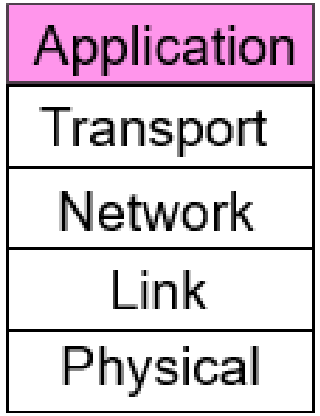
Capítulo 2

La Capa de Aplicación



Capa de aplicación

- En la **capa de aplicación** tenemos las **aplicaciones de red.**
- Cada aplicación de red ofrece un servicio específico
  - con su propia forma de interfaz con el usuario.

¿por qué es importante estudiar la capa de aplicación?

- Para poder entender cómo están organizadas las aplicaciones de red
- Para comprender el protocolo de una aplicación de red.
- Para poder diseñar y programar aplicaciones de red.
  - Para comprender los enfoques para programar las aplicaciones de red

Metas de la Capa de Aplicación

- **Vamos a aprender:**
  1. **Enfoques para desarrollar aplicaciones de red**
  2. Estilos de arquitectura de aplicaciones de red.
  3. Protocolos de capa de aplicación
  4. La Web

Capa de aplicación

- Veremos **dos enfoques para desarrollar aplicaciones de red**:

1. El programador para especificar la comunicación usa **una interfaz para programas de aplicación (API)**.
   - Una API es conjunto básico de funciones.
   - La **socket API** se usa para el software que se comunica sobre la internet.
   - Esto lo van a ver en el taller.
   - **El saber sobre el sistema operativo de red subyacente tiene su utilidad:**
     - permite al programador escribir mejor código y desarrollar aplicaciones más eficientes.

Capa de aplicación

2. **La Web**: El programador se apoya en **la tecnología de la web** para construir una aplicación de red..
   - La Web provee servicios al software de la aplicación que
     - hacen más fácil a los desarrolladores implementar la comunicación y la entrada/salida de modo que
     - se pueden enfocar en el propósito específico de la aplicación.
   - Se estudiará en el teórico-práctico de la materia.

Metas de la Capa de Aplicación

- **Vamos a aprender:**
  1. Enfoques para desarrollar aplicaciones de red
  2. **Estilos de arquitectura de aplicaciones de red.**
  3. Protocolos de capa de aplicación
  4. La Web

Arquitecturas de aplicaciones

Las aplicaciones red suelen usar uno de los siguientes **estilos de arquitectura**:

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

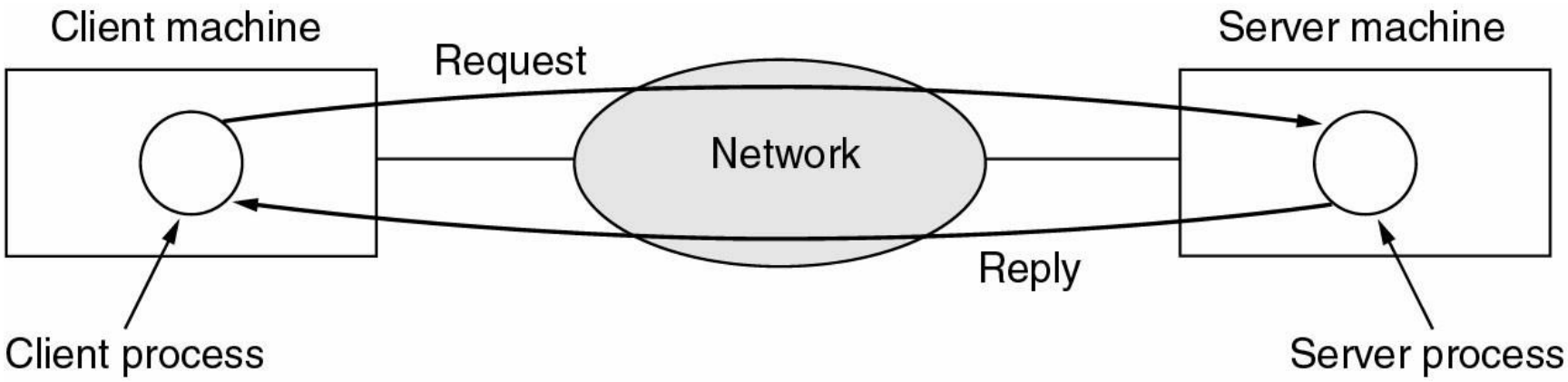
Conocer de estilos de arquitectura de aplicaciones de red tiene la siguiente utilidad:

- Aplicar un estilo de arquitectura facilita el diseño de una aplicación de red.

Hay que decidir cuál estilo de arquitectura es el más adecuado para una aplicación de red.

