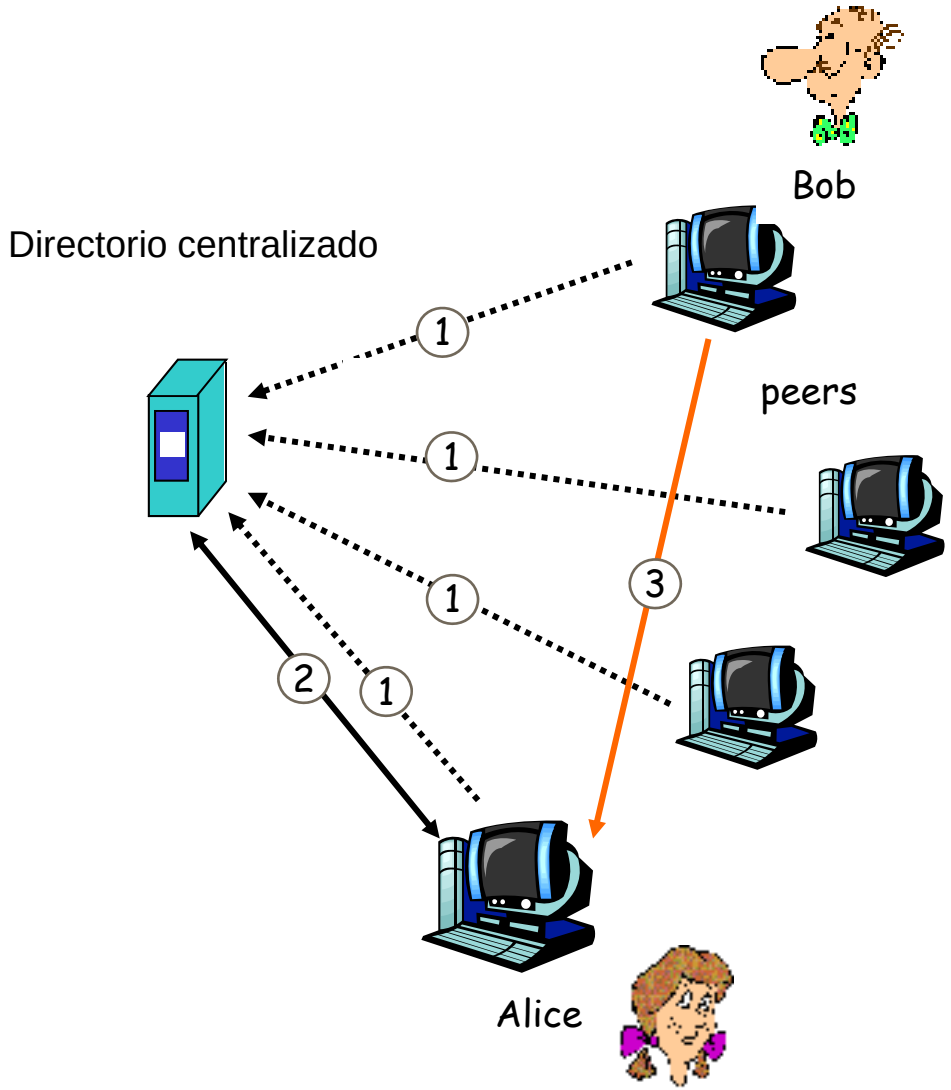
Indice

3.2 Napster (1999)

- Se basa en la utilización de servidores con función de directorio centralizado (híbrido cliente/servidor y P2P).
  - Cuando un usuario (peer) lanza la aplicación, se conecta automáticamente al directorio centralizado, registrando su IP y el nombre de los ficheros que está dispuesto a compartir.
  - El directorio centralizado lleva la información de ficheros y peers que lo poseen.
  - Para comprobar que un peer sigue activo periódicamente se le manda un mensaje que tiene que contestar. Si no lo hace se dará por desconectado.
  - Cuando un peer está interesado en un fichero, manda la petición al directorio centralizado y éste le devuelve el listado de peers a los que se puede conectar para obtener el fichero.
  - La conexión entre los peers se establece directamente para la transferencia del fichero.

Napster



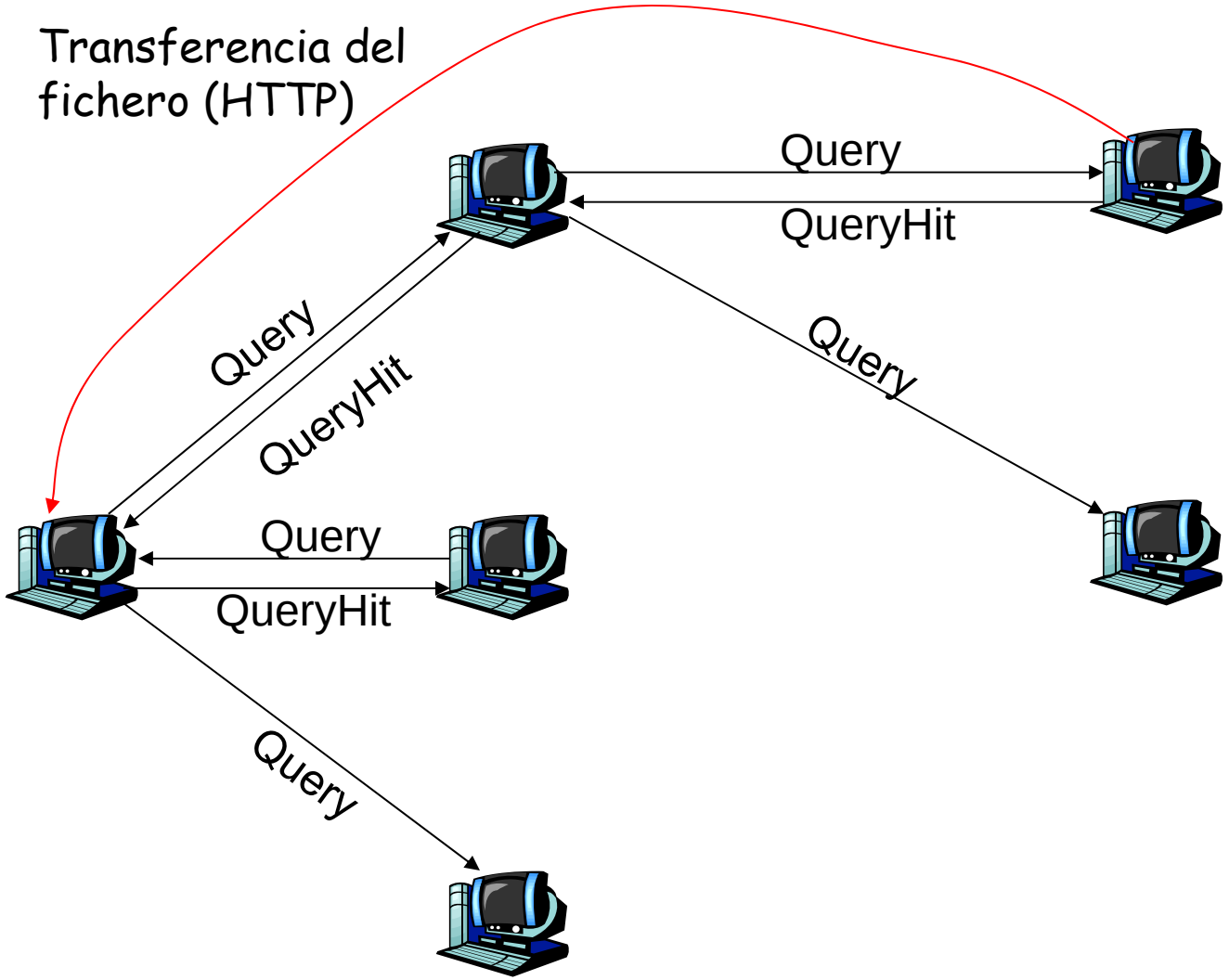
Napster

- Problemática del uso del directorio centralizado:
  - Un único punto de fallo: si el directorio centralizado se cae, toda la red P2P cae. Incluso con una granja de servidores realizando esa funcionalidad puede haber problemas con la red de acceso.
  - Cuello de botella: en una red P2P con miles de usuarios la base de datos del directorio centralizado es inmensa, lo que dificulta su mantenimiento y significa tener que atender muchísimas peticiones por segundo.
  - Problemas de copyright: al tener centralizada la información en un servidor, propiedad de una empresa en este caso, es un punto claro donde se infringe la legislación y con altas expectativas de ser clausurado con rapidez.

