Application Overlays (P2P); Network Transport Layer: Overview; UDP; Stop-and-Wait ARQ

Qiao Xiang, Congming Gao

https://sngroup.org.cn/courses/cnns-xmuf23/index.shtml

10/31/2023

This deck of slides are heavily based on CPSC 433/533 at Yale University, by courtesy of Dr. Y. Richard Yang.

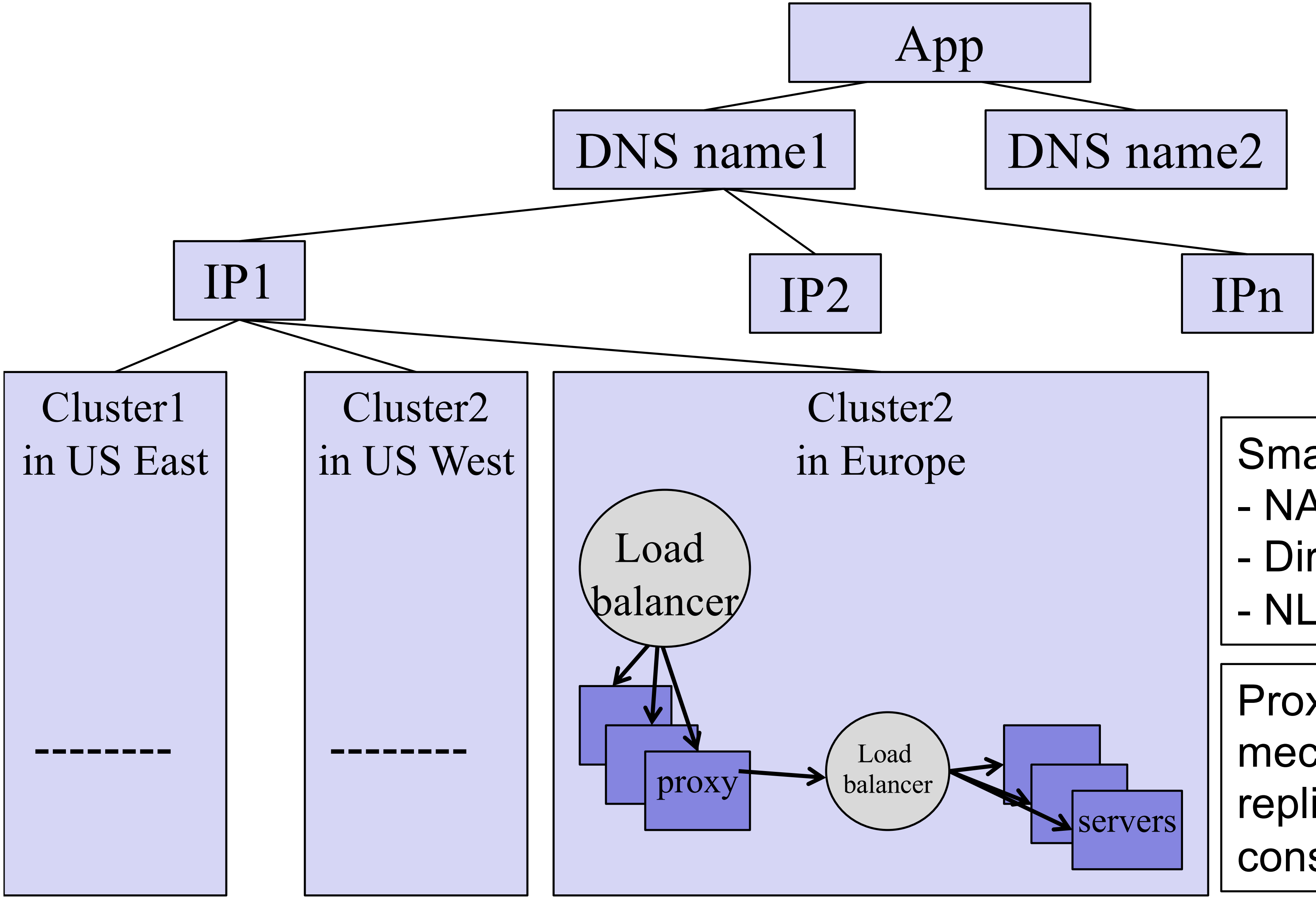
Outline

- Admin and recap
- Application overlays
- Overview of transport layer
- UDP
- Reliable data transfer, the stop-and-go protocols

Admin

- Lab assignment 3 due on Nov. 19
- Midterm exam on Nov. 9 (during lab class)
  - cover from introduction to application layer
  - 15-16 subjective questions over 100 minutes
  - 1-page cheat sheet allowed

