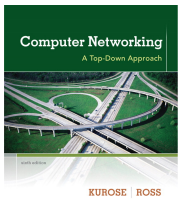
Chapter 2
Application Layer



Nota sobre el uso de estas diapositivas ppt:
Ponemos estas diapositivas a disposición de todos (profesores, estudiantes, lectores). Están en formato PowerPoint para que puedas ver las animaciones; y puedes añadir, modificar y eliminar diapositivas (incluida ésta) y el contenido de las mismas para adaptarlas a tus necesidades. Obviamente, representan *mucho* trabajo por nuestra parte. A cambio de su uso, solicitamos lo siguiente:

- Si utilizas estas diapositivas (por ejemplo, en una clase) que menciones su fuente (después de todo, ¡nos gustaría que la gente utilizara nuestro libro!)
- Si publica alguna diapositiva en un sitio www, que haga constar que está adaptada de nuestras diapositivas (o quizás sea idéntica a ellas), y que haga constar nuestros derechos de autor sobre este material.

Gracias y que lo disfruten. JFK/KWR

Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

Capítulo 2: esquema

2.1 principios de las aplicaciones de red
2.2 Web y HTTP
2.3 FTP
2.4 correo electrónico
- SMTP, POP3, IMAP

2.5 DNS
2.6 Aplicaciones P2P
2.7 programación de sockets con UDP y TCP

Capítulo 2: capa de aplicación

nuestros objetivos:

- aspectos conceptuales y de implementación de los protocolos de aplicación de la red
  - modelos de servicio de la capa de transporte
  - paradigma cliente-servidor
  - paradigma peer-to-peer
- aprender sobre protocolos examinando protocolos populares a nivel de aplicación
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- creación de aplicaciones de red
  - API de socket

Algunas aplicaciones de red

- correo electrónico
- web
- mensajes de texto
- acceso remoto
- Intercambio de archivos P2P
- juegos en red multiusuario
- transmisión de vídeo almacenado (YouTube, Hulu, Netflix)
- voz sobre IP (por ejemplo, Skype)
- videoconferencia en tiempo real
- redes sociales
- busque en
- …
- …

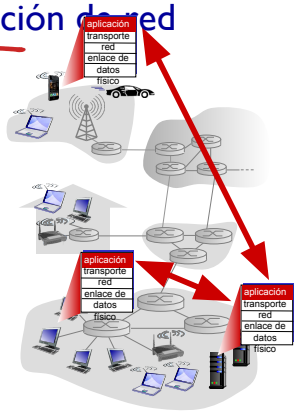
Creación de una aplicación de red

escribir programas que:

- se ejecutan en *sistemas finales* (diferentes)
- comunicarse a través de la red
- por ejemplo, el software del servidor web se comunica con el software del navegador

no es necesario escribir software para los dispositivos con núcleo de red

- los dispositivos con núcleo de red no ejecutan aplicaciones de usuario
- aplicaciones en los sistemas finales permite un rápido

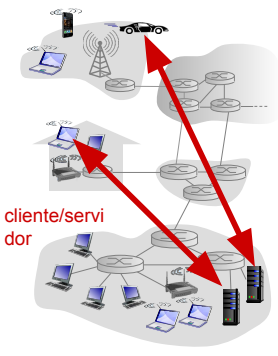


Arquitecturas de aplicación

posible estructura de las aplicaciones:

- cliente-servidor
- peer-to-peer (P2P)

Arquitectura cliente-servidor



servidor:

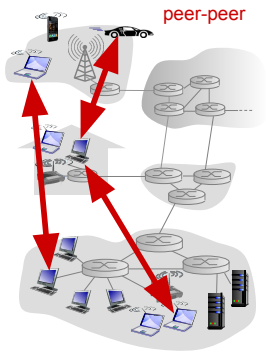
- anfitrión siempre activo
- dirección IP permanente
- centros de datos para escalar

clientes:

- comunicarse con el servidor
- puede estar conectado de forma intermitente
- pueden tener direcciones IP dinámicas
- no se comunican directamente entre sí

Arquitectura P2P

- *no hay* un servidor siempre activo
- los sistemas finales arbitrarios se comunican directamente
- los compañeros solicitan un servicio a otros compañeros, proporcionan un servicio a cambio a otros compañeros
  - *autoescalabilidad*: los nuevos compañeros aportan nueva capacidad de servicio, así como nuevas demandas de servicio
- los compañeros se conectan



Procesos de comunicación

proceso: programa que se ejecuta dentro de un host

- dentro del mismo host, dos procesos se comunican utilizando la comunicación entre procesos (definida por el SO)
- los procesos en diferentes hosts se comunican mediante el intercambio de mensajes

clientes, servidores

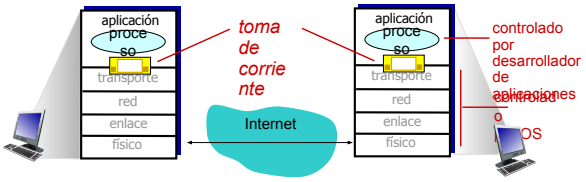
proceso cliente: proceso que inicia la comunicación

proceso servidor: proceso que espera ser contactado

- aparte: las aplicaciones con arquitecturas P2P tienen procesos de cliente y procesos de servidor

Tomas de corriente

- el proceso envía/recibe mensajes a/desde su socket
- enchufe análogo al de la puerta
  - el proceso de envío empuja el mensaje por la puerta
  - el proceso emisor depende de la infraestructura de transporte del otro lado de la puerta para entregar el mensaje al socket del proceso receptor



Procesos de direccionamiento

- para recibir mensajes, el proceso debe tener *el identificador*
- el dispositivo anfitrión tiene una dirección IP única de 32 bits
- *P:* ¿Es suficiente la dirección IP del host en el que se ejecuta el proceso para identificar el proceso?
  - *R:* no, muchos procesos pueden ser ejecutados en el mismo host

- El *identificador* incluye tanto la dirección IP como los números de puerto asociados al proceso en el host.
- números de puerto de ejemplo:
  - Servidor HTTP: 80
  - servidor de correo: 25
- para enviar un mensaje HTTP al servidor web gaia.cs.umass.edu:
  - Dirección IP: 128.119.245.12
  - número de puerto: 80
- más en breve...

El protocolo de la capa de aplicación define

- tipos de mensajes intercambiados,
  - por ejemplo, solicitud, respuesta
- sintaxis del mensaje:
  - qué campos hay en los mensajes y cómo se delimitan los campos
- semántica del mensaje
  - significado de la información en los campos
- reglas sobre cuándo y cómo los procesos envían y responden a los mensajes

protocolos abiertos:

- definidos en las RFC
- permite la interoperabilidad
- por ejemplo, HTTP, SMTP

protocolos propios:

- por ejemplo, Skype

¿Qué servicio de transporte necesita una aplicación?

integridad de los datos

- algunas aplicaciones (por ejemplo, transferencia de archivos, transacciones web) requieren una transferencia de datos 100% fiable
- otras aplicaciones (por ejemplo, de audio) pueden tolerar algunas pérdidas
- algunas aplicaciones (por ejemplo, la telefonía por Internet o los juegos interactivos) requieren un bajo retardo para ser "efectivas"

cronometraje

rendimiento

- algunas aplicaciones (por ejemplo, las multimedia) requieren una cantidad mínima de rendimiento para ser "efectivas"
- otras aplicaciones ("aplicaciones elásticas") utilizan el rendimiento que obtienen

seguridad

- encriptación, integridad de los datos, ...

Requisitos del servicio de transporte: aplicaciones comunes

aplicación	pérdida de datos	rendimiento	sensible al tiempo
transferencia de archivos	no hay	elástico	no
correo electrónico	pérdida	elástico	no
Documentos web	no hay pérdida	audio: 5kbps-1Mbps vídeo:10kbps-5Mbps	no sí, 100's mseg.
audio/vídeo en tiempo real	no hay pérdida	lo mismo que lo anterior	sí, unos segundos
audio/vídeo almacenado	tolerante a las pérdidas	pocos kbps arriba	sí, 100's mseg.
juegos interactivos	tolerante a las pérdidas	elástico	sí y no
mensajes de texto	tolerante a las pérdidas no hay		

Servicios de protocolos de transporte de Internet

Servicio TCP:

- *transporte fiable* entre el proceso de envío y el de recepción
- *control de flujo: el* emisor no sobrecargará al receptor
- *control de la congestión:* estrangulación del emisor cuando la red está sobrecargada
- *no proporciona:* temporización, garantía de rendimiento mínimo, seguridad
- *orientado a la conexión:* se requiere una configuración entre los procesos del cliente y del servidor

Servicio UDP:

- *transferencia de datos poco fiable* entre el proceso de envío y el de recepción
- *no proporciona:* fiabilidad, control de flujo, control de congestión, temporización, garantía de rendimiento, seguridad o configuración de la conexión,

P: ¿Por qué molestarse?

Aplicaciones de Internet: aplicación, protocolos de transporte

aplicación	aplicación protocolo de capa	subyacente protocolo de transporte
correo electrónico	SMTP [RFC 2821]	TCP
acceso remoto al terminal	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
transferencia de archivos	FTP [RFC 959]	TCP
streaming multimedia	HTTP (por ejemplo, YouTube), RTP [RFC 1889]	TCP o UDP
Telefonía por Internet	SIP, RTP, propietario (por ejemplo, Skype)	TCP o UDP

Asegurar el TCP

TCP Y UDP

- sin encriptación
- las contraseñas en texto claro enviadas al socket atraviesan Internet en texto claro

SSL

- proporciona una conexión TCP encriptada
- integridad de los datos
- autenticación en el

SSL está en la capa de la aplicación

- Las aplicaciones utilizan bibliotecas SSL, que "hablan" con TCP

API de socket SSL

- las contraseñas en texto claro enviadas al socket atraviesan Internet encriptadas
- Véase el capítulo 7

Capítulo 2: esquema

2.1 principios de las aplicaciones de red
- arquitecturas de aplicaciones
- requisitos de la aplicación

2.2 Web y HTTP

2.3 FTP

2.4 correo electrónico
- SMTP, POP3, IMAP

2.5 DNS

2.6 Aplicaciones P2P

2.7 programación de sockets con UDP y TCP

Web y HTTP

Primero, un repaso...

