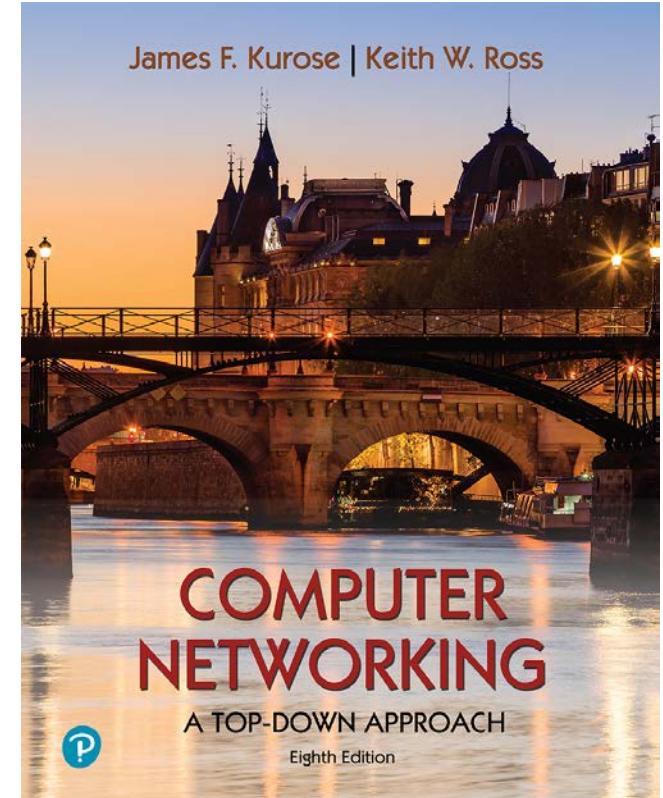


Chapter 2

Application Layer

All material copyright 1996-2023
J.F Kurose and K.W. Ross, All Rights Reserved



*Computer Networking: A
Top-Down Approach*
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

Application Layer: Overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
 - Application level service
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP



Application Layer: Overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
 - Translate host names to IP addresses
 - Application level service
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP



Domain Name System (DNS)

- DNS structure, function
- Resolving DNS queries
- DNS record format
- DNS protocol messages

DNS: Domain Name System

people: many identifiers:

- SSN, name, passport #

DNS: Domain Name System

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams e.g., 128.119.40.186.
- “name”, e.g., cs.umass.edu - used by humans

Q: how to map between IP address and name, and vice versa ?

DNS: Domain Name System

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams e.g., 128.119.40.186.
- “name”, e.g., cs.umass.edu - used by humans

Q: how to map between IP address and name, and vice versa ?

Domain Name System (DNS):

- *distributed database* implemented in hierarchy of many *name servers*

DNS: Domain Name System

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams e.g., 128.119.40.186.
- “name”, e.g., cs.umass.edu - used by humans

Q: how to map between IP address and name, and vice versa ?

Domain Name System (DNS):

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol:* hosts, DNS servers communicate to *resolve* names (address/name translation)
 - *note:* core Internet function, **implemented as application-layer protocol**
 - complexity at network’s “edge”

DNS: services, structure

DNS services:

- hostname-to-IP-address translation
- host aliasing
 - canonical, alias names
- mail server aliasing
- load distribution
 - replicated Web servers: many IP addresses correspond to one name

DNS: services, structure

DNS services:

- hostname-to-IP-address translation
- host aliasing
 - canonical, alias names
- mail server aliasing
- load distribution
 - replicated Web servers: many IP addresses correspond to one name

Q: Why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

DNS: services, structure

DNS services:

- hostname-to-IP-address translation
- host aliasing
 - canonical, alias names
- mail server aliasing
- load distribution
 - replicated Web servers: many IP addresses correspond to one name

Q: Why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

A: doesn't scale!

- Comcast DNS servers alone: 600B DNS queries/day
- Akamai DNS servers alone: 2.2T DNS queries/day

Thinking about the DNS

humongous distributed database:

- ~ billion records, each simple

Thinking about the DNS

humongous distributed database:

- ~ billion records, each simple

handles many *trillions* of queries/day:

- *many* more reads than writes
- *performance matters*: almost every Internet transaction interacts with DNS - msecs count!

Thinking about the DNS

humongous distributed database:

- ~ billion records, each simple

handles many *trillions* of queries/day:

- *many* more reads than writes
- *performance matters*: almost every Internet transaction interacts with DNS - msecs count!

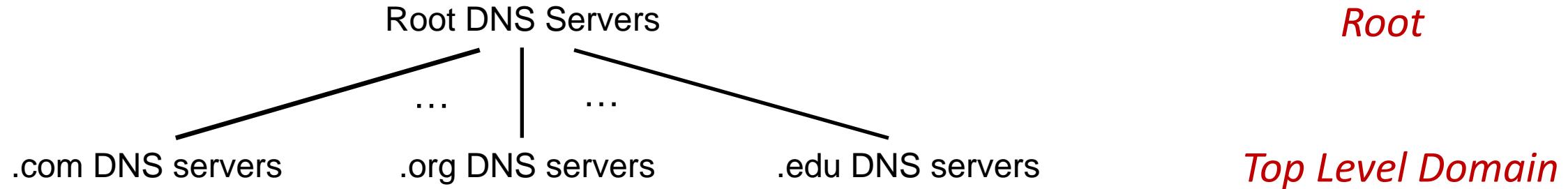
organizationally, physically decentralized:

- millions of different organizations responsible for their records

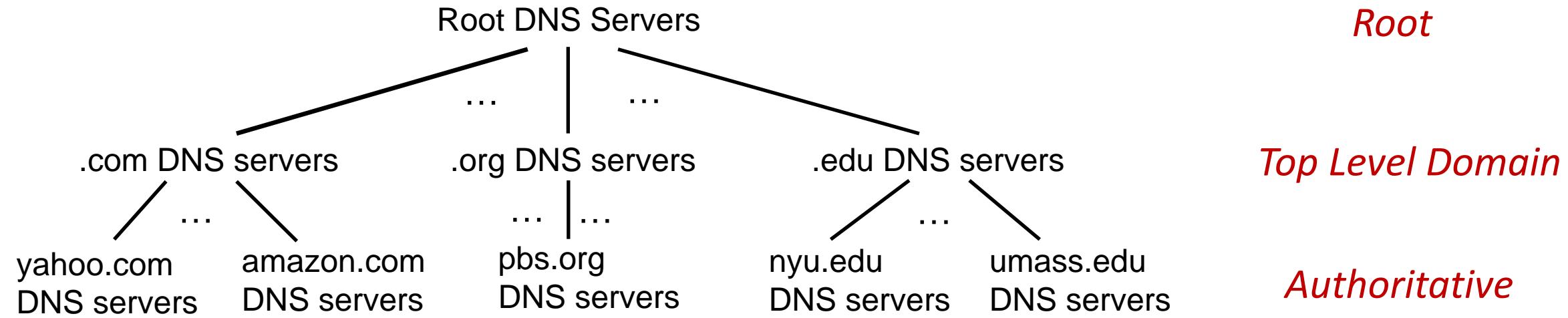
“bulletproof”: reliability, security



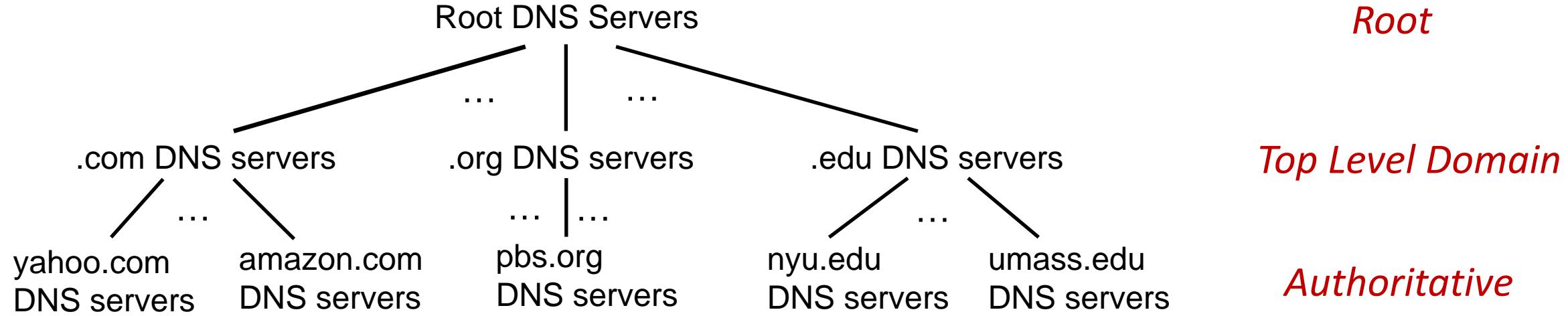
DNS: a distributed, hierarchical database



DNS: a distributed, hierarchical database



DNS: a distributed, hierarchical database

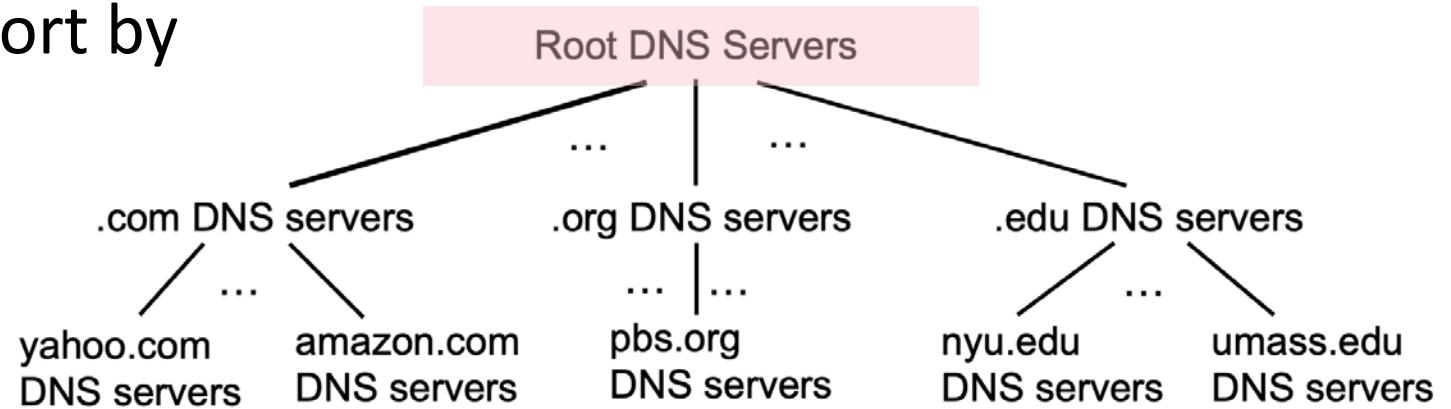


Client wants IP address for www.amazon.com; 1st approximation:

- client queries root server to find .com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: root name servers

- official, contact-of-last-resort by name servers that can not resolve name



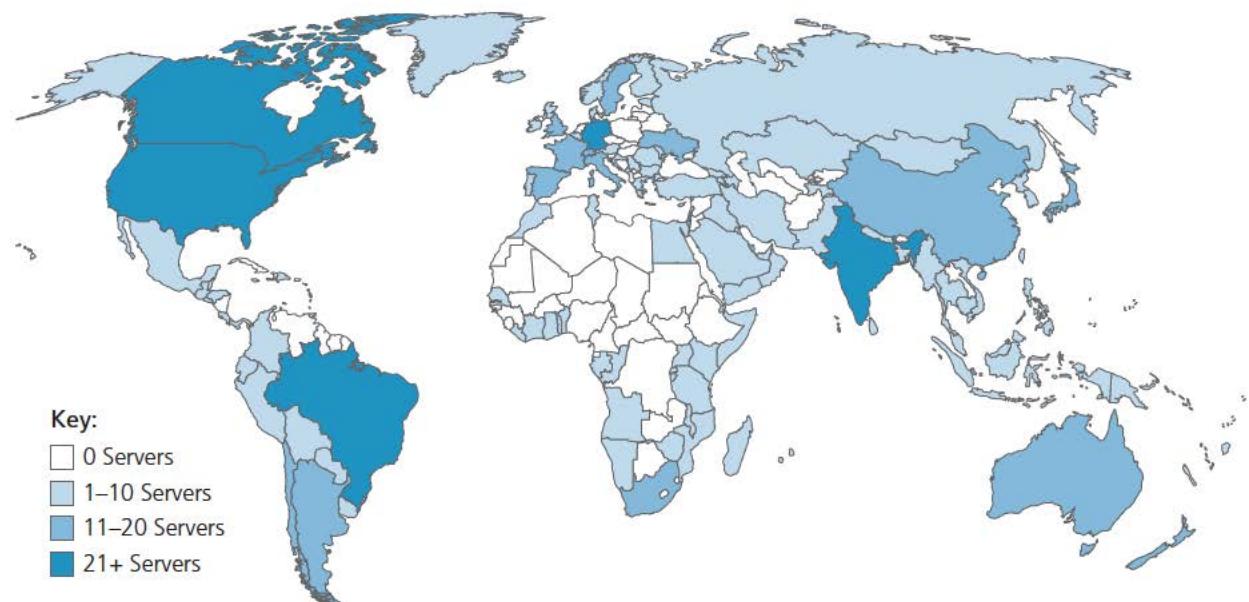
DNS: root name servers

- official, contact-of-last-resort by name servers that can not resolve name
- *incredibly important* Internet function
 - Internet couldn't function without it!
 - DNSSEC – provides security (authentication, message integrity)
- ICANN (Internet Corporation for Assigned Names and Numbers) manages root DNS domain

DNS: root name servers

- official, contact-of-last-resort by name servers that can not resolve name
- *incredibly important* Internet function
 - Internet couldn't function without it!
 - DNSSEC – provides security (authentication, message integrity)
- ICANN (Internet Corporation for Assigned Names and Numbers) manages root DNS domain

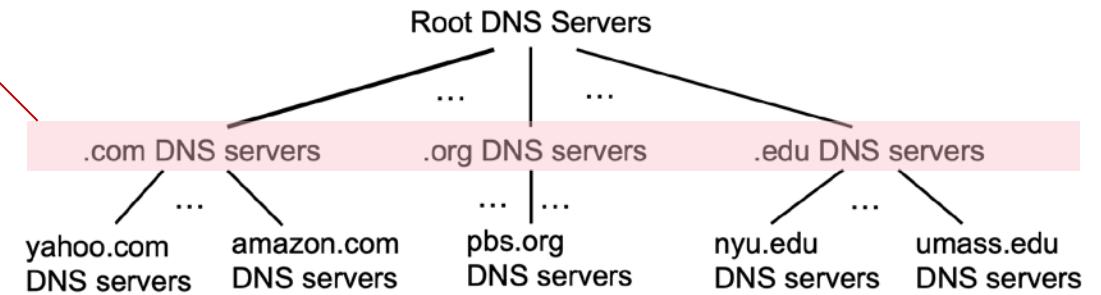
13 logical root name “servers” worldwide each “server” replicated many times (~200 servers in US)



Top-Level Domain, and authoritative servers

Top-Level Domain (TLD) servers:

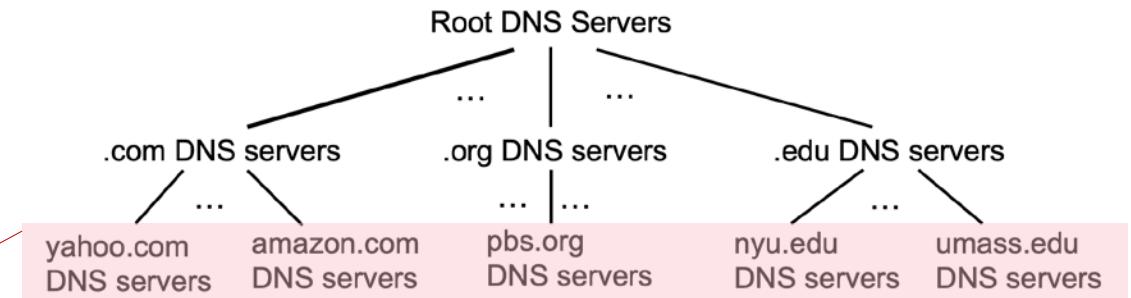
- responsible for .com, .org, .net, .edu, .aero, .jobs, .museums, and all top-level country domains, e.g.: .cn, .uk, .fr, .ca, .jp
- Network Solutions: authoritative registry for .com, .net TLD
- Educause: .edu TLD



Top-Level Domain, and authoritative servers

Top-Level Domain (TLD) servers:

- responsible for .com, .org, .net, .edu, .aero, .jobs, .museums, and all top-level country domains, e.g.: .cn, .uk, .fr, .ca, .jp
- Network Solutions: authoritative registry for .com, .net TLD
- Educause: .edu TLD



authoritative DNS servers:

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

Local DNS name servers

- when host makes DNS query, it is sent to its *local* DNS server
 - Local DNS server returns reply, answering:
 - from its local cache of recent name-to-address translation pairs (possibly out of date!)
 - forwarding request into DNS hierarchy for resolution

Local DNS name servers

- when host makes DNS query, it is sent to its *local* DNS server
 - Local DNS server returns reply, answering:
 - from its local cache of recent name-to-address translation pairs (possibly out of date!)
 - forwarding request into DNS hierarchy for resolution
 - each ISP has local DNS name server; to find yours:
 - MacOS: % scutil --dns
 - Windows: >ipconfig /all

