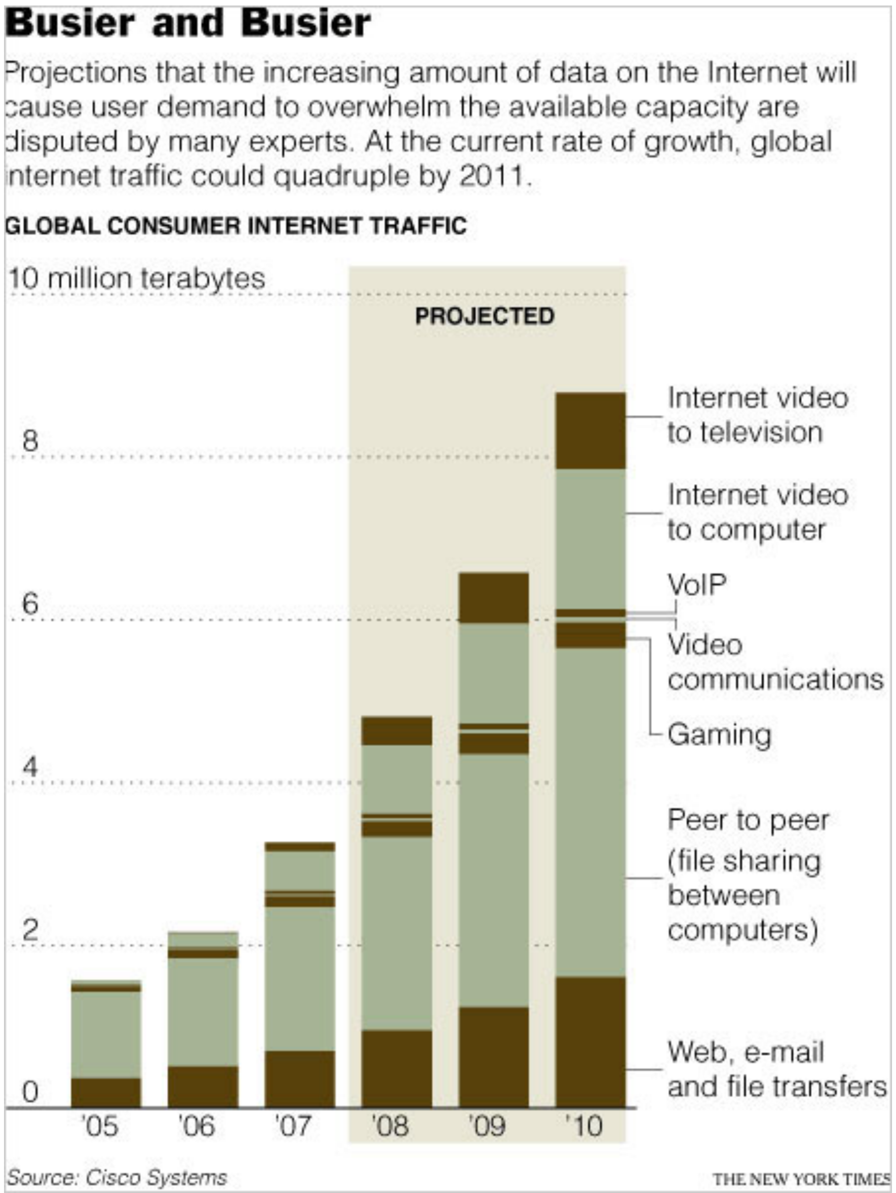
Last Lecture

- SMTP

This Lecture

- Peer-to-Peer (P2P) Applications

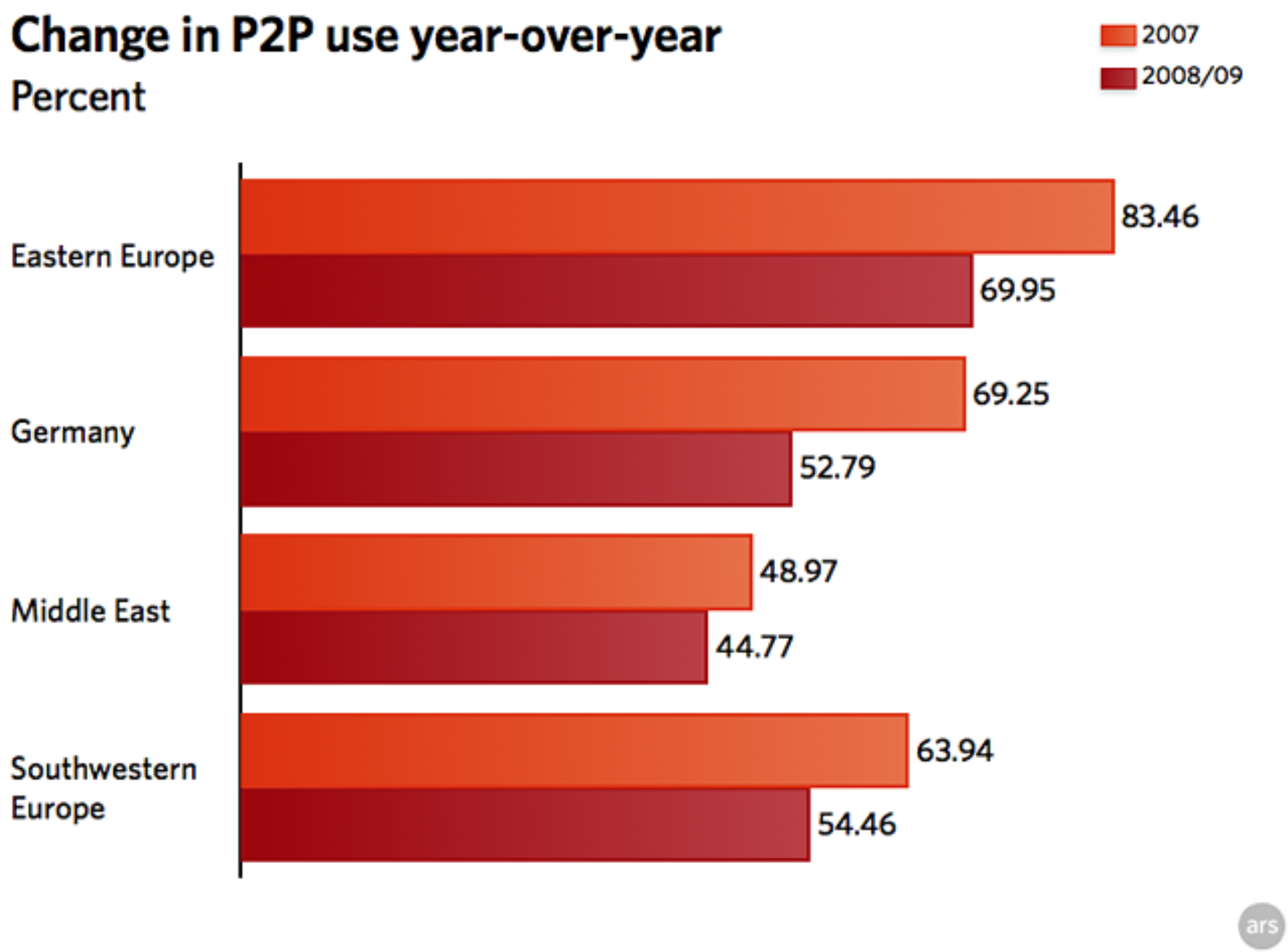
Internet Traffic Trend



Don't take the numbers too seriously