- La *página web* se compone de *objetos*
- El objeto puede ser un archivo HTML, una imagen JPEG, un applet Java, un archivo de audio,...
- *La* página web consiste en un *archivo HTML base* que incluye *varios objetos referenciados*
- cada objeto es direccionable mediante una *URL, por* ejemplo

`www.someschool.edu/someDept/pic.gif`

nombre del anfitrión — nombre de la ruta

Visión general de HTTP

HTTP: protocolo de transferencia de hipertexto

- Protocolo de la capa de aplicación de la Web
- modelo cliente/servidor
  - *cliente:* navegador que solicita, recibe (mediante el protocolo HTTP) y "muestra" objetos web
  - *servidor:* El servidor web envía (mediante el protocolo HTTP) objetos en respuesta a las solicitudes

Visión general de HTTP (continuación)

utiliza TCP:

- el cliente inicia una conexión TCP (crea un socket) con el servidor, puerto 80
- el servidor acepta la conexión TCP del cliente
- Mensajes HTTP (mensajes del protocolo de la capa de aplicación) intercambiados entre el navegador (cliente HTTP) y el servidor web (servidor HTTP)
- Conexión TCP cerrada

HTTP es "sin estado"

- el servidor no mantiene ninguna información sobre las solicitudes anteriores de los clientes

a un lado

Los protocolos que mantienen el "estado" son complejos.
- se debe mantener el historial (estado)
- si el servidor/cliente se bloquea, sus puntos de vista sobre el "estado" pueden ser inconsistentes, deben ser reconciliados

Conexiones HTTP

HTTP no persistente

- como máximo un objeto enviado a través de una conexión TCP
  - conexión entonces cerrada
- la descarga de varios objetos requiere múltiples conexiones

HTTP persistente

- se pueden enviar múltiples objetos a través de una única conexión TCP entre el cliente, el servidor

HTTP no persistente

suponga que el usuario introduce la URL: `www.someSchool.edu/someDepartment/home.index` (contiene texto, referencias a 10 imágenes jpeg)

1a. El cliente HTTP inicia la conexión TCP con el servidor HTTP (proceso) en www.someSchool.edu en el puerto 80

1b. Servidor HTTP en el host www.someSchool.edu esperando una conexión TCP en el puerto 80. "acepta" la conexión, notificando al cliente

2. El cliente HTTP envía *un mensaje de solicitud* HTTP (que contiene una URL) al socket de conexión TCP. El mensaje indica que el cliente quiere el objeto algúnDepartamento/hogar.index

3. El servidor HTTP recibe el mensaje de solicitud, forma el *mensaje de respuesta* que contiene el objeto solicitado y envía el mensaje a su socket

tiempo

HTTP no persistente (cont.)

4. El servidor HTTP cierra la conexión TCP.

5. El cliente HTTP recibe un mensaje de respuesta que contiene un archivo html, muestra el html. 6. El cliente HTTP recibe un mensaje de respuesta que contiene un archivo html, muestra el html.

6. Los pasos 1-5 se repiten para cada uno de los 10 objetos jpeg

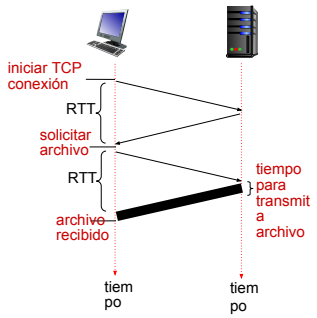
tiempo

HTTP no persistente: tiempo de respuesta

RTT (definición): tiempo que tarda un pequeño paquete en viajar del cliente al servidor y viceversa

Tiempo de respuesta HTTP:

- un RTT para iniciar la conexión TCP
- un RTT para la petición HTTP y los primeros bytes de la respuesta HTTP a devolver
- tiempo de transmisión del archivo
- tiempo de respuesta HTTP no persistente = 2RTT+ tiempo de transmisión de archivos



HTTP persistente

problemas de HTTP no persistentes:

- requiere 2 RTTs por objeto
- Sobrecarga del sistema operativo para *cada* conexión TCP
- los navegadores suelen abrir conexiones TCP paralelas para obtener los objetos referenciados

HTTP persistente:

- el servidor deja la conexión abierta después de enviar la respuesta
- mensajes HTTP posteriores entre el mismo cliente/servidor enviados a través de una conexión abierta
- el cliente envía peticiones tan pronto como encuentra un objeto referenciado
- tan sólo un RTT para todos los objetos referenciados

Mensaje de solicitud HTTP

- dos tipos de mensajes HTTP: *solicitud*, *respuesta*
- Mensaje de solicitud HTTP:
  - ASCII (formato legible para el ser humano)

carácter de retorno de carro
carácter de salto de línea

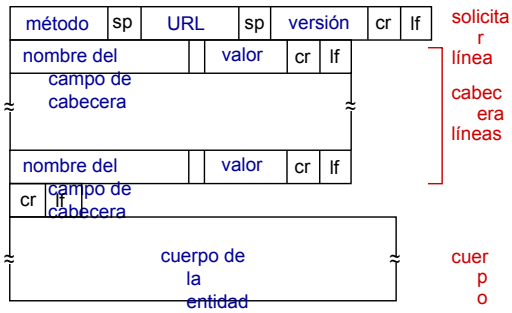
línea de solicitud (GET, POST, Comandos HEAD)

cabecera líneas

retorno de carro, alimentación de línea al inicio de la línea indica fin de las líneas de cabecera

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\n
Usuario-Agente: Firefox/3.6.10\n
Aceptar: text/html,application/xhtml+xml\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\n
Acepta el conjunto de caracteres:
   ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Conexión: keep-alive\r\n
\N - La vida en el mundo de los negocios
```


Mensaje de solicitud HTTP: formato general



Cargar la entrada del formulario

Método POST:

- La página web suele incluir un formulario
- la entrada se carga en el servidor en el cuerpo de la entidad

Método URL:

- utiliza el método GET
- La entrada se carga en el campo URL de la línea de solicitud:

```
www.somesite.com/animalsearch?monkeys&banana
```


Tipos de métodos

HTTP/1.0:

- GET
- POST
- CABEZA
  - pide al servidor que deje el objeto solicitado fuera de la respuesta

HTTP/1.1:

- OBTENER, PUBLICAR, CABEZA
- PUT
  - sube el archivo en el cuerpo de la entidad a la ruta especificada en el campo URL
- DELETE
  - elimina el archivo especificado en el campo URL

Mensaje de respuesta HTTP

línea de estado (protocolo código de situación frase de estado)

cabecera líneas

datos, por ejemplo, solicitado Archivo HTML

```
HTTP/1.1 200 OK\r\n
Fecha: Sun, 26 Sep 2010 20:09:20 GMTr\n
Servidor: Apache/2.0.52 (CentOS)\N-ES.
Última modificación: Tue, 30 Oct 2007
   17:00:02 GMTr\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Contenido-Longitud: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Conexión: Keep-Alive\r\n
Content-Type: text/html;
   charset=ISO-8859-1\r\n
\N - La vida en el mundo de los negocios
datos datos datos datos ...
```

Códigos de estado de respuesta HTTP

- El código de estado aparece en la primera línea del mensaje de respuesta del servidor al cliente.
- algunos códigos de muestra:

200 OK
- solicitud exitosa, objeto solicitado más adelante en este msg

301 Movido permanentemente
- objeto solicitado movido, nueva ubicación especificada más adelante en este msg (Ubicación:)

400 Solicitudes erróneas
- mensaje de solicitud no entendido por el servidor

404 no encontrado
- el documento solicitado no se encuentra en este servidor

505 Versión HTTP no soportada

Probar el HTTP (lado del cliente) por sí mismo

1. Conéctese por telnet a su servidor web favorito:

```
telnet cis.poly.edu
80
```

abre una conexión TCP al puerto 80 (puerto del servidor HTTP por defecto) en cis.poly.edu.
cualquier cosa que se escriba se envía al puerto 80 en cis.poly.edu

2. Escriba una petición GET HTTP:

