Distributed Systems

P2P Content Sharing

Dynamic and Distributed Information Systems

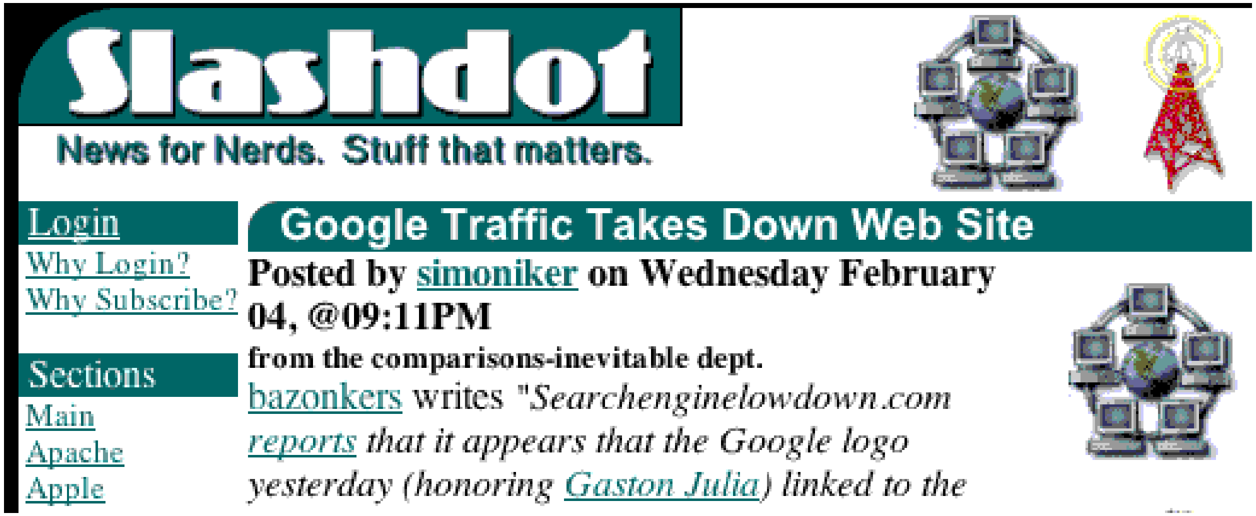
Today's Agenda

- Motivation behind Decentralized Content Distribution
- Napster
- Gnutella
- FastTrack
- BitTorrent
  - Measurements and Evaluation

A problem…

- Feb 3, 2004: Google linked banner to "julia fractals"
- Users clicking directed to Australian University web site
- …University's network link overloaded, web server taken down temporarily…

The problem strikes again!

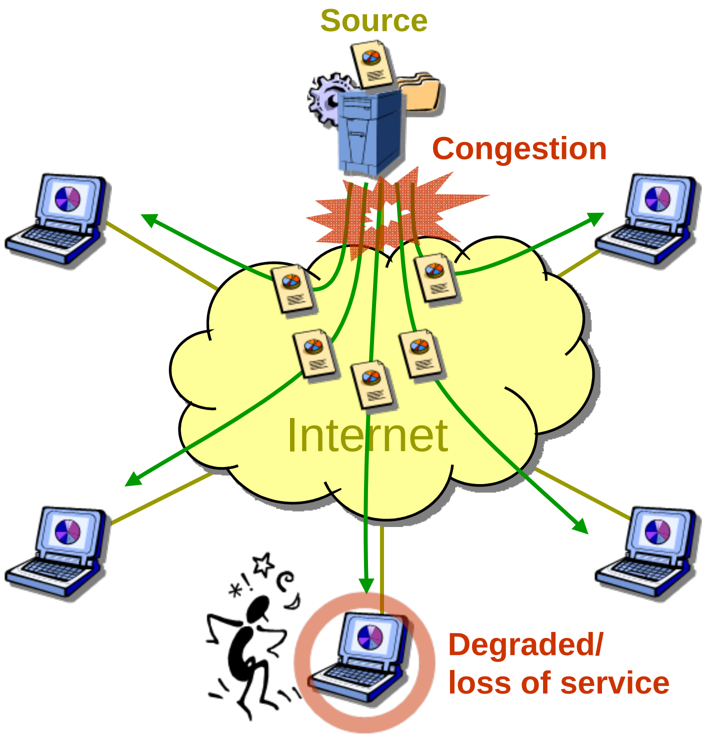


- Feb 4, 2004: Slashdot ran the story about Google
- …Site taken down temporarily…again

Context and Problem

- A growing number of well-connected users access **increasing amounts of content**
- But interest in content is often "Zipf" distributed (small fraction of very popular content)
- Servers and links are **overloaded**
  - Number of clients
  - Size of content
  - "Flash crowd" (e.g., 9/11)
- Tremendous engineering (and cost!) necessary to make server farms **scalable** and **robust**

Traditional client/server content distribution



Problem: scalable distribution of content

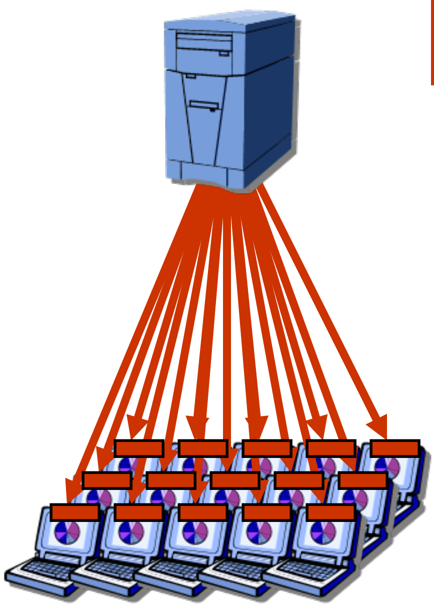
Real-World Scenarios

- Quick distribution of critical content
  - E.g., antivirus definitions
- Efficient distribution of large content
  - E.g., nightly update of a bank's branches, promotional movie from manufacturer to all car dealers
- Distribution of streaming content
  - E.g., live event, Internet TV
- Classical approaches have high cost
  - Source over-provisioning (for peak demand)
  - Highly organized Content Delivery Networks (CDNs)
    - Akamai, Digital Island, Mirror Image, etc.
- **Novel approach:** Cooperative CDNs

Intuition

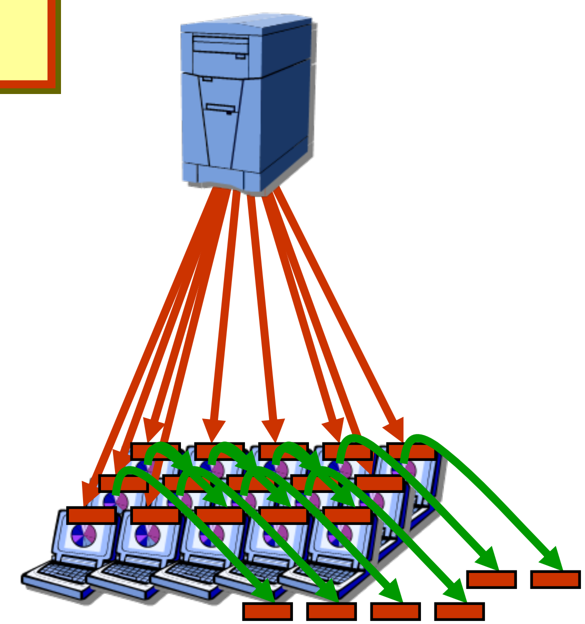
Source server: **100 Mb/s**
Clients: **10 Mb/s**
<u>1. Antivirus update</u>
100,000 clients
File: **4 MB**
<u>2. Daily database update</u>
1000 clients
File: **600 MB**

