SOCK_STREAM,PROTOCOLO) |
| **bind** (descrServer, PuntSockServer,longServer) | |
| | do { |
| **listen** (descrServer, longCola) | |
| descrClient = **accept** (descrServer,PuntSockClient,longClient) | result = **connect** (descrClient, PuntSockServer,longserver) |
| [ **close** (descrServer) ] | } while ( result == -1 ) |
| < DIALOGO > | < DIALOGO > |
| **close** (descrClient) | **close** (descrClient) |

**COMPARACION SOCKETS-PIPES COMO MECANISMOS DE COMUNICACION ENTRE PROCESOS**

| SOCKETS | PIPES |
|---|---|
| refenciado por descriptores | referenciado por array de descriptores |
| admite comunicación entre procesos de distintas máquinas | sólo admite comunicación entre procesos de la misma máquina |
| comunicación bidireccional | comunicación unidireccional |
| filosofía cliente-servidor | simple intercambio de información |

### *3.3 Http, Uniforrm Resource Locutor.*

**El protocolo HTTP**

El Protocolo de Transferencia de HiperTexto *(Hypertext Transfer Protocol)* es un sencillo protocolo cliente-servidor que articula los

intercambios de información entre los clientes Web y los servidores HTTP. La especificación completa del protocolo HTTP 1/0 está recogida en el RFC 1945. Fue propuesto por Tim Berners-Lee, atendiendo a las necesidades de un sistema global de distribución de información como el World Wide Web.

Desde el punto de vista de las comunicaciones, está soportado sobre los servicios de conexión TCP/IP, y funciona de la misma forma que el resto de los servicios comunes de los entornos UNIX: un proceso servidor escucha en un puerto de comunicaciones TCP (por defecto, el 80), y espera las solicitudes de conexión de los clientes Web. Una vez que se establece la conexión, el protocolo TCP se encarga de mantener la comunicación y garantizar un intercambio de datos libre de errores.

HTTP se basa en sencillas operaciones de solicitud/respuesta. Un cliente establece una conexión con un servidor y envía un mensaje con los datos de la solicitud. El servidor responde con un mensaje similar, que contiene el estado de la operación y su posible resultado. Todas las operaciones pueden adjuntar un **objeto o recurso** sobre el que actúan; cada objeto Web (documento HTML, fichero multimedia o aplicación CGI) es conocido por su URL.

Los recursos u objetos que actúan como entrada o salida de un comando HTTP están clasificados por su descripción MIME. De esta forma, el protocolo puede intercambiar cualquier tipo de dato, sin preocuparse de su contenido. La transferencia se realiza en modo binario, byte a byte, y la identificación MIME permitirá que el receptor trate adecuadamente los datos.

Las principales características del protocolo HTTP son:

* Toda la comunicación entre los clientes y servidores se realiza a partir de caracteres de 8 bits. De esta forma, se puede transmitir cualquier tipo de documento: texto, binario, etc., respetando su formato original.

* Permite la transferencia de objetos multimedia. El contenido de cada objeto intercambiado está identificado por su clasificación MIME.

* Existen tres verbos básicos (hay más, pero por lo general no se utilizan) que un cliente puede utilizar para dialogar con el servidor: **GET**, para recoger un objeto, **POST**, para enviar información al servidor y **HEAD**, para solicitar las características de un objeto (por ejemplo, la fecha de modificación de un documento HTML).

* Cada objeto al que se aplican los verbos del protocolo está identificado a través de la información de situación del final de la URL.

* Cada operación HTTP implica una conexión con el servidor, que es liberada al término de la misma. Es decir, en una operación se puede recoger un único objeto.

* No mantiene estado. Cada petición de un cliente a un servidor no es influida por las transacciones anteriores. El servidor trata cada petición como una operación totalmente independiente del resto.

HTTP se diseñó específicamente para el World Wide Web: es un protocolo rápido y sencillo que permite la transferencia de múltiples tipos de información de forma eficiente y rápida. Se puede comparar, por ejemplo, con FTP, que es también un protocolo de transferencia de ficheros, pero tiene un conjunto muy amplio de comandos, y no se integra demasiado bien en las transferencias multimedia.

**Etapas de una transacción HTTP**

Para profundizar más en el funcionamiento de HTTP, veremos primero un caso particular de una transacción HTTP; en los siguientes apartados se analizarán las diferentes partes de este proceso.

Cada vez que un cliente realiza una petición a un servidor, se ejecutan los siguientes pasos:

1. Un usuario accede a una URL, seleccionando un enlace de un documento HTML o introduciéndola directamente en el campo *Location* del cliente Web.
2. El cliente Web descodifica la URL, separando sus diferentes partes. Así identifica el protocolo de acceso, la dirección DNS o IP del servidor, el posible puerto opcional (el valor por defecto es 80) y el objeto requerido del servidor.
3. Se abre una conexión TCP/IP con el servidor, llamando al puerto TCP correspondiente.
4. Se realiza la petición. Para ello, se envía el comando necesario (GET, POST, HEAD,…), la dirección del objeto requerido (el contenido de la URL que sigue a la dirección del servidor), la versión del protocolo HTTP empleada (casi siempre HTTP/1.0) y un conjunto variable de información, que incluye datos sobre las capacidades del *browser*, datos opcionales para el servidor,…
5. El servidor devuelve la respuesta al cliente. Consiste en un código de estado y el tipo de dato MIME de la información de retorno, seguido de la propia información.
6. Se cierra la conexión TCP.

Este proceso se repite en cada acceso al servidor HTTP. Por ejemplo, si se recoge un documento HTML en cuyo interior están insertadas cuatro

imágenes, el proceso anterior se repite cinco veces, una para el documento HTML y cuatro para las imágenes.

En la actualidad se ha mejorado este procedimiento, permitiendo que una misma conexión se mantenga activa durante un cierto periodo de tiempo, de forma que sea utilizada en sucesivas transacciones. Este mecanismo, denominado ***HTTP Keep Alive***, es empleado por la mayoría de los clientes y servidores modernos. Esta mejora es imprescindible en una Internet saturada, en la que el establecimiento de cada nueva conexión es un proceso lento y costoso.

