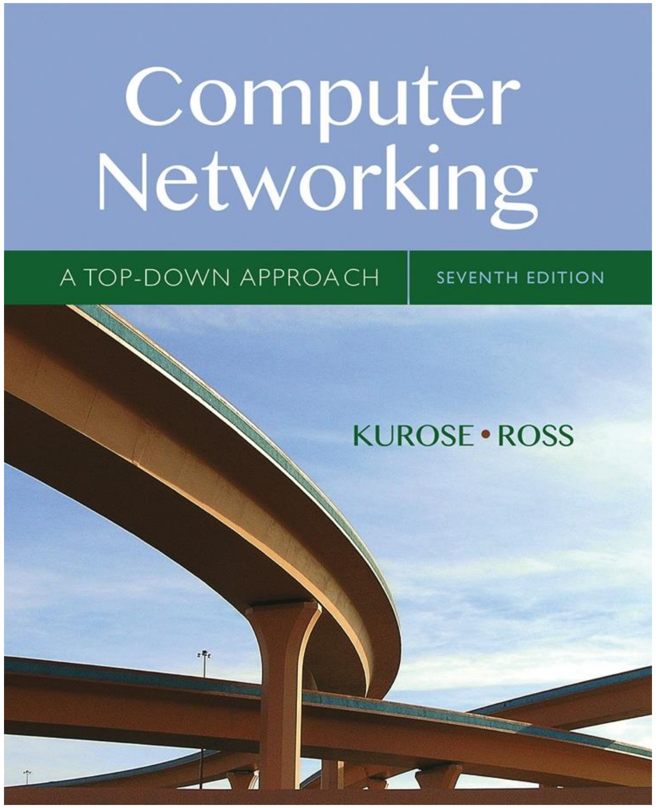
Chapter 2
Application Layer

A note on the use of these Powerpoint slides:

We’re making these slides freely available to all (faculty, students, readers). They’re in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we’d like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR



Computer Networking: A Top Down Approach

7th edition
Jim Kurose, Keith Ross
Pearson/Addison Wesley
April 2016

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail
- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

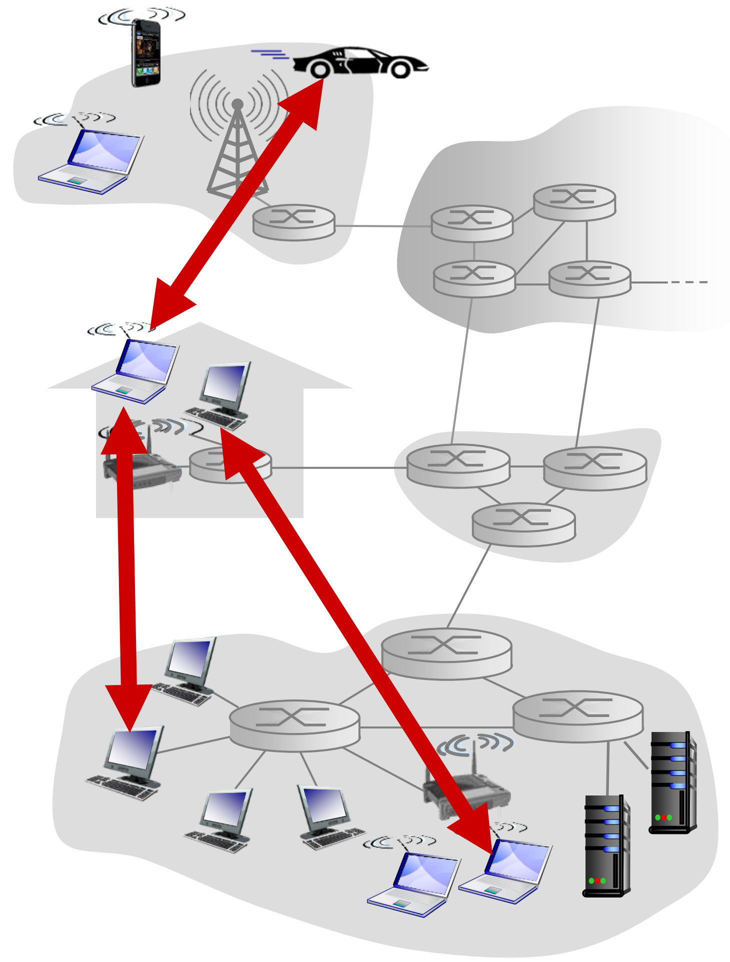
2.6 video streaming and content distribution networks

Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

examples:

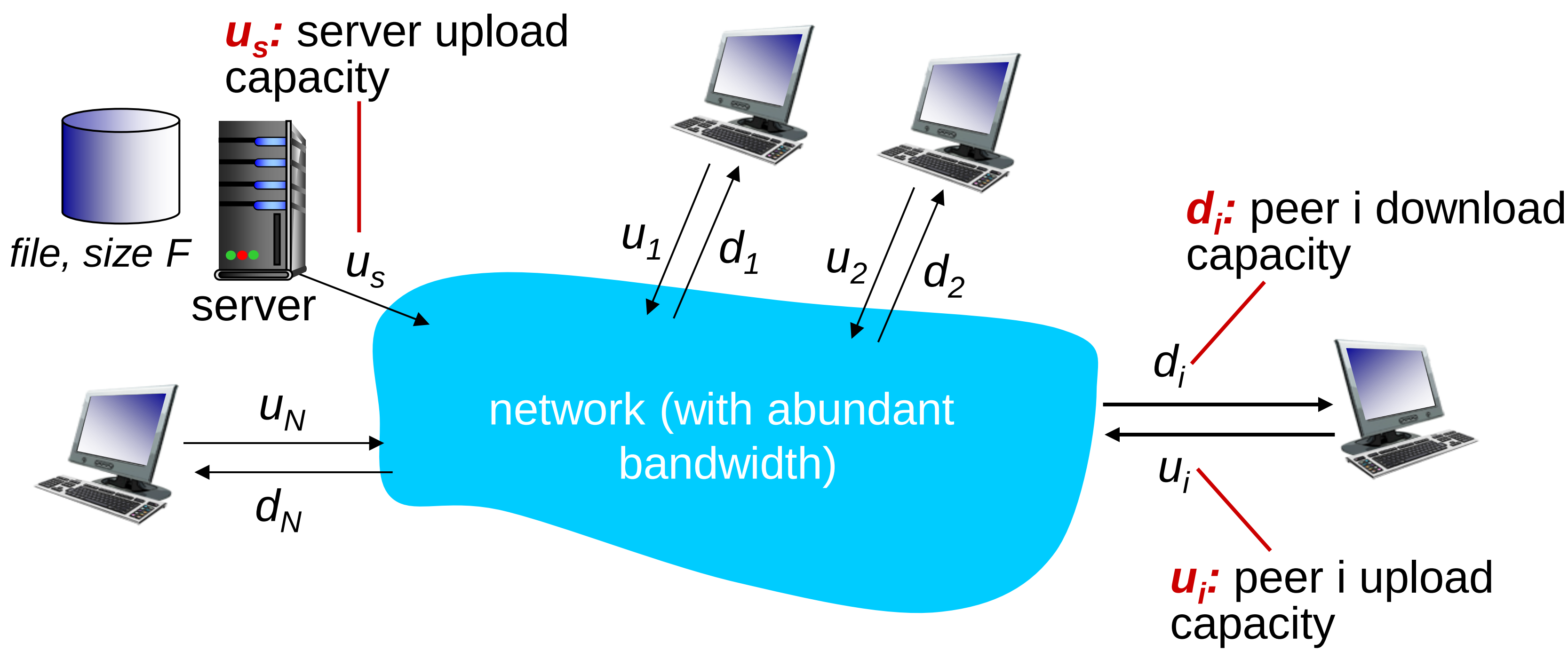
- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)



File distribution: client-server vs P2P

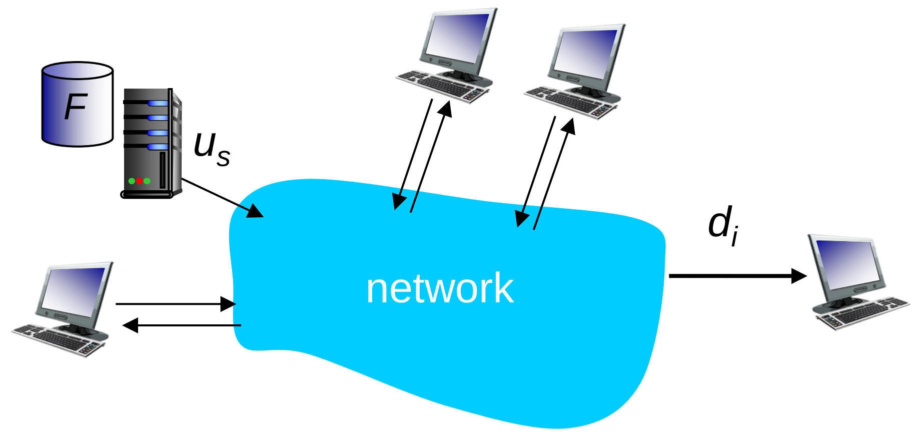
Question: how much time to distribute file (size *F*) from one server to *N* *peers*?