DNS name resolution: iterated query

Example: host at `engineering.nyu.edu`
wants IP address for `gaia.cs.umass.edu`

root DNS server



TLD DNS server



requesting host at
engineering.nyu.edu

local DNS server
dns.nyu.edu



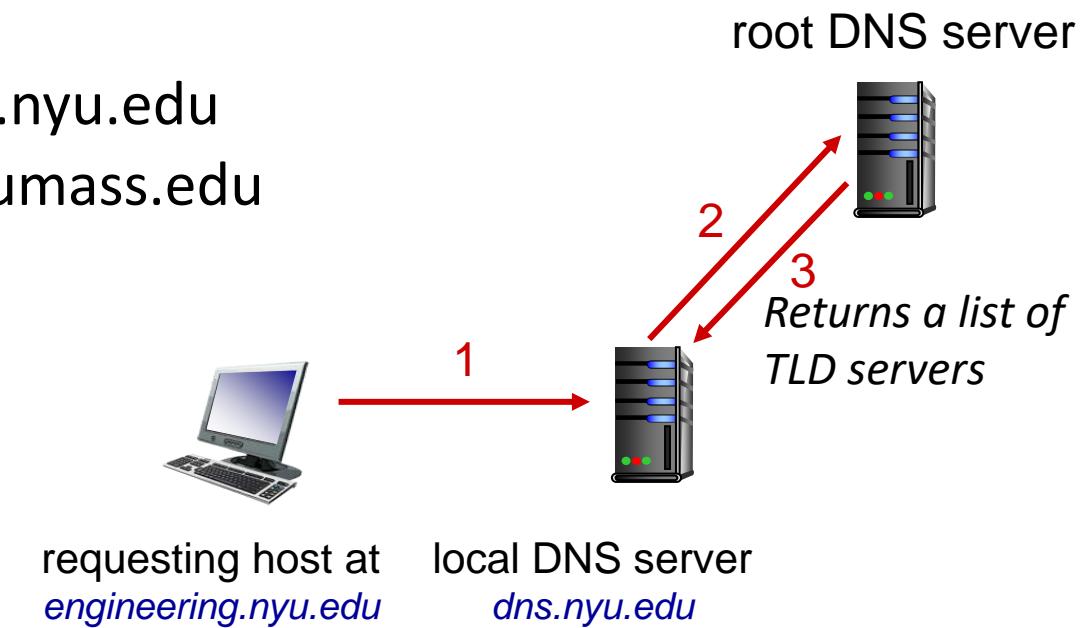
gaia.cs.umass.edu



authoritative DNS server
dns.cs.umass.edu

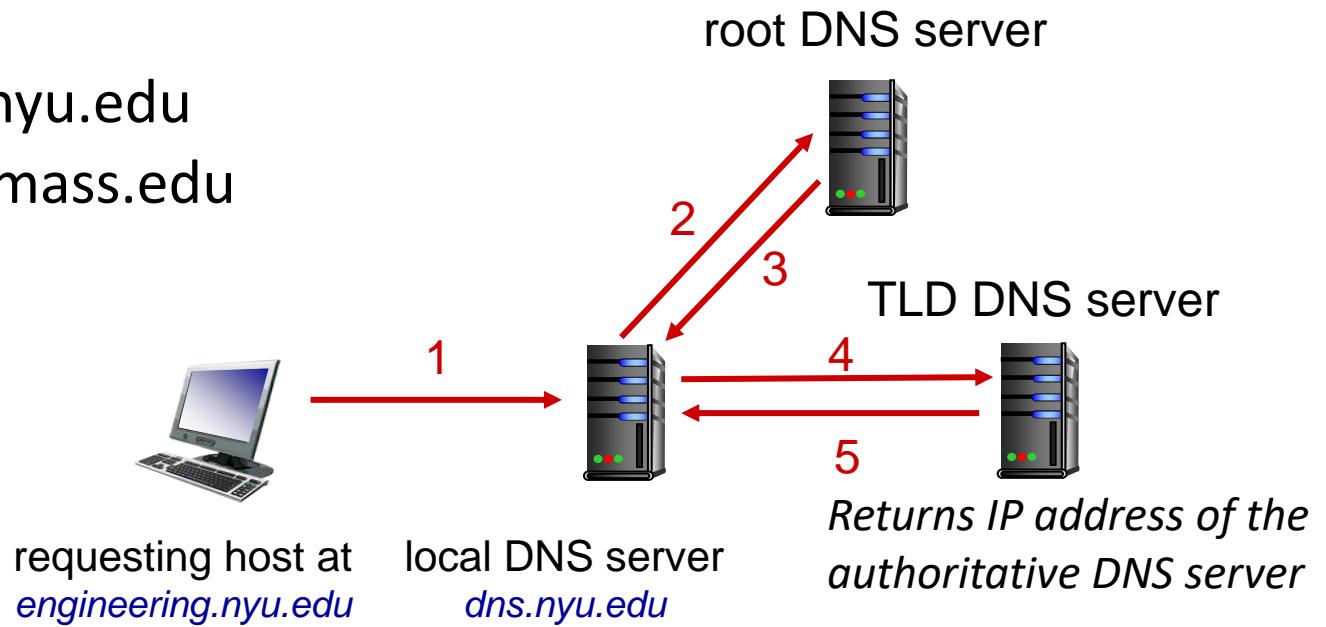
DNS name resolution: iterated query

Example: host at engineering.nyu.edu
wants IP address for gaia.cs.umass.edu



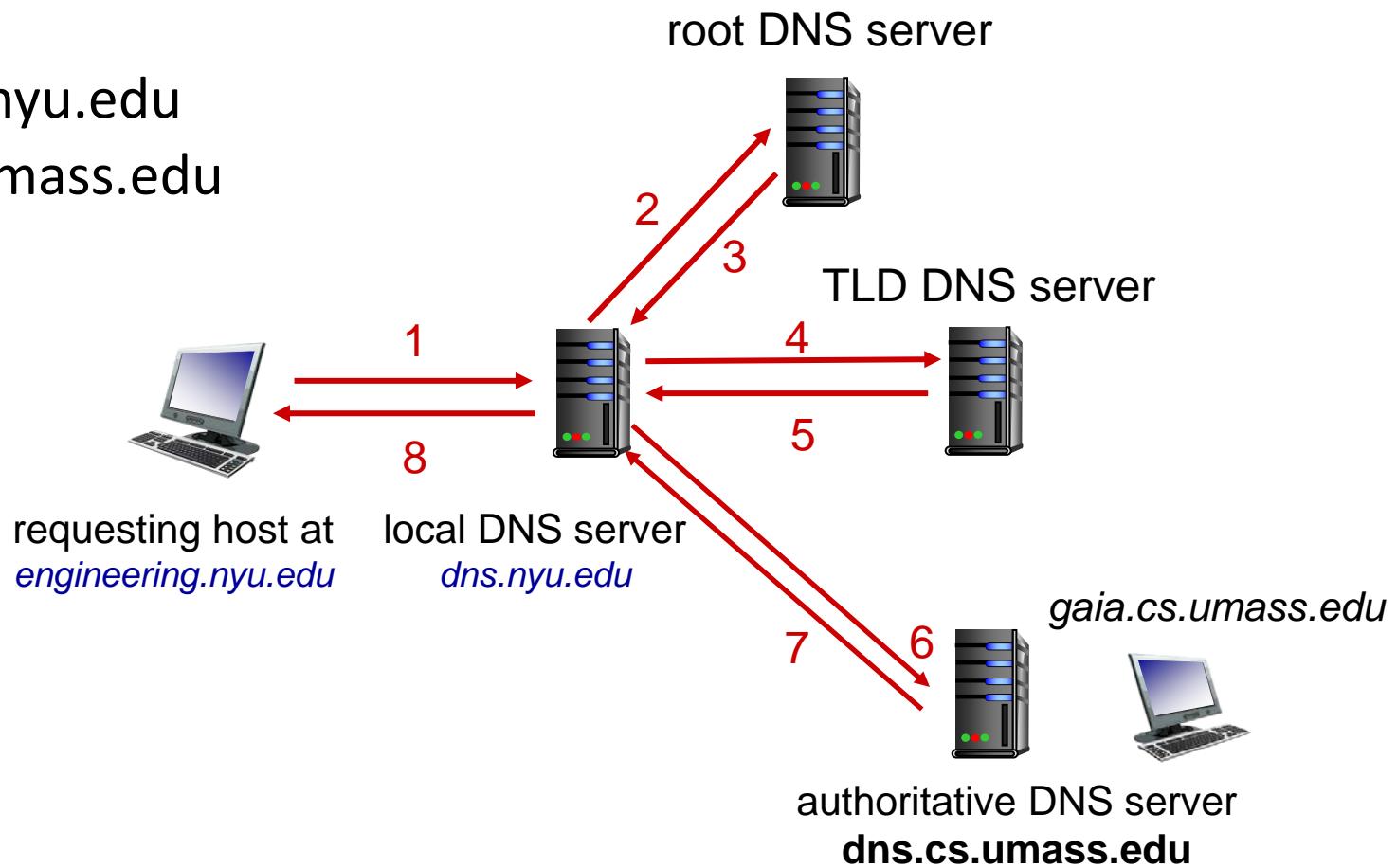
DNS name resolution: iterated query

Example: host at engineering.nyu.edu
wants IP address for gaia.cs.umass.edu



DNS name resolution: iterated query

Example: host at engineering.nyu.edu
wants IP address for gaia.cs.umass.edu

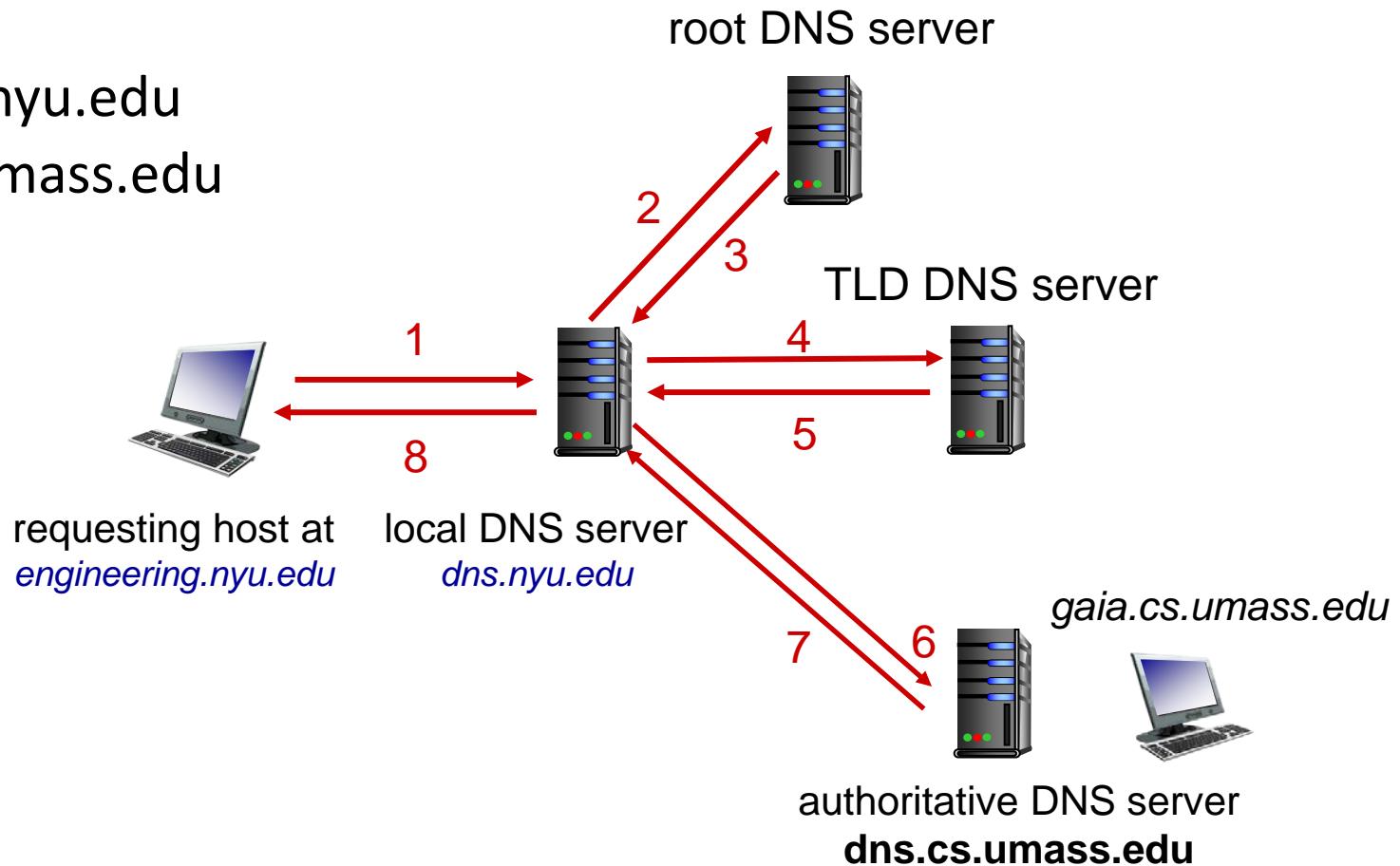


DNS name resolution: iterated query

Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

Iterated query:

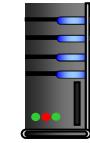
- contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”



DNS name resolution: recursive query

Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

root DNS server

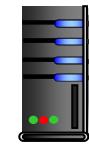


Recursive query:

- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?



requesting host at
engineering.nyu.edu



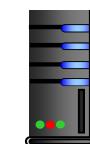
local DNS server
dhs.nyu.edu



TLD DNS server



gaia.cs.umass.edu



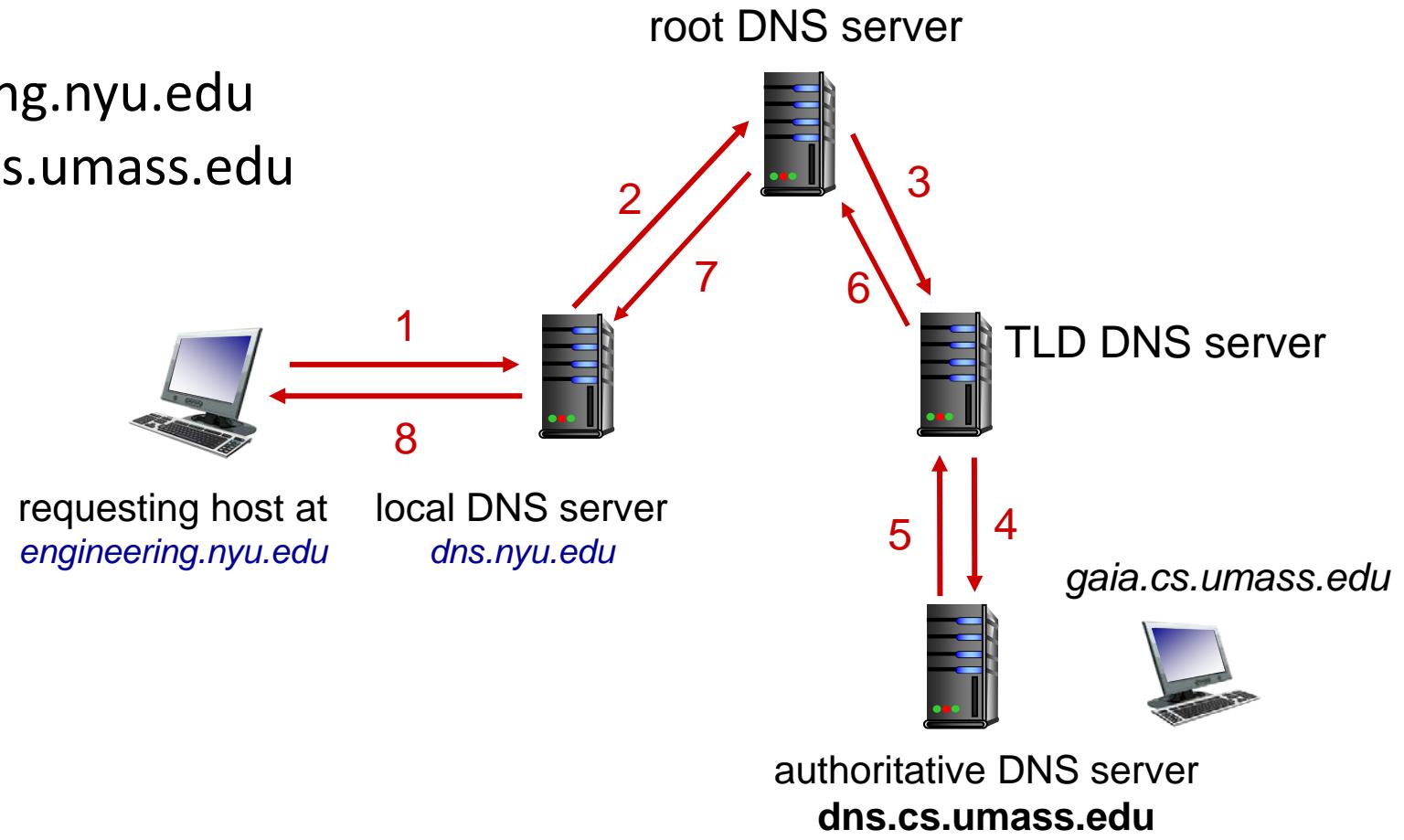
authoritative DNS server
dns.cs.umass.edu

DNS name resolution: recursive query

Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

Recursive query:

- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?



