Capa de Aplicación

(2da parte)

Mg.Ing.Miguel Solinas

miguel.solinas@unc.edu.ar

Capítulo 2: hoja de ruta

1. Principio de aplicaciones de red
   a) Arquitectura de las aplicaciones
   b) Sus requerimientos
2. Web y HTTP
3. FTP
4. Correo electrónico
   a) SMTP, POP3, IMAP
5. DNS
6. Aplicaciones P2P
7. Programación de socket con UDP y TCP

DNS: Domain Name System

Las Personas tenemos varios identificadores:

- SSN, nombre, número de DNI, etc…

Internet hosts, routers:

- Dirección IP (32 bit) – utilizada para identificar datagramas
- "nombre", Ej.:, www.yahoo.com, lo utilizan las personas..!!

P: ¿ Cómo mapear nombres con IP y viceversa ?

DNS: Domain Name System

Domain Name System:

- *DB distribuida* implementada como una jerarquía de varios *servidores DN*
- *Protocolo capa aplicación:* Los hosts se comunican con los "name servers" para *resolver* nombres (traducción address/name)
  - Función del core de Internet, implementada como protocolo
  - Complejidad del borde de la red

DNS: Servicios, Estructura

¿ Una Base de Datos distribuida ?

¿ Por qué no un único DNS centralizado ?

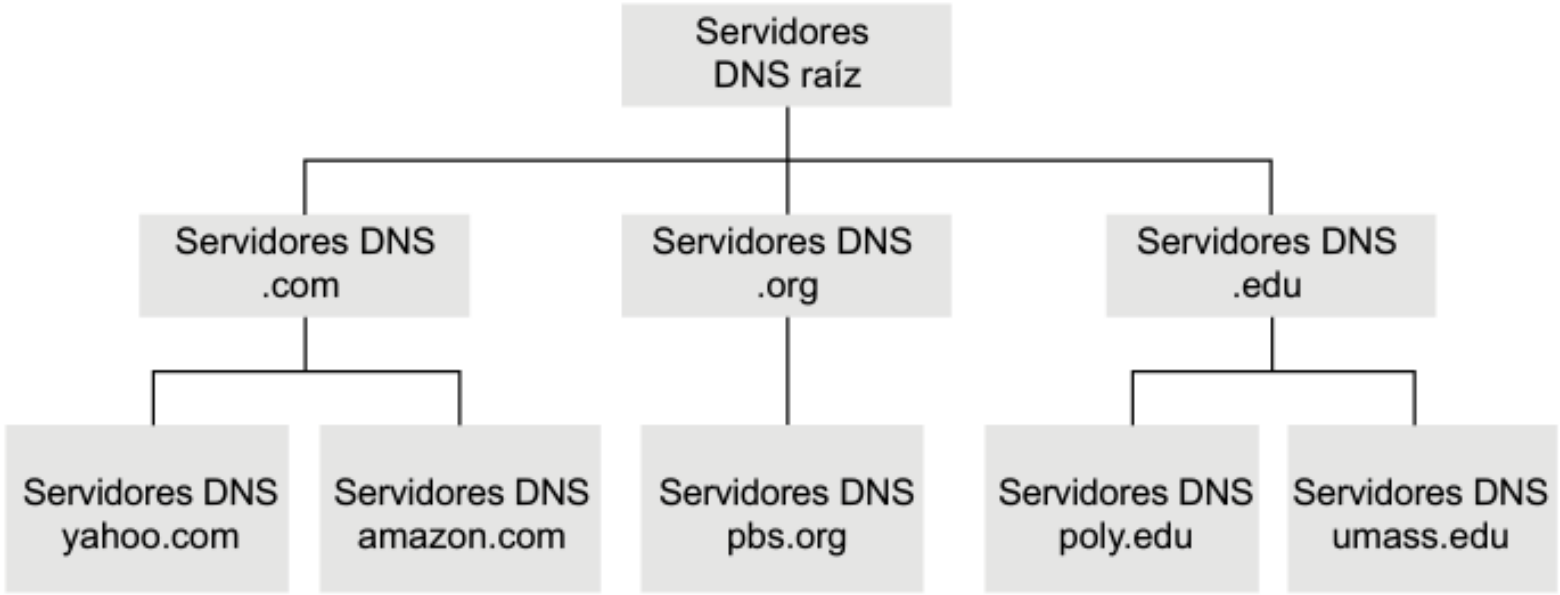
- Único punto de falla
- Volumen de tráfico
- DB centralizada distante
- Mantenimiento

DNS: Servicios, Estructura

Servicios de DNS

- Traducción de dirección IP a nombres de dominio
- Alias de host
  - Nombre canónico, nombres de alias
- Alias de servidores de correo
- Distribución de carga
  - Servidores web replicados : ¿ qué hacer cuando varias direcciones IP se corresponden a un único nombre de dominio ?

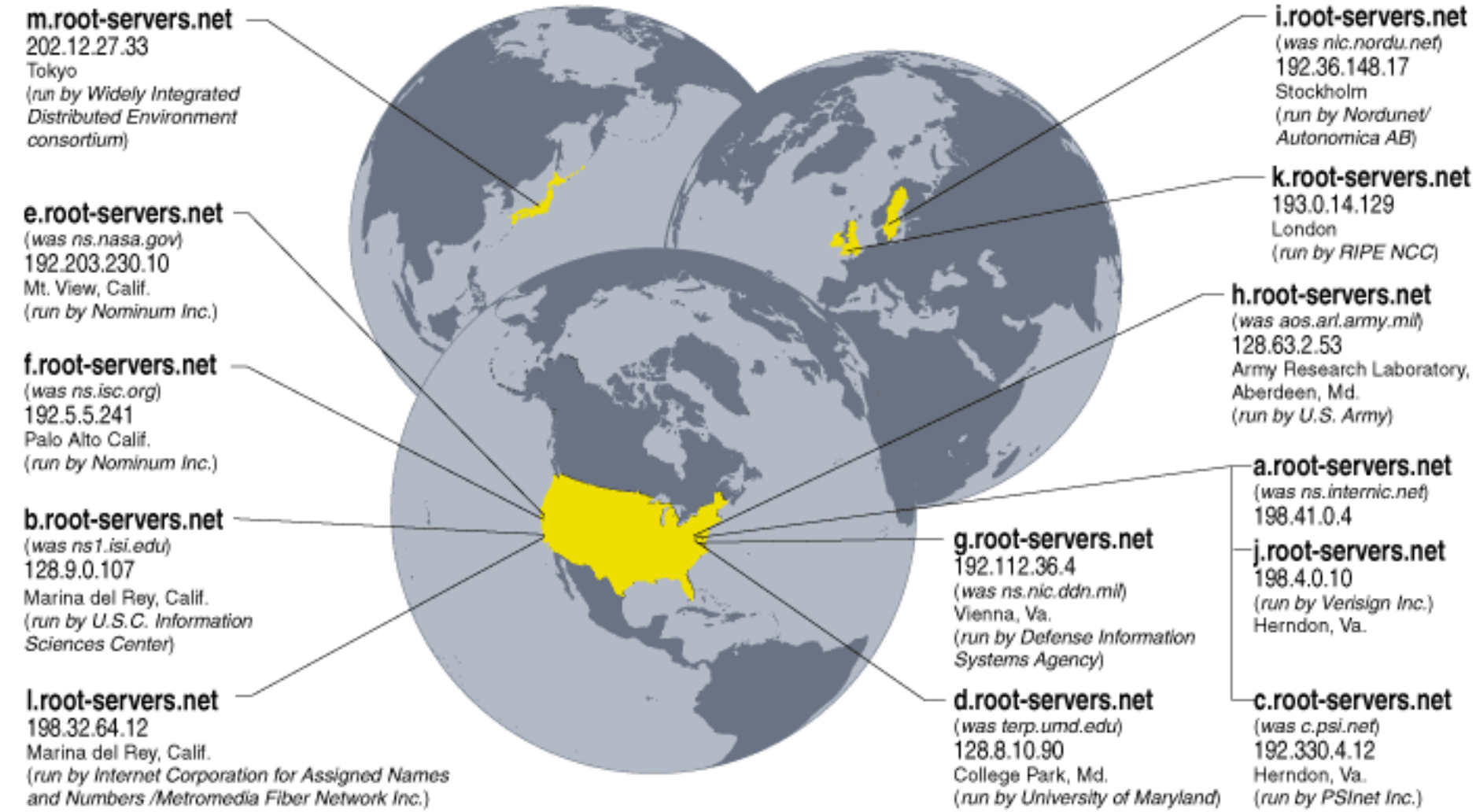
DNS: Una DB jerárquica distribuida



Necesito dirección IP para "www.amazon.com" (1st aproximación)

- Cliente consulta un "root server" para obtener IP del ".com DNS server"
- Cliente consulta ".com DNS server" para obtener IP del "amazon.com DNS server"
- Cliente consulta "amazon.com DNS server" para obtener IP de "www.amazon.com"

DNS: root name servers



DNS: root name servers

- Son contactados por los DNS server locales que no pueden resolver un nombre de dominio
- root name server:
  - Contacta DNS server autoritativo (TLD) si no conoce nombre de dominio
  - Obtiene un mapa para la consulta
  - Devuelve el “mapa” al DNS server local

TLD, authoritative servers

Top-Level Domain (TLD) servers:

- Responsables por los dominios com, org, net, edu, aero, jobs, museums, y todos los asociados a paises Ej.: uk, fr, ca, jp.
- Network Solutions mantiene servers para .com TLD
- Educause para .edu TLD

Authoritative DNS servers:

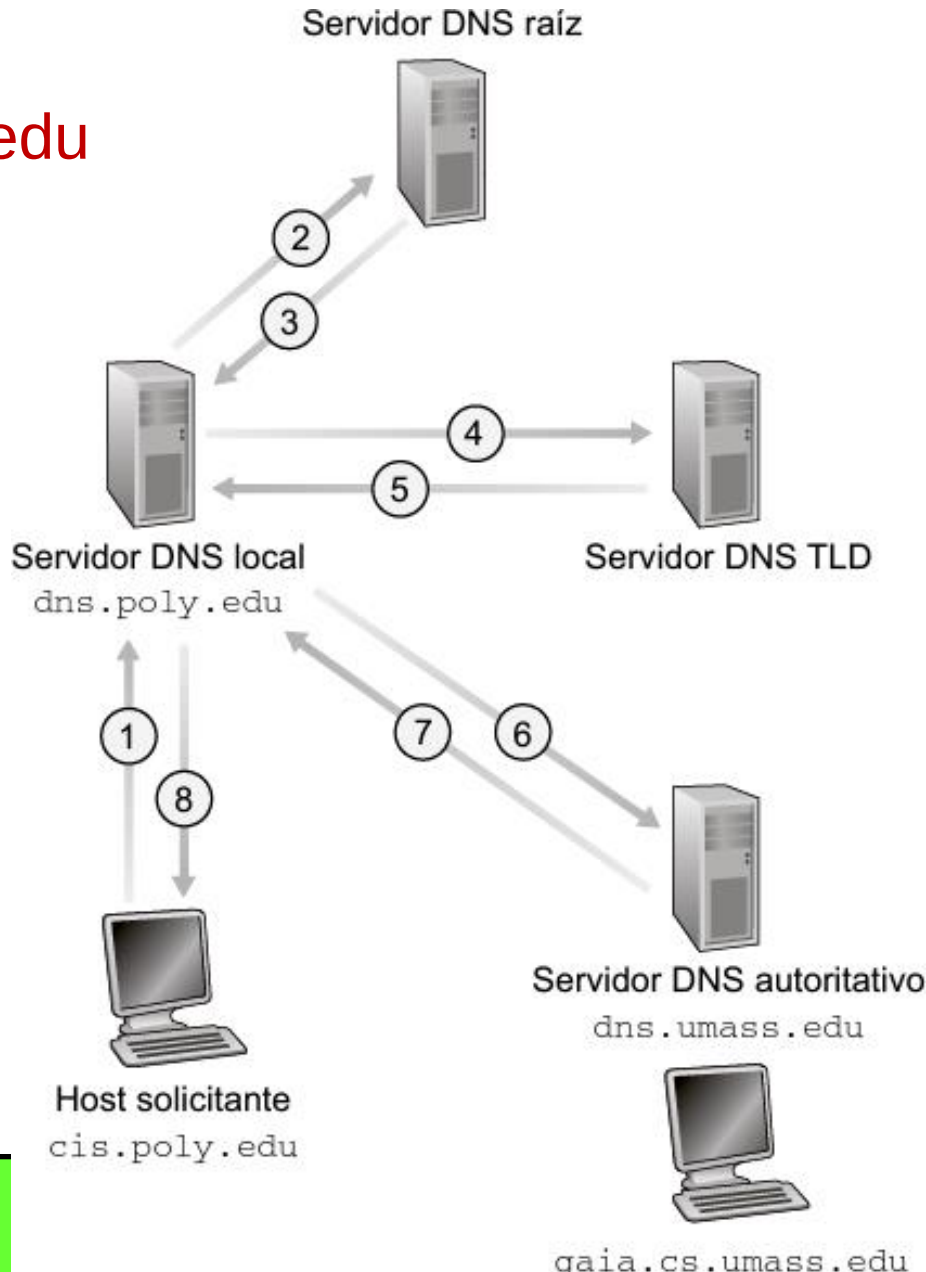
- DNS servers de las propias organizaciones. Proveen nombres con autoridad para mapear las direcciones IP de los nombres de host de cada organización.
- Pueden ser mantenidos por la propia organización o pueden tercerizar en los ISP.