Recap: Direction Mechanisms

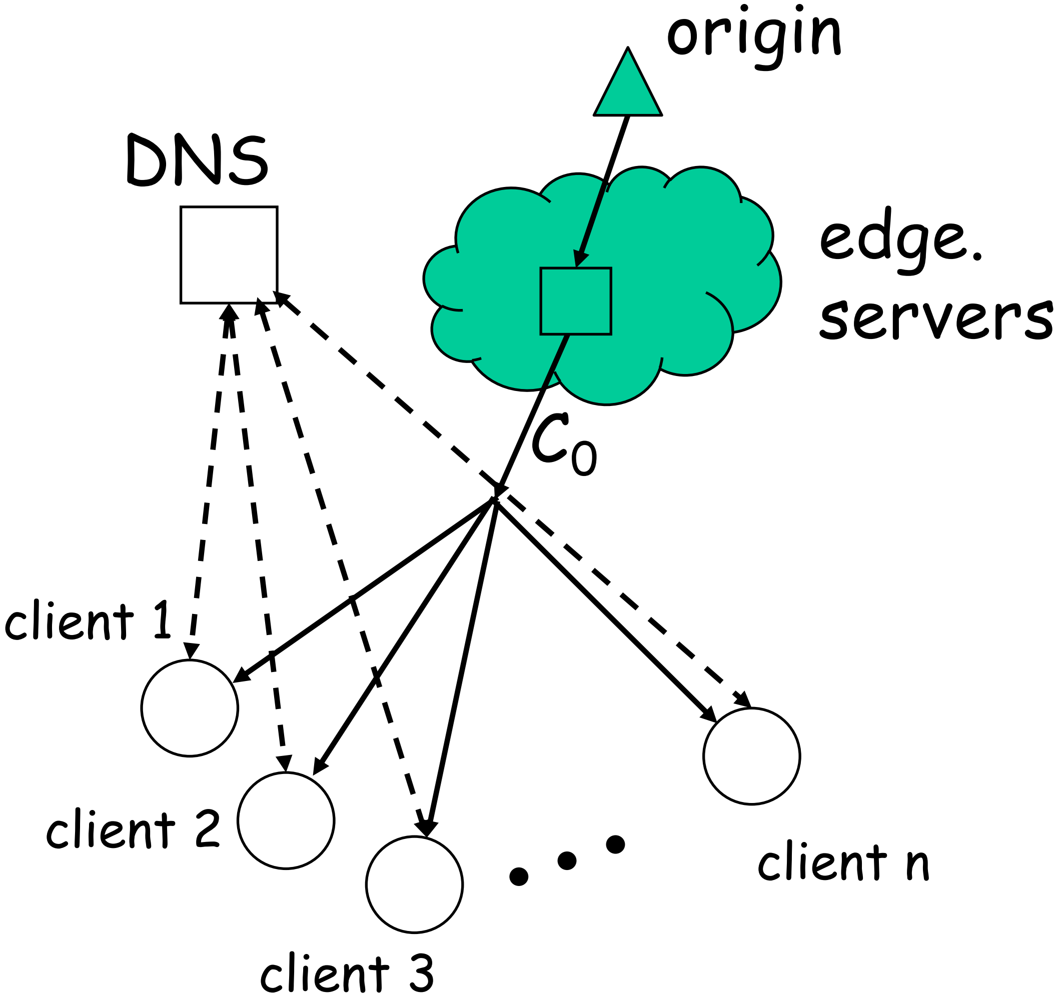


- cname
- hierarchy

Smart switch/LB
- NAT rewrite
- Direct reply
- NLB

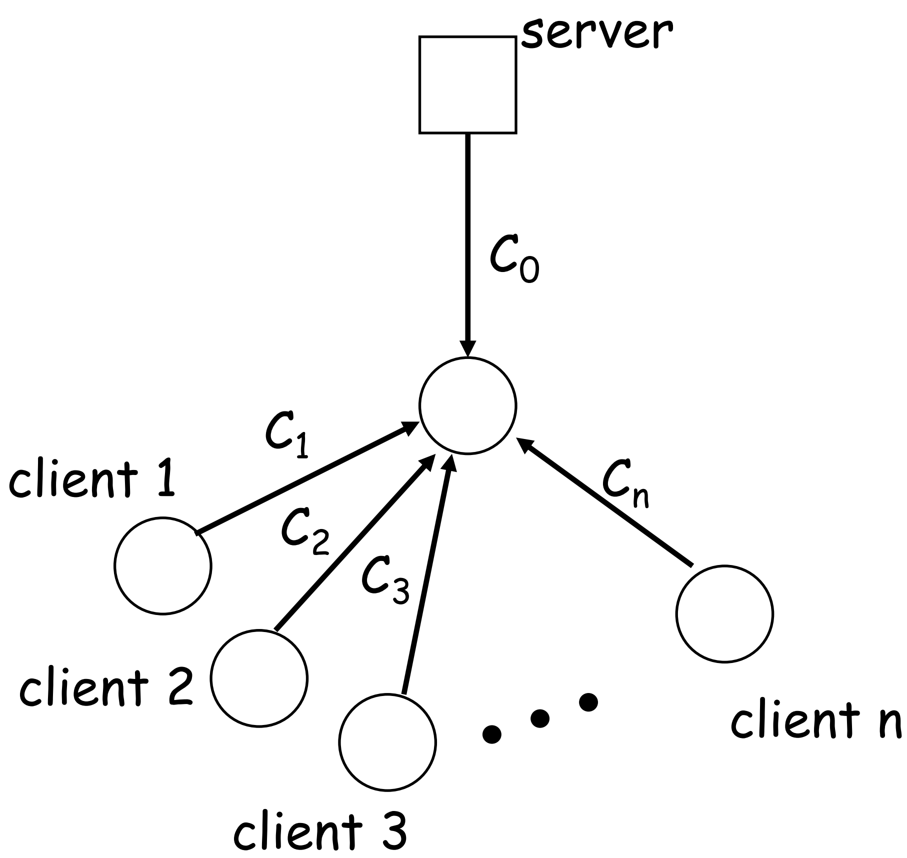
Proxy as a mechanism for replica content consistency

Scalability of Server-Only Approaches



An Upper Bound on Scalability

- Idea: use resources from both clients and the server
- Assume
  - need to achieve same rate to all clients
  - only uplinks can be bottlenecks
- What is an upper bound on scalability?