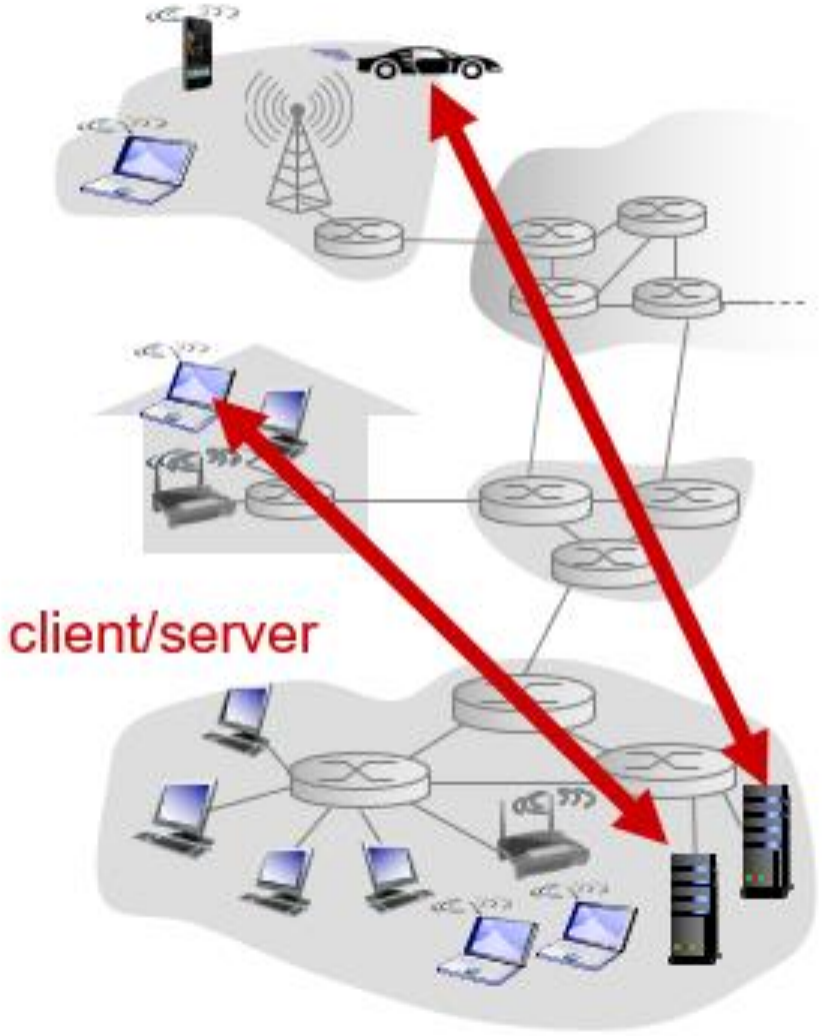
  - De modo que la aplicación use efectivamente la red.

Arquitecturas Cliente-Servidor



- En el **modelo cliente-servidor** hay dos procesos que se comunican: uno en la máquina cliente y otro en la máquina servidor.
- **Forma de la comunicación:**
  1. El proceso cliente manda **solicitud** al proceso servidor,
  2. el proceso cliente espera un **mensaje de respuesta;**
  3. luego el proceso servidor recibe y procesa la solicitud;
  4. el proceso servidor manda mensaje de **respuesta** al proceso cliente.

Arquitecturas Cliente-Servidor



Características de los servidores:

- Siempre están en un host;
- con dirección IP permanente;
- se pueden usar centros de datos para escalabilidad.

Características de los clientes:

- Pueden estar conectados **intermitentemente;**
- usando direcciones IP dinámicas;
- Los clientes no se comunican directamente entre sí.

Aplicaciones Cliente Servidor en internet usando UDP

Pasos de una aplicación cliente-servidor usando UDP.

1. Cliente crea datagrama con IP y puerto del servidor y envía datagrama
   - Datagrama puede perderse.
2. Si llega, servidor lee datagrama
3. Servidor envía respuesta especificando dirección y puerto de cliente
   - Datagrama puede perderse.
4. Si llega, cliente lee datagrama.
5. Cliente finaliza

- **Evaluación:**
  - No se dice qué se hace si respuesta no llega al cliente.
  - Es responsabilidad de la aplicación red manejar esto.