- peer upload/download capacity is limited resource



File distribution time: client-server

- *server transmission:* must sequentially send (upload) N file copies:
  - time to send one copy: F/u_s
  - time to send N copies: NF/u_s



- *client:* each client must download file copy
  - d_{min} = min client download rate
  - min client download time: F/d_{min}

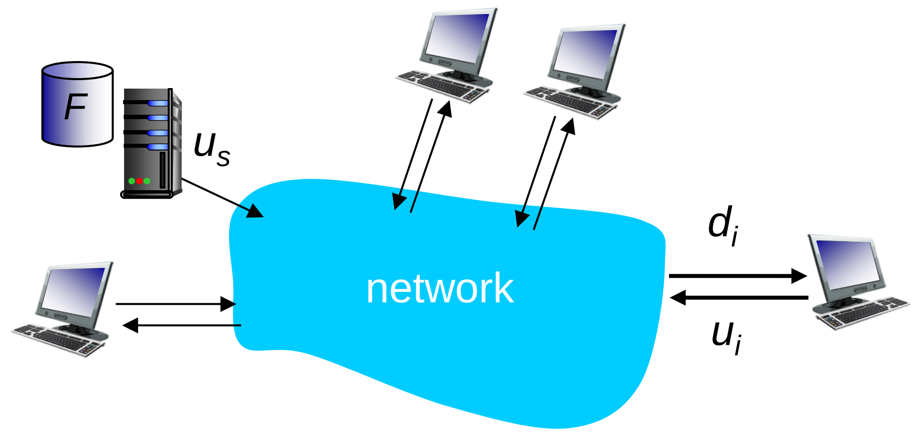
time to distribute F to N clients using client-server approach

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

increases linearly in N

File distribution time: P2P

- *server transmission:* must upload at least one copy
  - time to send one copy: F/u_s
- *client:* each client must download file copy
  - min client download time: F/d_{min}



- *clients:* as aggregate must download *NF* bits
  - fastest possible upload rate (assuming all nodes sending file chunks to same peer): $u_s + \sum_{i=1,N} u_i$

time to distribute F to N clients using P2P approach

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

increases linearly in *N* …

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

Client-server vs. P2P: example

All peers have the same upload rate, i.e., $u_1 = u_2 = \ldots u_N = u$

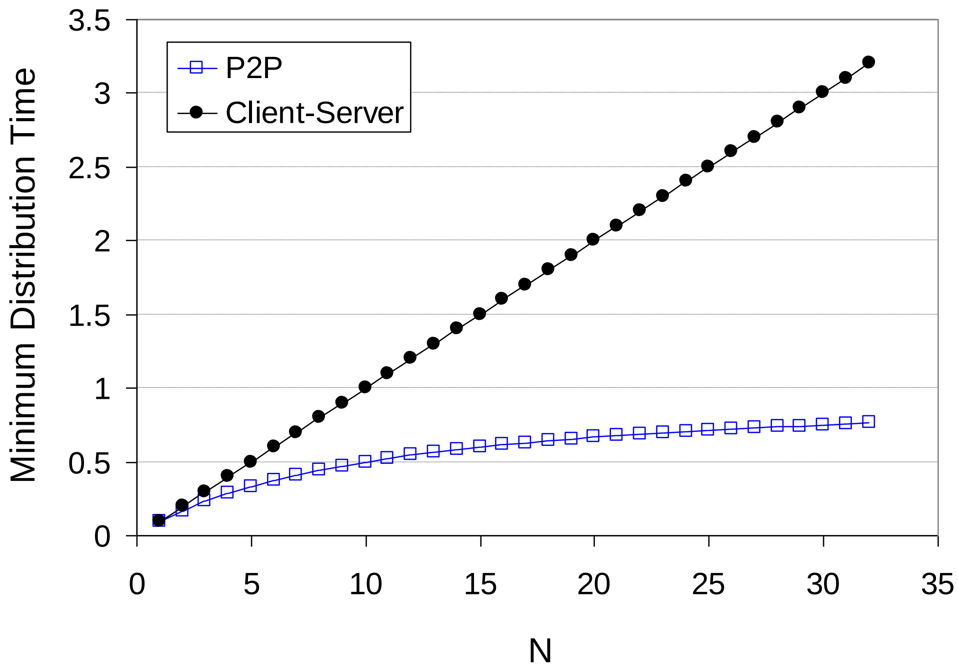
client upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$