Veamos el proceso completo con un ejemplo:

1. Desde un cliente se solicita la URL http://www.uazuay.edu.ec/invest/default.html
2. Se abre una conexión TCP/IP con el puerto 80 del sistema www.uazuay.edu.ec.
3. El cliente realiza la solicitud, enviando algo similar a esto:

GET /invest/default.html HTTP/1.0 Operación solicitada+objeto+versión de HTTP

Accept: text/plain Lista de tipos MIME que acepta o entiende

Accept: text/html el cliente

Accept: audio/*

Accept: video/mpeg

.. .. ..
Accept: */* Indica que acepta otros posibles tipos MIME

User-Agent: Mozilla/3.0 (WinNT; I) Información sobre el tipo de cliente

Línea en blanco, indica el final de la petición

4. El servidor responde con la siguiente información:

HTTP/1.0 200 OK Status de la operación; en este caso, correcto

Date: Monday, 7-Oct-96 18:00:00 Fecha de la operación

Server: NCSA 1.4 Tipo y versión del servidor

MIME-version: 1.0 Versión de MIME que maneja

Content-type: text/html Definición MIME del tipo de datos a devolver

Content-length: 254 Longitud de los datos que siguen

Last-modified: 6-Oct-96 12:30:00 Fecha de modificación de los datos

Línea en blanco

<HTML> Comienzo de los datos

<HEAD><TITLE>Recursos de investigación en uazuay</TITLE></HEAD>

<BODY>
.. .. ..

.. .. ..

</HTML>

5. Se cierra la conexión.

**Estructura de los mensajes HTTP**

El diálogo con los servidores HTTP se establece a través de mensajes formados por líneas de texto, cada una de las cuales contiene los diferentes comandos y opciones del protocolo. Sólo existen dos tipos de mensajes, uno para realizar peticiones y otro para devolver la correspondiente respuesta. La estructura general de los dos tipos de mensajes se puede ver en el siguiente esquema:

| **Mensaje de solicitud** | | **Mensaje de respuesta** |
|---|---|---|
| Comando HTTP + parámetros | | Resultado de la solicitud |
| Cabeceras del requerimiento | | Cabeceras de la respuesta |
| (línea en blanco) | | (línea en blanco) |
| Información opcional | | Información opcional |

La primera línea del mensaje de solicitud contiene el comando que se solicita al servidor HTTP, mientras que en la respuesta contiene el resultado de la operación, un código numérico que permite conocer el éxito o fracaso de la operación. Después aparece, para ambos tipos de mensajes, un conjunto de cabeceras (unas obligatorias y otras opcionales), que condicionan y matizan el funcionamiento del protocolo.

La separación entre cada línea del mensaje se realiza con un par CR-LF (retorno de carro más nueva línea). El final de las cabeceras se indica con una línea en blanco, tras la cual se pueden incluir los datos transportados por el protocolo, por ejemplo, el documento HTML que devuelve un servidor o el contenido de un formulario que envía un cliente

Los siguientes apartados describen con más detalle el contenido de cada una de las secciones de los mensajes. El conocimiento y empleo de los mensajes HTTP es necesario en las siguientes situaciones:

Para diseñar módulos CGI, ya que es preciso construir una respuesta similar a la que el servidor HTTP proporciona al cliente.

Para diseñar aplicaciones independientes que soliciten información a un servidor (automatizar la recuperación de documentos o construir robots de búsqueda) se debe construir una cabecera HTTP con la información de la petición al servidor.

**Comandos del protocolo**

Los comandos o verbos de HTTP representan las diferentes operaciones que se pueden solicitar a un servidor HTTP. El formato general de un comando es:

| **Nombre del comando** | **Objeto sobre el que se aplica** | **Versión de HTTP utilizada** |
|---|---|---|

Cada comando actúa sobre un objeto del servidor, normalmente un fichero o aplicación, que se toma de la URL de activación. La última parte de esta URL, que representa la dirección de un objeto dentro de un servidor HTTP, es el parámetro sobre el que se aplica el comando. Se compone de una serie de nombres de directorios y ficheros, además de parámetros opcionales para las aplicaciones CGI (ver *Ejecución de programas en un servidor HTTP* en la página *).

El estándar HTTP/1.0 recoge únicamente tres comandos, que representan las operaciones de recepción y envío de información y chequeo de estado:

**GET** Se utiliza para recoger cualquier tipo de información del servidor. Se utiliza siempre que se pulsa sobre un enlace o se teclea directamente a una URL. Como resultado, el servidor HTTP envía el documento correspondiente a la URL seleccionada, o bien activa un módulo CGI, que generará a su vez la información de retorno.

**HEAD** Solicita información sobre un objeto (fichero): tamaño, tipo, fecha de modificación… Es utilizado por los gestores de caches de páginas o

los servidores proxy, para conocer cuándo es necesario actualizar la copia que se mantiene de un fichero.

**POST** Sirve para enviar información al servidor, por ejemplo los datos contenidos en un formulario. El servidor pasará esta información a un proceso encargado de su tratamiento (generalmente una aplicación CGI). La operación que se realiza con la información proporcionada depende de la URL utilizada. Se utiliza, sobre todo, en los formularios.

Un cliente Web selecciona automáticamente los comandos HTTP necesarios para recoger la información requerida por el usuario. Así, ante la activación de un enlace, siempre se ejecuta una operación GET para recoger el documento correspondiente. El envío del contenido de un formulario utiliza GET o POST, en función del atributo de <FORM METHOD="..">. Además, si el cliente Web tiene un caché de páginas recientemente visitadas, puede utilizar HEAD para comprobar la última fecha de modificación de un fichero, antes de traer una nueva copia del mismo.

Posteriormente se han definido algunos comandos adicionales, que sólo están disponibles en determinadas versiones de servidores HTTP, con motivos eminentemente experimentales. La última versión de HTTP, denominada 1.1, recoge estas y otras novedades, que se pueden utilizar, por ejemplo, para editar las páginas de un servidor Web trabajando en remoto.

**PUT** Actualiza información sobre un objeto del servidor. Es similar a POST, pero en este caso, la información enviada al servidor debe ser almacenada en la URL que acompaña al comando. Así se puede actualizar el contenido de un documento.