```
GET /~ross/
HTTP/1.1
Alojamiento:
cis.poly.edu
```

escribiendo lo siguiente (pulsando el botón de transporte) devuelve dos veces), se envía enviando [por el servidor Solicitud GET al servidor HTTP

3. ¡mira el mensaje de respuesta enviado por el servidor HTTP!

(o utilice Wireshark para ver la solicitud/respuesta HTTP capturada)

Estado del usuario-servidor: cookies

muchos sitios web utilizan cookies

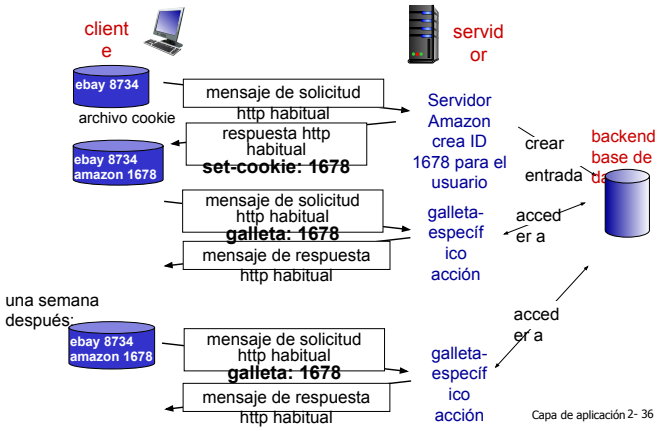
cuatro componentes:

1) línea de cabecera de la cookie del mensaje de *respuesta* HTTP
2) línea de cabecera de la cookie en el siguiente mensaje de *solicitud* HTTP
3) archivo de cookies guardado en el host del usuario, gestionado por el navegador del usuario

ejemplo:

- Susan siempre accede a Internet desde el PC
- visita por primera vez un sitio de comercio electrónico específico
- cuando las peticiones HTTP iniciales llegan al sitio, el sitio crea:
  - ID único
  - entrada en la base de datos del backend para el ID

Cookies: mantener el "estado" (cont.)



Cookies (continuación)

para qué se pueden utilizar las cookies:

- autorización
- carros de la compra
- recomendaciones
- estado de la sesión del usuario (correo electrónico de la web)

cookies y privacidad: a un lado

- las cookies permiten a los sitios web aprender mucho sobre usted
- puede suministrar el nombre y el correo electrónico a los sitios

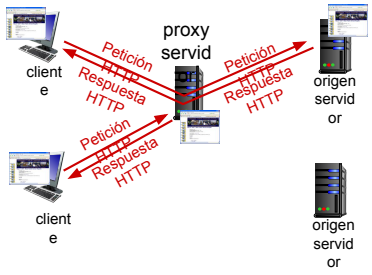
cómo mantener el "estado":

- puntos finales del protocolo: mantener el estado en el emisor/receptor a lo largo de múltiples transacciones
- cookies: los mensajes http llevan estado

Cachés web (servidor proxy)

objetivo: satisfacer la solicitud del cliente sin involucrar al servidor de origen

- El usuario establece el navegador: Los accesos a la web a través de la caché
- el navegador envía todas las peticiones HTTP a la caché
  - objeto en la caché: la caché devuelve el objeto
  - si no, la caché solicita el objeto al servidor de origen y lo devuelve al cliente



Más información sobre la caché web

- la caché actúa como cliente y servidor
  - servidor para el cliente solicitante original
  - cliente al servidor de origen
- normalmente la caché es instalada por el ISP (universidad, empresa, ISP residencial)

¿Por qué la caché web?

- reducir el tiempo de respuesta a las solicitudes de los clientes
- reducir el tráfico en el enlace de acceso de una institución
- Internet densa con cachés: permite a los proveedores de contenidos "pobres" entregarlos de forma efectiva (también lo hace

Ejemplo de caché:

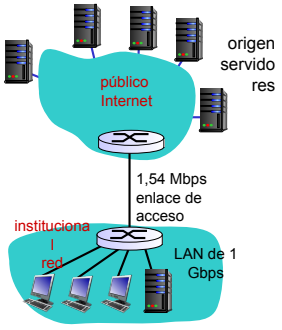
supuestos:

- tamaño medio del objeto: 100K bits
- tasa de solicitud media de los navegadores a los servidores de origen: 15/seg.
- Velocidad de datos media para los navegadores: 1,50 Mbps
- RTT desde el router institucional a cualquier servidor de origen: 2 segundos
- tasa de enlace de acceso: 1,54 Mbps

consecuencias:

- Utilización de la LAN: 15%
- utilización del enlace de acceso = 99%
- retardo total = retardo de Internet + retardo de acceso + retardo de LAN = 2 seg + minutos + usecs

¡problema!



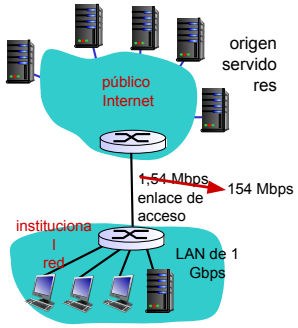
Ejemplo de caché: enlace de acceso más gordo

supuestos:

- tamaño medio del objeto: 100K bits
- tasa de solicitud media de los navegadores a los servidores de origen: 15/seg.
- Velocidad de datos media para los navegadores: 1,50 Mbps
- RTT desde el router institucional a cualquier servidor de origen: 2 segundos
- tasa de enlace de acceso: 1,54 Mbps

consecuencias:

- Utilización de la LAN: 15%
- utilización del enlace de acceso = 99%
- retardo total = retardo de Internet + retardo de acceso + retardo de LAN
  - 2 s + minutos + msecs



Coste: aumentar la velocidad del enlace de acceso (¡no es barato!)

Ejemplo de caché: instalar una caché local

supuestos:

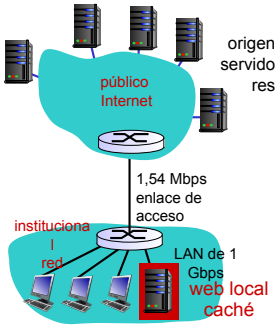
- tamaño medio del objeto: 100K bits
- tasa de solicitud media de los navegadores a los servidores de origen: 15/seg.
- Velocidad de datos media para los navegadores: 1,50 Mbps
- RTT desde el router institucional a cualquier servidor de origen: 2 segundos
- tasa de enlace de acceso: 1,54 Mbps

consecuencias:

- Utilización de la LAN: 15%
- utilización del enlace de acceso = 99%
- retardo total = retardo de Internet + retardo de acceso + retardo de LAN
  - 2 s + minutos + msecs

Cómo calcular el enlace de utilización, retraso?

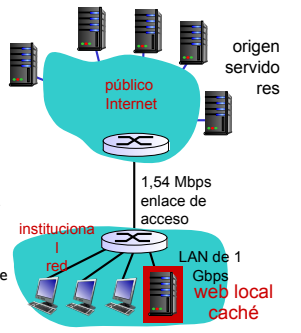
Coste: caché web (¡barato!)



Ejemplo de caché: instalar una caché local

Cálculo de la utilización del enlace de acceso, retraso con caché:

- supongamos que la tasa de aciertos de la caché es de 0,4
  - 40% de las solicitudes serán satisfechas en el caché, 60% de las solicitudes utilizan el enlace de acceso
- velocidad de datos a los navegadores a través del enlace de acceso = 0,6*1,50 Mbps = 0,9 Mbps
- retraso total
  - = 0,6 * (retraso desde los servidores de origen) +0,4 * (retraso cuando se satisface en la caché)
  - = 0,6 (2,01) + 0,4 (~mseg)
  - = ~ 1,2 seg.
  - menos que con un enlace de 154 Mbps (¡y más barato también!)



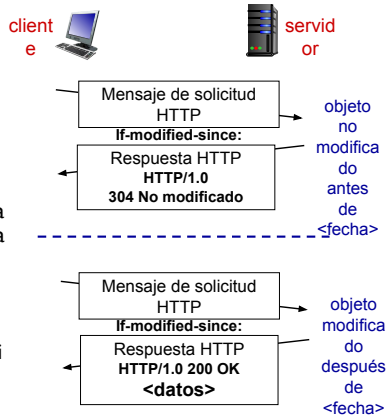
GET condicional

- *Objetivo:* no enviar el objeto si la caché tiene una versión actualizada
  - no hay retardo en la transmisión de objetos
  - menor utilización de los enlaces
- *caché:* especificar la fecha de la copia en caché en la petición HTTP

```
If-modified-since:
  <fecha>
```

- *servidor:* la respuesta no contiene ningún objeto si la copia en caché está actualizada:

```
HTTP/1.0 304 No
Modificado
```



Capítulo 2: esquema

2.1 principios de las aplicaciones de red
- arquitecturas de aplicaciones
- requisitos de la aplicación

2.2 Web y HTTP

2.3 FTP

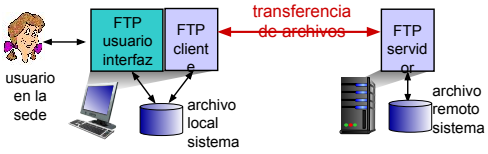
2.4 correo electrónico
- SMTP, POP3, IMAP

2.5 DNS

2.6 Aplicaciones P2P

2.7 programación de sockets con UDP y TCP

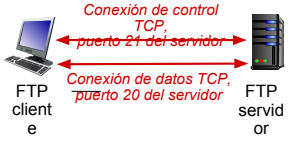
FTP: el protocolo de transferencia de archivos



- transferir un archivo a/desde un host remoto
- modelo cliente/servidor
  - *cliente:* lado que inicia la transferencia (ya sea hacia/desde el remoto)
  - *servidor:* host remoto
- ftp: RFC 959
- servidor ftp: puerto 21

FTP: control separado, conexiones de datos

- El cliente FTP contacta con el servidor FTP en el puerto 21, utilizando TCP
- cliente autorizado sobre la conexión de control
- el cliente navega por el directorio remoto, envía comandos a través de la conexión de control
- cuando el servidor recibe el comando de transferencia de archivos, *el servidor* abre *la segunda* conexión de datos TCP (para el archivo) *con* el cliente
- después de transferir un archivo, el servidor cierra la



- el servidor abre otra conexión de datos TCP para transferir otro archivo
- conexión de control: *"fuera de banda"*
- El servidor FTP mantiene el "estado": directorio actual, autentificación anterior

Comandos FTP, respuestas

comandos de muestra:

- enviado como texto ASCII por el canal de control
- ***Nombre de usuario***
- ***Contraseña* `PASS`**
- **`LIST`** devuelve la lista de archivos en el directorio actual
- **`RETR nombre de`** archivo recupera (obtiene) el archivo
- **`STOR nombre de archivo`** almacena (pone) el archivo en el host remoto

ejemplos de códigos de retorno

- código de estado y frase (como en HTTP)
- **`331 Nombre de usuario OK, se requiere contraseña`**
- **`125 conexión de datos ya abierta; transferencia iniciada`**
- **`425 No se puede abrir la conexión de datos`**
- **`452 Error al`**

Capítulo 2: esquema

2.1 principios de las aplicaciones de red
- arquitecturas de aplicaciones
- requisitos de la aplicación

2.2 Web y HTTP
2.3 FTP
2.4 correo electrónico
- SMTP, POP3, IMAP

2.5 DNS
2.6 Aplicaciones P2P
2.7 programación de sockets con UDP y TCP

El "correo electrónico" se define en el RFC 821 y el RFC 822

- El correo electrónico de Internet, es decir, no debe confundirse con el correo electrónico de la LAN, como cc:Mail o MS Mail, que utilizan protocolos propietarios
- El RFC 821 define el protocolo SMTP
- Cómo intercambian mensajes los MTA de correo
- El RFC 822 define el aspecto de un mensaje de correo

X.400

Sistema de tratamiento de mensajes y recomendación de resumen de servicios.

- X.400 es un conjunto de Recomendaciones del Sector de Normalización de las Telecomunicaciones (UIT-T) que define las normas de las redes de comunicación de datos para los sistemas de tratamiento de mensajes (MHS), más conocidos como "correo electrónico". Aunque X.400 nunca alcanzó la presencia universal del correo electrónico en Internet, se ha utilizado en las organizaciones y como parte de productos de correo electrónico patentados, como Microsoft Exchange.
- RFC1685, RFC1615, RFC1649

Direccionamiento X.400

- Una dirección X.400 se compone de varios elementos, entre ellos
- C (Nombre del país)
- ADMD (Administration Management Domain), normalmente un proveedor de servicios de correo público
- PRMD (Dominio de Gestión Privada)
- O (Nombre de la organización)
- OU (nombres de unidades organizativas)
- G (Nombre)
- I (Iniciales)
- S (Apellido)

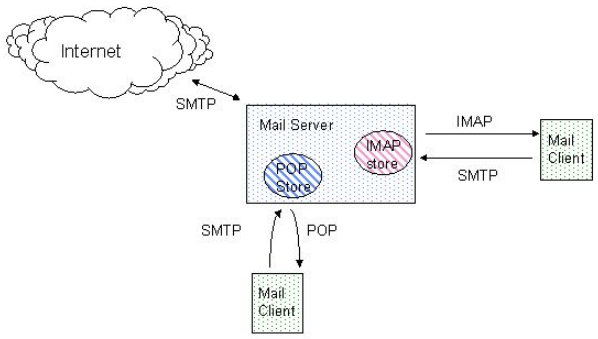
Las direcciones X.400 son feas

Contrasta las dos cadenas siguientes:

G=Harald; S=Alvestrand; O=sintef; OU=delab; PRMD=uninett; ADMD=uninett; C=no

Harald.Alvestrand@delab.sintef.no

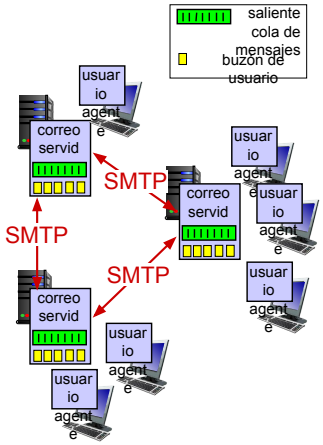
Un entorno de correo típico



Correo electrónico

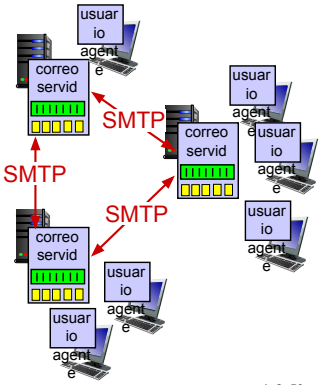
Tres componentes principales:
- agentes del usuario
- servidores de correo
- protocolo simple de transferencia de correo: SMTP

Agente del usuario
- También conocido como "lector de correo".
- redactar, editar y leer mensajes de correo
- Por ejemplo, Outlook, Thunderbird, cliente de correo del iPhone



Correo electrónico: servidores de correo

servidores de correo:
- El *buzón* contiene los mensajes entrantes del usuario
- *cola* de mensajes de correo saliente (por enviar)
- *Protocolo SMTP* entre servidores de correo para enviar mensajes de correo electrónico
  - cliente: servidor de correo de envío
  - "servidor": servidor de correo de recepción

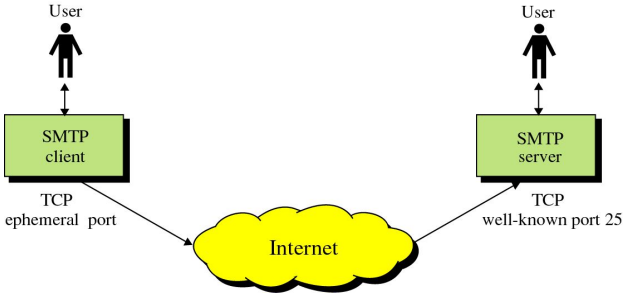


Correo electrónico: SMTP [RFC 2821]

- utiliza TCP para transferir de forma fiable los mensajes de correo electrónico del cliente al servidor, puerto 25
- transferencia directa: servidor emisor a servidor receptor
- tres fases de transferencia
  - apretón de manos (saludo)
  - transferencia de mensajes
  - cierre
- interacción comando/respuesta (como HTTP, FTP)
  - comandos: Texto ASCII
  - respuesta: código de estado y frase
- los mensajes deben estar en ASCI de 7 bits

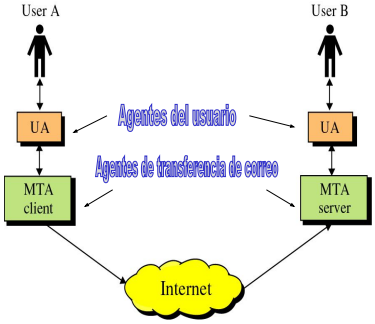
SMTP

- El protocolo se originó en 1982 (RFC821, Jon Postel)
- Formato de mensaje estándar (RFC822,2822, D. Crocker)
- Objetivo: transferir el correo de forma fiable y eficaz



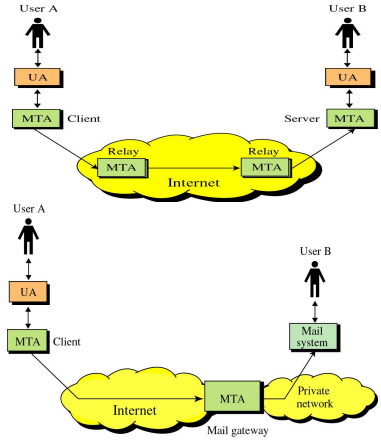
SMTP

- Los clientes y servidores SMTP tienen dos componentes principales
  - Agentes de usuario - Prepara el mensaje, lo encierra en un sobre. (ej. Ms-outlook, Eudora, Opera)
  - Agente de transferencia de correo - Transfiere el correo a través de Internet (por ejemplo, Sendmail, Exim)
  - Análogo al sistema postal en muchos aspectos



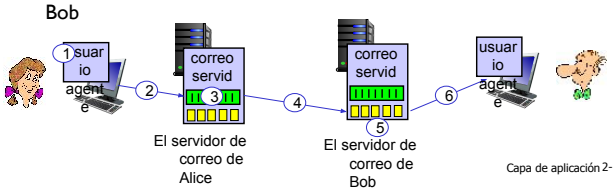
SMTP

- SMTP también permite el uso de Relays permitiendo a otros MTAs retransmitir el correo
- Las pasarelas de correo se utilizan para retransmitir el correo preparado por un protocolo distinto al SMTP y convertirlo en SMTP



Escenario: Alice envía un mensaje a Bob

1) Alice utiliza la UA para redactar el mensaje "para" `bob@someschool.edu`
2) La UA de Alice envía un mensaje a su servidor de correo; el mensaje se coloca en la cola de mensajes
3) el lado del cliente de SMTP abre una conexión TCP con el servidor de correo de Bob
4) El cliente SMTP envía el mensaje de Alice a través de la conexión TCP
5) El servidor de correo de Bob coloca el mensaje en el buzón de Bob
6) Bob invoca su agente de usuario para leer el mensaje



Ejemplo de interacción SMTP

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hola crepes.fr, encantado de conocerte
C: CORREO DE: <alice@crepes.fr>
S: 250 alice@crepes.fr... Remitente ok
C: RCPT A: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Destinatario ok
C: DATOS
S: 354 Introduzca el correo, termine con "." en una
línea sola
C: ¿Te gusta el ketchup?
C: ¿Y los pepinillos?
C: .
S: 250 Mensaje aceptado para su entrega
C: QUIT
S: 221 hamburger.edu cerrando la conexión
```


Pruebe usted mismo la interacción SMTP:

- **`telnet nombre de servidor 25`**
- ver 220 respuesta del servidor
- introduzca los comandos HELO, MAIL FROM, RCPT TO, DATA, QUIT

arriba le permite enviar un correo electrónico sin utilizar el cliente de correo electrónico (lector)

SMTP: palabras finales

- SMTP utiliza conexiones persistentes
- SMTP requiere que el mensaje (cabecera y cuerpo) esté en ASCII de 7 bits
- El servidor SMTP utiliza `CRLF.CRLF` para determinar el final del mensaje

comparación con HTTP:

- HTTP: pull
- SMTP: push
- ambos tienen interacción ASCII comando/respuesta, códigos de estado
- HTTP: cada objeto encapsulado en su propio msg de respuesta
- SMTP: envío de múltiples objetos en un mensaje multiparte

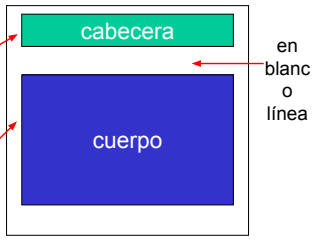
Formato del mensaje de correo

SMTP: protocolo de intercambio de mensajes de correo electrónico

RFC 822: norma para el formato de los mensajes de texto:

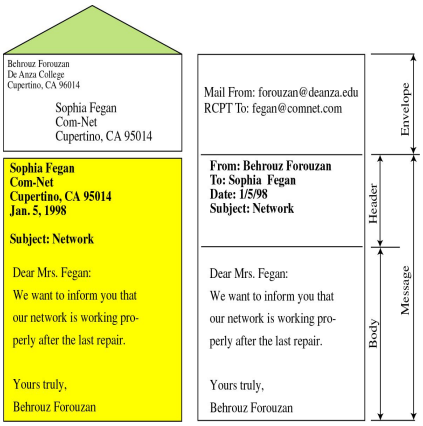
- líneas de cabecera, por ejemplo
  - Para:
  - De:
  - Asunto:

  diferente de los comandos SMTP MAIL FROM, RCPT TO:!
- Cuerpo: el "mensaje"
  - Sólo caracteres ASCII



Formato de un correo electrónico

- El correo es un archivo de texto
- Sobre -
  - dirección del remitente
  - dirección del receptor
  - otras informaciones
- Mensaje -
  - Encabezado del correo: define el remitente, el destinatario, el asunto del mensaje y otra información
  - Cuerpo del correo - Contiene la información real del mensaje