Local DNS name server

- No pertenecen estrictamente a la jerarquía
- Cada ISP (residential ISP, company, university) tiene uno
  - El conocido “default name server”
- Cuando un host hace una consulta a un DNS server, la consulta es enviada a uno de estos servidores
  - Tienen un cache local de los pares nombre/IP de las traducciones recientemente hechas (¿ desactualizadas ?)
  - Actuan como un proxy, reenviando consultas a la jerarquía.

Ej.: Resolución de nombre DNS

- Host en cis.poly.edu necesita dirección IP de gaia.cs.umass.edu



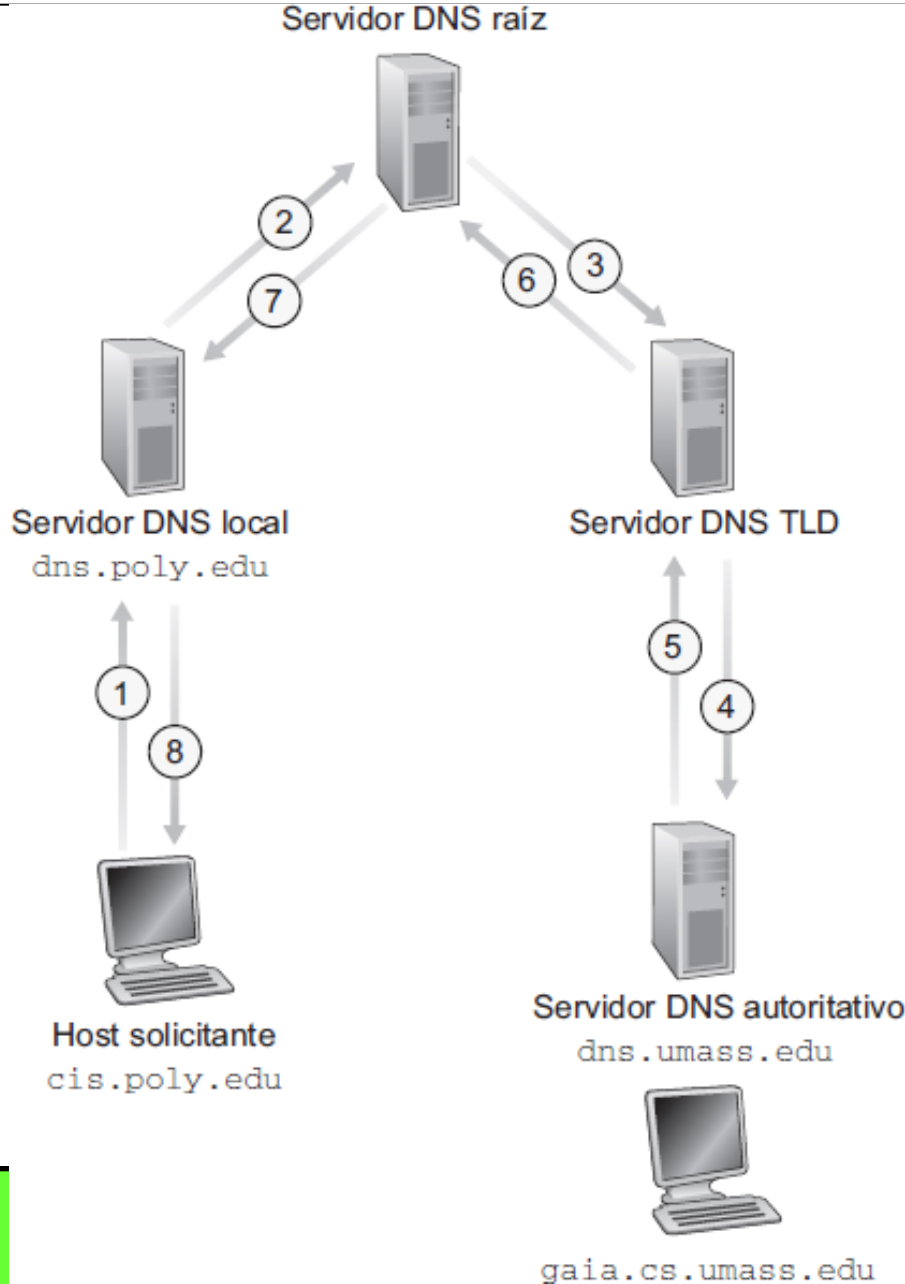
Una consulta iterativa:

- Servers contactados responden con el nombre del server a contactar
- "Yo no conozco este dominio pero pregúntele a este server.!!"

DNS name resolution example

Una Consulta recursiva:

- Pone la carga de la resolución de nombres en el DSN server contactado
- Carga pesada para niveles superiores de jerarquía



DNS: caching, updating records

- Una vez que un DNS Server (cualquiera) aprende el mapeo, pone en *cache* el mapeo.
  - Las entradas del cache desaparecen después de cierto tiempo (TTL)
  - Servidores TLD guardan los nombres locales
    - De este modo los root name servers no se visitan frecuentemente.
- Las entradas en cache pueden estar *out-of-date* (best effort name-to-address translation!)
  - Si un nombre de host cambia la dirección IP, puede que no sea conocido hasta que expire el TTL.
- RFC 2136 propone mecanismos de actualización y notificación.

Registros DNS

DNS: DB distribuida que almacena "Resource Records" (RR)

RR format: **(name, value, type, ttl)**

type = A

- **name** is hostname
- **value** is IP address

type = NS

- **name** is domain (e.g., foo.com)
- **value** is hostname of authoritative name server for this domain

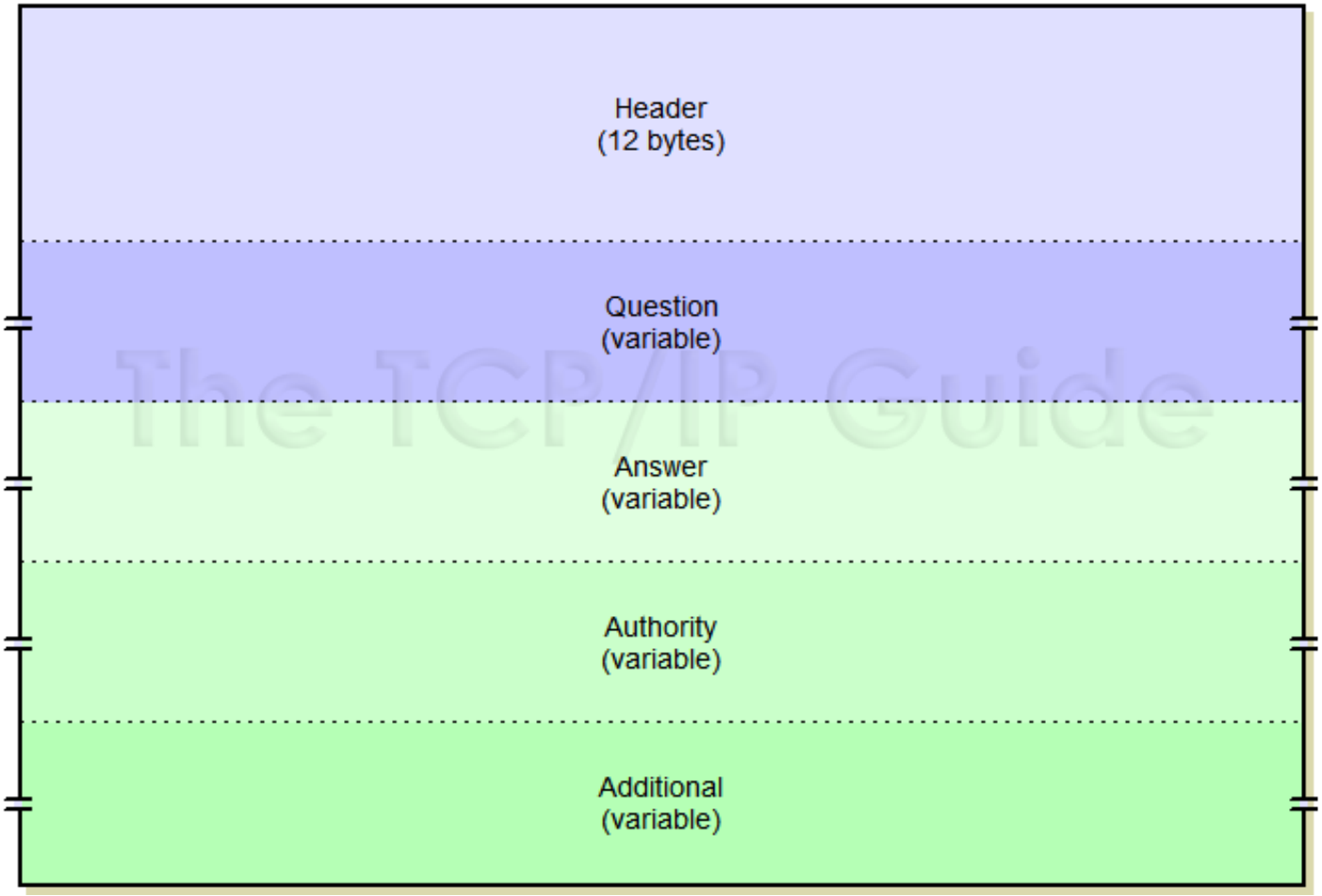
type = CNAME

- **name** is alias name for some "canonical" (the real) name
- **www.ibm.com** is really **servereast.backup2.ibm.com**
- **value** is canonical name

