Protocolos y arquitecturas de aplicaciones en internet

Aplicaciones Web / Sistemas Web

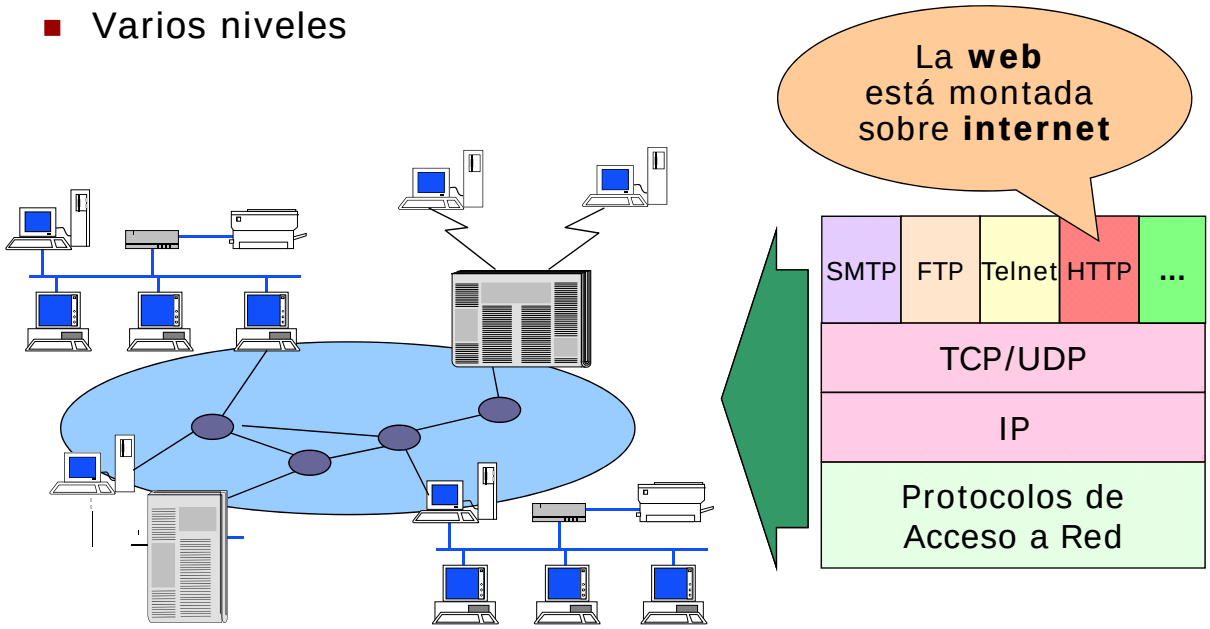
Juan Pavón Mestras
Dep. Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense Madrid

De internet a la Web

- 1969: ARPAnet (*Advanced Research Project Agency)*
  - Comienza a funcionar públicamente en 1971
- 1972: Correo electrónico
- 1974: TCP/IP (RFC 675)
  - Estándar publicado en 1983 (RFC 793)
- 1984: Se define el Sistema de Nombres de Dominio (DNS)
- 1986: Internet Engineering Task Force (IETF)
- 1989: Archie, 1991: Gopher
- 1989: Tim Berners-Lee inventa la *World Wide Web,* WWW
  - Propuesta inicial: http://www.w3.org/History/1989/proposal.html
  - En 1989 escribe el primer servidor, *httpd*, y en 1990 el primer cliente (navegador), *WorldWideWeb*
- 1990: Internet se separa de ARPAnet
- 1993: Primer navegador web público, *Mosaic*
- 1994: World Wide Web Consortium (W3C)
  - Fundado por Tim Berners-Lee cuando dejó el CERN y se fue al MIT
  - 351 miembros (Alcatel-Lucent, Telefónica, Univ. Oviedo, UPM, Ayto. Zaragoza)

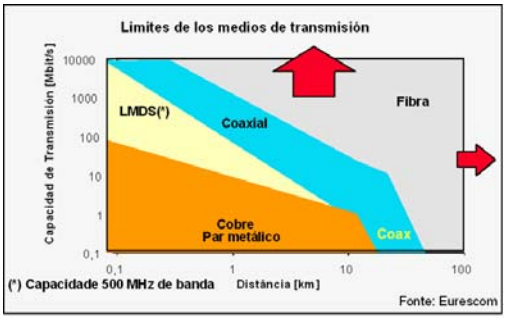
Protocolos de Internet

- Varios niveles



Protocolos de Internet – Nivel de acceso físico

- Ethernet
  - LAN con topología de bus o estrella
  - 10 - 100 Mbps, cable coaxial
  - Fast Ethernet: versión a 100Mbps
  - Gigabit Ethernet: 10GBASE-R/LR/SR (long range short range, etc.)
- DSL (Digital subscribe line)
  - Línea telefónica tradicional (cable de cobre)
  - ADSL (asymmetric DSL)
- HFC (Hybrid Fiber Coaxial)
  - Red de banda ancha por cable de TV
- FTTH (Fiber To The Home)
  - Fibra óptica



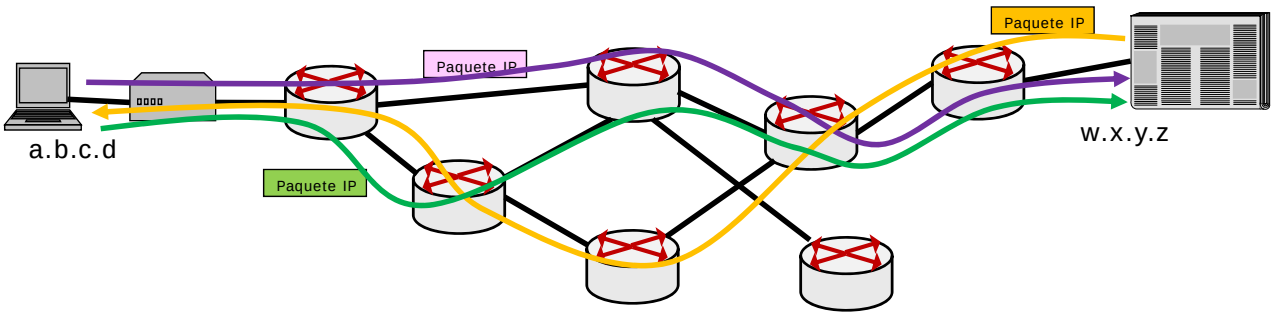
- Wi-Fi (wireless fidelity)
  - IEEE 802.11 **.11** (1-2 Mbps); **.11a** (hasta 54 Mbps); **.11b** (hasta 11 Mbps); .**11g** (más de 54 Mbps)
  - 802.11n: siguiente generación Wi-Fi
- WiMAX (Worldwide Interoperability for Microwave Access)
  - Alternativa al cable y al DSL