Caching DNS Information

- once (any) name server learns mapping, it *caches* mapping, and *immediately* returns a cached mapping in response to a query

Caching DNS Information

- once (any) name server learns mapping, it *caches* mapping, and *immediately* returns a cached mapping in response to a query
 - caching improves response time
 - cache entries timeout (disappear) after some time (TTL)
 - TLD servers typically cached in local name servers

Caching DNS Information

- once (any) name server learns mapping, it *caches* mapping, and *immediately* returns a cached mapping in response to a query
 - caching improves response time
 - cache entries timeout (disappear) after some time (TTL)
 - TLD servers typically cached in local name servers
- cached entries may be *out-of-date*
 - if named host changes IP address, may not be known Internet-wide until all TTLs expire!
 - *best-effort name-to-address translation!*

DNS records

DNS: distributed database storing resource records (RR)

RR format: (name, value, type, ttl)

DNS records

DNS: distributed database storing resource records (RR)

RR format: (name, value, type, ttl)

type=A

- name is hostname
- value is IP address

type=NS

- name is domain (e.g., foo.com)
- value is hostname of authoritative name server for this domain

type=CNAME

- name is alias name for some “canonical” (the real) name
- www.ibm.com is really servereast.backup2.ibm.com
- value is canonical name

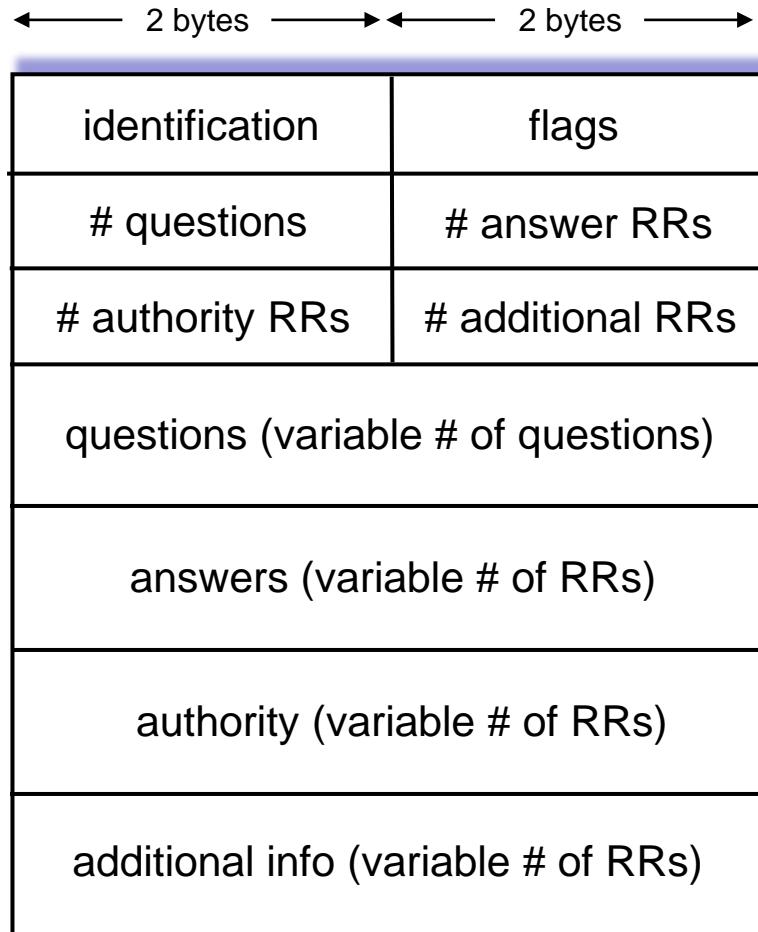
type=MX

- value is name of SMTP mail server associated with name

DNS protocol messages

DNS *query* and *reply* messages, both have same *format*:

message header:

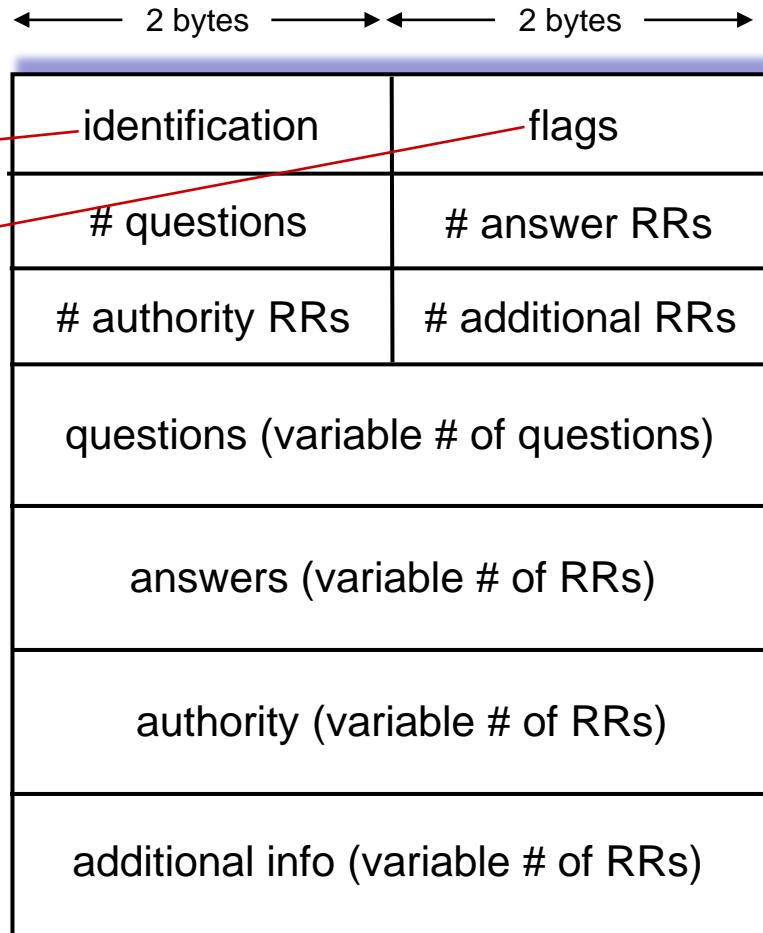


DNS protocol messages

DNS *query* and *reply* messages, both have same *format*:

message header:

- **identification:** 16 bit # for query,
reply to query uses same #
- **flags:**
 - query or reply
 - recursion desired
 - recursion available
 - reply is authoritative



DNS protocol messages

DNS *query* and *reply* messages, both have same *format*:

See RFC 1035 for details

↔ 2 bytes → ← 2 bytes →

identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

name, type fields for a query



questions (variable # of questions)

RRs in response to query



answers (variable # of RRs)

records for authoritative servers



authority (variable # of RRs)

additional “helpful” info that may be used



additional info (variable # of RRs)

How a DNS server works - recap

Source: <https://www.youtube.com/watch?v=mpQZVYPuDGU>

Getting your info into the DNS

example: new startup “Network Utopia” with site networkutopia.com

Getting your info into the DNS

example: new startup “Network Utopia”

- register name networkuptopia.com at *DNS registrar* (e.g., Network Solutions)

Getting your info into the DNS

example: new startup “Network Utopia”

- register name networkuptopia.com at *DNS registrar* (e.g., Network Solutions)
 - provide names, IP addresses of authoritative name server (primary and secondary)
 - registrar inserts NS, A RRs into .com TLD server:
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)

Getting your info into the DNS

example: new startup “Network Utopia”

- register name networkuptopia.com at *DNS registrar* (e.g., Network Solutions)
 - provide names, IP addresses of authoritative name server (primary and secondary)
 - registrar inserts NS, A RRs into .com TLD server:
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
- create authoritative server locally with IP address 212.212.212.1
 - type A record for www.networkuptopia.com
 - type MX record for networkutopia.com

DNS security

DDoS attacks (denial of service)

- bombard root servers with traffic
 - not successful to date
 - traffic filtering
 - local DNS servers cache IPs of TLD servers, allowing root server bypass
- bombard TLD servers
 - potentially more dangerous

Spoofing attacks

- intercept DNS queries, returning bogus replies
 - DNS cache poisoning
 - RFC 4033: DNSSEC authentication services

Break

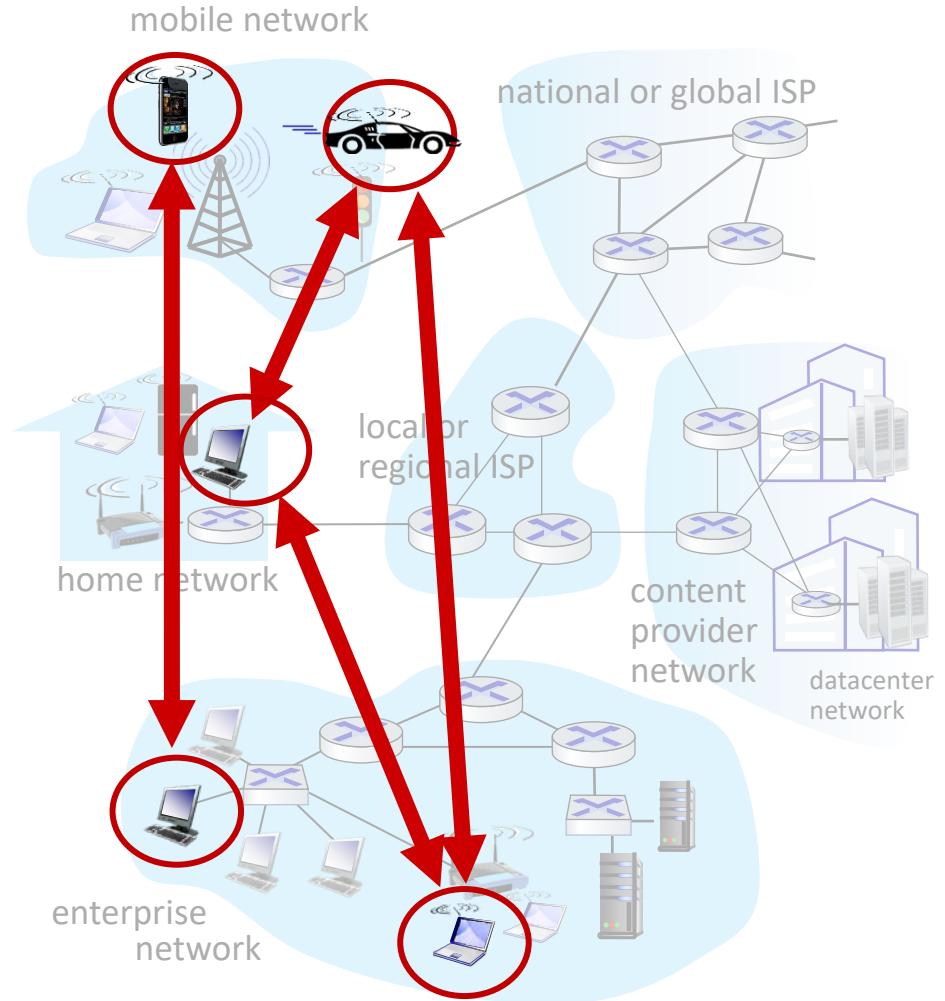
Application Layer: Overview

- Principles of network applications
 - Web and HTTP
 - E-mail, SMTP, IMAP
 - The Domain Name System DNS
- P2P applications
 - video streaming and content distribution networks
 - socket programming with UDP and TCP



Peer-to-peer (P2P) architecture

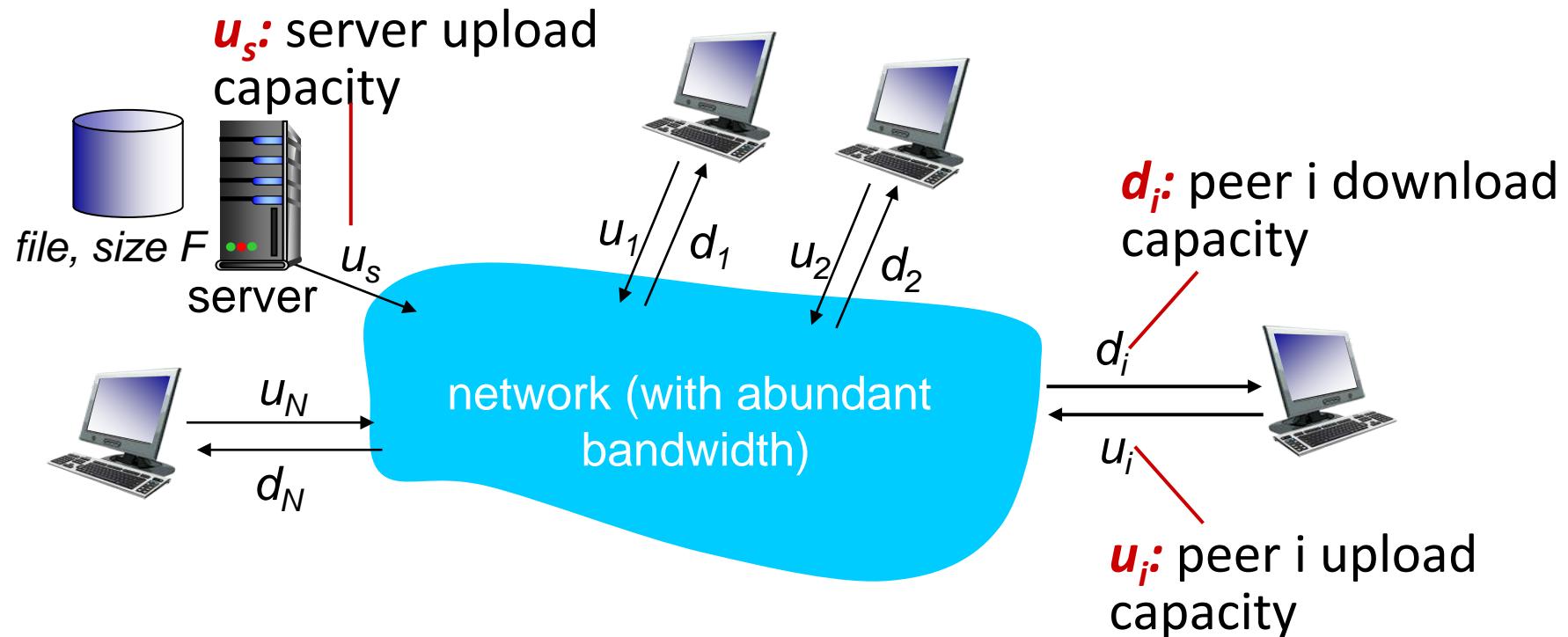
- no always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
 - *self scalability* – new peers bring new service capacity, and new service demands
- peers are intermittently connected and change IP addresses
 - complex management
- examples: P2P file sharing (BitTorrent), streaming (KanKan), VoIP (Skype)



File distribution: client-server vs P2P

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

- peer upload/download capacity is limited resource



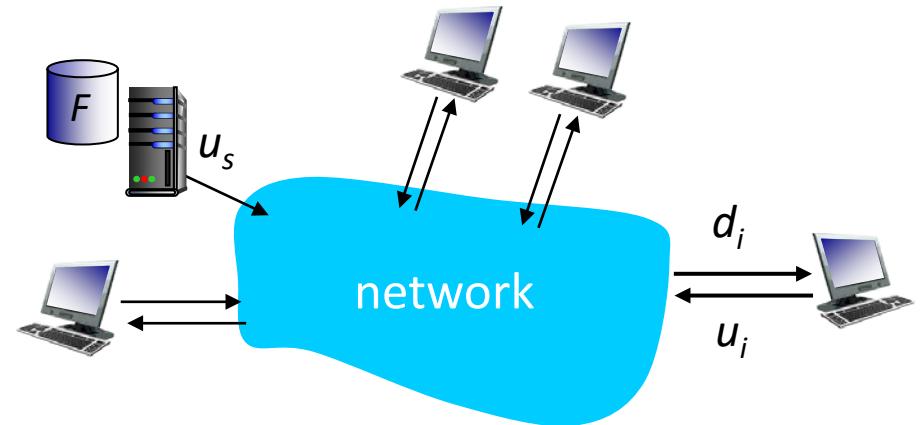
File distribution time: client-server

- *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

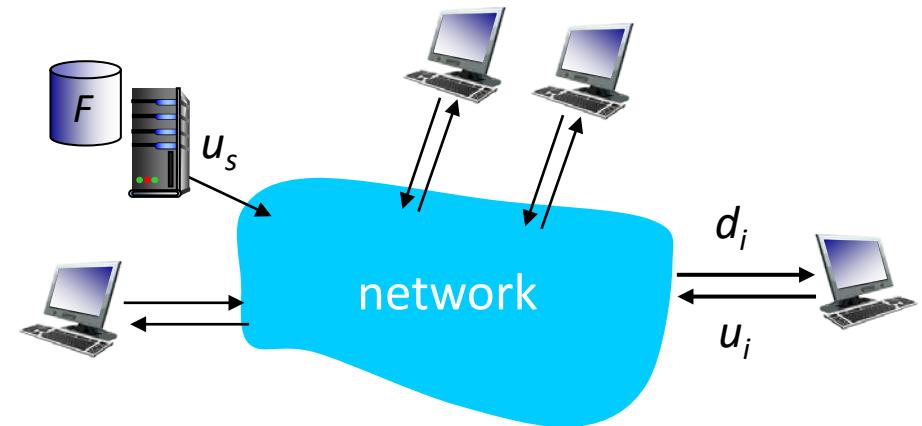
F : file size

u_s : upload bandwidth used by the server



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}



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

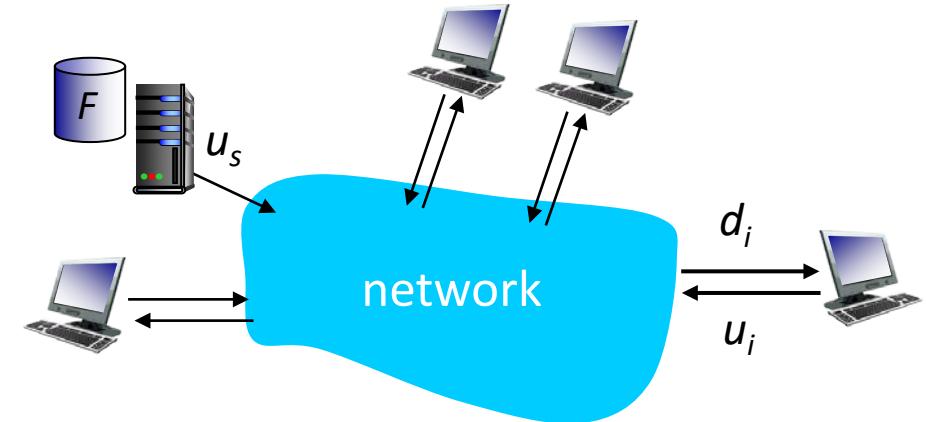
- *client*: each client must download file copy

- d_{min} = min client download rate (download rate of slowest client)
- min client download time: F/d_{min}

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

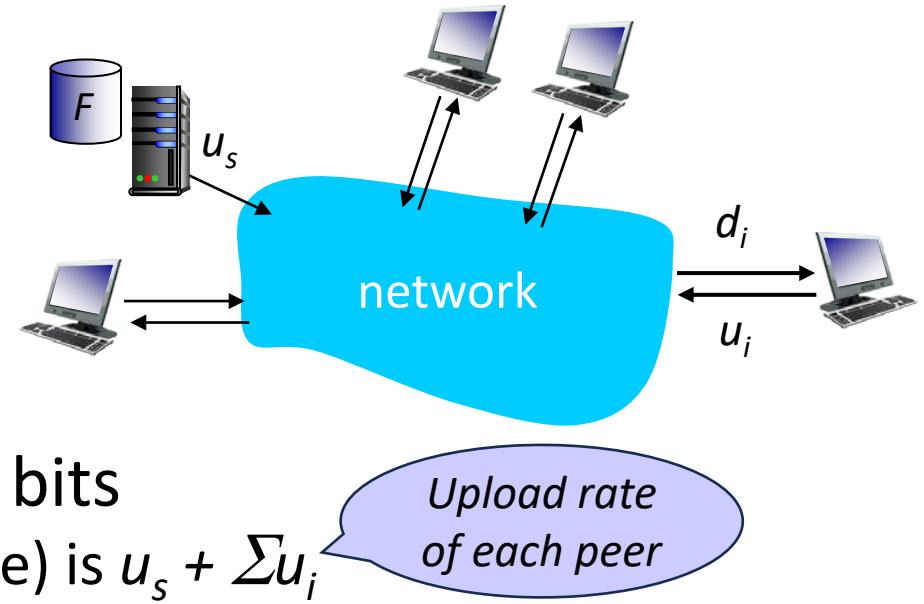
$$D_{c-s} \geq \max\{NF/u_s, F/d_{min}\}$$

increases linearly in N



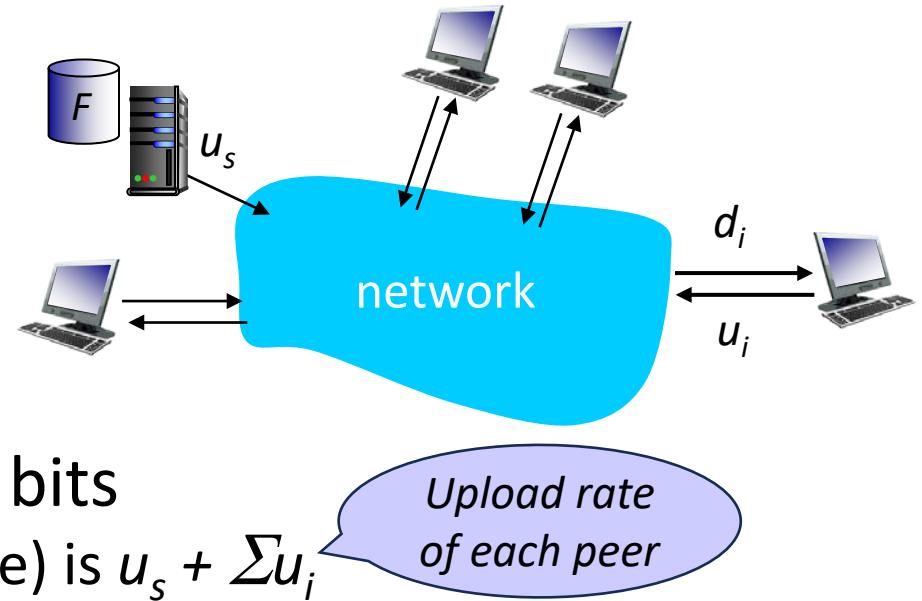
File distribution time: P2P

- *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
 - max upload rate (limiting max download rate) is $u_s + \sum u_i$