Oficina de Correos

Buzón

Oficina de Correos

y la ruta del correo

Receptor

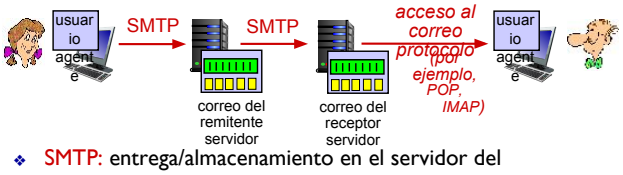
Buzón

```
De kissel@mail.acad.ece.udel.edu Tue Oct 25 20:27:21 2005
Return-Path: <kissel@mail.acad.ece.udel.edu>
X-Original-To: kissel@cis.udel.edu
Entregado a: kissel@cis.udel.edu

Recibido: por mail.eecis.udel.edu (Postfix, de userid 62)
        id 8EC8D18D; Tue, 25 Oct 2005 20:27:21 -0400 (EDT)
Recibido: de mail.acad.ece.udel.edu (devil-rays.acad.ece.udel.edu [128.4.60.10])
        by mail.eecis.udel.edu (Postfix) with ESMTP id 59888C9
        para <kissel@cis.udel.edu>; Tue, 25 Oct 2005 20:27:20 -0400 (EDT)
Recibido: por mail.acad.ece.udel.edu (Postfix, de userid 62)
        id 344482045; Tue, 25 Oct 2005 20:27:20 -0400 (EDT)
Recibido: de nimbus.acad.ece.udel.edu (nimbus.acad.ece.udel.edu [128.4.63.34])
        by mail.acad.ece.udel.edu (Postfix) with ESMTP id 3932E1ECA
        para <kissel@cis.udel.edu>; Tue, 25 Oct 2005 20:27:19 -0400 (EDT)
Fecha: Tue, 25 Oct 2005 20:27:19 -0400 (EDT)

De: Ezra Kissel <kissel@mail.acad.ece.udel.edu>
X-X-Sender: kissel@nimbus.acad.ece.udel.edu
Para: kissel@cis.udel.edu
Asunto: prueba de correo electrónico
Message-ID: <Pine.LNX.4.62.0510252026550.4176@nimbus.acad.ece.udel.edu>
X-Sanitizer: ¡Este mensaje ha sido desinfectado!
X-Sanitizer-URL: http://mailtools.anomy.net/
X-Sanitizer-Rev: UDEL-ECECIS: Sanitizer.pm,v 1.64 2002/10/22 MIME-Version: 1.0
X-Spam-Checker-Version: SpamAssassin 3.0.4 (2005-06-05) en louie.udel.edu
X-Spam-Level:
X-Spam-Status: No, score=-3.8 required=4.1 tests=ALL_TRUSTED,BAYES_00
        autolearn=ham version=3.0.4
X-Sanitizer: ¡Este mensaje ha sido desinfectado!
X-Sanitizer-URL: http://mailtools.anomy.net/
X-Sanitizer-Rev: UDEL-ECECIS: Sanitizer.pm,v 1.64 2002/10/22 MIME-Version: 1.0
Versión MIME: 1.0
Tipo de contenido: TEXT/PLAIN; charset="US-ASCII"; format=flowed
Estado: RO
Estado: V
```

Protocolos de acceso al correo



- SMTP: entrega/almacenamiento en el servidor del receptor
- protocolo de acceso al correo: recuperación del servidor
  - POP: Post Office Protocol [RFC 1939]: autorización, descarga
  - IMAP: Internet Mail Access Protocol [RFC 1730]: más funciones, incluida la manipulación de los mensajes almacenados en el servidor
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

Protocolo POP3

fase de autorización

- comandos del cliente:
  - `usuario:` declara el nombre de usuario
  - `pass:` contraseña
- respuestas del servidor
  - `+OK`
  - `-ERR`

fase de transacción, cliente:

- `list:` lista de números de mensajes
- `retr:` recuperar mensaje por número
- `dele:` borrar
- `dejar de`

```
S: +OK Servidor POP3 listo
C: usuario bob
S: +OK
C: pasar hambre
S: +OK usuario conectado con
éxito
C: lista
S: 1 498
S: 2 912
S: .
C: retr 1
S: <contenido del
mensaje 1>
S: .
C: dele 1
C: retr 2
S: <contenido del
mensaje 1>
S: .
C: dele 2
C: abandonar
S: +OK Cierre del
servidor POP3
```


POP3 (más) e IMAP

más sobre POP3

- el ejemplo anterior utiliza el modo "descargar y borrar" de POP3
  - Bob no puede releer el correo electrónico si cambia de cliente
- POP3 "download-and-keep": copias de mensajes en diferentes clientes
- POP3 no tiene estado en las sesiones

IMAP

- mantiene todos los mensajes en un solo lugar: en el servidor
- permite al usuario organizar los mensajes en carpetas
- mantiene el estado del usuario a través de las sesiones:
  - los nombres de las carpetas y las correspondencias entre los ID de los mensajes y el nombre de la carpeta

Capítulo 2: esquema

2.1 principios de las aplicaciones de red
- arquitecturas de aplicaciones
- requisitos de la aplicación

2.2 Web y HTTP

2.3 FTP

2.4 correo electrónico
- SMTP, POP3, IMAP

2.5 DNS

2.6 Aplicaciones P2P

2.7 programación de sockets con UDP y TCP

Extensiones de correo de Internet multipropósito (MIME)

MIME

- RFC importantes
  - RFC-822 Norma para el formato de los mensajes de prueba de Internet de ARPA
  - RFC-2045 MIME Parte 1: Formato de los cuerpos de los mensajes de Internet
  - RFC-2046 MIME Parte 2: Tipos de medios
  - RFC-2047 MIME Parte 3: Extensiones de encabezados de mensajes
  - RFC-2048 MIME Parte 4: Procedimiento de registro
  - RFC-2049 MIME Parte 5: Criterios de conformidad

MIME ¿Qué es?

- MIME es una norma oficial de Internet que especifica cómo deben formatearse los mensajes para que puedan intercambiarse entre diferentes sistemas de correo electrónico.
- MIME permite incluir prácticamente cualquier tipo de archivo o documento en un mensaje de correo electrónico.
- En concreto, los mensajes MIME pueden contener
  - texto
  - imágenes
  - audio
  - video
  - datos específicos de la aplicación.
    - hojas de cálculo
    - documentos de tratamiento de textos

Características de MIME

- Soporte de conjuntos de caracteres distintos de ASCII
- Sistema de etiquetado del tipo de contenido
- Soporte de contenido no textual en los mensajes de correo electrónico
- Apoyo a los documentos compuestos

Soporte del juego de caracteres no ASCII

- Encabezado del mensaje
  - campo de tipo de contenido
    - que el programa cliente que crea el correo electrónico pone en la cabecera para que lo utilice el programa cliente que se utiliza para mostrar el mensaje recibido
    - charset= parámetro opcional
      - si se asume la ausencia de ASCII
- Content-Type: text/plain; charset="ISO-8859-1"
  - ISO-8859-1 amplía el conjunto de caracteres básicos de ASCII para incluir muchos de los caracteres acentuados que se utilizan en idiomas como el español, el francés y el alemán.
  - US-ASCII es el conjunto de caracteres estándar utilizado en Estados Unidos

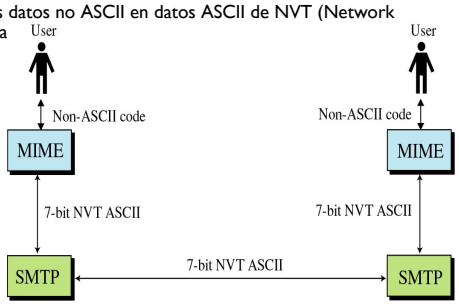
Etiquetado de contenidos

- un conjunto de tipos MIME registrados que se corresponden con tipos de archivo específicos
  - Los tipos MIME consisten en :
    - un tipo primario
    - un subtipo separado por un / (como text/html)
- Tipos de Mime comunes:

FileExtension	Tipo MIME	Descripción
.txt	texto/plano	Texto sin formato
.htm	text/html	Texto estilizado en formato HTML
Imagen	.jpg/jpeg	Imagen en formato JPEG
.gif	image/gif	Imagen en formato GIF
.wav	audio/x-wave	Sonido en formato WAVE
.mp3	audio/mpeg	Música en formato MP3
.mpg	video/mpeg	Video en formato MPEG
.zip	aplicación/zip	Archivo comprimido en formato PK-ZIP

Solución: Extensiones SMTP

- MIME - Extensiones de correo de Internet multipropósito
  - Transforma los datos no ASCII en datos ASCII de NVT (Network Virtual Termina
    - Texto
    - Aplicación
    - Imagen
    - Audio
    - Vídeo



RFC 1425, 1426, 1521

ejemplo de codificación base64

codificado en base64:

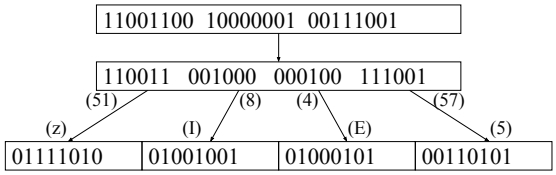
```
TWFuIGlzIGRp[illegible]aGlzIHNpbmd1bGFyIHBhc3Npb24gZnJvbSBvdGh1ciBhbmltYWxzLCB3aGljaCBpcyBhIGx
1
c3Qgb2YgdGh1IG1pbmQsIHRoYXQgYnkgYSBwZXJzZXZlcmFuY2Ugb2YgZGVsaWdodCBpbiB0
aGUgY29udGludWVkIGFuZCBpbmRlZmF0aWdhYmx1IGdlbmVyYXRpb24gb2Yga25vd2xlZGd1
LCBleGNlZWRzIHRoZSBzaG9ydCB2ZWhlbWuY2Ugb2YgW55IGNhcm5hbCBBGVhc3VyZS4=
....
```

Codificación de valores de 6 bits (codificación base-64).