Aplicaciones Cliente Servidor en internet usando TCP

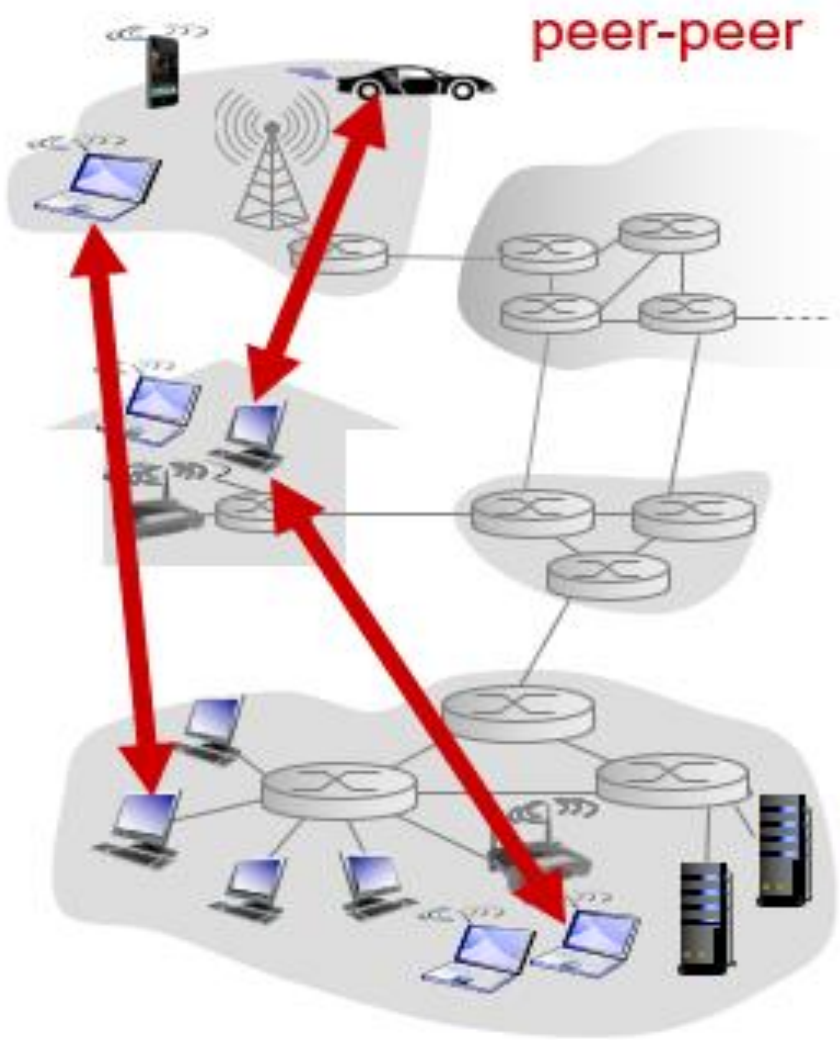
- **Pasos de una aplicación cliente-servidor usando TCP:**

1. Se ejecuta proceso del servidor
2. Servidor espera por pedido de conexión entrante.
3. El cliente requiere pedido de conexión al servidor
4. El servidor acepta la conexión con el cliente
5. El cliente envía pedido al servidor
6. El servidor lee el pedido
7. El servidor envía la respuesta
8. El cliente lee la respuesta
   - TCP provee transferencia de stream de bytes ordenada
9. Si hay más pedidos al servidor: Goto 5
10. El cliente cierra la conexión
11. El servidor cierra la conexión

Arquitectura P2P

Características de una arquitectura P2P:

- Mínimo o ningún apoyo en servidores.
- Hosts arbitrarios (llamados **compañeros**) se comunican directamente entre sí.
- Compañeros piden servicio de otros compañeros, y proveen servicio en retorno a otros compañeros
- Nuevos compañeros traen nueva capacidad de servicio, así como nuevas demandas de servicios.
- Los compañeros se conectan ***intermitentemente*** y cambian las direcciones IP.