1. 9h:52m
2. 14h:48m



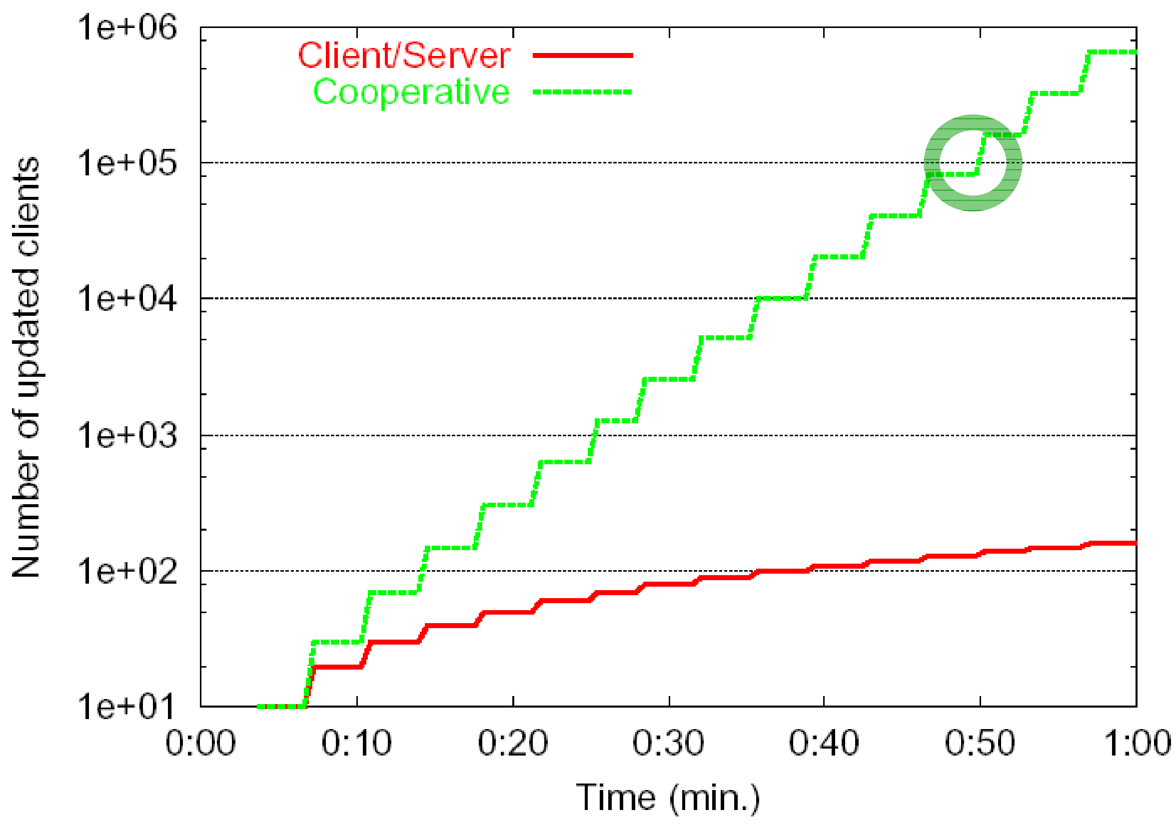
Client/Server

1. 52s
2. 09m:54s



Cooperative

Intuition


Client/Server
Cooperative
1e+06
1e+05
1e+04
1e+03
1e+02
1e+01
Number of updated clients
0:00
0:10
0:20
0:30
0:40
0:50
1:00
Time (min.)

Cooperative Distribution

- Principle: Utilize bandwidth of edge computers
- Self-scaling network:

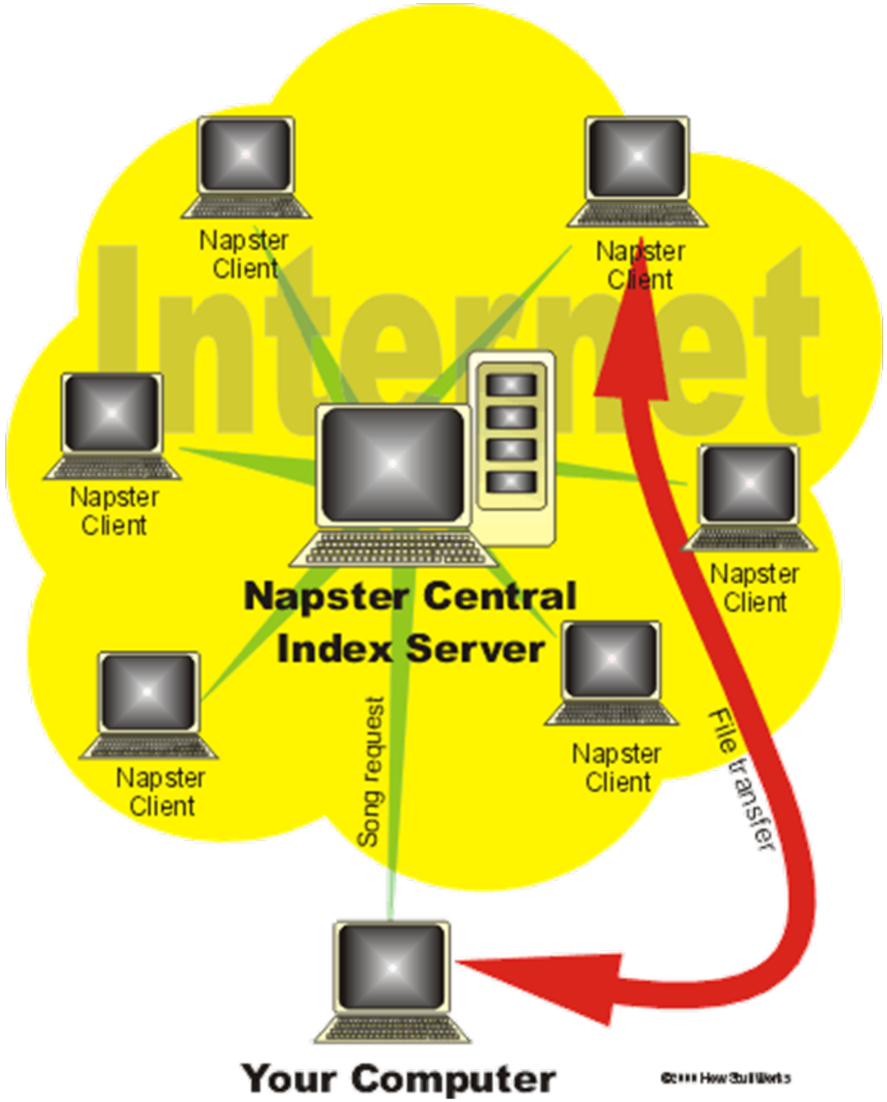
  more clients → more aggregate bandwidth → more scalability
- Self-organizing:

  robust against failures and flash crowds
- How well does it work in practice?