08 vs. 07: P2P, porn down; games and Flash up

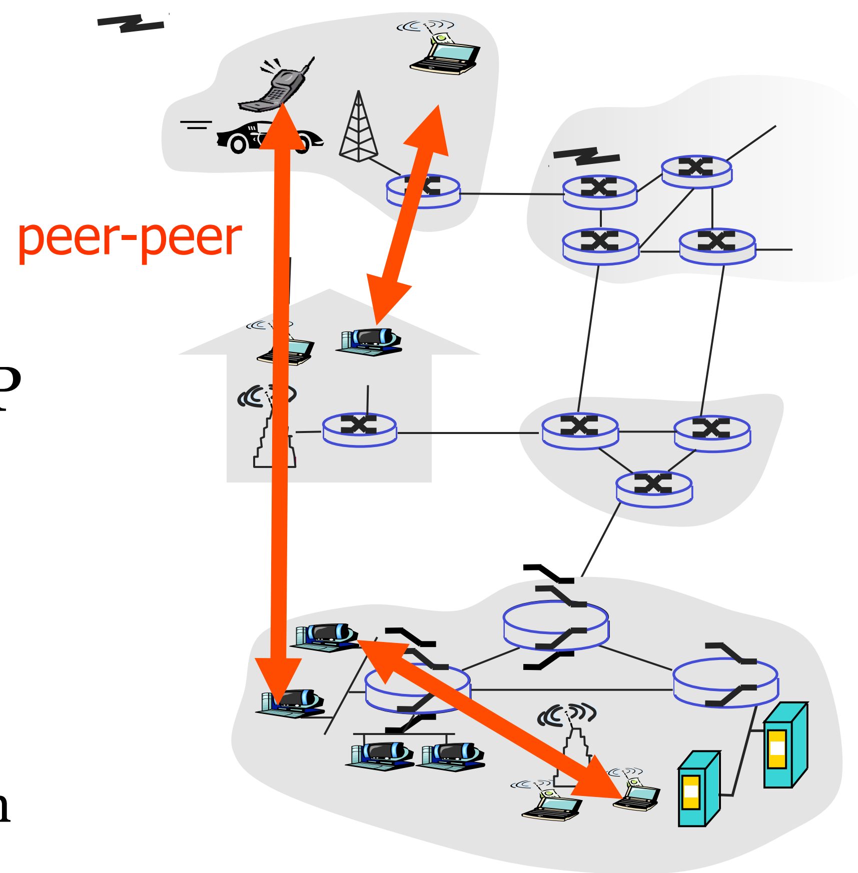
http://arstechnica.com/web/news/2009/02/internet-traffic-report-p2p-porn-down-games-and-flash-up.ars



What is P2P Architecture?