Protocolos de Internet – Nivel de red

- **Internet Protocol (IP)**
  - Encaminamiento de paquetes (datagramas) entre nodos de la red
- Protocolo datagrama, no orientado a conexión y no fiable
  - No hay recuperación de errores
  - Hay comprobación de errores, y los paquetes IP erróneos se tiran, sin notificar al emisor
  - Soporta fragmentación de datos en paquetes IP (<1400 bytes)
- Direcciones IP
  - Cada máquina (host) tiene una dirección única
  - IPv4
    - 32 bits (4 octetos, entre 0 y 255, separados por .)
    - 4 clases de direcciones IP: A, B, C (network ID+host ID)
    - 127.x.x.x se reservan para designar la propia máquina
    - dirección IP dinámica es una IP asignada mediante un servidor DHCP (Dynamic Host Configuration Protocol)
  - IPv6
    - 128 bits (32 dígitos): 3.4×10^{38} direcciones
    - Para cada persona en la Tierra se pueden asignar varios millones de IPs

Protocolos de Internet – Nivel de red

- Encaminamiento en una red IP



Protocolos de Internet – Nivel de red

- Con IPv4 hay hasta 2^{32} = 4.294.967.296 direcciones únicas (realmente son 3.200 a 3.300 por las clases que hay definidas)
  - Solución: Network Address Translation (NAT), RFC 2663 (1999)

- NAT Básico
  - Traduce una IP privada en una pública

- **NAPT** (*Network Address and Port Translation*)
  - Un grupo de nodos en una red privada comparten una IP pública (NAT de muchos a uno)
    - Se traduce IP_{ext}+$Puerto_{ext}$ -> IP_{int}+$Puerto_{int}$
    - Práctico para conexiones hacia el exterior
    - Problema: se requiere mantener el estado en el NAPT
  - NAT estática (*port forwarding*)
    - Se puede configurar un puerto para uso permanente
    - "abrir un puerto" para que sea accesible desde el exterior

    => A TENER EN CUENTA CUANDO SE INSTALA UN SERVIDOR EN UN NODO TRAS UN NAPT

Protocolos de Internet – Nivel de transporte

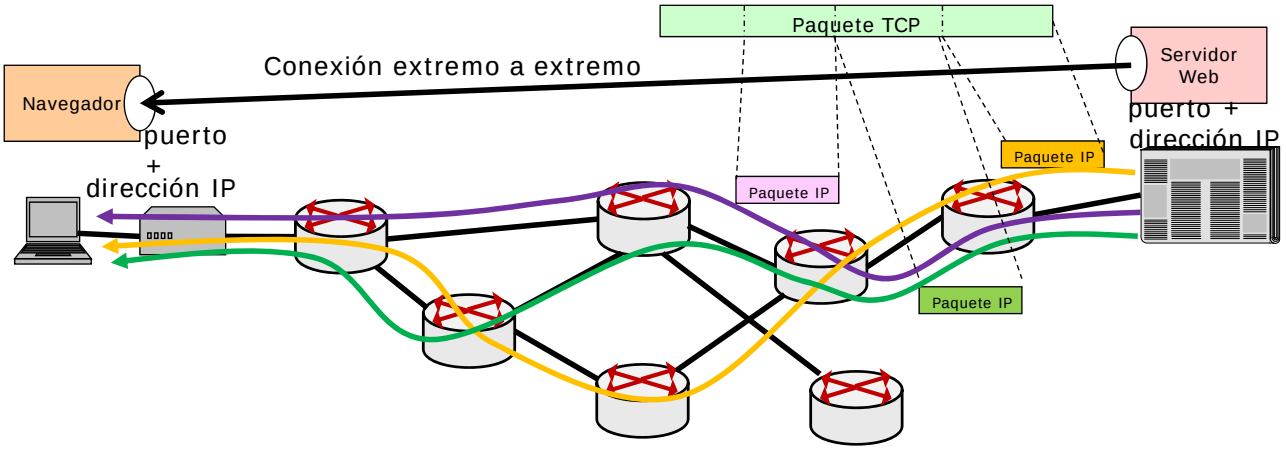
- TCP (Transmission Control Protocol)
  - Protocolo orientado a conexión, fiable (recuperación de errores), y con control de flujo
  - Establece un camino de bytes (*byte stream*)

- UDP (User Datagram Protocol)
  - Protocolo no orientado a conexión y no fiable
  - Si se recibe un paquete sin errores se pasa al proceso de usuario destinatario, si no, se descarta silenciosamente
  - Límite de tamaño de datagrama: 64 KB
  - UDP Multicast

- En una red local, UDP es más eficiente y normalmente no hay errores
- A través de Internet es más seguro utilizar TCP

Protocolos de Internet – Nivel de transporte

- TCP sobre IP



Protocolos de Internet – Nivel de aplicación

Servicios de aplicación tradicionales:

- Servicios de soporte
  - DNS: Domain name service protocol
    - Traducción de nombres en direcciones (funciona sobre UDP y TCP)
  - SNMP: Simple Network Management Protocol
    - Gestión de red
- Servicios de transferencia de ficheros
  - FTP: File Transport Protocol (sobre TCP)
  - TFTP: Trivial FTP (sobre UDP)
- Servicio de login
  - Telnet: Terminal virtual remoto (sobre TCP)
- Servicios de correo electrónico:
  - SMTP: Simple Mail Transfer Protocol
    - Protocolo para transferencia de emails entre servidores de email y de clientes al servidor
  - IMAP: Internet Message Access Protocol (recuperación de email)
  - POP: Post Office Protocol (recuperación de email)

