

Redes de Comunicaciones 2

3º del grado de Ingeniería Informática

Capa de
aplicación

Capa de aplicación

Eloy Anguiano
eloy.anguiano@uam.es

Oscar Delgado
oscar.delgado@uam.es

Escuela Politécnica Superior
Universidad Autónoma de Madrid



Capa de
aplicación

Principios

Arquitecturas de
aplicación

Comunicación
entre procesos

Parte I

Introducción

Principios

Objetivos

- ① **Conceptual:** Aspectos de implementación de protocolos de aplicación.
 - Modelos de servicio de la capa de transporte
 - Paradigma cliente–servidor
 - Paradigma Peer-to-peer (P2P)
- ② **Aplicación:** Aprender el uso de los protocolos de aplicación más comunes.
 - HTTP
 - FTP
 - SMTP / POP3 / IMAP
 - DNS
- ③ **Programación:** Programación de aplicaciones de red.
 - API de sockets

Principios Aplicaciones de red

Capa de
aplicación

Principios

Ap. de red

Características de las
aplicaciones

Arquitecturas de
aplicación

Comunicación
entre procesos

- e-mail
- web
- Mensajería instantánea
- Login remoto (ssh, telnet)
- Compartición de ficheros P2P
- Juegos multiusuario en red
- Vídeo en streaming
- Voz sobre IP
- Vídeo-conferencias en tiempo real
- Grid computing
- ...

Principios

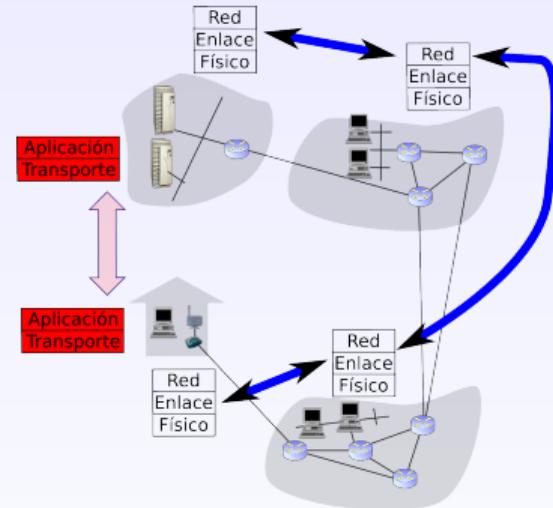
Características de las aplicaciones

Los programas deben tener las siguientes características

- Deben ejecutarse en varios sistemas finales
- Comunicarse a través de red
- Ejemplo: el servidor web se comunica con el browser independientemente del sistema en el que esté el browser o el servidor.

No se necesita escribir software para los elementos de red

- Los elementos intermedios de la red no ejecutan aplicaciones de usuario.
- Las aplicaciones en los extremos de la conexión permiten un desarrollo rápido y simple.



Arquitecturas de aplicación

Capa de
aplicación

Principios

Arquitecturas de
aplicación

Cliente-servidor

P2P

Híbridas

Comunicación
entre procesos

- Cliente-servidor
- Entre pares (peer-to-peer, P2P)
- Híbrido cliente-servidor y P2P

Arquitecturas de aplicación

Cliente–servidor

Capa de
aplicación

Principios

Arquitecturas de
aplicación

Cliente–servidor

P2P

Híbridas

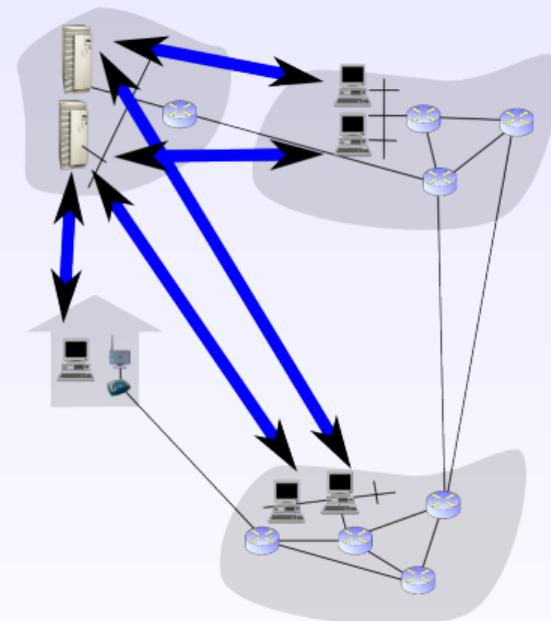
Comunicación
entre procesos

Servidor

- Siempre en un host
- Dirección IP permanente
- Granjas de servidores para el escalamiento

Cliente

- Se comunica con el servidor
- Puede conectarse intermitentemente
- Puede tener IP dinámica
- Los clientes no se comunican entre si



Arquitecturas de aplicación

P2P

Capa de
aplicación

Principios

Arquitecturas de
aplicación

Cliente-servidor

P2P

Híbridas

Comunicación
entre procesos

- Descentralizada
 - Ausencia de un Servidor Central para el control
 - Los participantes pueden comunicarse directamente entre si
 - Todos los nodos actúan como clientes y servidores
- Distribuida
 - La información no está alojada en un solo sitio
- Carga balanceada
 - Se intenta equilibrar la carga entre todos los participantes
- Redundancia de información
 - Se duplica información para hacerla más accesible
- Alta disponibilidad
 - La caída de un host no bloquea el servicio
- Optimización de uso de recursos
 - Procesamiento, almacenamiento, ancho de banda, etc.

Arquitecturas de aplicación

P2P

Capa de aplicación

Principios

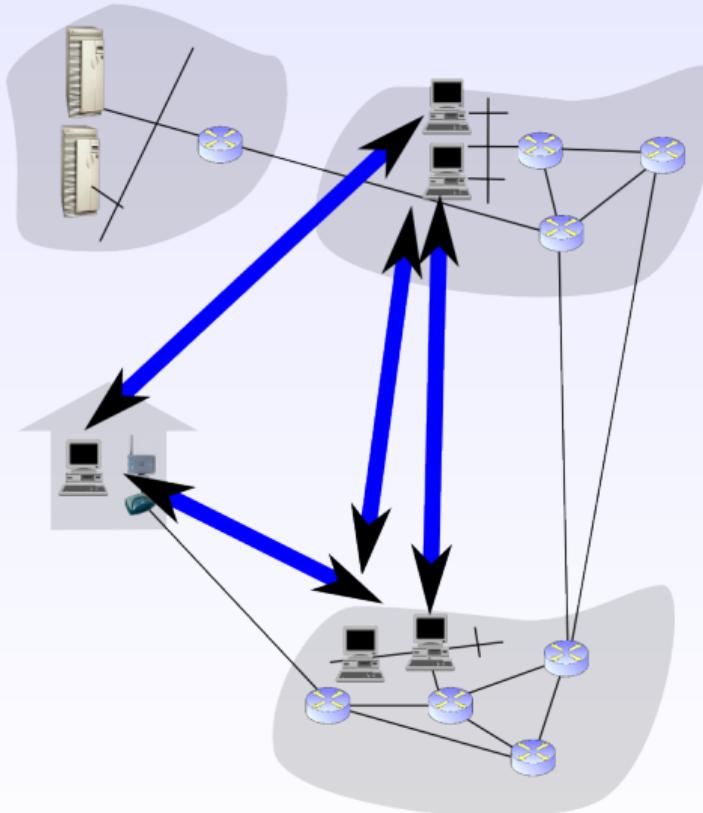
Arquitecturas de aplicación

Cliente-servidor

P2P

Híbridas

Comunicación entre procesos



Ejemplos

- Kazaa
- eMule
- Gnutella
- LimeWire
- WinMX
- BitTorrent

Arquitecturas de aplicación Híbridas

Capa de
aplicación

Principios

Arquitecturas de
aplicación

Cliente-servidor
P2P

Híbridas

Comunicación
entre procesos

Skype

- Aplicación de voz sobre IP
- Servidor centralizado: permite encontrar la dirección remota con la que se quiere conectar
- Conexión directa cliente–cliente

Mensajería instantánea

- La comunicación se realiza entre dos usuarios en P2P
- Servicio centralizado
 - Presencia de cliente:
 - Un usuario registra su IP asociada a su usuario cuando se registra
 - Localización
 - Un usuario localiza a la persona con la que quiere comunicar consultando al servidor

Comunicación entre procesos

Capa de
aplicación

Principios

Arquitecturas de
aplicación

Comunicación
entre procesos

Sockets

Direcccionamiento

Protocolos

Aplicación requisitos

Requerimientos

Transporte

Aplicaciones transporte

Proceso: programa en ejecución en un sistema informático

Dentro del mismo sistema informático: se comunican utilizando la comunicación entre procesos (IPC) definida por cada sistema operativo.

Entre diferentes sistemas informáticos: se comunican utilizando el intercambio de mensajes.

Proceso Cliente

Proceso que inicia la comunicación

Proceso Servidor

Proceso que espera peticiones de los clientes

Nota: Las aplicaciones P2P tienen procesos clientes y servidores en todos los sistemas informáticos en los que están funcionando.

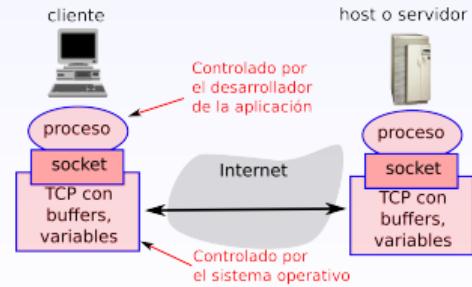
Comunicación entre procesos

Sockets

- Los procesos envían y reciben mensajes a y desde la red a través de sus sockets.
- Sockets: son análogos a puertas (puertos).
 - Proceso transmisor
 - Saca mensajes por la puerta
 - Confía en la infraestructura de transporte al otro lado de la puerta, la cual lleva los mensajes al socket en el proceso receptor
 - Proceso receptor
 - Recibe mensajes por la puerta
 - Los pasa al proceso receptor

API (Application Programming Interface)

- Se debe elegir el protocolo de transporte (UDP, TCP)
- Se pueden seleccionar muchos parámetros del protocolo



Comunicación entre procesos

Direccionamiento de procesos

Capa de
aplicación

Principios

Arquitecturas de
aplicación

Comunicación
entre procesos

Sockets

Direccionamiento

Protocolos

Aplicación requisitos

Requerimientos

Transporte

Aplicaciones transporte

- Para que un proceso reciba un mensaje, éste debe tener un **identificador único**
- Un host tiene una **dirección IP** única de 32 bits

¿Es suficiente con la dirección IP para identificar al proceso??

- Es necesario incluir un **número de puerto** para identificar a un proceso dentro del host.

Ejemplos de puertos

- Servidor HTTP: 80
- Servidor e-mail: 25
- En UNIX (linux) están descritos en el fichero /etc/services

Comunicación entre procesos

Características de los protocolos de aplicación

- Tipos de mensajes intercambiados
 - Request, response ...
- Sintáxis del mensaje
 - Qué campos hay en un mensaje y cómo se organizan
- Semántica del mensaje
 - Significado de los campos de información
- Reglas de cómo y cuando un proceso envía o responde a mensajes

Comunicación entre procesos

Características de los protocolos de aplicación

Tipos de protocolos

- De dominio público (abiertos)
 - Definidos en RFCs
 - Permite interoperabilidad
 - Ejemplos: HTTP, SMTP ...
- Propietarios
 - Restringido el acceso a las definiciones por las compañías propietarias
 - En general no permite interoperabilidad
 - Ejemplos: Skype, Messenger ...

Comunicación entre procesos

Servicios al transporte en la aplicación

Transferencia de datos confiable

- Algunas aplicaciones (audio/vídeo) pueden tolerar pérdida
- La mayoría (FTP, telnet) requieren transferencia 100 % confiable

Tasa de transferencia

- Aplicaciones con ancho de banda sensitivo (bandwidth-sensitive application): aplicaciones multimedia
- Aplicaciones elásticas (elastic applications) e-mail, FTP

Retardo

Algunas aplicaciones (Telefonía IP, juegos interactivos, teleconferencias) requieren bajo retardo para ser “efectivas”

Seguridad

Integridad de datos, encriptación, autenticación, etc.

Comunicación entre procesos

Requerimientos del servicio de transporte de aplicaciones

Aplicación	Pérdida de datos	Tasa de transferencia	Sensible al tiempo
Transferencia de ficheros	Sin pérdida	Elástica	NO
e-mail	Sin pérdida	Elástica	NO
documentos WEB	Sin pérdida	Elástica	NO
audio tiempo real	Tolerante	5 kbps–1 Mbps	SÍ (100's ms)
vídeo tiempo real	Tolerante	10 kbps–5 Mbps	SÍ (100's ms)
audio almacenado	Tolerante	5 kbps–1 Mbps	SÍ (pocos segundos)
juegos interactivos	Tolerante	pocos kbps o más	SÍ (100's ms)
mensajería instantánea	Sin pérdida	Elástica	SÍ y NO

Comunicación entre procesos

Servicios de protocolos de transporte en Internet

Capa de
aplicación

Principios

Arquitecturas de
aplicación

Comunicación
entre procesos

Sockets

Direcccionamiento

Protocolos

Aplicación requisitos

Requerimientos

Transporte

Aplicaciones transporte

Servicio TCP

Orientado a conexión: se requiere una acuerdo entre cliente y servidor

Trasporte confiable: sin pérdidas.

Control de flujo: el transmisor no sobrecargará al receptor.

Control de congestión: frena al transmisor cuando la red está sobrecargada

No provee: garantías de retardo ni ancho de banda mínimos

Servicio UDP

Transferencia de datos no confiable entre los procesos transmisor y receptor.

No provee acuerdo entre los procesos, confiabilidad, control de flujo, control de congestión, ni garantías de retardo o ancho de banda.

Comunicación entre procesos

Aplicaciones y protocolos de transporte

Capa de
aplicación

Principios

Arquitecturas de
aplicación

Comunicación
entre procesos

Sockets

Direcccionamiento

Protocolos

Aplicación requisitos

Requerimientos

Transporte

Aplicaciones transporte

Aplicación	Protocolo de aplicación	Protocolo de transporte
e-mail	SMTP (RFC 2821)	TCP
Terminal remota	Telnet (RFC 854)	TCP
WEB	HTTP (RFC 2616)	TCP
Transferencia de ficheros	FTP (RFC 959)	TCP
Streaming multimedia	RTP (RFC 1889)	TCP o UDP
Telefonía IP	SIP, RTP ...	Típicamente UDP



Capa de
aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

Algo de código

Tipos de
servidores

Parte II

Programación de sockets



Características API

Capa de aplicación

Características

API

Servicios proporcionados

Comunicaciones

Llamadas básica
biblioteca de
sockets

Algo de código

Tipos de
servidores

- Las API permiten desarrollar aplicaciones utilizando los servicios de TCP.
- La interfaz de sockets se introdujo en la versión 4.2BSD de UNIX.
- Se ha transformado en un estándar “de facto”.
- Una variante de los BSD sockets es el API Winsock desarrollada para sistemas WINTEL.

Características Servicios proporcionados

Capa de
aplicación

Características

API
Servicios
proporcionados

Comunicaciones

Llamadas básicas
biblioteca de
sockets

Algo de código

Tipos de
servidores

Se definen diferentes tipos de socket en función de los servicios proporcionados

Stream sockets: Confiables, orientados a conexión. Utilizan los servicios de TCP.

Datagram Sockets: No confiables ni orientados a conexión. Utilizan los servicios de UDP.

Raw Sockets: acceso a niveles más bajos del protocolo TCP/IP.

Características

Servicios proporcionados

Datagram Socket Interface (SOCK_DGRAM) - UDP

- Define servicio no orientado a conexión (no hay 'handshake' antes de enviar los datos).
- Los datagramas se envían como paquetes independientes.
- El emisor añade explícitamente la IP y puerto de destino a cada paquete.
- El receptor extrae la IP y puerto de destino del paquete.
- No hay garantías ni de que lleguen los paquetes ni de que lo hagan ordenados.
- No hay segmentación ni reconstrucción de los paquetes.
- Ejemplo: NFS.

Características Servicios proporcionados

Capa de
aplicación

Características

API

Servicios
proporcionados

Comunicaciones

Llamadas básica
biblioteca de
sockets

Algo de código

Tipos de
servidores

Stream Interface (SOCK_STREAM) - TCP

- Define una conexión fiable.
- Los datos se envían sin errores y en el mismo orden en el que se envían.
- Hay control de flujo.
- No se imponen límites a los datos (se considera un flujo continuo o stream).
- Ejemplo: FTP.

Comunicaciones

No orientada a conexión

Capa de
aplicación

Características

Comunicaciones

No orientada a
conexión

Orientada a conexión

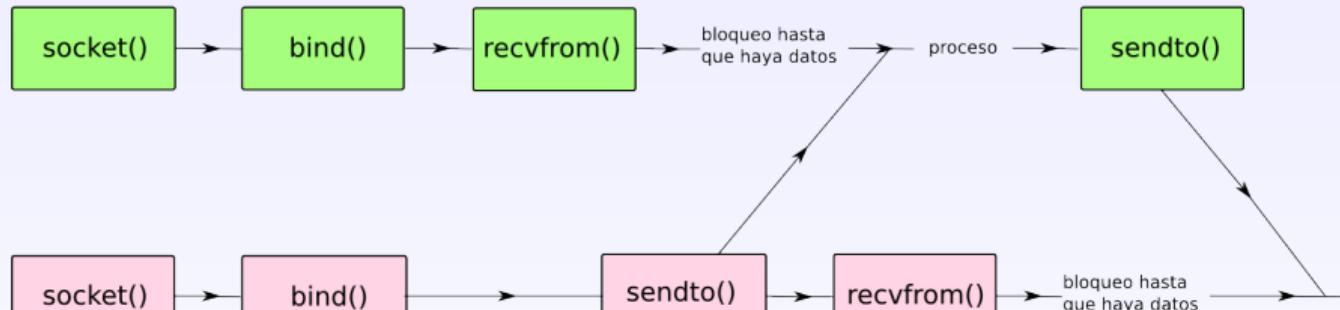
Interfaz bajo nivel

Llamadas básicas
biblioteca de
sockets

Algo de código

Tipos de
servidores

Servidor



Cliente

Comunicaciones Orientada a conexión

Capa de
aplicación

Características
Comunicaciones

No orientada a
conexión

Orientada a conexión
Interfaz bajo nivel

Llamadas básicas
biblioteca de
sockets

Algo de código

Tipos de
servidores

El cliente debe contactar al servidor

- El servidor debe estar ejecutándose previamente.
- El servidor debe haber creado un socket que espere el contacto de un cliente.

El cliente contacta al servidor:

- Crea un socket TCP especificando IP y puerto del proceso servidor.
- Cuando el cliente crea el socket, el cliente TCP establece conexión con el servidor TCP.

Cuando el servidor es contactado:

- El servidor TCP crea un nuevo socket para que el proceso servidor se comunique con ese cliente en particular.
- Un servidor puede comunicarse con distintos clientes.

Comunicaciones Orientada a conexión

Capa de
aplicación

Características

Comunicaciones

No orientada a
conexión

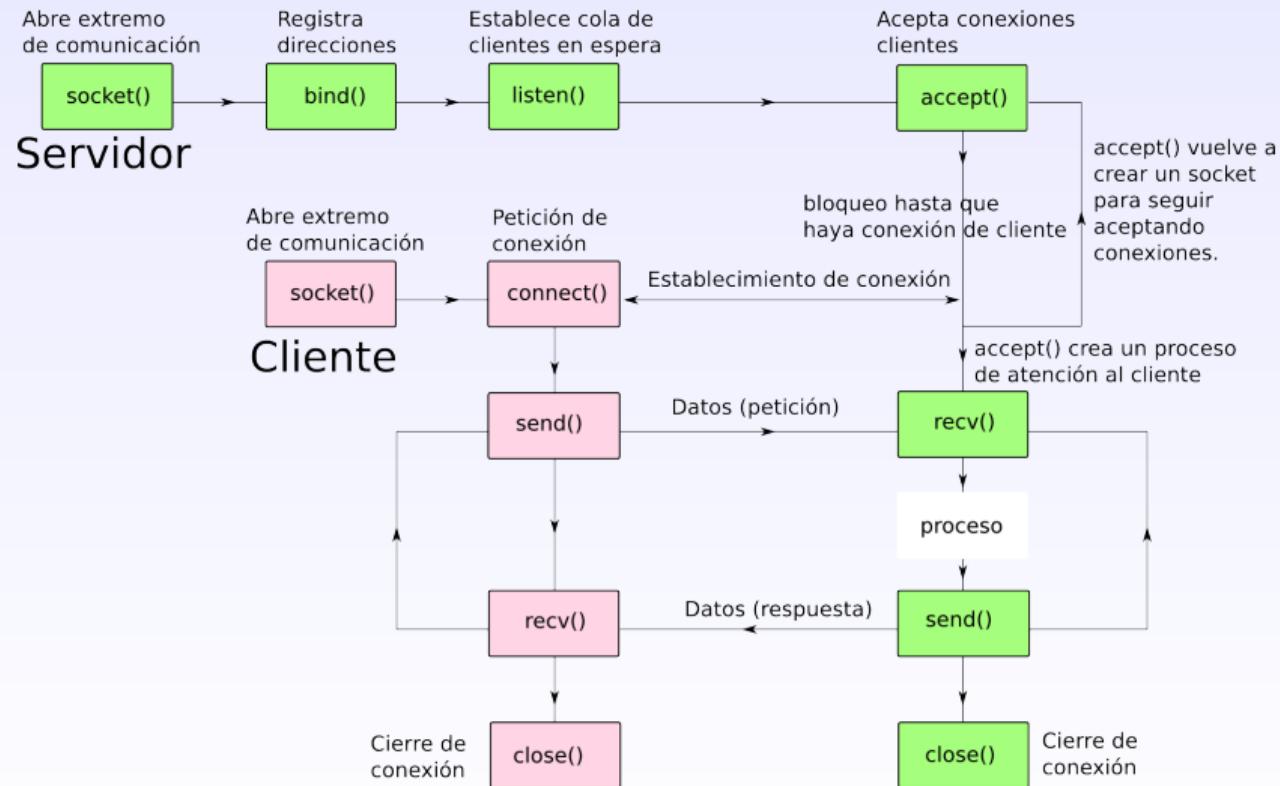
Orientada a conexión

Interfaz bajo nivel

Llamadas básicas
biblioteca de
sockets

Algo de código

Tipos de
servidores





Comunicaciones

Interfaz bajo nivel

Capa de aplicación

Características

Comunicaciones

No orientada a conexión

Orientada a conexión

Interfaz bajo nivel

Llamadas básicas
biblioteca de sockets

Algo de código

Tipos de servidores

Raw Socket Interface (SOCK_RAW)

- Acceso a los niveles más bajos de la pila de protocolos (IP,ICMP).
- Se utiliza a menudo para probar nuevos protocolos.
- Ejemplo: aplicación ping.