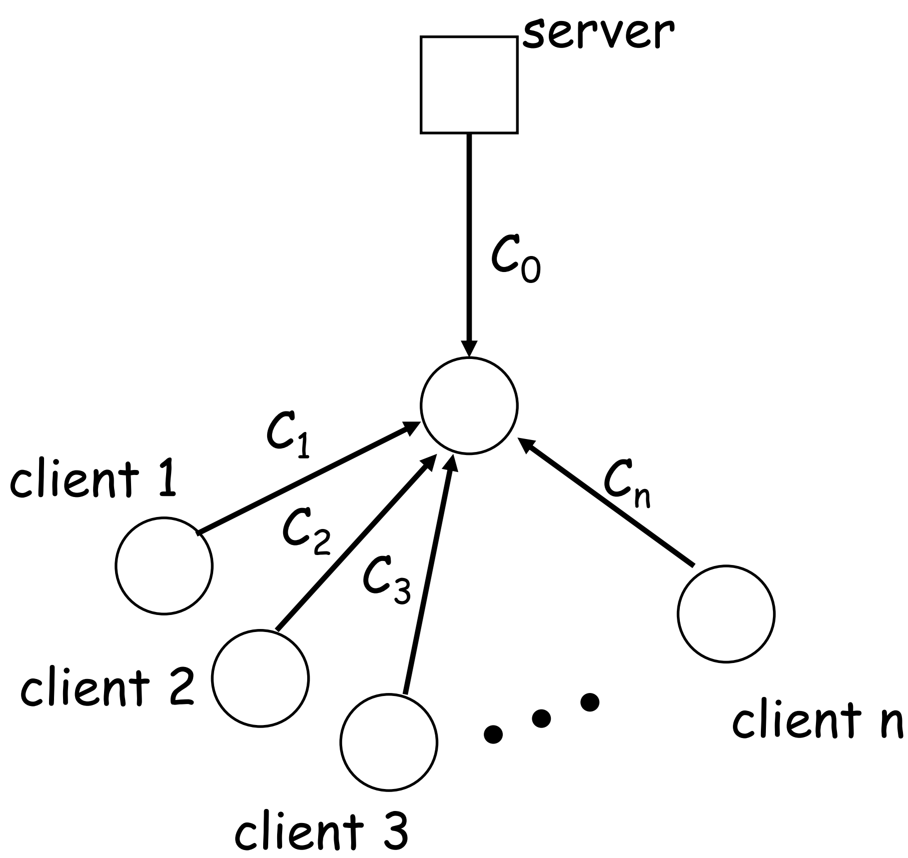


The Scalability Problem

- Maximum throughput

  $R = \min\{C_0, (C_0+\Sigma C_i)/n\}$

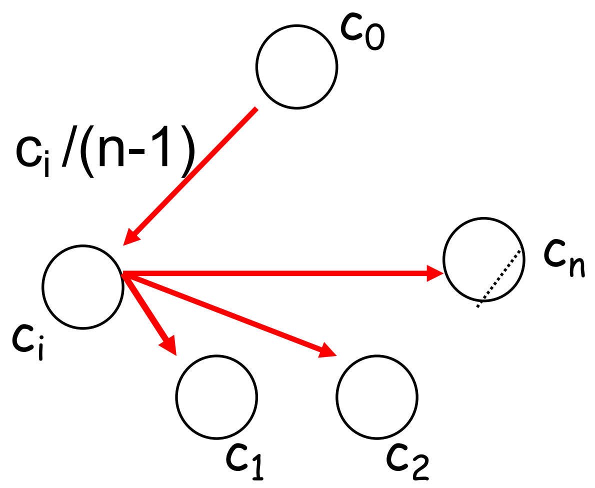
- The bound is theoretically approachable



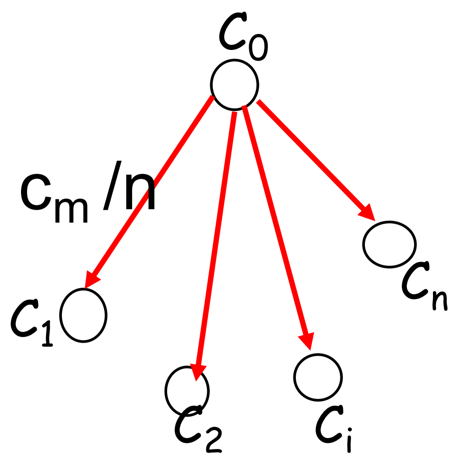
Theoretical Capacity: upload is bottleneck

$R = \min\{C_0, (C_0+\Sigma C_i)/n\}$

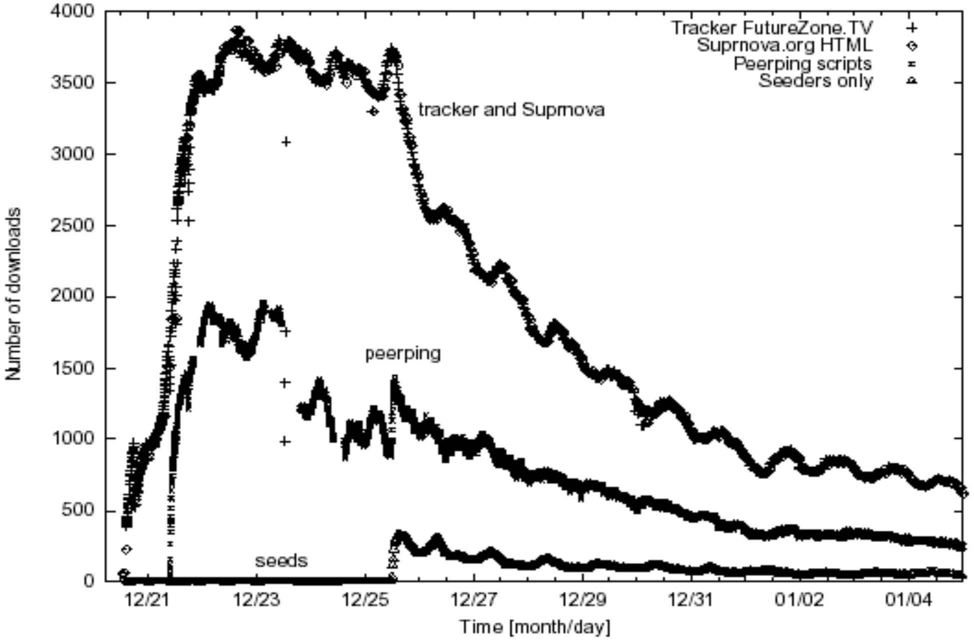
- Assume $c_0 > (C_0+\Sigma C_i)/n$
- Tree i:
  server → client i: $c_i/(n-1)$
  client i → other n-1 clients

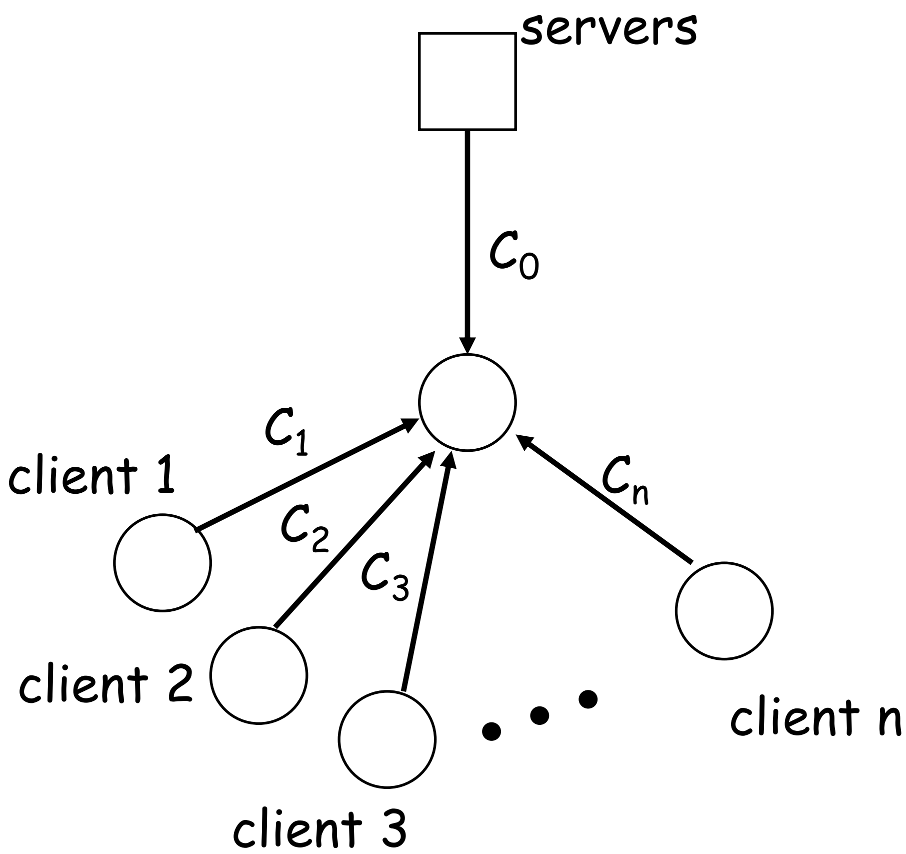


- Tree 0:
  server has remaining
  $c_m = c0 - (c1 + c2 + \ldots cn)/(n-1)$
  send to client i: c_m/n



Why not Building the Trees?