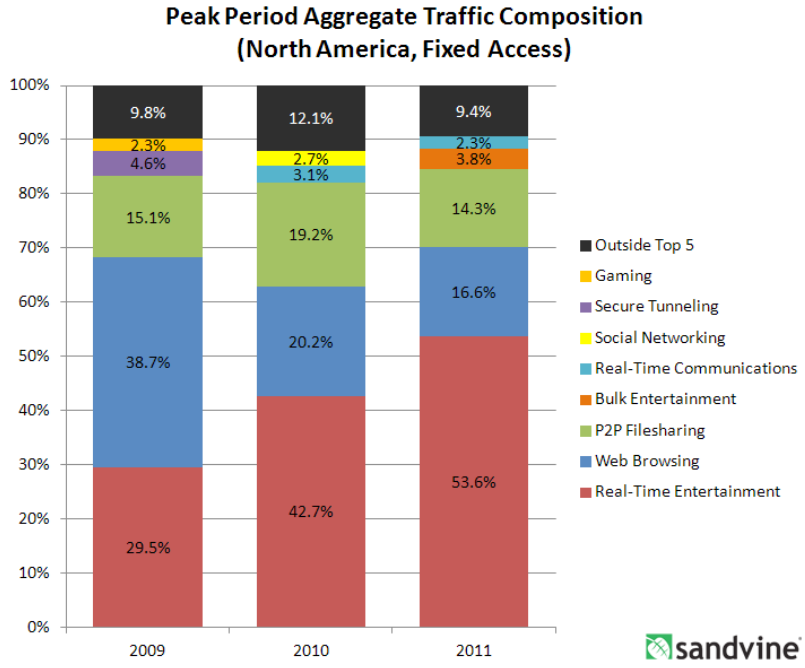
Protocolos de Internet – Nivel de aplicación

- BitTorrent
  - Compartición de archivos p2p
- NTP: Network Time Protocol
  - Sincronización de tiempo usando UDP
- HTTP: HyperText Transfer Protocol (Web)
  - Aplicación cliente/servidor que usa TCP para recuperar páginas HTML
- IRC: Internet Relay Chat
- LDAP: Lightweight Directory Access Protocol
  - Acceso y mantenimiento de información de directorio distibuida en una red IP
  - Organización jerárquica
- NFS: Network file system protocol
- NIS: Network information service (“Yellow Pages”)
  - Información de configuración de sistemas: nombres de login, passwords, directorios home, grupos
  - Estructura plana, se intentó arreglar con NIS+, pero hoy día se usa más LDAP
- NNTP: Network News Transfer Protocol (News)
- RPC: Remote Procedure Call
  - Comunicación entre aplicaciones simulando llamadas a procedimientos
- SSH: Secure Shell
  - Uso de criptografía de clave pública para transmisión segura de información

Protocolos de aplicación - Estadísticas

Sandvine Global Internet Phenomena Report - Fall 2011

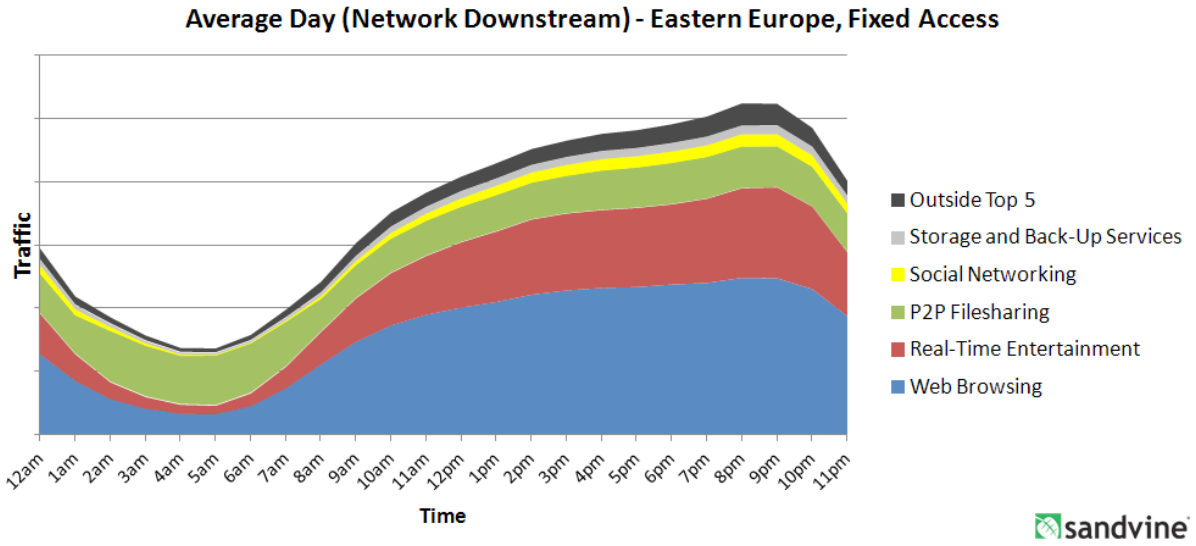
http://www.sandvine.com/downloads/documents/10-26-2011_phenomena/Sandvine%20Global%20Internet%20Phenomena%20Report%20-%20Fall%202011.PDF



Protocolos de aplicación - Estadísticas

Sandvine Global Internet Phenomena Report - Fall 2011

http://www.sandvine.com/downloads/documents/10-26-2011_phenomena/Sandvine%20Global%20Internet%20Phenomena%20Report%20-%20Fall%202011.PDF