For client-server

$D_{c\text{-}s} = NF/10u$

For P2P

$D_{P2P} = \max\{F/10u, NF/(N+10)u\}$

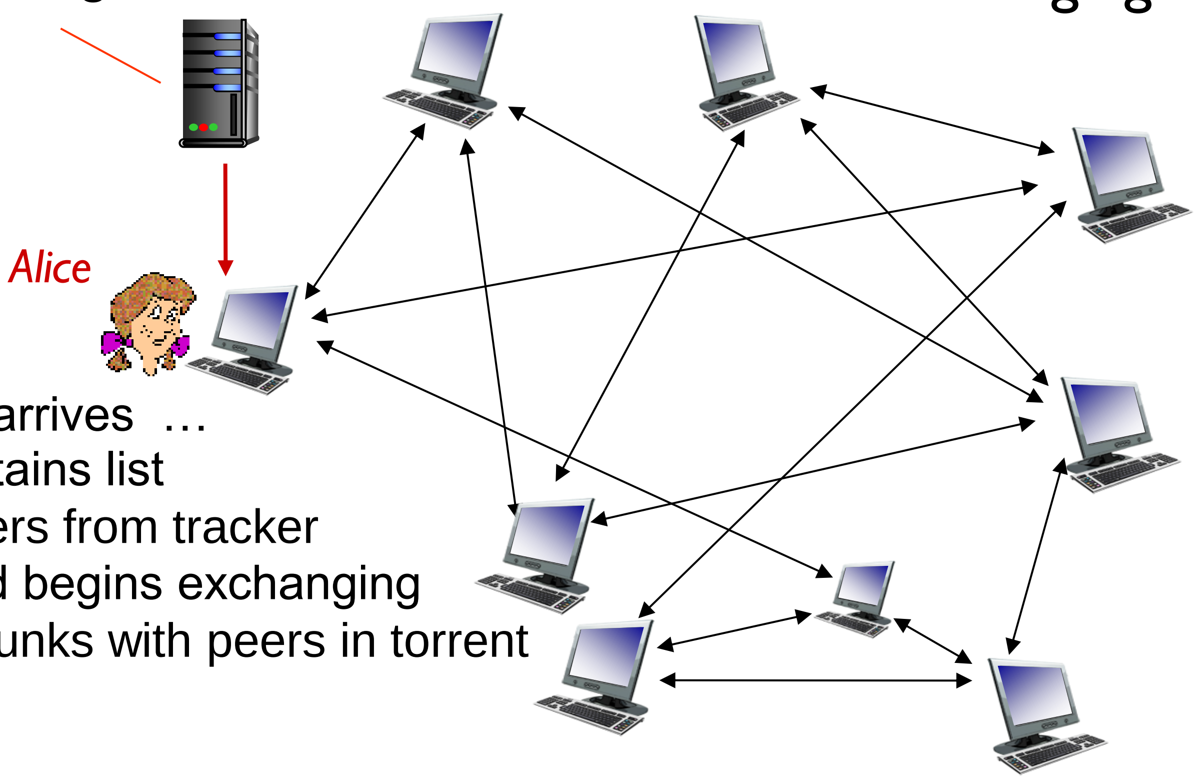


P2P file distribution: BitTorrent

- file divided into 256Kbytes chunks
- peers in torrent send/receive file chunks

tracker: tracks peers participating in torrent

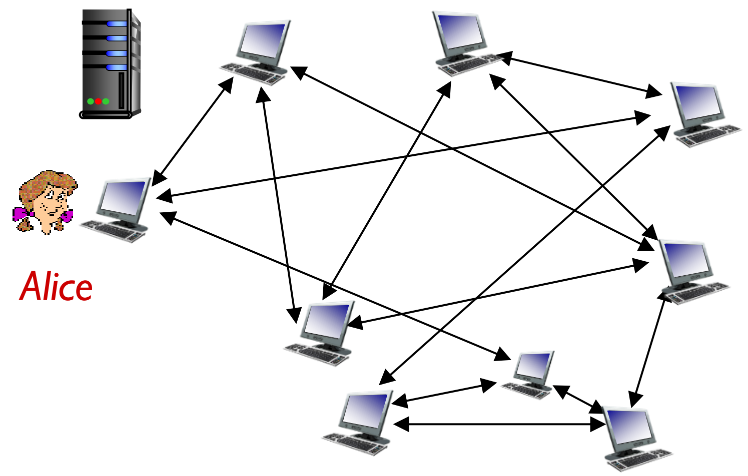
torrent: group of peers exchanging chunks of a file