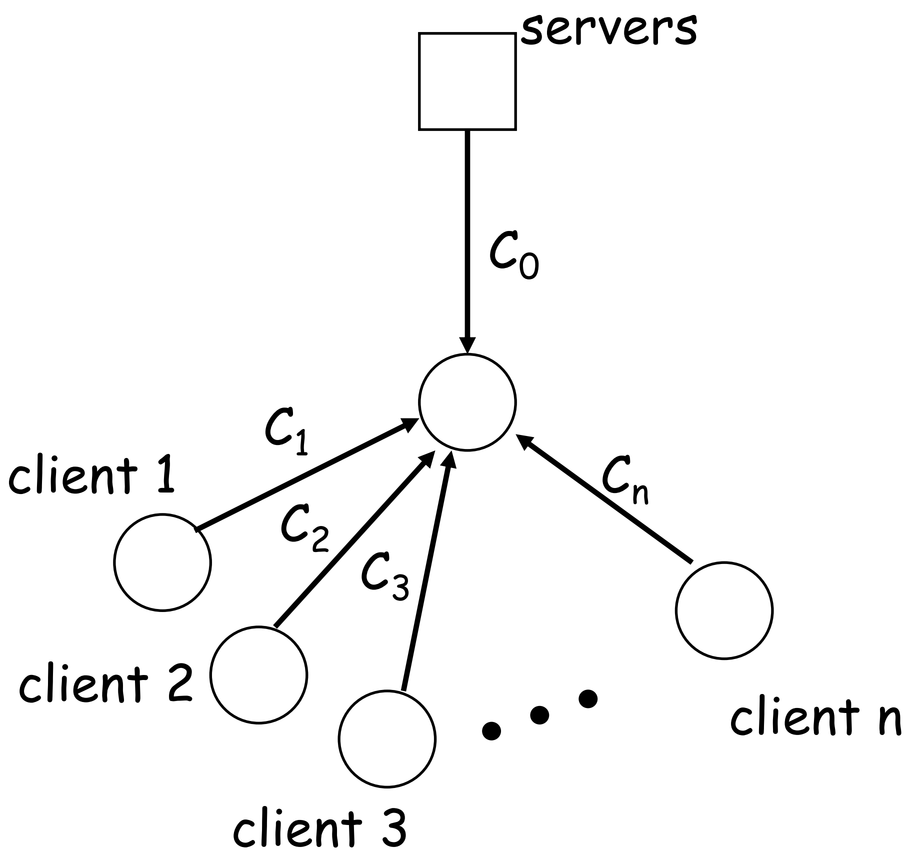




- Clients come and go (churns): maintaining the trees is too expensive
- Each client needs N connections

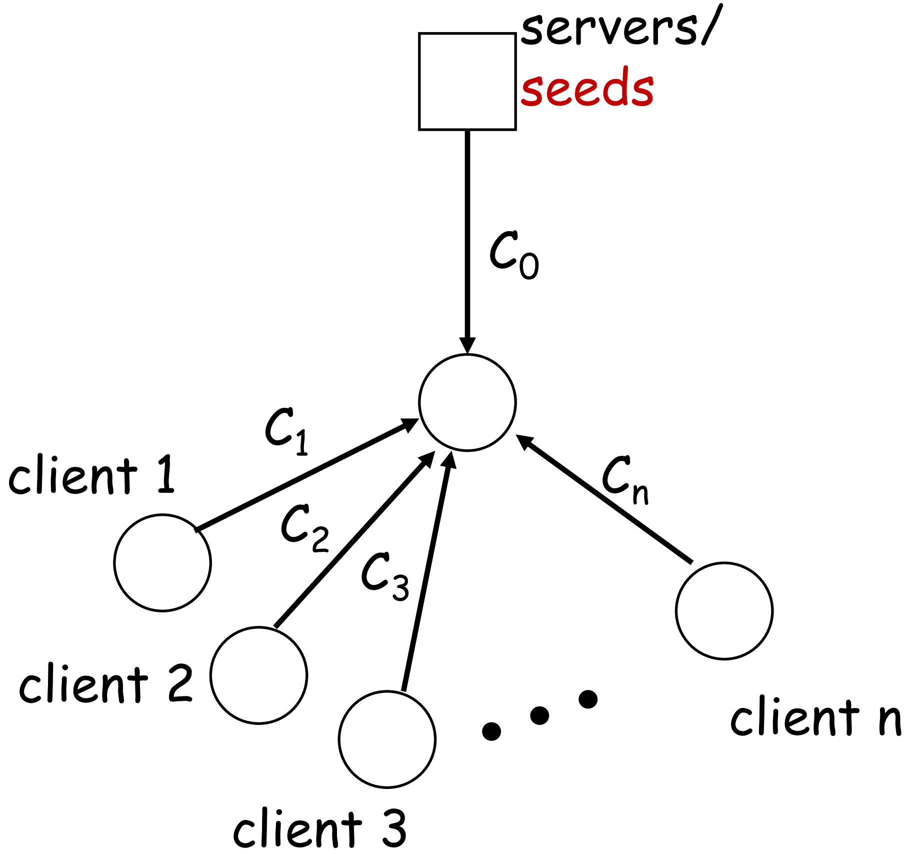
Server+Host (P2P) Content Distribution: Key Design Issues

- Robustness
  - Resistant to churns and failures
- Efficiency
  - A client has content that others need; otherwise, its upload capacity may not be utilized
- Incentive: clients are willing to upload
  - Some real systems nearly 50% of all responses are returned by the top 1% of sharing hosts



Discussion: How to handle the issues?