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
 - max upload rate (limiting max download rate) is $u_s + \sum u_i$

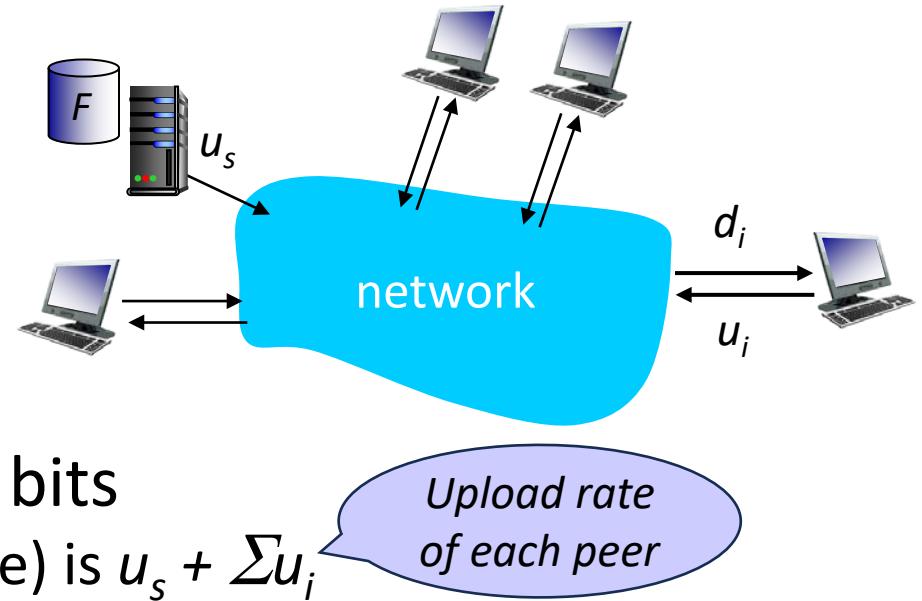


time to distribute F
to N clients using
P2P approach

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

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
 - max upload rate (limiting max download rate) is $u_s + \sum u_i$



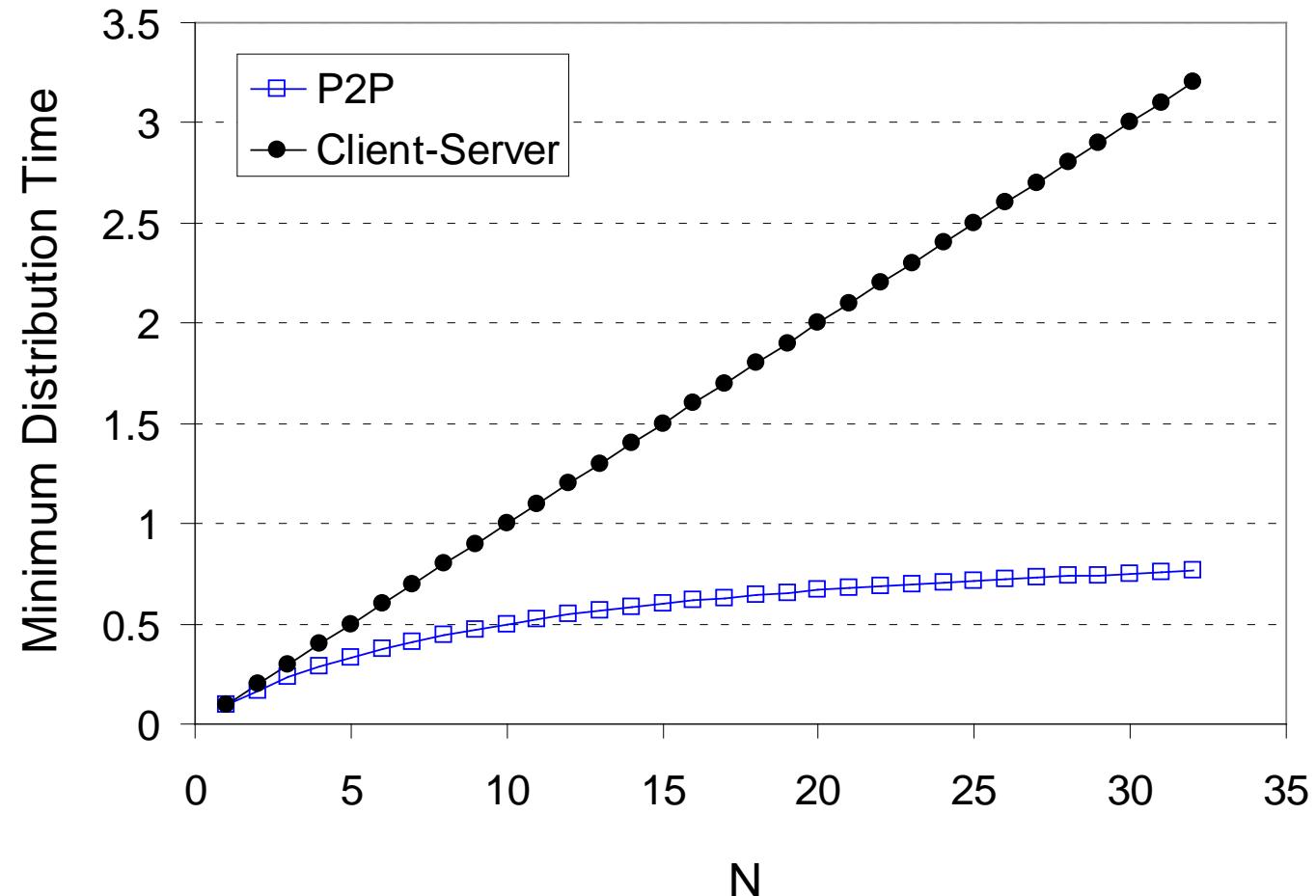
time to distribute F
to N clients using
P2P approach

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

increases linearly in N ...
... but so does this, as each peer brings service capacity

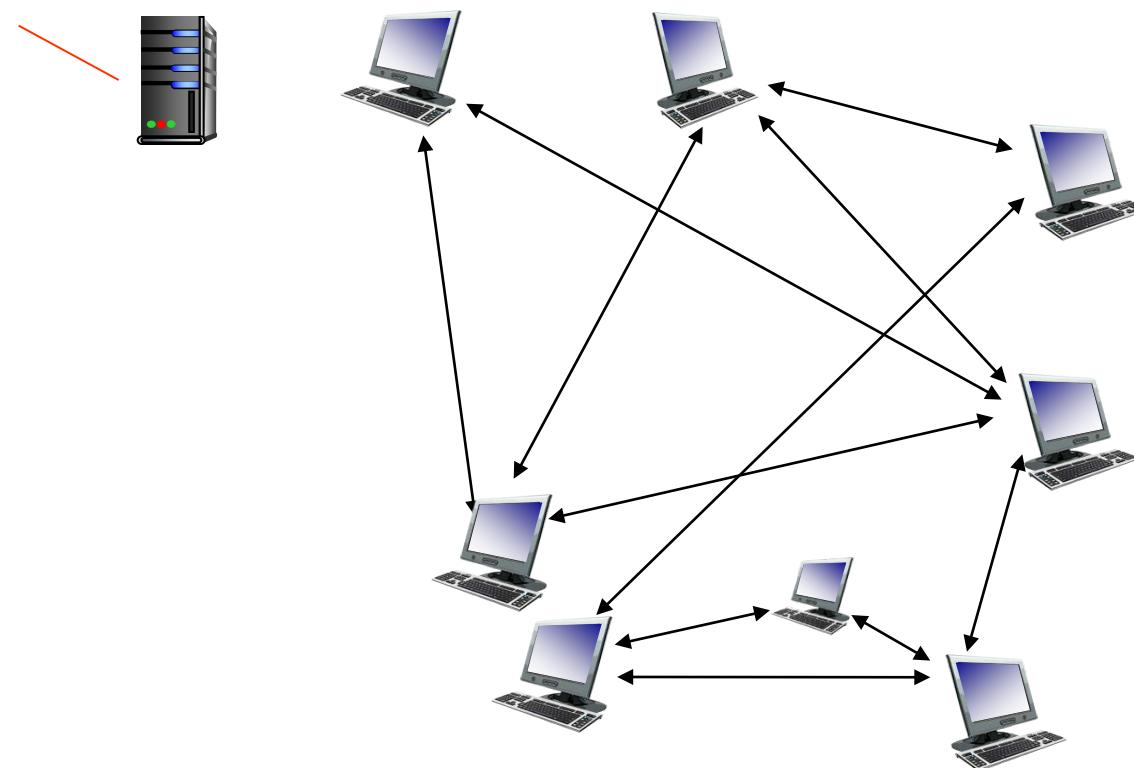
Client-server vs. P2P: example

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



P2P file distribution: BitTorrent

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



P2P file distribution: BitTorrent

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

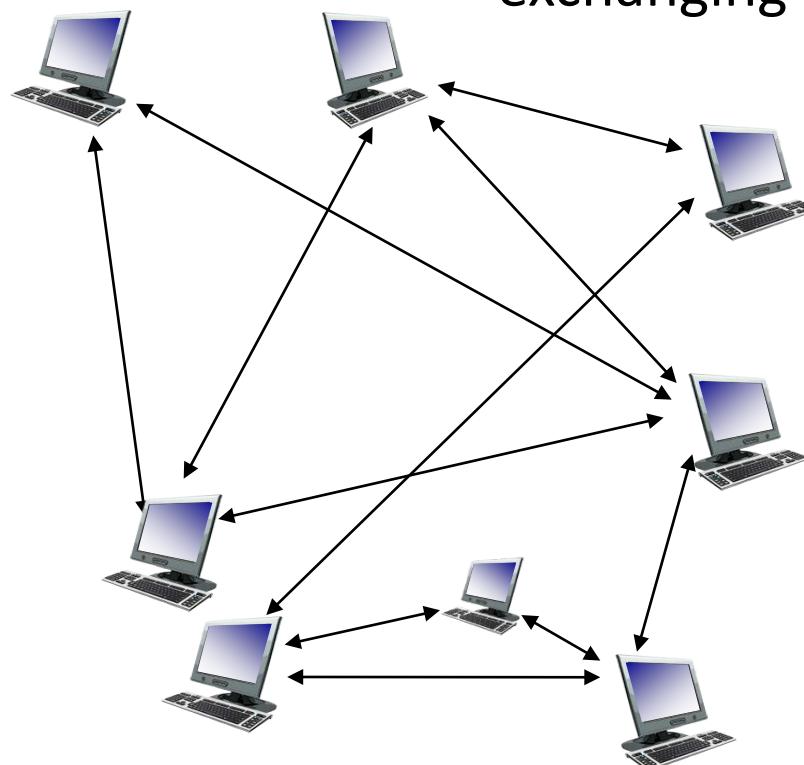
tracker: tracks peers participating in torrent



Still needs an always-on server (the tracker).

Improvements allow BitTorrent to work with a distributed tracker

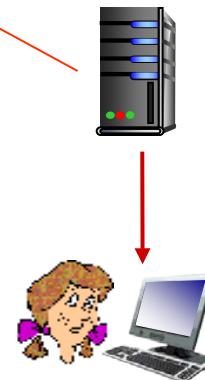
torrent: group of peers exchanging chunks of a file



P2P file distribution: BitTorrent

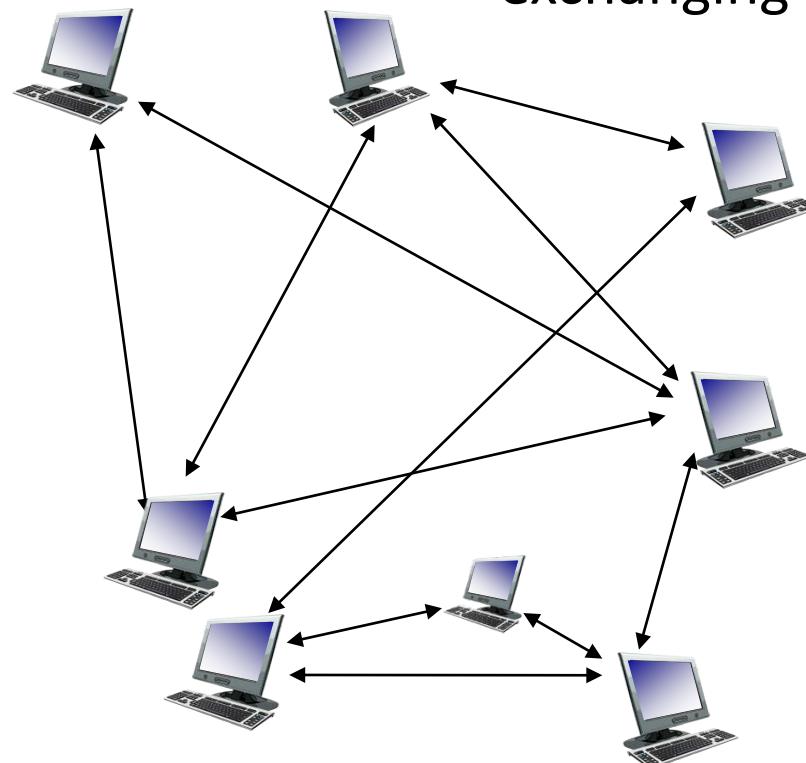
- file divided into 256Kb chunks
- peers in torrent send/receive file chunks

tracker: tracks peers
participating in torrent



Alice arrives ...
... obtains list
of peers from tracker

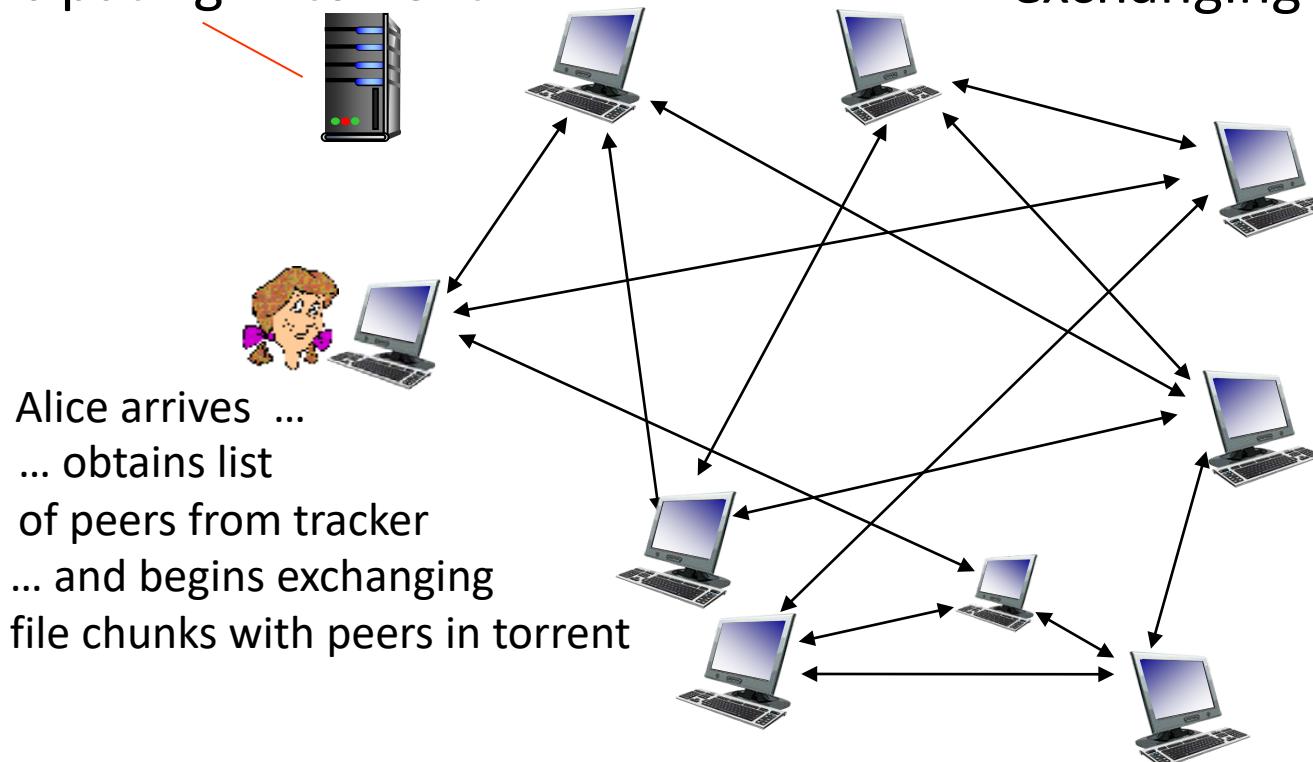
torrent: group of peers
exchanging chunks of a file