Napster

Napster: Centralized P2P

- Peer-to-peer
  - relies on a central index
  - but files don't reside on a central server
- Four steps:
  - Connect to Napster server
  - Upload your list of files (push) to server
  - Give server keywords to search the full list
  - Select "best" of correct answers (based on pings)



Napster: Clever Design

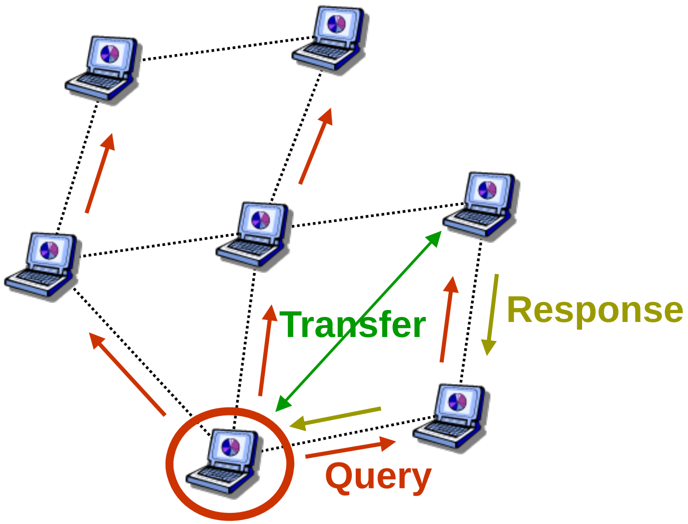
- Centralized user and song database
  - Quick searching
    - Faster/better than Gnutella
  - Users come and go
    - User/search database continually updated
  - Automatic file sharing
    - Easy to use file server

- ***But…***
  - Single server to bring down
  - This centralization is ultimately its downfall

Gnutella

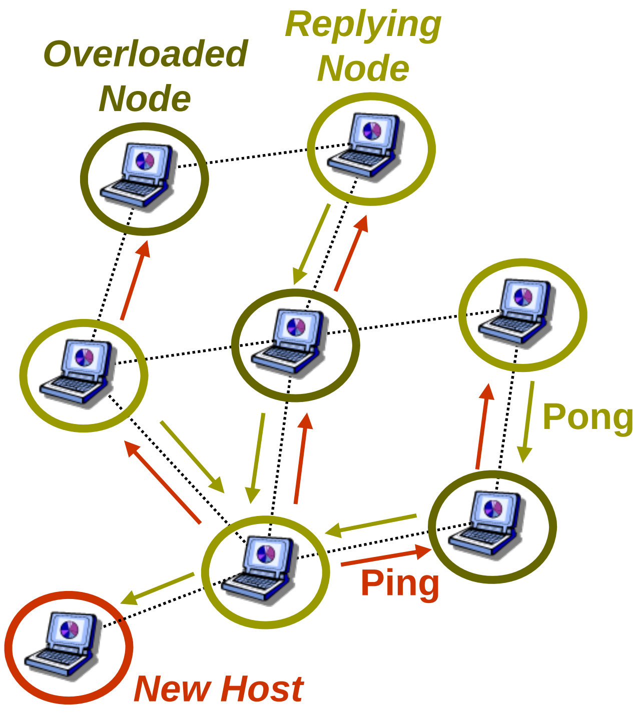
Gnutella: Pure P2P

- Focus: decentralized method of searching
  - harder to "pull the plug"
- Search by flooding
  - If you don't have the file you want, query 7 of your partners (neighbors)
  - If they don't have it, they contact 7 of their neighbors, for a maximum hop count of 10
  - Requests are flooded — may lead to scalability problems
  - No looping but packets may be received twice
- Querying node is sent responses with list of matching files and IP addresses
- File transfer is direct (no anonymity)



Gnutella: Overlay Maintenance

- Plug-in to a host and send a *broadcast ping*
  - Can be any host (hosts transmitted through word-of-mouth or host-caches)
  - Host broadcasts ping message with TTL of 7
- Hosts that are not overloaded respond with a *routed pong*
  - Gnutella caches IP addresses of replying nodes



Gnutella: Problems

- 24 hour survey showed:
  - 70% of people shared no files
  - 50% of search responses from top 1% of hosts
  - Reverting to client/server
    - Suddenly not so hard to shut down!
  - Verified hypotheses
    - H1: A significant portion of Gnutella peers are free riders
    - H2: Free riders are distributed evenly across domains
    - H3: Often hosts share files nobody is interested in
- Non-standard implementation
  - People implement their own Gnutella clients
  - Some clients are dodgier than others

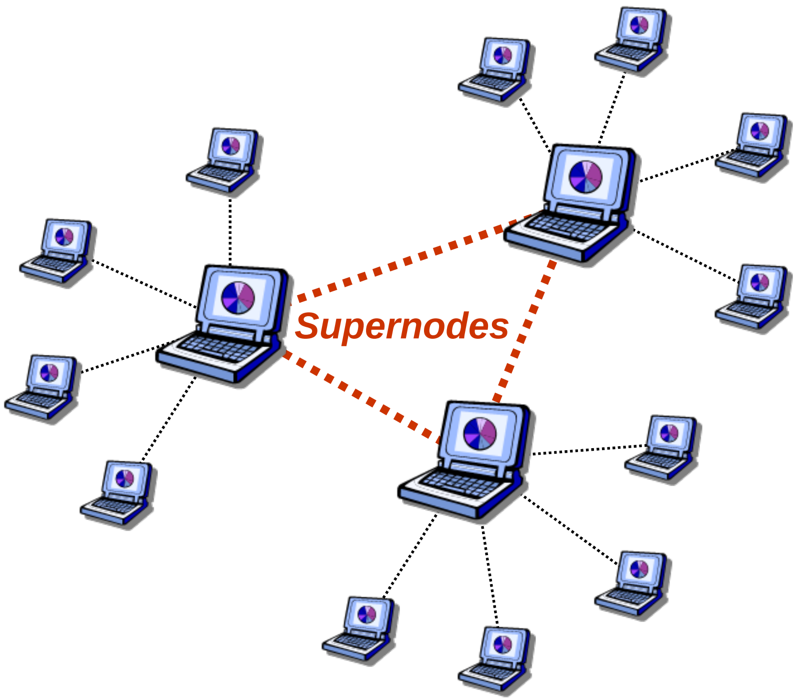
FastTrack (KaZaA)

FastTrack (KaZaA): Hybrid P2P

- Software
  - Proprietary
  - Files and control data encrypted
  - Everything in HTTP request and response messages
- Architecture
  - Hierarchical
  - Cross between Napster and Gnutella