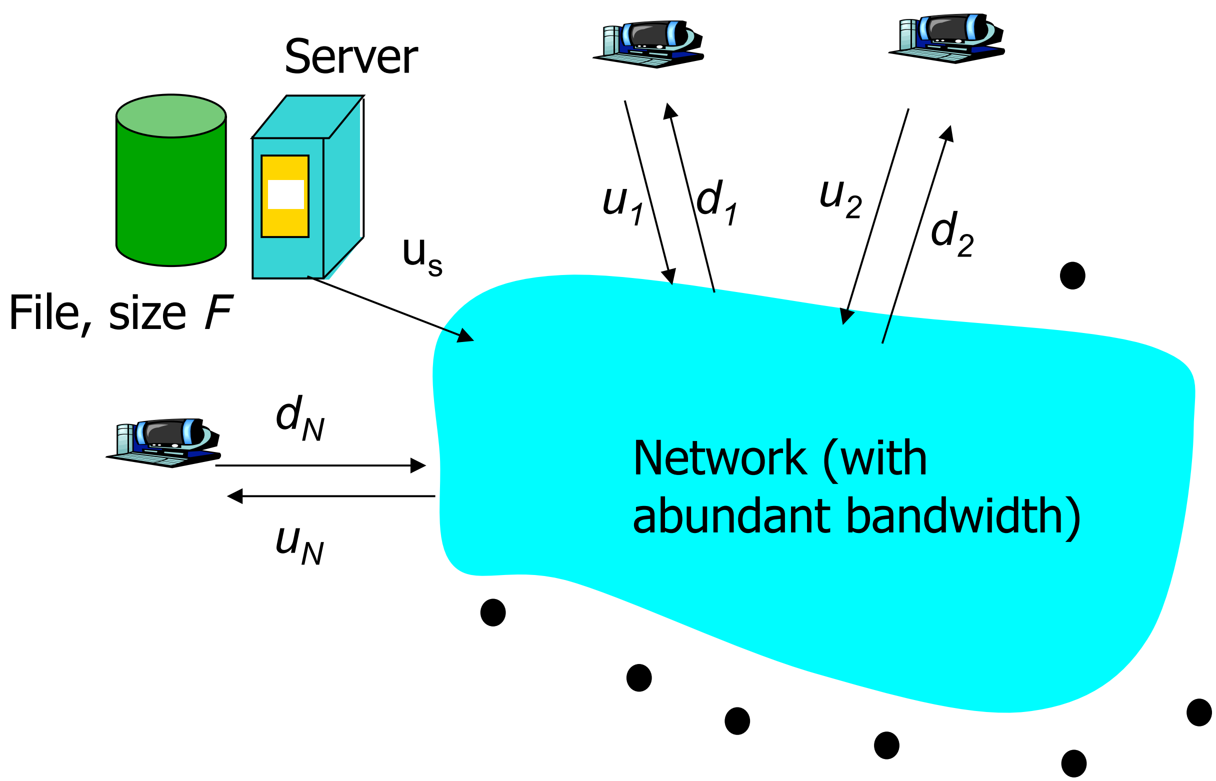
- *No* always-on server
- Arbitrary end systems directly communicate
- Peers are intermittently connected and change IP addresses

- *Apps that use P2P*
  - File sharing
  - Searching for information
  - Internet telephony (e.g., Skype)



Why P2P? Client-Server vs. P2P

Question : How much time to distribute file from one server to N *peers*?



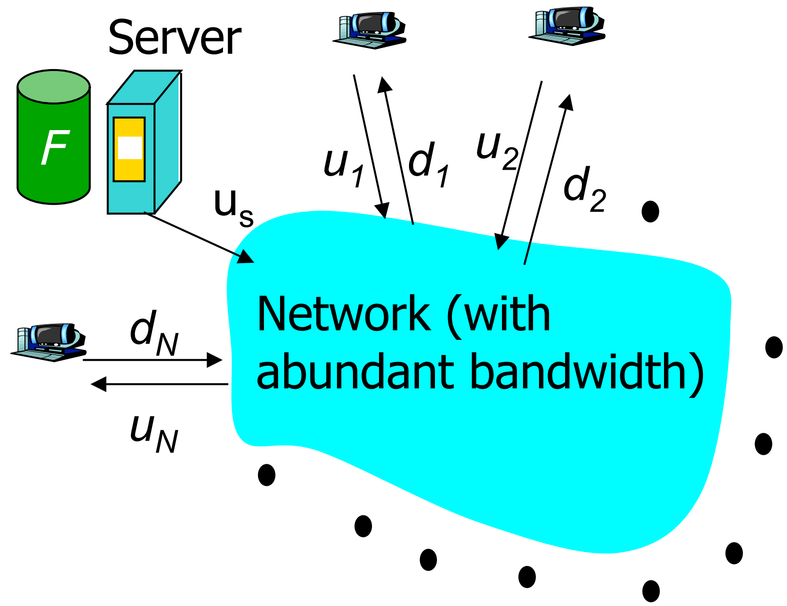
u_s: server upload bandwidth

u_i: peer i upload bandwidth

d_i: peer i download bandwidth

Distribution Time: Client-Server Arch.