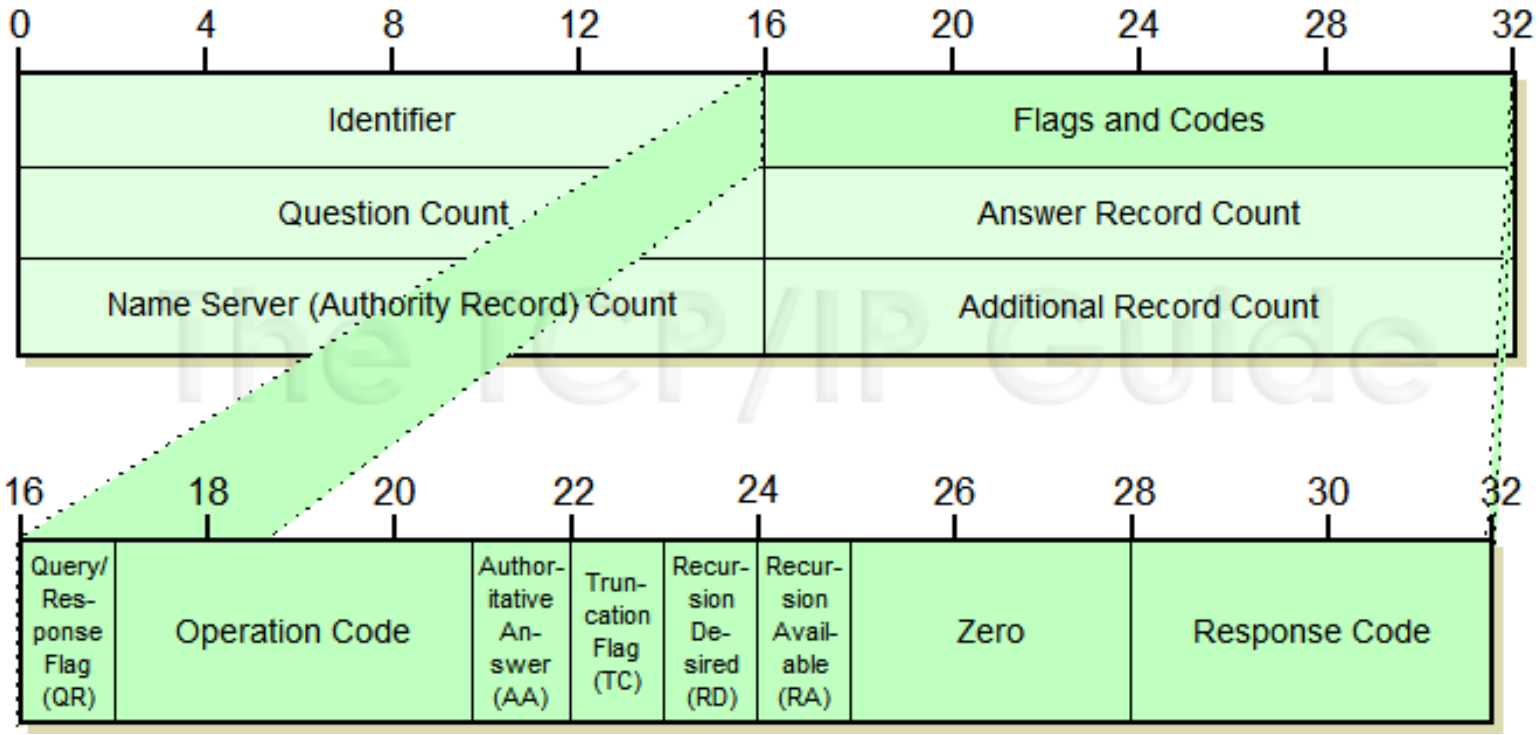
type = MX

- **value** is name of mail server associated with **name**

Protocolo DNS



Protocolo DNS

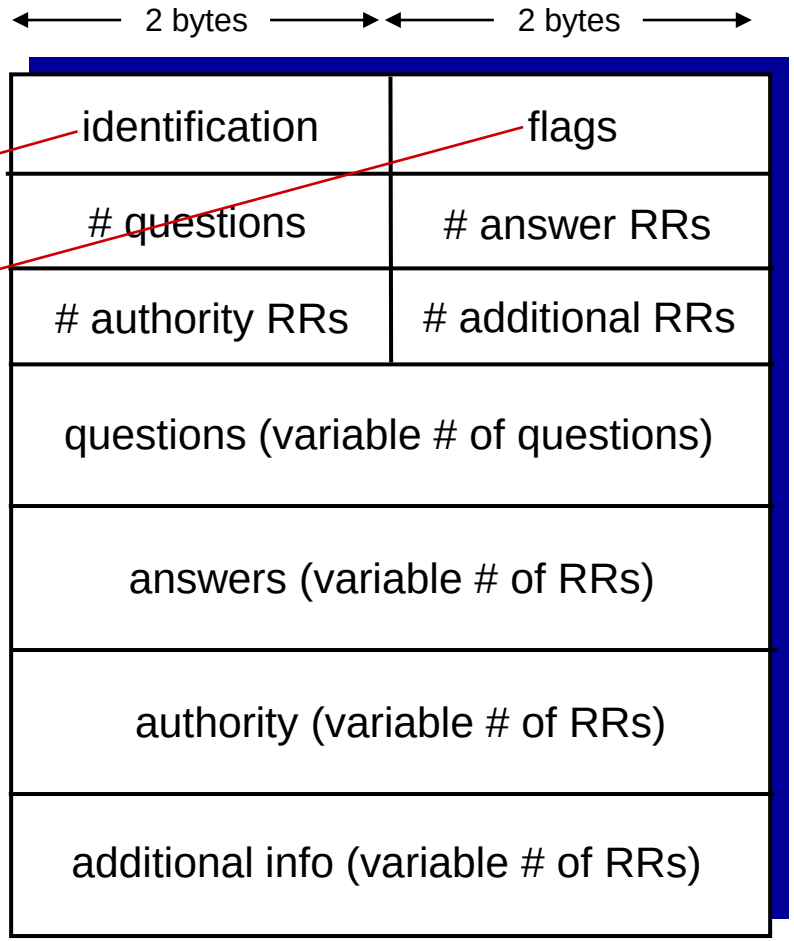


Protocolo DNS, mensajes

- Mensajes de *query* and *reply*, ambos comparten el mismo *formato de mensaje*

Cabecera del mensaje

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



Inserción de registros en DNS

Ej.: Nuevo emprendimiento "Network Utopia"

- Hay que registrar el dominio networkuptopia.com en *DNS registrar* (Ej.: Network Solutions)
  - Provee nombre de dominio, dirección IP de DNS server autoritativo (primario y secundario)
  - "registrar" inserta dos RRs en los .com TLD server:
    (networkutopia.com, dns1.networkutopia.com, NS)

    (dns1.networkutopia.com, 212.212.212.1, A)
- Debería crear un registro tipo A en un servidor autoritativo for www.networkuptopia.com; y otro registro tipo MX networkutopia.com

Ataques a DNS

Ataques de DDoS

- Bombardear root servers con tráfico
  - No han sido exitosos
  - Fitros de Tráfico
  - Local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombardear TLD servers
  - Potencialmente mas dañino

Ataques de Redireccionado

- Man-in-middle
  - Intercepta consultas
- Envenamiento de DNS
  - Send bogus relies to DNS server, which caches

Exploit DNS for DDoS

- Send queries with spoofed source address: target IP
- Requires amplification

Capítulo 2: hoja de ruta

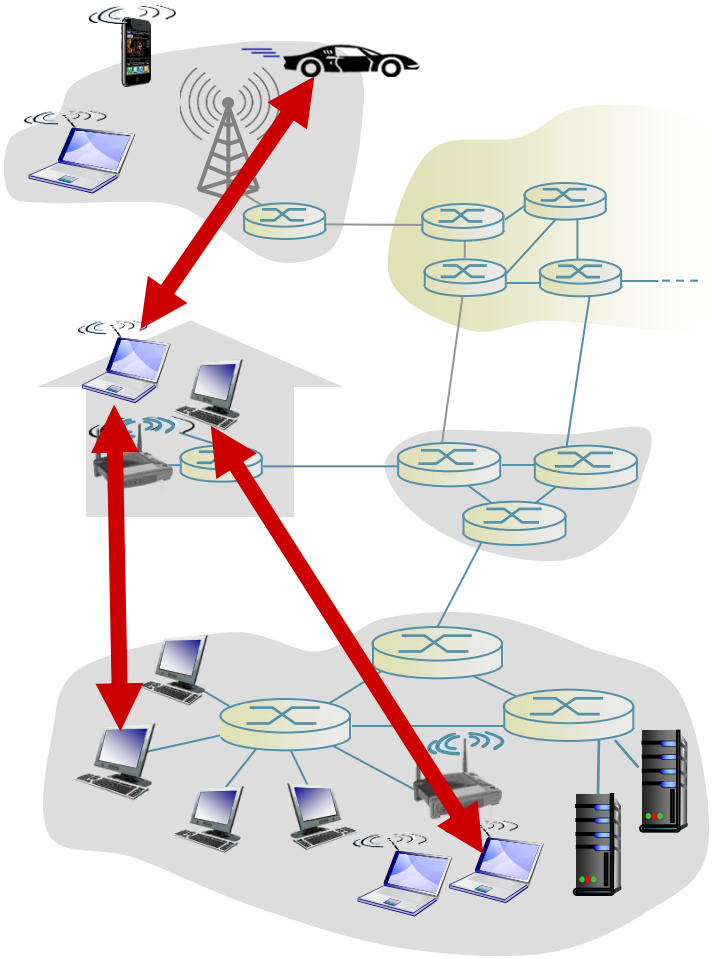
1. Principio de aplicaciones de red
   a) Arquitectura de las aplicaciones
   b) Sus requerimientos
2. Web y HTTP
3. FTP
4. Correo electrónico
   a) SMTP, POP3, IMAP
5. DNS
6. Aplicaciones P2P
7. Programación de socket con UDP y TCP

Pure P2P architecture

- *No siempre un servidor ON*
- Se comunican sistemas finales arbitrarios
- Los Peers están conectados de forma intermitente y cambian sus direcciones IP

Ejemplos:

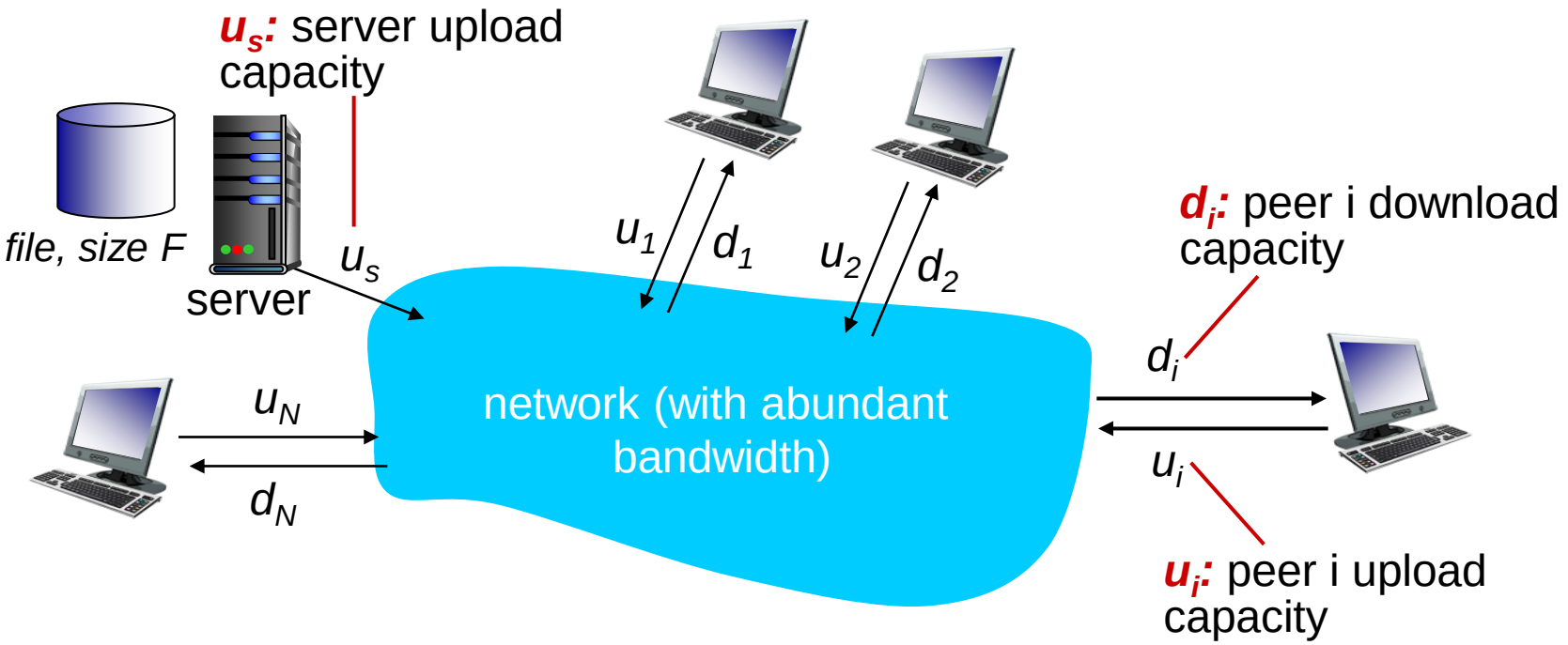
- Distribución de archivos (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)



Distribución de archivos: C/S vs P2P

Pregunta: ¿ Cuánto tiempo insume distribuir un archivo de tamaño F desde un servidor a N Peers ?