3.3 Gnutella (2000)

- Aplicación de dominio público para el intercambio de ficheros.
- La búsqueda de la información se realiza con un enfoque totalmente distribuido.
- Los peers forman una “overlay network” que no es mas que un grafo con los siguientes componentes únicamente:
  - Peers: los nodos del grafo.
  - Conexiones (TCP) entre peers: enlaces del grafo. No se trata de enlaces físicos.
- Un peer únicamente se conecta a algunos otros peers de la red (del orden de 10).
- El protocolo es simple, basado en HTTP.
- Búsqueda por inundación: cuando un peer quiere realizar una búsqueda manda la petición a sus peer vecinos, y estos a su vez la reenvían a sus vecinos. Los peers que tengan el fichero buscado devolverán una respuesta y entonces se podrá realizar la transferencia TCP directa entre ambos.

Gnutella


Transferencia del fichero (HTTP)
Query
QueryHit
Query
QueryHit
Query
Query
QueryHit
Query

Gnutella

- Búsqueda por inundación, problema: no es escalable. Cada vez que un peer lanza una búsqueda, esta se reenvía hasta llegar a todos los peers de la overlay, lo que supone introducir mucho tráfico.
  - Para solucionarlo se hace una búsqueda por inundación con cobertura limitada, que consiste en un contador del número de saltos (peer-count) que puede atravesar la búsqueda. Este contador se decrementa en cada peer que se atraviesa y la búsqueda se tira al llegar a 0.
  - Supone introducir menos tráfico pero sólo encuentra ficheros que estén en los peers cercanos.

Gnutella

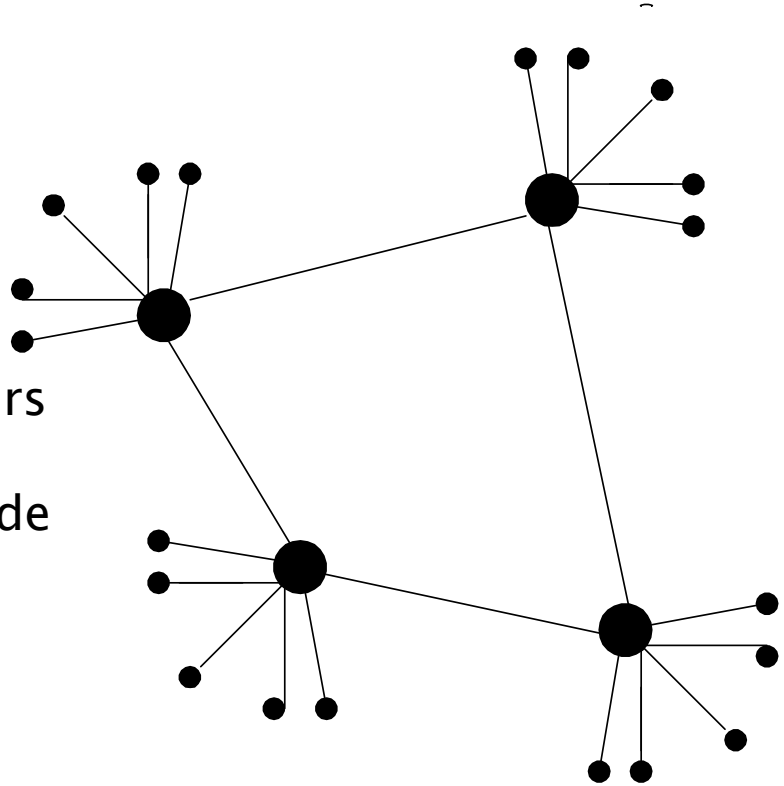
- Proceso de inclusión:
  - Un nodo para conectarse a la red necesita conocer la IP de al menos un peer que esté en esa red. Habitualmente el nodo mantiene un listado de IPs con las que se ha conectado alguna vez o se obtiene de algún website (*bootstrap list*).
  - Secuencialmente se intenta hacer una conexión con los nodos de la lista hasta conseguir una conexión.
  - Tras establecerse la conexión entre X e Y, X manda un Gnutella Ping a Y, el cual lo reenvía a todos sus vecinos en la overlay, y sucesivamente hasta que el peer-count llegue a 0.
  - Cada peer Z contesta al Gnutella Ping con un Gnutella Pong que incluye su dirección IP, el número de ficheros que intercambia y el número total de Kbytes que comparte.
  - Al recibir el Pong, X descubre otros peers de la red además de Y. Puede crear conexiones con algunos de estos peers Z o seguir chequeando los peers de la lista inicial.
  - El número de conexiones depende de la implementación del cliente.

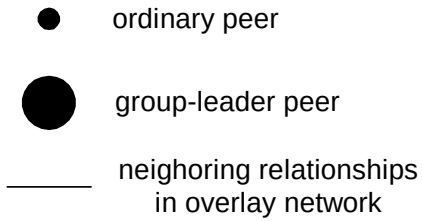
3.4 KaZaA (2001)

- Aplicación propietaria, incluso encripta todo el tráfico de control.
- Distingue una jerarquía de peers en la overlay:
  - Peers con mejor ancho de banda y tiempo de conectividad se designan líderes de grupo o hubs.
  - El resto son peers ordinarios y se asignan a un líder de grupo. Cada líder de grupo puede tener en el orden de pocos cientos de peers ordinarios asociados.
- Un peer al arrancar se conecta TCP a un líder de grupo al cual informa de los ficheros que quiere compartir. Esto le permite crear una base de datos con información del grupo.
- Para interconectarse con peers más allá del grupo, los líderes de grupo se conectan entre sí.
  - Pueden copiarse la base de datos de los demás líderes.
  - Pueden reenviar las peticiones de búsqueda a los demás líderes.

KaZaA

La estructura jerárquica permite buscar en más peers sin introducir una excesiva cantidad de tráfico en la red.



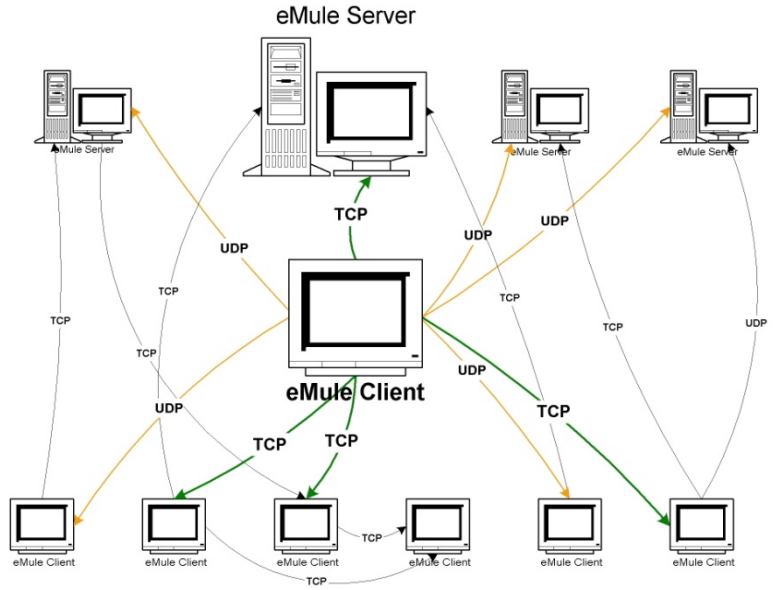


KaZaA

- Cada fichero se identifica con un hash además de con su nombre-descripción textual. Esto permite identificar como iguales ficheros que tengan descripciones con pequeñas diferencias.
- El resultado de la búsqueda ofrecido por el líder de grupo será un listado de peers que poseen el fichero y con los que se puede hacer la transferencia directa del mismo.
- Otras mejoras que implementa KaZaA:
  - Encolado de peticiones: se limita en cada peer el número de uploads simultáneos (3-7) y los siguientes se encolan para atenderlos más adelante.
  - Prioridad incentivada: mayor prioridad en las colas a los usuarios que han hecho más uploads que downloads.
  - Descarga paralela: se puede solicitar distintas rangos de bytes de un fichero a diferentes peers que lo posean simultáneamente (soportado por HTTP).