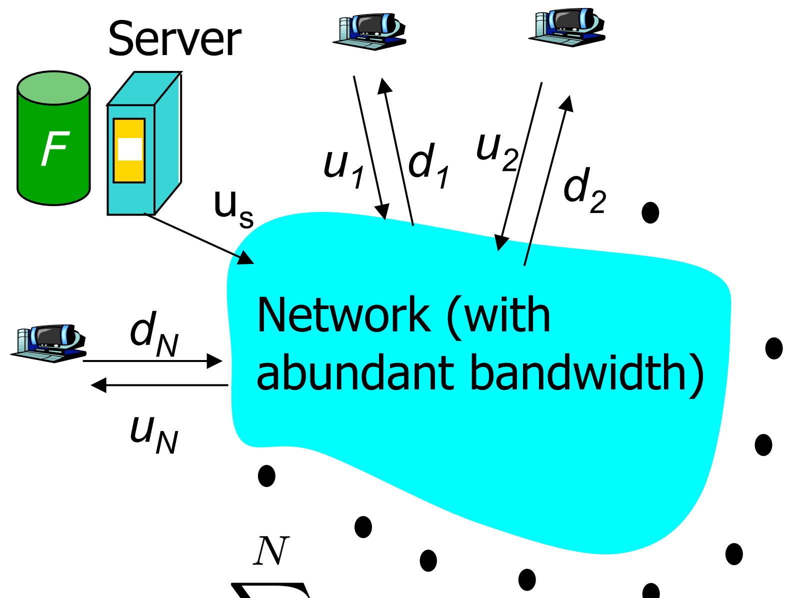
- Server sequentially sends N copies:
  - NF/u_s time
- Client i takes F/d_i time to download



Time to distribute F to N clients using client/server approach $\geq d_{cs} = \max\left\{\frac{NF}{u_s}, \frac{F}{\min\{d_i\}}\right\}$

increases linearly in N (for large N, millions in practice)

Distribution Time: P2P



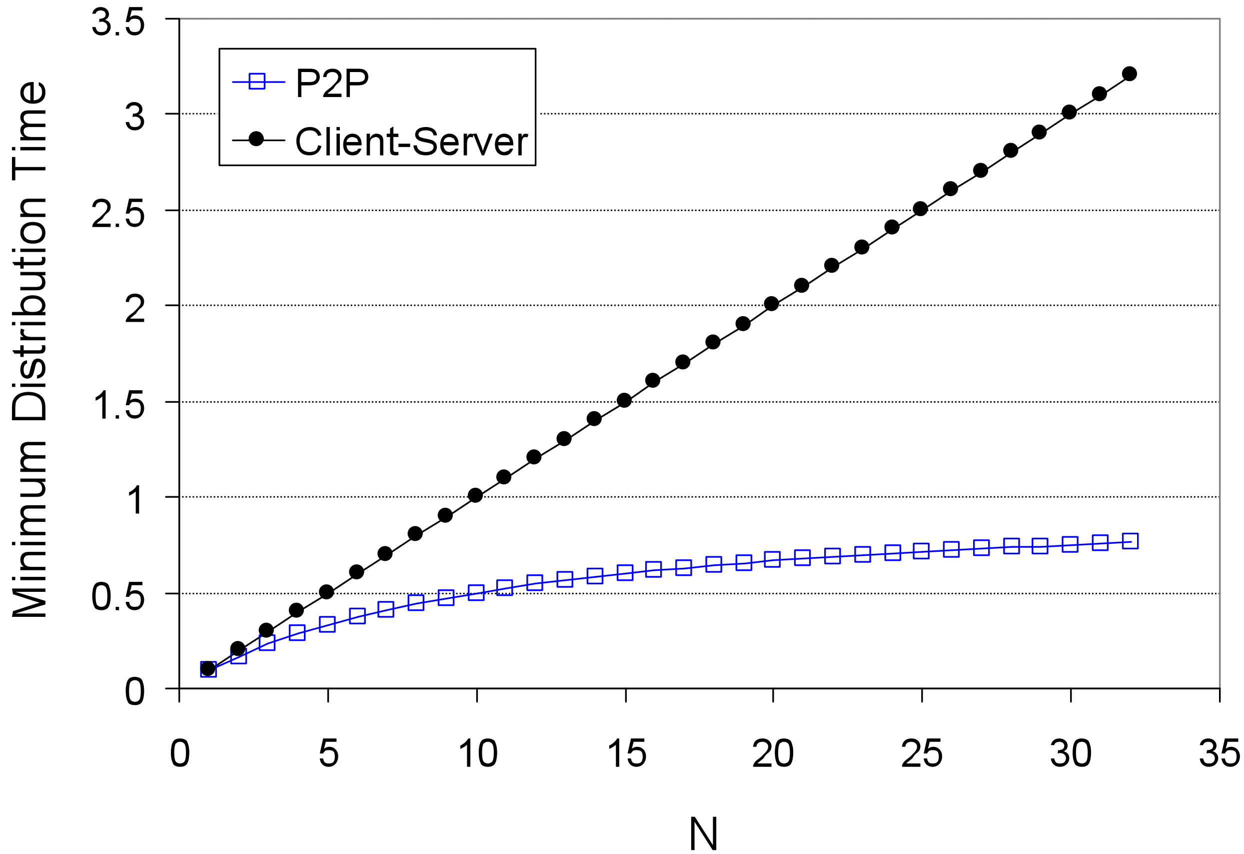
- Server must send one copy: F/u_s time
- Client i takes F/d_i time to download
- NF bits must be downloaded (aggregate)
  - Fastest possible upload rate: $u_s + \sum_{i=1}^{N} u_i$

$$d_{P2P} = \max\left\{ \frac{F}{u_s}, \frac{F}{\min\{d_i\}}, \frac{NF}{u_s + \sum_i u_i} \right\}$$