P2P file distribution: BitTorrent

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

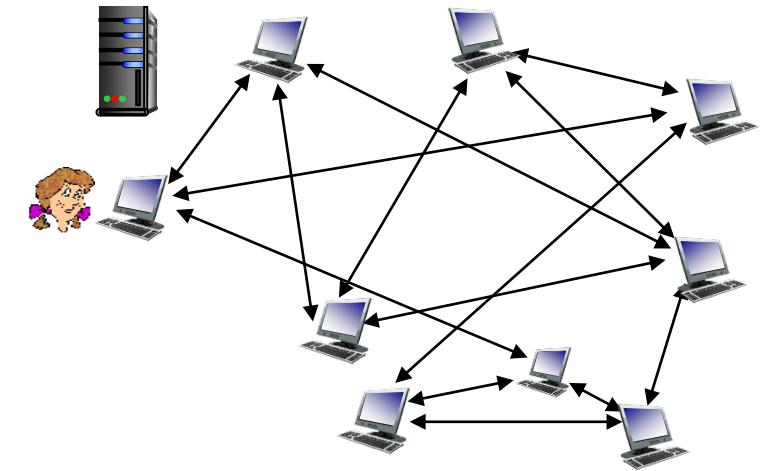
tracker: tracks peers
participating in torrent



torrent: group of peers
exchanging chunks of a file

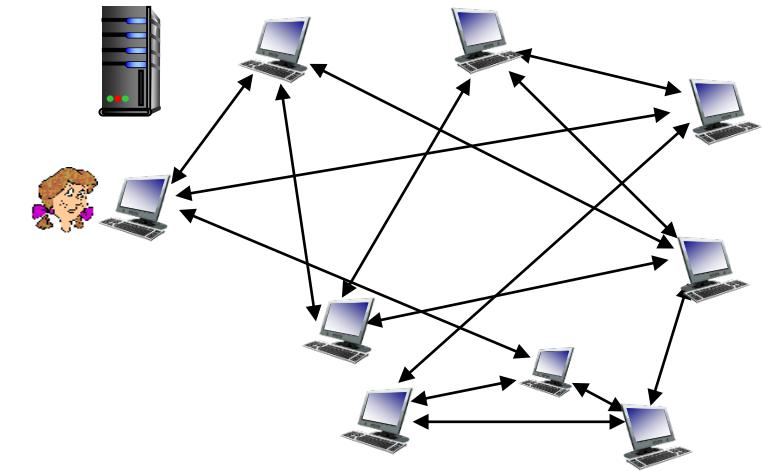
P2P file distribution: BitTorrent

- peer joining torrent:
 - has no chunks, but will accumulate them over time from other peers
 - registers with tracker to get list of peers, connects to subset of peers (“neighbors”)
- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn*: peers may come and go



P2P file distribution: BitTorrent

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



BitTorrent: requesting, sending file chunks

Requesting chunks:

- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have

BitTorrent: requesting, sending file chunks

Requesting chunks:

- 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, rarest first
 - Preserve the torrent

BitTorrent: requesting, sending file chunks

Requesting chunks:

- 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, rarest first
 - Preserve the torrent

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

BitTorrent: requesting, sending file chunks

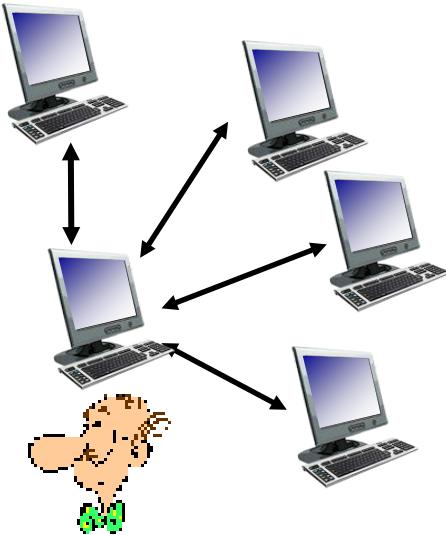
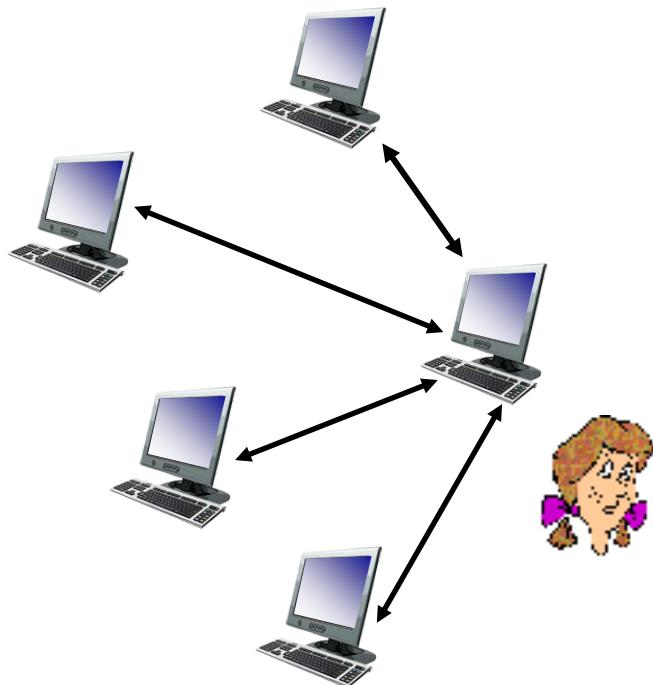
Requesting chunks:

- 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, rarest first
 - Preserve the torrent

Sending chunks: tit-for-tat

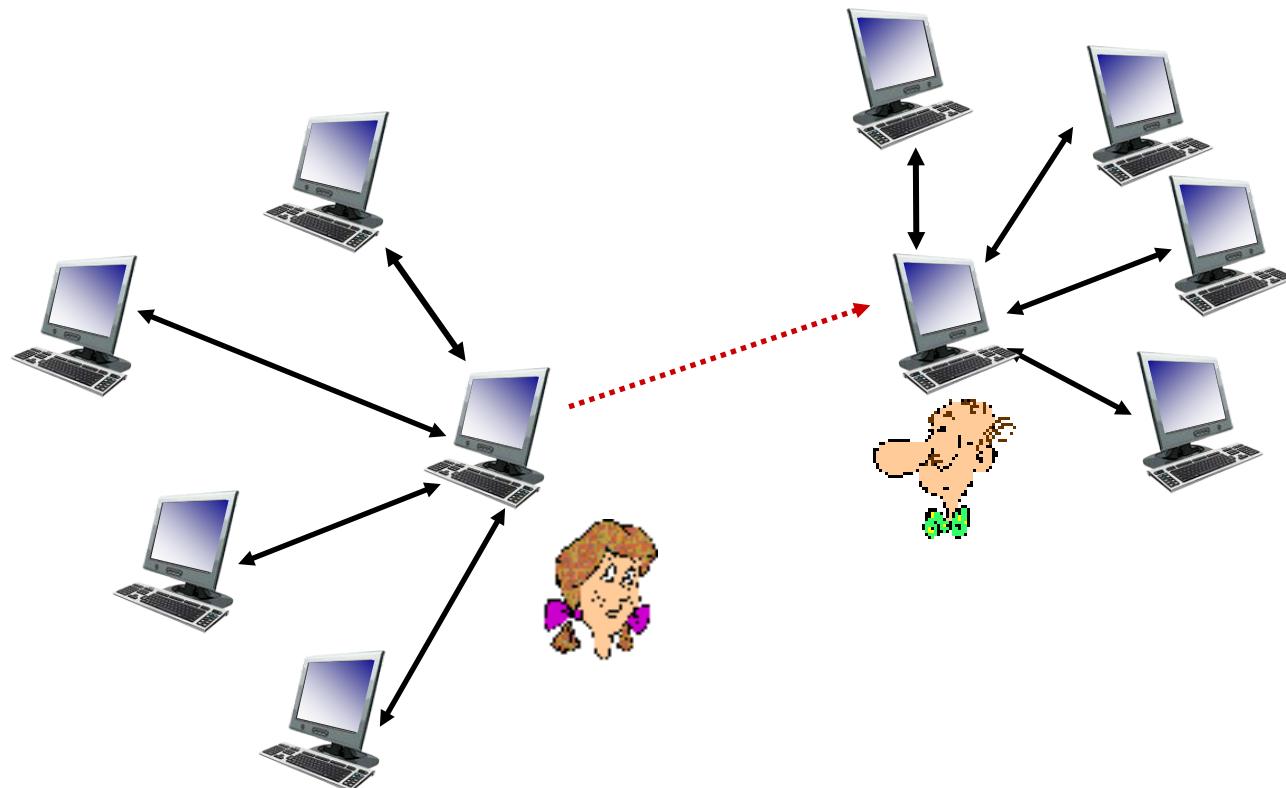
- Alice sends chunks to those four peers currently sending her chunks *at highest rate*
 - other peers are choked by Alice (do not receive chunks from her)
 - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
 - “optimistically unchoke” this peer
 -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

BitTorrent: tit-for-tat



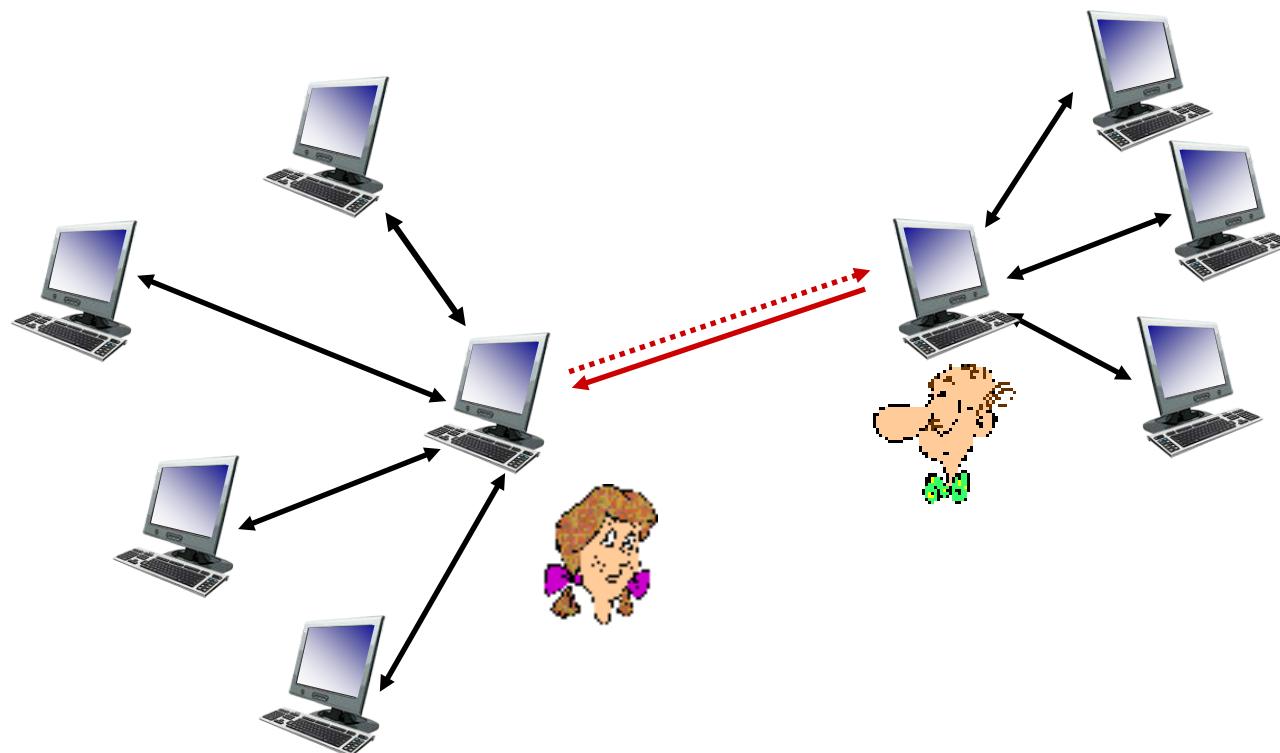
BitTorrent: tit-for-tat

(1) Alice “optimistically unchoke” Bob



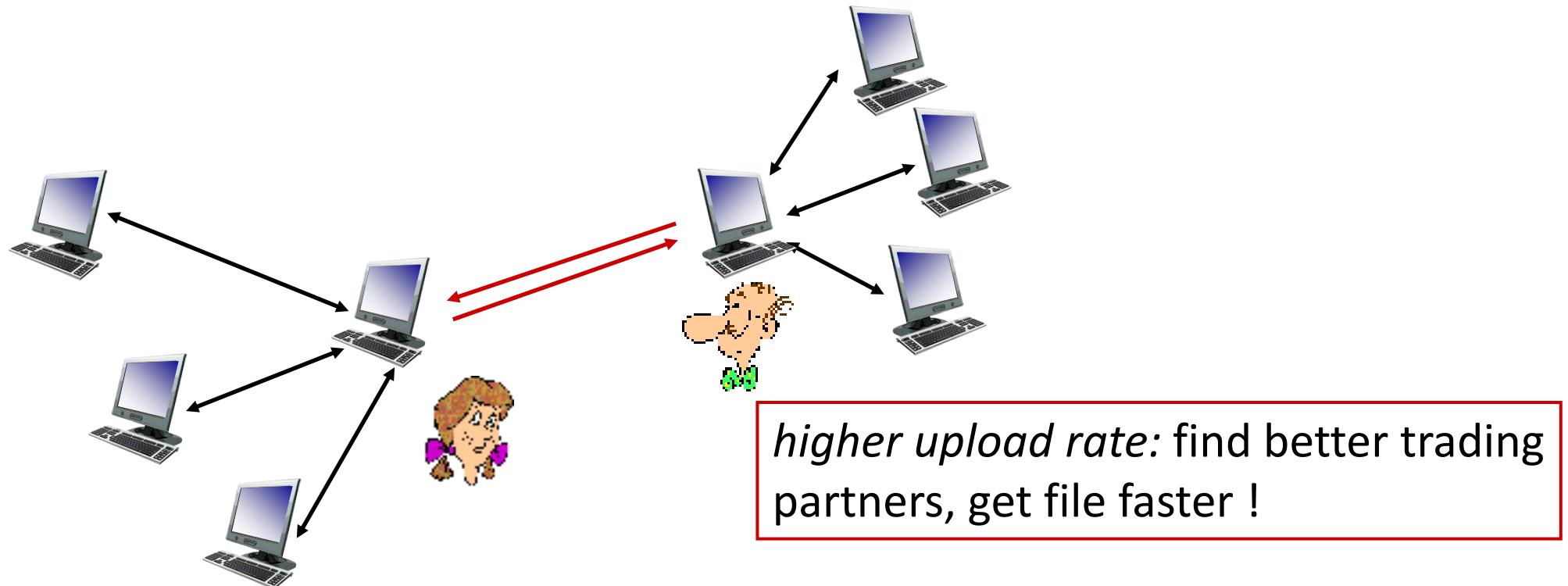
BitTorrent: tit-for-tat

- (1) Alice “optimistically unchoke” Bob
- (2) Alice becomes one of Bob’s top-four providers; Bob reciprocates



BitTorrent: tit-for-tat

- (1) Alice “optimistically unchoke” Bob
- (2) Alice becomes one of Bob’s top-four providers; Bob reciprocates
- (3) Bob becomes one of Alice’s top-four providers



Application layer: overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP



Video Streaming and CDNs

- Video characteristics
- Streaming stored video
 - Buffering
 - Playout
- DASH – dynamic client-driven streaming
- CDNs example

Video Streaming and CDNs: context

- stream video traffic: major consumer of Internet bandwidth
 - Netflix, YouTube, Amazon Prime: 80% of residential ISP traffic (2020)

Video Streaming and CDNs: context

- stream video traffic: major consumer of Internet bandwidth
 - Netflix, YouTube, Amazon Prime: 80% of residential ISP traffic (2020)
- *challenge:* scale - how to reach ~1B users?
- *challenge:* heterogeneity
 - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- *solution:* distributed, application-level infrastructure



Multimedia: video

- video: sequence of images displayed at constant rate
 - e.g., 24 images/sec
- digital image: array of pixels
 - each pixel represented by bits



frame *i*



frame *i+1*

Multimedia: video

- video: sequence of images displayed at constant rate
 - e.g., 24 images/sec
- digital image: array of pixels
 - each pixel represented by bits
- coding: use redundancy *within* and *between* images to decrease # bits used to encode image
 - spatial (within image)

spatial coding example: instead of sending N values of same color (all purple), send only two values: color value (*purple*) and number of repeated values (N)



frame i

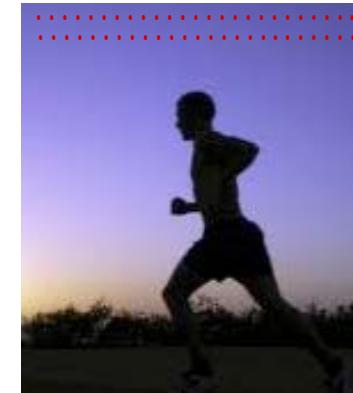


frame $i+1$

Multimedia: video

- video: sequence of images displayed at constant rate
 - e.g., 24 images/sec
- digital image: array of pixels
 - each pixel represented by bits
- coding: use redundancy *within* and *between* images to decrease # bits used to encode image
 - spatial (within image)
 - temporal (from one image to next)

spatial coding example: instead of sending N values of same color (all purple), send only two values: color value (*purple*) and number of repeated values (N)



frame *i*

temporal coding example: instead of sending complete frame at $i+1$, send only differences from frame i



frame *i+1*

Multimedia: video

- CBR: (constant bit rate): video encoding rate fixed
- VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes
- examples:
 - MPEG 1 (CD-ROM) 1.5 Mbps
 - MPEG2 (DVD) 3-6 Mbps
 - MPEG4 (often used in Internet, 64Kbps – 12 Mbps)

spatial coding example: instead of sending N values of same color (all purple), send only two values: color value (*purple*) and number of repeated values (N)



frame i

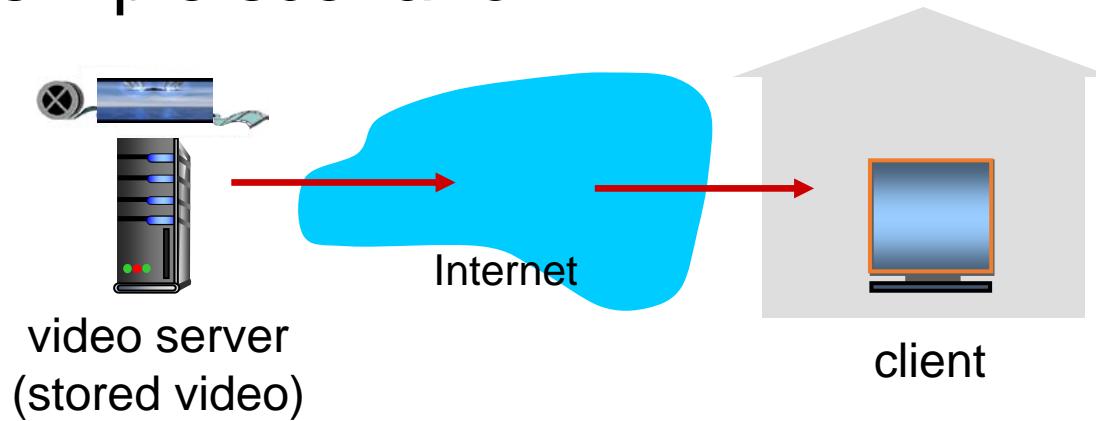
temporal coding example: instead of sending complete frame at $i+1$, send only differences from frame i



frame $i+1$

Streaming stored video

simple scenario:

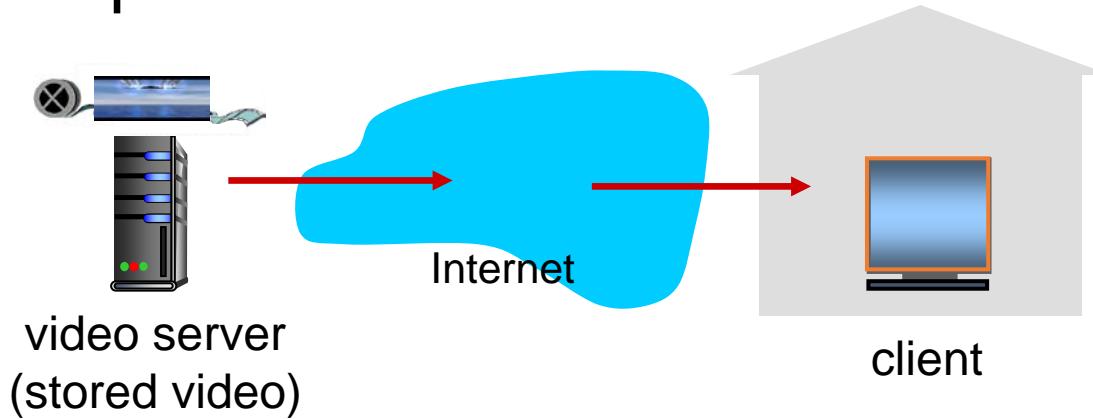


Main challenges:

- server-to-client bandwidth will *vary* over time, with changing network congestion levels (in house, access network, network core, video server)

Streaming stored video

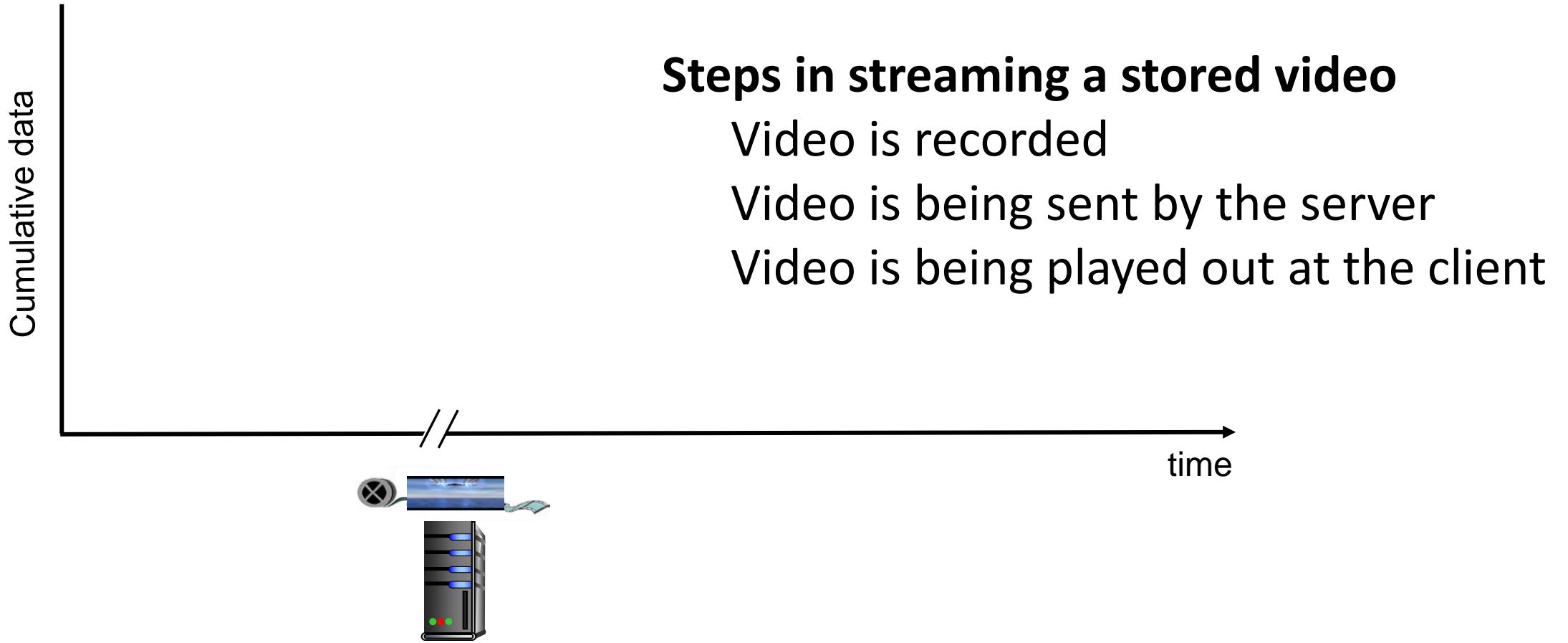
simple scenario:



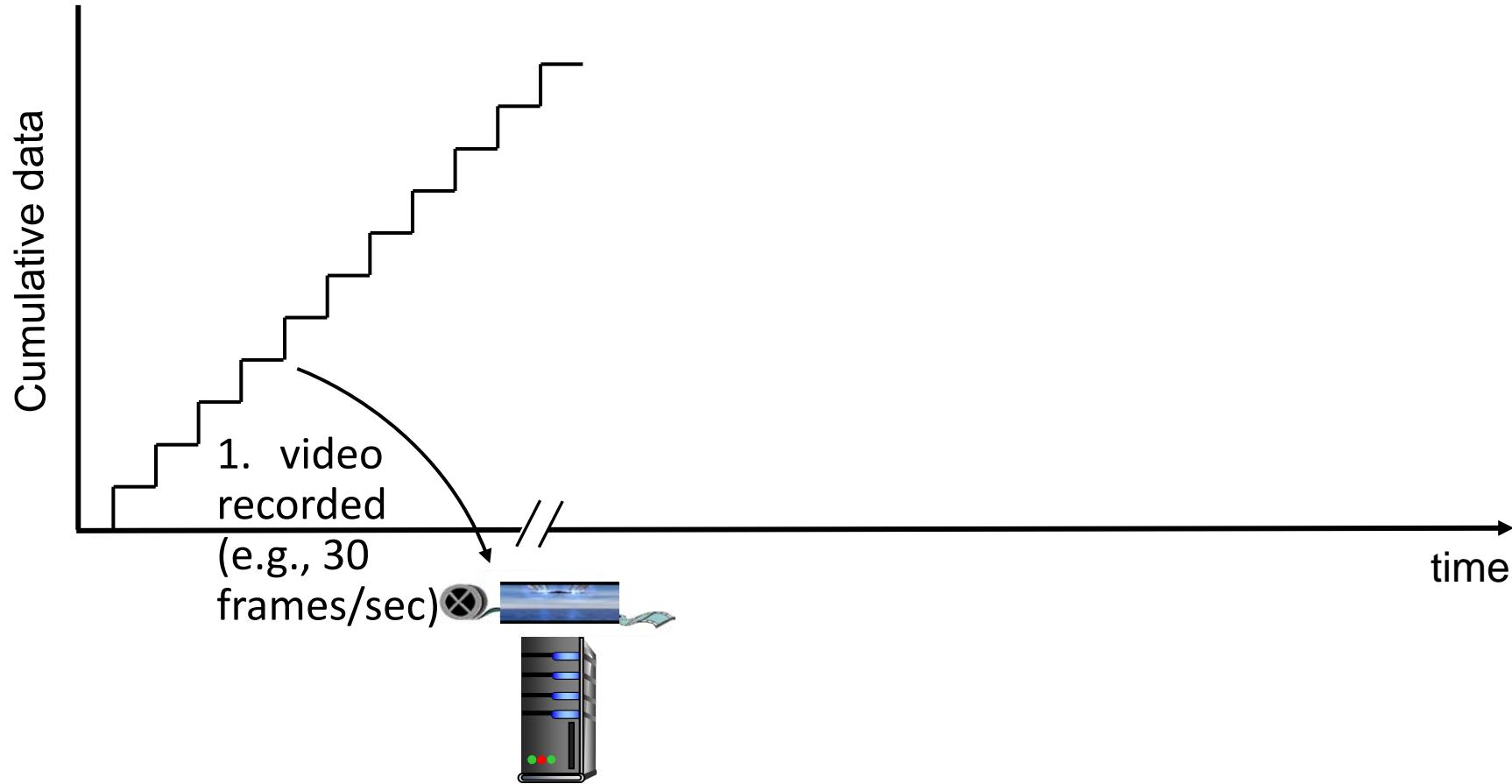
Main challenges:

- server-to-client bandwidth will *vary* over time, with changing network congestion levels (in house, access network, network core, video server)
- packet loss, delay due to congestion will delay playout, or result in poor video quality