**DELETE** Elimina el documento especificado del servidor.

**LINK** Crea una relación entre documentos.

**UNLINK** Elimina una relación existente entre documentos del servidor.

**Las cabeceras**

Son un conjunto de variables que se incluyen en los mensajes HTTP, para modificar su comportamiento o incluir información de interés. En función de su nombre, pueden aparecer en los requerimientos de un cliente, en las respuestas del servidor o en ambos tipos de mensajes. El formato general de una cabecera es:

**Nombre de la variable : Cadena ASCII con su valor**

Los nombres de variables se pueden escribir con cualquier combinación de mayúsculas y minúsculas. Además, se debe incluir un espacio en blanco entre el signo **:** y su valor. En caso de que el valor de una variable ocupe varias líneas, éstas deberán comenzar, al menos, con un espacio en blanco o un tabulador.

**Cabeceras comunes para peticiones y respuestas**

Content-Type: descripción MIME de la información contenida en este mensaje. Es la referencia que utilizan las aplicaciones Web para dar el correcto tratamiento a los datos que reciben.

Content-Length: longitud en bytes de los datos enviados, expresado en base decimal.

Content-Encoding: formato de codificación de los datos enviados en este mensaje. Sirve, por ejemplo, para enviar datos comprimidos (x-gzip o x-compress) o encriptados.

Date: fecha local de la operación. Las fechas deben incluir la zona horaria en que reside el sistema que genera la operación. Por ejemplo: Sunday, 12-Dec-96 12:21:22 GMT+01. No existe un formato único en las fechas; incluso es posible encontrar casos en los que no se dispone de la zona horaria correspondiente, con los problemas de sincronización que esto produce. Los formatos de fecha a emplear están recogidos en los RFC 1036 y 1123.

Pragma: permite incluir información variada relacionada con el protocolo HTTP en el requerimiento o respuesta que se está realizando. Por ejemplo, un cliente envía un Pragma: no-cache para informar de que desea una copia nueva del recurso especificado.

**Cabeceras sólo para peticiones del cliente**

Accept: campo opcional que contiene una lista de tipos MIME aceptados por el cliente. Se pueden utilizar * para indicar rangos de tipos de datos; tipo/* indica todos los subtipos de un determinado medio, mientras que */* representa a cualquier tipo de dato disponible.

Authorization: clave de acceso que envía un cliente para acceder a un recurso de uso protegido o limitado. La información incluye el formato de autorización empleado, seguido de la clave de acceso propiamente dicha. La explicación se incluye más adelante.

From: campo opcional que contiene la dirección de correo electrónico del usuario del cliente Web que realiza el acceso.

If-Modified-Since: permite realizar operaciones GET condicionales, en función de si la fecha de modificación del objeto requerido es anterior o posterior a la fecha proporcionada. Puede ser utilizada por los sistemas de almacenamiento temporal de páginas. Es equivalente a realizar un HEAD seguido de un GET normal.

Referer: contiene la URL del documento desde donde se ha activado este enlace. De esta forma, un servidor puede informar al creador de ese documento de cambios o actualizaciones en los enlaces que contiene. No todos los clientes lo envían.

User-agent: cadena que identifica el tipo y versión del cliente que realiza la petición. Por ejemplo, los *browsers* de Netscape envían cadenas del tipo User-Agent: Mozilla/3.0 (WinNT; I)

**Cabeceras sólo para respuestas del servidor http**

Allow: informa de los comandos HTTP opcionales que se pueden aplicar sobre el objeto al que se refiere esta respuesta. Por ejemplo, Allow: GET, POST.

Last-modified: fecha local de modificación del objeto devuelto. Se puede corresponder con la fecha de modificación de un fichero en disco, o, para información generada dinámicamente desde una base de datos, con la fecha de modificación del registro de datos correspondiente.

Location: informa sobre la dirección exacta del recurso al que se ha accedido. Cuando el servidor proporciona un código de respuesta de la serie 3xx, este parámetro contiene la URL necesaria para accesos posteriores a este recurso.

Server: cadena que identifica el tipo y versión del servidor HTTP. Por ejemplo, Server: NCSA 1.4.

WWW-Autenticate: cuando se accede a un recurso protegido o de acceso restringido, el servidor devuelve un código de estado 401, y utiliza este campo para informar de los modelos de autentificación válidos para acceder a este recurso.

**Códigos de estado del servidor**

Ante cada transacción con un servidor HTTP, éste devuelve un código numérico que informa sobre el resultado de la operación, como primera línea del mensaje de respuesta. Estos códigos aparecen en algunos casos en la pantalla del cliente, cuando se produce un error. El formato de la línea de estado es:

| Versión de protocolo HTTP utilizada | Código numérico de estado (tres dígitos) | Descripción del código numérico |
|---|---|---|

Dependiendo del servidor, es posible que se proporcione un mensaje de error más elaborado, en forma de documento HTML, en el que se explican las causas del error y su posible solución.

Existen cinco categorías de mensajes de estado, organizadas por el primer dígito del código numérico de la respuesta:

1xx : mensajes informativos. Por ahora (en HTTP/1.0) no se utilizan, y están reservados para un futuro uso.

2xx : mensajes asociados con operaciones realizadas correctamente.

3xx : mensajes de redirección, que informan de operaciones complementarias que se deben realizar para finalizar la operación.

4xx : errores del cliente; el requerimiento contiene algún error, o no puede ser realizado.

5xx : errores del servidor, que no ha podido llevar a cabo una solicitud.

Los más comunes se recogen en la siguiente tabla:

| ***Código*** | ***Comentario*** | ***Descripción*** |
|---|---|---|
| ***200*** | OK | Operación realizada satisfactoriamente. |
| ***201*** | Created | La operación ha sido realizada correctamente, y como resultado se ha creado un nuevo objeto, cuya URL de acceso se proporciona en el cuerpo de la respuesta. Este nuevo objeto ya está disponible. Puede ser utilizado en sistemas de edición de documentos. |
| ***202*** | Accepted | La operación ha sido realizada correctamente, y como resultado se ha creado un nuevo objeto, cuya URL de acceso se proporciona en el cuerpo de la respuesta. El nuevo objeto no está disponible por el momento. En el cuerpo de la |