Alice arrives …
… obtains list
of peers from tracker
… and begins exchanging
file chunks with peers in torrent

P2P file distribution: BitTorrent

- peer joining torrent:
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
  - has no chunks, but will accumulate them over time from other peers



- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn:* peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

BitTorrent: requesting, sending file chunks

requesting chunks: rarest first

- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests ***missing chunks*** from peers, the technique is called *rarest first*
  - Chunks having fewest repeating copies.

sending chunks: tit-for-tat

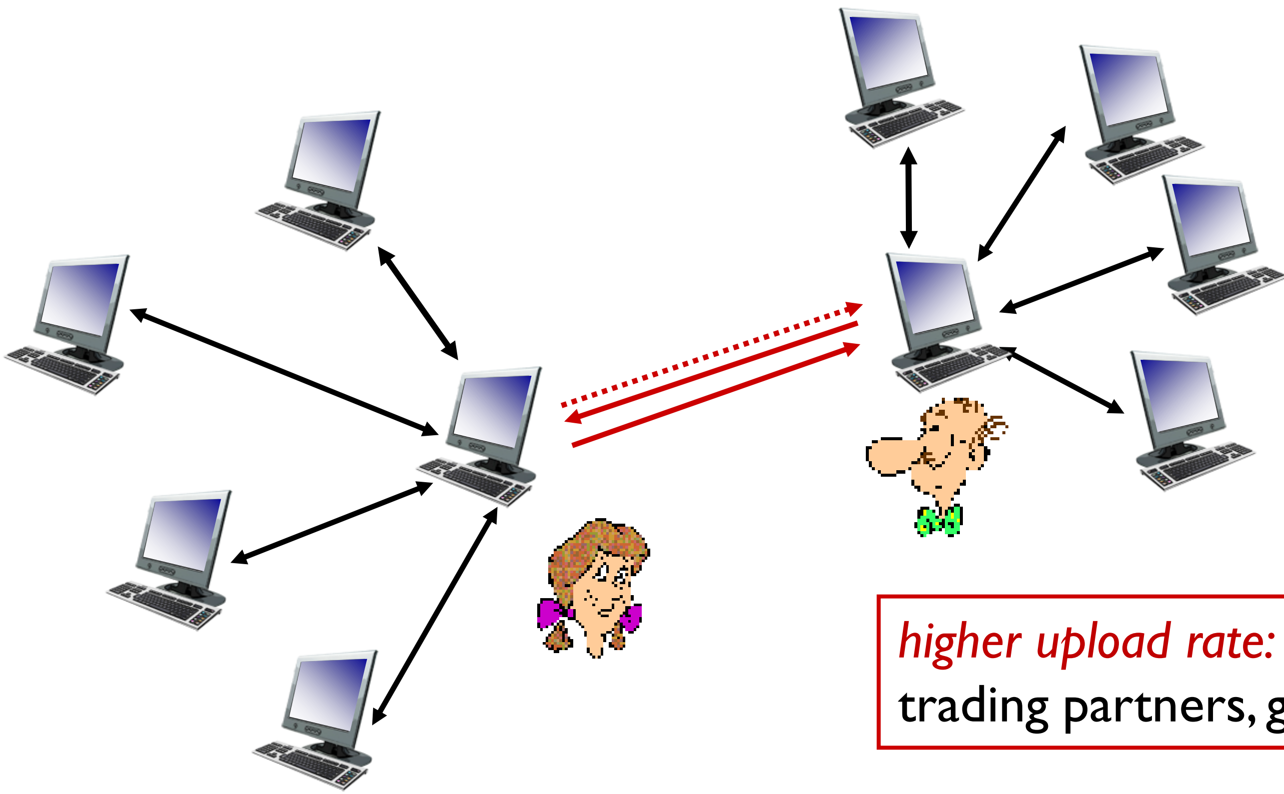
- Alice sends chunks to those four peers currently sending her chunks ***at highest rate***
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "*optimistically unchoked*" this peer
  - newly chosen peer may join top 4

BitTorrent: tit-for-tat

(1) Alice "optimistically unchokes" Bob

(2) Alice becomes one of Bob's top-four providers; Bob reciprocates

(3) Bob becomes one of Alice's top-four providers



higher upload rate: find better trading partners, get file faster !