6-bit value	ASCII char	6-bit value	ASCII char	6-bit value	ASCII char	6-bit value	ASCII char
0	A	10	Q	20	g	30	w
1	B	11	R	21	h	31	x
2	C	12	S	22	i	32	y
3	D	13	T	23	j	33	z
4	E	14	U	24	k	34	0
5	F	15	V	25	l	35	1
6	G	16	W	26	m	36	2
7	H	17	X	27	n	37	3
8	I	18	Y	28	o	38	4
9	J	19	Z	29	p	39	5
a	K	1a	a	2a	q	3a	6
b	L	1b	b	2b	r	3b	7
c	M	1c	c	2c	s	3c	8
d	N	1d	d	2d	t	3d	9
e	O	1e	e	2e	u	3e	+
f	P	1f	f	2f	v	3f	/

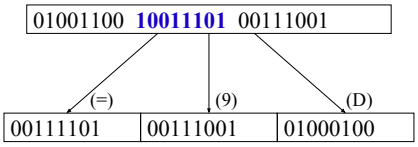
Codificación Base64

- Divide los datos binarios en bloques de 24 bits
- Cada bloque se divide en trozos de 6 bits
- Cada sección de 6 bits se interpreta como un carácter, con un 25% de sobrecarga



Codificación de citas imprimibles

- Se utiliza cuando los datos tienen una pequeña parte no ASCII
- Los caracteres no ASCII se envían como 3 caracteres
- El primero es '=', el segundo y el tercero son la representación hexadecimal del byte



DNS: sistema de nombres de dominio

personas: muchos identificadores:
- SSN, nombre, número de pasaporte

Hosts de Internet, routers:
- Dirección IP (32 bits) - se utiliza para direccionar los datagramas
- "nombre", por ejemplo, www.yahoo.com - utilizado por los humanos

P: ¿Cómo se puede asignar una dirección IP a un

Sistema de nombres de dominio:
- *base de datos distribuida* implementada en una jerarquía de muchos *servidores de nombres*
- *protocolo de la capa de aplicación:* los hosts y los servidores de nombres se comunican para *resolver* los nombres (traducción dirección/nombre)
  - nota: función central de Internet, implementada como protocolo de la capa de aplicación

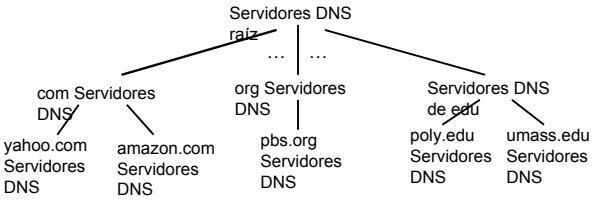
DNS: servicios, estructura

Servicios DNS
- traducción de nombre de host a dirección IP
- aliasing de host
  - canónico, nombres de alias
- alias del servidor de correo
- distribución de la carga
  - servidores web replicados: muchas direcciones IP corresponden a un solo nombre

¿por qué no centralizar el DNS?
- punto único de fallo
- volumen de tráfico
- base de datos centralizada y distante
- mantenimiento

¡no escala!

DNS: una base de datos distribuida y jerárquica



el cliente quiere la IP para www.amazon.com; 1° aprox:
- el cliente consulta el servidor raíz para encontrar el servidor com DNS
- el cliente consulta el servidor DNS de .com para obtener el servidor DNS de amazon.com
- el cliente consulta el servidor DNS de amazon.com para obtener la dirección IP de www.amazon.com

DNS: servidores de nombres raíz

- contactado por el servidor de nombres local que no puede resolver el nombre
- servidor de nombres raíz:
  - contacta con el servidor de nombres autoritativo si no se conoce la asignación de nombres
  - obtiene el mapeo
  - devuelve la asignación al servidor de nombres local

13 nombre raíz "servidores" en todo el mundo

TLD, servidores autorizados

servidores de dominios de nivel superior (TLD):
- responsable de com, org, net, edu, aero, jobs, museums, y todos los dominios de nivel superior de países, por ejemplo: uk, fr, ca, jp
- Network Solutions mantiene los servidores del dominio .com
- Educause para el dominio .edu

servidores DNS autoritativos:
- el servidor(es) DNS de la organización, que proporciona(n) asignaciones autorizadas de nombre de host a IP para los hosts con nombre de la organización
- puede ser mantenido por la organización o el proveedor de servicios

Servidor de nombres DNS local

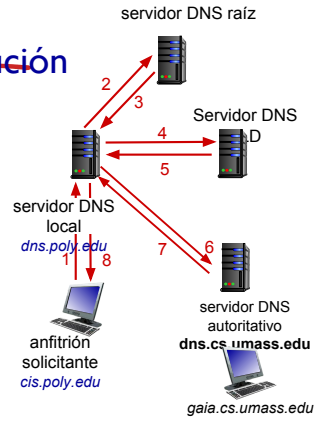
- no pertenece estrictamente a la jerarquía
- cada ISP (ISP residencial, empresa, universidad) tiene una
  - también llamado "servidor de nombres por defecto"
- cuando el host realiza una consulta DNS, la consulta se envía a su servidor DNS local
  - tiene un caché local de pares de traducción nombre-dirección recientes (¡pero puede estar desactualizado!)
  - actúa como proxy, reenvía la consulta a la jerarquía

Ejemplo de resolución de nombres DNS ejemplo de resolución

- host at cis.poly.edu quiere la dirección IP de gaia.cs.umass.edu

consulta iterada:

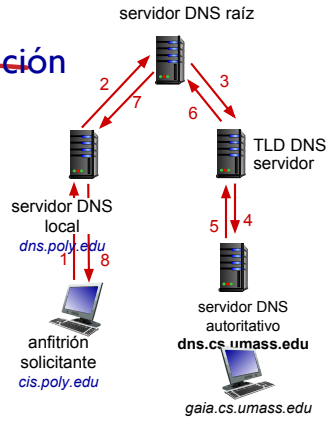
- el servidor contactado responde con el nombre del servidor a contactar
- "No conozco este nombre, pero pregunta a este servidor"



Ejemplo de resolución de nombres DNS ejemplo de resolución

consulta recursiva:

- pone la carga de la resolución de nombres en el servidor de nombres contactado
- ¿una carga pesada en los niveles superiores de la jerarquía?



DNS: almacenamiento en caché, actualización de registros

- una vez que (cualquier) servidor de nombres aprende el mapeo, *almacena en caché* el mapeo
  - las entradas de la caché expiran (desaparecen) después de un tiempo (TTL)
  - Los servidores de TLD suelen almacenarse en caché en los servidores de nombres locales
    - por lo que los servidores de nombres raíz no se visitan a menudo
- las entradas en caché pueden estar *desactualizadas* (¡traducción de nombre a dirección al máximo!)
  - si el nombre del host cambia de dirección IP, puede que no se conozca en todo Internet hasta que expiren todos los TTL
- actualizar/notificar los mecanismos propuestos por la norma IETF

Registros DNS

DNS: base de datos distribuida que almacena registros de recursos (RR)

formato RR: `(nombre, valor, tipo, ttl)`

tipo=A

- `el nombre` es hostname
- el `valor` es la dirección IP

tipo=NS

- `el nombre` es el dominio (por ejemplo, foo.com)
- es el nombre del servidor de nombres autorizado para este dominio

tipo=CNAME

- `name` es el nombre del alias de algún nombre "canónico" (el real)
- `www.ibm.com` es realmente `servereast.backup2.ibm.com`
- el `valor` es el nombre canónico

tipo=MX

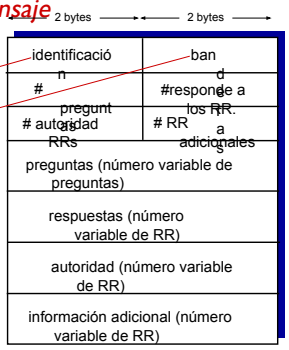
- el `valor` es el nombre del servidor de correo asociado al `nombre`

Protocolo DNS, mensajes

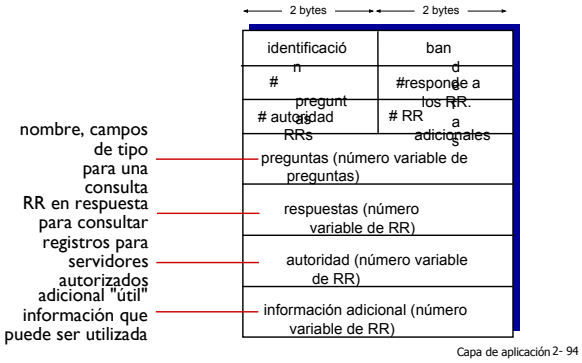
- mensajes de *consulta* y *respuesta*, ambos con el mismo *formato de mensaje*

encabezado del mensaje

- identificación: 16 bits # para la consulta, la respuesta a la consulta utiliza el mismo #
- banderas:
  - consulta o respuesta
  - Recurrencia deseada
  - Recurrencia disponible
  - la respuesta es autorizada



Protocolo DNS, mensajes



Inserción de registros en el DNS

- ejemplo: la nueva empresa "Network Utopia"
- registrar el nombre networkuptopia.com en el *registrador de DNS* (por ejemplo, Network Solutions)
  - proporcionar los nombres y las direcciones IP de los servidores de nombres autorizados (primario y secundario)
  - registrar inserts two RRs into .com TLD server:
    `(networkutopia.com, dns1.networkutopia.com, NS)`
    `(dns1.networkutopia.com, 212.212.212.1, A)`
- crear un registro A de tipo servidor autoritativo para www.networkuptopia.com; registro MX de tipo para networkutopia.com

Ataque al DNS

Ataques DDoS

- Bombardear los servidores raíz con tráfico
  - Sin éxito hasta la fecha
  - Filtrado del tráfico
  - Los servidores DNS locales almacenan en caché las IP de los servidores TLD, lo que permite eludir el servidor raíz
- Bombardear los servidores de TLD
  - Potencialmente más

Ataques de redirección

- Hombre en el medio
  - Consultas de interceptación
- Envenenamiento del DNS
  - Enviar falsos relevos al servidor DNS, que almacena en caché

Explotar el DNS para el DDoS

- Enviar consultas con dirección de origen falsa: IP de destino

Capítulo 2: esquema

2.1 principios de las aplicaciones de red
- arquitecturas de aplicaciones
- requisitos de la aplicación

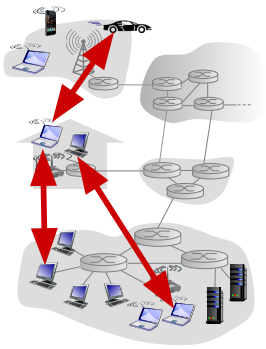
2.2 Web y HTTP
2.3 FTP
2.4 correo electrónico
- SMTP, POP3, IMAP