| | | |
|---|---|---|
| | | respuesta se debe informar sobre la disponibilidad de la información. |
| ***204*** | No Content | La operación ha sido aceptada, pero no ha producido ningún resultado de interés. El cliente no deberá modificar el documento que está mostrando en este momento. |
| ***301*** | Moved Permanently | El objeto al que se accede ha sido movido a otro lugar de forma permanente. El servidor proporciona, además, la nueva URL en la variable Location de la respuesta. Algunos browsers acceden automáticamente a la nueva URL. En caso de tener capacidad, el cliente puede actualizar la URL incorrecta, por ejemplo, en la agenda de *bookmarks*. |
| ***302*** | Moved Temporarily | El objeto al que se accede ha sido movido a otro lugar de forma temporal. El servidor proporciona, además, la nueva URL en la variable Location de la respuesta. Algunos browsers acceden automáticamente a la nueva URL. El cliente no debe modificar ninguna de las referencias a la URL errónea. |
| ***304*** | Not Modified | Cuando se hace un GET condicional, y el documento no ha sido modificado, se devuelve este código de estado. |
| ***400*** | Bad Request | La petición tiene un error de sintaxis y no es entendida por el servidor. |
| ***401*** | Unauthorized | La petición requiere una autorización especial, que normalmente consiste en un nombre y clave que el servidor verificará. El campo WWW-Autenticate informa de los protocolos de autentificación aceptados para este recurso. |
| ***403*** | Forbidden | Está prohibido el acceso a este |

| | | |
|---|---|---|
| | | recurso. No es posible utilizar una clave para modificar la protección. |
| ***404*** | Not Found | La URL solicitada no existe. |
| ***500*** | Internal Server Error | El servidor ha tenido un error interno, y no puede continuar con el procesamiento. |
| ***501*** | Not Implemented | El servidor no tiene capacidad, por su diseño interno, para llevar a cabo el requerimiento del cliente. |
| ***502*** | Bad Gateway | El servidor, que está actuando como proxy o pasarela, ha encontrado un error al acceder al recurso que había solicitado el cliente. |
| ***503*** | Service Unavailable | El servidor está actualmente deshabilitado, y no es capaz de atender el requerimiento. |

**Cachés de páginas y servidores proxy**

Muchos clientes Web utilizan un sistema para reducir el número de accesos y transferencias de información a través de Internet, y así agilizar la presentación de documentos previamente visitados. Para ello, almacenan en el disco del cliente una copia de las últimas páginas a las que se ha accedido. Este mecanismo, denominado **"caché de páginas"**, mantiene la fecha de acceso a un documento y comprueba, a través de un comando HEAD, la fecha actual de modificación del mismo. En caso de que se detecte un cambio o actualización, el cliente accederá, ahora a través de un GET, a recoger la nueva versión del fichero. En caso contrario, se procederá a utilizar la copia local.

Las versiones de Netscape Navigator anteriores a la 2.02 tienen fallos en su gestión de caché, que hace que se muestre una copia antigua de un documento almacenada en un caché, a pesar de que el documento original haya cambiado y se solicite la nueva versión a través de un *RELOAD*.

Un sistema parecido, pero con más funciones es el denominado **"servidor *proxy*"**. Este tipo de servidor, una mezcla entre servidor HTTP y cliente Web, realiza las funciones de cache de páginas para un gran número de clientes. Todos los clientes conectados a un *proxy* dejan que éste sea el encargado de recoger las URLs solicitadas. De esta

forma, en caso de que varios clientes accedan a la misma página, el servidor *proxy* la podrá proporcionar con un único acceso a la información original.

La principal ventaja de ambos sistemas es la drástica reducción de conexiones a Internet necesarias, en caso de que los clientes accedan a un conjunto similar de páginas, como suele ocurrir con mucha frecuencia. Además, determinadas organizaciones limitan, por motivos de seguridad, los accesos desde su organización al exterior y viceversa. Para ello, se dispone de sistemas denominados "cortafuegos" *(firewalls)*, que son los únicos habilitados para conectarse con el exterior. En este caso, el uso de un servidor *proxy* se vuelve indispensable.

En determinadas situaciones, el almacenamiento de páginas en un caché o en un *proxy* puede hacer que se mantengan copias no actualizadas de la información, como por ejemplo en el caso de trabajar con documentos generados dinámicamente. Para estas situaciones, los servidores HTTP pueden informar a los clientes de la expiración del documento, o de la imposibilidad de ser almacenado en un caché, utilizando la variable Expires en la respuesta del servidor.

***Magic Cookies***

Las ‘galletas mágicas’ son una de las incorporaciones más recientes al protocolo HTTP, y son parte del nuevo estándar HTTP/1.1. Las *cookies* son pequeños ficheros de texto que se intercambian los clientes y servidores HTPP, para solucionar una de las principales deficiencias del protocolo: la falta de información de estado entre dos transacciones. Fueron introducidas por Netscape, y ahora han sido estandarizadas en el RFC 2109.

Cada intercambio de información con un servidor es completamente independiente del resto, por lo que las aplicaciones CGI que necesitan recordar operaciones previas de un usuario (por ejemplo, los supermercados electrónicos) pueden optar por dos soluciones, que complican bastante su diseño: almacenar temporalmente en el sistema del servidor un registro de las operaciones realizadas, con una determinada política para eliminarlas cuando no son necesarias, o bien incluir esta información en el código HTML devuelto al cliente, como campos HIDDEN de un formulario.

Las *cookies* son una solución más flexible a este problema. La primera vez que un usuario accede a un determinado documento de un servidor, éste proporciona una *cookie* que contiene datos que relacionarán posteriores operaciones. El cliente almacena la *cookie* en su sistema para usarla después. En los futuros accesos a este servidor, el *browser* podrá proporcionar la *cookie* original, que servirá de nexo entre este acceso y los anteriores. Todo este proceso se realiza automáticamente,

sin intervención del usuario. El siguiente ejemplo muestra una *cookie* generada por la revista *Rolling Stone*.

EGSOFT_ID
191.46.211.13-655193640.29148285
www.rollingstone.com/
0
2867435528
30124157
4206779936
291478284

La información dentro de una *cookie* se organiza a partir de líneas de texto. El campo más interesante es la tercera línea. El cliente Web la compara cada URL a la que accede; si se produce una concordancia entre ambas, se enviará la *cookie* correspondiente. Es decir, una *cookie* asociada a www.rollingstone.com/shop sólo se enviará con accesos por debajo de la sección /shop, mientras que la *cookie* del ejemplo servirá para todo el servidor *Rolling Stone*. Las *cookies* pueden tener asociada una fecha de expiración, a partir de la cual el cliente Web tratará de obtener una nueva versión.