Llamadas básica biblioteca de sockets socket

Capa de
aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

socket
bind
listen
accept
connect
send/receive
close
Utilización de llamadas

Algo de código

Tipos de
servidores

Su función es iniciar un socket

Función

```
int sockfd = socket(int family, int type, int protocol)
```

family: Indica la familia de direccionamiento: AF_UNIX, AF_INET, AF_NS, AF_OS2, y AF_IUCV.

type: Indica el tipo de interfaz de sockets que se quiere utilizar: SOCK_STREAM, SOCK_DGRAM, SOCK_RAW y SOCK_SEQPACKET.

protocol: Protocolo, en el caso de INET: UDP, TCP, IP, o ICMP.

sockfd: Entero (similar a un descriptor de ficheros) devuelto por la llamada a socket(). Deberá pasarse como parámetro en las siguientes llamadas.

Llamadas básica biblioteca de sockets

bind

Capa de
aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

socket

bind

listen

accept

connect

send /receive

close

Utilización de llamadas

Algo de código

Tipos de
servidores

Su función es registrar un socket en un puerto

Función

int **bind**(int sockfd, struct sockaddr *localaddr, int addrlen)

sockfd: Parámetro devuelto por la llamada socket()

localaddr, addrlen: Parámetros para recibir la dirección del socket.

Después de la llamada a bind se tienen valores para los primeros tres parámetros de la 5-tupla:

- {protocol, local-address, local-port, foreign-address, foreign-port}

Llamadas básica biblioteca de sockets

listen

Capa de
aplicación

Características
Comunicaciones

Llamadas básica
biblioteca de
sockets

socket
bind
listen
accept
connect
send/receive
close

Utilización de llamadas

Algo de código

Tipos de
servidores

Su función es indicar la disposición a recibir llamadas por un puerto

Función

`int listen(int sockfd, int queue-size)`

`sockfd`: Parámetro devuelto por `socket()`

`queue-size`: Entero que indica el número de peticiones de conexión que pueden ser encoladas por el sistema antes de que el proceso local ejecute un `accept`.

Llamadas básica biblioteca de sockets

accept

Capa de
aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

socket
bind
listen
accept
connect
send/receive
close

Utilización de llamadas

Algo de código

Tipos de
servidores

Su función es registrar un socket en un puerto

Función

int **accept**(int sockfd, struct sockaddr *foreign-address, int addrlen)

sockfd: Parámetro devuelto por la llamada `socket()`

foreign-address: Estructura devuelta por la función con la información de dirección remota.

addrlen: Tamaño de la estructura devuelta por la función.

Esta llamada es típica de un servidor. Si hay una petición de conexión en la cola, `accept` la saca de la cola y crea un nuevo socket con las mismas propiedades de `sockfd`. Si no hay peticiones, se bloquea al proceso hasta que llegue alguna.

Llamadas básica biblioteca de sockets **connect**

Capa de
aplicación

Características
Comunicaciones

Llamadas básica
biblioteca de
sockets

socket
bind
listen
accept
connect
send/receive
close

Utilización de llamadas

Algo de código

Tipos de
servidores

Su función es la de realizar la conexión a un servidor. Es la llamada típica de un proceso cliente

Función

int **connect**(int sockfd, struct sockaddr *foreign-address, int addrlen)

sockfd: Parámetro devuelto por la llamada socket()

foreign-address: Estructura devuelta por la función con la información de dirección remota.

addrlen: Tamaño de la estructura devuelta por la función.

Llamadas básica biblioteca de sockets

send/receive

Capa de
aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

socket

bind

listen

accept

connect

send/receive

close

Utilización de llamadas

Algo de código

Tipos de
servidores

Su función es la de recibir y enviar datos a través del socket

Funciones

`ssize_t readv(int s, void *buf, size_t lon, int flags)`

`ssize_t recv(int s, void *buf, size_t lon, int flags)`

`ssize_t readfrom(int s, void *buf, size_t lon, int flags, struct sockaddr *desde, socklen_t *londesde)`

`ssize_t recvmsg(int s, struct msghdr *msg, int flags)`

`ssize_t read(int fd, void *buf, size_t nbytes)`

`ssize_t send(int s, const void *msg, size_t len, int flags)`

`ssize_t sendto(int s, const void *msg, size_t len, int flags, const struct sockaddr *to, socklen_t tolen)`

`ssize_t sendmsg(int s, const struct msghdr *msg, int flags) ssize_t write(int fd, void *buf, size_t nbytes)`

Llamadas básica biblioteca de sockets close

Capa de
aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

socket
bind
listen
accept
connect
send/receive
close

Utilización de llamadas

Algo de código

Tipos de
servidores

Su función es la de cerrar el socket, la conexión y liberar las estructuras de datos

Función

`int close(int sockfd)`

`sokcfd`: Parámetro devuelto por la llamada `socket()`

Llamadas básica biblioteca de sockets

Utilización de llamadas

Capa de
aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

socket
bind
listen
accept
connect
send /receive
close

Utilización de llamadas

Algo de código

Tipos de
servidores

Tipo	Conexión	Envío	Recepción
Servidor orientado a conexión	bind, list accept	write sendto	read recvfrom
Cliente orientado a conexión	connect, list	write sendto	read recvfrom
Servidor no orientado a conexión	bind, list	sendto sendto	recvfrom
Cliente no orientado a conexión	bind	sendto	recvfrom



Algo de código

Ejemplo de aplicación - Python

Capa de aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

Algo de código

Ejemplo de aplicación -
Python

Demo nio

Apertura

Aceptando

Proceso

Tipos de
servidores

Descripción aplicación de ejemplo

- El cliente lee una línea de caractéres (datos) desde el teclado y envía los datos al servidor.
- El servidor recibe los datos y convierte los caractéres a mayúsculas (procesa).
- El servidor envía los datos modificados al cliente.
- El cliente recibe los datos modificados y los muestra por pantalla.

Algo de código

Ejemplo de aplicación - Python

Cliente UDP

```
1 from socket import *
2
3 serverName = 'hostname'
4 serverPort = 12000
5 clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
6 message = raw_input('Input lowercase sentence:')
7 clientSocket.sendto(message,(serverName, serverPort))
8 modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
9 print modifiedMessage
10 clientSocket.close()
```



Algo de código

Ejemplo de aplicación - Python

Capa de aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

Algo de código

Ejemplo de aplicación -
Python

Demo nio

Apertura

Aceptando

Proceso

Tipos de
servidores

Servidor UDP

```
1 from socket import *
2
3 serverPort = 12000
4 serverSocket = socket(AF_INET, SOCK_DGRAM)
5 serverSocket.bind(('', serverPort))
6 print "Servidor preparado para recibir"
7 while 1:
8     message, clientAddress = serverSocket.recvfrom(2048)
9     modifiedMessage = message.upper()
10    serverSocket.sendto(modifiedMessage, clientAddress)
```

Algo de código

Ejemplo de aplicación - Python

Cliente TCP

```
1 from socket import *
2
3 serverName = 'servername'
4 serverPort = 12000
5 clientSocket = socket(socket.AF_INET, socket.SOCK_STREAM)
6 clientSocket.connect((serverName,serverPort))
7 sentence = raw_input('Ingrese texto en minusculas:')
8 clientSocket.send(sentence)
9 modifiedSentence = clientSocket.recv(1024)
10 print 'Desde el servidor:', modifiedSentence
11 clientSocket.close()
```

Algo de código

Ejemplo de aplicación - Python

Capa de
aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

Algo de código

Ejemplo de aplicación -
Python

Demonio

Apertura

Aceptando

Proceso

Tipos de
servidores

Servidor TCP

```
1 from socket import *
2
3 serverPort = 12000
4 serverSocket = socket(AF_INET, SOCK_STREAM)
5 serverSocket.bind(("", serverPort))
6 serverSocket.listen(1)
7 print "Servidor preparado para recibir"
8 while 1:
9     connectionSocket, addr = serverSocket.accept()
10    sentence = connectionSocket.recv(1024)
11    capitalizedSentence = sentence.upper()
12    connectionSocket.send(capitalizedSentence)
13    connectionSocket.close()
```

Algo de código

Conversión del proceso en demonio en C

```
194 void do_daemon(void){  
195     pid_t pid;  
196  
197     pid = fork(); /* Fork off the parent process */  
198     if (pid < 0) exit(EXIT_FAILURE);  
199     if (pid > 0) exit(EXIT_SUCCESS); /* Exiting the parent process. */  
200  
201     umask(0); /* Change the file mode mask */  
202     setlogmask (LOG_UPTO (LOG_INFO)); /* Open logs here */  
203     openlog ("Server\u25a1system\u25a1messages:", LOG_CONS | LOG_PID | LOG_NDELAY, LOG_LOCAL3);  
204     syslog (LOG_ERR, "Initiating\u25a1new\u25a1server.");  
205  
206     if (setsid() < 0) { /* Create a new SID for the child process */  
207         syslog (LOG_ERR, "Error\u25a1creating\u25a1a\u25a1new\u25a1SID\u25a1for\u25a1the\u25a1child\u25a1process.");  
208         exit(EXIT_FAILURE);  
209     }  
210  
211     if ((chdir("/")) < 0) { /* Change the current working directory */  
212         syslog (LOG_ERR, "Error\u25a1changing\u25a1the\u25a1current\u25a1working\u25a1directory\u25a1=\u25a1/\u25a1");  
213         exit(EXIT_FAILURE);  
214     }  
215  
216     syslog (LOG_INFO, "Closing\u25a1standard\u25a1file\u25a1descriptors");  
217     close(STDIN_FILENO); close(STDOUT_FILENO); close(STDERR_FILENO); /* Close out the  
218         standard file descriptors */  
219     return;  
220 }
```

Algo de código

Apertura del socket en C

Capa de
aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

Algo de código

Ejemplo de aplicación -
Python

Demonio

Apertura

Aceptando

Proceso

Tipos de
servidores

```
231 int initiate_server(void)
232 {
233     int sockval;
234     struct sockaddr_in Direccion;
235
236     syslog (LOG_INFO, "Creating\u00a5socket");
237     if ( (sockval = socket(AF_INET, SOCK_STREAM, 0)) < 0 ){
238         syslog(LOG_ERR, "Error\u00a5creating\u00a5socket");
239         exit(EXIT_FAILURE);
240     }
241
242     Direccion.sin_family=AF_INET; /* TCP/IP family */
243     Direccion.sin_port=htons(NFC_SERVER_PORT); /* Asigning port */
244     Direccion.sin_addr.s_addr=htonl(INADDR_ANY); /* Accept all adresses */
245     bzero((void *)&(Direccion.sin_zero), 8);
246
247     syslog (LOG_INFO, "Binding\u00a5socket");
248     if (bind (sockval, (struct sockaddr *)&Direccion, sizeof(Direccion))<0){
249         syslog(LOG_ERR, "Error\u00a5binding\u00a5socket");
250         exit(EXIT_FAILURE);
251     }
252
253     syslog (LOG_INFO, "Listening\u00a5connections");
254     if (listen (sockval, MAX_CONNECTIONS)<0){
255         syslog(LOG_ERR, "Error\u00a5listening");
256         exit(EXIT_FAILURE);
257     }
258     return sockval;
259 }
```



Algo de código

Aceptando conexiones en C

Capa de aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

Algo de código

Ejemplo de aplicación -
Python

Demonio

Apertura

Aceptando

Proceso

Tipos de
servidores

```
271 void accept_connection(int sockval)
272 {
273     int desc, len;
274     struct sockaddr Conexion;
275
276     len = sizeof(Conexion);
277
278     if ((desc = accept(sockval, &Conexion, &len)) < 0){
279         syslog(LOG_ERR, "Error\u00a1accepting\u00a1connection");
280         exit(EXIT_FAILURE);
281     }
282
283     launch_service(desc);
284     wait_finished_services();
285
286     return;
287 }
```

Algo de código

Lanzando proceso de servicio en C

Capa de
aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

Algo de código

Ejemplo de aplicación -
Python

Demonio

Apertura

Aceptando

Proceso

Tipos de
servidores

```
163 void launch_service(int connval)
164 {
165     int pid;
166     long type, aux;
167
168     pid = fork();
169     if (pid < 0) exit(EXIT_FAILURE);
170     if (pid == 0) return;
171
172     syslog (LOG_INFO, "New_access");
173     recv(connval, &aux, sizeof(long), 0);
174     type = ntohl(aux);
175
176     database_access(connval, type, NULL);
177     close(connval);
178     syslog (LOG_INFO, "Exiting_service");
179     exit(EXIT_SUCCESS);
180 }
```

Tipos de servidores

Implementación de servidores

Capa de
aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

Algo de código

Tipos de
servidores

Implementación de
servidores

Iterativo

Fork()

Pools

Pools

Pools

Threads

Iterativos

- Procesan las peticiones de una en una
- NO sirven para servidores reales

Concurrentes

- Pueden atender a varios clientes a la vez
- El proceso padre acepta la conexión (`accept()`), que es procesada por un proceso hijo o un thread.



Tipos de servidores

Servidor iterativo

Capa de aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

Algo de código

Tipos de
servidores

Implementación de
servidores

Iterativo

Fork()

Pools

Pools

Pools

Threads

```
1 int
2 main(int argc, char **argv)
3 {
4     int listenfd, connfd;
5     socklen_t clilen, addrlen;
6     struct sockaddr *cliaddr;
7
8     /* Contiene las llamadas a socket(), bind() y listen() */
9     listenfd = Tcp_listen(argv[1], argv[2], &addrlen);
10
11    for ( ; ; ) {
12        connfd = Accept(listenfd, cliaddr, &clilen);
13
14        process_request(connfd);
15
16        Close(connfd);
17    }
18}
19}
```

Tipos de servidores

Servidor concurrente: fork() para cada petición

Capa de
aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

Algo de código

Tipos de
servidores

Implementación de
servidores

Iterativo

Fork()

Pools

Pools

Pools

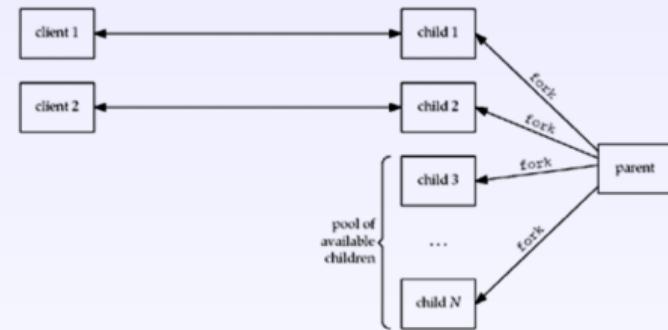
Threads

```
1
2 int
3 main(int argc, char **argv)
4 {
5     int    listenfd, connfd;
6     socklen_t clilen, addrlen;
7     struct sockaddr *cliaddr;
8
9     /* Contiene las llamadas a socket(), bind() y listen() */
10    listenfd = Tcp_listen(argv[1], argv[2], &addrlen);
11
12    for ( ; ; ) {
13        connfd = Accept(listenfd, cliaddr, &clilen);
14
15        if ( (chldpid = Fork()) == 0) {
16            process_request(connfd); /* Procesa la petición */
17            exit(0);
18        }
19
20        /* Padre cierra el descriptor de la conexión del hijo (duplicada) */
21        Close(connfd);
22    }
23 }
```

Tipos de servidores

Pools de procesos o threads

- Crear un proceso (en menor medida, un thread) es un proceso computacionalmente costoso
- Para peticiones cortas, puede tardarse más en crear el proceso que en atender la propia petición
- **Solución:** pre-crear los procesos o threads → pool



Tipos de servidores

Pools de procesos o threads

Capa de
aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

Algo de código

Tipos de
servidores

Implementación de
servidores

Iterativo

Fork()

Pools

Pools

Pools

Pools

Threads

Gestión dinámica del tamaño del pool

- ¿Qué tamaño debe tener el pool?
- Gestión dinámica del pool:
 - + Si el número de procesos libres está a punto de agotarse: crear nuevos
 - El sistema vuelve al equilibrio conforme van acabando las conexiones

Tipos de servidores

Servidor concurrente: pool de procesos

Capa de
aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

Algo de código

Tipos de
servidores

Implementación de
servidores

Iterativo

Fork()

Pools

Pools

Pools

Pools

Threads

```
1 int
2 main(int argc, char **argv)
3 {
4     int listenfd, i;
5     socklen_t addrlen;
6
7     /* Contiene las llamadas a socket(), bind() y listen() */
8     listenfd = Tcp_listen(argv[1], argv[2], &addrlen);
9
10    my_lock_init(NULL);
11    for (i = 0; i < nchildren; i++)
12        if ((childpid = Fork()) == 0) {
13            child_main(i, listenfd, addrlen); /* Esta funcion nunca retorna */
14        }
15
16    for ( ; ; )
17        pause(); /* Todo el trabajo lo hacen los hijos */
18
19 }
```

Tipos de servidores

Servidor concurrente: pool de procesos

Capa de
aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

Algo de código

Tipos de
servidores

Implementación de
servidores

Iterativo

Fork()

Pools

Pools

Pools

Pools

Threads

```
21 void
22 child_main(int i, int listenfd, int addrlen)
23 {
24
25     for ( ; ; ) {
26         clilen = addrlen;
27         my_lock_wait();
28         connfd = Accept(listenfd, cliaddr, &clilen);
29         my_lock_release();
30
31         process_request(connfd);
32         Close(connfd);
33     }
34 }
```

Tipos de servidores

Un nuevo thread para cada petición

Capa de aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

Algo de código

Tipos de
servidores

Implementación de
servidores

Iterativo

Fork()

Pools

Pools

Pools

Threads

```
1 int main(int argc, char **argv)
2 {
3     int listenfd, connfd;
4     pthread_t tid;
5     socklen_t clilen, addrlen;
6     struct sockaddr *cliaddr;
7
8     /* Contiene las llamadas a socket(), bind() y listen() */
9     listenfd = Tcp_listen(argv[1], argv[2], &addrlen);
10
11    for ( ; ; ) {
12        connfd = Accept(listenfd, cliaddr, &clilen);
13
14        Pthread_create(&tid, NULL, &doit, (void *) connfd);
15    }
16
17 }
18
19 void * doit(void *arg)
20 {
21     void web_child(int);
22
23     Pthread_detach(pthread_self());
24     process_request((int) arg);
25     Close((int) arg);
26     return(NULL);
27 }
```

Tipos de servidores

Un nuevo thread para cada petición

Capa de
aplicación

Características

Comunicaciones

Llamadas básica
biblioteca de
sockets

Algo de código

Tipos de
servidores

Implementación de
servidores

Iterativo

Fork()

Pools

Pools

Pools

Pools

Threads

Tipo	Tiempo de proceso en CPU (en segundos)
Servidor iterativo (baseline)	-
Concurrente, un fork() por cada cliente	20.90
Concurrente, pool de hijos	2.07
Concurrente, pool de threads	1.93
Concurrente, un thread para cada cliente	0.99



Capa de
aplicación

Introducción

Funcionamiento
HTTP

Petición HTTP

Respuesta HTTP

Práctica

Cookies

Proxy

Parte III

HTTP (HyperText Transfer Protocol)

Introducción

Algo de “jerga”

Capa de
aplicación

- Una **página web** está constituida por **objetos**
- Un objeto puede ser un fichero en HTML, una imagen JPEG, un applet Java
- ...
- Una página web está constituida por un **fichero HTML base** que incluye referencias a objetos.
- Cada objeto es direccionable por una **URL**
- URL: Unified Resource Locator
- HTML: HyperText Markup Language
- HTTP: HyperText Transfer Protocol