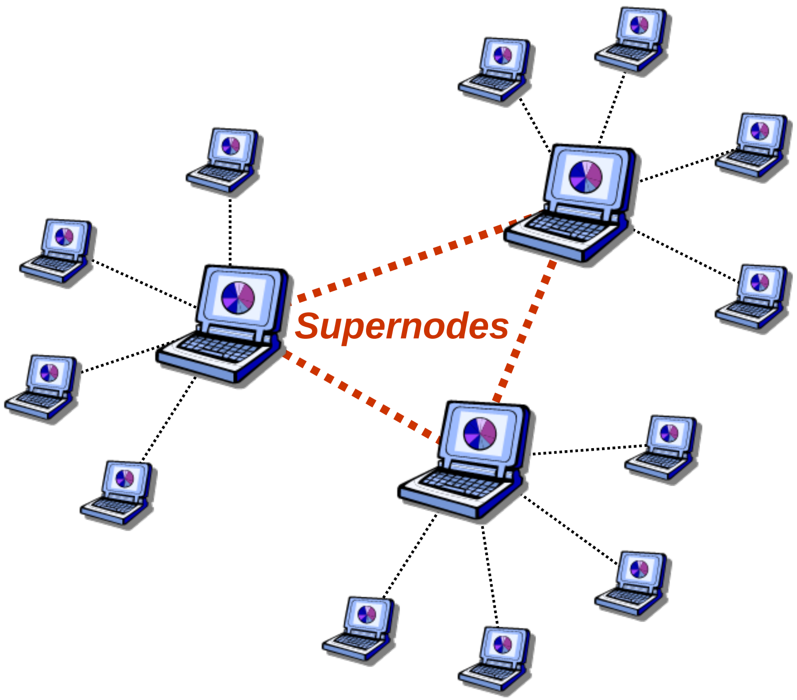
KaZaA: Architecture

- Each peer is either a supernode or is assigned to a supernode
  - Nodes with more bandwidth and that are more available are designated as supernodes
  - Each supernode knows about many other supernodes (almost mesh overlay)
  - Supernodes act as mini-Napster hubs tracking the content and IP addresses of their descendants
  - Guess: ~10,000 supernodes with 200-500 descendants each
  - Dedicated user authentication server and supernode list server



KaZaA: Queries

- Node first sends query to supernode
  - Supernode responds with matches
  - If x matches found, done
- Otherwise, supernode forwards query to subset of supernodes
  - If total of x matches found, done
- Otherwise, query further forwarded
  - Probably by original supernode rather than recursively



KaZaA: Overlay Maintenance

- List of potential supernodes included within software download
- New peer goes through list until it finds operational supernode
  - Connects, obtains more up-to-date list
  - Node then pings 5 nodes on list and connects with the one with smallest RTT
- If supernode goes down, node obtains updated list and chooses new supernode

KaZaA: Corporate Structure

- Software developed by FastTrack in Amsterdam
- FastTrack also deploys KaZaA service
- FastTrack licenses software to Music City (Morpheus) & Grokster
- Later, FastTrack terminates license, leaves only KaZaA with killer service
- International "cat-and-mouse" game
- Summer 2001, Sharman networks, founded in Vanuatu (small island in Pacific), acquires FastTrack
  - Board of directors, investors: secret
- Employees spread around, hard to locate
- Code in Estonia

BitTorrent

Dynamic and Distributed
Information Systems

BitTorrent

- Designed for the transfer of large files to many clients
  - Based on **swarming**: a server sends different parts of a file to different clients, and the clients exchange chunks with one another
- Terminology
  - One session = distribution of a single (large) file
  - Seeder = a node that has the whole file
  - Leecher = a node still downloading the file
- Elements
  - An ordinary web server
  - Torrent file: A static “meta-info” file
  - A tracker
  - A seeder (an initial client with the complete file)
  - On end user side: web browser + BitTorrent client

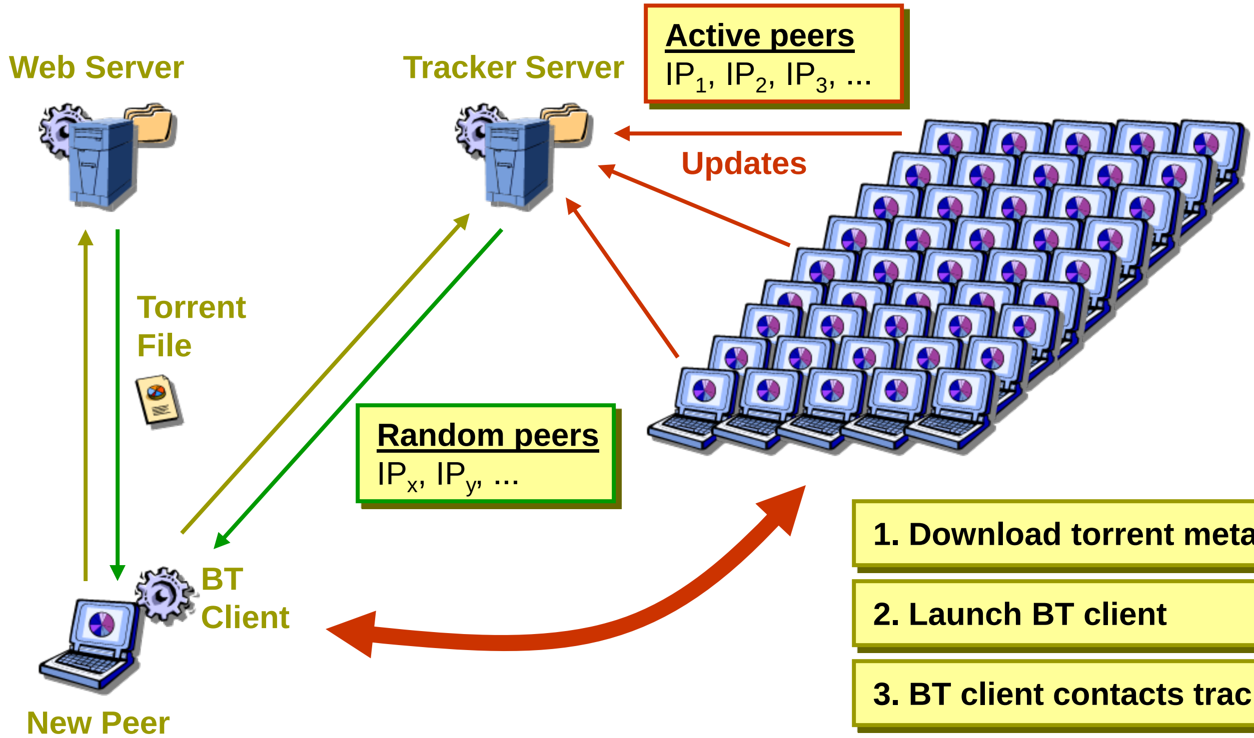
The Torrent file contains:

- Tracker address (IP + port)
- Bytes per chunk
- Number of chunks
- For each chunk, the SHA1 hash value
  - Helps validate the correctness of downloaded chunks

Session Initiation

- Make the torrent file available on a web server
  - The torrent file contains the IP address of the tracker
- The tracker tracks peers
  - Initially, it knows at least one seeder
  - Matches new peers with existing ones, to allow them collaborate
  - Usually does not run on the same machine as the Web server
- On the client side
  - Client contacts the tracker (through HTTP or HTTPS)
  - The tracker returns a set of active peers (typically 40, leechers & seeders)
  - Clients regularly report state (% of download) to tracker