¿Para qué se pueden utilizar? Su aplicación más inmediata son los sistemas de compra electrónica. Estos supermercados virtuales necesitan relacionar el contenido de un pedido con el cliente que lo ha solicitado. Sin *cookies*, la solución más sencilla es incluir dentro de los documentos HTML el contenido de las operaciones o pedidos previos, a través de variables ocultas. Con ellas es posible relacionar los sucesivos accesos al sistema con su origen y simplificar la gestión de la aplicación Web.

Otro uso muy interesante son los sistemas personalizados de recepción de información, en los que es posible construir una página a medida, con información procedente de fuentes muy diversas (ver por ejemplo my.yahoo.com/ o www.snap.com). En el primer acceso al sistema, se procede al registro; a partir de él, se genera una *cookie* que recibe el cliente. En accesos sucesivos, el cliente enviará la *cookie*, y el servidor podrá generar una página personalizada con las preferencias del usuario.

Por último, algunas compañías emplean las *cookies* para realizar un seguimiento de los accesos a sus servidores WWW, identificando las páginas más visitadas, la manera en que se pasa de una a otra sección, etc.

Para utilizar las *cookies*, es necesario que los clientes estén preparados para recogerlas y almacenarlas. Netscape Navigator e Internet Explorer disponen ya de esta capacidad. Sin embargo, los servidores no

necesitan capacidades especiales, ya que son las aplicaciones CGI las encargadas de su gestión.

Por defecto, los clientes Web reciben y envían *cookies* sin solicitar confirmación al usuario, pero es posible recibir avisos de la recepción de *cookies*, para rechazarlas en caso de no ser de nuestro agrado. ¿Por qué rechazar las *cookies*?. Bueno, se pueden utilizar para seguir una pista del tipo de información que buscamos en Internet, y ofrecer información comercial en función de ello. Sin embargo, algunos servicios Web como tiendas on-line dependen de ellas para operar correctamente.

Las *cookies* tienen un tiempo de vida, que hace que sean descartadas pasados un cierto tiempo de actividad. Algunas expiran al salir del cliente Web, pero otras *cookies* tienen una duración mayor, por lo que el cliente Web las almacena en un fichero. Netscape utiliza el fichero cookies.txt de su directorio de instalación, mientras que Microsoft almacena cada uno en un fichero independiente del caché de páginas.

Las *cookies* son imprescindibles para operar con determinados servidores de acceso restringido, en los cuales es preciso seguir un proceso de registro, y posiblemente abonar algún tipo de tasa. Cuando se pierde una de esas *cookies*, por una reinstalación, fallo del ordenador o limpieza del caché de páginas, en algunos casos puede ser posible regenerarla siguiendo un nuevo proceso de registro en el servidor que la proporcionó, de forma que se relacione algún tipo de información personal o clave de acceso proporcionado en la primera visita al servidor.

**Uso de las *cookies***

Una *cookie* es simplemente una serie de líneas de texto, con pares variable/valor. Existe un conjunto predefinido de nombres de variable, necesario para el correcto funcionamiento de las *cookies*, pero se pueden crear nuevas variables para cubrir las necesidades de una aplicación concreta. Las principales variables predefinidas son:

Domain= conjunto de direcciones Internet para el que es válida la *cookie*. Se puede dar una dirección única (www.mitienda.es) o un rango (.netscape.com).

Path= fija el subconjunto de URLs para las que sirve esta *cookie*.

Version= Permite seleccionar entre diferentes versiones del modelo de *cookies*.

Expires= Fecha de expiración de la información. Si no se incluye, los datos son descartados al finalizar la sesión con el cliente Web. En caso

contrario se almacenará en el disco del cliente. El RFC 2109 ha cambiado esta variable por Max-Age, una duración relativa en segundos.

Un servidor HTTP envía los diferentes campos de una *cookie* con la nueva cabecera HTTP Set-Cookie:

Set-Cookie: Domain=www.unican.es; Path=/; Nombre=Luis; Expires Fri, 15-Jul-97 12:00:00 GMT

Cuando se accede a una URL que verifica el par dominio/path registrado, el cliente enviará automáticamente la información de los diferentes campos de la cookie con la nueva cabecera HTTP Cookie:

Cookie: Domain=www.unican.es; Path=/; Nombre=Luis

### *3.4 Real Time Protocol (RTP).*

La cantidad de servicios ofrecidos a través de Internet crece constantemente. Además, teniendo en cuenta su expansión, resulta especialmente atractiva la idea de utilizarla como transporte de información multimedia. Debido a que Internet es una red de conmutación de paquetes, altamente heterogénea, no es naturalmente adecuada para el tráfico en tiempo real. Los problemas principales que se encuentran son:

a) Necesidad de ancho de banda: la información multimedia es extremadamente “pesada” (en comparación con información textual), por lo que se requiere mayor disponibilidad de canales con gran ancho de banda.

b) Distribución: La entrega de información multimedia debiera realizarse mediante técnicas de multicast para llegar solamente a los receptores, sin duplicar los mensajes, reduciendo así el tráfico sobre la red.

c) Disponibilidad de recursos: la red debiera tener disponibilidad de recursos para asegurar ancho de banda y tiempos de retardo “aceptables” para aplicaciones multimedia. Internet es una red de datagramas que hace su “mejor esfuerzo” para la entrega, donde los paquetes son ruteados independientemente por diferentes caminos y pueden sufrir distintos inconvenientes en cada uno. Dada esta característica, no se puede asegurar que los datos alcanzarán el destino.

   Los protocolos para el transporte de multimedia sobre Internet deben contemplar estas características y asegurar que un video o

un flujo de audio sean reproducidos constantemente, con la frecuencia y sincronización adecuada.

RTP es un protocolo basado en IP que provee soporte para el transporte de datos en tiempo real como flujos de audio y video. Entre los servicios que provee se encuentran reconstrucción de tiempos, detección de pérdidas, seguridad e identificación de contenido.

Está diseñado para trabajar en conjunto con el protocolo auxiliar RTCP para obtener información sobre calidad de la transmisión y participantes de la sesión.