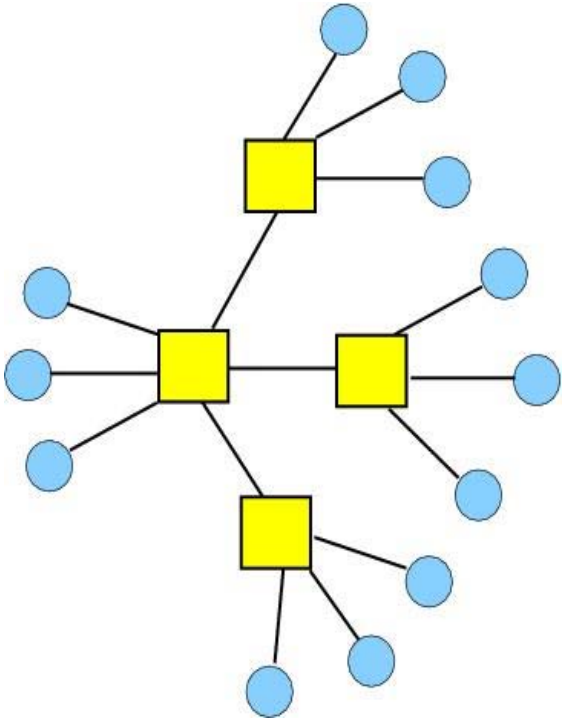
3.5 eDonkey (2001)

- Arquitectura jerárquica estilo KaZaA.
- UDP/TCP.
- Servidores centrales para tareas de localización de peers y búsqueda de ficheros.
- Descarga de varios peers que tengan el mismo fichero.
- Velocidad controlada por un sistema de créditos.



3.6 Gnutella2 (2002)

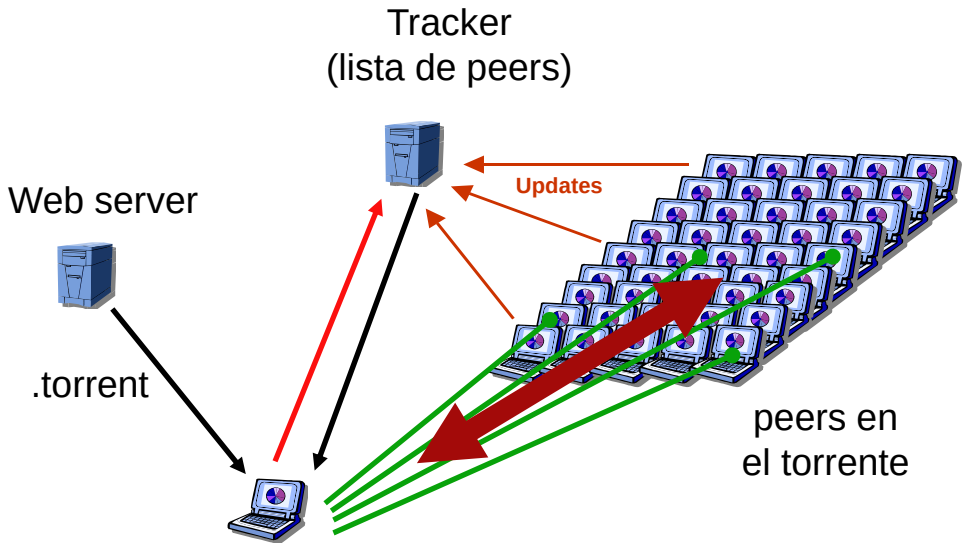
- Arquitectura jerárquica estilo KaZaA (hubs y leafs).
  - Protocolo abierto.
- UDP, aportando fiabilidad.



3.7 BitTorrent (2002)

- Optimizado para descarga localizada temporalmente de contenidos populares y *flashcrowds*. Por ejemplo, días cercanos a la salida de una distribución Linux.
- Búsqueda centralizada vía web: no integra sistema de busqueda de ficheros. Se centra en optimizar el transporte entre múltiples fuentes.

- Obtención de metafiles con extensión ".torrent".
- Descarga simultánea de diferentes trozos desde diferentes peers.
- Nuevo elemento: Tracker

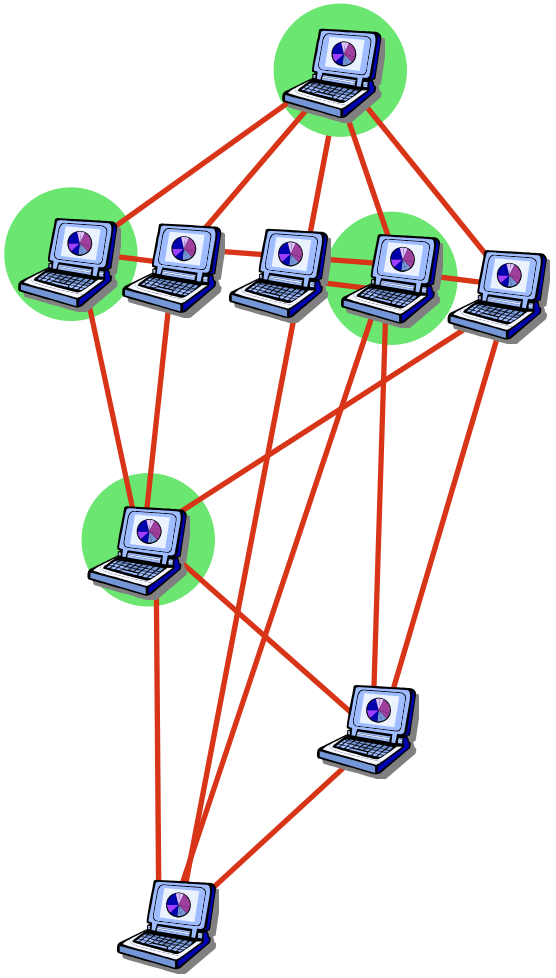


BitTorrent

- Malla de peers interesados en un fichero concreto: **Torrente**
- **Tracker** para encontrar otros peers
- Fichero *.torrent* describiendo la descarga
  - Partes del fichero con sus hashes
- Uniendose a un torrente
  - Bajar el .torrent
  - Lanzar peer de BT
  - Conseguir lista de peers
  - Conectarse a peers
  - Colaborar con los vecinos
- En un torrente
  - **Seeds**: tienen el fichero completo
  - **Leechers:** no tienen el fichero completo, pero pueden tener parte.
  - Un seed original. Leechers que acaban de bajarse el fichero son seeds.

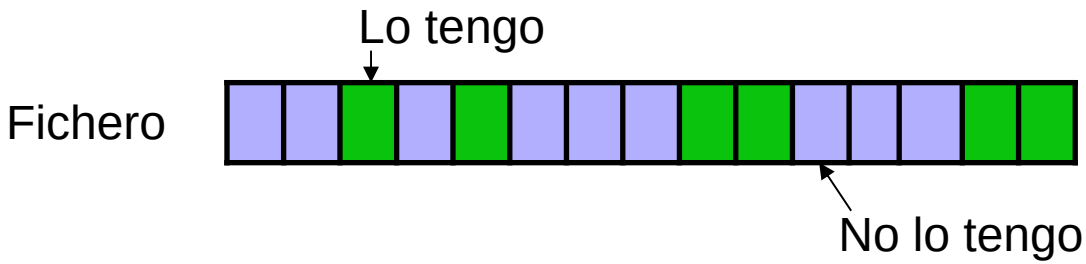
3.7.1 Malla de peers

- Un peer se conecta (TCP) a N vecinos (*por defecto 30*)
- Recibirá conexiones de peers
  A menos que esté detrás de un Firewall o NAT
- Los vecinos se convierten en seeds y/o se desconectan
- Si el número de vecinos cae por debajo de un mínimo se piden más al tracker