Ejemplo de URL: www.escuela.edu/departamento/imag.jpg

Introducción
Jerga
Generalidades

Funcionamiento
HTTP

Petición HTTP

Respuesta HTTP

Práctica

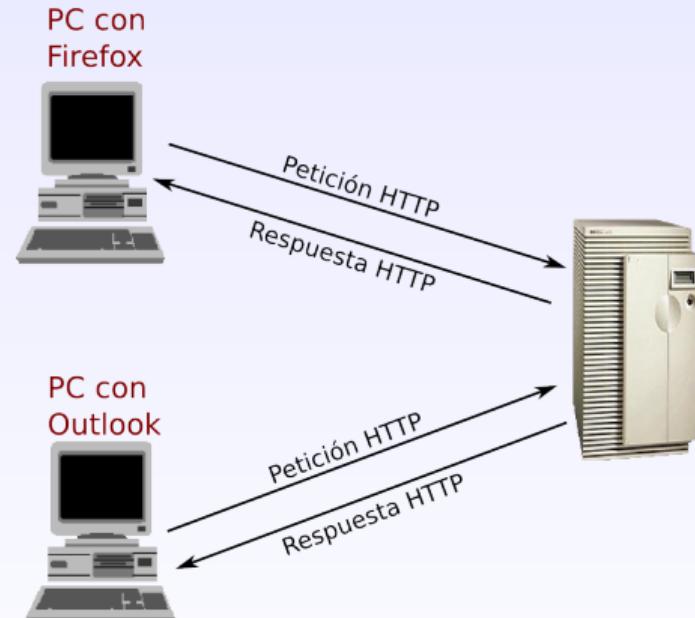
Cookies

Proxy

Introducción

Generalidades HTTP

- HTTP es el protocolo de la capa de aplicación para la Web
- Sigue un modelo cliente servidor
 - cliente:** es un “browser” que solicita, recibe y muestra los objetos Web
 - servidor:** el servidor recibe peticiones, responde y envía objetos



Funcionamiento HTTP

Capa de
aplicación

Introducción

Funcionamiento
HTTP

Tiempo de respuesta
Conexiones
No persistente
Problemas
Persistente

Petición HTTP

Respuesta HTTP

Práctica

Cookies

Proxy

- Utiliza TCP
 - ① El cliente inicia la conexión TCP con el servidor (puerto 80)
 - ② El servidor acepta la conexión TCP del cliente
 - ③ Se intercambian mensajes HTTP entre el browser (cliente HTTP) y el servidor.
 - ④ Se cierra la conexión TCP
- HTTP no conserva el estado
 - El servidor no mantiene información sobre las peticiones del cliente

Funcionamiento HTTP

Tiempo de respuesta

Capa de
aplicación

Introducción

Funcionamiento
HTTP

Tiempo de respuesta
Conexiones
No persistente
Problemas
Persistente

Petición HTTP

Respuesta HTTP

Práctica

Cookies

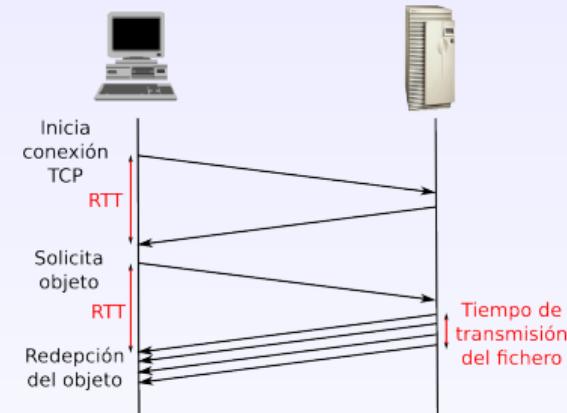
Proxy

Definición de RTT

Tiempo ocupado en enviar un paquete pequeño desde el cliente al servidor y su regreso

Tiempo de respuesta

- Un RTT para iniciar la conexión
- Un RTT por requerimiento HTTP y primeros bytes de la respuesta
- Tiempo de transmisión del archivo



Funcionamiento HTTP

Tipos de conexiones HTTP

Capa de
aplicación

Introducción

Funcionamiento
HTTP

Tiempo de respuesta
Conexiones
No persistente
Problemas
Persistente

Petición HTTP

Respuesta HTTP

Práctica

Cookies

Proxy

- **HTTP no persistente**

- Como mucho se envía un objeto por conexión TCP
- HTTP/1.0 usa HTTP no persistente

- **HTTP persistente**

- En una conexión TCP pueden enviarse múltiples objetos entre el cliente y el servidor
- HTTP/1.1 usa conexiones persistentes de forma predeterminada

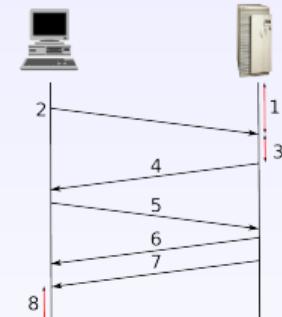
Funcionamiento HTTP

HTTP no persistente

Supongamos que se introduce la URL

www.escuela.edu/departamento/index.html

- ① El **servidor espera** por conexiones en el puerto 80 de el host www.escuela.edu
- ② El **cliente HTTP inicia la conexión** TCP con el servidor www.escuela.edu en el puerto 80
- ③ El **servidor acepta** la conexión
- ④ El **servidor** le comunica al cliente la **aceptación**
- ⑤ El **cliente HTTP envía el mensaje de solicitud** al socket TCP con la URL, indicando que se solicita el objeto `departamento/index.html`
- ⑥ El **servidor recibe la petición y envía un mensaje de respuesta** conteniendo el objeto solicitado y lo envía a su socket
- ⑦ El **servidor cierra** la conexión TCP
- ⑧ El **cliente recibe el mensaje de respuesta** **conteniendo el objeto** solicitado, lo analiza, lo presenta y encuentra 10 referencias a otros objetos.



Con las referencias encontradas el cliente repite todo el proceso hasta que todas las referencias a objetos están resueltas

Funcionamiento HTTP

Problemas de HTTP no persistente

Capa de
aplicación

Introducción

Funcionamiento
HTTP

Tiempo de respuesta
Conexiones
No persistente

Problemas
Persistente

Petición HTTP

Respuesta HTTP

Práctica

Cookies

Proxy

- Requiere 2 RTTs por objeto
- OS debe trabajar y dedicar recursos para cada conexión TCP
- El navegador abre conexiones paralelas generalmente para traer objetos referenciados

Funcionamiento HTTP

HTTP persistente

Capa de
aplicación

Introducción

Funcionamiento
HTTP

Tiempo de respuesta
Conexiones
No persistente
Problemas
Persistente

Petición HTTP

Respuesta HTTP

Práctica

Cookies

Proxy

- El servidor deja las conexiones abiertas después de enviar la respuesta
- Mensajes HTTP subsecuentes entre los mismos cliente/servidor son enviados por la conexión abierta

Persistencia sin “pipelining”

- El cliente envía nuevo requerimiento sólo cuando el previo ha sido recibido
- Un RTT por cada objeto referenciado

Persistencia con “pipelining”

- Predeterminado en HTTP/1.1
- El cliente envía requerimientos tan pronto éste encuentra un objeto referenciado
- Tan pequeño como un RTT por todas las referencia

Petición HTTP

Formato

Capa de
aplicación

Introducción

Funcionamiento
HTTP

Petición HTTP

Formato

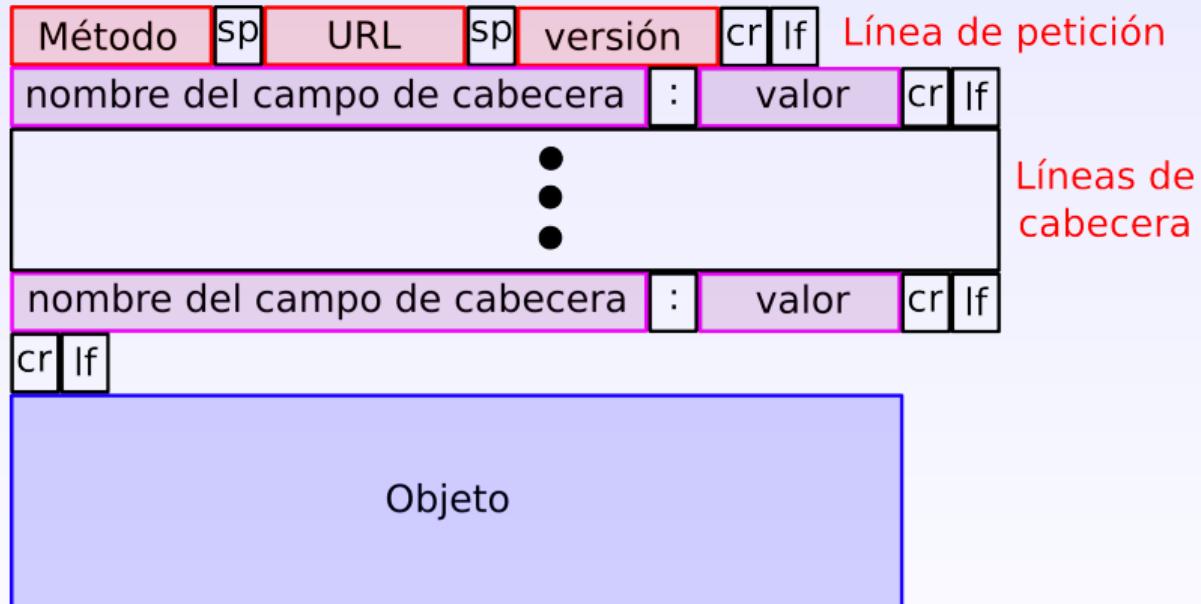
Método
Ejemplo

Respuesta HTTP

Práctica

Cookies

Proxy



Petición HTTP

Métodos

Capa de
aplicación

Introducción

Funcionamiento
HTTP

Petición HTTP

Formato

Métodos

Ejemplo

Respuesta HTTP

Práctica

Cookies

Proxy

Métodos HTTP/1.0

- GET
- POST
- HEAD

Métodos HTTP/1.1

- GET
- POST
- HEAD
- PUT
- DELETE

Método GET

La URL del objeto solicitado está en la cabecera del mensaje

Método POST

- El servidor suele disponer de un formulario de petición
- La petición se realiza según ese formulario como objeto del mensaje

Petición HTTP

Ejemplo

Capa de
aplicación

Introducción

Funcionamiento
HTTP

Petición HTTP

Formato

Métodos

Ejemplo

Respuesta HTTP

Práctica

Cookies

Proxy

Línea de petición → GET /somedir/page.html HTTP/1.1
Líneas de cabecera → Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
Extra CR/LF
indica el fin
del mensaje → (extra carriage return, line feed)

Respuesta HTTP

Formato

Capa de
aplicación

Introducción

Funcionamiento
HTTP

Petición HTTP

Respuesta HTTP

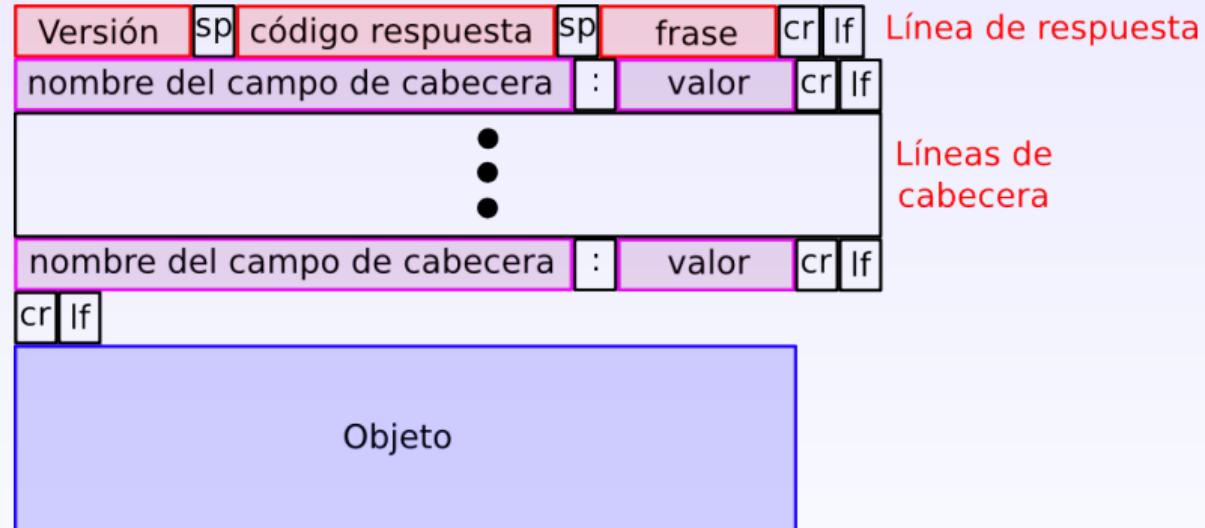
Formato

Tipos

Práctica

Cookies

Proxy



Respuesta HTTP

Tipos

- **200 OK**
 - Petición exitosa
- **301 Moved Permanently**
 - Se movió el objeto pedido. La nueva ubicación es especificada en el mensaje (Location:)
- **400 Bad Request**
 - Petición no entendida por el servidor
- **404 Not Found**
 - Documento no encontrado en el servidor
- **505 HTTP Version Not Supported**
 - Versión de HTTP no soportada

Un poco de práctica

Capa de
aplicación

Introducción

Funcionamiento
HTTP

Petición HTTP

Respuesta HTTP

Práctica

Cookies

Proxy

- Navegación normal por nombre
- Navegación normal por IP
- Descarga con wget
- Descarga con telnet

Cookies

Definición

Capa de
aplicación

Introducción

Funcionamiento
HTTP

Petición HTTP

Respuesta HTTP

Práctica

Cookies

Definición

Ejemplo

Información
transportada

Proxy

Muchos sitios Web importantes usan cookies (RFC 2965)

Componentes

- ① Línea encabezado cookie en el mensaje respuesta HTTP
- ② Línea de encabezado cookie en petición HTTP
- ③ Archivo cookie es almacenado en la máquina del usuario y administrada por su navegador
- ④ Base de datos en sitio Web

Cookies

Ejemplo

Capa de aplicación

Introducción

Funcionamiento
HTTP

Petición HTTP

Respuesta HTTP

Práctica

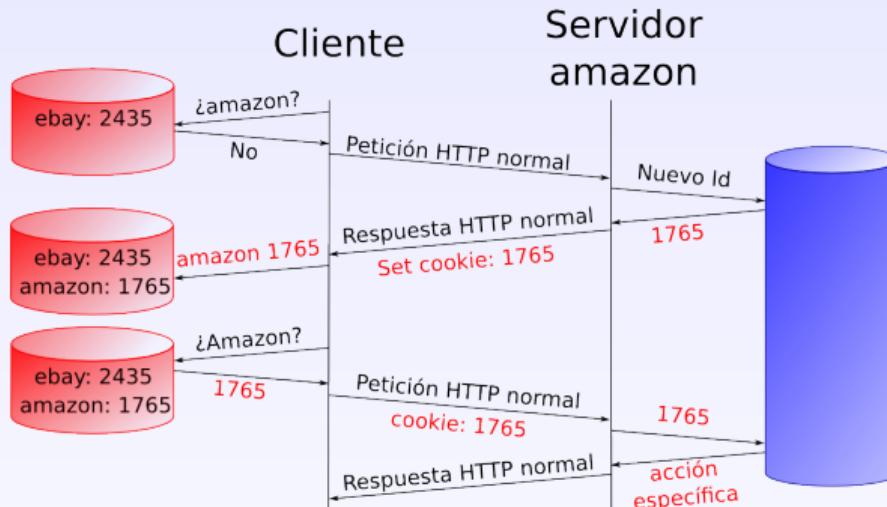
Cookies

Definición

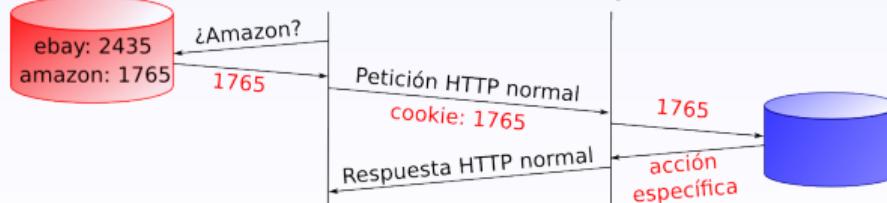
Ejemplo

Información
transportada

Proxy



Una semana después



Cookies

Información transportada

Capa de
aplicación

Introducción

Funcionamiento
HTTP

Petición HTTP

Respuesta HTTP

Práctica

Cookies

Definición

Ejemplo

Información
transportada

Proxy

¿Qué transportan las cookies?

- autorización
- shopping carts
- sugerencias
- Estado de la sesión del usuario (Web e-mail)
- ...

Cookies y privacidad

- Las cookies permiten que el sitio aprenda mucho sobre el usuario
- Se puede proveer nombre y correo al sitio
- Los motores de búsqueda usan redirecciones y cookies para aprender aún más
- Las compañías de avisos obtienen información de los sitios WEB

Web caché (servidores proxy)

Definición

Capa de aplicación

Introducción

Funcionamiento
HTTP

Petición HTTP

Respuesta HTTP

Práctica

Cookies

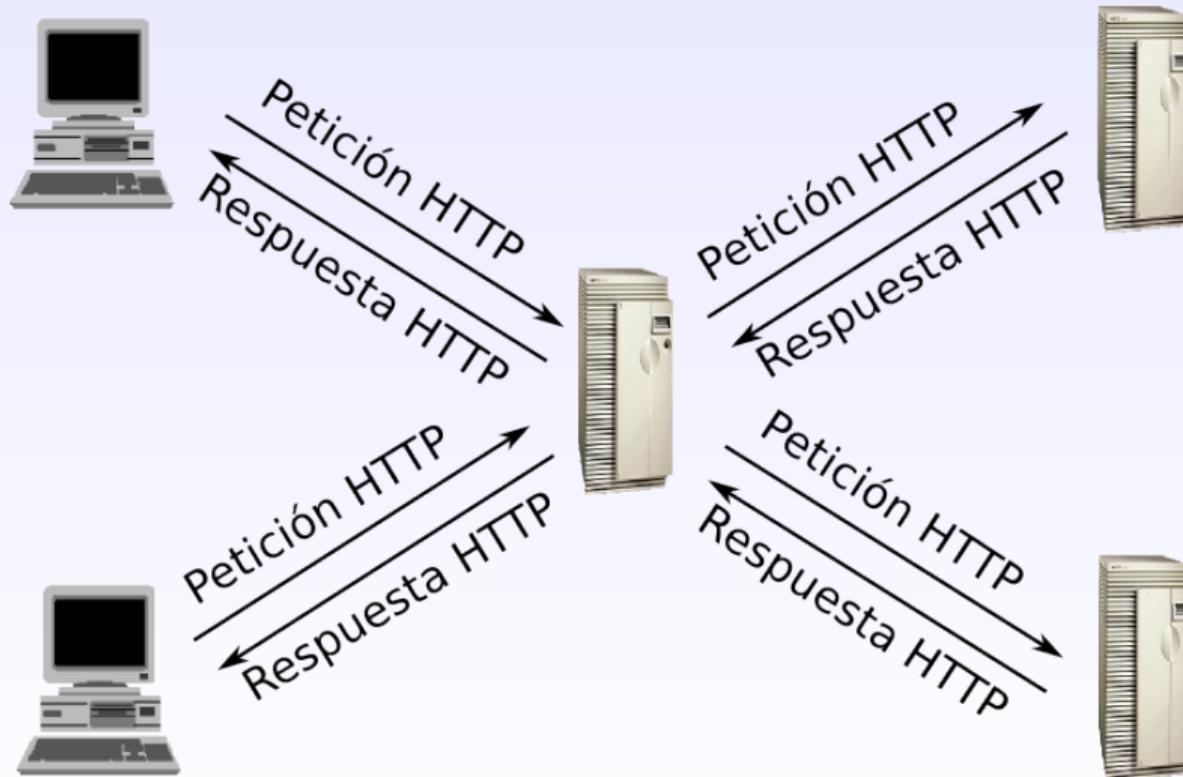
Proxy

Definición

Utilidad

Ejemplo

GET condicional



Web caché (servidores proxy)

Utilidad

Capa de
aplicación

Introducción

Funcionamiento
HTTP

Petición HTTP

Respuesta HTTP

Práctica

Cookies

Proxy

Definición

Utilidad

Ejemplo

GET condicional

- Cache actúan como clientes y servidores
- Típicamente el cache está instalado en el ISP (universidad, compañía, ISP residencial ...)

¿Por qué Web caching?