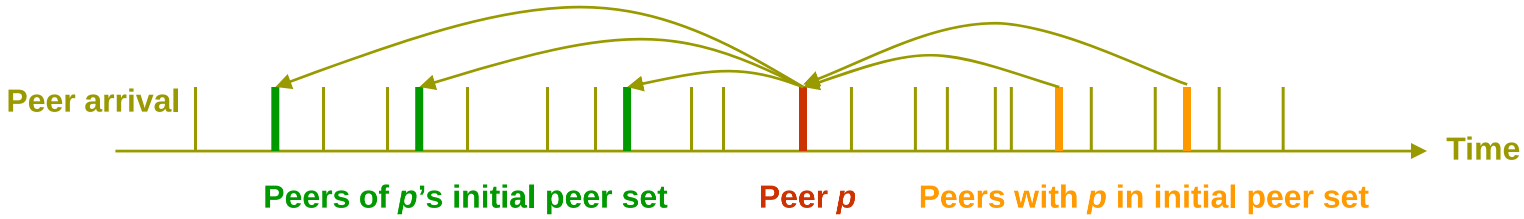
Joining a BT Session



1. Download torrent meta-info
2. Launch BT client
3. BT client contacts tracker (HTTP)
4. Tracker picks 40 peers at random for the new client
5. BT client cooperates with peers returned by the tracker

Peer Sets

- Tracker picks peers at random in its list
- Once a peer is incorporated in the BitTorrent session, it can also be picked to be in the peer set of another peer
- This technique allows a wide temporal diversity
  - A peer knows both **older peers and newcomers**!
  - Ensures transfer of chunks between "generations"
- Note: a peer communicates with its initial peer set and the other peers that contacted it but NOT with other peer sets



File Transfer Algorithm

- Initial file broken into chunks (typically 256 kB)
  - The torrent file contains the SHA1 hash for each chunk: allows to check integrity of each chunk
- Reports sent regularly (at start-up, shutdown, and every 30 minutes) to tracker
  - Unique peer ID, IP, port, quantity of data uploaded and downloaded, status (started, completed, stopped), etc.
- Peers connect with each other over TCP, full duplex (data transit in both directions)
  - Upon connection, peers exchange their list of chunks
  - Each time a peer has downloaded a chunk and checked its integrity, it advertises it to its peer set

Connection States

On each side, a connection maintains two variables:

- **"Interesting":** you have a chunk that I want
  - Allows a peer to know its possible clients for upload
- **"Chocked":** I don't want to send you data at the time
  - Possible reasons: I have found faster peers, you did not/can't reciprocate enough, …

Chunk Selection Policy

- Which missing chunk should we request from other peers?
- Simple strategy: **random** selection
  - Choose at random among chunks available in peer set
  - Randomness ensures diversity
- Biased strategy: peers apply the **rarest-first** policy
  - Choose the least represented missing chunk in the peer set
  - Rare chunks can more easily be traded with others
  - Maximize the minimum number of copies of any given chunk in each peer set
- BitTorrent uses rarest-first policy except for newcomers that use random to quickly obtain a first block

Peer Selection Policy

- Serving too many peers simultaneously is not efficient
  - BitTorrent serves a few (around 4 or 5) hosts in parallel
- Which hosts to serve?
  - Seeders' policy: The ones that offer the **best upload rates**
  - Leechers' policy: The ones that also serve us: **tit for tat**
  - Choke the rest peers
- Can there be any better hosts?
  - Reconsider choking/unchoking every 10 sec (long enough for TCP to reach steady state)
  - Optimistically unchoke a random peer every 30 sec to give a chance to another host to provide better service
  - Newcomers have less data to offer → give them "priority" in the optimistic unchoke

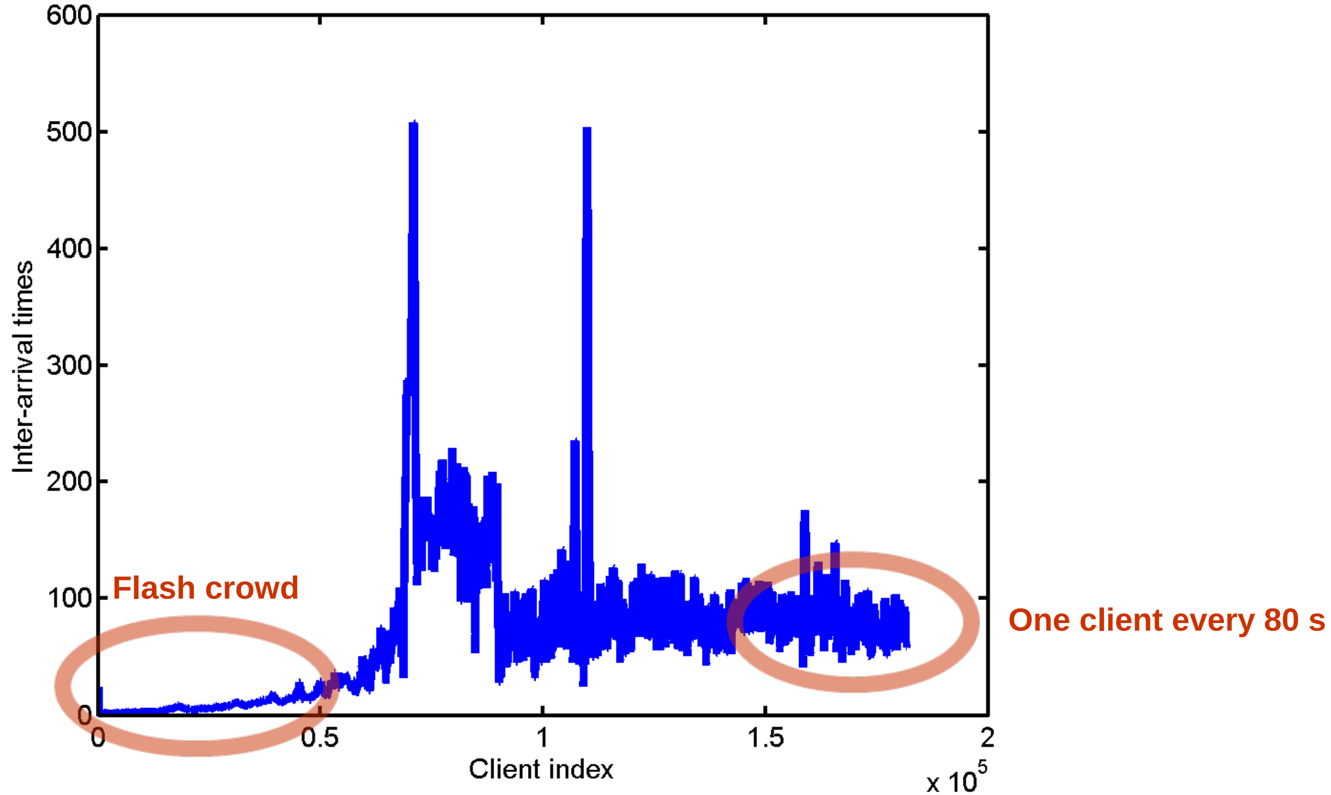
BitTorrent:
Measurements & Evaluation

BitTorrent Study

- Five months (April to August 2003) **tracker log** of a very popular BT session
  - Linux RedHat 9
  - 1.77 GB
  - Log contains all the reports of all the clients (ID, IP, amount of bytes uploaded and downloaded)
- In addition, an **instrumented client** observed a given peer set for three days
  - Log contains blocks uploaded to and downloaded by each host (each time a host has a new block, it advertises its peer set)
  - Exhibits the behavior of BitTorrent during the download phase and once the client becomes a seeder