2.5 DNS

2.6 Aplicaciones P2P
2.7 programación de sockets con UDP y TCP

Arquitectura P2P pura

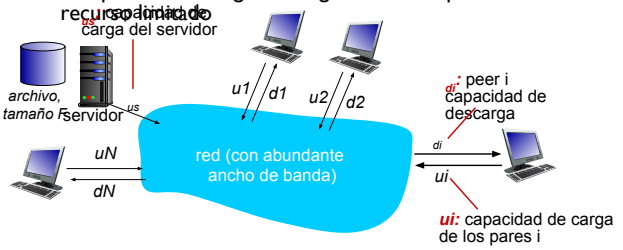
- *no hay* un servidor siempre activo
- los sistemas finales arbitrarios se comunican directamente
- los compañeros se conectan de forma intermitente y cambian de dirección IP



ejemplos:
- distribución de archivos (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)

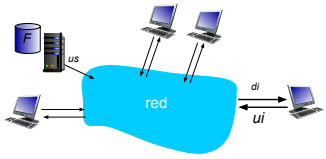
Distribución de archivos: cliente-servidor vs P2P

Pregunta: ¿cuánto tiempo se tarda en distribuir un archivo (tamaño *F*) desde un servidor a *N compañeros*?
- la capacidad de carga/descarga de los compañeros es un recurso limitado



Tiempo de distribución de archivos: cliente-servidor

- *transmisión al servidor:* debe enviar secuencialmente (cargar) *N* copias de archivos:
  - tiempo para enviar un ejemplar: *F/us*
  - tiempo para enviar N copias: *NF/us*
- *cliente:* cada cliente debe descargar una copia del archivo
  - dmin = tasa de descarga mínima del cliente
  - Tiempo mínimo de descarga del cliente: *F/dmin*

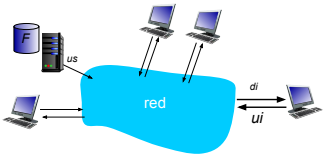


tiempo para distribuir F a N clientes utilizando enfoque cliente-servidor $D_{c-s} > \max\{NF/u_s, F/d_{min}\}$

aumenta linealmente en N

Tiempo de distribución de archivos: P2P

- *transmisión al servidor:* debe cargar al menos una copia
  - tiempo para enviar un ejemplar: *F/us*
- *cliente:* cada cliente debe descargar una copia del archivo
  - Tiempo mínimo de descarga del cliente: F/dmin
- *clientes:* como agregado debe descargar los bits *NF*
  - La tasa máxima de carga (limitando la tasa máxima de descarga) es $u_s + \sum u_i$



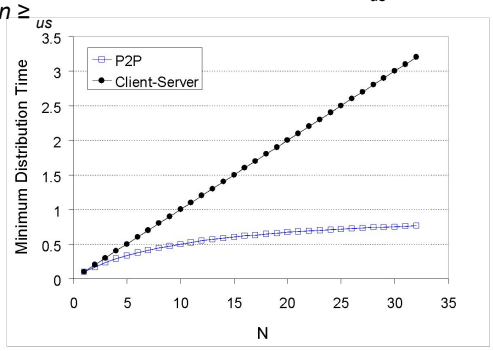
tiempo para distribuir F a N clientes utilizando Enfoque P2P $D_{P2P} > \max\{F/u_s, F/d_{min}, NF/(u_s + \sum u_i)\}$

aumenta linealmente en N

... pero también esto, ya que cada compañero aporta capacidad de servicio

Cliente-servidor vs. P2P: ejemplo

tasa de carga del cliente = *u*, *F/u* = 1 hora, $u_s = 10u$, $d_{min} \geq u_s$

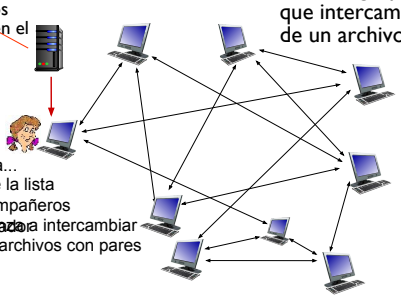


Distribución de archivos P2P: BitTorrent

- archivo dividido en trozos de 256Kb
- los pares en torrent envían/reciben trozos de archivos

tracker: rastrea a los compañeros participar en el torrente

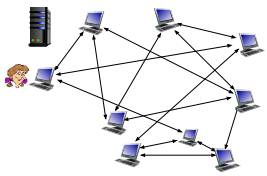
torrent: grupo de pares que intercambian trozos de un archivo



Alice llega...
... obtiene la lista de los compañeros del rastreador
... comienza a intercambiar trozos de archivos con pares en torrent

Distribución de archivos P2P: BitTorrent

- unión de pares torrent:
  - no tiene chunks, pero los acumulará con el tiempo de otros compañeros
  - se registra en el rastreador para obtener una lista de pares, se conecta a un subconjunto de pares ("vecinos")



- mientras se descarga, los compañeros suben trozos a otros compañeros
- el par puede cambiar los pares con los que intercambia trozos
- *churn:* los compañeros pueden ir y venir
- una vez que el par tiene el archivo completo, puede (egoístamente) salir o (altruistamente) permanecer en el torrent

BitTorrent: solicitar, enviar trozos de archivos

solicitando trozos:

- en un momento dado, diferentes pares tienen diferentes subconjuntos de trozos de archivos
- periódicamente, Alice pide a cada compañero la lista de los chunks que tiene
- Alice solicita los trozos que faltan a sus compañeros, los más raros primero

envío de trozos: tit-for-tat

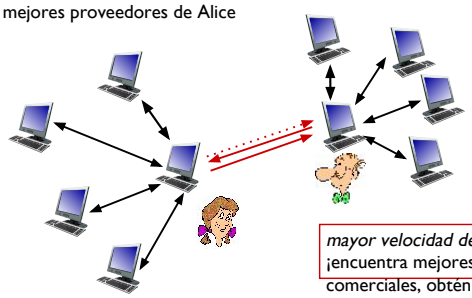
- Alice envía trozos a los cuatro compañeros que actualmente le envían trozos *a la mayor velocidad*
  - los demás compañeros son ahogados por Alice (no reciben trozos de ella)
  - reevaluar los 4 primeros cada 10 segundos
- cada 30 segundos: selecciona aleatoriamente otro peer, comienza a enviar chunks
  - "descobijar con optimismo" a este compañero
  - los nuevos elegidos pueden unirse a los 4 primeros

BitTorrent: ojo por ojo

(1) Alicia "desencadena con optimismo" a Bob

(2) Alice se convierte en uno de los cuatro mejores proveedores de Bob; Bob le corresponde

(3) Bob se convierte en uno de los cuatro mejores proveedores de Alice



mayor velocidad de carga: ¡encuentra mejores socios comerciales, obtén los archivos más rápido!

Tabla Hash Distribuida (DHT)

- Tabla Hash
- El paradigma de la DHT
- DHT circular y redes superpuestas
- Rotación de los compañeros

Base de datos simple

Base de datos simple con pares (clave, valor):

- clave: nombre humano; valor: seguridad social

Clave	Valor
John Washington	132-54-3570
Diana Louise Jones	761-55-3791
Xiaoming Liu	385-41-0902
Rakesh Gopal	441-89-1956
Linda Cohen	217-66-5609
.....	.....
Lisa Kobayashi	177-23-0199

- clave: título de la película; valor: Dirección IP

Tabla Hash

- Más conveniente para almacenar y buscar en la representación numérica de la clave
- clave = hash(clave original)

Clave original	Clave	Valor
John Washington	8962458	132-54-3570
Diana Louise Jones	7800356	761-55-3791
Xiaoming Liu	1567109	385-41-0902
Rakesh Gopal	2360012	441-89-1956
Linda Cohen	5430938	217-66-5609
.....		.....
Lisa Kobayashi	9290124	177-23-0199

Tabla Hash Distribuida (DHT)

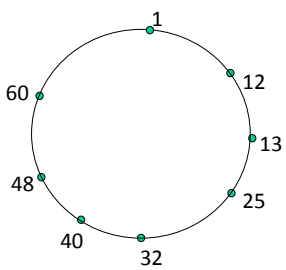
- Distribuir pares (clave, valor) entre millones de pares
  - los pares se distribuyen uniformemente entre los pares
- Cualquier compañero puede consultar la base de datos con una clave
  - la base de datos devuelve el valor de la clave
  - Para resolver la consulta, se intercambia un pequeño número de mensajes entre pares
- Cada compañero sólo conoce un pequeño número de otros compañeros
- Robustez ante la entrada y salida de compañeros (churn)

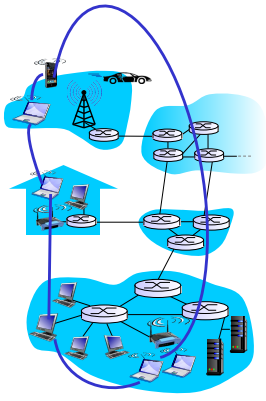
Asignar pares clave-valor a los pares

- regla: asignar el par clave-valor al par que tenga el ID *más cercano*.
- convención: el más cercano es el *sucesor inmediato* de la llave.
- Por ejemplo, el espacio ID {0,1,2,3,...,63}
- suponga 8 compañeros: 1,12,13,25,32,40,48,60
  - Si la clave = 51, entonces se asigna al par 60
  - Si la clave = 60, entonces se asigna al par 60
  - Si la clave = 61, entonces se asigna al par 1

DHT circular

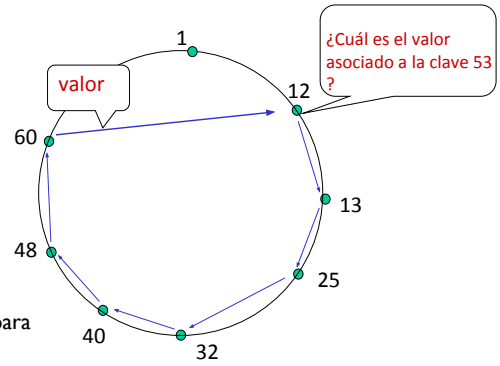
- cada par *sólo* conoce a su sucesor inmediato y predecesor.