3.7.2 Descarga

- Los peers colaboran entre si (protocolo peer-to-peer) intercambiando partes del fichero.
- El fichero se divide en bloques o piezas
  - La lista de hashes de piezas están en el *.torrent*
  - Un peer sabe si tiene una pieza verificando el hash



- Protocolo BitTorrent entre peers, por la conexión TCP se envían entre vecinos
  - >> Información del estado (bitmap de piezas y pieza nueva)
    Siempre sé lo que tienen mis vecinos
  - >> Peticiones de piezas
  - >> Partes del fichero (en unidades mas pequeñas: subpiezas)
  - >> **Información de colaboración**

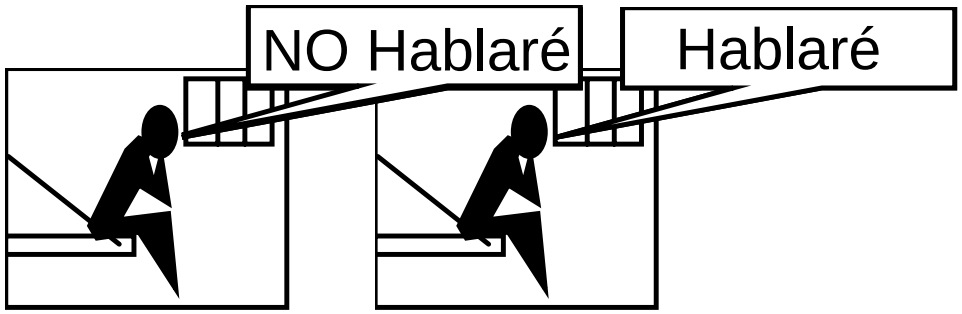
3.7.3 Colaboración

- Los peers colaboran entre sí. Tarde o temprano alguien escribirá un peer que se aproveche

  ¿Se puede construir un protocolo que obligue a la colaboracion?

- Teoría de juegos

  El dilema del prisionero



La policía arresta a dos sospechosos. No hay pruebas suficientes para condenarlos y, tras haberlos separado, los visita a cada uno y les ofrece el mismo trato.

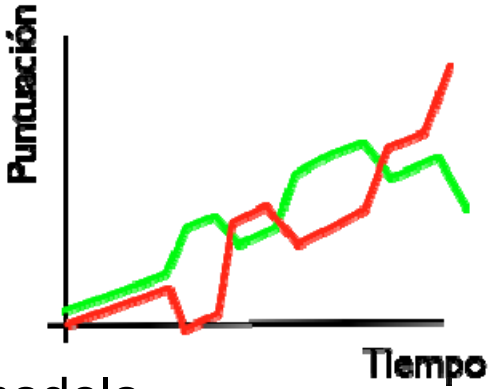
- Si uno confiesa y su cómplice no, el cómplice será condenado a la pena total, diez años, y el primero será liberado.
- Si uno calla y el cómplice confiesa, el primero recibirá esa pena y será el cómplice quien salga libre.
- Si ambos permanecen callados, todo lo que podrán hacer será encerrarlos durante seis meses por un cargo menor.
- Si ambos confiesan, ambos serán condenados a seis años.

- Juego iterativo: un dilema del prisionero cada turno

  ¿Hay alguna estrategia para ganar?

  ¿Qué tiene que ver esto con la descarga de ficheros?

Colaboración



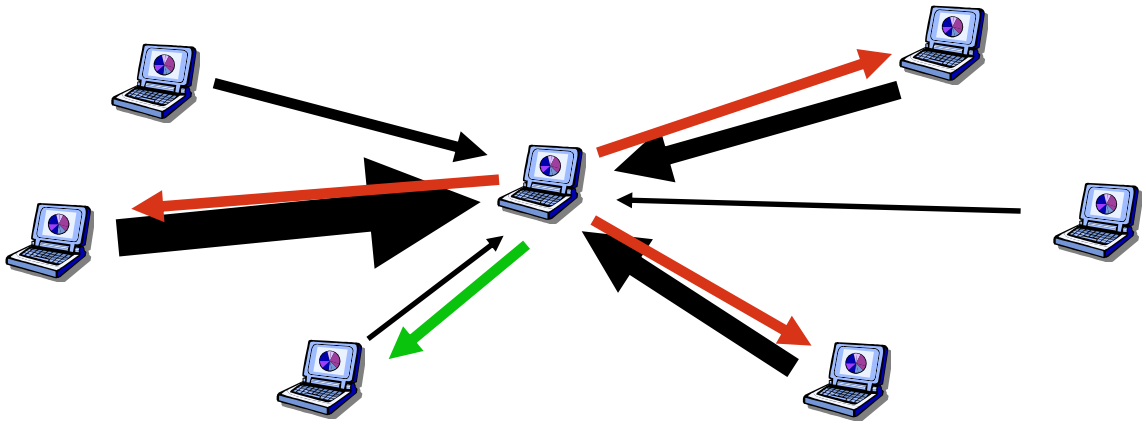
- La mejor estrategia a largo plazo se llama **tit-for-tat**
  - Empiezo colaborando
  - Repito lo que ha hecho el otro en el turno anterior
- 2 peers que se comunican siguen el mismo modelo
  Colaborar = enviar al otro peer lo que me pide:
  - Si no envío, todo el ancho de banda es para recibir.
  - Si los dos enviamos aun así recibo más que si los dos paramos.

	A Colabora	A No colabora
B Colabora	A +1 puntos B +1 puntos	A +3 puntos B -2 puntos
B No colabora	A -2 puntos B +3 puntos	A -1 puntos B -1 puntos

- BitTorrent se basa en este principio
  Por defecto envía lo que le piden
  Dejar de enviar: CHOKE
  Volver a enviar: UNCHOKE

Colaboración

- Un peer de BitTorrent envía sólo a 4 vecinos (UNCHOKE) y no envía (CHOKE) al resto.



- Cada 10 segundos decide a quien enviar
  - Elige a los 4 que más han enviado (Throughput 30seg)
- Elige además a uno al azar para Optimistic UNCHOKE
- Algoritmo **tit-for-tat** (teoría de juegos)
- Los seeds eligen con el throughput de subida. Intentan maximizar su upload

Colaboración

- Para que el tit-for-tat funcione, en general, cada peer siempre mantiene peticiones pendientes a cada vecino.
- Conozco todas las piezas que tiene mi vecino. ¿Qué pieza debo pedir?
  - Aproximación 1: Pieza aleatoria
  - Aproximación 2: Buscar la **pieza mas rara** para tener piezas que no tengan los demás
    Pido la pieza que menos tengan mis vecinos
    Esto no quiere decir que pida sólo una pieza a la vez !!