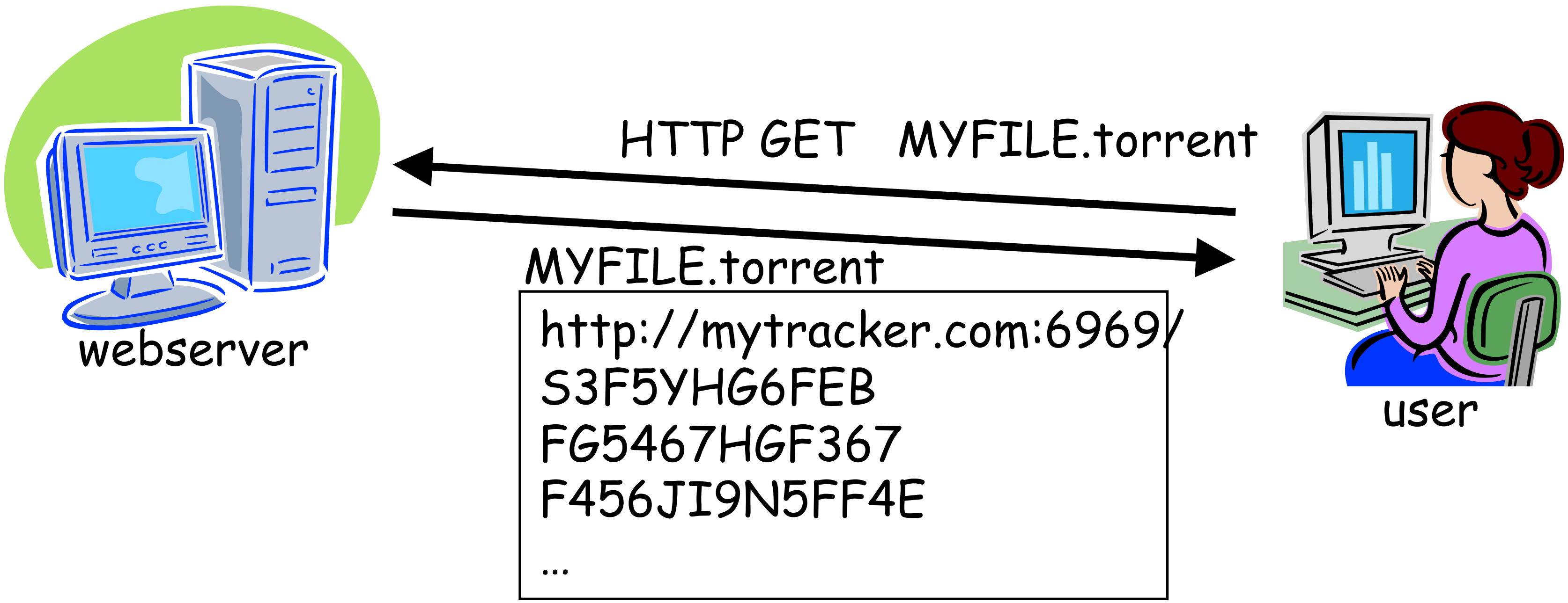
- Robustness
- Efficiency
- Incentive



Example: BitTorrent

- A P2P file sharing protocol
- Created by Bram Cohen in 2004
  - Spec at bep_0003: http://www.bittorrent.org/beps/bep_0003.html

BitTorrent: Lookup



Metadata (.torrent) File Structure

- Meta info contains information necessary to contact the tracker and describes the files in the torrent
  - URL of tracker
  - file name
  - file length
  - piece length (typically 256KB)
  - SHA-1 hashes of pieces for verification
  - also creation date, comment, creator, …