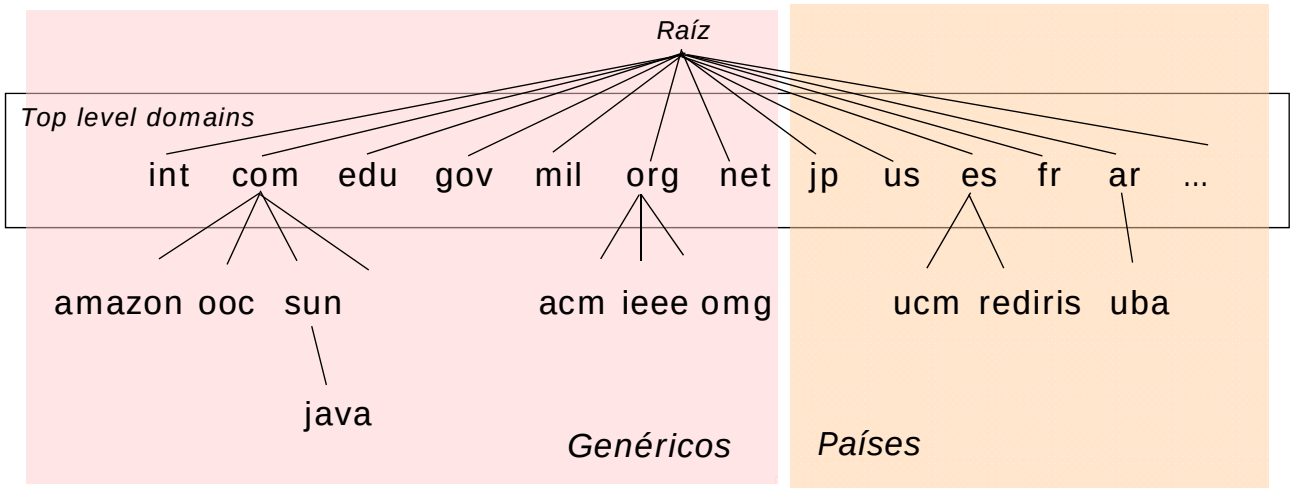


Internet - DNS

- DNS (Domain Name System)
  - Resolución de nombres
    - Dado el nombre de un host (*www.ucm.es*), obtener su IP (*147.96.1.5*)
  - Resolución inversa de direcciones
    - Dado la IP, devuelve el nombre asociado
  - Resolución de servidores de correo
    - Dado un nombre de dominio (*gmail.com*) obtener el servidor a través del cual debe realizarse la entrega del correo electrónico (*gmail-smtp-in.l.google.com*)
- Fully qualified host name (FQHN)
  - Nombre completo de un host
  - Host name (*www*) + Domain name (*ucm.es*)

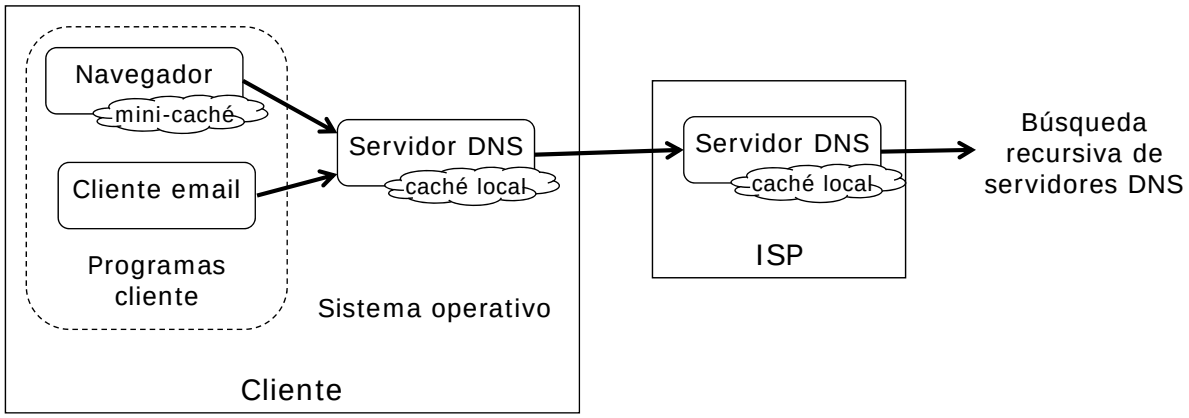
Internet - DNS

- Los nombres de dominio están jerarquizados
- También los servidores DNS
  - Primarios o maestros: Guardan los datos de un espacio de nombres
  - Secundarios o esclavos: Obtienen los datos de los servidores primarios
  - Locales o caché



Internet - DNS

- Funcionamiento



Internet - DNS

- Servidores DNS
  - Raíz
    - 13 DNS root servers (letras A a M): 10 en Estados Unidos, 1 en Suecia, 1 en Reino Unido y 1 en Japón
    - Mirrors (En España hay una réplica del F, gestionada por Espanix)
  - Operadoras en España: http://www.adslayuda.com/dns.html

- BIND (Berkeley Internet Name Domain)
  - Servidor DNS más común en Unix
  - Disponible en http://www.isc.org/
    - Actualmente BIND 9
      - *named*, una bibioteca de resolución de sistemas de nombres de dominio y un paquete de herramientas para monitorizar el correcto funcionamiento de todo el sistema (bind-utils)
      - Protocolos de seguridad DNSSEC y TSIG (*Transaction SIGnature*), soporte de IPv6, *nsupdate* (actualizaciones dinámicas), notificación DNS, *rndc flush*, vistas y procesamiento en paralelo

Ejercicios DNS

- Prueba en una máquina con Unix los siguientes comandos:
  - **host**
    - Permite pedir información a servidores DNS sobre máquinas
    - Si se usa host sin parámetros lista los parámetros

```
host www.ucm.es
host -t a www.ucm.es
host 147.96.1.15
```

  - **dig** (*domain information groper*)
    - Tiene más opciones de control, por lo que es la más útil para detectar problemas en la configuración de los servidores de DNS

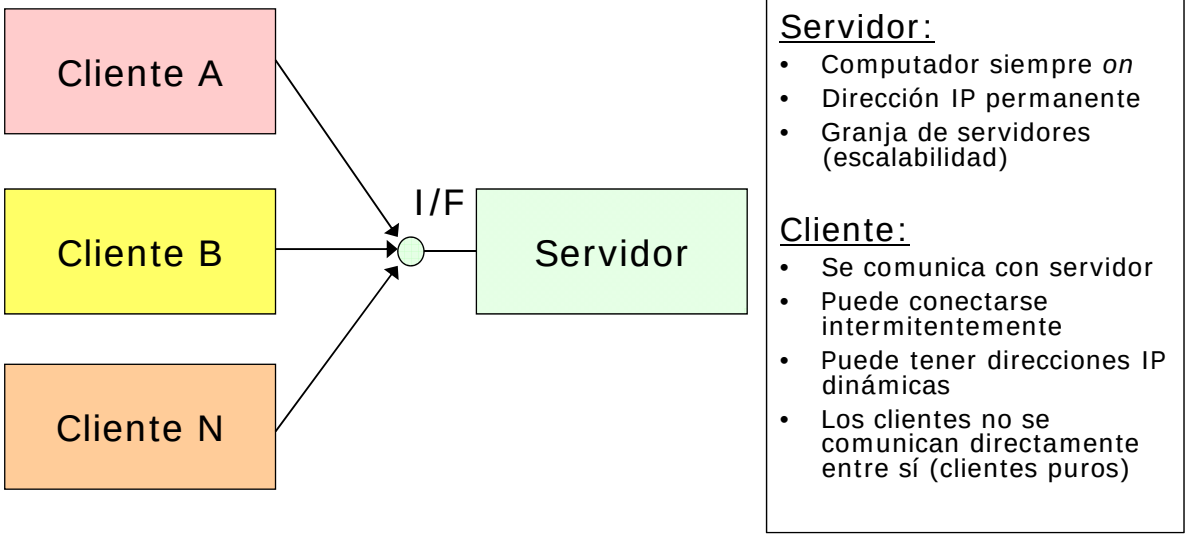
```
dig www.ucm.es
dig +trace www.ucm.es
```

  - **nslookup** (también en windows)
    - Está *deprecated* en BIND 9. Mejor usar *host*

- Investiga cómo saber qué DNS estás usando en tu computadora
  - ¿Cómo se puede cambiar?