- Reduce tiempo de respuesta de las peticiones del cliente
- Reduce tráfico de los enlaces Internet de la institución
- Con caches la información en Internet es más densa y permite a proveedores de contenido “pobres” entregar contenido en forma eficiente

Web caché (servidores proxy)

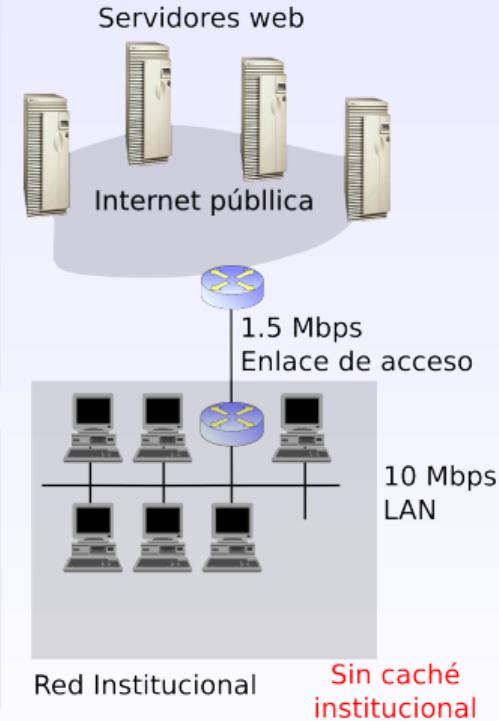
Ejemplo

Características

- Tamaño promedio de objetos = 100,000 bits
- Tasa de requerimientos promedio desde browsers de la institución al servidor WEB = 15/s
- Retardo desde el router institucional a cualquier servidor web y su retorno = 2 s

Consecuencias

- utilización de la LAN = 15 %
- utilización del enlace de acceso = 100 %
- Retardo total = retardo Internet + retardo de acceso + retardo LAN = 2 s + minutos + ms



Web caché (servidores proxy)

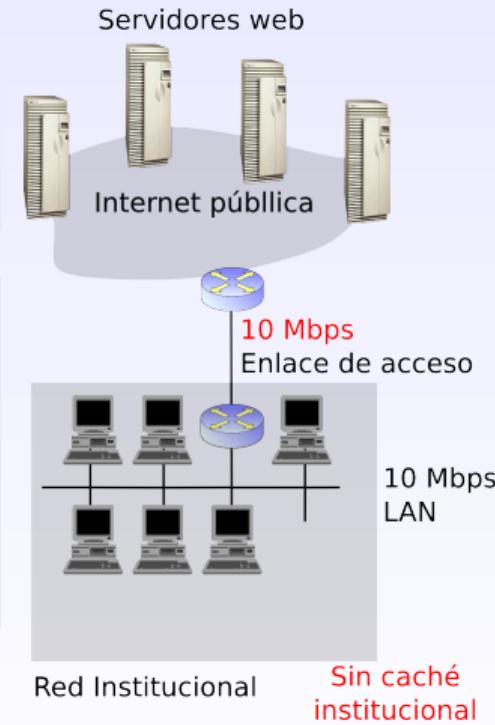
Ejemplo

Possible solución

- Aumentar ancho de banda del enlace a, por ejemplo, 10 Mbps

Consecuencias

- Utilización de la LAN = 15 %
- Utilización del enlace de acceso = 15 %
- Retardo Total = Retardo Internet + retardo de acceso + retardo LAN = 2 s + ms + ms
- A menudo un upgrade caro



Web caché (servidores proxy)

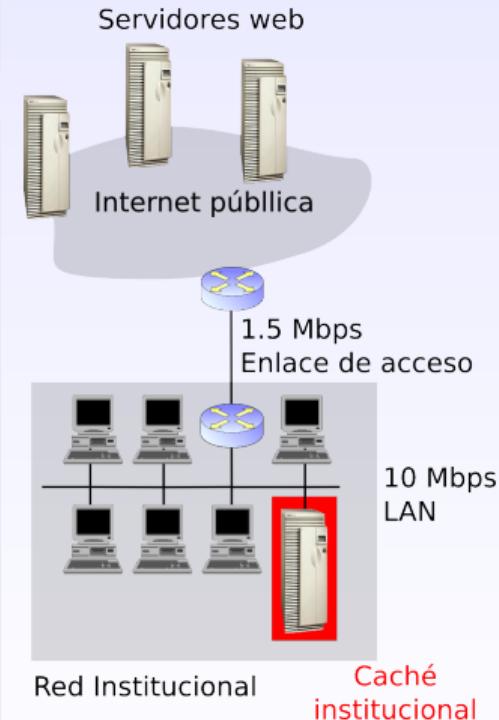
Ejemplo

Instalar un web Cache

- Supongamos tasa de éxito (acierto) de 0.4

Consecuencias

- 40 % de las peticiones serán satisfechos en forma casi inmediata (~ 10 ms)
- 60 % de las peticiones satisfechos por el servidor WEB
- Utilización del enlace de acceso es reducido al 60 %, resultando en retardo despreciable (digamos 10 msec)
- Retardo total = Retardo Internet + retardo acceso + retardo LAN = $0.6 \times 2.01\text{ s} + 0.4 \times 0.01 < 1.3\text{ s}$



Web caché (servidores proxy)

GET condicional

Capa de
aplicación

Introducción

Funcionamiento
HTTP

Petición HTTP

Respuesta HTTP

Práctica

Cookies

Proxy

Definición

Utilidad

Ejemplo

GET condicional

No enviar un objeto si el Web-Caché está actualizado

cache : especifica la fecha de la copia
que se está solicitando

- If-modified-since: <date>

servidor : La respuesta del servidor no
contiene el objeto si tiene una
copia actualizada

- HTTP/1.0 304 Not Modified



Capa de
aplicación

Características

Arquitectura

Formato de
transferencia

Comandos FTP

Códigos de
respuesta

Apertura de
conexión

Ejemplo de
utilización

Comando PASV

Servicio de
reinicio

Parte IV

FTP (File Transfer Protocol)

Características

- FTP transfiere ficheros entre dos sistemas (diferente de acceso transparente de ficheros NFS).
- Fue diseñado para trabajar entre diferentes ...
 - ... sistemas operativos.
 - ... estructuras de ficheros.
 - ... representaciones de texto y datos en ficheros.
 - ... restricciones de acceso a archivos.
 - ... reglas para recorrer los directorios.
- Servicio publico usando como user anonymous.
- Servicio privado usando user y passwd.

Arquitectura

Capa de aplicación

Características

Arquitectura

Descripción

Formato de transferencia

Comandos FTP

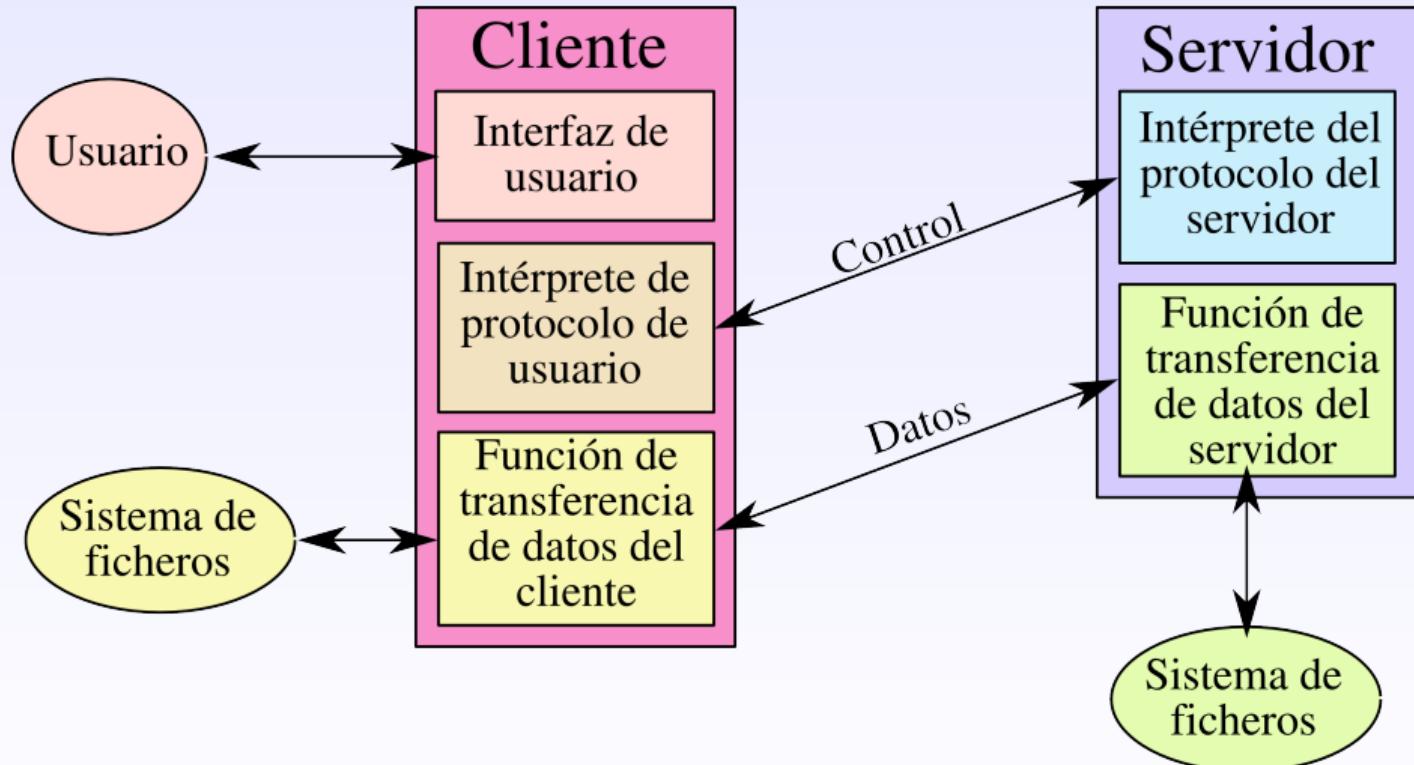
Códigos de respuesta

Apertura de conexión

Ejemplo de utilización

Comando PASV

Servicio de reinicio



Arquitectura

Descripción

Capa de
aplicación

Características

Arquitectura
Descripción

Formato de
transferencia

Comandos FTP

Códigos de
respuesta

Apertura de
conexión

Ejemplo de
utilización

Comando PASV

Servicio de
reinicio

- El cliente esta conectado con el servidor mediante una conexión de control.
- El cliente envía comandos que son contestados por el servidor. Esta conexión utiliza el protocolo NVT de telnet. Estos comandos permiten cambiar, crear, borrar directorios, renombrar y borrar archivos, Indicar modo de trasferencia de los archivos.
- Si el cliente pide un directorio o una trasferencia de archivo, se establece una nueva conexión de datos por la que se envía dicho contenido. Finalizada la trasferencia, la conexión de datos se cierra.

Formato de transferencia

Representación de Datos

Capa de
aplicación

Características

Arquitectura

Formato de
transferencia

Representación de
Datos

Ficheros y modos

Comandos FTP

Códigos de
respuesta

Apertura de
conexión

Ejemplo de
utilización

Comando PASV

Servicio de
reinicio

Representación de Datos

ASCII y EBDIC: pueden ser de varios tipos:

No print: sin información vertical.

Formato Telnet: tiene controles de formato vertical NVT.

CR FORTRAN: el primer carácter de cada línea es el del formato de control FORTRAN.

IMAGE: formato binario.

LOCAL: binario pero entre hosts con diferente tamaño de byte.

Formato de transferencia

Ficheros y modos

Estructura de Archivos

Estructura de fichero (**defecto**): se considera una cadena de bytes.

Estructura de registro: archivo compuesto por una secuencia de registros. Sólo con ficheros texto ASCII o EBCDIC.

Estructura de página: cada página se transmite con un número que es almacenado por el receptor (S.O. TOPS-20 de DEC, obsoleto).

Modos de Transmisión

Modo stream: los datos se envían tal como están en el fichero.

Modo bloque: se incluyen cabeceras con cada bloque.

Modo comprimido: se realizan operaciones de compresión y descompresión de los datos antes y después de la trasferencia. Poco utilizado por ineficaz.

Comandos FTP

Descripción

Capa de
aplicación

Características

Arquitectura

Formato de
transferencia

Comandos FTP

Descripción

Códigos de
respuesta

Apertura de
conexión

Ejemplo de
utilización

Comando PASV

Servicio de
reinicio

ABOR	Interrumpe el comando anterior
LIST <lista>	Lista ficheros o directorios
USER <username>	Nombre del usuario
PASS <passwd>	Palabra de paso en el servidor
TYPE <tipo de fichero>	Tipo de fichero a transferir
MODE <formato trasmis.>	S (flujo), B (bloque), C (comprimido)
STRU <org. del archivo>	F (fichero), R (registro)
QUIT	Desconexión del servidor
SYST	Tipo del sistema del servidor
REINT	Reinicializa al estado de comienzo
PORT n1,n2,n3,n4,n5,n6	Dirección IP del cliente (n1,n2,n3,n4) y puerto (n5x256+n6)
CWD <directorio>	Cambiar de directorio
PWD	Muestra directorio actual
RMD <directorio>	Borra directorio
MKD <directorio>	Crear directorio
LIST <directorio>	Listar los archivos
DELE <fichero>	Borrar archivo
RETR <fichero>	Traer un fichero desde el servidor
STOR <fichero>	Almacena un archivo en el servidor
APPE <fichero>	Añade a un archivo remoto
STAT	Información sobre la conexión

Códigos de respuesta

Códigos

Capa de
aplicación

Características

Arquitectura

Formato de
transferencia

Comandos FTP

Códigos de
respuesta

Códigos

Apertura de
conexión

Ejemplo de
utilización

Comando PASV

Servicio de
reinicio

3 dígitos seguidos de un mensaje opcional

Código de Respuesta	Descripción
1XX	Inicio de una acción con éxito.
2XX	Comando realizado con éxito.
3XX	Respuesta intermedia con éxito.
4XX	Errores pasajeros.
5XX	Errores permanentes.

Ejemplos

FTP foreignhost

220 service ready

USERNAME cmsp1

331 user name okay

PASSWORD xyxyx

230 user logged in

Capa de
aplicación

Características

Arquitectura

Formato de
transferencia

Comandos FTP

Códigos de
respuesta

Apertura de
conexión
descripción

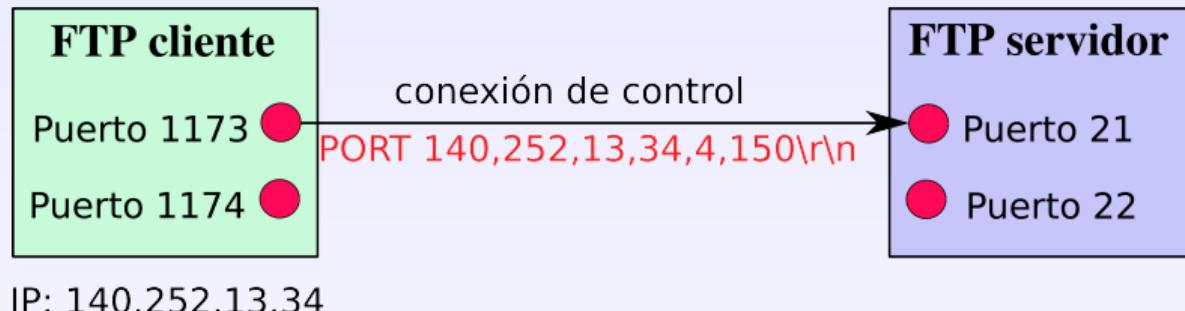
Ejemplo de
utilización

Comando PASV

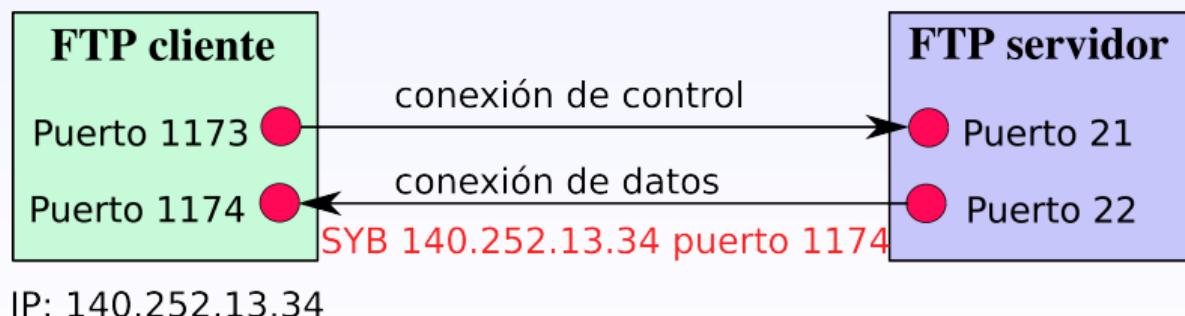
Servicio de
reinicio

Apertura de conexión descripción

Se comunica que el cliente aceptará una conexión por el puerto 1174.



Se realiza la conexión de datos por parte del servidor





Ejemplo de utilización

Conexión de control

Capa de aplicación

Características

Arquitectura

Formato de transferencia

Comandos FTP

Códigos de respuesta

Apertura de conexión

Ejemplo de utilización

Conexión de control
Instrucciones en la conexión de control

Transferencia de datos

Comando PASV

Servicio de reinicio

```
letona% ftp -d afrodita
connected to afrodita
220 afrodita FTP server (Version 5.60) ready
Name (afrodita:anguiano):
    El usuario pulsa retorno
        - - - > USER anguiano
331 Password required for afrodita.
Passwd:
    El usuario teclea su clave
        - - - > PASS XXXXXXXXX
```

Ejemplo de utilización

Conexión de control

Capa de aplicación

Características

Arquitectura

Formato de transferencia

Comandos FTP

Códigos de respuesta

Apertura de conexión

Ejemplo de utilización

Conexión de control
Instrucciones en la conexión de control

Transferencia de datos

Comando PASV

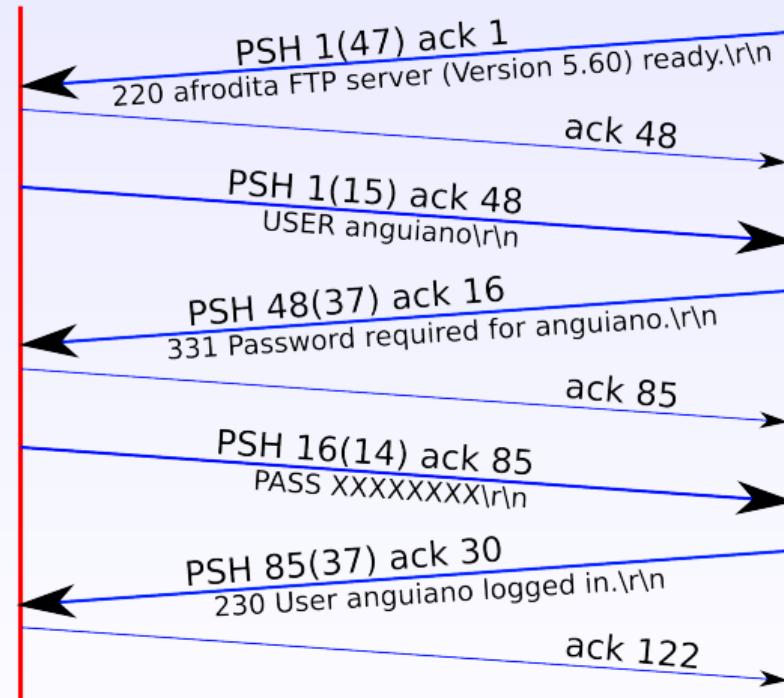
Servicio de reinicio

El usuario introduce el login y pulsa retorno

El usuario introduce el password y pulsa retorno

letona.1173

lafroditा.21



Ejemplo de utilización

Intrucciones en la conexión de control

Capa de
aplicación

Características

Arquitectura

Formato de
transferencia

Comandos FTP

Códigos de
respuesta

Apertura de
conexión

Ejemplo de
utilización

Conexión de control

Intrucciones en la
conexión de control

Transferencia de datos

Comando PASV

Servicio de
reinicio

```
ftp> dir hello.c
-> PORT 140,252,13,34,4,150
200 PORT command successful.
-->LIST hello.c
150 Opening ASCII mode data connection for /bin/ls.
-rw-r-r- 1 rsteven staff 38 Jul 17 12:47 hello.c
226 Transfer complete.
Remote: hello.c
56 bytes received in 0.03 seconds (1.8 Kbytes/s)
ftp> quit
--> QUIT
221 Goodbye
```