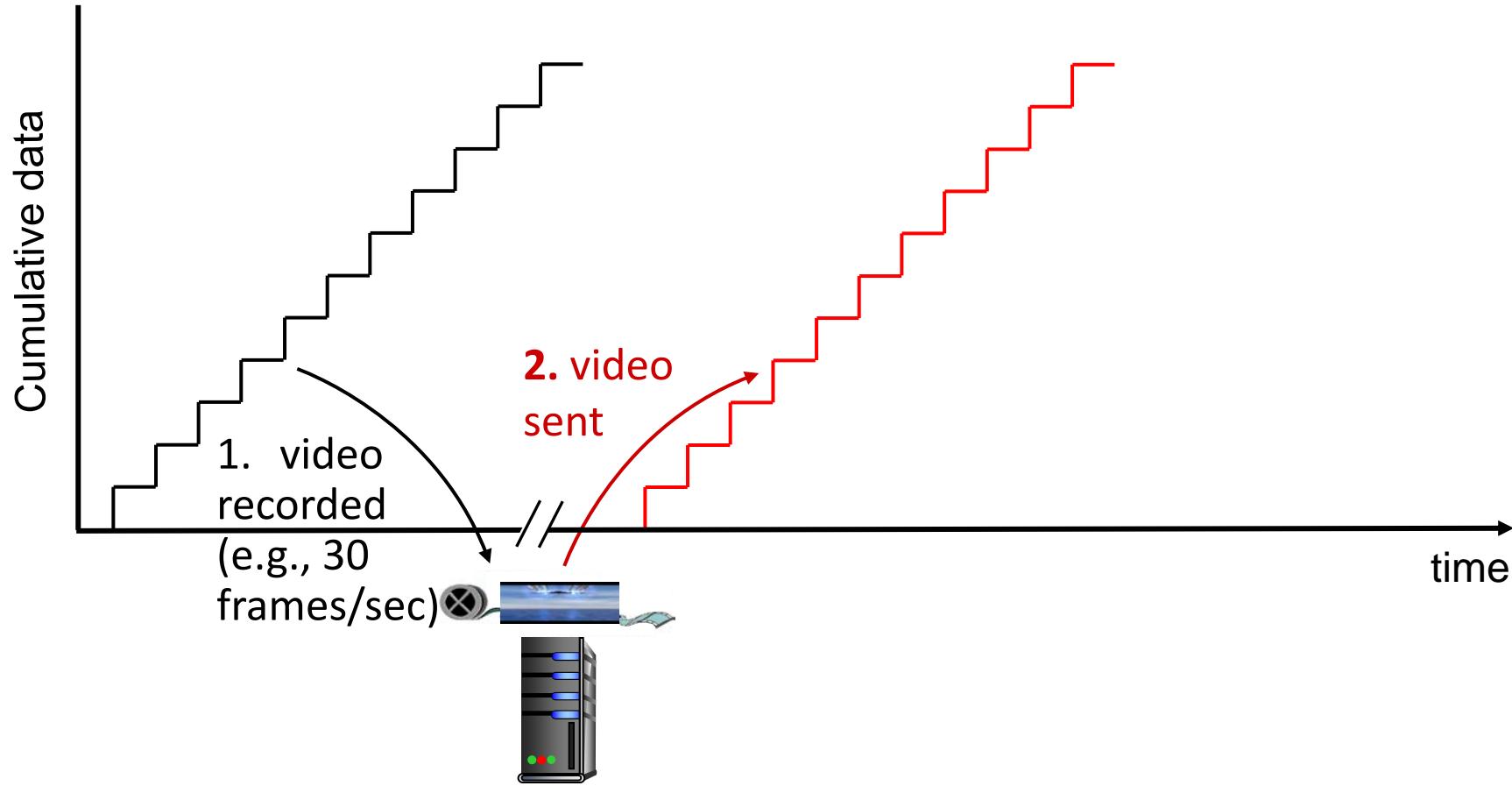
Streaming stored video



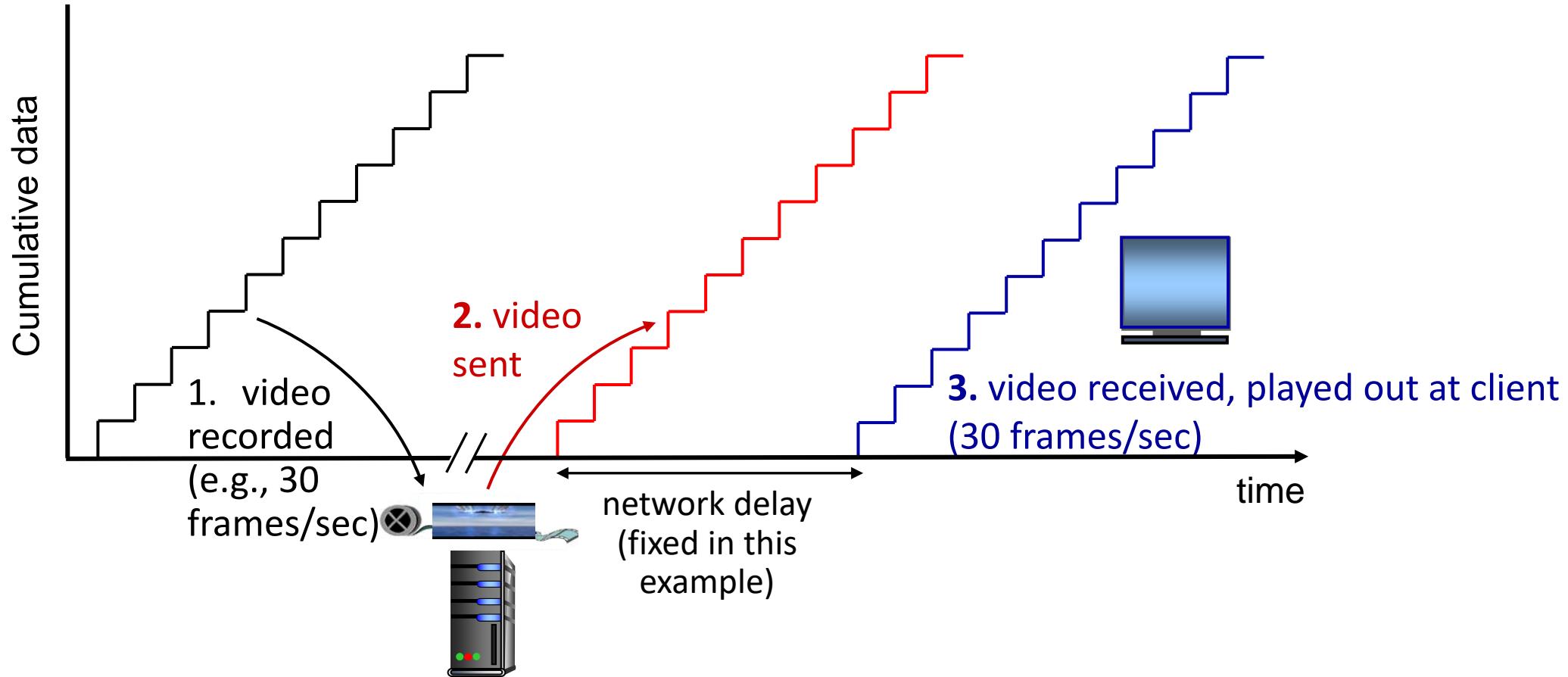
Streaming stored video



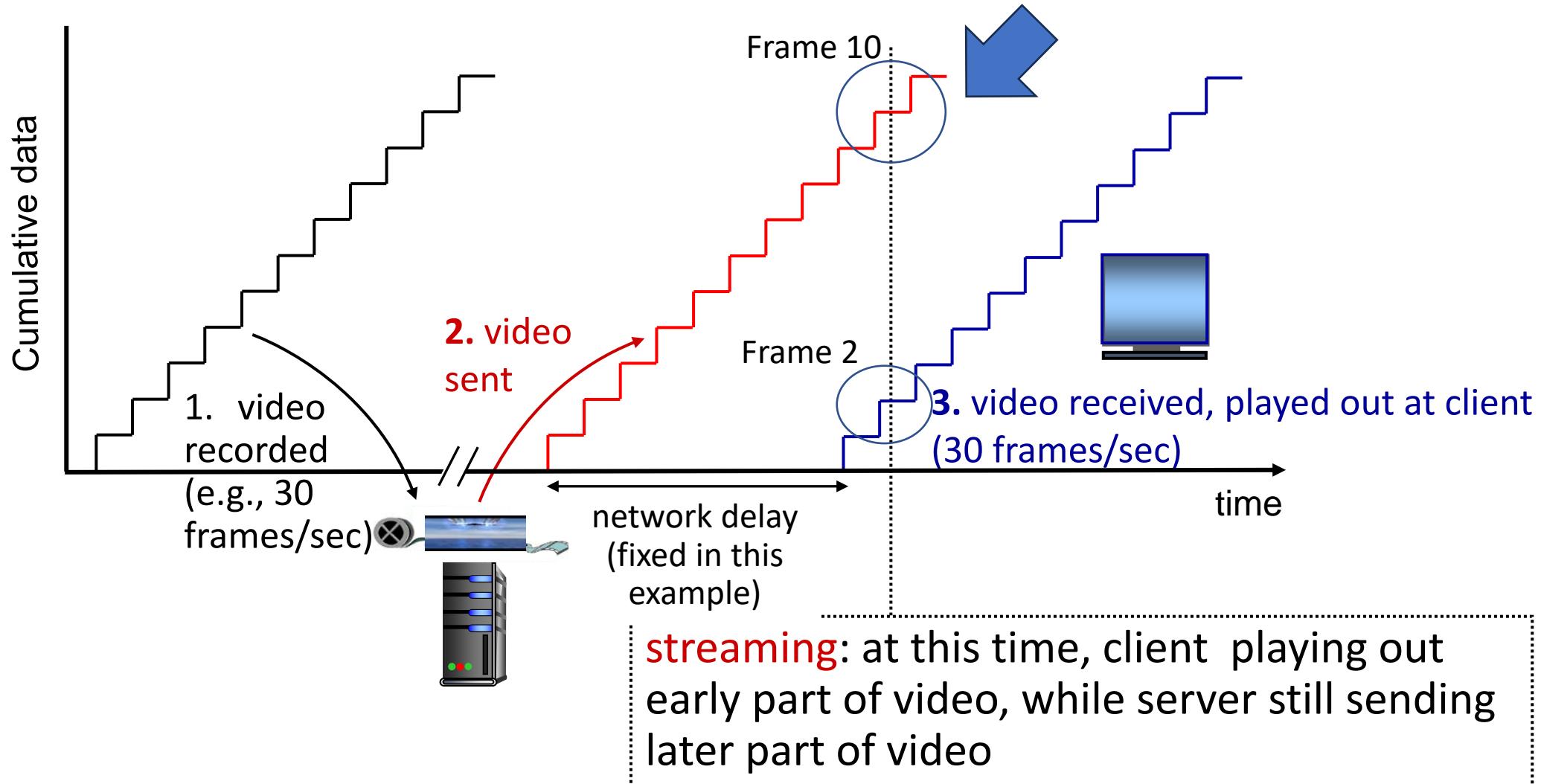
Streaming stored video



Streaming stored video

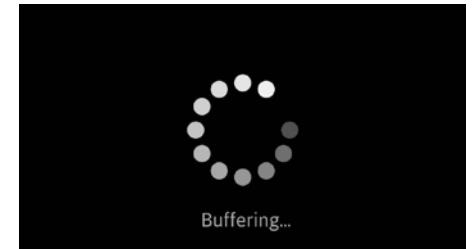


Streaming stored video



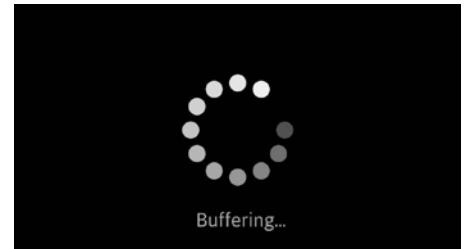
Streaming stored video: challenges

- **continuous playout constraint:** during client video playout, playout timing must match original timing
 - ... but **network delays are variable** (jitter), so will need **client-side buffer** to match continuous playout constraint

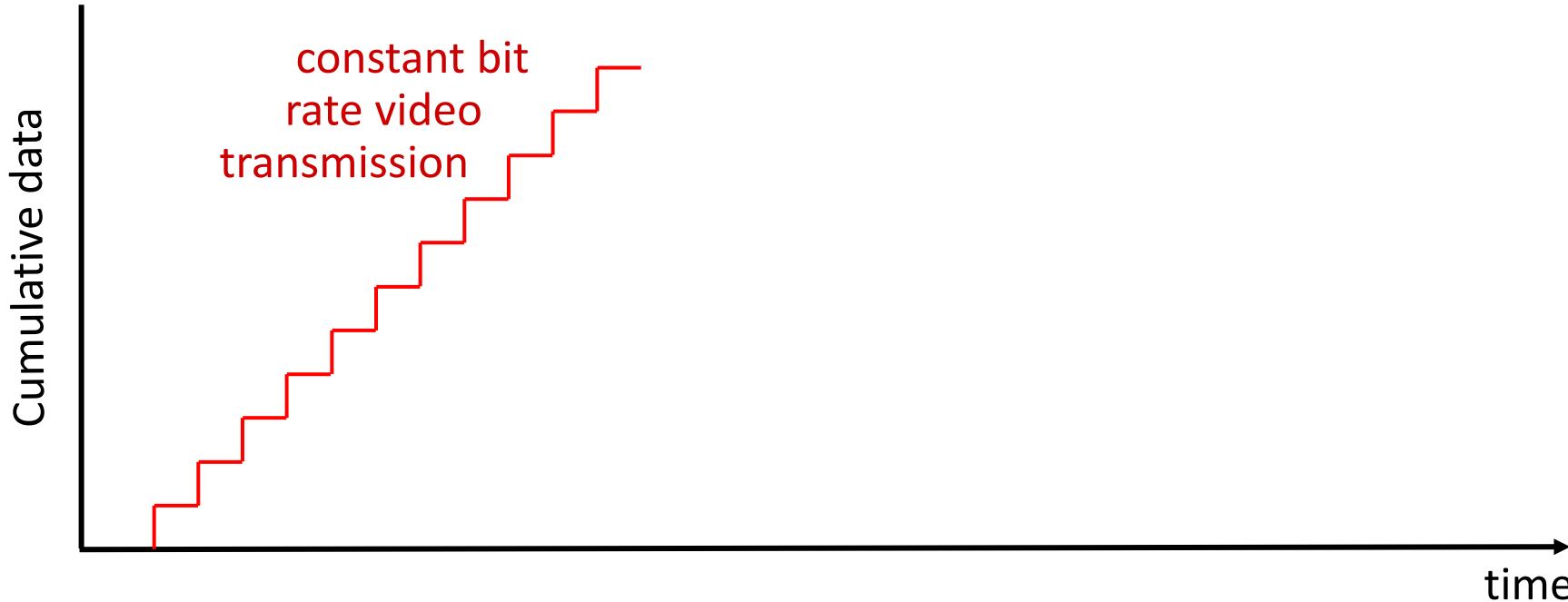


Streaming stored video: challenges

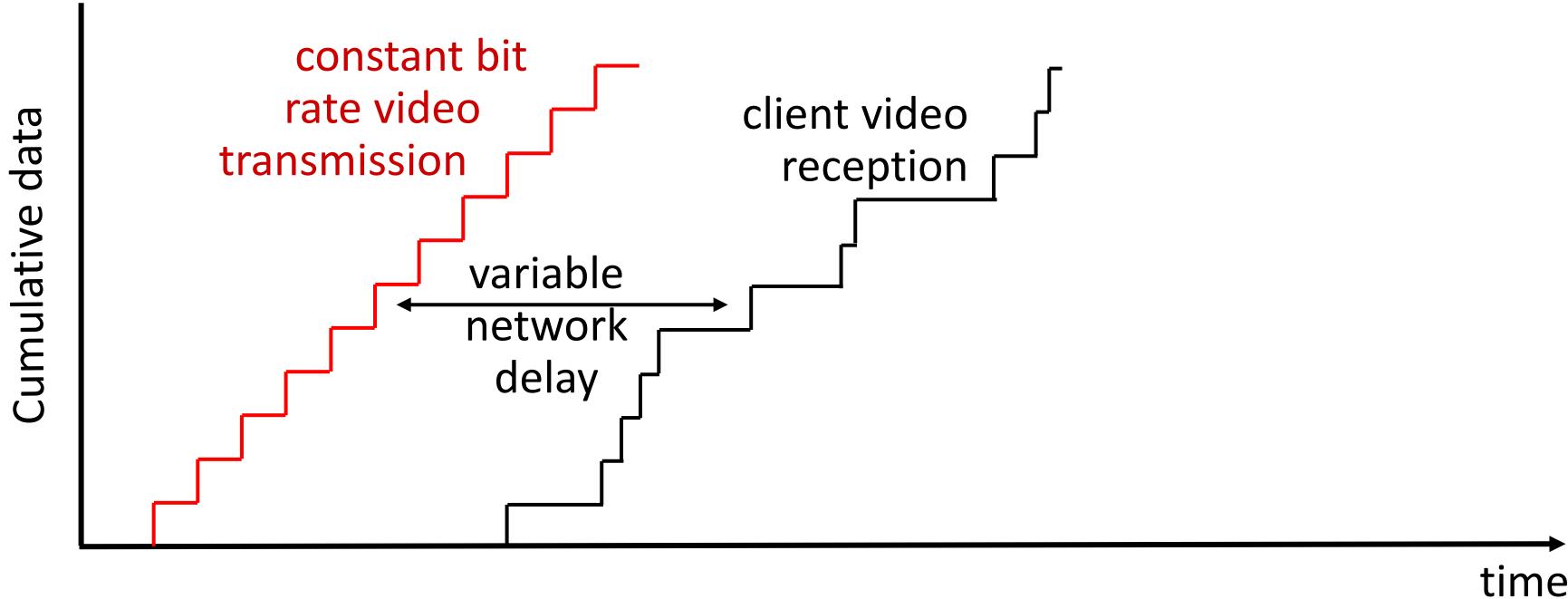
- **continuous playout constraint:** during client video playout, playout timing must match original timing
 - ... but **network delays are variable** (jitter), so will need **client-side buffer** to match continuous playout constraint
- other challenges:
 - client interactivity: pause, fast-forward, rewind, jump through video
 - video packets may be lost, retransmitted
 - variable network delays



Streaming stored video: playout buffering

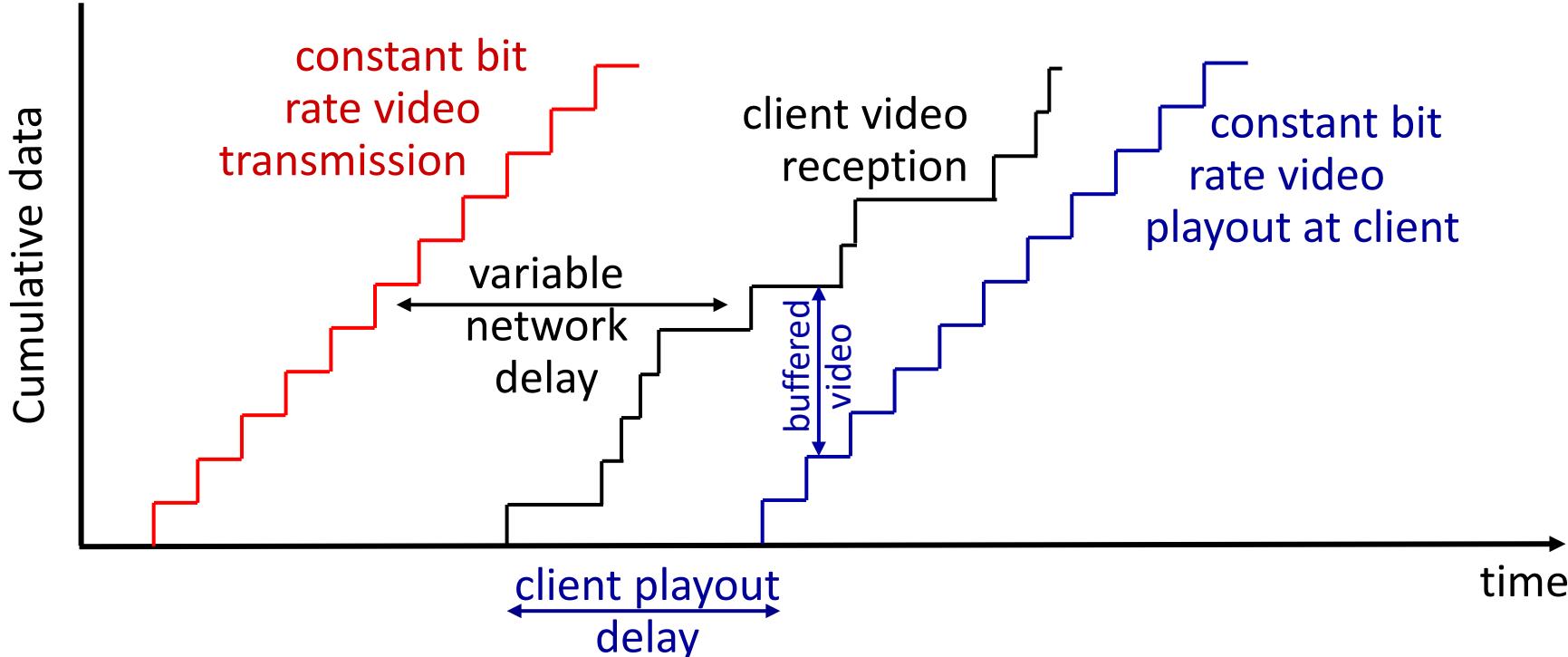


Streaming stored video: playout buffering



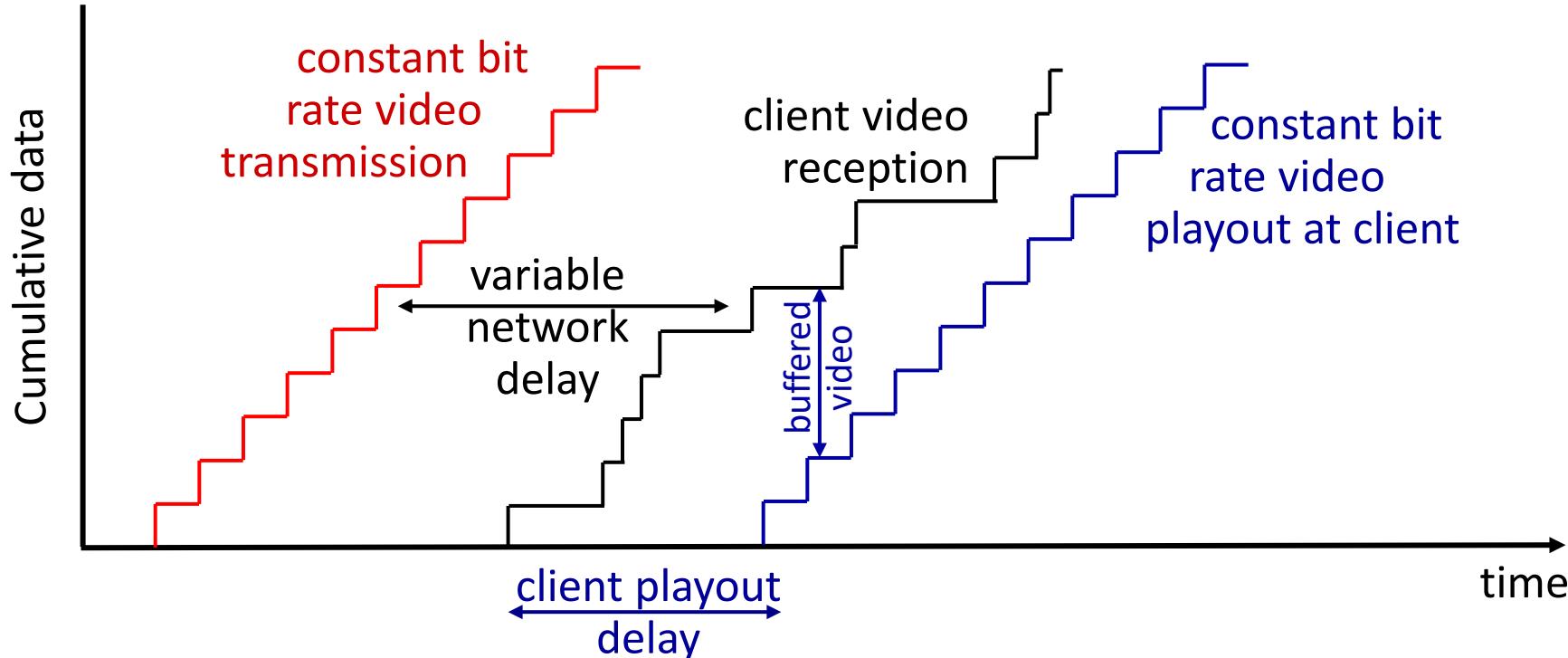
Variable network delay: frames are no longer received with a timing that matches the timing needed for playout.

Streaming stored video: playout buffering



- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

Streaming stored video: playout buffering



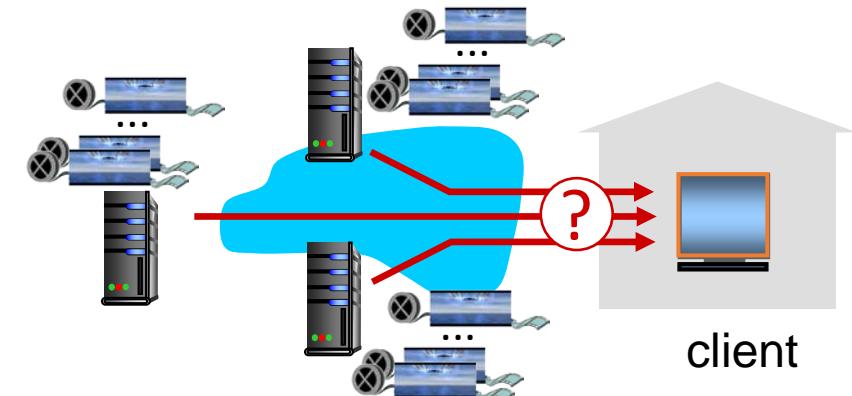
■ *Client playout delay dilemma*

- *Too short: starvation (video freezes)*
- *Too long: bad perceived performance (users do not like to wait)*

Streaming multimedia: DASH

*Dynamic, Adaptive
Streaming over HTTP*

What happens when the amount of available bandwidth, that exists between the client and the server, just isn't enough to support the rate at which video is being transmitted from client to server?