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

Internet video:

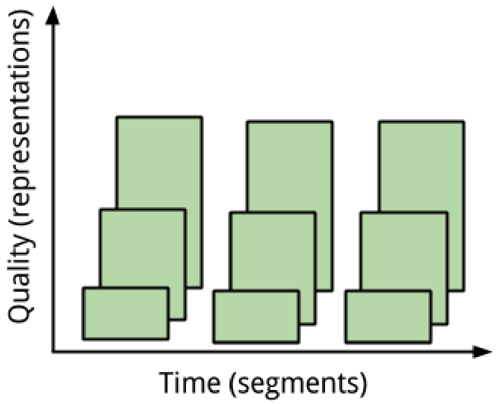
- Internet companies providing streaming videos are Netflix, Youtube(Google), Amazon, Youku
  - Netflix and Youtube, consumed *37%* and *16%* residential ISP traffic in 2015.
- A video is a sequence of images displayed at constant rate at 24 or 30 images per second.
- Videos can be compressed using compression algorithms.
- Trading off video quality with bit rate:
  - higher the bit rate→ the better the image quality → better overall user viewing experience → needs more space for storage
  - Ranges from 100kbps for low quality to 3Mbps for streaming HD movies

Streaming Multimedia using HTTP

- Using traditional HTTP streaming, a client requests a video from the server
  - Server sends video file in HTTP response message
  - Client collects bytes in application buffer and starts displaying them on screen
  - All clients receive the same encoding of the video regardless of the amount of bandwidth available to the clients.

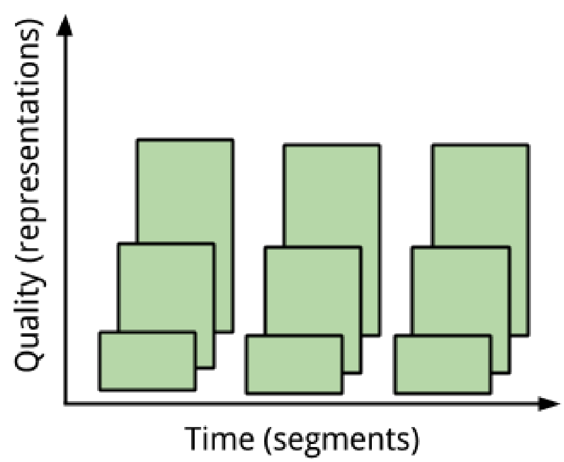
Streaming multimedia: DASH

- *DASH: D*ynamic, *A*daptive *S*treaming over *H*TTP
- *server:*
  - divides video file into multiple chunks
  - each chunk stored, encoded at different rates
  - *manifest file:* provides URLs for different chunks
- *client:*
  - periodically measures server-to-client bandwidth
  - consulting manifest, requests one chunk at a time
    - chooses maximum coding rate sustainable given current bandwidth
    - can choose different coding rates (and/or resolution) at different points in time (depending on available bandwidth at time)



Streaming multimedia: DASH

- *DASH: Dynamic, Adaptive Streaming over HTTP*
- *"intelligence"* at client: client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is "close" to client or has high available bandwidth)



Video Streaming and CDNs: context

- video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
  - ~1B YouTube users, ~75M Netflix users
- Straight forward approach: build a single massive data center, store all videos, stream videos to the clients directly
- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?
- *option 1:* single, large "mega-server"
  - single point of failure
  - point of network congestion
  - long path to distant clients
  - multiple copies of video sent over outgoing link