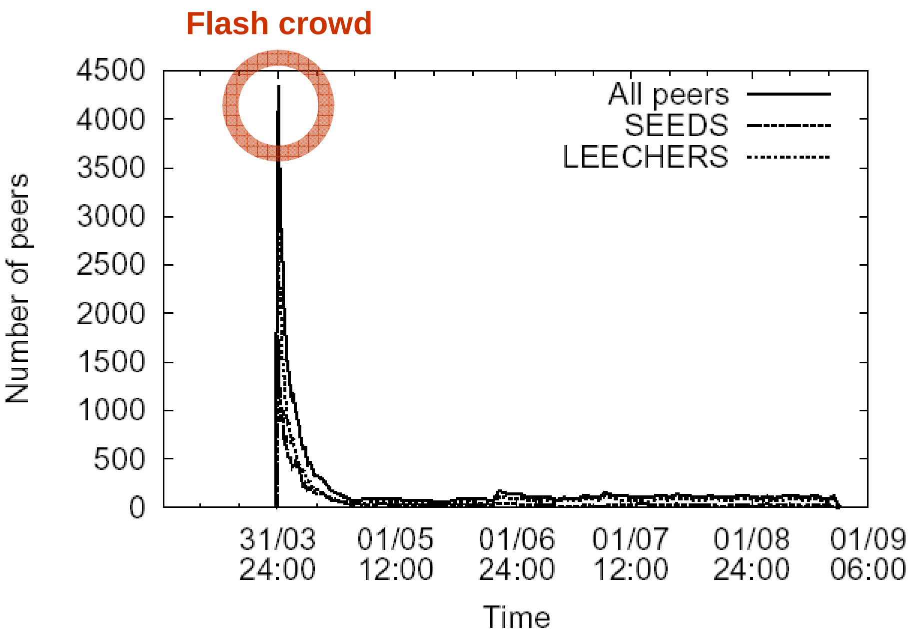
Tracker Log

- 180,000 clients during the 5 five months period
- Initial flash crowd: 51,000 clients in the first 5 days

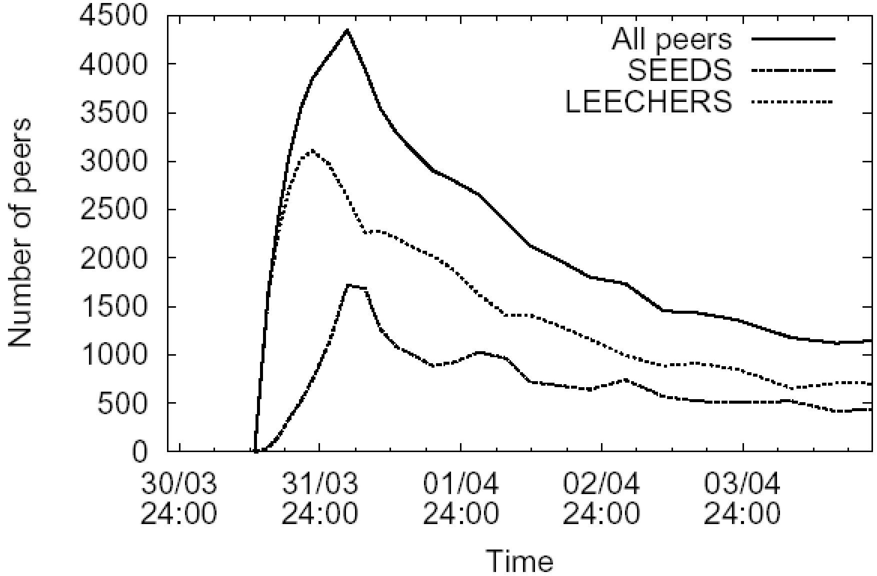


Tracker Log: Number of Clients

- Reaches 4000+ active clients on the first day
- Remains in the interval [100,200] later



Complete trace (5 months)



First 5 days

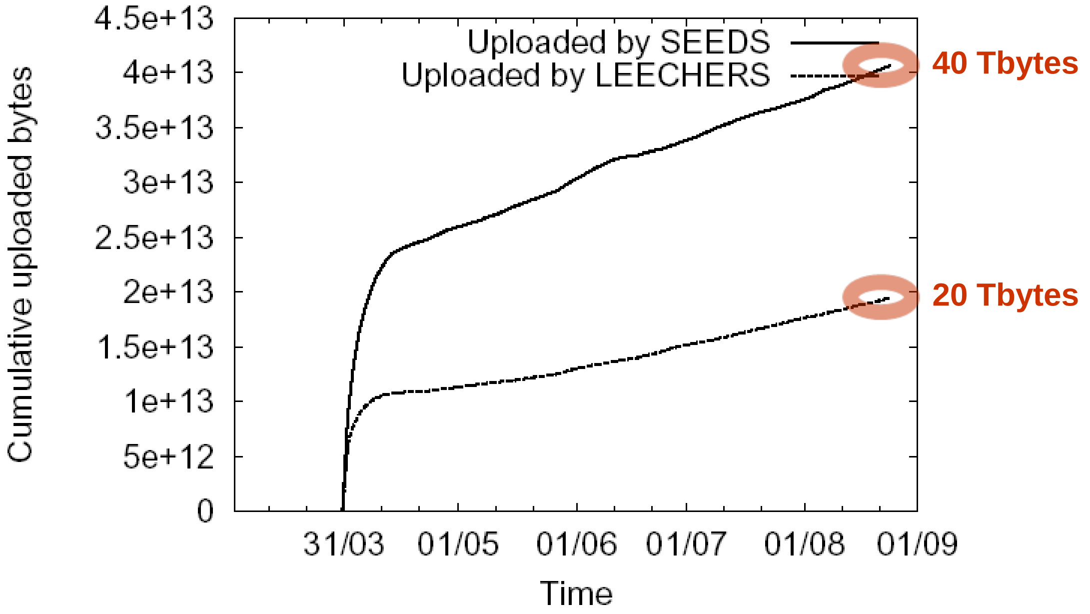
Tracker Log: Clients' behavior

Clients are very altruistic

- When they are leechers
  - They have no choice due to tit-for-tat
- Once download is completed
  - Clients stay on average **3 hours** after download
  - The transfer is long, may complete overnight
  - The content is legal (RIAA will not sue!)
  - The users are very kind ☺

Tracker Log: Seeders vs. Leechers

- Presence of seeders is a key feature of BitTorrent
  - Over the 5 months they contributed twice as much volume as leechers