3.7.4 Observando un torrente

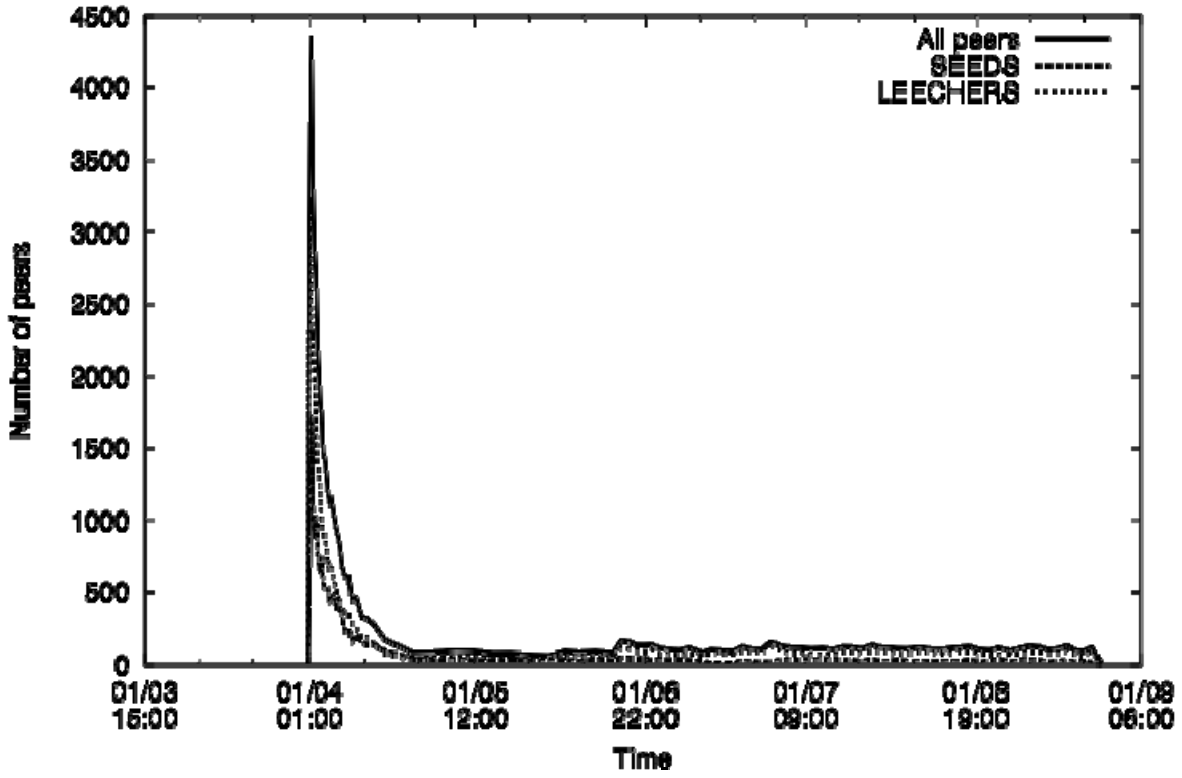
- Fuente: traza del tracker en f.scarywater.net del torrente de la distribución de Linux RedHat 9
- Traza conseguida en Ago2003 de 4 meses (aprox Abril-Julio 03)
- Es un apache-log-like file:

130.233.220.23 - - [**31/Mar/2003:12:52:30**] "GET /announce?**info_hash**=E%E9...%18%1D
& **peer_id**=%00...%8E/ & **port**=6882 & **ip**=130.233.20.169
& **uploaded**=0 & **downloaded**=0 & **left**=1855094951
& **event**=started HTTP/1.0" 200 94 217.160.111.64 - - [31/Mar/2003:12:52:48] "GET /announce?info_hash=E%e9...%1d ... 80.198.193.222 gzip - [31/Mar/2003:12:53:17] "GET /announce?info_hash=E%E9..%1D ... 80.198.193.222 gzip - [31/Mar/2003:12:53:17] "GET /announce?info_hash=E%E9...%1D ...

- Uploaded / downloaded / left bytes

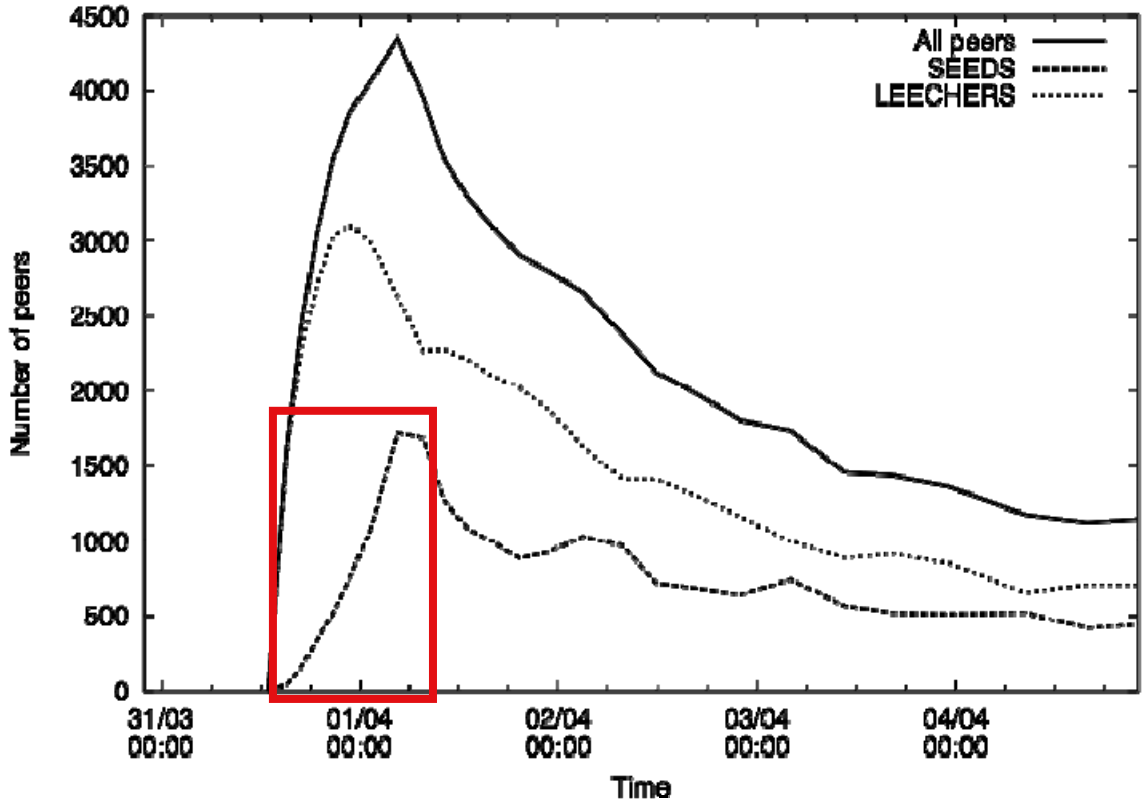
Observando un torrente

- El número de peers en un momento dado obtenido de la primera y última vez que se ve un session_id
  - Soportó un pico de más de 4000 sesiones simultáneas
  - Constantemente unos 100 peers



Observando un torrente

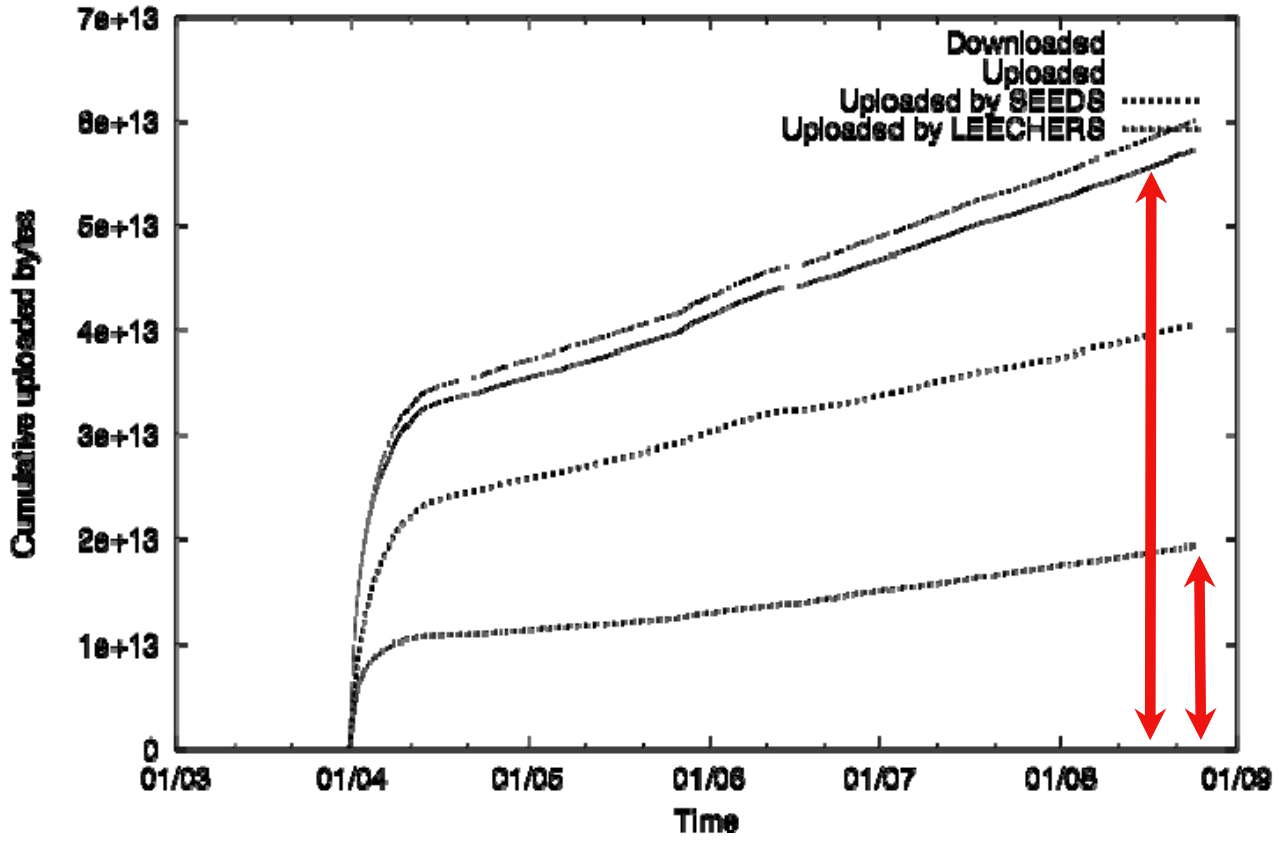
- Composicion de Seeds / Leechers en los primeros dias
  - Muchos usuarios pidiendo el fichero. Se crean SEEDS muy rápidamente
  - En medio día tenemos más de 1500 SEEDS !!!