RTP provee transporte entre sistemas finales para datos en tiempo real sobre una red de datagramas. Generalmente opera sobre UDP por varios motivos. Aprovecha las funciones de control de error y de multiplexación, y además – por ser un protocolo de transporte no orientado a la conexión – no ofrece confiabilidad, por lo que no generará retransmisiones que puedan congestionar la red (para datos en tiempo real, la confiabilidad no es tan importante como la entrega rápida).

Las características principales de RTP son:

a) Timestamping, el emisor setea el timestamp de acuerdo al instante en que el primer octeto del paquete fue muestreado. El timestamp aumenta mientras se completa un paquete. Por otro lado, el receptor usa estas marcas para reconstruir el timing original para reproducir la secuencia a la frecuencia correcta.

b) Secuenciación: Debido a la necesidad de entregar los paquetes en orden (UDP no provee esta característica) RTP incorpora un número de secuencia que – además – sirve para la detección de paquetes perdidos.

c) Source identification: permite conocer al receptor de la información de dónde provienen los datos.

d) Payload type identificator: especifica el formato de datos de la carga de RTP (definidos en el RFC 1890, por ejemplo MPEG1, JPEG, PCM), referente al formato de codificación, de manera que la aplicación receptora pueda saber como reproducirlo.

En la practica RTP se implementa en la aplicación. Para establecer una sesión RTP, la aplicación define una dirección de red y un par de puertos para RTP y RTCP. En una sesión multimedia, cada medio es transportado por sesiones RTP separadas, con paquetes de RTCP propios de reporte de calidad de recepción.

## GLOSARIO

**ACK**: Acknowledgment. Reconocimiento. Señal de respuesta.

**ANSI**: American National Standard Institute. Instituto Nacional Americano de Estandar.

**API**: Aplication Program Interface. Es el conjunto de rutinas del sistema que se pueden usar en un programa para la gestión de entrada/salida, gestión de ficheros etc.

**ASCII**: American Standard Code for Information Interchange. Estandar Americano para Intercambio de Informacion. La tabla básica de caracteres ASCII esta compuesta por 128 caracteres incluyendo símbolos y caracteres de control. Existe una version extendida de 256

**ASN**: Autonomus System Number. Número de sistema autónomo. Grupo de routers y redes controlados por una única autoridad administrativa.

**ATM**: Asyncronous Transmision Mode. Modo de Transmisión Asíncrona.

**BBS**: Bulletin Board System. Tablero de Anuncios Electrónico. Servidor de comunicaciones que proporciona a los usuarios servicios variados como e-mail o transferencia de ficheros. Originalmente funcionaban a través de líneas telefónicas.

**Bandwith**: Ancho de Banda. Capacidad de un medio de transmisión.

**BIT**: Binary Digit. Digito Binario. Unidad mínima de información, puede tener dos estados "0" o "1".

**Backbone**: Estructura de trasmisión de datos de una red o conjunto de ellas en Internet. Literalmente: "esqueleto"

**Baudio**: Unidad de medida. Numero de bits de información por segundo.

**Browser**: Término aplicado normalmente a los programas que permiten acceder al servicio WWW.

**CGI**: Common Gateway Interface. Interface de Acceso Común. Programas usado s para hacer llamadas a rutinas o controlar otros programas o bases de datos desde una página Web. También pueden generar directamente HTML.

**CIX**: Comercial Internet Exchange. Intercambio Comercial Internet.

**CSLIP**: Compressed Serial Line Protocol. Protocolo de Linea Serie Comprimido. Es una versión mejorada del SLIP desarrollada por Van Jacobson. Principalmente se trata de en lugar de enviar las cabeceras completas de los paquetes enviar solo las diferencias.

**Caudal**: Cantidad de ocupación en un ancho de banda. Ejp. En una línea de 1Mbps. Puede haber un caudal de 256Kbps. con lo que los 768Kbps. restantes de el ancho de banda permanecen desocupados.

**DATAGRAM**: Datagráma. Usualmente se refiere a la estructura interna de un paquete de datos.

**DCD**: Data Carrier Detected. Detectada Portadora de Datos.

**DDE**: Dynamic Data Exchange. Intercambio Dinámico de Datos. Conjunto de especificaciones de microsoft para el intercambio de datos y control de flujo entre aplicaciones.

**DES**: Data Encryption Standard.Algoritmo de Encriptacion de Estandar. Algoritmo desarrollado por IBM, utiliza bloques de datos de 64 bits y una clave de 56 bits. Es utilizado por el gobierno americano.

**DNS**: Domain Name System. Sistema de nombres de Dominio. Base de datos distribuida que gestiona la conversión de direcciones de Internet expresadas en lenguaje natural a una dirección numérica IP. Ejemplo: 121.120.10.1

**DSP**: Digital Signal Procesor. Procesador Digital de Señal.

**DSR**: Data Set Ready (MODEM).

**DTR**: Data Transfer Ready. Preparado para Transmitir Datos (MODEM).

**Domain**: Dominio. Sistema de denominación de Hosts en Internet. Los dominios van separados por un punto y jerárquicamente están organizados de derecha a izquierda. ejp: ISLOGICA.COM

**ETSI**: European Telecommunication Standars Institute. Instituto Europeo de estándares en Telecomunicaciones.

**E-mail**: Electronic Mail. Correo Electrónico. Sistema de mensajeria informática similar en muchos aspectos al correo ordinario pero muchísimo mas rápido.

**FAQ**: Frecuent Asked Question. Preguntas Formuladas Frecuentemente. Las FAQs de un sistema son archivos con las preguntas y respuestas mas habituales sobre el mismo.

**FTP**: File Transfer Protocol. Protocolo de Transferencia de Ficheros. Uno de los protocolos de trasferencia de ficheros mas usado en Internet.

**Firewall**: Literalmente " Muro de Fuego". Se trata de cualquier programa que protege a una red de otra red. El firewall da acceso a una maquina en una red local a Internet pero Internet no ve mas allá del firewall.

**Frame**: Estructura. También trama de datos.

**Frame Relay**: Protocolo de enlace mediante circuito virtual permanente muy usado para dar conexión directa a Internet.