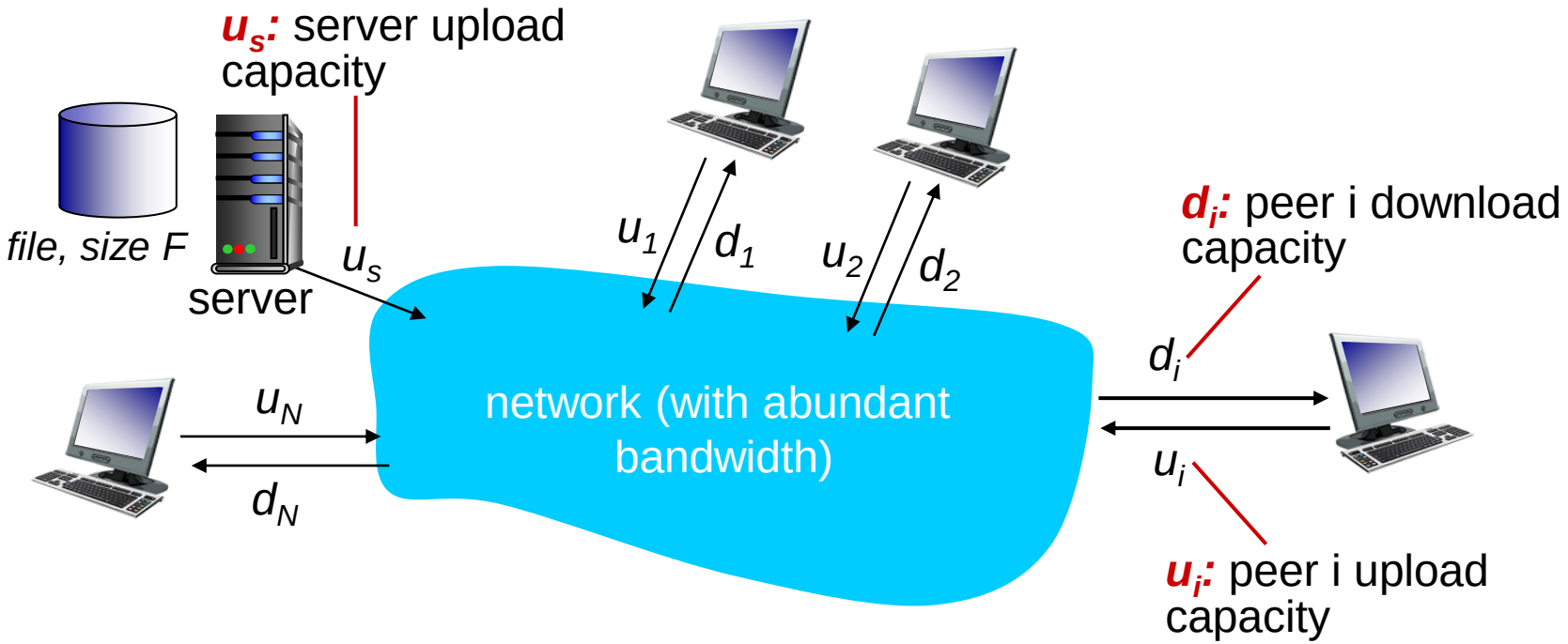
Arquitectura P2P

- **Ejemplos de aplicaciones P2P:**
  - **Distribución de archivos**: la aplicación distribuye un archivo de una única fuente a un gran número de compañeros.
    - Un ejemplo es BitTorrent.
  - **Bases de datos distribuidas** sobre una gran comunidad de compañeros.
  - **Streaming**: video on demand - (P.ej: KanKan)
  - **VoIP**: voz sobre IP (P.ej: Skype)

Distribución de archivos:
P2P vs cliente-servidor

Problema: ¿Cuánto tiempo se requiere para distribuir un archivo (de tamaño *F*) de un servidor a *N compañeros*?

- La capacidad de subida y de bajada de compañeros es un recurso limitado.
- Responderemos la pregunta para cliente-servidor y para P2P.

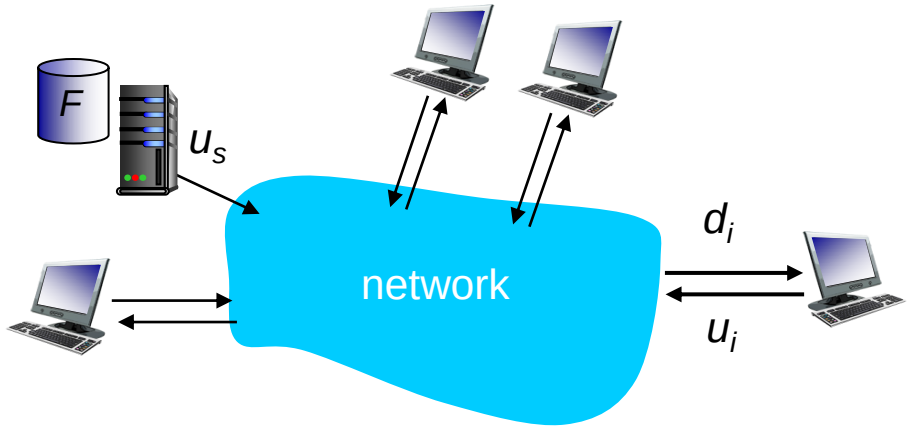


Distribución de Archivos: Cliente-Servidor

- **¿Qué Parámetros hay que considerar?**
  - Tasa de subida del enlace de acceso al compañero i: u_i
  - Tasa de subida del enlace de acceso al servidor: u_s
  - Tasa de descarga del enlace de acceso al compañero i: d_i
  - Tamaño del archivo a ser distribuido: F
  - Número de compañeros que quieren adquirir una copia del archivo: N
- El **tiempo de distribución** es el tiempo que toma obtener una copia del archivo por los N compañeros.
  - Asumimos que la internet ***tiene abundante ancho de banda*** y todos los cuellos de botella suceden en ISP de acceso.
  - Asumimos que los servidores y clientes ***no participan*** de otras aplicaciones de red.

Distribución de archivos: cliente-servidor

- *Transmisión del servidor:* debe enviar secuencialmente (subida) *N* copias de archivo a cada peer (manda *NF* bits).
  - Tiempo para enviar 1 copia: F/u_s
  - Tiempo para enviar N copias: NF/u_s
  - Demasiado trabajo del servidor



- *Descarga del Cliente:* cada cliente debe descargar una copia de archivo.
  - $d_{min} = \min\{d_1, d_2, \ldots, d_N\}$.
  - Tiempo de descarga del cliente con d_{min}: F/d_{min} *segs*
  - Este es el tiempo de descarga peor.

Ningún compañero ayuda a distribuir el archivo.