Arquitecturas de aplicaciones en internet

- Modelo **cliente-servidor**

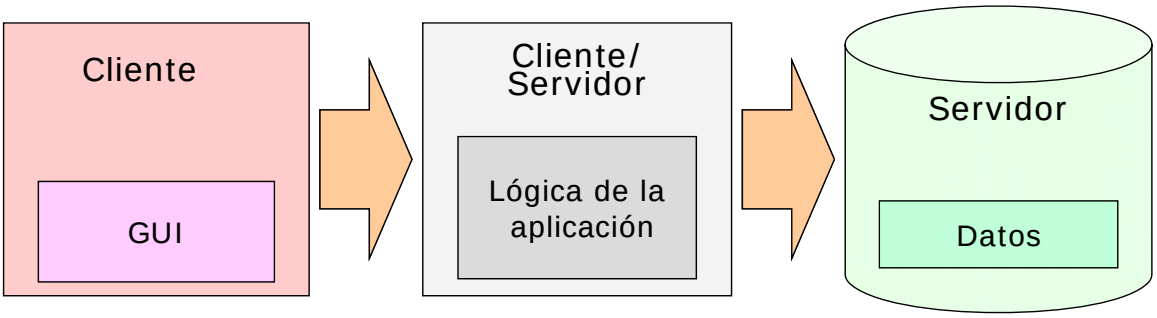


Arquitecturas de aplicaciones en internet

- Clientes pesados vs. Servidores pesados

Arquitecturas de aplicaciones en internet

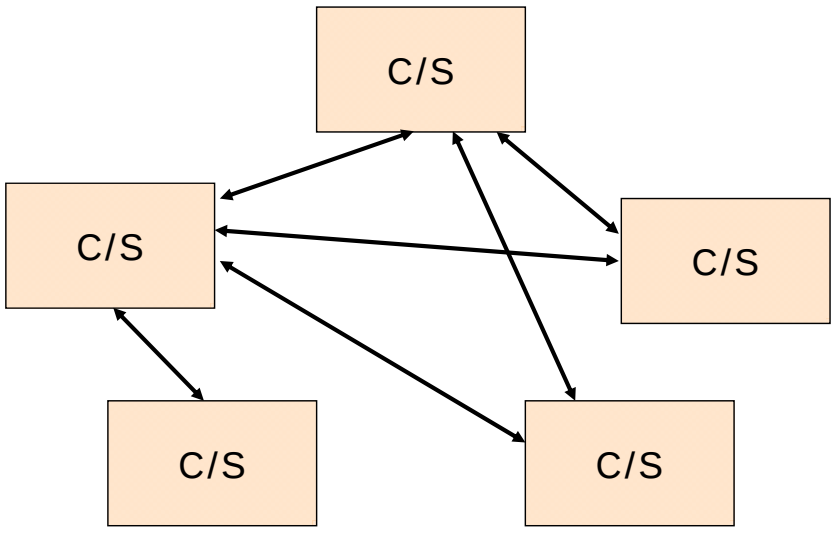
- Sistemas Cliente/Servidor a 3 niveles (*three-tier systems*)
  - Es un caso particular de arquitectura de n-niveles (*multi-tier system*)
  - Independencia de la plataforma e implementación de cada parte



Cuestión: ¿En qué se diferencia del patrón MVC?

Arquitecturas de aplicaciones en internet

- Modelo **p2p (peer-to-peer)**
  - Compartición de archivos, telefonía VoIP, cálculo científico



- Sistemas terminales arbitrarios se comunican directamente
- Pares se conectan intermitentemente y cambian sus direcciones IP
- Ningún nodo es imprescindible: alta robustez
- Altamente escalable
- Distribución de costes entre los usuarios
- Difícil de administrar

Arquitecturas de aplicaciones en internet

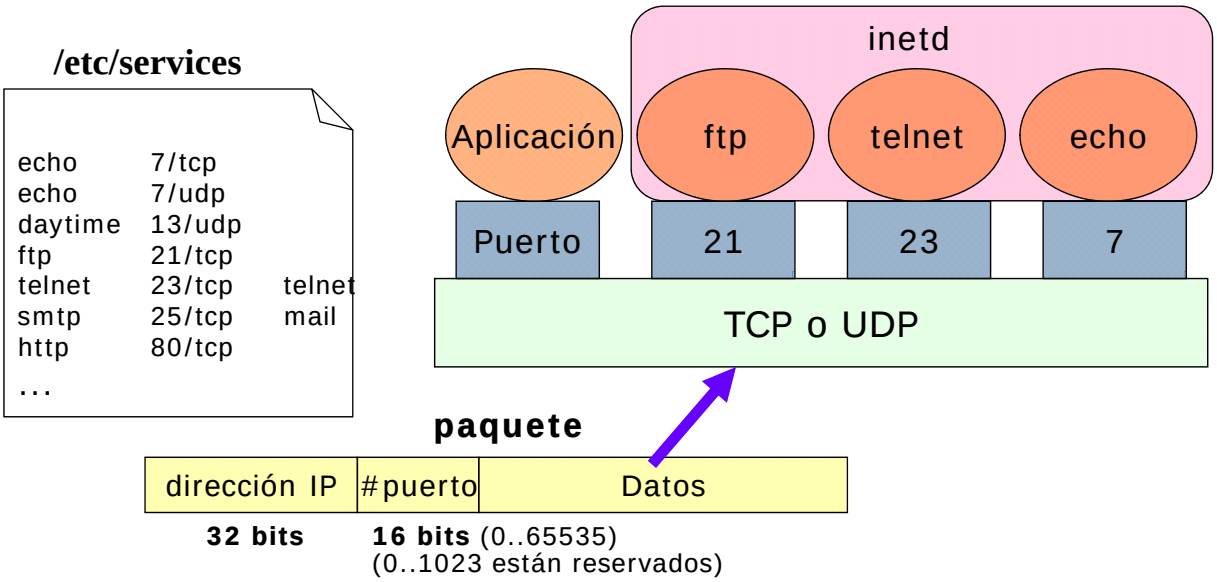
- Tipos de redes p2p
  - Redes P2P centralizadas
    - Un único servidor sirve para establecer las transacciones entre los nodos de la red
      - Almacena y distribuye los nodos donde se almacenan los contenidos
      - Todas las comunicaciones (peticiones y encaminamientos entre nodos) dependen exclusivamente de la existencia del servidor
    - Modelo de las primeras redes p2p (Napster)
  - Redes P2P híbridas, semicentralizadas o mixtas
    - Uno o varios servidores sirven como hub
      - Administra los recursos de banda ancha, enrutamientos y comunicación entre nodos pero sin saber la identidad de cada nodo y sin almacenar información alguna
      - El servidor no comparte archivos de ningún tipo con ningún nodo
    - BitTorrent, eDonkey
  - Redes P2P puras o totalmente descentralizadas
    - Todas las comunicaciones son directamente de usuario a usuario
      - Es algún usuario quien permite enlazar esas comunicaciones
      - No existe un servidor central que maneje conexiones de red ni direcciones
    - Kademlia, Ares Galaxy, Gnutella, Freenet