....quite simple but this solution *doesn't scale*

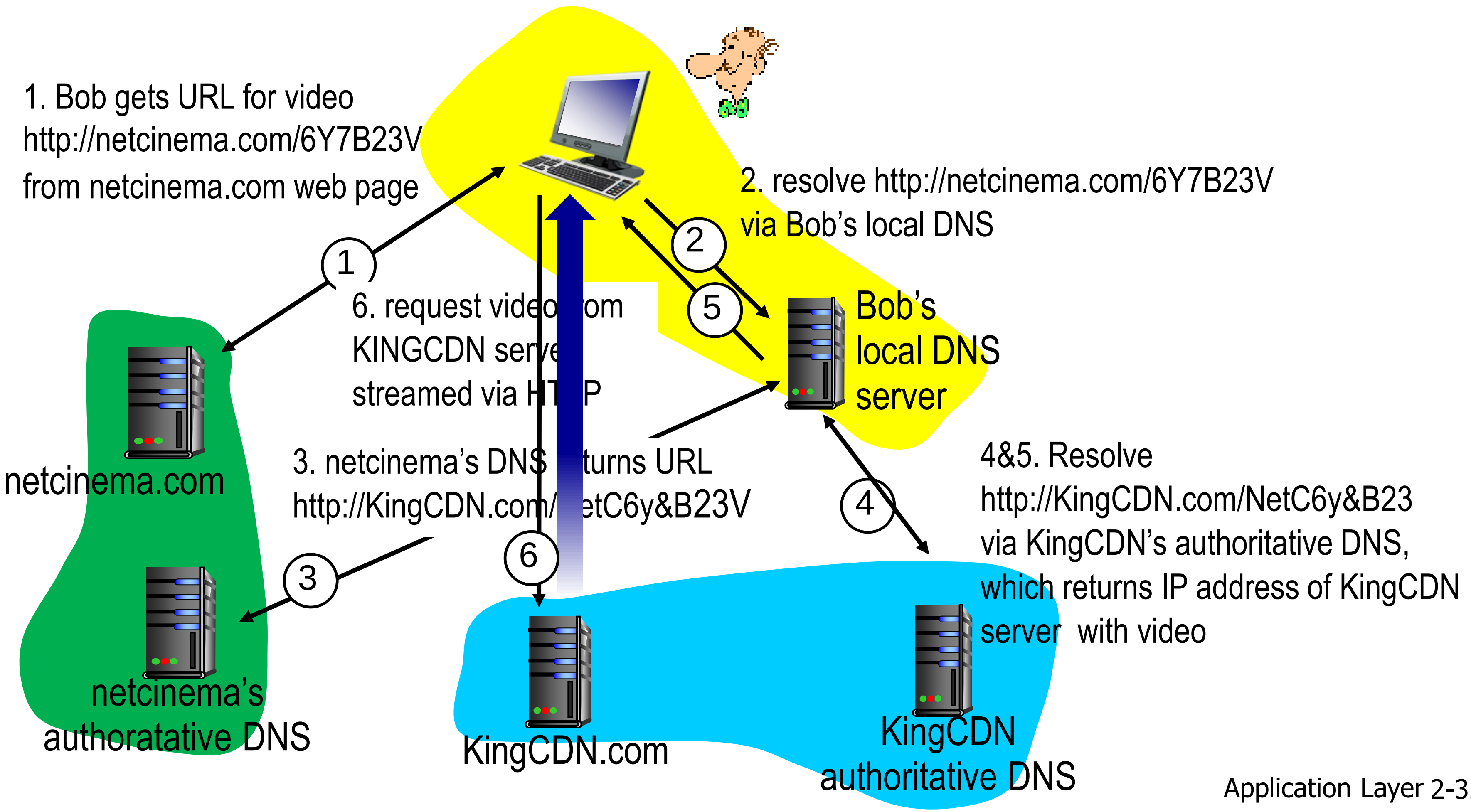
Content distribution networks

- *option 2:* store/serve multiple copies of videos at multiple geographically distributed sites *(CDN)*
  - *enter deep:* push CDN servers deep into many access networks
    - close to users, but the task of maintaining and managing the clusters becomes challenging
    - used by Akamai, 1700 locations
  - *bring home:* smaller number (10's) of larger clusters in POPs (Point of Presence) near (but not within) access networks
    - Lower maintenance and management overhead, but at the expense of higher delay and lower throughput to end users
    - used by Limelight

CDN content access: a closer look

Bob (client) requests video from provider Netcinema http://netcinema.com/6Y7B23V

- video stored in 3rd party CDN company (kingCDN) at http://KingCDN.com/NetC6y&B23V
- CDN uses intercepting and redirecting the request.



Cluster Selection Strategy

- *challenge:* how does CDN DNS select "good" CDN node to stream to client
  - pick CDN node geographically closest to client
  - pick CDN node with shortest delay (or min # hops) to client (CDN nodes periodically ping access ISPs, reporting results to CDN DNS)