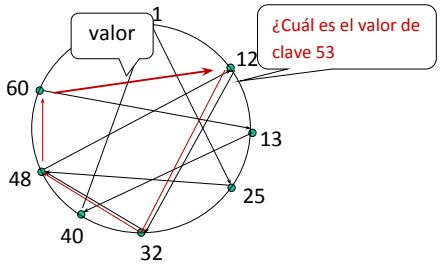
"red superpuesta"

Resolver una consulta



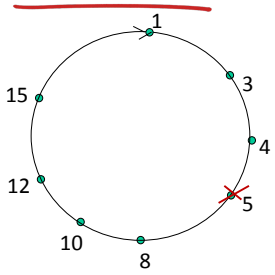
O(N) mensajes en el promedio para resolver consulta, cuando hay con *N* compañeros

DHT circular con accesos directos



- cada par mantiene un registro de las direcciones IP del predecesor, del sucesor y de los atajos.
- reducido de 6 a 3 mensajes.
- posible diseñar atajos con *O(log N)* vecinos, *O(log N)* mensajes en la consulta

Rotación de los compañeros

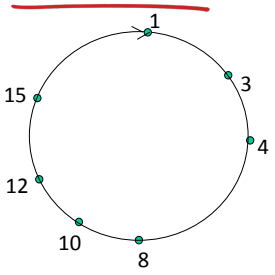


gestión de la rotación de los compañeros:

- los compañeros pueden ir y venir (churn)
- cada compañero conoce la dirección de sus dos sucesores
- cada par hace pings periódicamente a sus dos sucesores para comprobar la disponibilidad
- si el sucesor inmediato se va, elige al siguiente sucesor como nuevo sucesor inmediato

ejemplo: el compañero 5 se va abruptamente

Rotación de los compañeros



gestión de la rotación de los compañeros:

- los compañeros pueden ir y venir (churn)
- cada compañero conoce la dirección de sus dos sucesores
- cada par hace pings periódicamente a sus dos sucesores para comprobar su disponibilidad
- si el sucesor inmediato se va, elige al siguiente sucesor como nuevo sucesor inmediato

ejemplo: el compañero 5 se va abruptamente

- el par 4 detecta la salida del par 5; hace del 8 su sucesor inmediato
- 4 pregunta a 8 quién es su sucesor inmediato; hace que el sucesor inmediato de 8 sea su segundo sucesor.

Capítulo 2: esquema

2.1 principios de las aplicaciones de red
- arquitecturas de aplicaciones
- requisitos de la aplicación

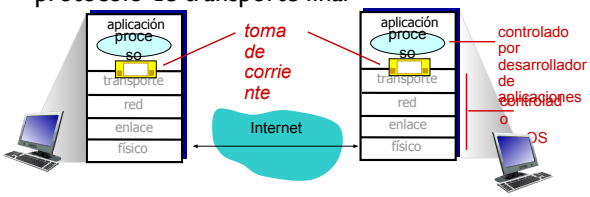
2.2 Web y HTTP
2.3 FTP
2.4 correo electrónico
- SMTP, POP3, IMAP

2.5 DNS
2.6 Aplicaciones P2P
2.7 programación de sockets con UDP y TCP

Programación de zócalos

objetivo: aprender a crear aplicaciones cliente/servidor que se comunican mediante sockets

socket: puerta entre el proceso de aplicación y el protocolo de transporte final



Programación de zócalos

Dos tipos de enchufes para dos servicios de transporte:

- *UDP:* datagrama no fiable
- *TCP:* fiable, orientado al flujo de bytes

Ejemplo de aplicación:

1. El cliente lee una línea de caracteres (datos) de su teclado y envía los datos al servidor.
2. El servidor recibe los datos y convierte los caracteres en mayúsculas.
3. El servidor envía los datos modificados al cliente.
4. El cliente recibe los datos modificados y muestra la línea en su pantalla.

Programación de sockets *con UDP*

UDP: no hay "conexión" entre el cliente y el servidor

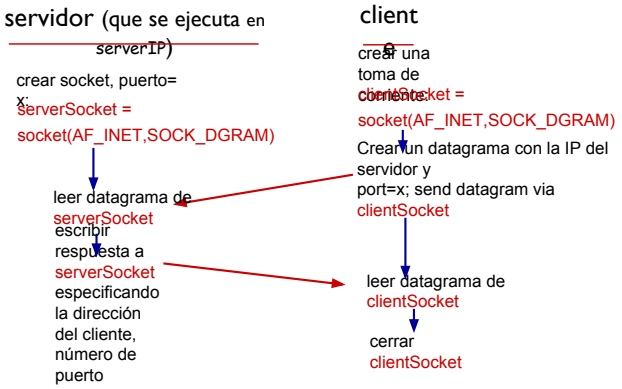
- no hay handshaking antes de enviar los datos
- el remitente adjunta explícitamente la dirección IP de destino y el número de puerto a cada paquete
- rcvr extrae la dirección IP del rer tente y el número de puerto del paquete recibido

UDP: los datos transmitidos pueden perderse o recibirse fuera de orden

Punto de vista de la aplicación:

- UDP proporciona una transferencia *poco fiable* de grupos de bytes ("datagramas") entre el cliente y el servidor

Interacción de socket cliente/servidor: UDP



Ejemplo de aplicación: Cliente UDP

Python UDPClient

incluir el socket de Python biblioteca
obtener el teclado del usuario
crear un socket UDP para el servidor
adjuntar el nombre del servidor, el puerto al mensaje; enviar al socket
leer los caracteres de respuesta de socket into string
imprimir la cadena recibida y cerrar el socket

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(socket.AF_INET,
                      socket.SOCK_DGRAM)
message = raw_input('Entrada de frase en
minúsculas:')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress =
                      clientSocket.recvfrom(2048)
print modifiedMessage
```


Ejemplo de aplicación: Servidor UDP

Python UDPServer

crear un socket UDP
enlazar el socket al puerto local número 12000
bucle para siempre
Lectura del socket UDP en el mensaje, obteniendo la dirección del cliente (IP del cliente y puerto)
enviar la cadena en mayúsculas a este cliente

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "El servidor está listo para recibir"
mientras que 1:
    mensaje, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```


Programación de sockets *con TCP*

el cliente debe contactar con el servidor

- el proceso del servidor debe ejecutarse primero
- el servidor debe haber creado un socket (puerta) que acoja el contacto del cliente

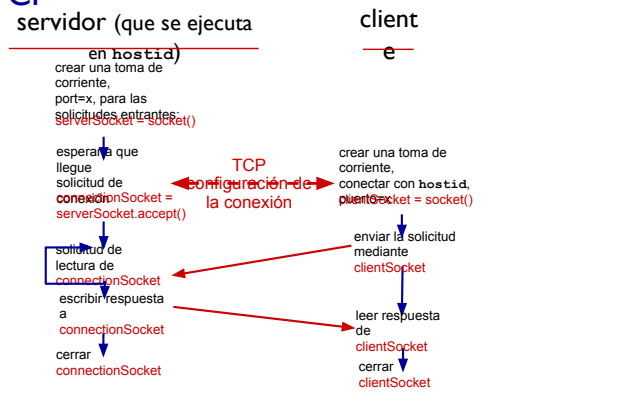
el cliente contacta con el servidor por:

- Creación de un socket TCP, especificando la dirección IP y el número de puerto del proceso del servidor
- *cuando el cliente crea un socket:* el cliente TCP establece una conexión con el servidor TCP

- cuando es contactado por el cliente, *el servidor TCP crea un nuevo socket* para que el proceso del servidor se comunique con ese cliente en particular
  - permite al servidor hablar con varios clientes
  - números de puerto de origen utilizados para distinguir a los clientes (más información en el capítulo 3)

punto de vista de la aplicación:
"CP proporciona un servicio fiable y en orden transferencia de flujo de bytes ("pipe") entre el cliente y el servidor

Interacción de socket cliente/servidor: TCP



Ejemplo de aplicación: Cliente TCP

Python TCPClient

crear un socket TCP para el servidor, puerto remoto 12000
No es necesario adjuntar el nombre del servidor, el puerto

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
frase = raw_input('Entrada de frase en
minúsculas:')
clientSocket.send(sentencia)
modifiedSentence = clientSocket.recv(1024)
print 'Desde el servidor:', modifiedSentence
clientSocket.close()
```


Ejemplo de aplicación: Servidor TCP

Python TCPServer

crear una bienvenida TCP toma de corriente
el servidor comienza a escuchar las peticiones TCP entrantes
bucle para siempre
el servidor espera en accept()
para las solicitudes entrantes, nuevo socket creado a la vuelta
leer bytes del socket (pero no la dirección como en UDP)
cerrar la conexión con este cliente (pero *no* dar la bienvenida al socket)

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'El servidor está listo para recibir'
mientras que 1:
    connectionSocket, addr = serverSocket.accept()

    sentencia = connectionSocket.recv(1024)
    frase en mayúsculas = frase.superior()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```


Capítulo 2: resumen

nuestro estudio de las aplicaciones de red ya está completo

- arquitecturas de aplicación
  - cliente-servidor
  - P2P
- requisitos de los servicios de aplicación:
  - fiabilidad, ancho de banda, retraso
- Modelo de servicio de transporte por Internet
  - orientado a la conexión, fiable: TCP
  - poco fiables, los datagramas: UDP
- protocolos específicos:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent, DHT
- Programación de sockets: Sockets TCP, UDP

Capítulo 2: resumen

lo más importante: ¡aprendí sobre los protocolos!

- intercambio típico de mensajes de solicitud/respuesta:
  - el cliente solicita información o servicio
  - el servidor responde con datos, código de estado
- formatos de mensajes:
  - cabeceras: campos que dan información sobre los datos
  - datos: información que se comunica

temas importantes:

- mensajes de control vs. datos
  - dentro de la banda, fuera de la banda
- centralizado vs. descentralizado
- sin estado vs. con estado
- transferencia de mensajes fiable o no fiable
- "complejidad en el borde de la red"