Programación de aplicaciones en internet

- Bajo nivel
  - **Sockets**
- Middleware
  - RPC
  - CORBA
  - DCOM
  - Java RMI
  - ODBC/JDBC (para acceso a bases de datos)
  - **Servicios Web**

Puertos en internet

- Cada servicio está asociado a un puerto



Sockets

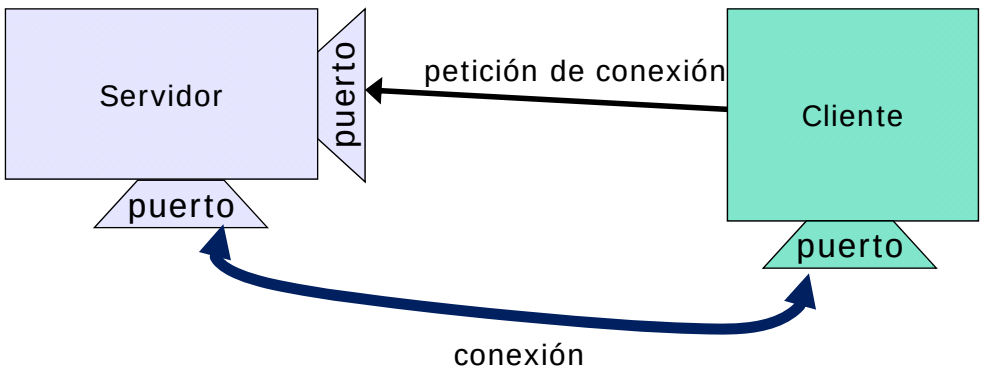
- Abstracción programable de canal de comunicación
  - dirección de socket = dirección IP + número de puerto
- Dos procesos se pueden intercambiar información usando un par de sockets:
  - Los mensajes van entre un socket de un proceso y otro socket en otro proceso
  - Cuando los mensajes son enviados, se encolan en el socket hasta que el protocolo de red los haya transmitido
  - Cuando llegan, los mensajes son encolados en el socket de recepción hasta que el proceso receptor los recoja
- Ciclo de vida igual al sistema de E/S Unix:
  - Creación: apertura del socket
  - Lectura y Escritura: recepción y envío de datos por el socket
  - Destrucción: cierre del socket

Sockets

- Tipos de sockets
  - **Socket Stream** (TCP)
    - Servicio de transporte orientado a conexión:
      - En el servidor un socket atiende peticiones de conexión
      - En el cliente un socket solicita una conexión
      - Una vez conectados, se pueden usar para transmitir datos en ambas direcciones
  - **Socket Datagrama** (UDP)
    - Servicio de transporte no orientado a conexión:
      - Permite enviar paquetes independientes de información
      - En cada datagrama es necesario enviar el descriptor del socket local y la dirección del socket que debe recibir el datagrama
      - No se garantiza la transmisión del paquete
  - **Socket Raw**
    - Permite acceder a la capa de software de red subyacente o a protocolos de más bajo nivel
      - Se utiliza para depuración de código de los protocolos