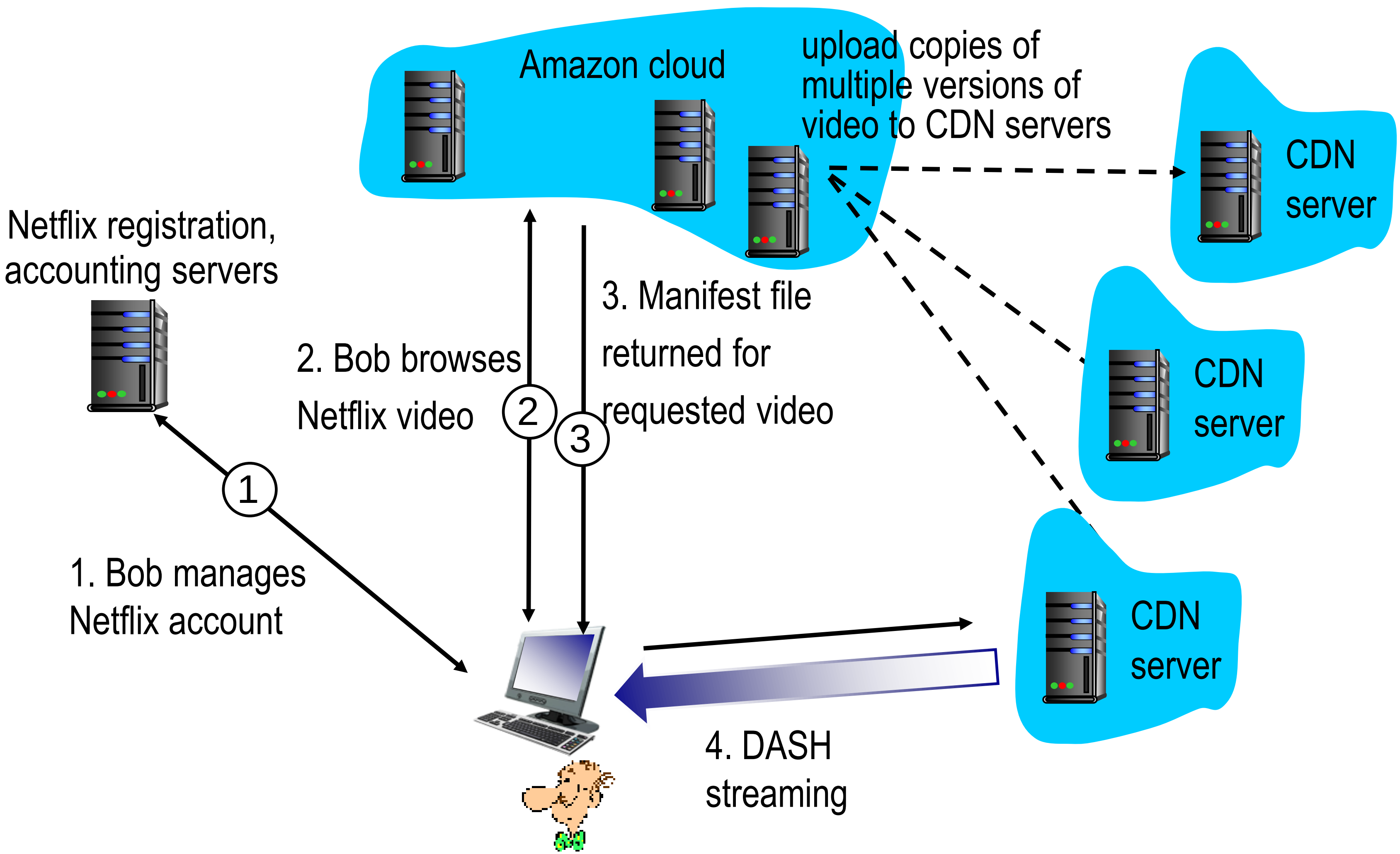
- *alternative:* let *client* decide - give client a list of several CDN servers
  - client pings servers, picks "best"
  - Netflix approach

CASE STUDIES

Case Study: Netflix

- owns very little infrastructure, uses 3rd party services:
  - own registration, payment servers
  - Amazon (3rd party) cloud services:
    - Netflix uploads studio master to Amazon cloud
    - create multiple version of movie (different endodings) in cloud
    - upload versions from cloud to CDNs
    - Cloud hosts Netflix web pages for user browsing
  - *three* 3rd party CDNs host/stream Netflix content: Akamai, Limelight, Level-3
- Netflix distributes by pushing the videos to its CDN servers during off-peak hours.
  - Pushes only the most popular videos, determined on day to day basis.

Case study: Netflix



Case study: Youtube

- *Private CDN*
- use DNS to redirect client to a cluster
  - usually smallest RTT between client and cluster
  - may be directed to more distant cluster for load balancing
  - may also be redirected if cluster doesn't have the file

Case study: Kankan

- P2P video distribution, in China
- similar to BitTorrent
  - Client contacts tracker
  - download chunks of video from peers in parallel
  - focus on downloading chunks needed soon

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks

Chapter 2: summary

most importantly: learned about protocols!

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - *headers*: fields giving info about data
  - *data:* info(payload) being communicated

important themes:

- control vs. messages
- centralized vs. decentralized
- reliable vs. unreliable message transfer
- "complexity at network edge"