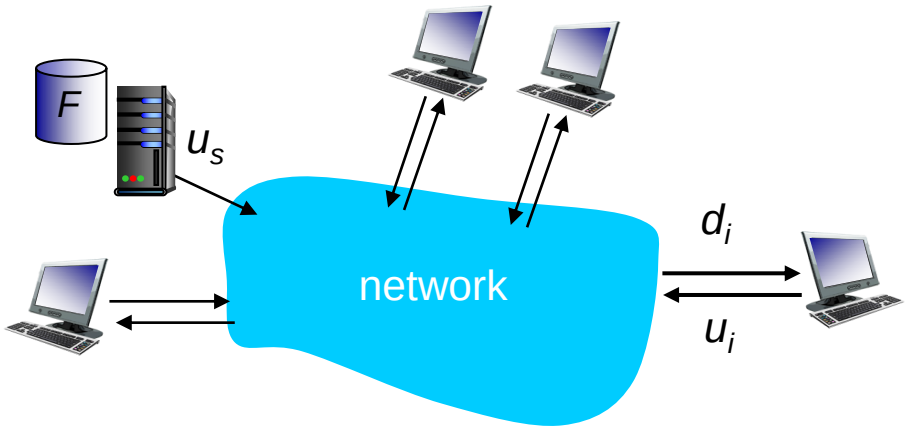
La capacidad de carga/descarga es un recurso limitado



Distribución de archivos: tiempo C/S

- *Server transmisor:* envia secuencialmente (upload) *N* copias de archivo F
  - Tiempo para enviar una copia: F/u_s
  - Tiempo para enviar N copias: $\mathrm{N}F/u_s$
- *Cliente:* cada uno debe descargar una copia del archivo F
  - Velocidad mínima de descarga: d_{min}
  - Tiempo mínimo de descarga : F/d_{min}

$$D_{c/s} \geq max\left\{\frac{\mathrm{N}F}{u_s}, \quad \frac{F}{d_{min}}\right\}$$



Distribución de archivos: tiempo P2P

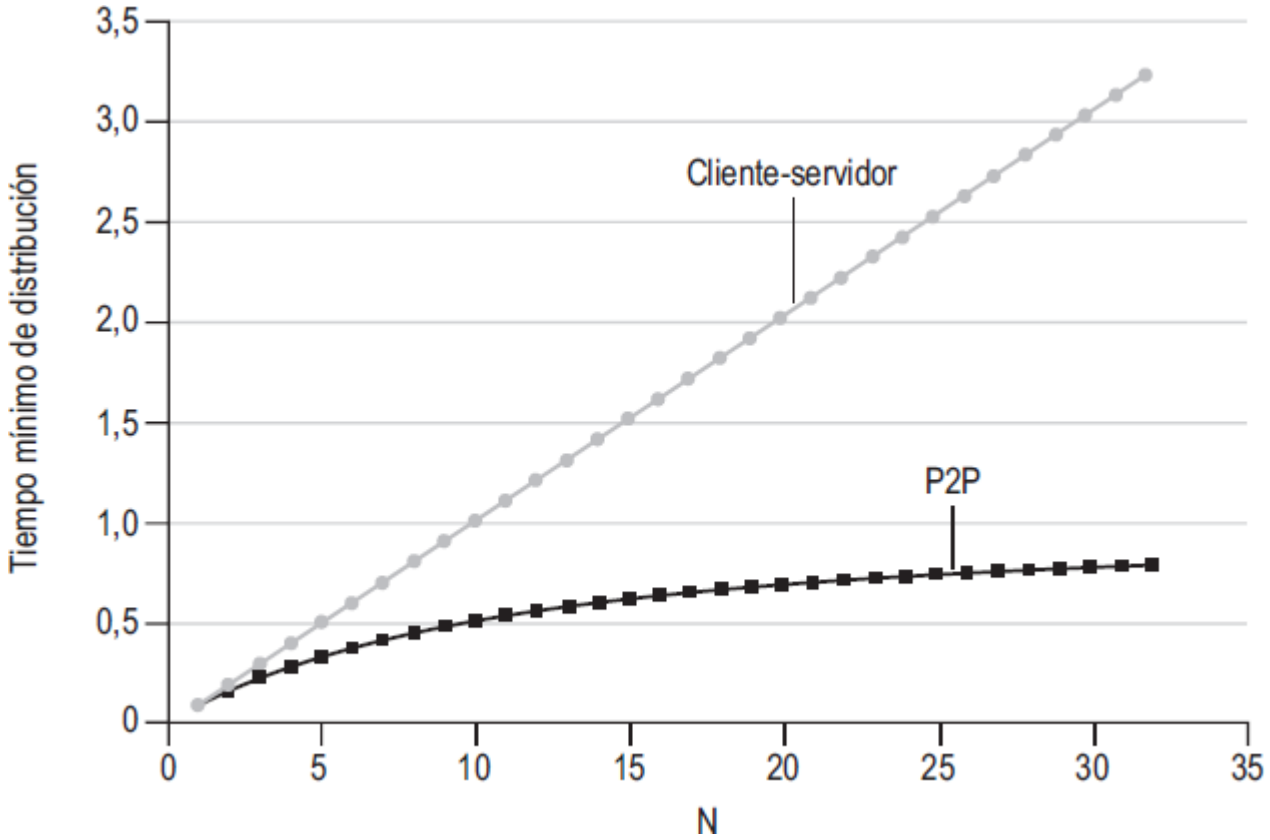
- *Server transmisor:* debe subir al menos una copia de archivo F
  - Tiempo para enviar una copia: F/u_s
- *Cliente:* cada uno descarga una copia del archivo F
  - Tiempo mínimo de descarga : F/d_{min}
- *Clientes en conjunto:* como sistema deben descargar NF bits
  - Máxima velocidad de subida (limitado por la máx velocidad de bajada) es la suma de:

$$u_{Total} = u_s + u_1 + u_2 + \cdots + u_N = u_s + \sum_{i=1}^{N} u_i$$

$$D_{P2P} \geq max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\}$$

Ejemplo: Client-server vs. P2P

client upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$

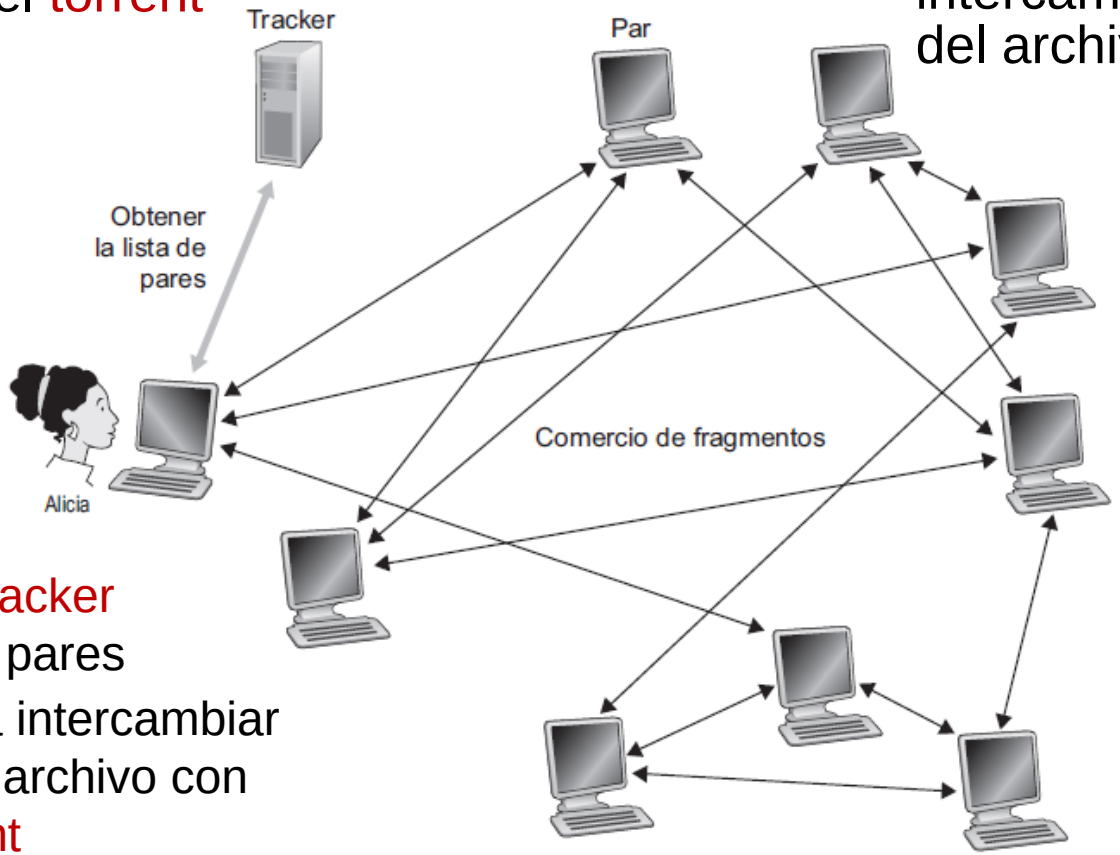


Disbribución de archivos P2P: BitTorrent

- El archivo se divide en fragmentos de 256Kb
- Los pares envían/reciben fragmentos del archivo

tracker: rastrea pares que participan en el torrent

torrent: colección de pares intercambiando fragmentos del archivo



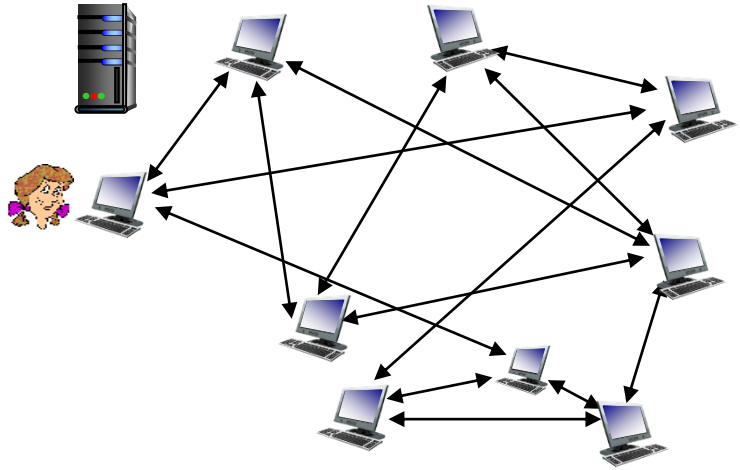
Llega Alicia …

…obtiene del tracker una lista de los pares

…y comienza a intercambiar fragmentos del archivo con pares del torrent

Disbribución de archivos P2P: BitTorrent

- Un “peer” se une a Torrent:
  - No tiene fragmentos, los obtendrá de otros pares a lo largo del tiempo
  - Se registra con el tracker para obtener una lista de pares, se conecta a un subconjunto de pares (vecinos)



- Mientras descarga, otros “peer” suben fragmentos a otros
- Los pares pueden intercambiar compañeros
- *churn:* los pares van y vienen
- Una vez que un par tiene un archivo entero, puede retirarse (egoista) o permanecer (altruisa)

BitTorrent: requesting, sending file chunks

requesting chunks:

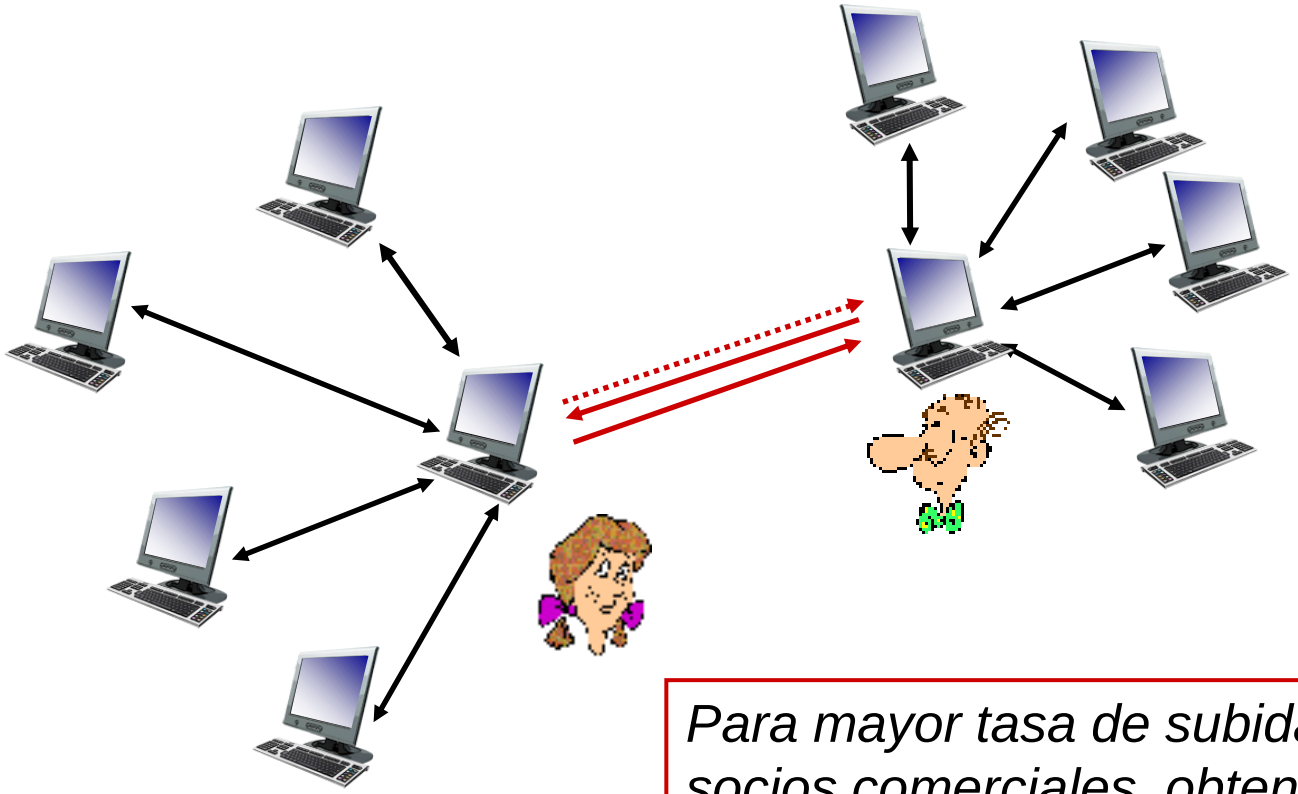
- En un momento diferentes pares tienen diferentes subconjuntos de framentos
- Periodicamente, Alicia solicita a sus pares la lista fragmentos que ellos tienen
- Alicia solicita fragmentos perdidos de los pares, primero los mas raros

sending chunks: tit-for-tat

- Alicia envía fragmentos a los cuatro pares que le envían a ella a mayor velocidad
  - Los otros pares son ignorados por Alicia (no reciben)
  - Re evalúa los mejores 4 cada 10 segundos
- Cada 30 seg: al azar elige nuevo par p/enviar fragmento
  - Tiene una actitud optimista respecto este par
  - Recien llegado puede unirse a los top 4

BitTorrent: tit-for-tat

1) Alicia “optimista no filtra” Bob
2) Alicia deviene uno de los cuatro proveedores de Bob, Bob actúa recíprocamente
3) Bob deviene en uno de los cuatro proveedores de Alicia



Para mayor tasa de subida: encontrar mejores socios comerciales, obtener archivo más rápido!

Distributed Hash Table (DHT)

- Tablas de hash
- Paradigma DHT
- DHT Circular y redes solapadas
- Retiro de peers

Una DB simple

Una DB simple tiene un par (clave, valor) :

- Ej.1: clave: nombre; valor: seguridad social#
- Ej.2: clave: título de película; valor: IP address

Key	Value
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

¿ Y una DB distribuida en un sistema P2P ?

Tablas de Hash

- Resulta mas conveniente almacenar y buscar una representación numérica de la clave
- Key = hash(original key)

Original Key	Key	Value
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

http://hash-functions.online-domain-tools.com/

Distributed Hash Table (DHT)

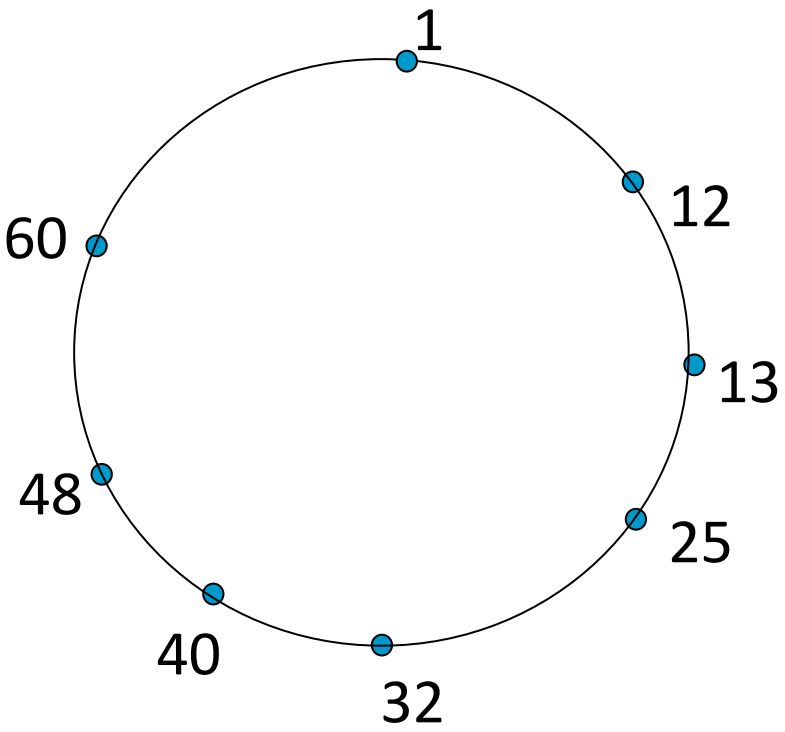
- Distribuir pares (clave, valor) sobre millones de peers
  - Los pares están distribuidos uniformemente entre peers
- Cualquier peer puede consultar la DB con una clave
  - La DB retorna el valor para esa clave
  - Para resolver la consulta, se desea un número pequeño de mensajes entre peers
- Cada peer sólo conoce un número pequeño de otros peers
- ¿ Cómo encontrar un compromiso entre número de consultas y pares almacenados ?

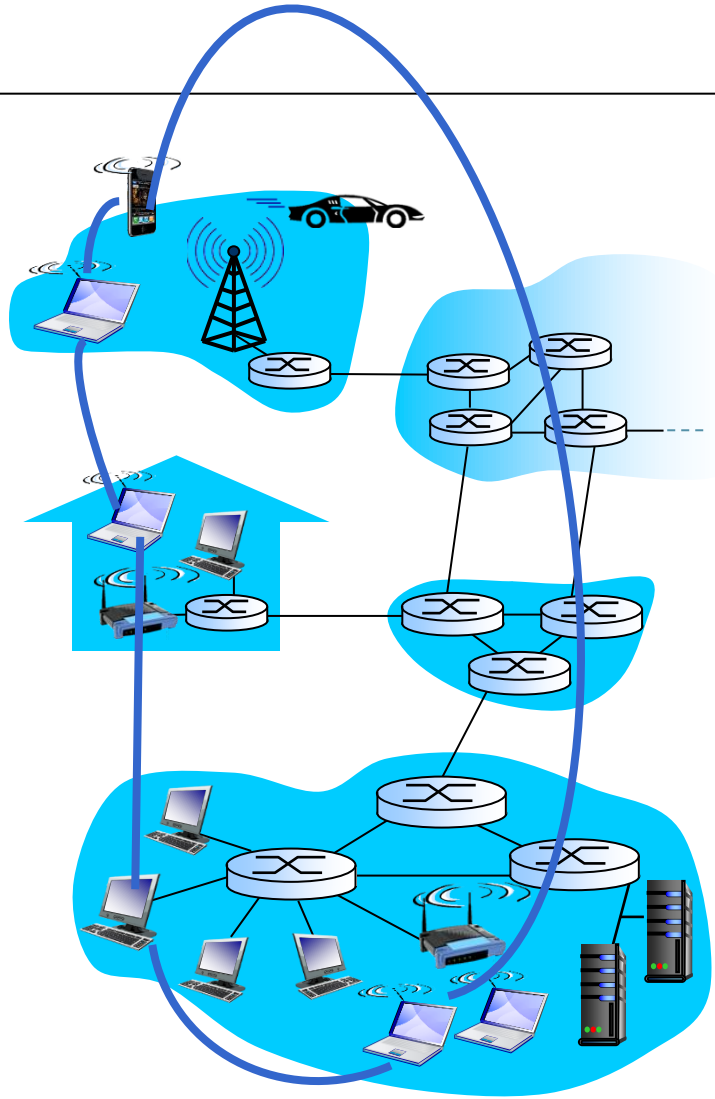
Asignando pares key-value a los peers

- Regla: asignar par key-value al peer que tiene el ID *mas próximo*.
- Convención: mas próximo es el *sucesor inmediato* de la clave.
- Ej.: Espacio de ID $\{0, 1, 2, 3, \ldots, 63\}$; $[0, 2^n - 1]$; $n = 6$
- Suponiendo 8 peers: 1, 12, 13, 25, 32, 40, 48, 60
  - If key = 51, then assigned to peer 60
  - If key = 60, then assigned to peer 60
  - If key = 61, then assigned to peer 1

DHT Circular

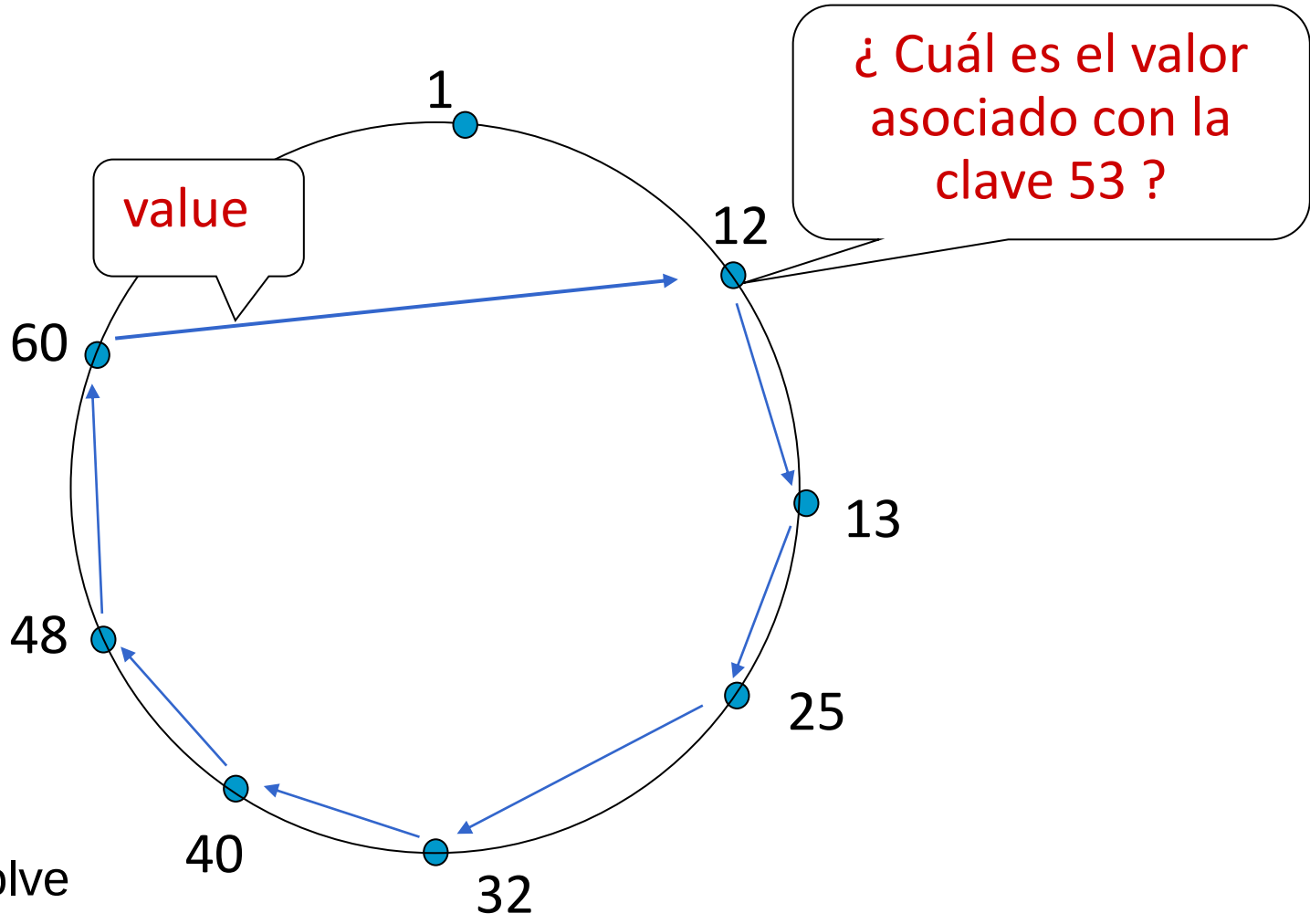
- Cada peer solo conoce su sucesor inmediato y su predecesor.