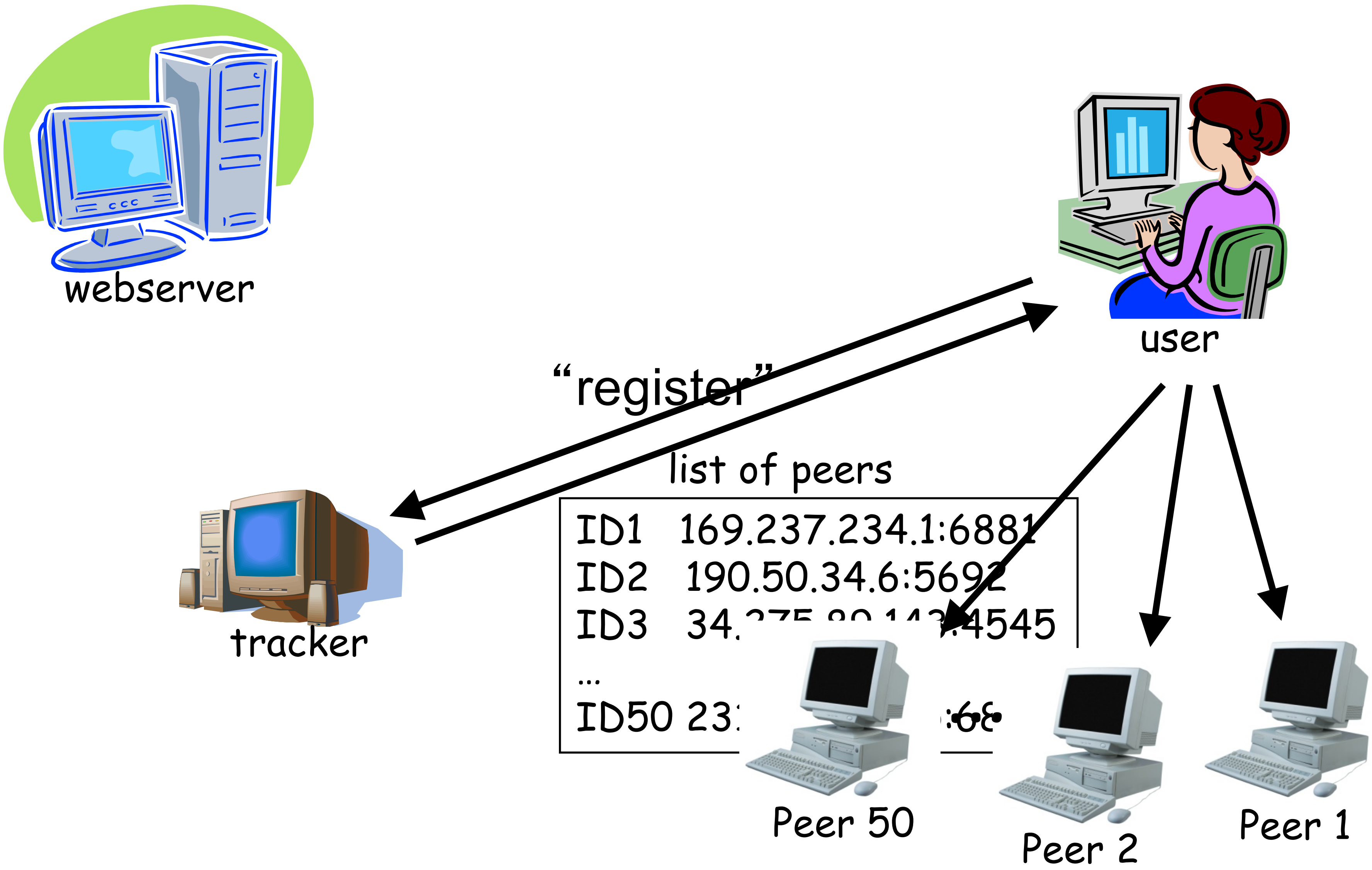
Tracker Protocol

- Communicates with clients via HTTP/HTTPS

- Client GET request
  - info_hash: uniquely identifies the file
  - peer_id: chosen by and uniquely identifies the client
  - client IP and port
  - numwant: how many peers to return (defaults to 50)
  - stats: e.g., bytes uploaded, downloaded

- Tracker GET response
  - interval: how often to contact the tracker
  - list of peers, containing peer id, IP and port
  - stats

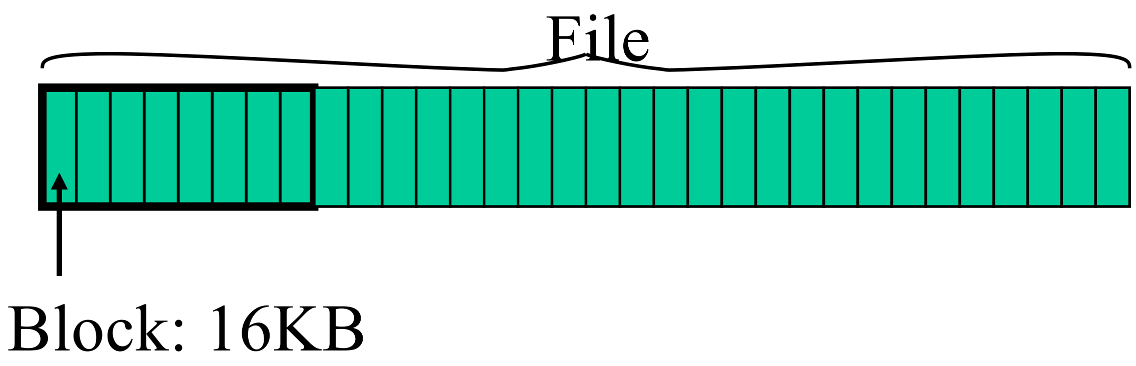
Tracker Protocol



Robustness and efficiency: Piece-based Swarming

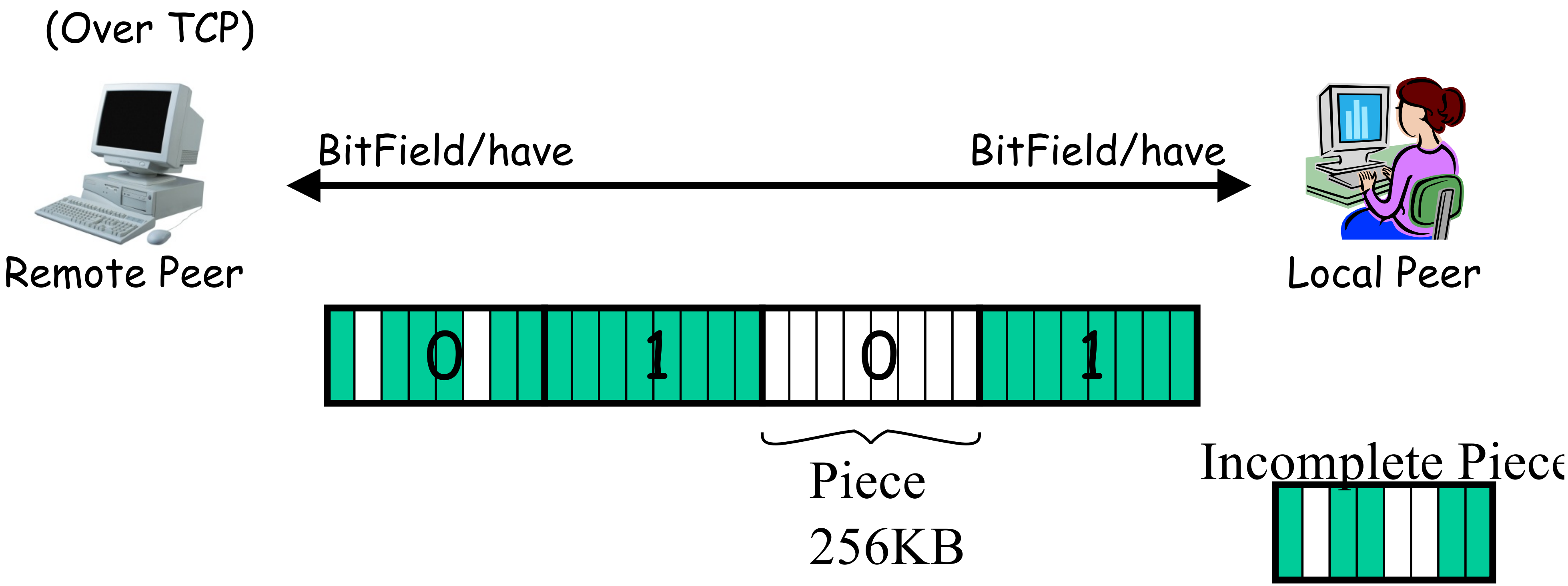
- Divide a large file into small blocks and request block-size content from different peers (why?)