Socket streams

- Establecimiento de conexión

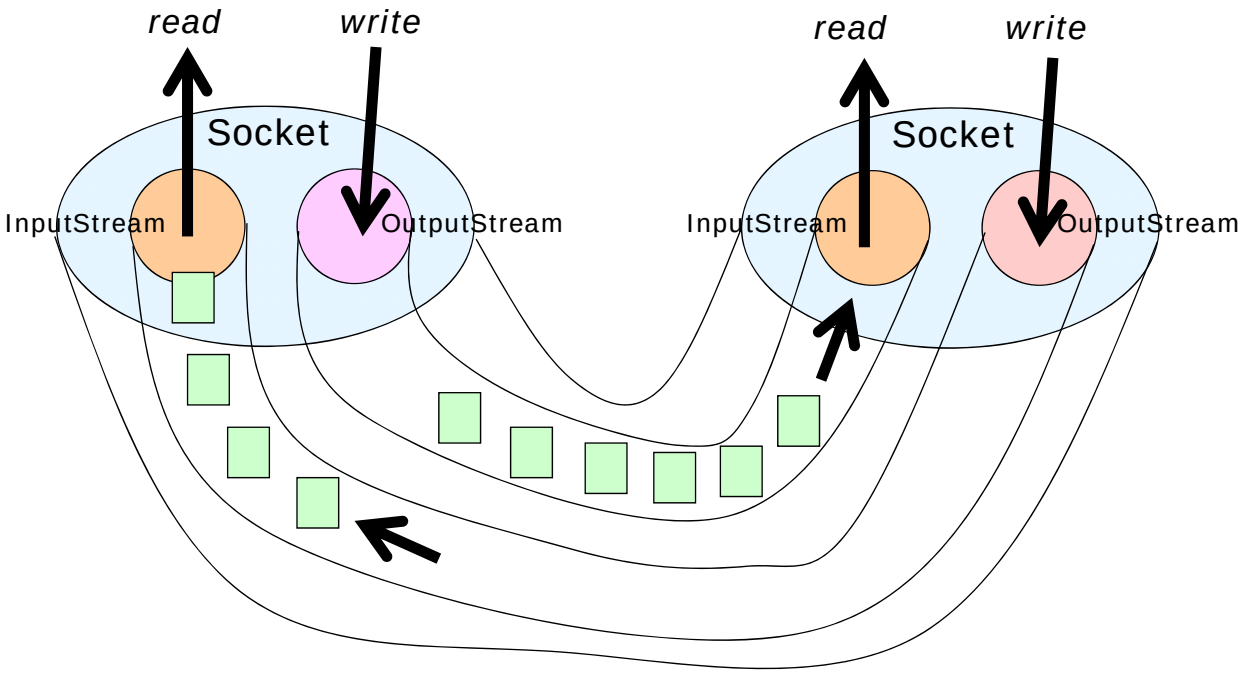


Programación de sockets con Java

- Paquete **java.io**
  - Flujos (streams) de datos
    - Flujos de bytes: InputStream, OutputStream, DataInputStream, etc.
    - Flujos de caracteres: Reader, Writer, PrintWriter, BufferedReader, etc.
    - System.in, System.out, System.err

- Paquete **java.net**
  - Dirección internet
    - InetAddress
  - Datagramas
    - DatagramPacket: paquete (datagrama) a enviar/recibir
    - DatagramSocket: socket para enviar/recibir datagramas
  - Conexiones TCP (sockets)
    - ServerSocket: socket donde el servidor espera peticiones
    - Socket: implementa el socket de comunicaciones, con dos streams (InputStream y OutputStream)
  - URL: Uniform Resource Locator, se refiere a un recurso del a Web

Programación de sockets con Java



Programación de sockets con Java - Cliente

- Crear un socket

```
Socket ladoCliente;
ladoCliente = new Socket (“maquina”, numeroPuerto);
```

- Asociar un flujo (stream) de datos para entrada (recepción) y otro para salida (emisión)

```
DataInputStream entrada; PrintStream salida;
entrada = new DataInputStream(ladoCliente.getInputStream());
salida = new PrintStream (ladoCliente.getOutputStream() );
```

- Leer y escribir de los flujos asociados

```
String texto = entrada.readLine();
salida.println(texto);
```

- Para finalizar, cerrarlos flujos y el socket

```
salida.close( );
entrada.close( );
ladoCliente.close( );
```


Programación de sockets con Java - Cliente

- Ejemplo: Cliente de eco

```
import java.net.*;
import java.io.*;

class ClienteEco {
public static void main( String args[] ) {
   final intPUERTOECO = 7;
   String maquina="localhost";

   Socketeco = null;
   BufferedReaderentrada;
   PrintWritersalida;

   BufferedReaderstdin = new BufferedReader (new InputStreamReader (System.in));
   String texto;
```

Programación de sockets con Java - Cliente

- Ejemplo: Cliente de eco

```
try {
 eco = new Socket ( maquina, PUERTOECO );
 salida =  new PrintWriter(new OutputStreamWriter (eco.getOutputStream()), true);
 entrada = new BufferedReader(new InputStreamReader (eco.getInputStream()) );
 if (eco != null && entrada != null && salida != null) {
       while ((texto = stdin.readLine()) != null) {
                salida.println(texto);
                System.out.println("echo: " + entrada.readLine());
       }

       salida.close();
       entrada.close();
       eco.close();
       stdin.close();
  }
 }
 catch (UnknownHostException e) {
       System.err.println("Máquina desconocida: maquina");
 }
 catch (IOException e) {
       System.err.println("Fallo en la conexión: "+e);
 }
 } // main
} // ClienteEco
```


Programación de sockets con Java - Servidor

- Crea un socket de servidor

```
ServerSocket servicio;
servicio = new ServerSocket (numeroPuerto);
```

- Espera la recepción de peticiones de conexión

```
Socket socketServicio;
socketServicio = servicio.accept();
```

- Acepta la nueva conexión y crea flujos de entrada y salida de datos que asocia al nuevo socket
  - Lo normal es crear una hebra asociada para dar servicio en la nueva conexión
- Lee y escribe de los flujos asociados
- Para finalizar, cierra los flujos y los sockets

Programación de sockets con Java - Servidor

- Ejemplo: servidor de eco

```
import java.net.*;
import java.io.*;

class ServidorEco {
         public static void main( String args[] ) {
         final intPUERTOECO = 7;

         ServerSockets = null;
         Socket cliente = null;
         BufferedReader entrada;
         PrintWritersalida;

         String texto;
```


Programación de sockets con Java - Servidor

- Ejemplo: servidor de eco

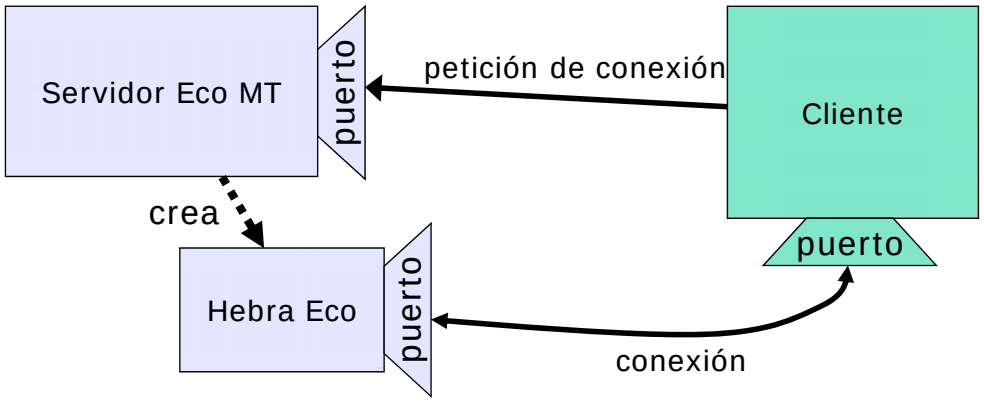
```
   try {
         s = new ServerSocket (PUERTOECO);

         cliente = s.accept();
         salida =  new PrintWriter( new OutputStreamWriter(
                                        cliente.getOutputStream()), true );
         entrada = new BufferedReader( new InputStreamReader
                                        (cliente.getInputStream()) );
         if (cliente != null && entrada != null && salida != null)
                  while ( true ) {
                           texto = entrada.readLine();
                           salida.println(texto);
                           System.out.println("echo: " + texto);
                  }

    } catch (IOException e) {
         System.err.println("Fallo en la conexion: "+e);
    }
  } // main
} // ServidorEco
```