Ejemplo de utilización

InSTRUCCIONES en la conexión de control

Capa de aplicación

Características

Arquitectura

Formato de transferencia

Comandos FTP

Códigos de respuesta

Apertura de conexión

Ejemplo de utilización

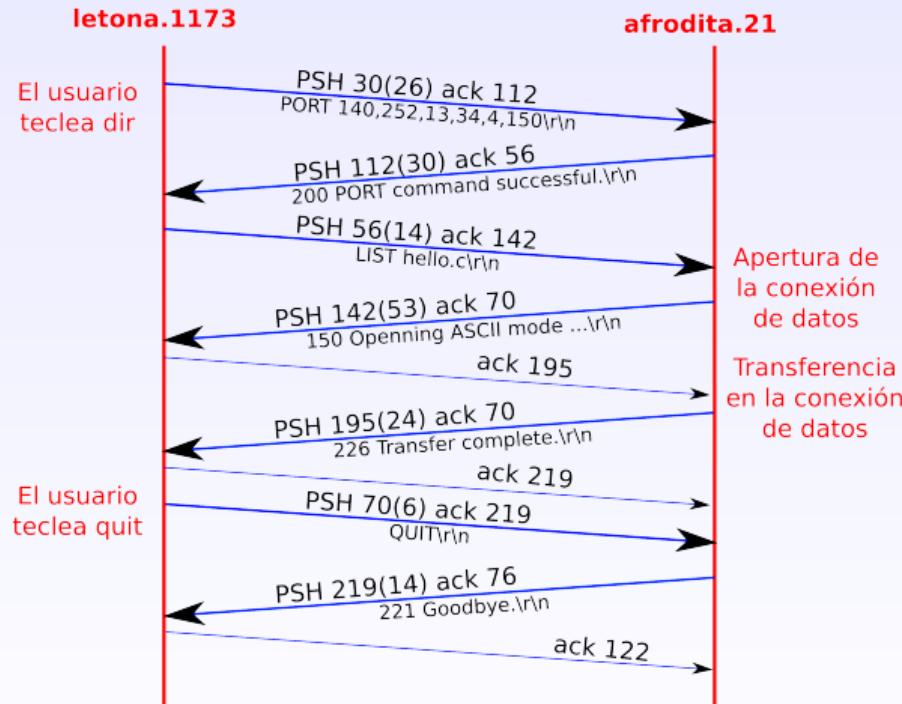
Conexión de control

InSTRUCCIONES en la conexión de control

Transferencia de datos

Comando PASV

Servicio de reinicio



Ejemplo de utilización

Transferencia de datos

Capa de aplicación

Características

Arquitectura

Formato de transferencia

Comandos FTP

Códigos de respuesta

Apertura de conexión

Ejemplo de utilización

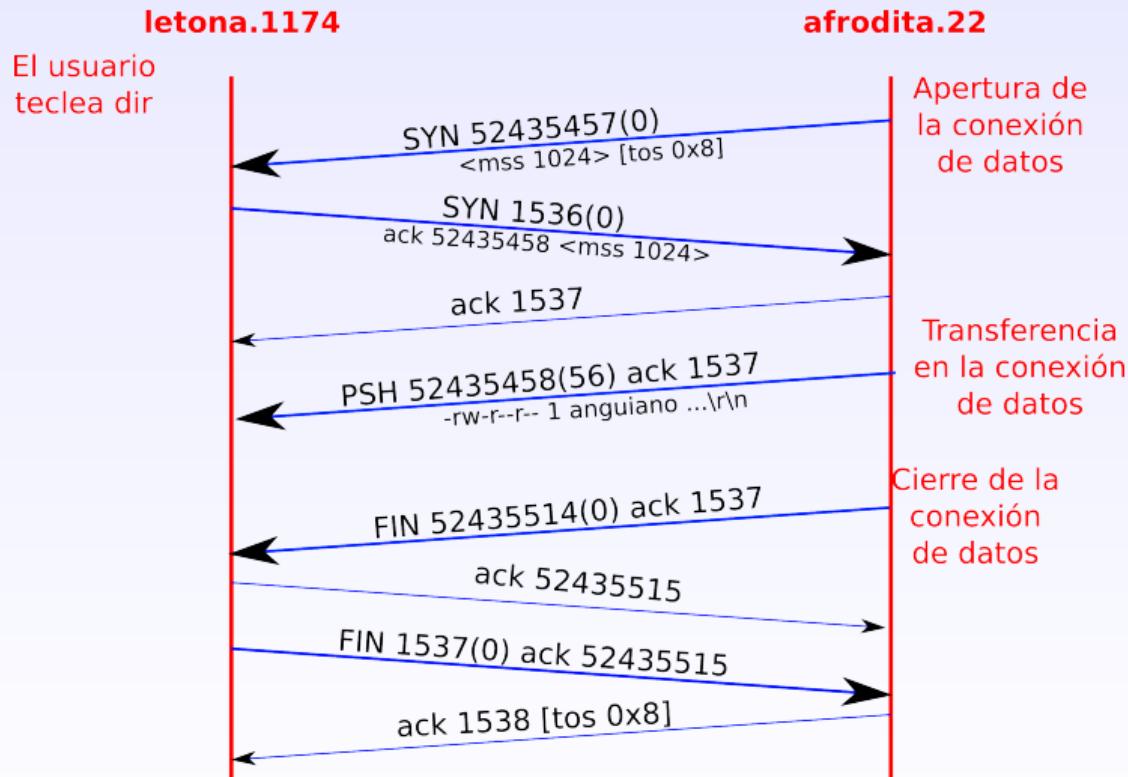
Conexión de control

Instrucciones en la conexión de control

Transferencia de datos

Comando PASV

Servicio de reinicio



Comando PASV

Descripción

Capa de
aplicación

Características

Arquitectura

Formato de
transferencia

Comandos FTP

Códigos de
respuesta

Apertura de
conexión

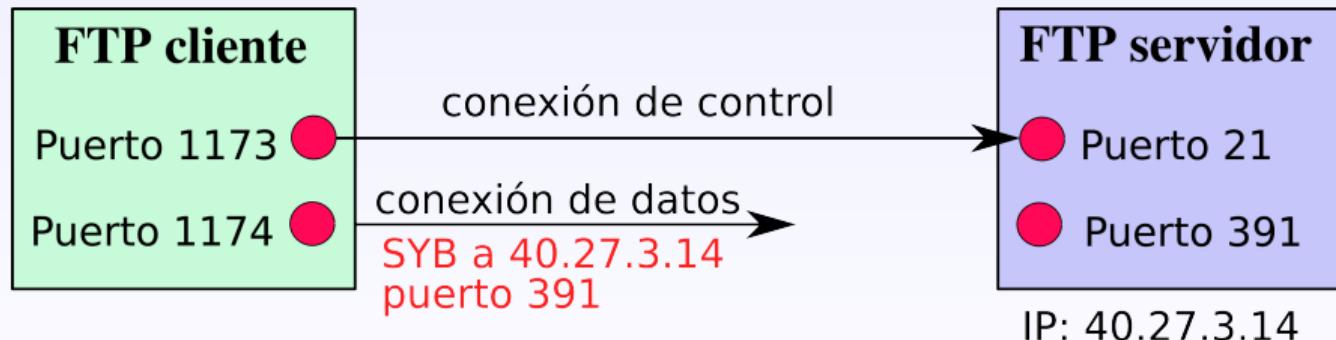
Ejemplo de
utilización

Comando PASV

Descripción

Servicio de
reinicio

- Por seguridad , no es conveniente que un cliente ftp acepte las solicitudes de conexiones de datos.
- En vez de utilizar PORT, el cliente envía un comando PASV al servidor, que le contesta con un puerto del servidor para la conexión de datos.



Servicio de reinicio

Descripción

Capa de
aplicación

Características

Arquitectura

Formato de
transferencia

Comandos FTP

Códigos de
respuesta

Apertura de
conexión

Ejemplo de
utilización

Comando PASV

Servicio de
reinicio

Descripción

- Sólo algunos servidores de FTP implementan este servicio.
- El FTP emisor trasmite bloques con marcadores de reinicio en puntos adecuados de la trasferencia de datos. Cada marcador es una cadena de texto.
- Cuando el receptor recibe un marcador, almacena los datos en disco con la posición del marcador.
- Si el receptor es el cliente, notifica la recepción del marcador a la aplicación FTP.
- Si el receptor es el servidor, se notifica al cliente por la conexión de control la recepción del marcador.
- Si se produce un fallo, se puede reiniciar indicando el valor del marcador adecuado.



Capa de
aplicación

Componentes

Comparación con
HTTP

Prot. correo

MIME

Parte V

SMTP (Simple Mail Transfer Protocol)

Componentes

Capa de
aplicación

Componentes

Arquitectura

Interacción con SMTP

Comparación con
HTTP

Prot. correo

MIME

Los tres componentes principales son:

- Agente de usuario
- Servidor de correo
- Protocolo SMTP

Componentes

Agente de usuario

- También conocido como “lector de correo”
- Escritura, edición, lectura de mensajes de correos
- Mensajes de salida y entrada son almacenados en servidor
- Thunderbird, Eudora, Outlook, Evolution, Netscape Messenger ...

Servidor de correo

- **Casillero** contiene mensajes de entrada para el usuario
- **Cola de mensajes** de los correos de salida

Componentes

Capa de
aplicación

Componentes

Arquitectura

Interacción con SMTP

Comparación con
HTTP

Prot. correo

MIME

SMTP (RFC 2821)

- Usa TCP para transferir confiablemente mensajes e-mail desde el cliente al servidor, **puerto 25**.
- Transferencia directa: servidor envía correos al servidor receptor
- Tres fases en la transferencia
 - Handshaking
 - Transferencia de mensajes
 - Cierre
- Interacción comandos/respuestas
 - **Comandos**: Texto ASCII
 - **Respuesta**: código de estatus y frase
- Mensajes deben ser enviados en ASCII de 7-bits

Componentes

Arquitectura

Capa de
aplicación

Componentes

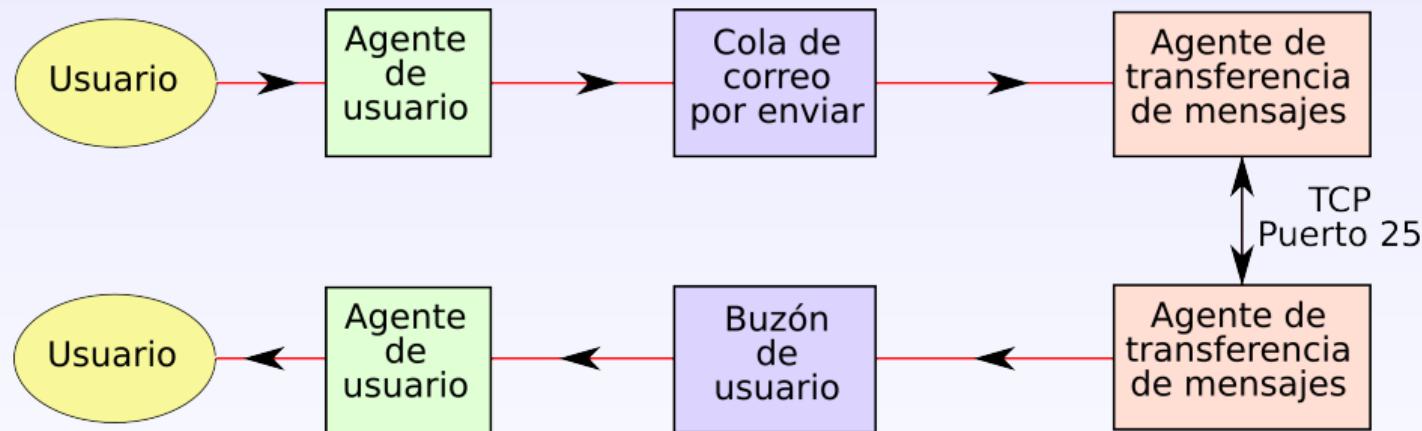
Arquitectura

Interacción con SMTP

Comparación con
HTTP

Prot. correo

MIME



Componentes

Interacción con SMTP

Capa de
aplicación

telnet servername 25

- Esperar respuesta 220 desde el servidor
- Usar los comandos HELO, MAIL FROM, RCPT TO, DATA, QUIT. (El detalle de cada uno de los encabezados se encuentran especificados en el RFC 822)
- Estos comandos nos permite enviar correo sin usar el cliente de correo
- SMTP requiere que el mensaje (encabezado y cuerpo) sean en ASCII de 7-bits
- En servidor SMTP necesita CR/LF.CR/LF para terminar el mensaje

Componentes

Arquitectura

Interacción con SMTP

Comparación con
HTTP

Prot. correo

MIME

Comparación con HTTP

Capa de
aplicación

Componentes

Comparación con
HTTP

Prot. correo

MIME

HTTP pull (saca contenido desde servidor)

SMTP push (pone contenido en servidor)

HTTP cada objeto es encapsulado en su propio mensaje

SMTP múltiples objetos son enviados en un mensaje multiparte

Ambos tienen interacción comando/respuesta en ASCII, y tienen códigos de estatus

Prot. de recepción correo electrónico

Capa de
aplicación

Componentes

Comparación con
HTTP

Prot. correo

Protocolo POP3

Protocolo IMAP

MIME

POP: Post Office Protocol (RFC 1939)

- Autorización, Transacción y Actualización (**puerto 110**)

IMAP: Internet Mail Access Protocol (RFC 350)

- Permite manipulación de los mensajes almacenados en el servidor (**puerto 143**)

HTTP: Analiza el servidor localmente y presenta los resultados al usuario vía WEB.

Todos ellos permiten el uso de la capa SSL (Secure Sockets Layer) usando otros puertos.

Prot. de recepción correo electrónico

Protocolo POP3

Capa de
aplicación

Componentes

Comparación con
HTTP

Prot. correo

Protocolo POP3

Protocolo IMAP

MIME

Fase de autorización

- Comandos del cliente

`user` : declara username

`pass` : password

- Respuestas del servidor

`+OK` :

`-ERR` :

Fase de transacción

`list` : lista números de mensajes

`retr` : extrae mensajes por su número

`delete` : borra

`quit` : termina

Prot. de recepción correo electrónico

Protocolo POP3

Capa de
aplicación

Componentes

Comparación con
HTTP

Prot. correo

Protocolo POP3
Protocolo IMAP

MIME

Observaciones

- En el ejemplo previo usa modo “bajar y borrar”
- Luis no puede releer el correo si cambia el cliente
- “bajar y conservar”: mantiene copia de los mensajes en diferentes clientes
- POP3 no mantiene el estado de una sesión a otra (“stateless”)

Prot. de recepción correo electrónico

Protocolo POP3

Capa de
aplicación

Componentes

Comparación con
HTTP

Prot. correo
Protocolo POP3

Protocolo IMAP

MIME

Fase de autorización

C : +OK POP3 server ready

C : user bob

S : +OK

C : pass hungry

S : +OK user successfully logged on

Prot. de recepción correo electrónico

Protocolo POP3

Capa de
aplicación

Componentes

Comparación con
HTTP

Prot. correo

Protocolo POP3

Protocolo IMAP

MIME

Fase de transacción

```
C : list
S : 1 498
S : 2 912
S :
C : retr 1
S : <message 1 contents>
S :
C : dele 1
C : retr 2
S : <message 1 contents>
S :
C : dele 2
C : quit
S : +OK POP3 server signing off
```

Prot. de recepción correo electrónico

Protocolo IMAP

Capa de
aplicación

Componentes

Comparación con
HTTP

Prot. correo

Protocolo POP3

Protocolo IMAP

MIME

- Mantiene todos los mensajes en el servidor
- Permite que el usuario organice sus correos en carpetas
- Permite tener acceso al correo electrónico desde cualquier equipo que tenga una conexión a Internet
- Permite visualizar los mensajes de manera remota sin la necesidad de descargar los mensajes
- IMAP mantiene el estado del usuario de una sesión a otra:
 - Nombre de carpetas mapeo entre Ids (identificadores) de mensajes y nombres de carpetas

MIME

Descripción

Capa de
aplicación

Componentes

Comparación con
HTTP

Prot. correo

MIME

Descripción

Añade al SMTP las siguientes mejoras:

- Protocolos para incluir objetos diferentes a texto US ASCII.
- Se definen *media types* en la RFC 2046.
- Protocolo de codificación de texto diferente del US-ASCII en las cabeceras o en el cuerpo del correo.



Capa de
aplicación

Características
Organización
Form. protocolo
Algo de práctica

Parte VI

DNS (Domain Name System)

Características de DNS

Capa de
aplicación

Características

Organización

Form. protocolo

Algo de práctica

- Utilizado fundamentalmente por aplicaciones TCP-IP para obtener la dirección IP a partir del nombre del *host*.
- La aplicación cliente encargada de la resolución de nombres se denomina resolver. Se implementa mediante las funciones de librería `gethostbyname` y `gethostbyaddress`.
- El resolver obtiene la información:
 - Consultando un fichero *hosts* en la maquina.
 - Contactando con un servidor de nombres.
- Este servicio opera tanto sobre UDP como TCP por el puerto 53.

Características de DNS

- Una organización debe disponer de un Servidor local de Nombres donde se encuentra la base de datos con los nombres de las maquinas y sus direcciones IP.
- Cualquier consulta desde un cliente de la organización sobre una maquina interna será resuelta en el servidor local.
- Cualquier consulta desde un cliente de la organización sobre una maquina externa será resuelta por el servidor local que contactará con el servidor del *host* externo. ¿Cómo encontrar dicho servidor?. La respuesta será enviada al cliente y almacenada en la caché del servidor local.
- El servidor antes de contactar con un servidor externo, consulta si tiene dicha información en la caché.

Organización del DNS

Estructura del DNS

Capa de aplicación

Características

Organización

Estructura

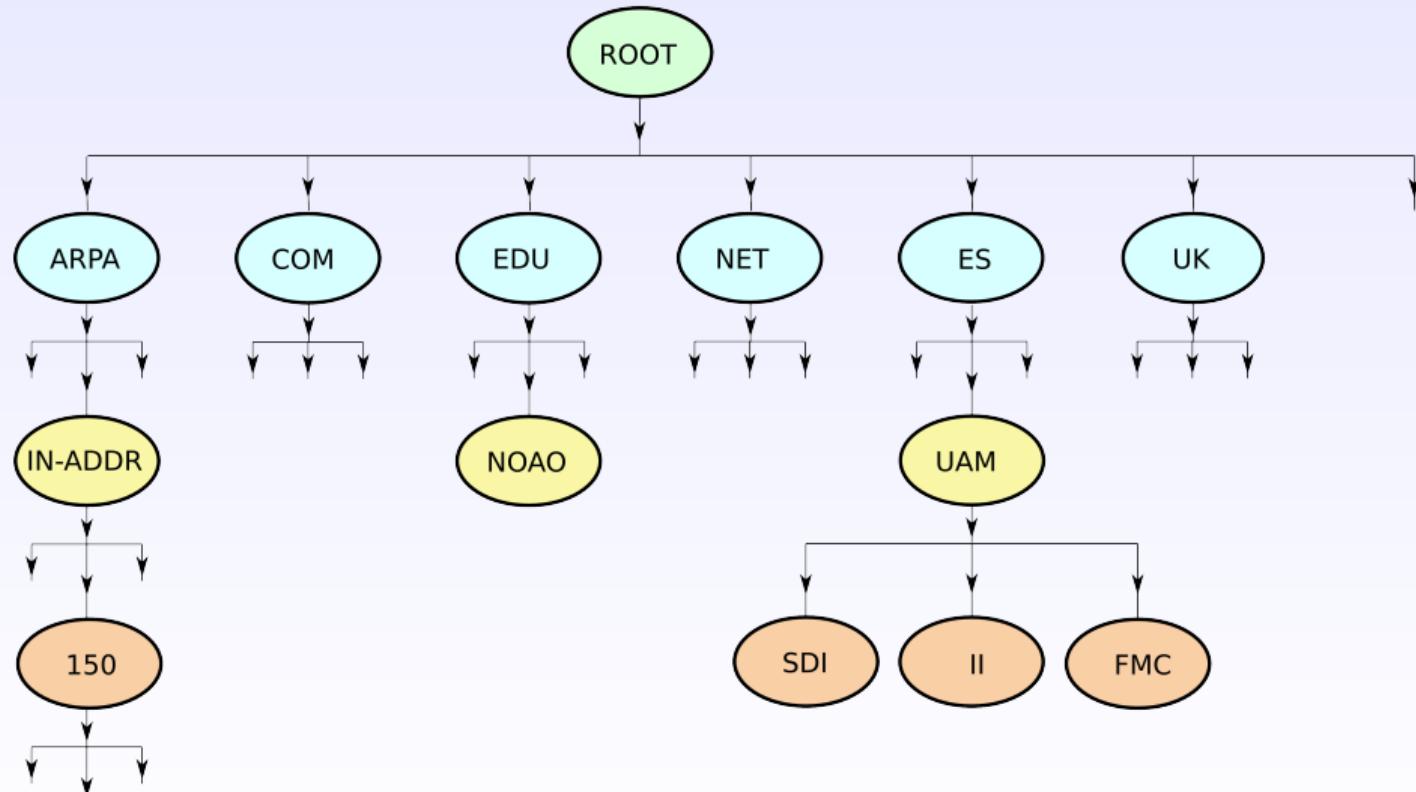
Serv. raíz

Operación

Op. Alternativa

Form. protocolo

Algo de práctica



Organización del DNS

Estructura del DNS

Capa de
aplicación

Características

Organización

Estructura

Serv. raíz

Operación

Op. Alternativa

Form. protocolo

Algo de práctica

- Estructura jerárquica en forma de árbol.
- Cada nodo tiene asignada una etiqueta (máximo 63 caracteres) excepto el root.
- El nombre del dominio se forma añadiendo las etiquetas separadas por un punto, desde un nodo terminal hasta el root. Ejemplo ii.uam.es
- El FQDN (full qualified domain name) de un host se forma añadiendo el nombre de la maquina al dominio al que pertenece. Ejemplo debod.ii.uam.es
- En el nivel superior del árbol se encuentran:

Arpa: es un dominio utilizado para resolver direcciones a nombres

Dominios genéricos asociados a actividades: com, edu, gov, net, org, mil.

Dominios geográficos de 2 caracteres: es, it, us, fr.

- Los dominios de nivel superior son administrados por el NIC.