Block: unit of download



- If do not finish downloading a block from one peer within timeout (say due to churns), switch to requesting the block from another peer

Detail: Peer Protocol

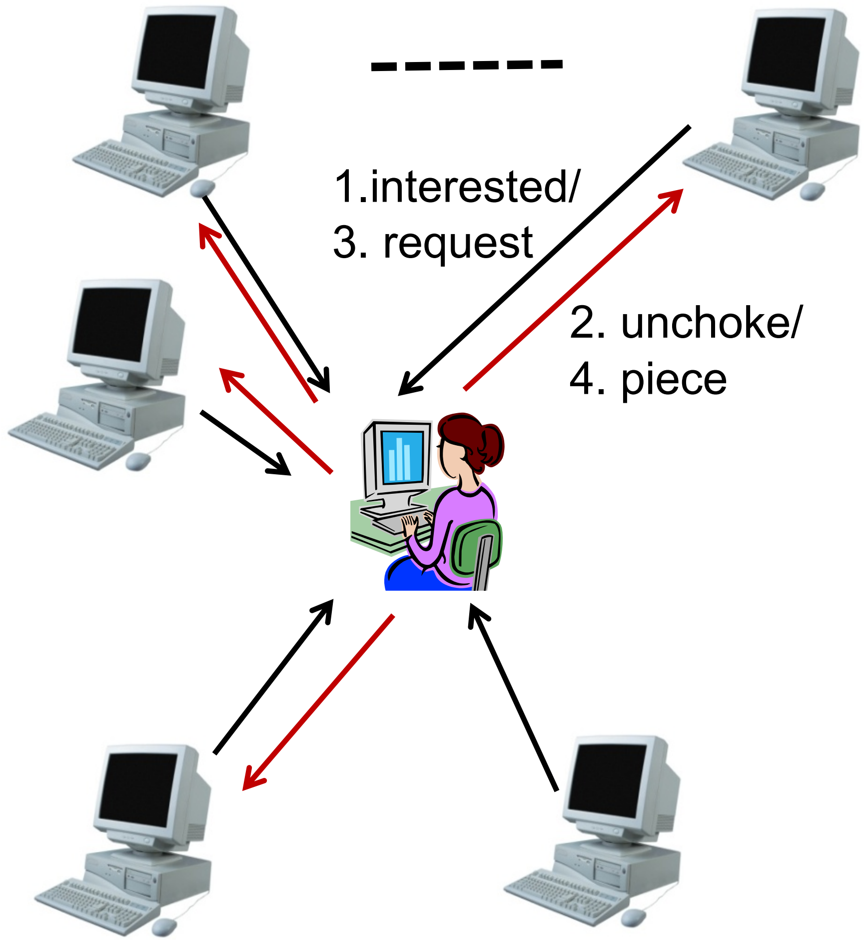


- Peers exchange bitmap representing content availability
  - `bitfield` msg during initial connection
  - `have` msg to notify updates to bitmap
  - to reduce bitmap size, aggregate multiple blocks as a piece

Peer Request

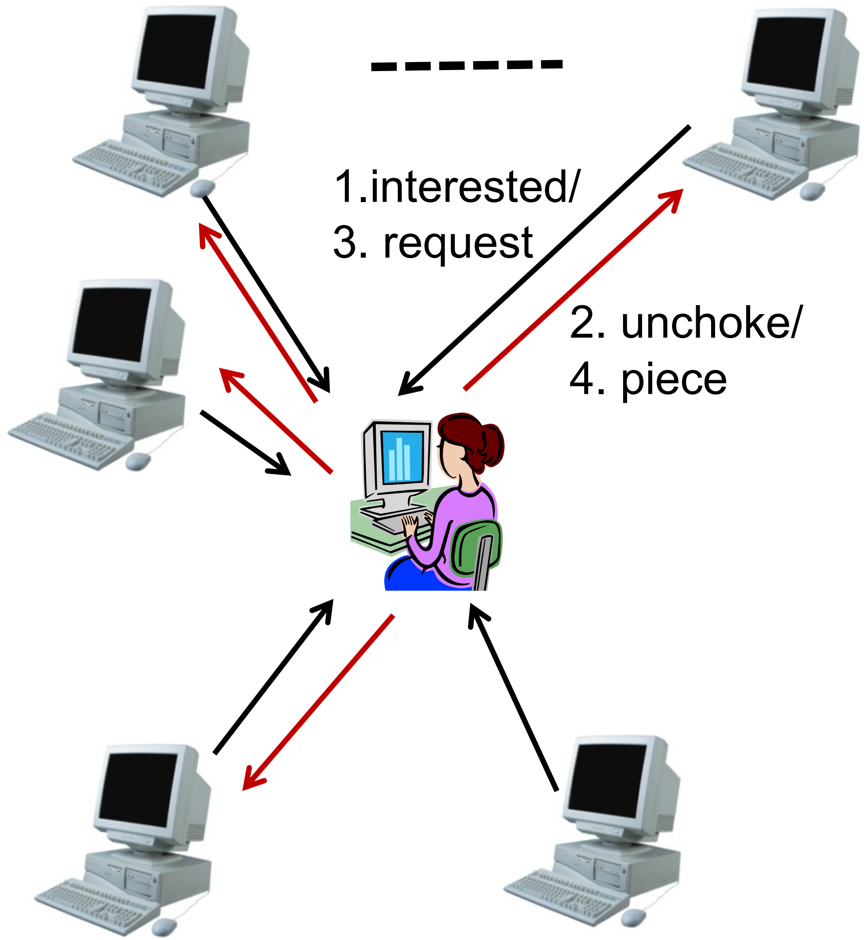
http://www.bittorrent.org/beps/bep_0003.html

- If peer A has a piece that peer B needs, peer B sends `interested` to A
- `unchoke`: indicate that A allows B to request
- `request`: B requests a specific block from A
- `piece`: specific data



Key Design Points

- `request:`
  - which data blocks to request?
- `unchoke:`
  - which peers to serve?



Request: Block Availability

- Request (local) rarest first
  - achieves the fastest replication of rare pieces
  - obtain something of value