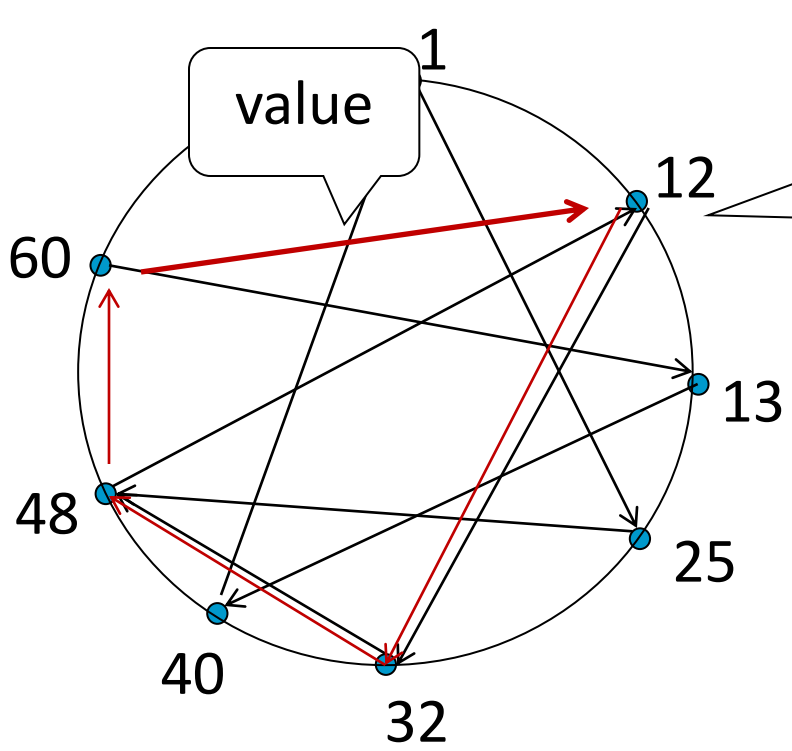
"red solapada"

Resolviendo una consulta



O(N) messages on avgerage to resolve query, when there are *N* peers

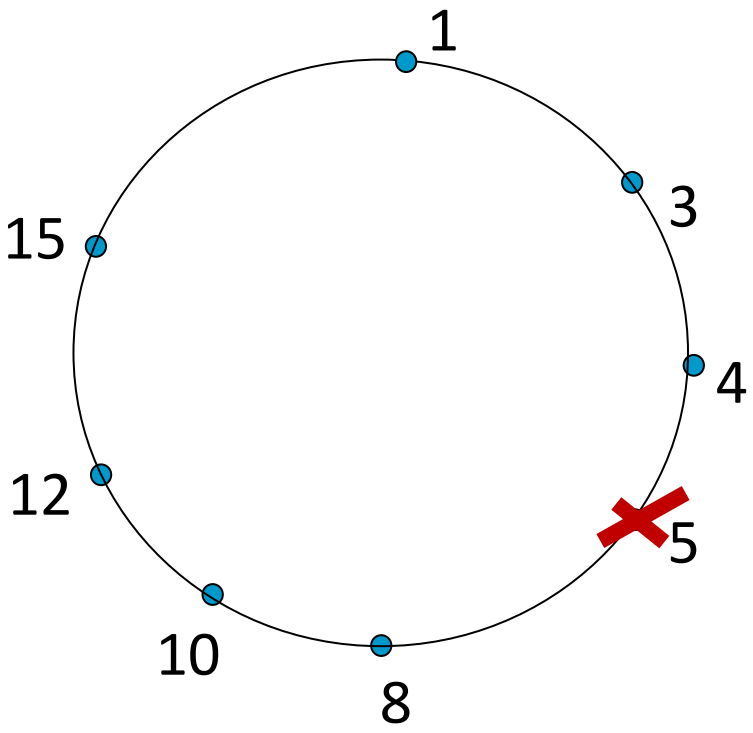
DHT circular con atajos



¿ Cuál es el valor asociado con la clave 53 ?

- Cada peer conoce la IP de predecesor, sucesor y atajos.
- Se reduce de 6 a 3 los mensajes necesarios.
- Es posible diseñar atajos con vecinos que permitan un comportamiento O(log N) mensajes en la consulta (footplot.com).

La rotación de peers

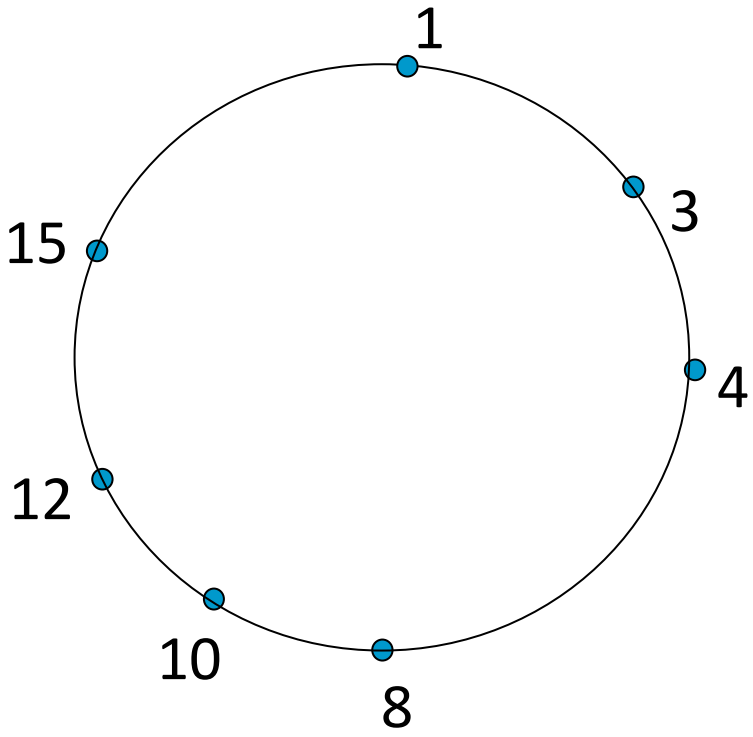


Gestionando la rotación de peers:

- Peers pueden ir y venir (churn)
- Cada peer conoce IP de dos sucerores
- Cada peer pingea periódicamente a sus dos suceres para ver si están vivos
- Si su sucesor inmediato se retira, elige como nuevo sucesor a su sucesor inmediato

Ej.: peer 5 se retira abruptamente..!!

La rotación de peers



Gestionando la rotación de peers:

- Peers pueden ir y venir (churn)
- Cada peer conoce IP de dos sucerores
- Cada peer pingea periódicamente a sus dos suceres para ver si están vivos
- Si su sucesor inmediato se retira, elige como nuevo sucesor a su sucesor inmediato

Ej.: peer 5 se retira abruptamente..!!

- Peer 4 detecta la salida de peer 5; hace a peer 8 sucesor inmediato
- Peer 4 pregunta a peer 8 quién es su sucesor; hace al sucesor de peer 8 su segundo sucesor.

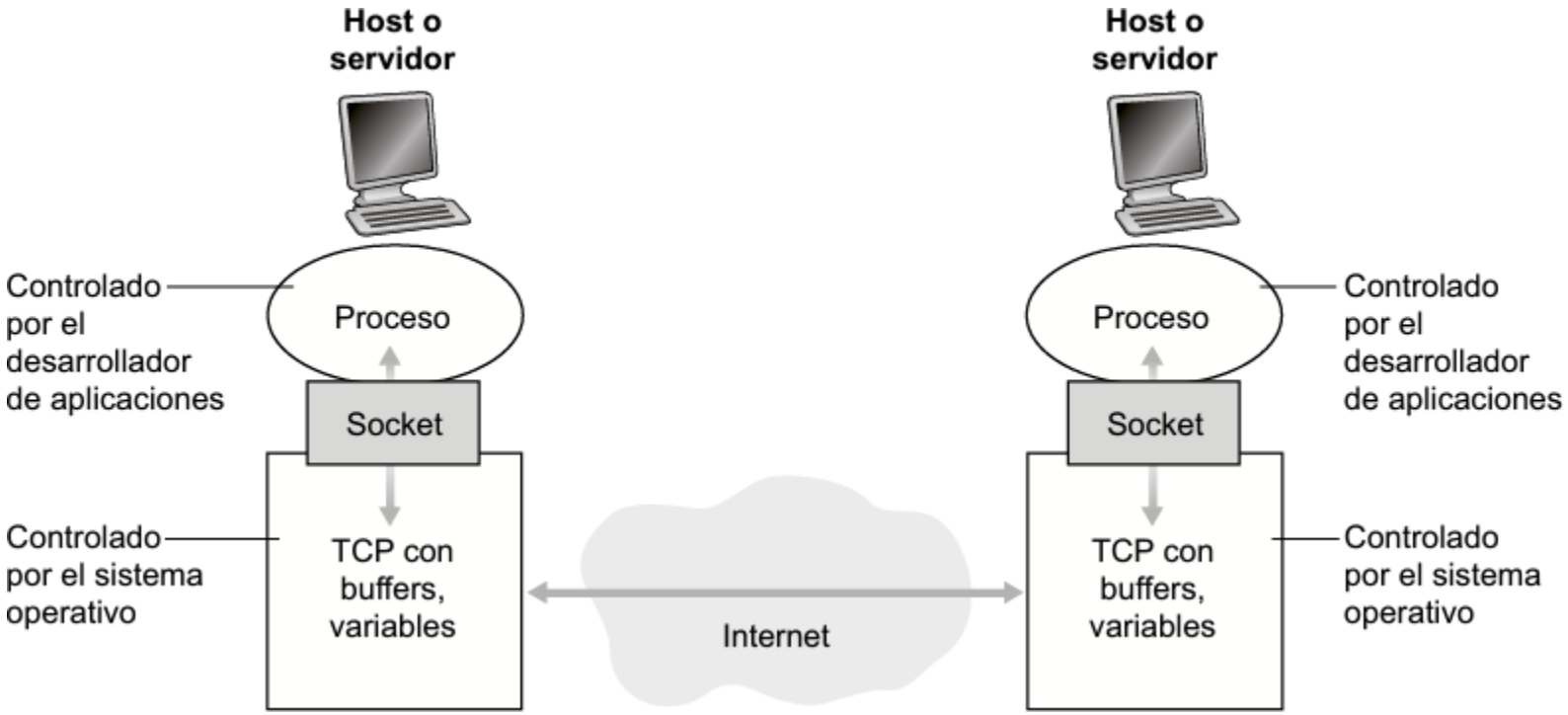
Capítulo 2: hoja de ruta

1. Principio de aplicaciones de red
   a) Arquitectura de las aplicaciones
   b) Sus requerimientos
2. Web y HTTP
3. FTP
4. Correo electrónico
   a) SMTP, POP3, IMAP
5. DNS
6. Aplicaciones P2P
7. Programación de socket con UDP y TCP