Tiempo para distribuir F a N clientes usando enfoque cliente-servidor

$$D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$$

aumenta linealmente en N

Distribución de Archivos: P2P

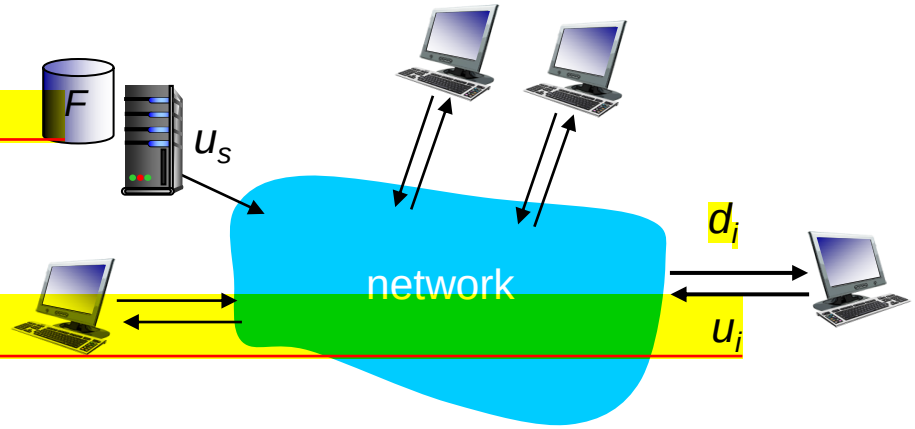
- Al comienzo de la distribución
  - solo el servidor tiene el archivo.
- Para que la comunidad de compañeros reciba este archivo,
  - el servidor debe enviar cada bit del archivo al menos una vez en su enlace de acceso.
- En P2P cada compañero puede redistribuir cualquier porción del archivo que ha recibido a cualesquiera otros compañeros.
  - Así los compañeros asisten al servidor en el proceso de distribución.
  - Cuando un compañero recibe algo de datos de un archivo, puede usar su capacidad de subida para redistribuir los datos a los otros compañeros.
- La capacidad total de subida del sistema es:

$$\mathbf{u}_{total} = u_s + \sum u_i$$

- Por lo tanto el **tiempo mínimo de distribución** es: $NF/\mathbf{u}_{total}$

Tiempo de Distribución de Archivos: P2P

- *Transmisión de servidor:* debe subir al menos una copia.
  - Tiempo para enviar una copia : F/u_s
- *cliente:* cada cliente debe descargar la copia de un archivo
  - Tiempo mínimo de descarga de cliente: F/d_{min}
- *clientes:* como agregado deben subir *NF* bits
  - Tasa de subida máxima (tasa máxima limitante de descarga) es $u_s + \Sigma u_i$



tiempo para distribuir F a N clientes usando enfoque P2P

$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

incrementa linealmente en *N* …

… pero lo hace mientras cada compañero brinda servicio

Distribución de Archivos

Tiempo para distribuir F a N clientes usando enfoque cliente-servidor

$$D_{c\text{-}s} \geq \max\{NF/u_s, F/d_{min}\}$$

increases linearly in N

versus

tiempo para distribuir F a N clientes usando enfoque P2P

$$D_{P2P} \geq \max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

increases linearly in N …

… but so does this, as each peer brings service capacity

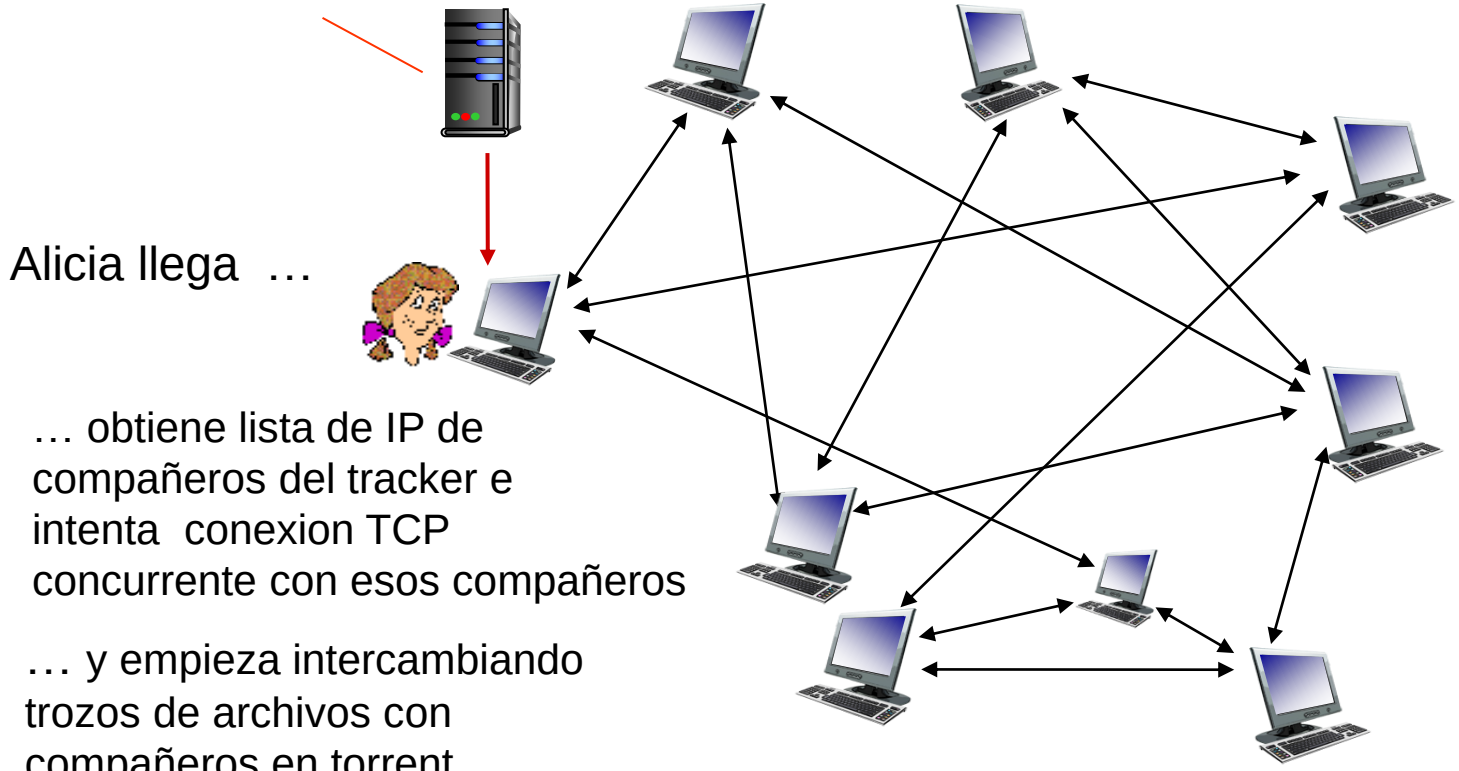
¿Qué diferencia observan?