Organización del DNS

Servidores raíz

Capa de
aplicación

Características

Organización

Estructura

Serv. raíz

Operación

Op. Alternativa

Form. protocolo

Algo de práctica

/netinfo/root-servers.txt Sep 97

The following hosts are functioning as root domain name servers for the Internet

HOSTNAME	NET ADDRESSES	SERVER PROGRAM
A.ROOT-SERVERS.NET	198.41.0.4	BIND(UNIX)
B.ROOT-SERVERS.NET	128.9.0.107	BIND(UNIX)
C.ROOT-SERVERS.NET	192.33.4.12	BIND(UNIX)
D.ROOT-SERVERS.NET	128.8.10.90	BIND(UNIX)
E.ROOT-SERVERS.NET	192.203.230.10	BIND(UNIX)
F.ROOT-SERVERS.NET	192.5.5.241	BIND(UNIX)
G.ROOT-SERVERS.NET	192.112.36.4	BIND(UNIX)
H.ROOT-SERVERS.NET	128.63.2.53	BIND(UNIX)
I.ROOT-SERVERS.NET	192.36.148.17	BIND(UNIX)
J.ROOT-SERVERS.NET	198.41.0.10	BIND(UNIX)
K.ROOT-SERVERS.NET	193.0.14.129	BIND(UNIX)
L.ROOT-SERVERS.NET	198.32.64.12	BIND(UNIX)
M.ROOT-SERVERS.NET	202.12.27.33	BIND(UNIX)

Organización del DNS

Servidores raíz

Capa de
aplicación

Características

Organización

Estructura

Serv. raíz

Operación

Op. Alternativa

Form. protocolo

Algo de práctica

- La administración de una zona, rama del árbol DNS, se delega en una organización. Un servidor de nombres es autoridad para dicha zona, debiendo tener en su base de datos los nombres de máquinas y direcciones IP de la zona. Una zona puede englobar a varios nodos del árbol DNS.
- Además del servidor primario de nombres, pueden existir servidores secundarios que son autoridad para la zona. Su información es redundante y actúan en caso que falle el servidor primario.
- Cada servidor de nombres conoce la dirección IP del servidor de nombres que se encuentra en el nivel superior dentro del árbol de DNS, así como los de nivel inferior.

Organización del DNS

Operación del DNS

Capa de aplicación

Características

Organización

Estructura

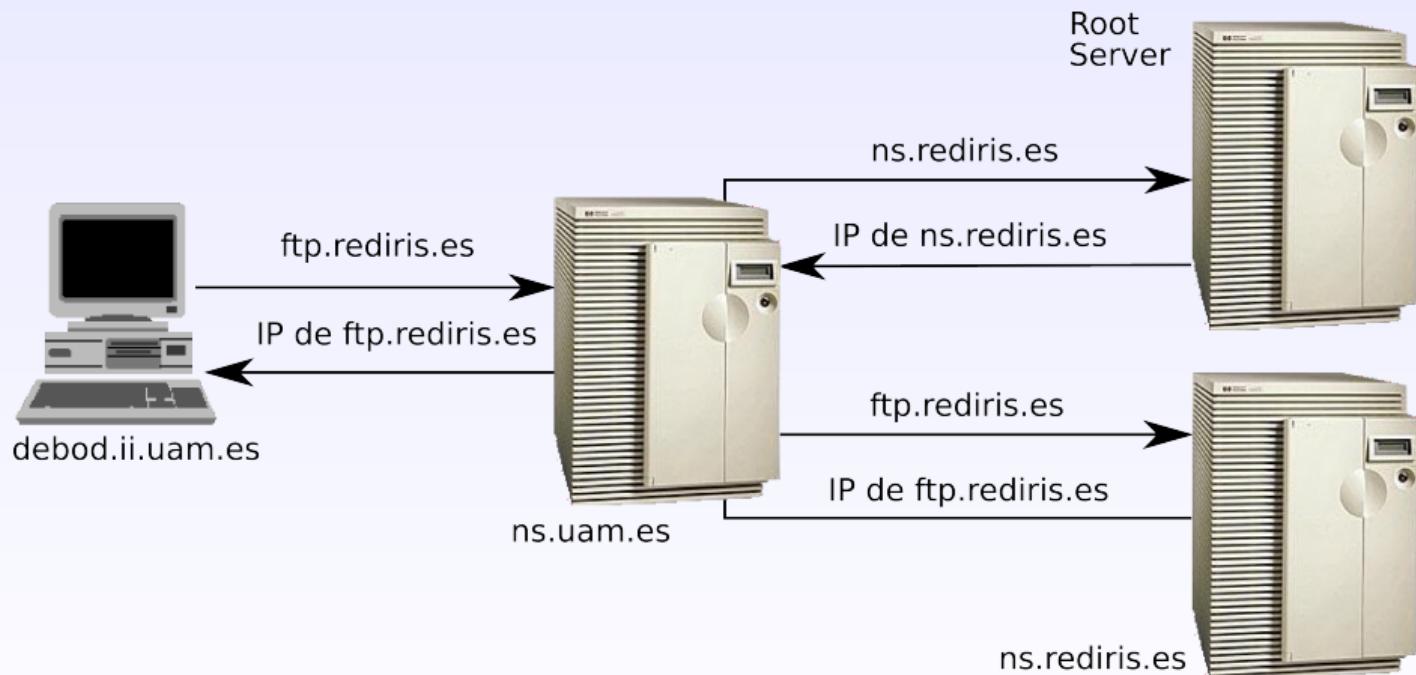
Serv. raíz

Operación

Op. Alternativa

Form. protocolo

Algo de práctica



Organización del DNS

Operación del DNS

Capa de
aplicación

Características

Organización

Estructura

Serv. raíz

Operación

Op. Alternativa

Form. protocolo

Algo de práctica

Resolución de un destino

- ① Hace una solicitud a su servidor de nombres local.
- ② El servidor de nombres local hace una solicitud a un servidor root para conocer el servidor de nombres que tiene autoridad sobre el host destino.
- ③ El servidor de nombres local contacta con el servidor de nombres destino y obtiene la dirección IP del host destino.
- ④ El servidor de nombres almacena esta información en su cache y contesta al host que hizo la solicitud inicial.



Organización del DNS

Operación del DNS

Capa de aplicación

Características

Organización

Estructura

Serv. raíz

Operación

Op. Alternativa

Form. protocolo

Algo de práctica

```
cat /etc/resolv.conf
```

```
search ii.uam.es
```

```
nameserver 150.244.9.100
```

```
nameserver 150.244.9.200
```

```
nslookup gmail.com
```

```
Server: 192.168.0.1
```

```
Address: 192.168.0.1#53
```

```
Non-authoritative answer:
```

```
Name: gmail.com
```

```
Address: 64.233.161.83
```

```
Name: gmail.com
```

```
Address: 64.233.171.83
```

```
Name: gmail.com
```

```
Address: 72.14.253.83
```

Organización del DNS

Operación del DNS alternativa

Capa de aplicación

Características

Organización

Estructura

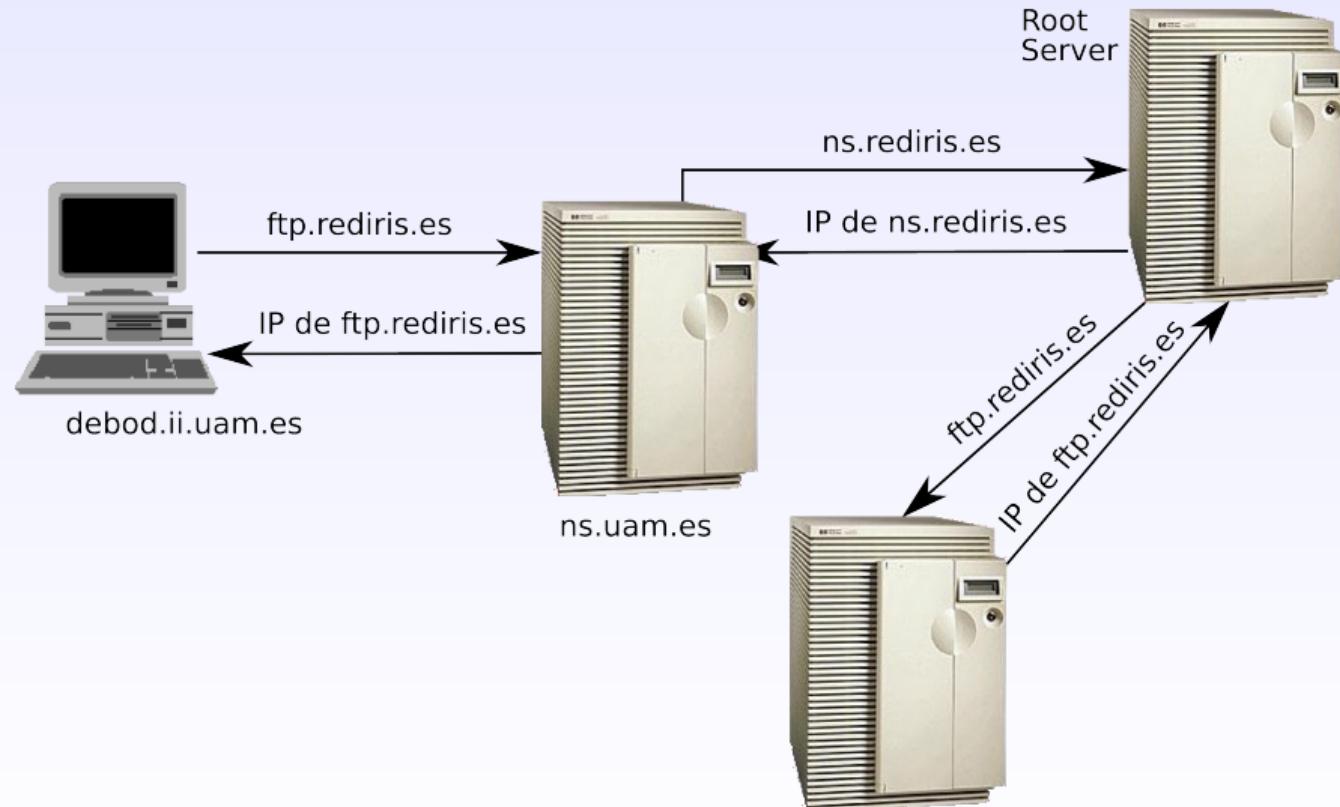
Serv. raíz

Operación

Op. Alternativa

Form. protocolo

Algo de práctica



Organización del DNS

Operación del DNS alternativa

Capa de
aplicación

Características

Organización

Estructura

Serv. raíz

Operación

Op. Alternativa

Form. protocolo

Algo de práctica

Resolución de un destino

- ① Hace una solicitud a su servidor de nombres local.
- ② El servidor de nombres local hace una solicitud a su servidor superior. Así hasta los servidores de dominio de nivel superior. Desde allí se contacta con un root server siguiendo el procedimiento anterior.
- ③ Si el dominio destino esta por debajo de un dominio, procede a contactar con el servidor de dicho dominio.

Formato de protocolo

Descripción

Capa de
aplicación

Características

Organización

Form. protocolo

Descripción

Flags

Pregunta

Respuesta

Pregunta inversa

Validación por
respuesta inversa

Algo de práctica

Descripción

0 1 215	16 31		
Identificación		Flags			
Nº de RR preguntas		Nº de RR respuestas			
Nº de RR de autoridad		Nº de RR adicionales			
Preguntas					
Respuestas					
Autoridad					
Información adicional					

Formato de protocolo

Flags

Descripción

1	4	1	1	1	1	3	4
QR	OPCODE	AA	TC	RD	RA	0 0 0	RCODE

QR: 0 pregunta, 1 respuesta.

OPCODE: 0 pregunta normal, 1 pregunta inversa.

AA: respuesta con autoridad.

TC: respuesta truncada (primeros 512 bytes).

RD: 0 uso iterativo., 1 solicitud de uso recursivo.

RA: 0 uso no permitido, 1 uso permitido.

RCODE: código de retorno.

0: sin error

1: error de formato

2: fallo de servidor

3: *host* inexistente

...

Formato de protocolo

Pregunta

Capa de
aplicación

Características

Organización

Form. protocolo

Descripción

Flags

Pregunta

Respuesta

Pregunta inversa

Validación por
respuesta inversa

Algo de práctica

Pregunta

0	1	215	16 31
Nombre de la pregunta							
Tipo de pregunta				Clase de pregunta			

Formato del nombre

3	w	w	w	2	i	i	3	u	a	m	2	e	s	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Formato de compresión

1	1	offset
---	---	--------

8	a	f	r	o	d	i	t	a	C0	10
---	---	---	---	---	---	---	---	---	----	----

continúa en el offset 16.

Formato de protocolo

Pregunta

Capa de
aplicación

Características
Organización

Form. protocolo

Descripción

Flags

Pregunta

Respuesta

Pregunta inversa

Validación por
respuesta inversa

Algo de práctica

Tipo de pregunta

Tipo	Valor	Significado
A	1	dirección IP
NS	2	servidor de nombre
CNAME	5	nombre canónico (alias)
PTR	12	nombre asignado una dirección IP
HINFO	13	información del host
MX	15	intercambiador de correo
ALL	255	todos los RR disponibles

Clase de pregunta

Valor	Significado
1	Internet

Formato de protocolo

Respuesta

Capa de
aplicación

Características

Organización

Form. protocolo

Descripción

Flags

Pregunta

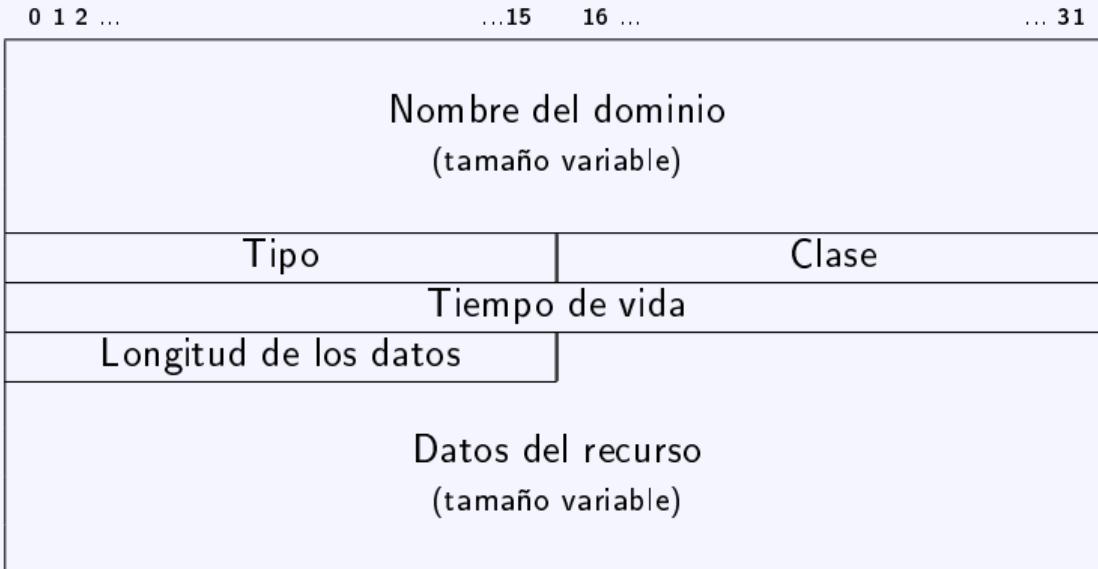
Respuesta

Pregunta inversa

Validación por
respuesta inversa

Algo de práctica

Respuesta



Nombre del dominio: es el nombre al que corresponde los datos del recurso

TTL: segundos durante los cuales el valor del RR es válido

Formato de protocolo

Pregunta inversa

Capa de
aplicación

Características

Organización

Form. protocolo

Descripción

Flags

Pregunta

Respuesta

Pregunta inversa

Validación por
respuesta inversa

Algo de práctica

Características

- Solicitud de un nombre a partir de su dirección IP
- Cada organización debe mantener un servidor de nombres para resolver estas preguntas
- Estos servidores se encuentran en la organización DNS por debajo del dominio arpa.in-addr
- El campo nombre en estas preguntas debe indicarse con la dirección IP en orden inverso. Por ejemplo para conocer el nombre de la maquina 150.244.28.254 debe indicarse: 254.28.244.150.in-addr.arpa

Formato de protocolo

Validación por respuesta inversa

Capa de
aplicación

Características

Organización

Form. protocolo

Descripción

Flags

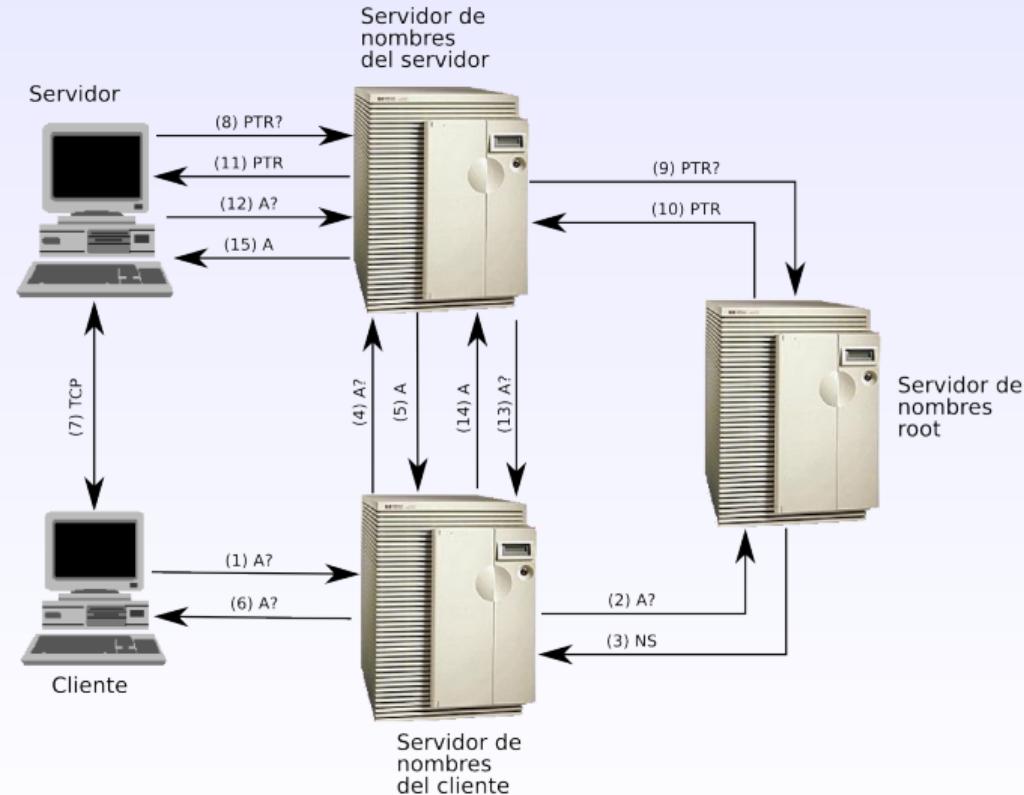
Pregunta

Respuesta

Pregunta inversa

Validación por
respuesta inversa

Algo de práctica



Formato de protocolo

Validación por respuesta inversa

Elementos

- ① El cliente consulta la dirección del ordenador de destino a su servidor de nombres.
- ② El servidor de nombres consulta a servidores superiores.
- ③ Los servidores superiores le devuelven la dirección del servidor de nombres autoridad sobre la dirección a la que se pretende conectar.
- ④ Se realiza la consulta al servidor de nombres identificado.
- ⑤ El servidor de nombres del servidor devuelve la dirección correspondiente.
- ⑥ El servidor de nombres del cliente le devuelve la dirección del ordenador con el que quiere conectar.
- ⑦ Se realiza la conexión sin validación.
- ⑧ Se inicia la validación por parte del servidor realizando la consulta inversa solicitando el nombre de la máquina que se ha conectado.

Formato de protocolo

Validación por respuesta inversa

Elementos

- ⑨ Usando ARPA el servidor raíz devuelve la dirección al servidor de nombres del servidor
- ⑩ El servidor de nombres del servidor le pasa la dirección al servidor.
- ⑪ El servidor le pregunta a su servidor de nombres por la dirección asociada a ese nombre.
- ⑫ Se realiza la consulta al servidor autoridad del cliente. No es necesario consultar al root pues ya se tiene esa dirección en caché de la conexión anterior.
- ⑬ El servidor de nombres autoridad del cliente le devuelve la dirección al servidor de nombres del servidor.
- ⑭ El servidor de nombres del servidor le devuelve la dirección al servidor.
- ⑮ Si esta dirección coincide con la de la comunicación inicial se comprueba que la comunicación es correcta y que no hay suplantación.



Algo de práctica

Capa de aplicación

Características

Organización

Form. protocolo

Algo de práctica

- ➊ nslookup
- ➋ host
- ➌ /etc/hosts



Capa de
aplicación

Características

Mensaje DHCP

Operación DHCP

Agente Router de
Reenvío

Parte VII

DHCP (Dynamic Host Configuration Protocol)

Características de DHCP

Información suministrada por DHCP para configurar TCP-IP

- Dirección IP.
- Máscara de subred.
- Router por defecto.

Ventajas

- Facilita la administración.
- Reduce el número total de direcciones IP.

Inconvenientes

- Información en el DNS no actualizada. Hay que usar Dynamic DNS (DDNS).

Características de DHCP

UDP puerto 67

Capa de
aplicación

Características

UDP puerto 67

Mensaje DHCP

Operación DHCP

Agente Router de
Reenvío

DHCP soporta tres mecanismos para asignar direcciones

Automática: La dirección IP asignada es permanente para el cliente.