**GIF**: Graphics Interchange Format. Formato Grafico de Intercambio.

**GSM**: Global System Mobile comunications. Sistema Global de Comunicaciones Moviles. Sistema digital de telecomunicaciones pricipalmente usado para telefonía móvil. Existe compatibilidad entre redes por tanto un teléfono GSM puede funcionar en todo el mundo.

**GUI**: Graphic User Interface. Interface Gráfico de Usuario.

**Gateway**: Prueta de Acceso. Dispositivo que permite conectar entre si dos redes normalmente de distinto protocolo o un Host a una red. En Español: Pasarela.

**HDLC**: High-Level Data Link Control. Control de Enlace de Datos de Alto Nivel.

**HPFS**: High Performance File System. Sistema de Archivos de Alto Rendimiento. Sistema que utiliza el OS/2 opcionalmente para organizar el disco duro en lugar del habitual de FAT.

**HTML**: HyperText Markup Language. Lenguaje de Marcas de Hypertexto. Lenguaje para elaborar paginas Web actualmente se encuentra en su version 3. Fue desarrollado en el CERN.

**HTTP**: HyperText Transfer Protocol. Protocolo de transferencia de Hypertexto. Protocolo usado en WWW.

**Hacker**: Experto en informática capaz de de entrar en sistemas cuyo acceso es restringido. No necesariamente con malas intenciones.

**Host**: Ordenador conectado a Internet. Ordenador en general. Literalmente anfitrián.

**IANA**: Internet Assigned Number Authority. Autoridad de Asignación de Números en Internet. Se trata de la entidad que gestiona la asignación de direcciones IP en Internet.

**IETF**: Internet Engineering Task Force. Grupo de Tareas de Ingeniería de Internet. Asociación de técnicos que organizan las tareas de ingeniería principalmente de telecomunicaciones en Internet. Por ejemplo: mejorar protocolos o declarar obsoletos otros.

**INTERNIC**: Entidad administrativa de Internet que se encarga de gestionar los nombres de dominio de EEUU.

**INTRANET**: Se llaman así a las redes tipo Internet pero que son de uso interno, por ejemplo, la red corporativa de una empresa que utilizara protocolo TCP/IP y servicios similares como www.

**IP**: Internet Protocol. Protocolo de Internet. Bajo este se agrupan los protocolos de internet. También se refiere a las direcciones de red Internet.

**IPX**: Internet Packet Exchange. Intercambio de Paquetes entre Redes. Inicialmente protocolo de Novell para el intercambio de información entre aplicaciones en una red Netware.

**IRC**: Internet Relay Chat. Canal de Chat de Internet. Sistema para transmisión de texto multiusuario a través de un servidor IRC. Usado normalmente para conversar on-line también sirve para transmitir ficheros.

**ISDN**: Integrated Services Digital Network. Red Digital de Servicios Integrados. En español RDSI.

**ISO**: International Standard Organization. Organizacion Internacional de Standard.

**ISS**: Internet Security Scanner. Rastreador de Seguridad de Internet. Programa que busca puntos vulnerables de la red con relación a la seguridad.

**JAVA**: Lenguaje de programación orientado a objeto parecido al C++. Usado en WWW para la telecarga y telejecucion de programas en el ordenador cliente. Desarrollado por Sunmicrosystems.

**JPEG**: Join Photograph Expert Group. Unión de Grupo de Expertos Fotográfico. Formato grafico con compresión con perdidas que consigue elevados ratios de compresión.

**LAN**: Local Area Network. Red de Area Local. Red de ordenadores reducidas dimensiones. Por ejemplo una red distribuida en una planta de un edificio.

**LCP**: Link Control Protocol. Protocolo de Control de Enlace.

**Link**: Enlace. Unión.

**Linux**: Version Shareware del conocido systema operativo Unix. Es un sistema multitarea multiusuario de 32 bits para pc.

**LU**: Logic Unit. Unidad Lógica.

**Lock**: Cerrado. Bloqueado.

**MAN**: Metropolitan Area Network. Red de Area Metropolitana.

**MNP**: Microcom Networking Protocol. Protocolo de Redes de Microcom. Protocolo de corrección de errores desarrollado por Microcom muy usado en comunicaciones con modem. Existen varios niveles MNP2(asíncrono), MNP3(síncrono) y MNP4(síncrono).

**MODEM**:Modulator/Demodulator. Modulador/Demodulador. Dispositivo que adapta las señales digitales para su transmisión a través de una línea anal lógica. Normalmente telefónica.

**MPEG**: Motion Pictures Expert Group. Grupo de Expertos en Imagen en Movimiento. Formato gráfico de almacenamiento de video. Utiliza como el JPEG compresión con perdidas alcanzando ratios muy altos.

**MRU**: Maximum Receive Unit. Unidad Máxima de Recepción. En algunos protocolos de Internet se refiere al máximo tamaño del paquete de datos.

**MS-DOS**: Microsoft Disk Operating System. Sistema Operativo en Disco de Microsoft. Sistema operativo muy extendido en PC del tipo de línea de comandos.

**MTU**: Maximum Transmission Unit. Unidad Máxima de Transmision. Tamaño máximo de paquete en protocolos IP como el SLIP.

**NACR**: Network Announcement Request. Petición de participación en la Red. Es la petición de alta en Internet para una subred o dominio.

**NAP**: Network Access Point. Punto de Acceso a la Red. Normalmente se refiere a los tres puntos principales por los que se accede a la red Internet en U.S.

**NCP**: Network Control Protocol. Protocolo de Control de Red. Es un protocolo del Network Layer.

**NET**: Red.

**NETBIOS**: Network BIOS. Network Basic Input/Output System. Bios de una red, es decir, Sistema Básico de Entrada/Salida de red.

**Navegador**: Aplicado normalmente a programas usados para conectarse al servicio WWW.

**Nodo**: Por definición punto donde convergen mas de dos líneas. A veces se refiere a una única máquina en Internet. Normalmente se refiere a un punto de confluencia en una red.

**OEM**: Original Equipament Manufactured. Manufactura de Equipo Original. Empresa que compra un producto a un fabricante y lo integra en un producto propio. Todos los fabricantes por ejemplo, que incluyen un Pentium en su equipo actúan como OEM.

**OSI**: Open Systems Interconnection. Interconexión de Sistemas Abiertos. Modelo de referencia de interconexión de sistemas abiertos propuesto por la ISO. Divide las tareas de la red en siete niveles.