  - Only a lower-bound, but we can design transmission schedule approaching it

Client-Server vs. P2P: A Plot

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



Main Problems for P2P File Sharing System

1. Joining (bootstrapping problem)
2. Announcing what is shared
3. Searching for peers having a file
   - Including a sort of "string matching" problem
4. Downloading *efficiently* once the peers are found
5. Solving the *free-riding* problem
6. Avoiding single point of failure
   - And avoiding law suits by RIAA, MPAA, …
7. Handling the intermittent nature of the P2P network
8. And, perhaps, providing users' anonymity

Good Old Time: the Napster Design

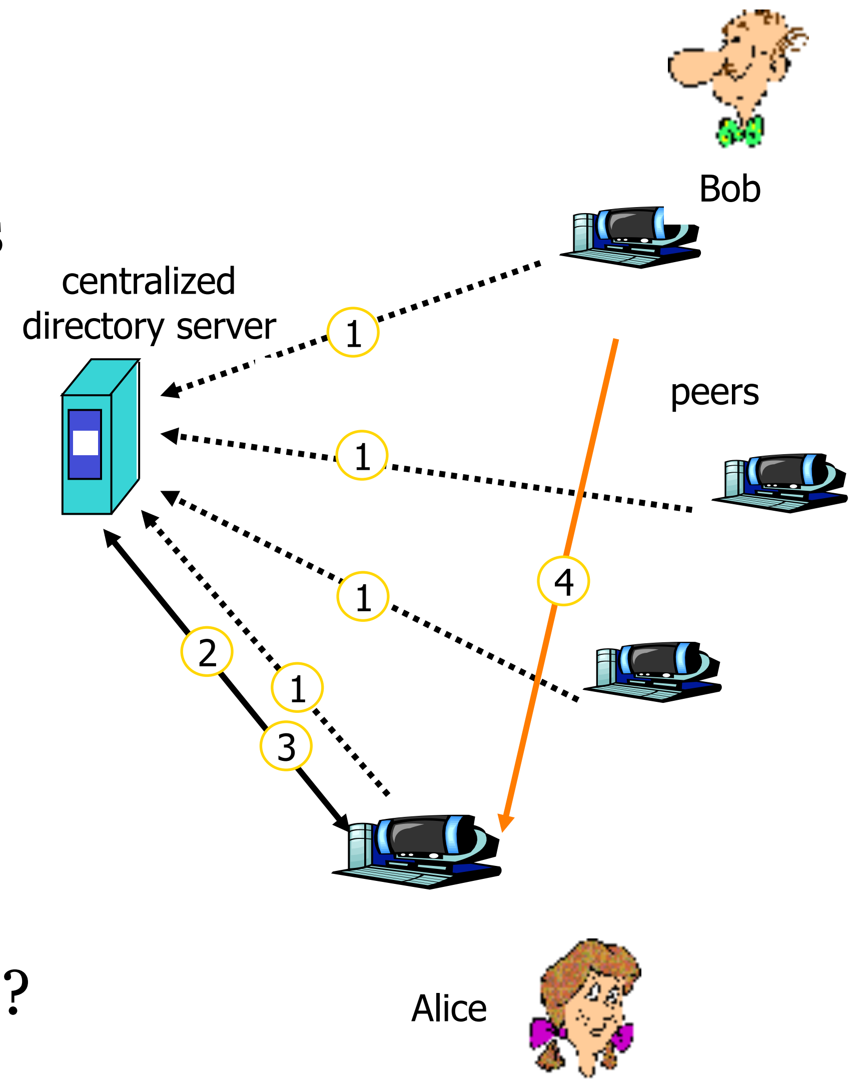
A *hybrid* between CS and P2P:

1) When peer connects, it informs central server:
- IP address
- Content it wants to share

2) Alice queries for “Hey Jude”