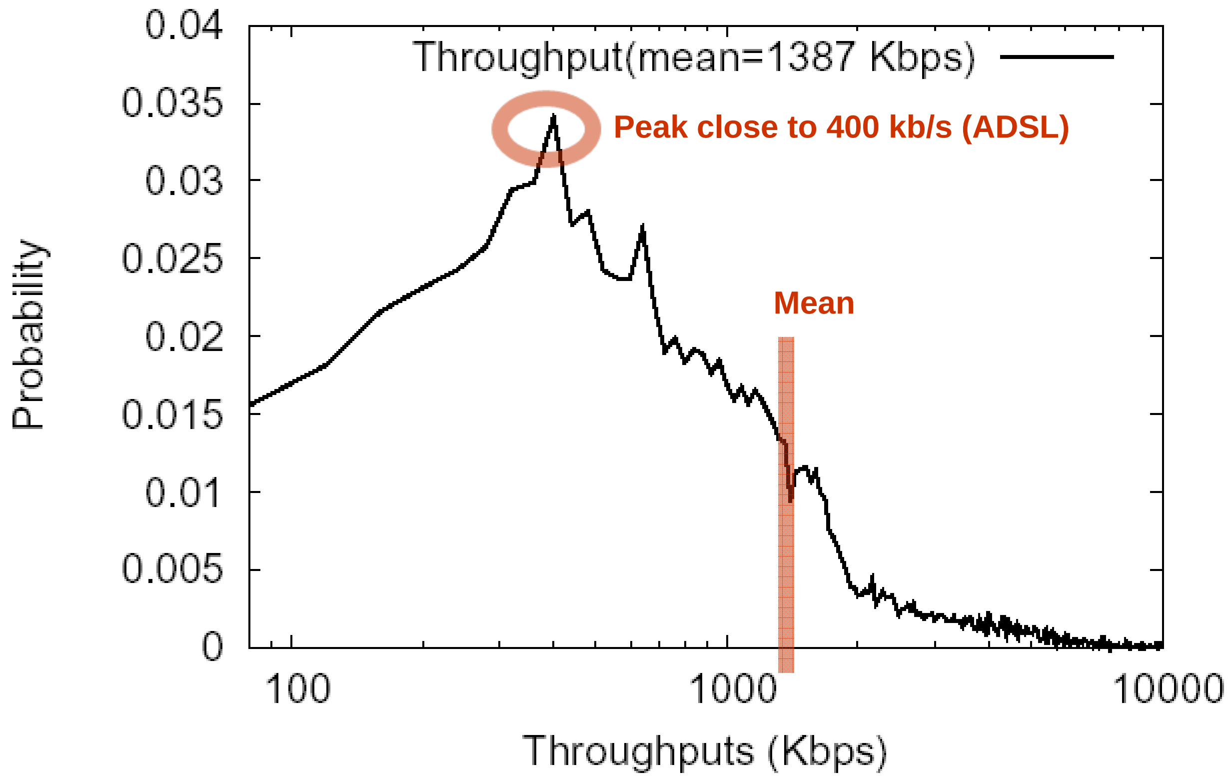


Tracker Log: BT vs. Mirroring

- Throughput per leecher is always **above 500 kb/s**
  - At least ADSL client
- Aggregate throughput of system (sum over all leechers at each instant) was **higher than 800 Mb/s**
  - More than 80 mirrors, each sustaining a 10 Mb/s service
- Considering only the 20,000 hosts that completed download in a single session (BT allows resume)
  - Throughput is better than average: **1.3 Mb/s**
  - Average download time is **30,000 s** (8.3 h)
  - 1.77 GB / 1.3 Mb/s = 10,000s (2.7 h)
  - Conclusion: a **high variance** in download throughputs!