**PAP**: Password Authentication Protocol. Protocolo de Autentificación por P assword. Protocolo que permite al sistema verificar la identidad del otro punto de la conexión mediante password.

**PDA**: Personal Digital Assistan. Asistente Personal Digital. Programa que se encarga de atender a un usuario concreto en tareas como búsquedas de información o selecciones atendiendo a criterios personales del mismo. Suele tener tecnología de IA (Inteligencia Artificial).

**PEER**: En una conexión punto a punto se refiere a cada uno de los extremos.

**PEM**: Private Enhanced Mail. Correo Privado Mejorado. Sistema de correo con encriptación.

**POP**: Post Office Protocol. Protocolo de Oficina de Correos. Protocolo usado por ordenadores personales para manejar el correo sobre todo en recepción.

**PPP**: Point to Point Protocol. Protocolo Punto a Punto. Protocolo Internet para establecer enlace entre dos puntos.

**PVC**: Permanent Virtual Circuit. Circuito Virtual Permanente. Linea punto a punto virtual establecida normalmente mediante conmutaciones de carácter permanente. Es decir a través de un circuito establecido.

**Packet Driver**: Pequeño programa situado entre la tarjeta de red y el programa de TCP de manera que proporciona un internase estándar que los programas pueden usar como si de undriver se tratase.

**Proveedor de Acceso**: Centro servidor que da acceso lógico a internet, es decir sirve de pasarela (Gateway) entre el usuario final e Internet.

**Proveedor de Conexión**: Entidad que proporciona y gestiona enlace físico a Internet. Por ejemplo ETAPAONLINE, SATNET.

**RARP**: Reverse Address Resolution Protocol. Protocolo de Resolución de Dirección de Retorno. Protocolo de bajo nivel para la asignación de direcciones IP a maquinas simples desde un servidor en una red física.

**RDSI**: Red Digital de Servicios Integrados. Red de telefónica con anchos de banda desde 64Kbps.

**ROOT**: Raíz. En sistemas de ficheros se refiere al directorio raíz. En Unix se refiere al usuario principal.

**RSA**: Rivest, Shamir, Adelman [public key encryption algorithm]. Algoritmo de encriptación de clave publica desarrollado por Rivest, Shamir y Adelman.

**RTC**: Red telefónica Conmutada. Red telefónica para la transmisión de voz.

**RTP**: Real Time Protocol. Protocolo de Tiempo Real. Protocolo utilizado para la trasmisión de información en tiempo real como por ejemplo audio y video en una video-conferencia.

**RWIN**: Receive Windows. Ventana de recepción. Parámetro de TCP que determina la cantidad máxima de datos que puede recibir el ordenador que actúa como receptor.

**RX**: Abreviatura de Recepción o Recibiendo.

**Router**: Dispositivo conectado a dos o mas redes que se encarga únicamente de tareas de comunicaciones.

**SDLC**: Syncronous Data Link Controller. Controlador de Enlace de Datos Sincrono. También se trata de un protocolo para enlace sincrono a través de línea telefónica.

**SEPP**: Secure Electronic Payment Protocol. Protocolo de Pago Electrónico Seguro. Sistema de pago a través de Internet desarrollado por Netscape y Mastercard.

**SMPT**: Simple Mail Transfer Protocol. Protocolo de Transferencia Simple de Correo. Es el protocolo usado para trasportar el correo a través de Internet.

**SNA**: System Network Arquitecture. Arquitectura de Sistemas de Redes. Arquitectura de red exclusiva de IBM. Principalmente orientada a Mainframes.

**STT**: Secure Transaction Technology. tecnología de Transacción Segura. Sistema desarrollado por Microsoft y Visa para el comercio electrónico en Internet.

**Sniffer**: Literalmente "Husmeador". Pequeño programa que busca una cadena numérica o de caracteres en los paquetes que atraviesan un nodo con objeto de conseguir alguna información. Normalmente su uso es ilegal.

**S-HTTP**: Secure HTTP. HTTP seguro. Protocolo HTTP mejorado con funciones de seguridad con clave simétrica.

**TCP**: Transmission Control Protocol. Protocolo de control de Transmision. Uno de los protocolos mas usados en Internet. Es un protocolo del Transport Layer.

**TELNET**: Tele Network. Tele Red. conexión a un Host en la que el ordenador cliente emula un terminal de manera que se configura como terminal virtual del ordenador servidor.

**TTD**: telefónica Trasmisión de Datos. División de Telefónica para la transmisión de datos.

**TX**: Abreviatura de Transmisión o Transmitiendo.

**UDP**: User Datagram Protocol. Protocolo de Datagrama de Usuario. Protocolo abierto en el que el usuario (programador) define su propio tipo de paquete.

**URL**: Uniform Resource Locator. Localizador Uniforme de Recursos. Denominación que no solo representa una dirección de Internet sino que apunta aun recurso concreto dentro de esa dirección.

**UUCP**: Unix to Unix Communication Protocol. Protocolo de Comunicaciones de Unix a Unix. Uno de los protocolos que utilizan los sistemas Unix para comunicarse entre si.

**VR**: Virtual Reality. Realidad Virtual.

**VRML**: Virtual Reality Modeling Language. Lenguaje para Modelado de Realidad Virtual.Lenguage para crear mundos virtuales en la Web.

**WINDOWS**: Pseudo sistema operativo. Mas bien se trata de un entorno gráfico con algunas capacidades multitarea. La version actual WINDOWS 95 funciona parcialmente a 32 bits.

**WWW, WEB o W3**: World Wide Web. Telaraña mundial, para muchos la WWW es Internet, para otros es solo una parte de esta. Podríamos decir estrictamente que la WEB es la parte de Internet a la que accedemos a través del protocolo HTTP y en consecuencia gracias a Browsers normalmente gráficos como Netscape.

**BIBLIOGRAFIA**

**Redes de Computadores. Protocolos, normas e interfaces.**
2ª Edición
Uyless Black
Ed. Alfaomega

**JAVA Network Programing.**
Hughes, Manning, 1997. E. Rusty Harold, O´Reilly, 1997

**Interworking with TCP/IP**
4 Edición
Autor: D. E. Comer, Prentice Hall, 2000

**Computer Networks**
Andrew S. Tanennbaum
4 Edición.

## INDICE