Distribución de archivos P2P: BitTorrent

- El archivo se divide en trozos de 256Kb.
- Los compañeros en torrent envían/reciben trozos.

tracker: lleva la pista de compañeros Participando en Torrent

torrent: grupo de compañeros intercambiando trozos de un archivo



Alicia llega …

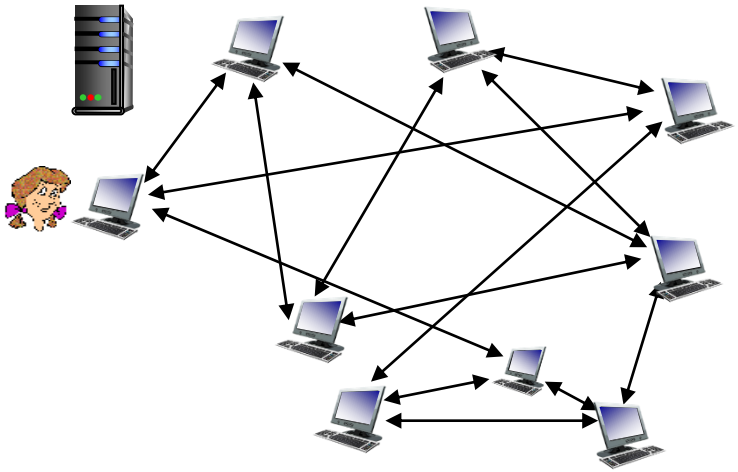
… obtiene lista de IP de compañeros del tracker e intenta conexion TCP concurrente con esos compañeros

… y empieza intercambiando trozos de archivos con compañeros en torrent.

Los compañeros de Alicia variarán con el tiempo.

Distribución de archivos P2P: BitTorrent

- **Cuando compañero se une a Torrent:**
  - No tiene trozos, pero va a acumularlos a lo largo del tiempo de otros compañeros
  - Se registra con tracker para obtener lista de compañeros.
  - Se conecta con un subconjunto de compañeros ("vecinos.")
  - Un compañero avisa periódicamente a tracker que está en BitTorrent.



- Mientras descarga, compañero sube trozos a otros compañeros.
- Un compañero puede cambiar de compañeros con los cuales intercambia trozos.
- Los compañeros pueden ir y venir.
- **Una vez que un compañero tiene un archivo completo:**
  - Puede (egoístamente) irse o (altruísticamente) permanecer en Torrent subiendo trozos.

BitTorrent: pedir y enviar trozos de archivos

Pedir trozos:

- En un momento dado, diferentes compañeros tienen diferentes subconjuntos de trozos de archivos
- Periódicamente, Alicia pide a cada compañero por una lista de trozos que ellos tienen.
- **Alicia puede hacer esto porque**
  - sabe qué trozos tienen sus compañeros.
- **Conviene pedir primero los trozos**
  - menos comunes (i.e. con menos copias en compañeros).

Enviar trozos: tit-for-tat

- Alicia envía trozos a aquellos 4 compañeros actualmente enviándole a Alicia trozos a la velocidad mayor.
  - re-evalua los 4 mejores cada 10 seg
- Cada 30 seg: elegir aleatoriamente otro compañero, y comenzar enviándole trozos.
  - Un compañero nuevo elegido puede unirse a los 4 de más arriba.
  - Pero para esto tiene que ser uno de los 4 mejores subidores para Alicia.