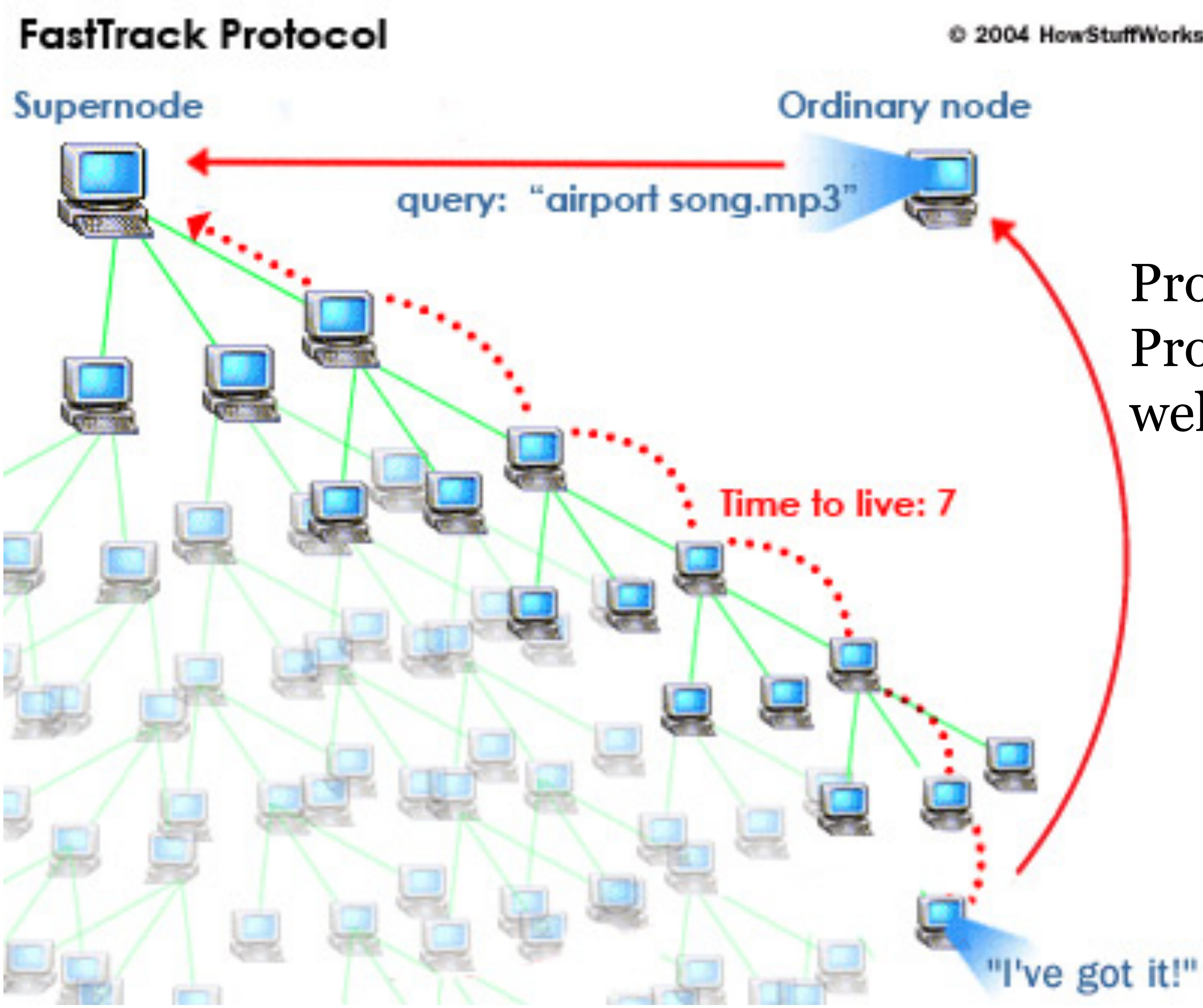
3) Server says “Bob has it”

4) Alice requests file from Bob



Which problems fundamentally cannot be solved by this design?

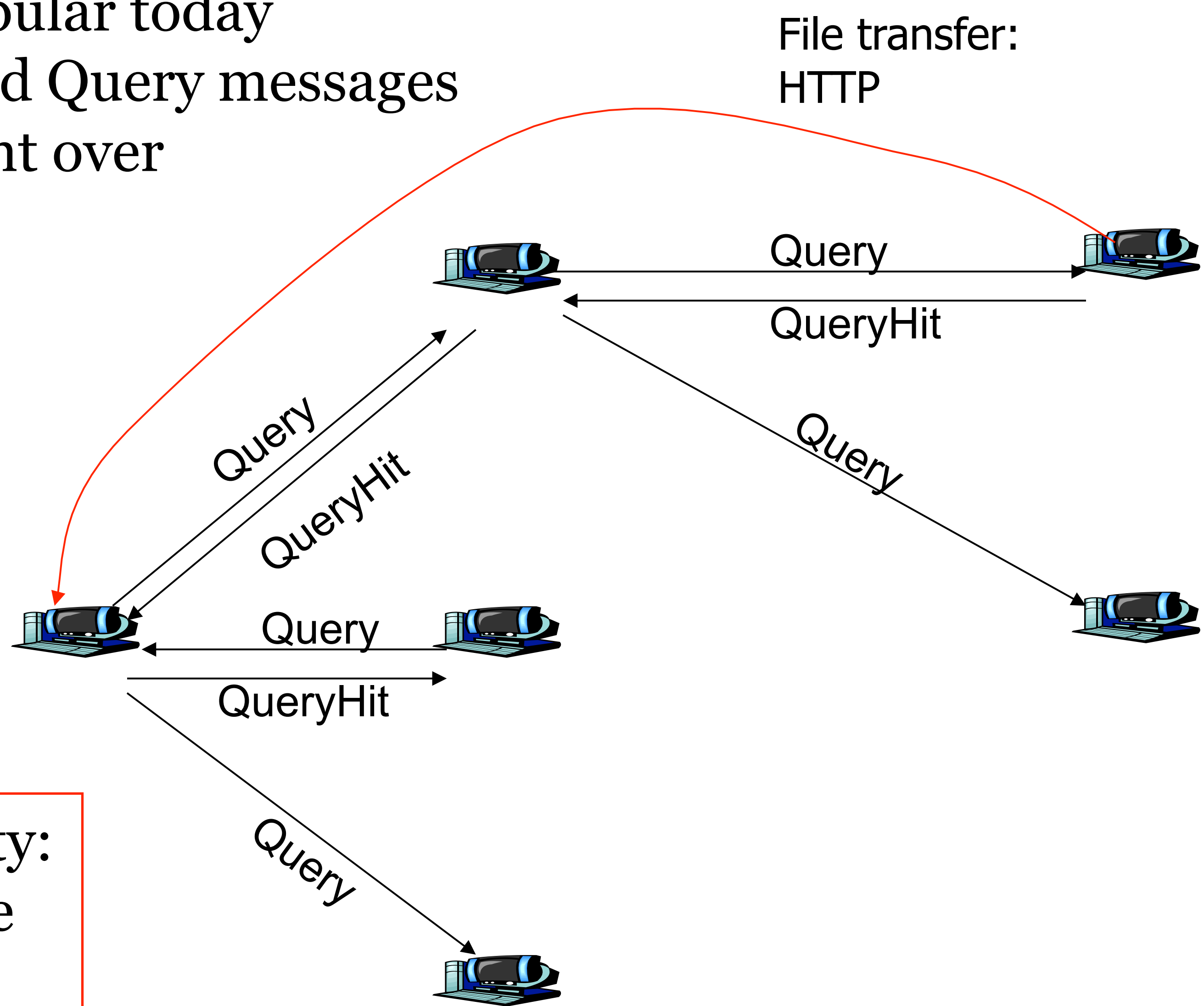
Second Generation P2P: KaZaa's FastTrack