Programación de sockets con Java - Servidor

- Ejemplo: servidor de eco para múltiples clientes



Programación de sockets con Java - Servidor

- Ejemplo: servidor de eco para múltiples clientes

```
import java.net.*;
import java.io.*;

public class ServidorEcoMT {
    public static void main( String args[] ) {
          final intPUERTOECO = 7;

          ServerSockets = null;

          try {
                    s = new ServerSocket (PUERTOECO);

                    while ( true )
                              new HebraEco ( s.accept() ).start();

          } catch (IOException e) {
             System.err.println("Fallo en la conexion: "+e);
          }
   } // main
} // ServidorEco
```

Programación de sockets con Java - Servidor

```
import java.net.*;
import java.io.*;

class HebraEco extends Thread {
    private Socket socket = null;
    private String texto;

    public HebraEco (Socket s) {
         super("HebraEco");
         socket = s;
    }

    public void run() {
       try {
         PrintWriter salida = new PrintWriter(new OutputStreamWriter(
                                                  socket.getOutputStream()), true);
         BufferedReader entrada = new BufferedReader( new InputStreamReader
                                                  (socket.getInputStream()) );
         while ( true ) {
                  texto = entrada.readLine();
                  salida.println(texto);
                  System.out.println("echo ["+socket.getPort()+"]: " + texto);
         }
       } catch (IOException e) { System.out.println("Se cerro " + socket.getPort());}
    } // run
} // HebraEco
```

Programación de datagramas con Java

- Clase **DatagramPacket**
  - Representa un paquete datagrama
  - Consta de:
    - Una dirección (SocketAddress=InetAddress+puerto)
    - Un contenido o buffer de byes (byte[])
  - Se acceden con métodos get/set
- Clase **DatagramSocket**
  - Representa el socket para enviar y recibir datagramas

```
DatagramSocket puertoDatagramas;
puertoDatagramas = new DatagramSocket (numeroPuerto);
```

  - Recibe y envía datagramas
    - receive(DatagramPacket p)
    - void send(DatagramPacket p)
  - Cerrar al finalizar su uso:
    - close()

Programación de datagramas con Java

- Ejemplo: cliente de eco

```
import java.net.*;
import java.io.*;

class ClienteEcoDatagrama {
    public static void main( String args[] ) {
    final intPUERTOECO = 7;

    BufferedReader stdIn = new BufferedReader (new InputStreamReader (System.in));
    String texto, recibido;
    byte[] buf = new byte[256];
```


Programación de datagramas con Java

```
  try {
    InetAddress direccion = InetAddress.getByName("localhost");
    DatagramSocket eco = new DatagramSocket();
    DatagramPacket paquete;

    while ((texto = stdIn.readLine()) != null) {
          // envia el texto al servidor de eco:
          buf = texto.getBytes();
          paquete = new DatagramPacket (buf, buf.length, direccion, PUERTOECO);
          eco.send(paquete);
          System.out.println(texto);
          // recibe la respuesta:
          paquete = new DatagramPacket(buf, buf.length);
          eco.receive(paquete);
          recibido = new String(paquete.getData());
          System.out.println("echo: " + recibido);
    }
    eco.close();
  }
  catch (UnknownHostException e) {System.err.println(e); }
  catch (SocketException e) {System.err.println(e); }
  catch (IOException e) {System.err.println(e); }
 } // main
} // ServidorEcoDatagrama
```

URL

- URL (Universal Resource Locator)
  - Representa un objeto o servicio en Internet
    - http://maquina/directorio/fichero.html

identificador de protocolo

nombre de recurso (p.ej., un fichero)

- Clase URL

```
URL javasoft = new URL("http://www.javasoft.com/");
URL javadownload = new URL(javasoft, "download.html");
```

  - Métodos para manipular el URL:
    - getProtocol
    - getHost
    - getPort
    - getFile
    - getRef
    - openStream: establece una conexión y devuelve un InputStream para leer
      - Equivale a: openConnection().getInputStream()

URL

- Lectura de un URL

```
import java.io.*;
import java.net.*;

class EjemploLecturaURL {
public static void main(String[] args) {
        try {
            URL ucm = new URL("http://www.ucm.com/");
            DataInputStream dis = new DataInputStream(ucm.openStream());
            String inputLine;

            while ((inputLine = dis.readLine()) != null) {
                System.out.println(inputLine);
            }
            dis.close();
        } catch (MalformedURLException me) {
            System.out.println("MalformedURLException: " + me);
        } catch (IOException ioe) {
            System.out.println("IOException: " + ioe);
        }
    }
}
```

Ejercicios

1. La clase `java.net.NetworkInterface` tiene el método `getHardwareAddress()`, que permite obtener la dirección MAC de un dispositivo. Haz un programa para imprimir la dirección MAC de tu PC
2. El método `isReachable()` de la clase `InetAddress` permite comprobar que el host remoto es alcanzable y está activo. Haz un programa que haga `ping` al host que se dé como entrada
3. ¿Cómo se puede conseguir la dirección IP del *localhost*?
4. Escribe un programa que escanee los puertos de un host dado
   - Se trata de hacer un bucle que intente establecer conexión (abrir socket) con todos los puertos del host, desde 0 a 65535
5. Adapta la clase `EjemploLecturaURL` para que guarde la página HTML leída en el fichero `pagina.html`
6. Escribe un cliente SMTP para enviar emails
   - El formato que hay que utilizar se especifica en la RFC 1822/3

Bibliografía

- Sobre este tema hay información abundante en internet, tanto sobre los protocolos de internet como de la programación con Java
- Libros sobre internet:
  - James F. Kurose y Keith W. Ross. *Computer Networking: A top-Down Approach Featuring the Internet*. 6ª ed. Addison-Wesley 2013
- Algunas referencias prácticas sobre programación con Java:
  - API de Java (javadocs)
    - Versión reciente: http://docs.oracle.com/javase/7/docs/api/
  - Tutorial de Java sobre programación con sockets: http://docs.oracle.com/javase/tutorial/networking/sockets/
  - Learn Java by Examples: http://www.kodejava.org
  - Java World: http://www.javaworld.com/jw-12-1996/jw-12-sockets.html