Metas de la Capa de Aplicación

- **Vamos a aprender:**
  1. Enfoques para desarrollar aplicaciones de red
  2. Estilos de arquitectura de aplicaciones de red.
  3. Protocolos de capa de aplicación
  4. La Web

Protocolos de capa de aplicación

- Para hacer un diseño detallado de una aplicación de red conviene definir un **protocolo de capa de aplicación**.

Protocolos de capa de aplicación

Cosas a definer en un protocolo de capa de aplicación:

- Tipos de mensajes intercambiados
  - P.ej: de pedido, de respuesta
- Sintaxis del mensaje:
  - Qué campos hay en un mensaje y Cómo los campos son delineados
- Semántica del mensaje
  - Significado de la información en los campos

Reglas de cuándo y cómo los procesos envían y responden a mensajes.

Estado de la aplicación. En qué consiste y cómo se lo mantiene.

Tipos de protocolos

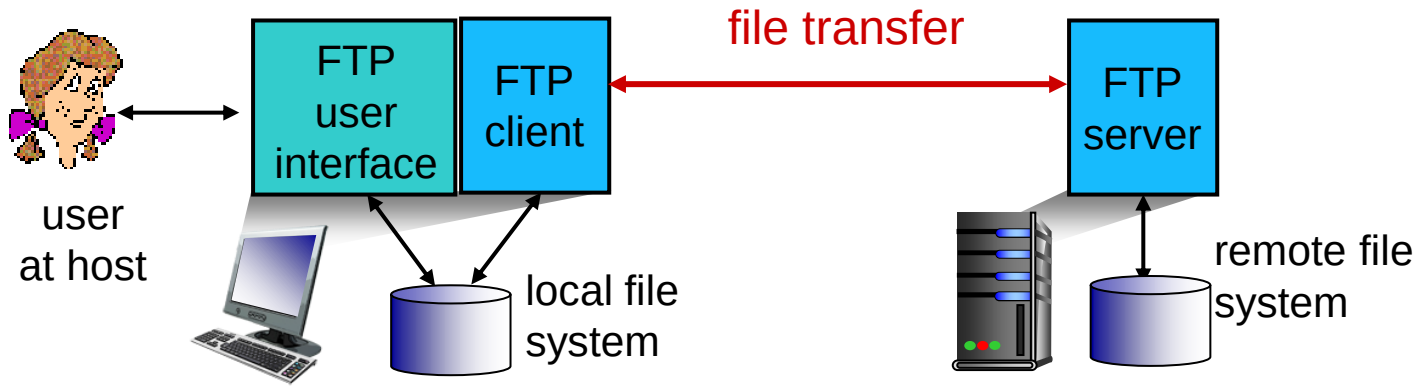
Protocolos abiertos:

- Son definidos en RFCs
- Permiten interoperabilidad
- P.ej: HTTP, SMTP, FTP

Protocolos propietarios:

- P.ej: Skype

FTP: Protocolo de Transferencia de Archivos



- **Algunas características de FTP:**
  - Usado para transferir archivo hacia/desde host remoto
  - Cada archivo tiene restricciones de acceso y poseción.
  - FTP permite inspeccionar carpetas.
  - FTP permite mensajes de control textuales.
- Modelo cliente/servidor
  - *cliente:* lugar que inicia la transferencia (hacia o desde el host remoto)
  - *servidor:* host remoto.
- Servidor ftp: puerto 21

FTP: Protocolo de Transferencia de Archivos

- **3 tipos de mensajes son intercambiados:**
  - Uso de **comandos** enviados al servidor ftp
    - Enviados como texto ASCII sobre un canal de control
  - P. ej: comando de transferencia de archivo.
  - **Mensajes de respuesta** a comandos del servidor ftp
  - **Mensajes con datos enviados**.
    - Listado de folder, archivo, etc.

FTP: Protocolo de Transferencia de Archivos

Sintaxis de comandos

- **USER *username***
- **PASS *password***
- **LIST** return list of file in current directory
- **RETR filename** retrieves (gets) file
- **STOR filename** stores (puts) file onto remote host

Sintaxis de Mensajes de respuesta

- Código de estatus y frase

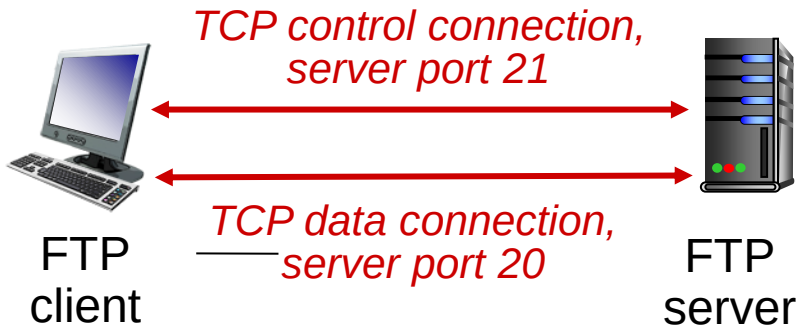
Ejemplos de Mensajes de respuesta

- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

FTP: Protocolo de Transferencia de Archivos

Reglas:

1. Cliente FTP contacta servidor FTP en puerto 21, usando TCP.
2. El cliente es autorizado en la conexión de control.
3. El cliente inspecciona directorio remoto, envía comandos sobre la conexión de control.
   - Se comienza con identificación de usuario y password.
4. Cuando el servidor recibe un comando de transferencia de archivo, el *servidor* abre una *2da conexión de datos* TCP (para el archivo) *con el cliente.*
5. Luego de transferir un archivo, el servidor cierra la conexión de datos.