Parece que BitTorrent aguanta muy bien los *flash-crowds* o *slashdot-effect*

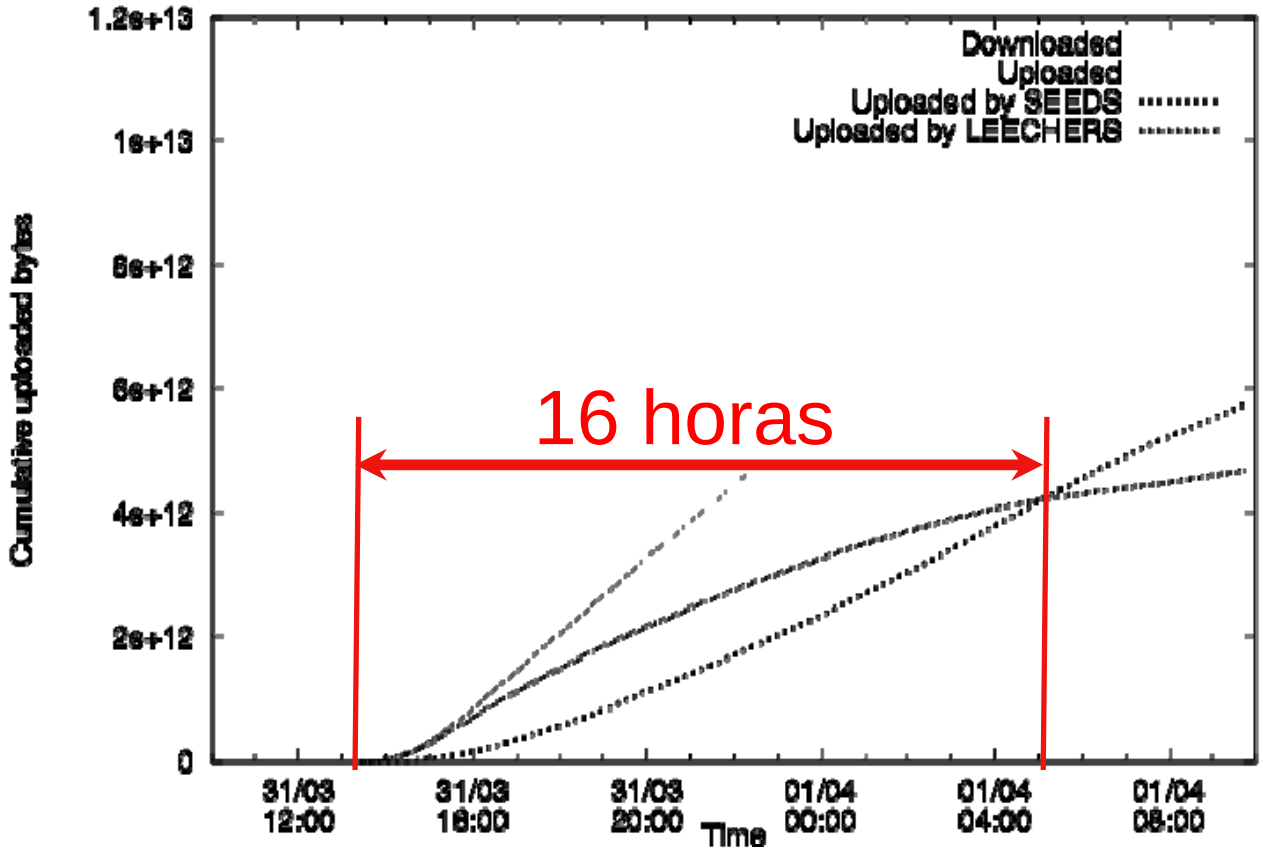
Observando un torrente

- Volumen total manejado por el torrente.
  - Upload y download no cuadran.
  - 1/3 del upload procede de Leechers.



Observando un torrente

- En los primeros momentos del torrente los Leechers contribuyen más que los Seeds.



Observando un torrente

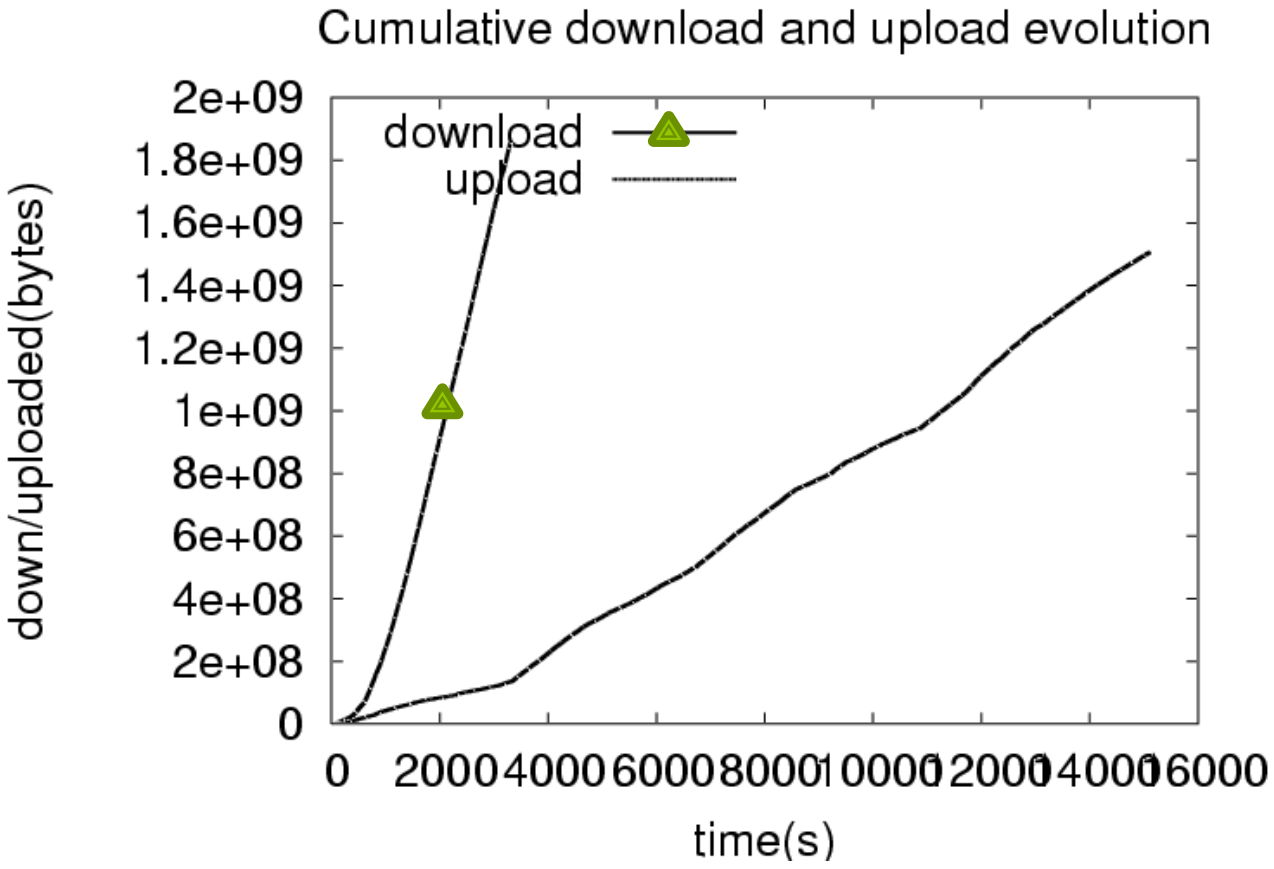
- **Problemas de la medida**
  - Resolución temporal de 15 minutos.
  - Sólo estado global del peer, no se puede reconstruir la malla de peers.
  - Es dificil de repetir la medida.
- **Ventajas / aspectos interesantes**
  - Se pueden detectar proxies/Nat.
  - Estadísticas de todo el torrente.
  - Estadísticas de 4000 usuarios interesados en el mismo fichero.