Proprietary Protocol not well understood

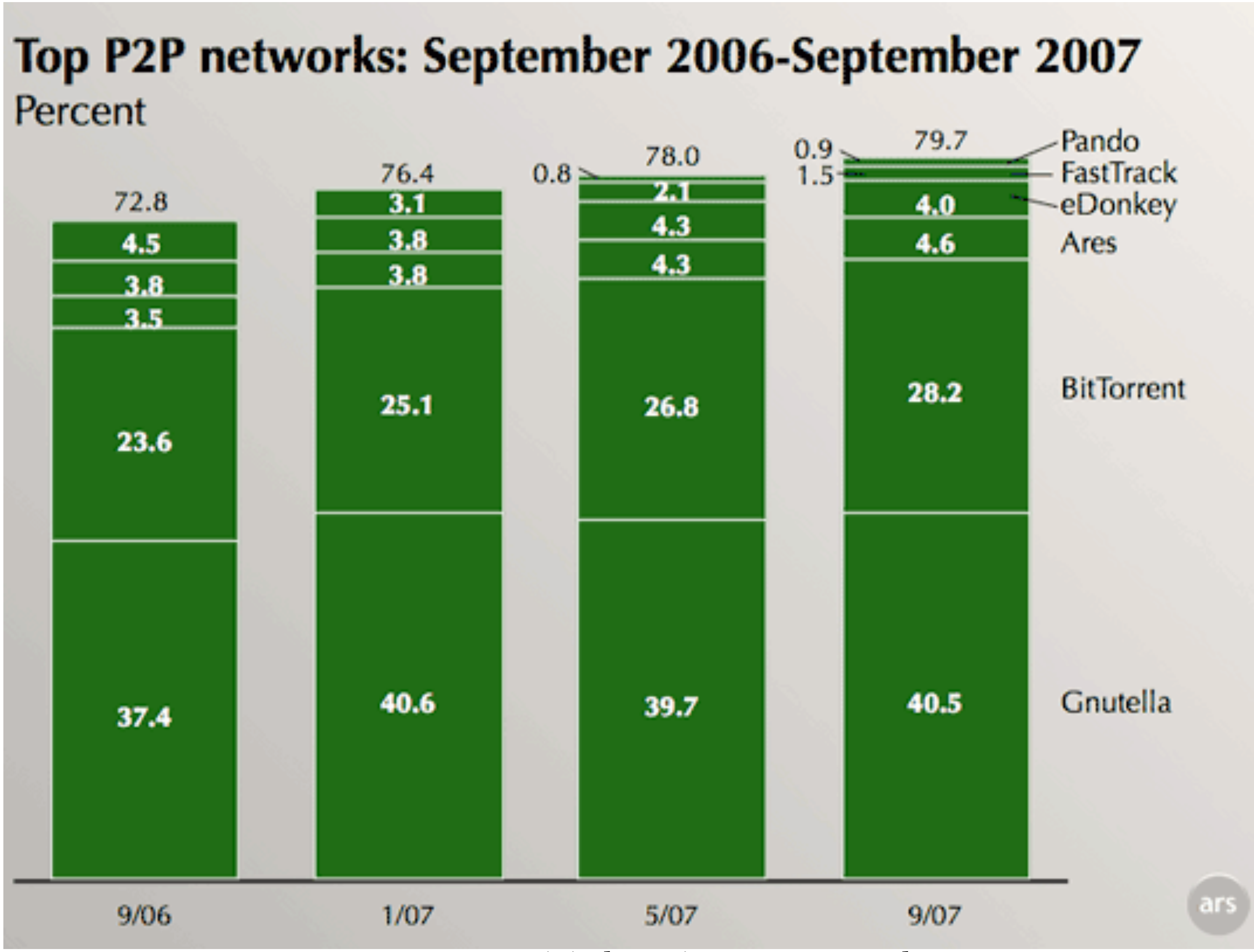
Second Generation P2P: Gnutella

- Still very popular today
- Peers forward Query messages
- QueryHit sent over reverse path