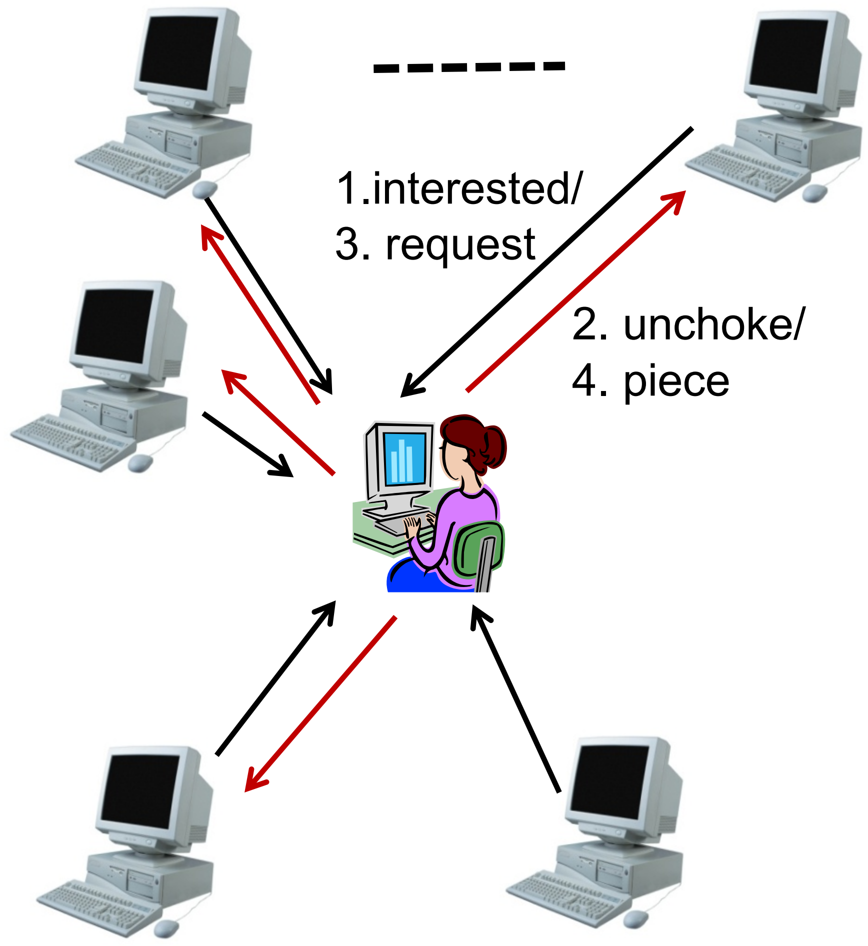
Block Availability: Revisions

- When downloading starts (first 4 pieces): choose at random and request them from the peers
  - get pieces as quickly as possible
  - obtain something to offer to others

- Endgame mode
  - defense against the "last-block problem": cannot finish because missing a few last pieces
  - send requests for missing pieces to all peers in our peer list
  - send `cancel` messages upon receipt of a piece

BitTorrent: Unchoke

- Periodically (typically every 10 seconds) calculate data-receiving rates from all peers
- Upload to (*unchoke*) the fastest
  - constant number (4) of unchoking slots
  - partition upload bw equally among unchoked



commonly referred to as "tit-for-tat" strategy

Optimistic Unchoking

- Periodically select a peer at random and upload to it
  - typically every 3 unchoking rounds (30 seconds)
- Multi-purpose mechanism
  - allow bootstrapping of new clients
  - continuously look for the fastest peers (exploitation vs exploration)

BitTorrent Fluid Analysis

- Normalize file size to 1
- $x(t)$: number of downloaders (also known as leechers) who do not have all pieces at time t.
- $y(t)$: number of seeds in the system at time t.
- λ: the arrival rate of new requests.
- μ: the uploading bandwidth of a given peer.
- c: the downloading bandwidth of a given peer, assume $c \geq \mu$.
- θ: the rate at which downloaders abort download.
- γ: the rate at which seeds leave the system.
- η: indicates the effectiveness of downloader sharing, η takes values in [0, 1].

System Evolution

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \lambda - \theta x(t) - \min\{cx(t), \mu(\eta x(t) + y(t))\},$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} = \min\{cx(t), \mu(\eta x(t) + y(t))\} - \gamma y(t),$$

Solving steady state: $\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \frac{\mathrm{d}y(t)}{\mathrm{d}t} = 0$

Define $\frac{1}{\beta} = \max\{\frac{1}{c}, \frac{1}{\eta}(\frac{1}{\mu} - \frac{1}{\gamma})\}$

$$\bar{x} = \frac{\lambda}{\beta(1 + \frac{\theta}{\beta})}$$

$$\bar{y} = \frac{\lambda}{\gamma(1 + \frac{\theta}{\beta})}.$$

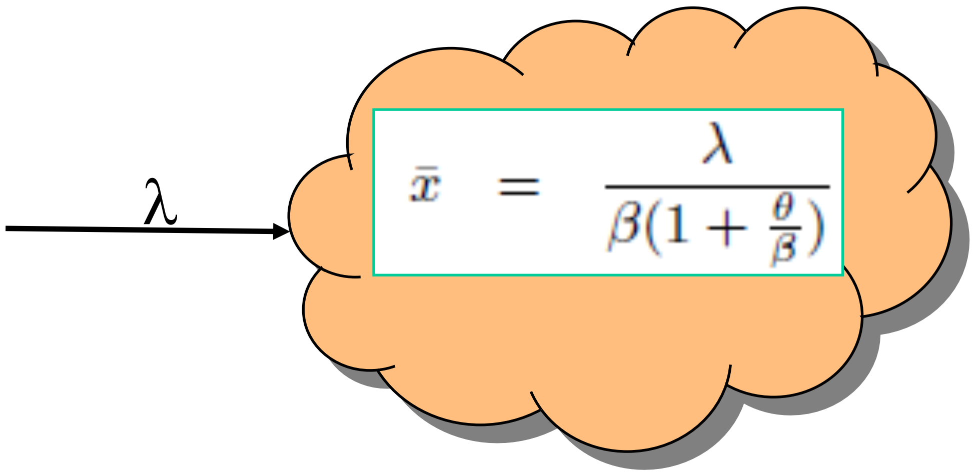
"Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks", SIGCOMM'04
https://conferences.sigcomm.org/sigcomm/2004/papers/p444-qiu1.pdf

System State

$$\bar{x} = \frac{\lambda}{\beta(1+\frac{\theta}{\beta})}$$

$$\bar{y} = \frac{\lambda}{\gamma(1+\frac{\theta}{\beta})}.$$

Q: How long does each downloader stay as a downloader?



$$\bar{x} = \frac{\lambda}{\beta(1+\frac{\theta}{\beta})}$$

$$T = \frac{1}{\theta + \beta}$$

$$\frac{1}{\beta} = \max\{\frac{1}{c}, \frac{1}{\eta}(\frac{1}{\mu} - \frac{1}{\gamma})\}$$

Key take-away: not scaling inverse with system size (x)

- New requests comes, new bandwidth also comes

Recap

- Applications
  - Client-server applications
    - Single server
    - Multiple servers load balancing
  - Application overlays (distributed network applications) to
    - scale bandwidth/resource (BitTorrent)
    - distribute content lookup (Freenet, DHT, Chord) [optional]
    - distribute content verification (Block chain) [optional]
    - achieve anonymity (Tor) [optional]