3.7.5 Siguiendo a un peer

- Análisis del código (python) de BitTorrent.
- Estudio de la dinámica de un peer de BitTorrent.
  - **BitTorrent modificado**
    - Log a fichero de variables del peer.
    - Log a fichero (por cada vecino) de eventos .
  - **Estudiar estas trazas de *nivel de aplicación***
- Datos tomados en Julio 2005.
  - Enlace UPNA.
  - Enlace Eurecom.
  - ADSL.

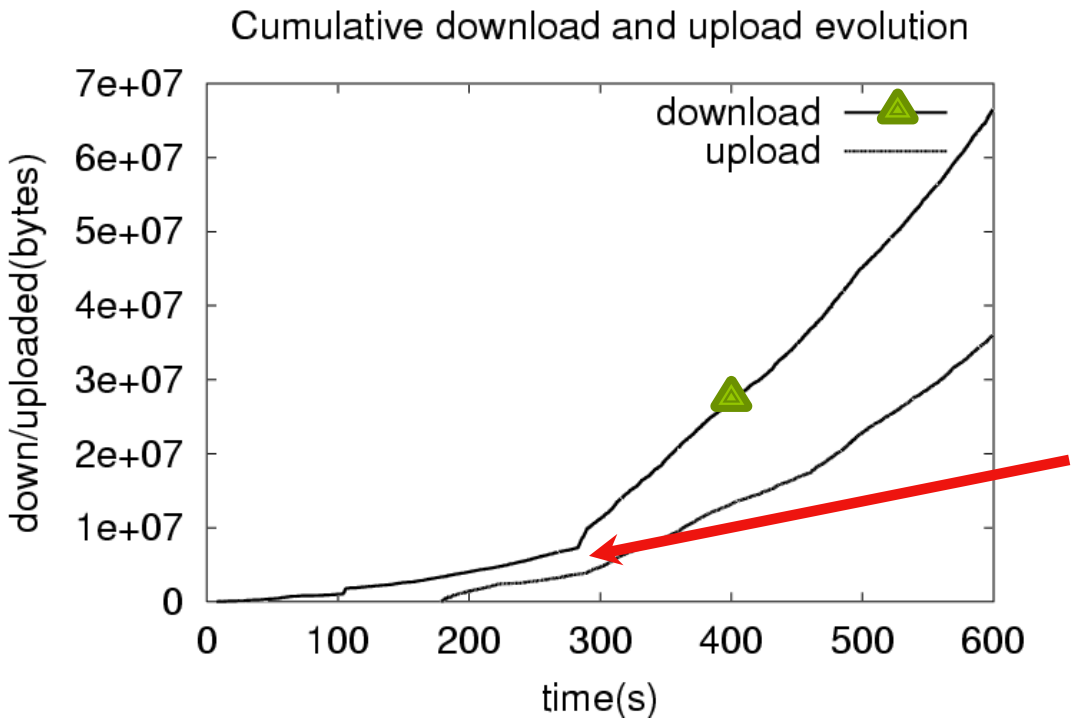
Siguiendo a un peer

- Bytes downloaded y uploaded sólo por el peer modificado.
  - Download mayor que el upload (enlace UPNA).



Siguiendo a un peer

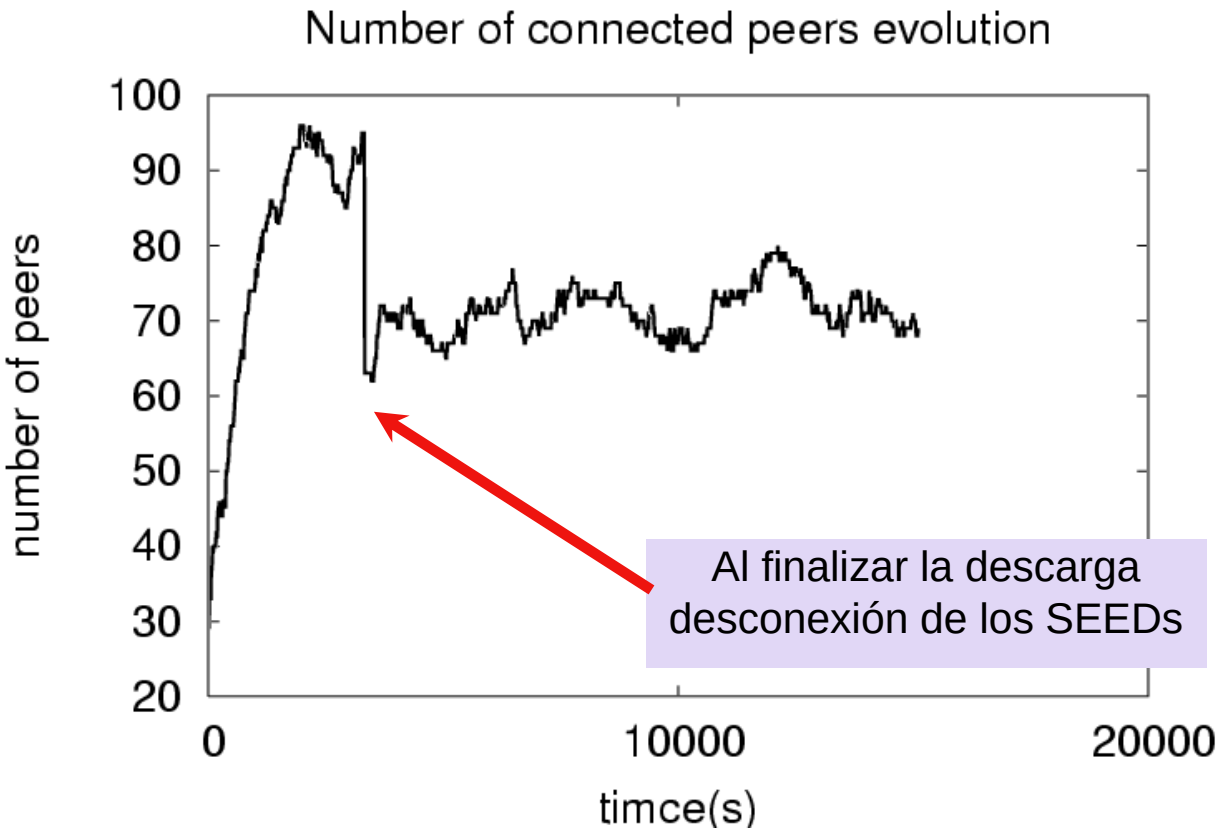
- El comienzo de la descarga se podría mejorar.
  - Puede tardar unos minutos en conseguir piezas para negociar.