Dinámica: La dirección IP asignada esta limitada a un periodo de tiempo.

Manual: La dirección IP se configura manualmente en el cliente.

El protocolo usa UDP puerto 67.

Mensaje DHCP

Formato del mensaje DHCP

Descripción

0 ...	8 ...	16 ...	24 31
OpCode	Tipo HW	Longitud	Saltos	
Identificador				
Tiempo				Flags
Dirección IP cliente				
Dirección IP asignada				
Dirección IP servidor DHCP				
Dirección IP router				
Dirección física cliente (16 bytes)				
Nombre servidor (64 bytes)				
Nombre del Archivo de Arranque (128 bytes)				

Mensaje DHCP

Campos

Capa de
aplicación

Características

Mensaje DHCP

Formato del mensaje
DHCP

Campos

Operación DHCP

Agente Router de
Reenvío

Opcode

DISCOVER: Usado en difusión por el cliente.

OFFER: Respuesta del servidor con la dirección IP y otros parámetros.

REQUEST: Sigue la respuesta del servidor para solicitar los parámetros ofrecidos. También se usa para extender en el tiempo la validez de una dirección IP.

PACK: Reconocimiento por el servidor de los parámetros a utilizar.

NPACK:

: Reconocimiento negativo. No puede seguir usando la dirección IP o es incorrecta.

DECLINE: El cliente comunica al servidor que la dirección IP esta en uso.

RELEASE: El cliente comunica que no seguirá usando la dirección IP asignada.

INFORM: Mensaje enviado por el cliente que ya conoce su dirección IP, para obtener otros parámetros de configuración.

Mensaje DHCP

Campos

Capa de aplicación

Características

Mensaje DHCP

Formato del mensaje
DHCP

Campos

Operación DHCP

Agente Router de
Reenvío

Tipo HW: Ethernet (1). IEEE-802 (6).

Longitud: nº de bytes de la dirección física

Saltos: Puesto por el cliente a 0, se incrementa por cada maquina *relay* que redirige el mensaje

Tiempo: Segundos transcurridos desde que se inició el intento de arranque de la maquina.

Flags: Bit 15. Difusión (1). Se utiliza para indicar al servidor que el cliente solo puede recibir mensajes en difusión (no conoce su IP).

Dirección IP cliente: Puesto por el cliente, puede ser 0.0.0.0 o su dirección IP si la conoce.

Dirección IP asignada: Dada por el servidor.

Dirección IP del router de reenvío: Introducido por una maquina relay DHCP, para que el servidor de DHCP le devuelva la contestación.

Nombre del servidor DHCP: Optativo.

Nombre del archivo de arranque: Optativo. Indicado por el servidor para indicar al cliente un archivo de arranque en el cliente o en un servidor de configuración.

Operación DHCP

Esquema

Capa de
aplicación

Características

Mensaje DHCP

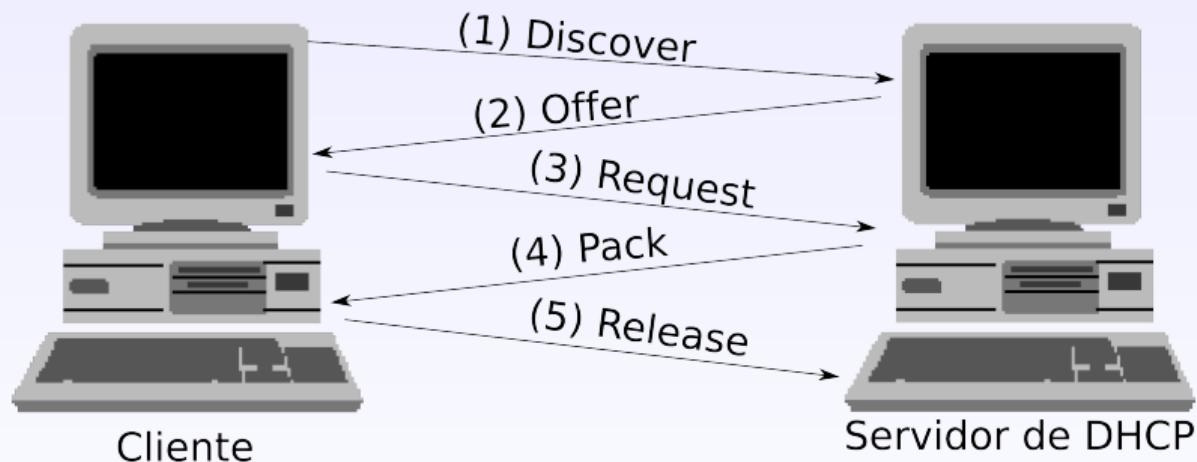
Operación DHCP

Esquema

Modo de operación

Agente Router de
Reenvío

En el protocolo se definen diferentes tipos de mensaje:



Operación DHCP

Modo de operación

Capa de
aplicación

Características

Mensaje DHCP

Operación DHCP

Esquema

Modo de operación

Agente Router de
Reenvío

- ① El cliente manda en difusión un mensaje DISCOVER (IP origen 0.0.0.0, IP destino 255.255.255.255) que puede incluir como opciones una posible dirección IP y el tiempo de validez.
- ② Cada servidor puede responder con un mensaje OFFER que incluye una dirección IP y otras opciones de configuración. Se envía con MAC del cliente y con IP 255.255.255.255. Los servidores recuerdan las direcciones ofertadas, evitando ofrecerlas en posteriores mensajes de DISCOVER.
- ③ El cliente elige una de las direcciones IP y manda en difusión un REQUEST en el que indica el servidor y la dirección IP elegida. La recepción de este mensaje por los servidores no elegidos, sirven de notificación de que el cliente no ha aceptado su oferta. Se pueden incluir parámetros de configuración adicionales.
- ④ El servidor elegido, manda un mensaje PACK confirmando la dirección IP y otros parámetros de configuración, así como dos tiempos T1 y T2 relacionados con el tiempo máximo T en que la dirección IP es valida.

Operación DHCP

Modo de operación

Capa de
aplicación

Características

Mensaje DHCP

Operación DHCP

Esquema

Modo de operación

Agente Router de
Reenvío

- ⑤ Cuando el temporizador T1 expira, el cliente manda un REQUEST (unicast) al servidor solicitando extender en el tiempo la validez de la configuración. Si el servidor esta de acuerdo, manda un mensaje PACK con nuevos valores de T1 y T2.
- ⑥ Si no llega el mensaje PACK, cuando el temporizador T2 expira, el cliente manda un mensaje REQUEST(difusión). Esta solicitud puede ser contestada por alguno de los servidores con un mensaje PACK.
- ⑦ Si el cliente no recibe PACK y vence el tiempo T, el cliente debe dejar de usar la configuración IP actual e iniciar el proceso con un mensaje DISCOVER.
- ⑧ Un cliente puede decidir dejar de usar una configuración IP asignada enviando un mensaje RELEASE al servidor.

Agente Router de Reenvío

Descripción

Capa de
aplicación

Características

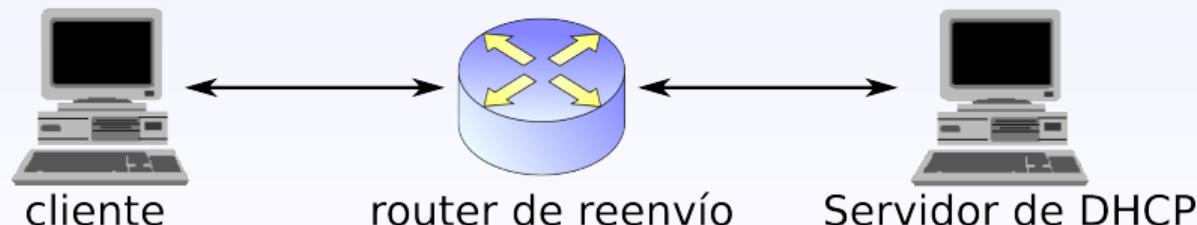
Mensaje DHCP

Operación DHCP

Agente Router de
Reenvío

Descripción

- Se emplea en el caso de que el servidor DHCP no este en la misma subred que el cliente.
- Cuando el agente detecta una solicitud introduce la dirección IP de la red por la que llegó la petición en el campo Router Relay y lo reenvía al servidor remoto DHCP.
- La respuesta desde el servidor se hace a través del agente Router de Reenvío





Capa de
aplicación

Protocolo XMPP
(Jabber)
Tipos de Stanzas

Parte VIII

Protocolo XMPP (Jabber)

Protocolo XMPP (Jabber)

Introducción

Origen

- Protocolo Extensible de Mensajes y Presencia (Extensible Messaging and Presence Protocol)
- Desarrollado por la comunidad de código abierto Jabber para mensajería instantánea, información de presencia y mantenimiento de listas de contactos.
- Basado en el intercambio de ficheros XML.

Diseñado para ser extensible, se ha usado también para

- Sistemas de suscripción-publicación.
- Señales para VoIP
- Video
- Transferencia de archivos
- Juegos

Protocolo XMPP (Jabber)

Introducción

Capa de
aplicación

Protocolo XMPP
(Jabber)

Introducción

Características

Descentralización y
direccionalamiento

Utilizando XMPP para
comunicarse

Ejemplo de
comunicación
Cliente-Servidor

Tipos de Stanzas

El Linux de los mensajes instantáneos

Definido como un protocolo de dominio público

- Esto lo diferencia de la mayoría de los protocolos de mensajería.
- Definido por RFC 3922, 3923, 6120, 6121 y 6122.
- Además la XMPP Standards Fundation continúa desarrollando extensiones abiertas.
- Existe software (servidores y clientes) con distintas licencias: código abierto y gratuito, freeware e incluso comercial.

Protocolo XMPP (Jabber)

Introducción

Utilizado (con variantes) por

- Facebook (ofrece interfaz XMPP a clientes, pero no lo usa internamente)
- Tuenti
- WhatsApp (variante propietaria)
- Nimbuzz
- GoogleTalk (actualmente reemplazado por Hangouts)
- Skype (sólo en la capa de aplicación)

Protocolo XMPP (Jabber)

Características

Ventajas

Descentralización: Cualquiera puede ejecutar un servidor XMPP, no hay un servidor central "maestro"

Abierto: No está atado a un desarrollador ni se deben pagar royalties

Seguridad: Los servidores XMPP se pueden aislar de la red pública si fuera necesario. Además layers de seguridad (SASL y TLS) han sido incorporados en las especificaciones.

Flexibilidad: Se puede programar funcionalidad específica sobre XMPP.

- chat grupales, compartición de ficheros, geolocalización, etc.

Desventajas

Transferencia ineficiente de binarios: Hay que codificarlos mediante Base64. Generalmente se complementa con otros protocolos (P.E: http)

Protocolo XMPP (Jabber)

Descentralización y direccionamiento

Arquitectura

- Utiliza arquitectura **Cliente/Servidor**
- Aunque basado en servidores, el modelo es **descentralizado** (no hay servidor principal)
- No es P2P ni anónimo

Direccionamiento

- Cada usuario recibe un identificador (JID): nombre y el servidor en el que está registrado (`nombreusuario@dominio.com`).
- Además se puede conectar desde múltiples sitios o dispositivos (llamados **recursos**):
 - Se incluye al final de JID. PE: `nombreusuario@dominio.com-mobile`.
 - Cada recurso tiene una prioridad asociada. Si no se especifica un recurso, el mensaje va al que tenga mayor prioridad.

Protocolo XMPP (Jabber)

Descentralización y direccionamiento

Capa de aplicación

Protocolo XMPP
(Jabber)

Introducción

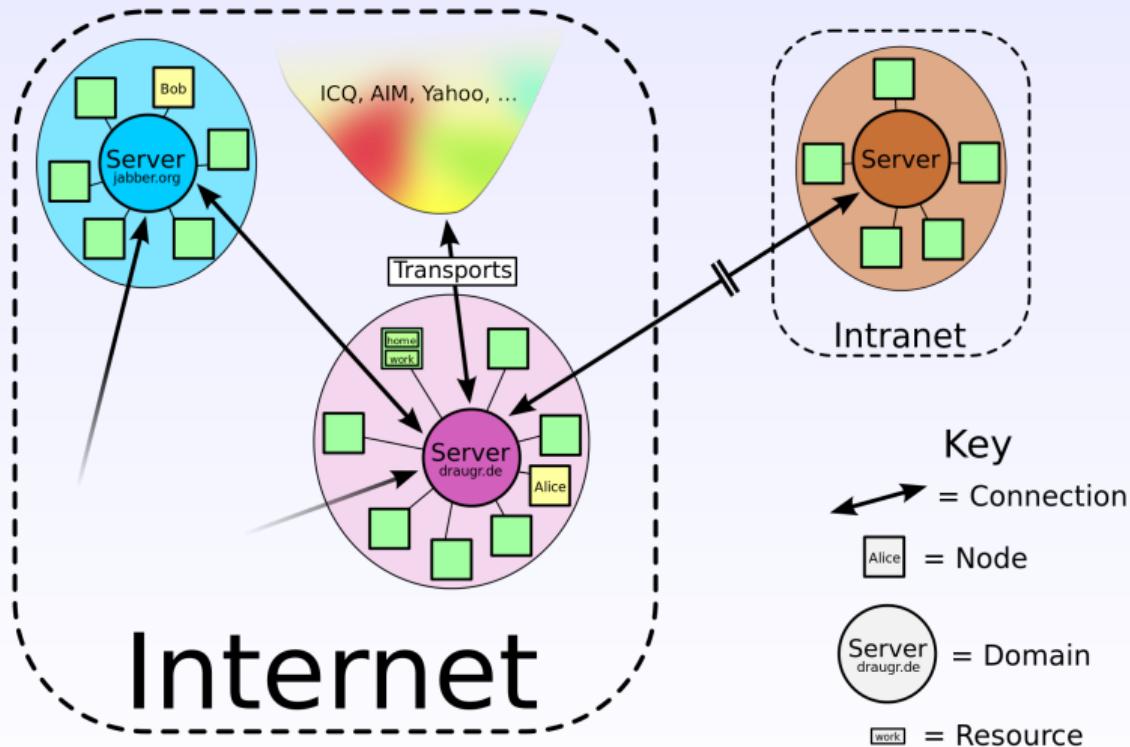
Características

Descentralización y
direcccionamiento

Utilizando XMPP para
comunicarse

Ejemplo de
comunicación
Cliente-Servidor

Tipos de Stanzas



Protocolo XMPP (Jabber)

Descentralización y direccionamiento

Ejemplo de mensaje

```
1 <message xml:lang='es'  
2   to='romeo@montesco.net'  
3   from='juliet@capuleto.com/balcon'  
4   type='chat'>  
5   <body> Romeo, Romeo!!! </body>  
6 </message>
```

Protocolo XMPP (Jabber)

Introducción

Características

Descentralización y
direccionamiento

Utilizando XMPP para
comunicarse

Ejemplo de
comunicación
Cliente-Servidor

Tipos de Stanzas

Protocolo XMPP (Jabber)

Utilizando XMPP para comunicarse

Capa de
aplicación

Protocolo XMPP
(Jabber)

Introducción

Características

Descentralización y
direccionalamiento

Utilizando XMPP para
comunicarse

Ejemplo de
comunicación
Cliente-Servidor

Tipos de Stanzas

Stream

- Para comenzar se abre un socket TCP permanente y se negocia un stream XML con el servidor. El servidor abre otro stream XML en la dirección contraria.
- Un stream es un sobre XML abierto, enviado antes que cualquier otro elemento (*Stanza*) entre el cliente y el servidor XMPP.
- Comienza opcionalmente con una instrucción de procesamiento XML (Prólogo).
- A continuación un elemento <stream:stream/> no terminado. Se va construyendo incrementalmente durante la comunicación.
- Además el stream contiene información la dirección del servidor, versión del protocolo y varias declaraciones de espacios de nombres

Protocolo XMPP (Jabber)

Utilizando XMPP para comunicarse

Capa de
aplicación

Protocolo XMPP (Jabber)

Introducción

Características

Descentralización y
direccionalamiento

Utilizando XMPP para comunicarse

Ejemplo de
comunicación
Cliente-Servidor

Tipos de Stanzas

Stanza

- Una vez negociado el stream, el cliente y el servidor pueden intercambiar los componentes XML (*Stanzas*).
- Un *stanza* es un componente XML completo y bien formado, son las unidades básicas de significado en XMPP.
- Siempre son los hijos de primer nivel en el documento XML.
- Se definen tres tipos de *stanzas*:<presence/>, <message/> y <iq/>

Protocolo XMPP (Jabber)

Ejemplo de comunicación Cliente-Servidor

Capa de
aplicación

Protocolo XMPP (Jabber)

Introducción

Características

Descentralización y
direccionalamiento

Utilizando XMPP para
comunicarse

Ejemplo de
comunicación
Cliente-Servidor

Tipos de Stanzas

```
1 C: <stream:stream>
2
3 C: <presence/>
4
5 C: <iq type="get">
6     <query xmlns="jabber:iq:roster"/>
7 </iq>
8
9 S: <iq type="result">
10    <query xmlns="jabber:iq:roster">
11        <item jid="alicet@wonderland.lit"/>
12        <item jid="madhatter@wonderland.lit"/>
13        <item jid="whiterabbit@wonderland.lit"/>
14    </query>
15
16 C: <message from="queen@wonderland.lit"
17           to="madhatter@wonderland.lit">
18     <body>Que le corten la cabeza</body>
19   </message>
20
21 S: <message from="king@wonderland.lit"
22           to="party@conference.wonderland.lit">
23     <body>Se os perdona a todos</body>
24   </message>
25
26 C: <presence type="unavailable"/>
```



Capa de aplicación

Protocolo XMPP
(Jabber)

Tipos de Stanzas

Message

Presence

Info / Query (IQ)

Tipos de Stanzas

Message

Descripción

- Método 'push' básico.
- Típicamente se envía sin esperar respuesta.
- Usado para mensajes instantáneos, chats grupales, alertas y notificaciones y similares.

Cinco tipos (atributo *type*)

normal: mensajes tipo e-mail; pueden o no tener respuesta; asíncrono.

chat: intercambiados en sesiones síncronas. Por ej: mensaje instantáneo.

groupchat: para grupos

headline: usados para enviar alertas y notificaciones, y no deben tener respuesta.

error: retornado por una entidad cuando se detecta un error en el mensaje anteriormente recibido.

Tipos de Stanzas

Presence

Descripción

- Características distintivas de los sistemas de comunicación síncronos.
- Se necesita autorizar a otros usuarios para que vean mi estado.
- Por lo tanto es un modelo simple de suscripción-publicación.
- Básicamente indica presencia / no presencia, pero se puede extender.

Ejemplo

```
1 <presence from= 'alice@wonderland.lit/pda'>
2   <show>xa</show>
3   <status>down the rabbit hole!</status>
4 </presence>
```

Tipos de Stanzas

Info/Query (IQ)

Descripción

- Estructura para interacciones pedido/respuesta y flujos de trabajo simples.
- A diferencia del *message*, un *iq* solo puede tener una “carga” (hijo).
- Siempre debe tener respuesta.
- Pedidos y respuestas se rastrean con un id especial anexado

Cuatro tipos (atributo *type*)

get: similar al GET de HTTP.

set: similar al POST o PUT de HTTP.

result: retorna algo pedido con get, o confirma algo solicitado con set
(similar al código de estado 200 del HTTP)

error: El destino o un servidor intermedio informan que la solicitud no se ha podido completar (XML en vez de códigos de error).