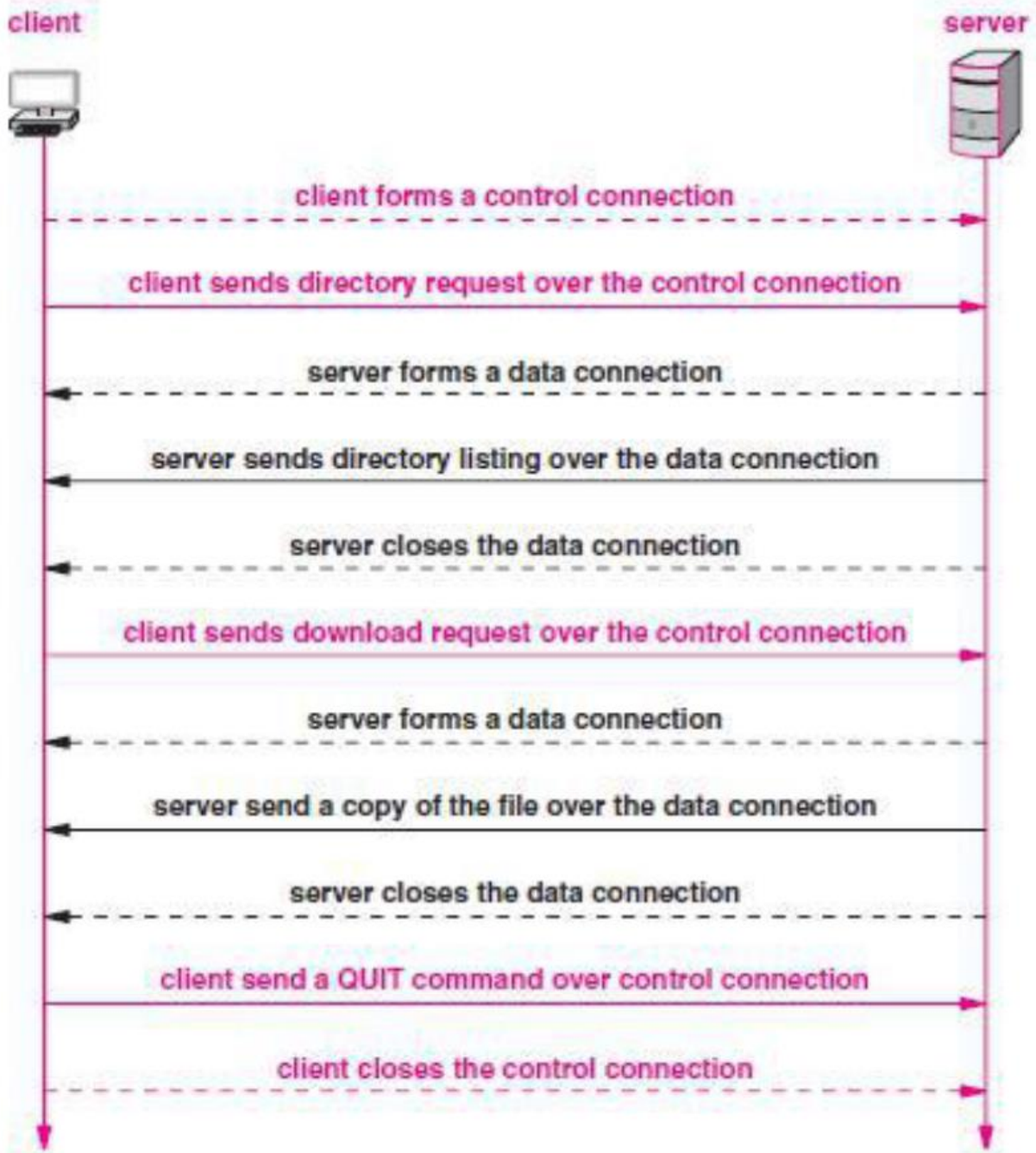


- El servidor abre otra conexión TCP de datos para transferir otro achivo.

Estado

- El servidor FTP mantiene el "estado": **directorio corriente**, **autenticación previa**.

FTP


client
server
client forms a control connection
client sends directory request over the control connection
server forms a data connection
server sends directory listing over the data connection
server closes the data connection
client sends download request over the control connection
server forms a data connection
server send a copy of the file over the data connection
server closes the data connection
client send a QUIT command over control connection
client closes the control connection