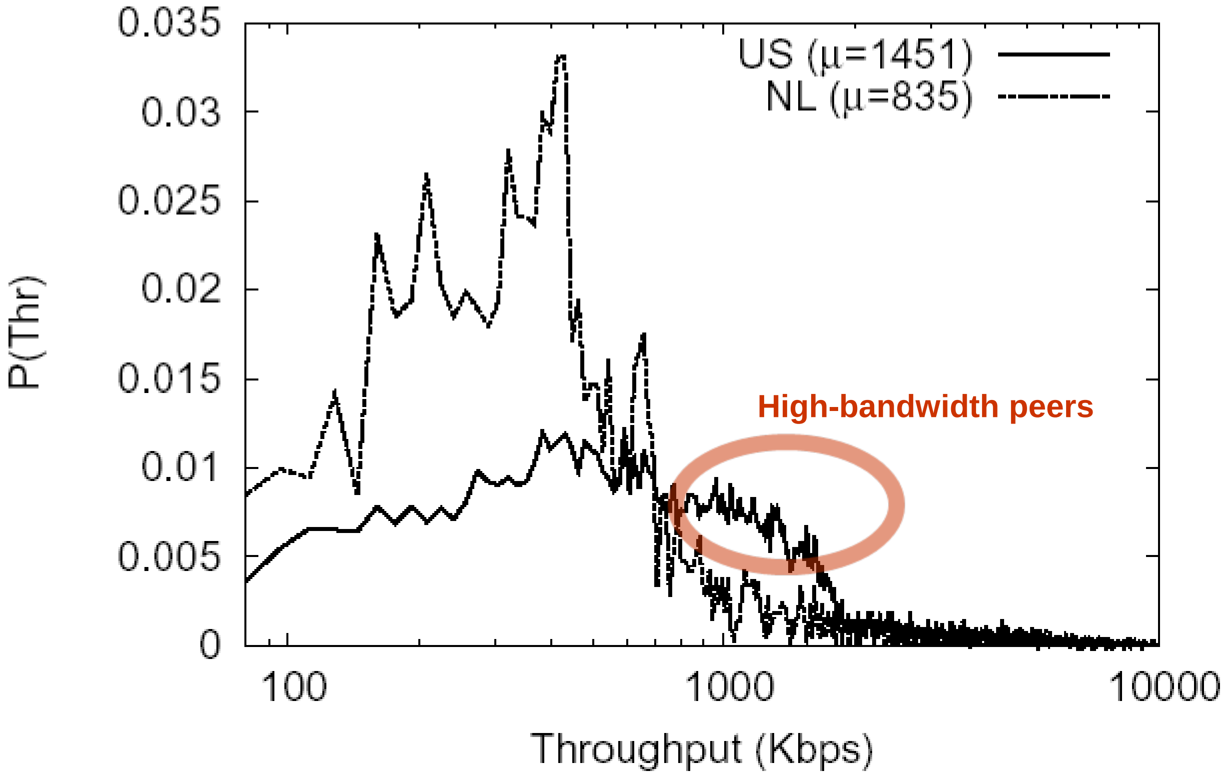
Tracker Log: Complete Sessions

- Peak value around ADSL speed
- Some hosts have very high bandwidth



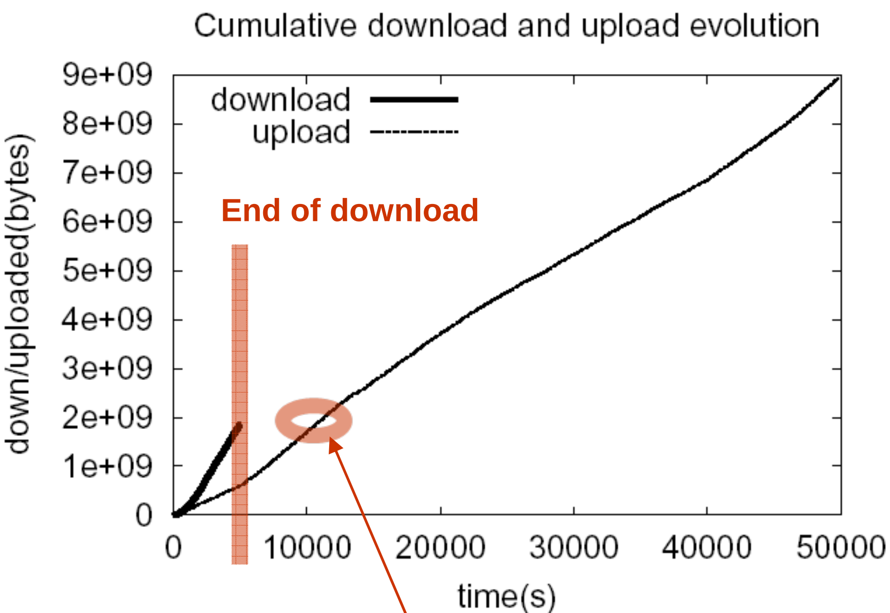
Tracker Log: US vs. Europe

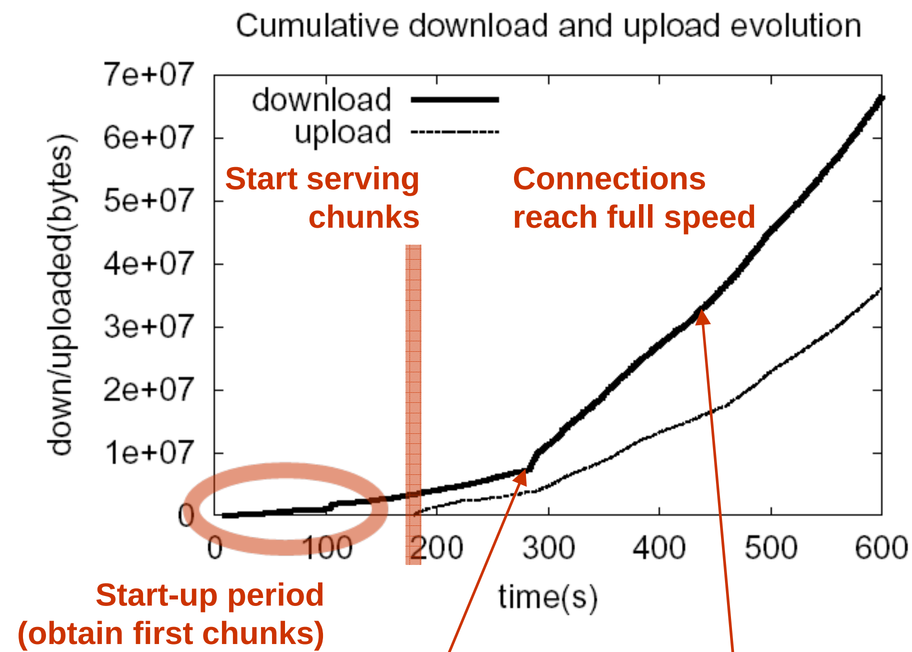
- In the first 4 weeks: 45% from US, 15% from Europe
  - US clients have better access links than European clients



Client Log: Upload and Download

Client never gets stalled: we always find peers to serve and download chunks from → good efficiency



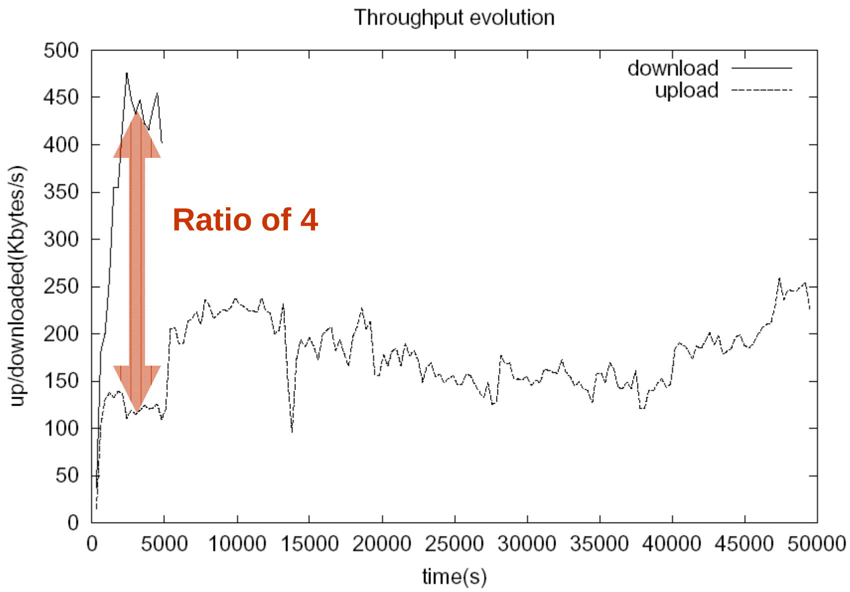


We uploaded as much as we downloaded after 10,000 s = twice the download time

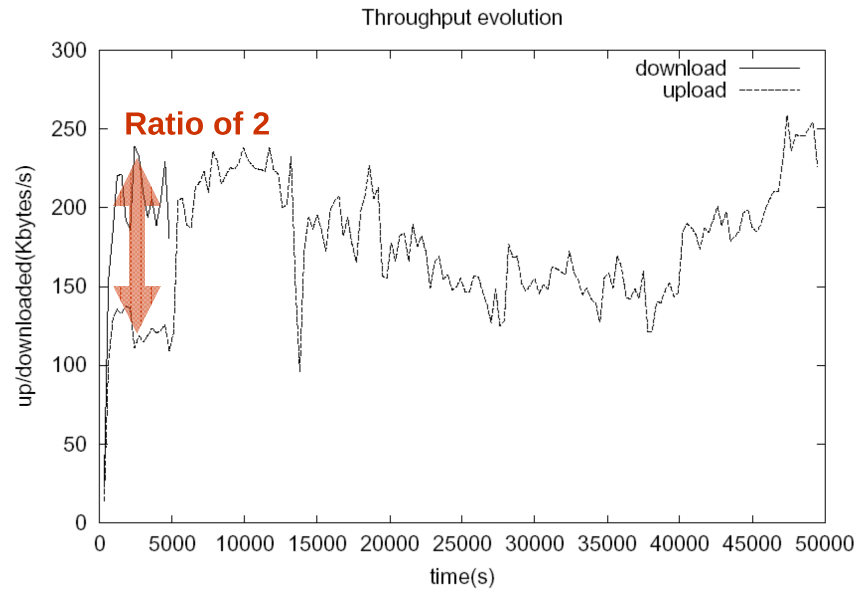
Cooperation is enforced: the download rate increases because the upload rate increases

Client Log: Tit-for-Tat

- Client received more than it gave, even if we do not account for seeders traffic
  - Probably due to this client's good download capacity and to tit-for-tat enforcement



(a) All peers



(b) Non-seeds only

Client Log: Tit-for-Tat

- Who gave the file, seeders or leechers?
  - 40% from seeders and 60% from leechers
  - 85% of the file was provided by only 25% peers
  - Most of the file provided by peers that connected to us (not from original peer set)
- How good is the tit-for-tat policy?
  - Two conflicting goals
  - Must enforce cooperation among peers
  - Must allow transfer even if bandwidth not perfectly balanced
    - Example: I don't give you anything because I can send you at 100 kb/s whereas you can only send at 80 kb/s

Summary

- BitTorrent seems very efficient for highly popular downloads
  - Still, its performance might be affected if clients do not stay long enough as seeders, e.g., in case of illegal content…
  - What happened to 160,000 incomplete downloads?
- BitTorrent is clearly able to sustain large flash crowds
- Some open questions
  - Could we do better by using different peer and chunk selection strategies?
  - Could we do better if all peers arrive at the same time (e.g., antivirus update)?
  - Could we do better if peers have symmetric bandwidth (e.g., private network)?