Tipos de Stanzas

Info/Query (IQ)

Capa de
aplicación

Protocolo XMPP
(Jabber)

Tipos de Stanzas

Message

Presence

Info/Query (IQ)

Solicitud

```
1 <iq from="alice@wonderland.lit"
2     id="rr82a1z7"
3     to="alice@wonderland.lit"
4     type="get">
5     <query xmlns="jabber:iq:roster"/>
6   </iq>
```

Respuesta

```
1 <iq from="alice@wonderland.lit"
2     id="rr82a1z7"
3     to="alice@wonderland.lit/pda"
4     type="result">
5     <query xmlns="jabber:iq:roster">
6         <item jid="whiterabbit@wonderland.lit"/>
7         <item jid="lory@wonderland.lit"/>
8         <item jid="mouse@wonderland.lit"/>
9         <item jid="sister@wonderland.lit"/>
10    </query>
11  </iq>
```

Capa de
aplicación

Arquitectura P2P
pura

Distribución

Bit Torrent

DHT (Distributed
Hash Table)

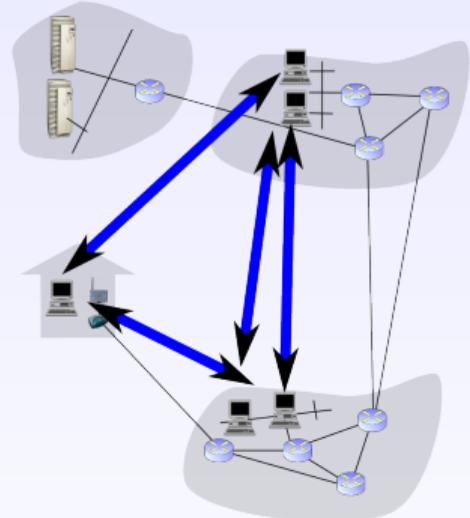
Skype

Parte IX

Aplicaciones P2P

Arquitectura P2P pura

- No tiene necesidad de servidores
- Se comunican dos sistemas finales arbitrarios
- Los pares se conectan intermitentemente y cambian sus IPs



Distribución de ficheros

Cliente–Servidor vs. P2P

Capa de aplicación

Arquitectura P2P pura

Distribución

Cliente–Servidor vs.
P2P

Tiempo
cliente–servidor

Tiempo P2P

Comparativa

BitTorrent

DHT (Distributed
Hash Table)

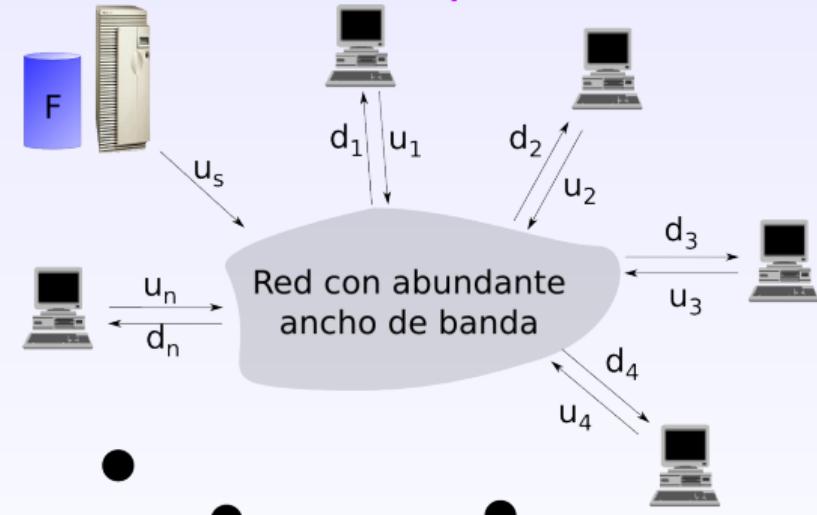
Skype

¿Cuanto se tarda en distribuir un fichero en ambas arquitecturas?

u_s Ancho de banda de subida en el servidor

u_i Ancho de banda de subida del par i

d_i Ancho de banda de bajada del par i



Distribución de ficheros

Tiempo de distribución cliente–servidor

Capa de
aplicación

Arquitectura P2P
pura

Distribución

Cliente–Servidor vs.
P2P

Tiempo
cliente–servidor

Tiempo P2P

Comparativa

BitTorrent

DHT (Distributed
Hash Table)

Skype

- El servidor manda N copias de un fichero de tamaño F:

$$\frac{NF}{u_s}$$

- El cliente i tarda en descargar el fichero:

$$\frac{F}{d_i}$$

Por tanto el tiempo T que tarda el servidor en distribuir el fichero a los n clientes es:

$$d_{cs} = \max \left(\frac{NF}{u_s}, \frac{F}{\min(d_i)} \right)$$

que para grandes N crece linealmente.

Distribución de ficheros

Tiempo de distribución P2P

Tomando como el equivalente al servidor en único par que dispone inicialmente del fichero:

- El servidor tiene que enviar una copia: $\frac{NF}{u_s}$
- El cliente tarda en recibir el fichero: $\frac{F}{d_i}$
- El tiempo de subida con eficiencia de distribución absoluta es:

$$u_s + \sum u_i$$

Por tanto el tiempo total es:

$$d_{P2P} = \max \left(\frac{NF}{u_s}, \frac{F}{\min(d_i)}, \frac{NF}{u_s + \sum u_i} \right)$$

Distribución de ficheros

Comparativa de ejemplo

Capa de
aplicación

Arquitectura P2P
pura

Distribución

Cliente-Servidor vs.
P2P

Tiempo
cliente-servidor

Tiempo P2P

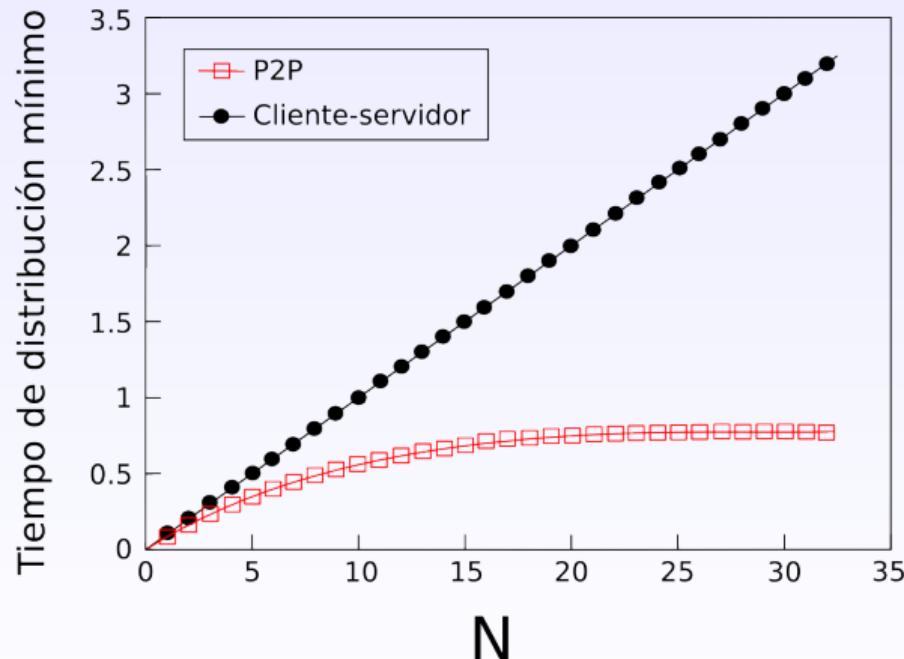
Comparativa

Bit Torrent

DHT (Distributed
Hash Table)

Skype

Velocidad de subida del cliente = u , $\frac{F}{u} = 1 \text{ h}$, $u_s = 10u$, $d_{min} \geq u_s$



BitTorrent

Arquitectura

Capa de
aplicación

Arquitectura P2P
pura

Distribución

Bit Torrent

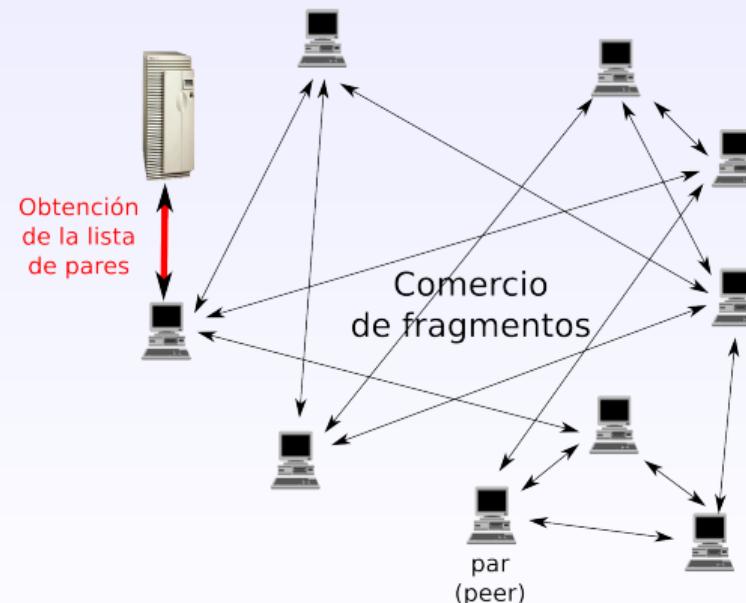
Arquitectura
Características
Toma y daca

DHT (Distributed
Hash Table)

Skype

Tracker: rastrea a los participantes en Torrent

Torrent: grupo de pares intercambiando fragmentos (chunks) de ficheros



BitTorrent

Características

Recogiendo fragmentos

- En cualquier momento, diferentes pares tienen subcolecciones de fragmentos del mismo fichero diferentes
- Periódicamente, un par le pide a sus vecinos la lista de fragmentos
- Este par solicita los fragmentos que no tiene a sus vecinos
 - Siempre se pide primero el más raro (menos reproducido)

Enviando fragmentos: toma y daca (tit-for-tat)

- Un determinado par envía sus fragmentos a los cuatro vecinos que le envían fragmentos a mayor ritmo.
 - Se vuelve a evaluar el top 4 cada 10 s
- Cada 30 s: selecciona aleatoriamente otro par e inicia envío de fragmentos.
 - El nuevo par siempre estará en el top 4 en esa ronda
 - De forma optimista este proceso deshace atascos de pares parados

BitTorrent

Toma y daca (tit-for-tat)

Capa de aplicación

Arquitectura P2P pura

Distribución

Bit Torrent

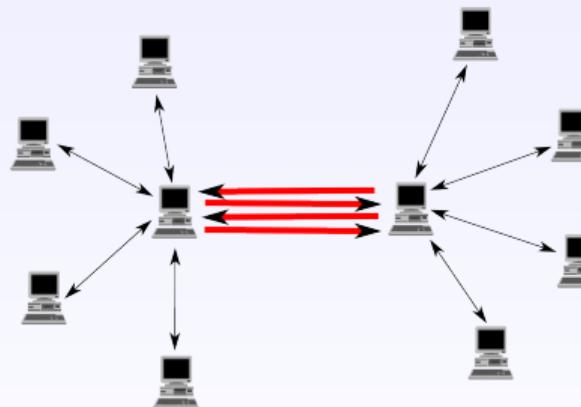
Arquitectura
Características

Toma y daca

DHT (Distributed Hash Table)

Skype

- ① El par A desatasca de forma optimista al par B
- ② El par A lo pone en su top-4
- ③ Si B transfiere bien de forma recíproca se mantendrá en el top-4
- ④ Si B no transfiere bien de forma recíproca no se mantendrá en el top-4



Este comportamiento permite el descubrimiento de mejores pares para el intercambio y por tanto permite descargar el fichero de forma más rápida

DHT (Distributed Hash Table)

Descripción

Capa de
aplicación

Arquitectura P2P
pura

Distribución

Bit Torrent

DHT (Distributed
Hash Table)

Descripción

Identificadores DHT

Claves y pares

DHT circular

DHT circular con
atajos

Desaparición de pares

Skype

- DHT es la base de datos distribuida de P2P
- La base de datos contiene pares (**clave, valor**)
 - clave: identificador usuario; valor nombre
 - valor: identificador de contenido; valor dirección IP
- Los pares **consultan** la base de datos con la llave
 - La base de datos responde con el valor correspondiente
- Los pares (peers) sólo pueden **insertar** pares (**clave, valor**) en la base de datos

DHT (Distributed Hash Table)

Identificadores DHT

Capa de
aplicación

Arquitectura P2P
pura

Distribución

Bit Torrent

DHT (Distributed
Hash Table)

Descripción

Identificadores DHT

Claves y pares

DHT circular

DHT circular con
atajos

Desaparición de pares

Skype

- Asigna un identificador a cada par (peer) en el rango $[0, 2^n - 1]$
 - Cada identificador puede ser representado por n bits
- Todas las claves tienen que estar en el mismo rango
- Si la clave es un literal hay que crear previamente un “hash” de éste.
 - Por eso se denomina “tabla hash distribuida”
 - Ejemplo: clave = hash (“Led Zeppelin IV”)

DHT (Distributed Hash Table)

Claves y pares

Capa de
aplicación

Arquitectura P2P
pura

Distribución

Bit Torrent

DHT (Distributed
Hash Table)

Descripción

Identificadores DHT

Claves y pares

DHT circular

DHT circular con
atajos

Desaparición de pares

Skype

Objetivo: asignar pares (clave,valor) a cada par (peer)

Regla: Asignación por vecinos más cercanos

Convención: cercano es el sucesor inmediato de la clave

Ejemplo: con $n=4$ y los estando presente los pares 1,3,4,5,8,10,12,14;

- Clave 13 \Rightarrow sucesor par 14
- Clave 15 \Rightarrow sucesor par 1

DHT (Distributed Hash Table)

DHT circular

Capa de aplicación

Arquitectura P2P pura

Distribución

Bit Torrent

DHT (Distributed Hash Table)

Descripción

Identificadores DHT

Claves y pares

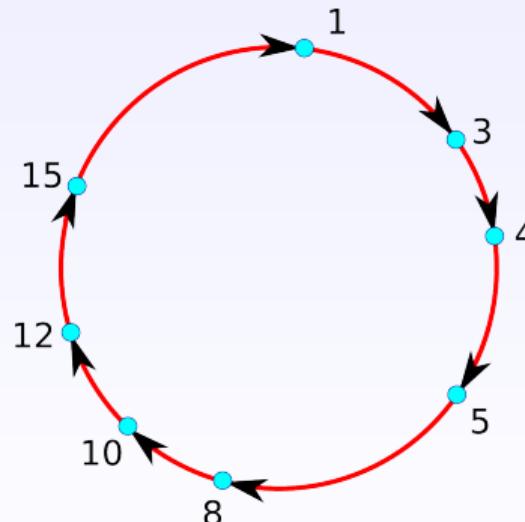
DHT circular

DHT circular con atajos

Desaparición de pares

Skype

- Cada par sólo tiene conocimiento directo de su sucesor
- Cada par sólo tiene conocimiento indirecto de su predecesor
- Red solapada
- $N/2$ mensajes de media



DHT (Distributed Hash Table)

DHT circular

Capa de
aplicación

Arquitectura P2P
pura

Distribución

Bit Torrent

DHT (Distributed
Hash Table)

Descripción
Identificadores DHT

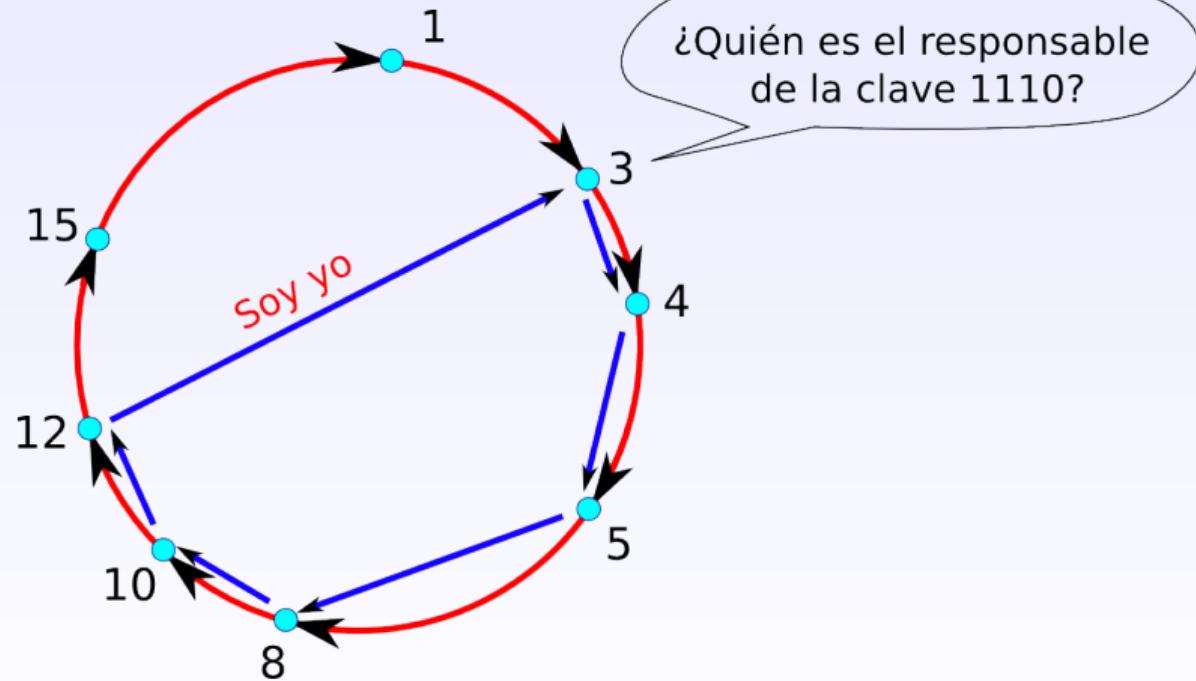
Claves y pares

DHT circular

DHT circular con
atajos

Desaparición de pares

Skype



DHT (Distributed Hash Table)

DHT circular con atajos

Capa de
aplicación

Arquitectura P2P
pura

Distribución

Bit Torrent

DHT (Distributed
Hash Table)

Descripción

Identificadores DHT

Claves y pares

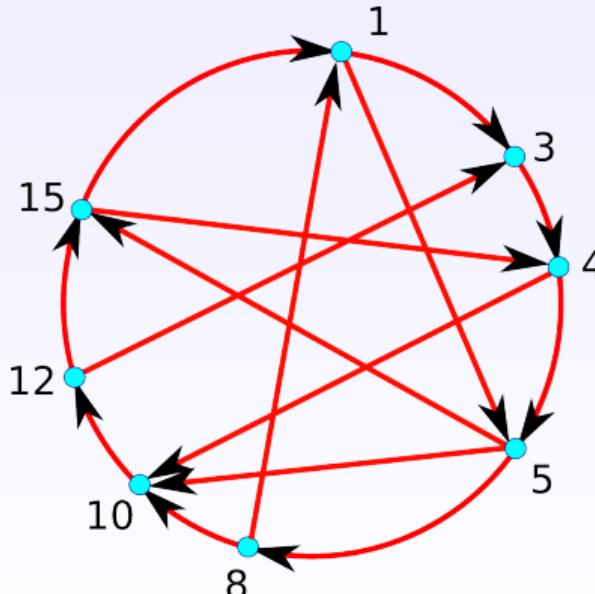
DHT circular

DHT circular con
atajos

Desaparición de pares

Skype

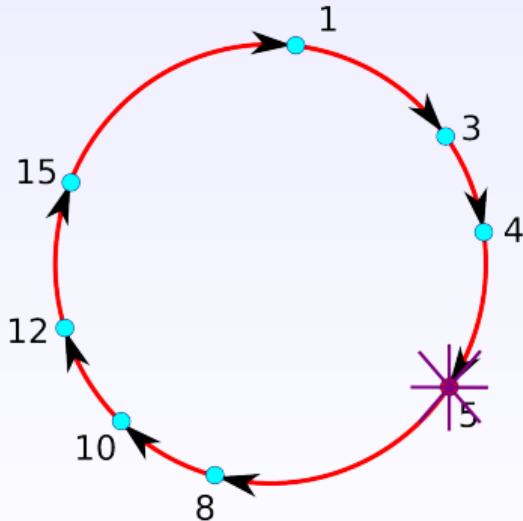
- Cada par tiene conocimiento directo de su sucesor y los atajos
- Se reduce considerablemente en número de mensajes
- Con el número adecuado de vecinos se puede reducir a $O(\log N)$



DHT (Distributed Hash Table)

Desaparición de pares

- Cada par necesita conocer las IPs de sus dos inmediatos sucesores
- Cada par comprueba periódicamente la presencia de sus dos inmediatos sucesores



Ejemplo

- El par 5 abandona
- El par 4 lo detecta; hace de 8 su inmediato sucesor; pregunta a 8 por su inmediato sucesor; convierte la respuesta en su segundo sucesor
- Aparece par 13
- Sólo sabe de la existencia del par 1
- Le pregunta a este par y la pregunta circula por toda la DHT

Skype

Arquitectura

Capa de
aplicación

Arquitectura P2P
pura

Distribución

Bit Torrent

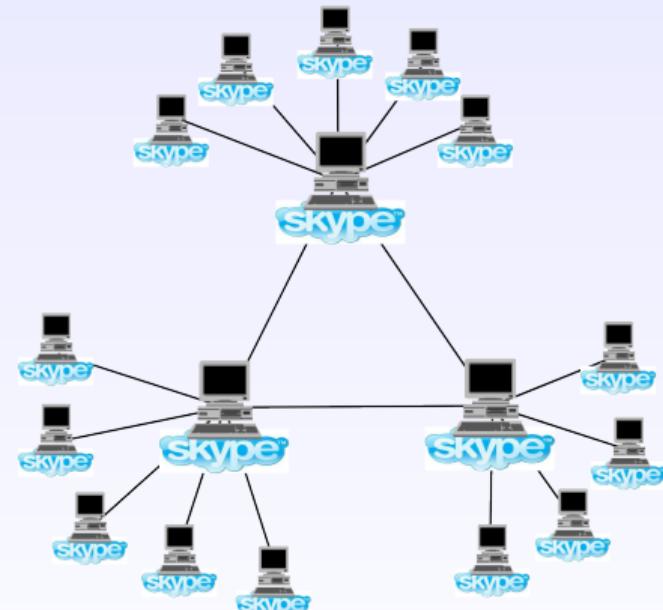
DHT (Distributed
Hash Table)

Skype

Arquitectura

Retransmisiones

- Básicamente P2P: se comunican dos usuarios
- Protocolo de aplicación privativo (inferido por ingeniería inversa)
- Estructura jerárquica con retransmisores (relays)
- Índices de usuarios distribuidos en los retransmisores



Skype

Retransmisores

Capa de
aplicación

Arquitectura P2P
pura

Distribución

Bit Torrent

DHT (Distributed
Hash Table)

Skype

Arquitectura
Retransmisores

Solución

- Los supernodos se comunican y determinan el retransmisor de la comunicación
- Se lo comunican a los pares
- Los pares inician la conexión con el retransmisor