For scalability: limited scope flooding

P2P Market Share (2007)



Data source: Digital Music News Research Group

3rd/4th Generation P2P: BitTorrent

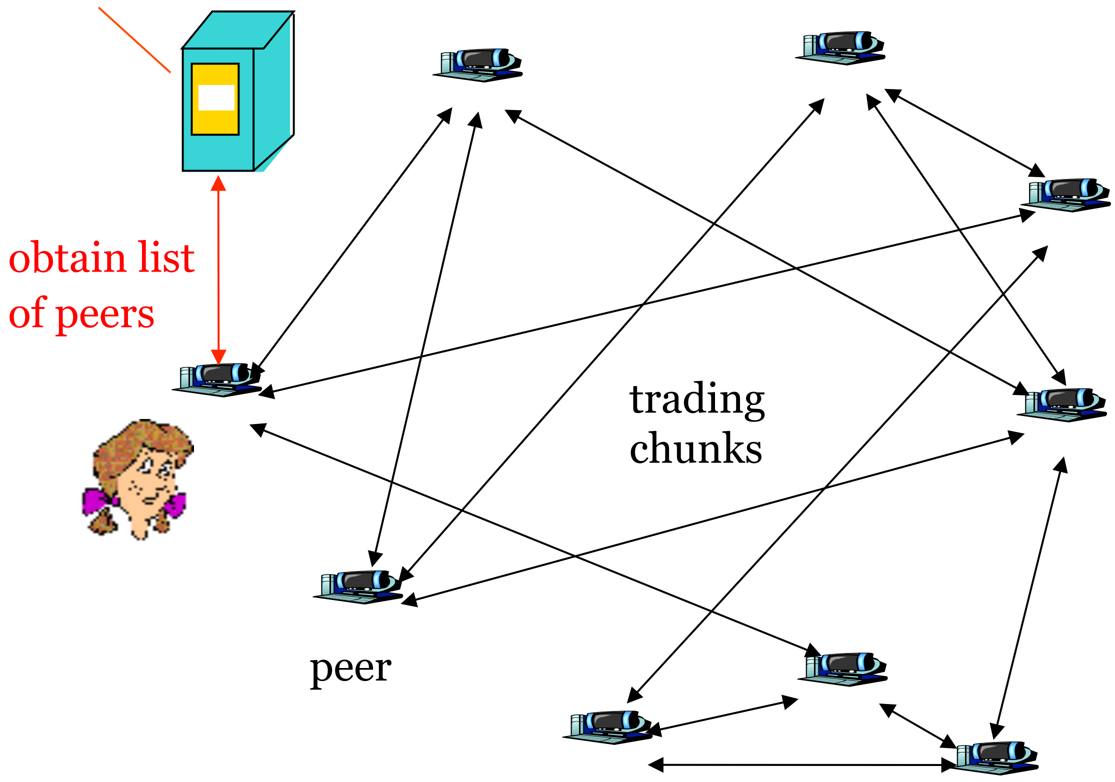
This protocol is fairly tedious, here's an outline

- Announcing what is shared:
  - Publish a *.torrent* file (meta info about the shared file)
  - Select a *tracker* (program keeping track of who share this particular file)
- Joining:
  - Download *.torrent* files, connect to corresponding tracker (s)
  - Connect to peers who are downloading the target file(s)
- Searching for peers having target file(s)
  - Google! (for the .torrent files)
  - Use directories, i.e. websites, like ThePirateBay.org

Tracker and Torrent/Swarm

Tracker: tracks peers participating in torrent

Torrent/Swarm: group of peers exchanging chunks of a file



BitTorrent: Efficiently Download from Peers

- File divided into equal-sized *chunks*
  - .torrent file list SHA1 checksums for chunks
- Periodically ask peers for chunks they have and announce what it has
- While downloading, peer uploads chunks to other peers
- Randomly request & download missing chunks
  - Or use *rarest first* strategy

BitTorrent: Dealing with Free-Riders

Tit-for-Tat

- Sends chunks to 4 peers currently giving **me** chunks *at highest rate*

- Periodically, randomly select another peer, starts sending chunks
  - newly chosen peer may join top 4
  - "optimistically unchoke"
  - Allow new peer (no chunk to share yet) to join the torrent

BitTorrent: Avoiding Single Point of Failure

- Trackers are single points of failure
  - Use distributed trackers (also called *tracker-less*)
  - But how to keep track of *file-name to peer-set mapping* in a distributed manner? Answer: **DHT**
  - Many current **Torrent* implementations are based on the *Kademlia* DHT protocol
  - *Chord* is another well-known DHT protocol which is a candidate for programming assignment 2
- How to search for *.torrent* files with partial string matchings?
  - Largely a (very good) research problem
  - *CuBit* (from Cornell): TechReport December 2008, plus a *pluggin* for BitTorrent

Plus Many Other File Sharing P2P Protocols

- Noticable ones
  - Freenet
  - GNUnet
  - eDonkey/eMule
  - …