Hasta aqui recibimos por *optimistic unchoke*

Siguiendo a un peer

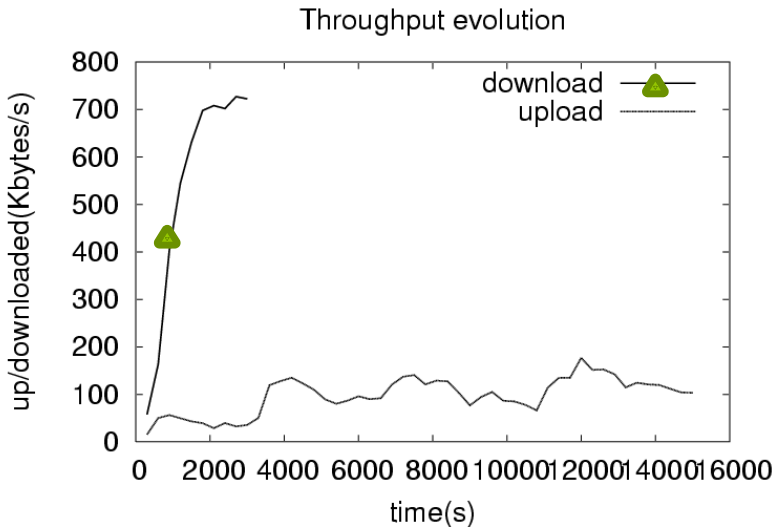
- Vecinos conectados a mi peer.



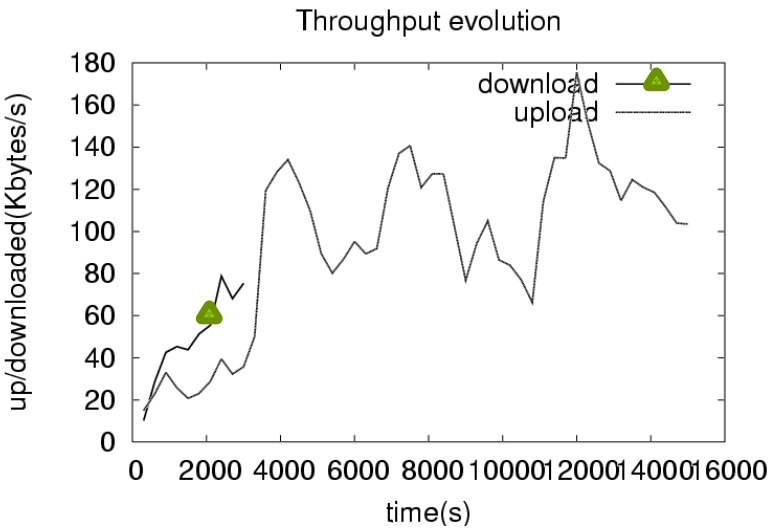
Siguiendo a un peer

- Parece que recibimos más de lo que enviamos
  - Es normal porque los seeds sólo envían.
- Incluso sin contar lo que recibimos de los seeds recibimos más
  - ¿Es normal o es que tenemos ventaja?

Throughput Up y Down

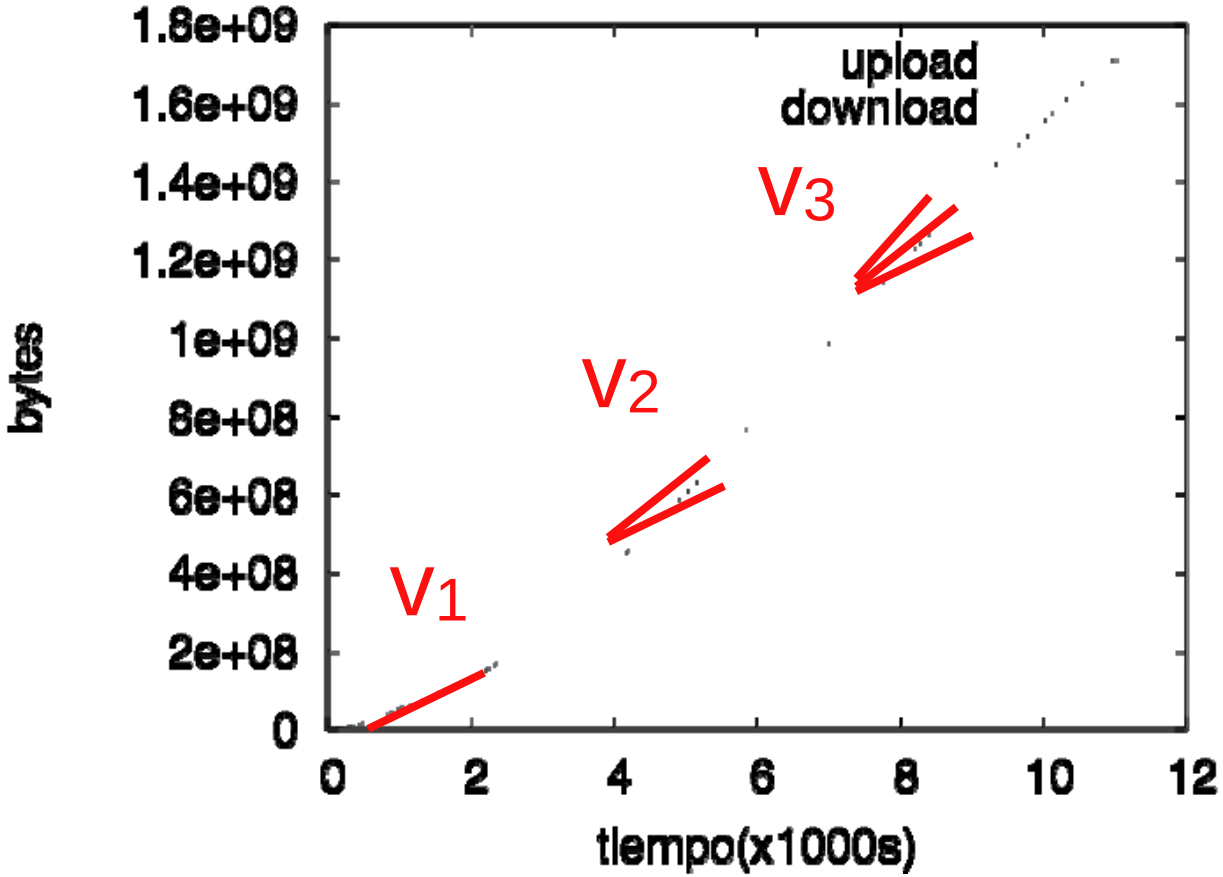


Throughput Up y Down
Sin contar SEEDS



Siguiendo a un peer

- En general la velocidad es mayor a más tiempo de estancia en el torrente.
  - ¿Por qué los peers más veteranos reciben más rápido?
  - ¿Tiene que ver con que tengan más porcentaje del fichero?



Siguiendo a un peer: conclusiones

- Arranque lento hasta obtener piezas para negociar (minutos en descargas de horas).
- Los seeds contribuyen en gran medida sobre todo para peers con buen BW de acceso
  - BitTorrent es muy eficiente creando rápido SEEDS.
- Peers veteranos tienen ventaja sobre los recien llegados
  - Tienen mayor número de piezas raras.
  - Son capaces de mantener el tit-for-tat por más tiempo.
  - Capturan más fácilmente el flujo de otros peers porque les envían más datos.
- En general la velocidad de descarga aumenta con el tiempo.
- En general los peers que llegaron antes acaban antes.
  - El torrente es aproximadamente FIFO.

4 Conclusiones

- Si bien un paradigma P2P es más escalable y mejora al disponibilidad del servicio, normalmente
  - Provee una peor calidad del servicio (lentitud de descarga)
  - Está sometido a ataques (polución, falta de colaboración)
- Un paradigma cliente-servidor se puede aproximar a las ventajas de uno P2P
  - Granjas de servidores
  - Esquemas de balanceo
  - Replicación de servidores en múltiples localizaciones

Referencias

- [Tanenbaum]
  - Capítulo 1, secciones 1.4.3 y 1.5
  - Capítulo 2
- [Kurose]
  - Capítulo 2, sección 2.6
- M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. A. Hamra, and L. Garcés-Erice. Disecting bittorrent: Five months in a torrent's lifetime. In Proceedings of the Passive and Active Network Measurement 5th International Workshop, Antibes Juan-les-Pins, France, April 2004.
- T Karagiannis, A Broido, N Brownlee, K Claffy, "Is P2P dying or just hiding?".IEEE Globecom 2004.
- Keith Ross and Dan Rubenstein. P2P Systems. Tutorial at INFOCOM 2007