Programación de Socket

Meta: aprender a construir aplicaciones client/server que se comunican utilizando sockets

Socket: puerta entre proceso de aplicación y protocolo de transporte extremo-extremo



Programación de Socket

Dos tipos de socket para dos tipos de servicios de transporte:

- *UDP:* datagrama no confiable
- *TCP:* confiable, byte stream-oriented

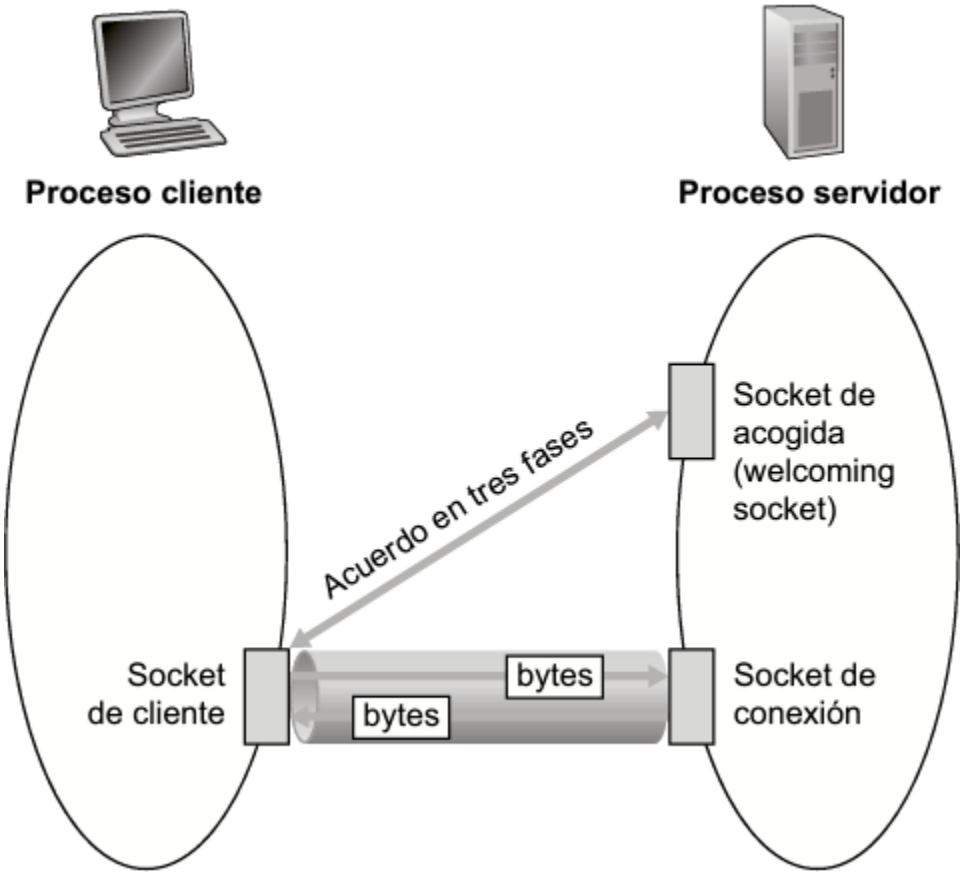
Ejemplo de una aplicación:

1. Cliente lee un línea de caracteres (datos) desde su teclado y envía los datos al servidor.
2. El servidor recibe datos y convierte letras a mayúsculas.
3. El servidor envía datos modificados al cliente.
4. El cliente recie los datos modificados y los muestra por pantalla.

Programación de Socket TCP

Meta: aprender a construir aplicaciones client/server que se comunican utilizando sockets

Socket: puerta entre proceso de aplicación y protocolo de transporte extremo-extremo



Programación de Socket TCP

Cliente debe contactar al Server, luego:

- Proceso Server debe estar corriendo
- Proceso Server tiene un método de bienvenida

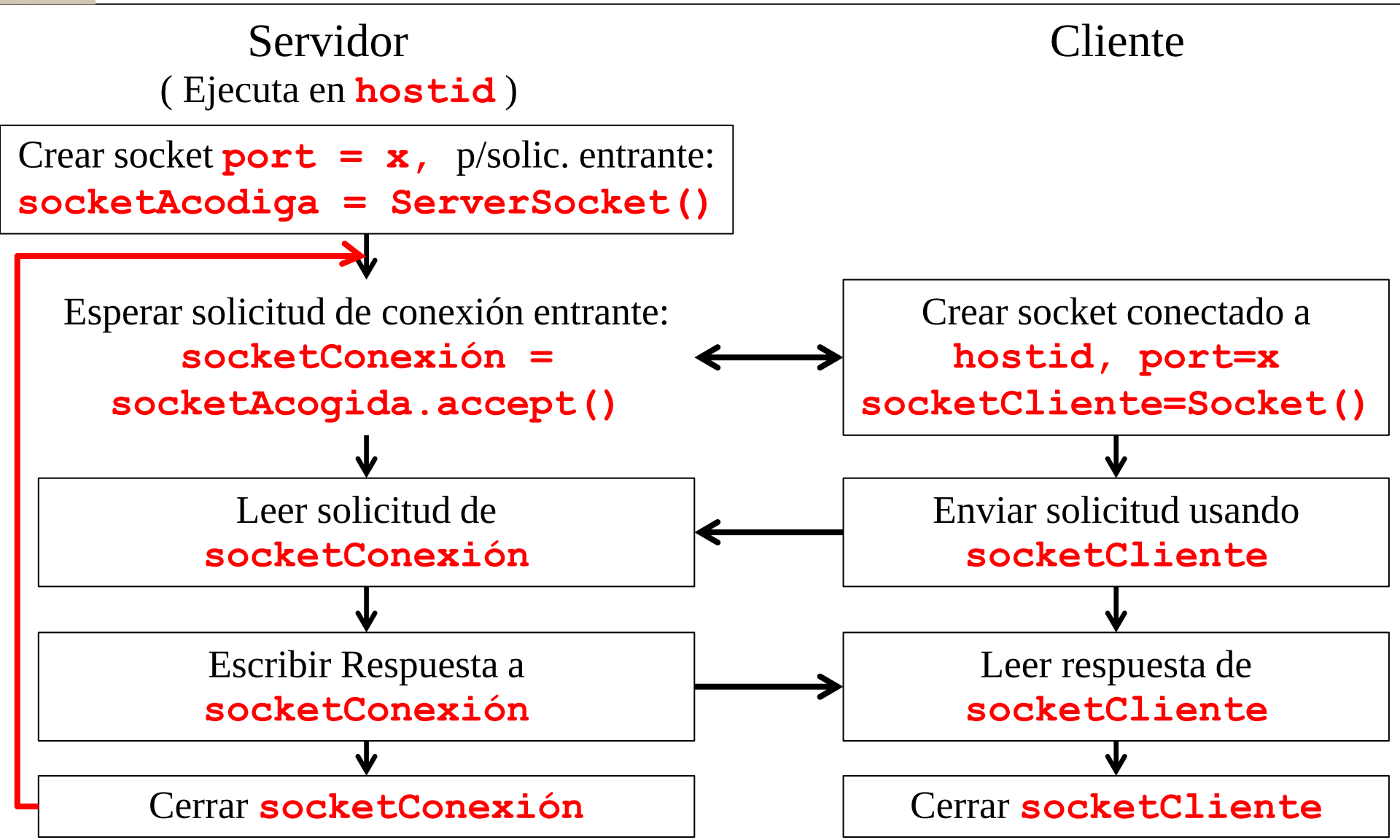
Cliente contacta Server :

- Creando un socket TCP, especificando IP y puerto de proceso Server
- Cuando Cliente crea el socket: Cliente TCP establece la conexión al Server TCP

- Cuando es contactado por Cliente, Server TCP crea un nuevo socket para que el proceso servidor pueda comunicarse con el nuevo Cliente
- Esto permite al Server dialogar con múltiples Clientes
  - Utiliza número de puerto de origen para distinguier Clientes (mas en Cap 3)

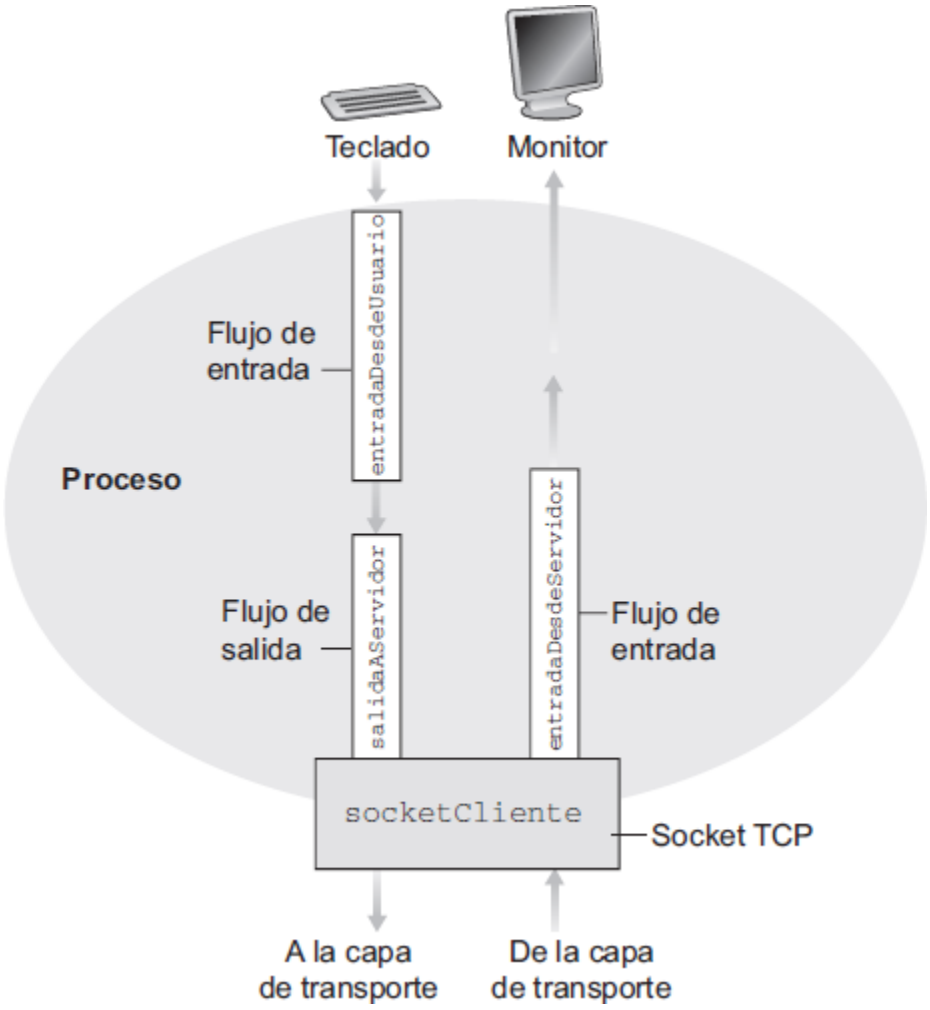
- Punto de vista de la Aplicación
  - *TCP brinda una transferencia confiable de bytes entre Cliente y Servidor*

Programación de Socket TCP



Programación de Socket TCP

El programa TCPCliente crea tres flujos y un socket



Programación de socket UDP

UDP no genera una "connexión" entre cliente & servidor

- Ningún handshaking antes de enviar datos
- Emisor adjunta IP de destino + # puerto a cada paquete
- Receptor extrae IP de origen + # puerto de cada paquete

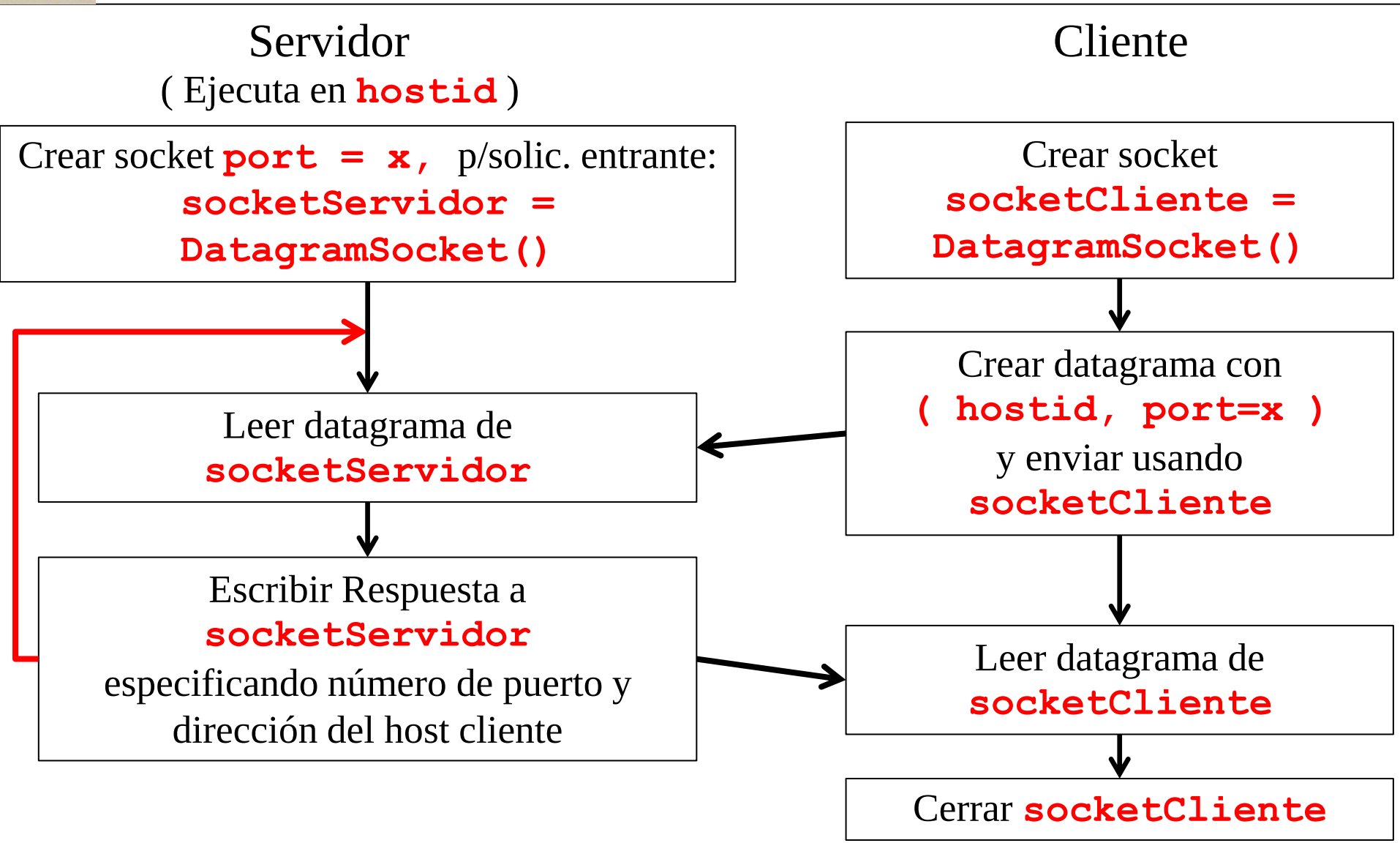
Los datos transmitidos puede perderse

o recibirse fuera de orden..!!

Punto de vista de la aplicación:

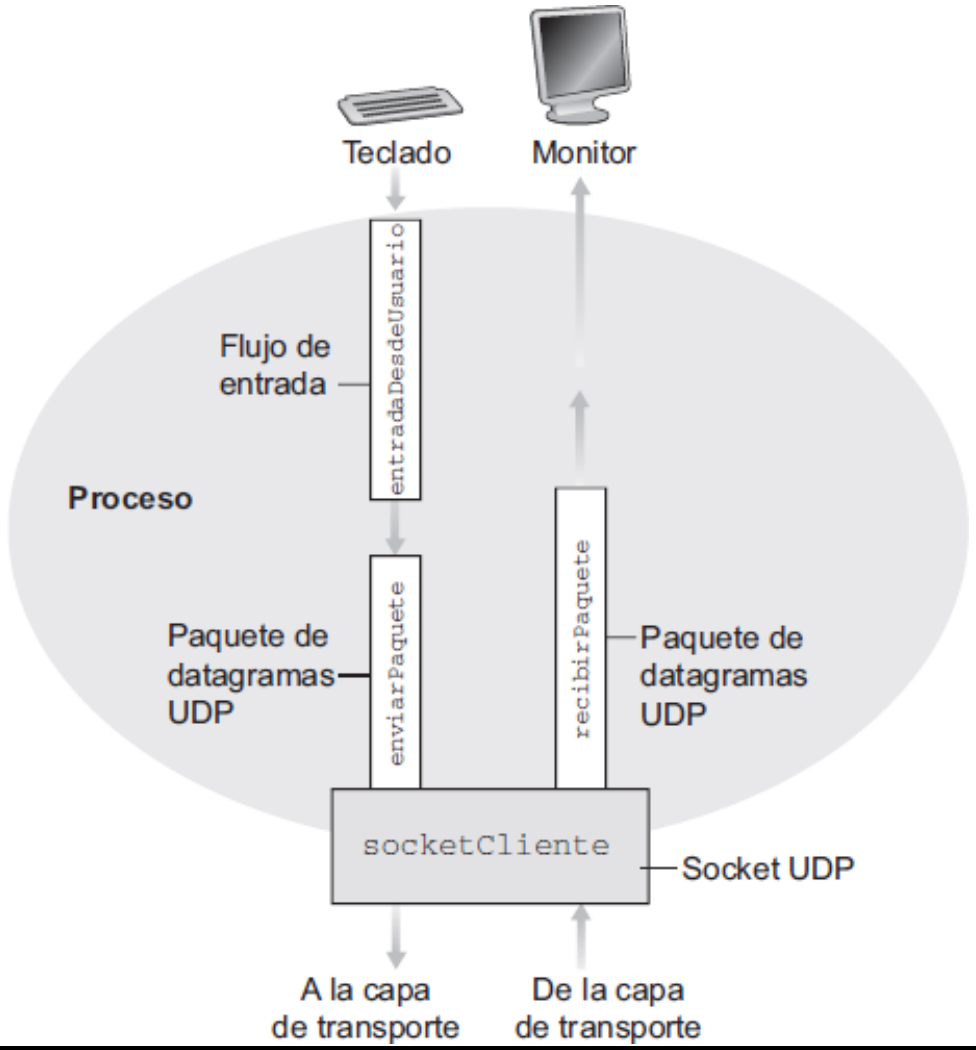
- UDP provee tranferencia no confiable de grupo de bytes ("datagrams") entre cliente y servidor

Programación de socket UDP



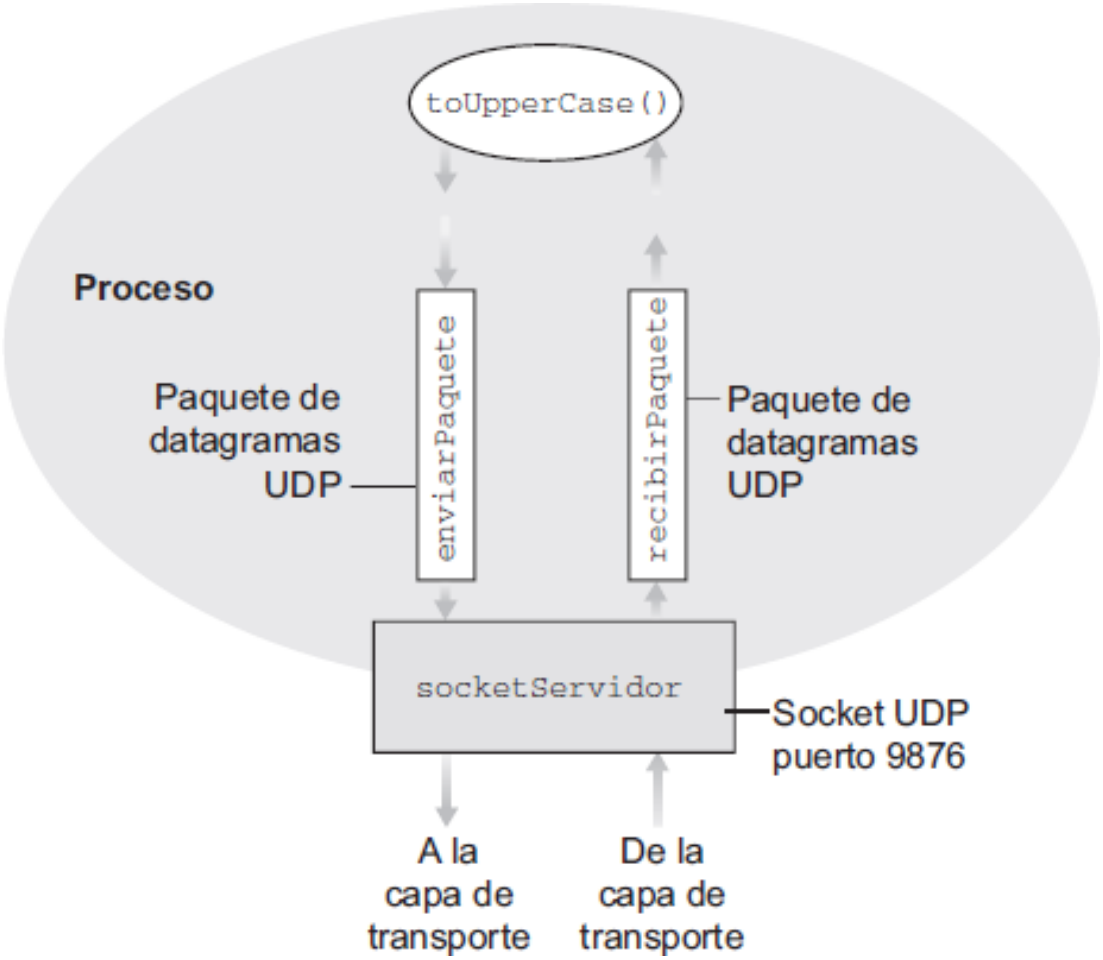
Programación de socket UDP

El programa UDPCliente crea un flujo y un socket



Programación de socket UDP

El programa UDPServer no tiene flujos



Capítulo 2: resumen

Hemos recorrido un estudio de la capa de aplicación..!!

- Arquitecturas de aplicaciones
  - client-server
  - P2P
- Servicios requeridos por la capa de aplicación:
  - confiabilidad, BW, retardo
- Modelos de servicio de transporte de Internet
  - Orientado a la conexión, confiable: TCP
  - No confiable, datagramas: UDP
- Protocolos específicos:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent, DHT
- socket programming: TCP, UDP sockets

Capítulo 2: resumen

Hemos aprendido de los protocolos..!!

- Intercambio típico de mensajes request/reply :
  - Cliente requests información o servicio
  - Servidor responde con datos, código de status
- Formato de mensajes:
  - Cabecera: brinda información sobre los datos
  - Datos: información a ser comunicada

Capítulo 2: resumen

Hemos aprendido de los protocolos..!!

Temas importantes:

- Mensajes de control vs datos
  - in-band, out-of-band
- Centralizado vs. Descentralizado
- Sin estado vs. stateful
- Transferencia confiable vs. no confiable
- “La complejidad está en el borde de la red”


GRACIAS!