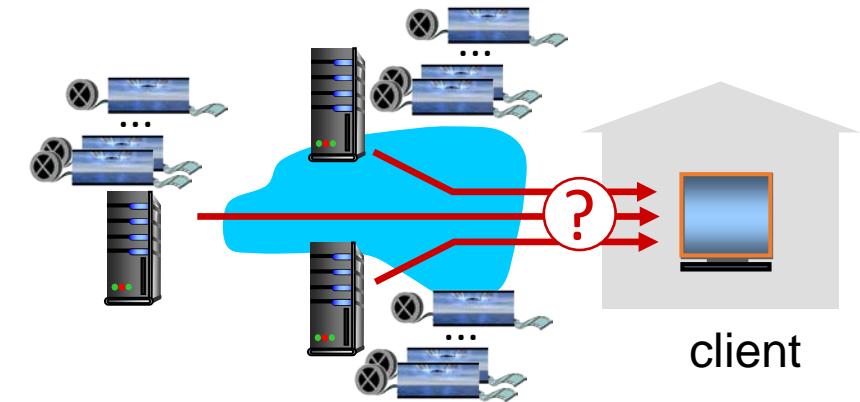


Streaming multimedia: DASH

*D*ynamic, *A*daptive
*S*treaming over *H*TTP

server:

- divides video file into multiple chunks
- each chunk encoded at multiple different rates
- different rate encodings stored in different files

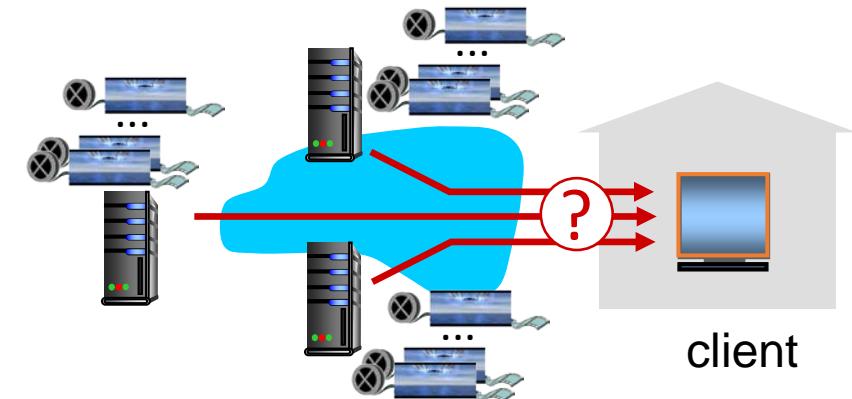


Streaming multimedia: DASH

*Dynamic, Adaptive
Streaming over HTTP*

server:

- divides video file into multiple chunks
- each chunk encoded at multiple different rates
- different rate encodings stored in different files
- files replicated in various nodes
 - (Content Distribution Network CDN)
- *manifest file*: provides URLs for different chunks



Streaming multimedia: DASH

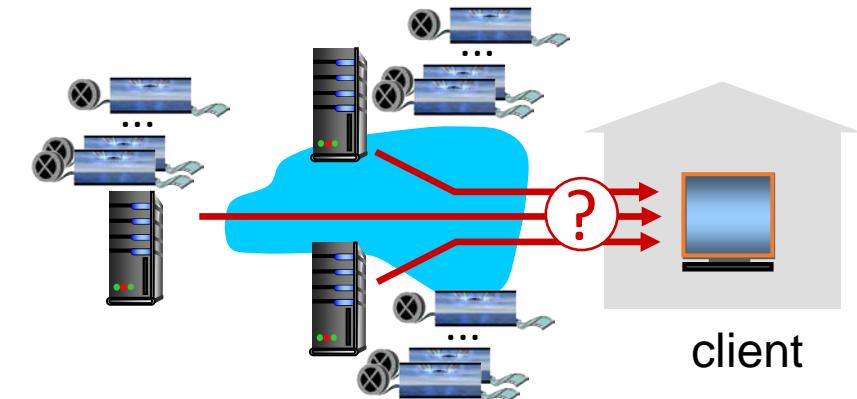
*Dynamic, Adaptive
Streaming over HTTP*

server:

- divides video file into multiple chunks
- each chunk encoded at multiple different rates
- different rate encodings stored in different files
- files replicated in various CDN nodes
- *manifest file*: provides URLs for different chunks

client:

- periodically estimates server-to-client bandwidth
- consulting manifest, requests one chunk at a time

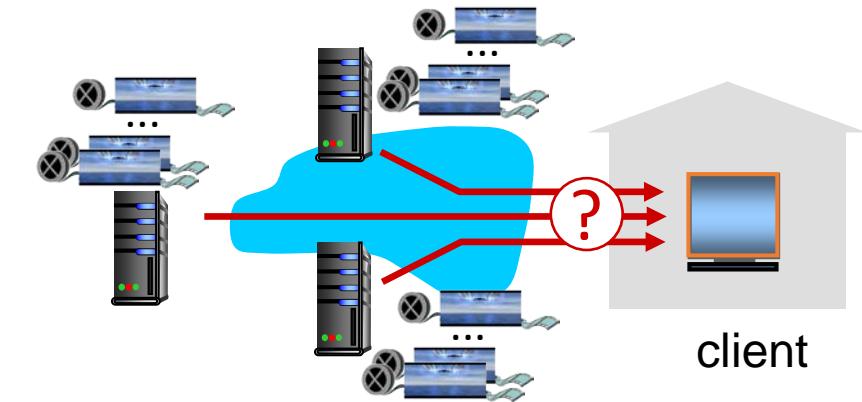


Streaming multimedia: DASH

Dynamic, Adaptive
Streaming over HTTP

server:

- divides video file into multiple chunks
- each chunk encoded at multiple different rates
- different rate encodings stored in different files
- files replicated in various CDN nodes
- *manifest file*: provides URLs for different chunks

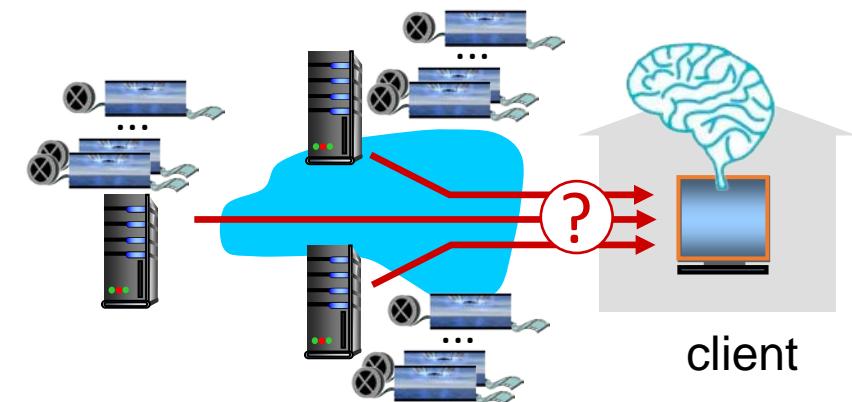


client:

- periodically estimates 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 at different points in time (depending on available bandwidth at time), and from different servers

Streaming multimedia: DASH

- “*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)



Streaming video = encoding + DASH + playout buffering

Content distribution networks (CDNs)

challenge: how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?

Content distribution networks (CDNs)

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 (and possibly congested) path to distant clients

....quite simply: this solution *doesn't scale*

Content distribution networks (CDNs)

challenge: how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?

- *option 2:* store/serve multiple copies of videos at multiple geographically distributed sites (*CDN*)
 - *Severs loaded with content*
 - *Manifest file point a client to the requested content*

Content distribution networks (CDNs)

challenge: how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?

- *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
 - Akamai: 240,000 servers deployed in > 120 countries (2015)



Content distribution networks (CDNs)

challenge: how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?

- *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
 - Akamai: 240,000 servers deployed in > 120 countries (2015)
 - *bring home:* smaller number (10's) of larger clusters in POPs near access nets
 - used by Limelight

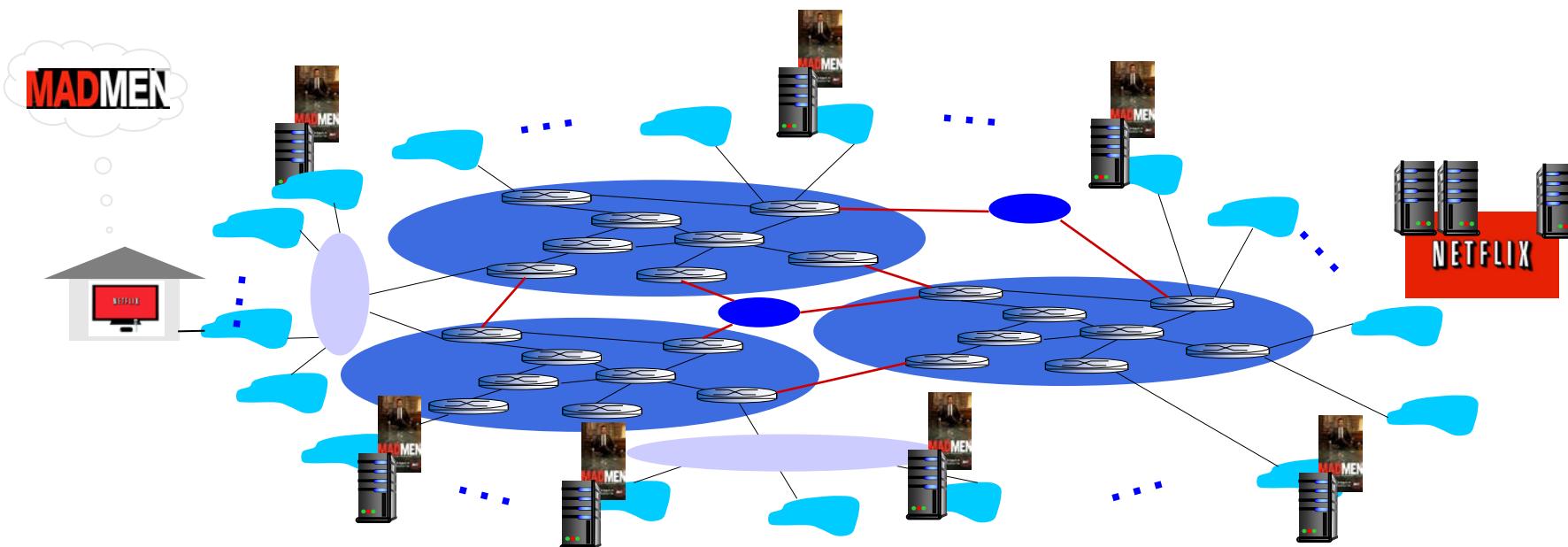


Video streaming - recap

Source: <https://www.youtube.com/watch?v=kCAXpAikMVc>

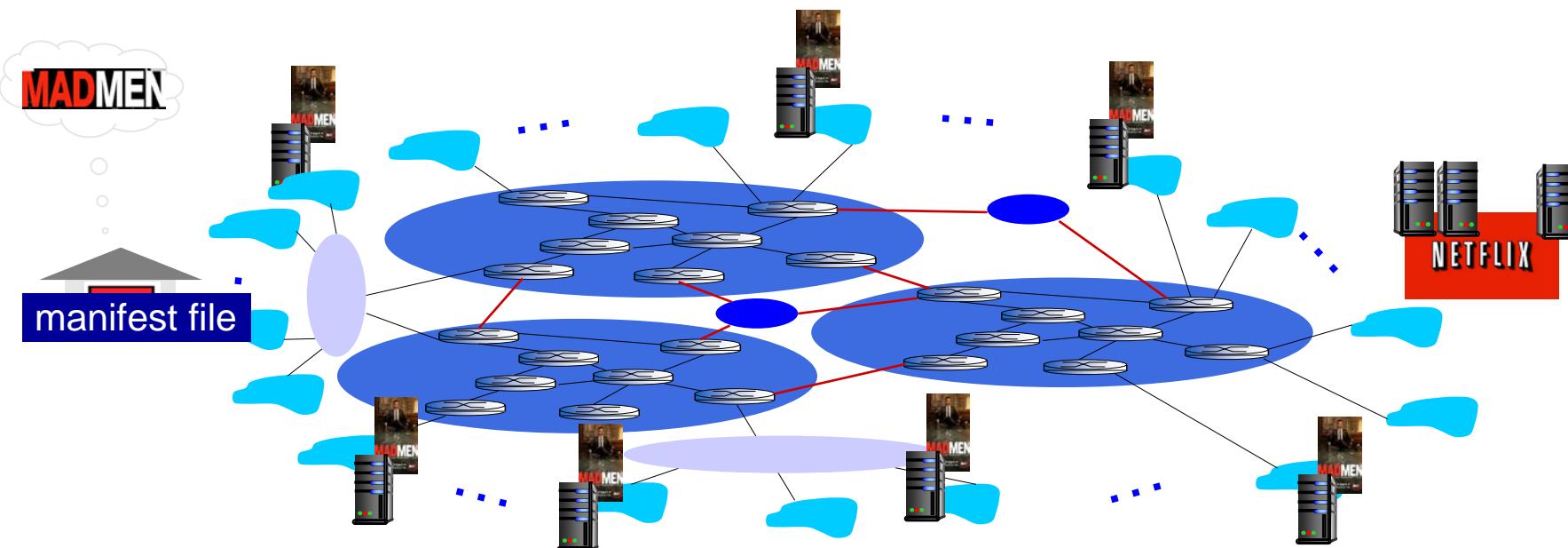
How does Netflix work?

- Netflix: stores copies of content (e.g., MADMEN) at its (worldwide) OpenConnect CDN nodes



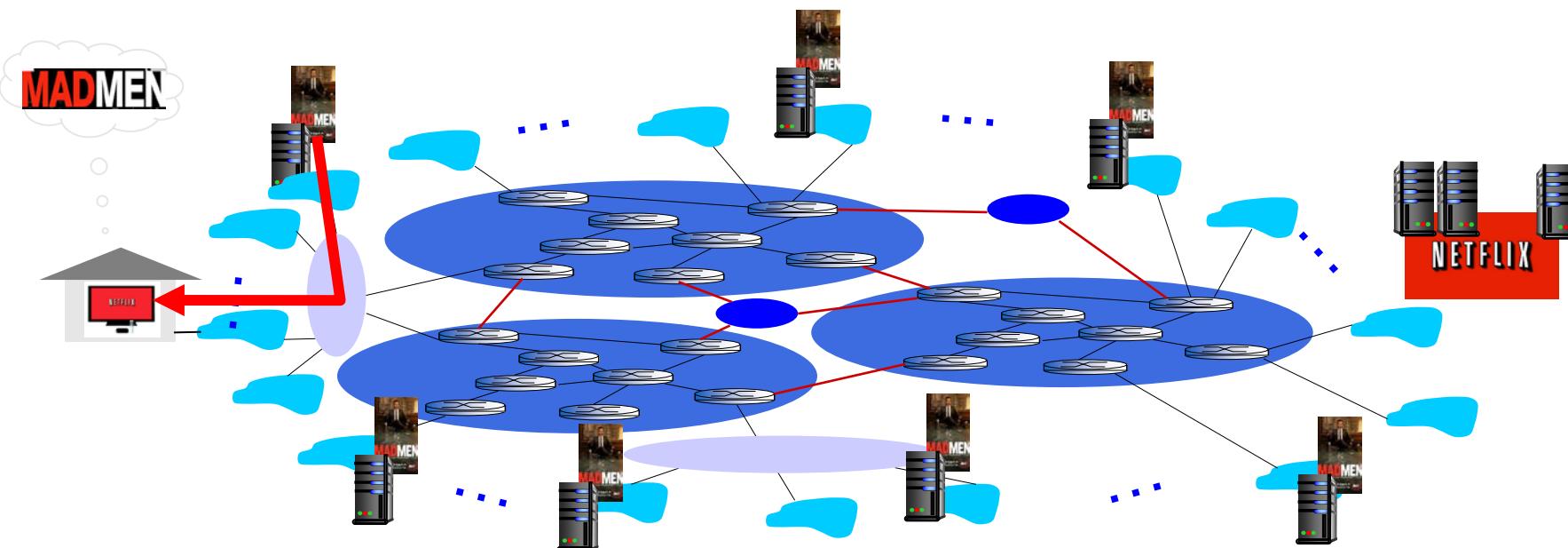
How does Netflix work?

- Netflix: stores copies of content (e.g., MADMEN) at its (worldwide) OpenConnect CDN nodes
- subscriber requests content, service provider returns manifest



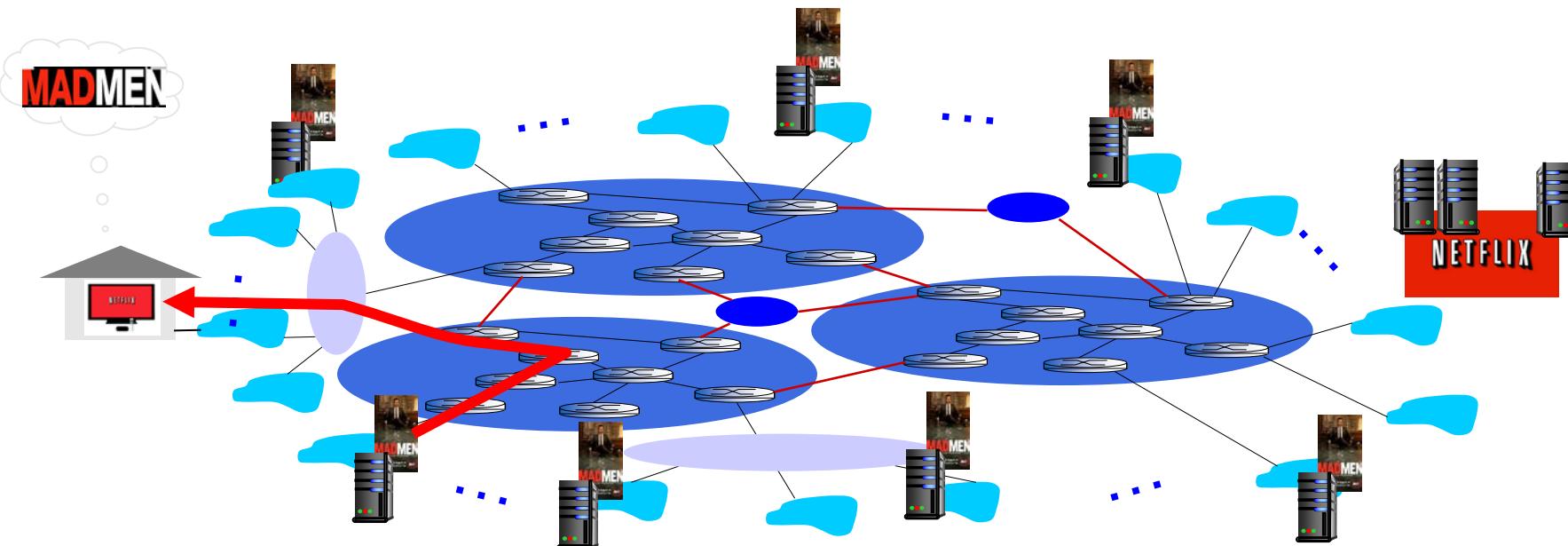
How does Netflix work?

- Netflix: stores copies of content (e.g., MADMEN) at its (worldwide) OpenConnect CDN nodes
- subscriber requests content, service provider returns manifest
 - using manifest, client retrieves content at highest supportable rate

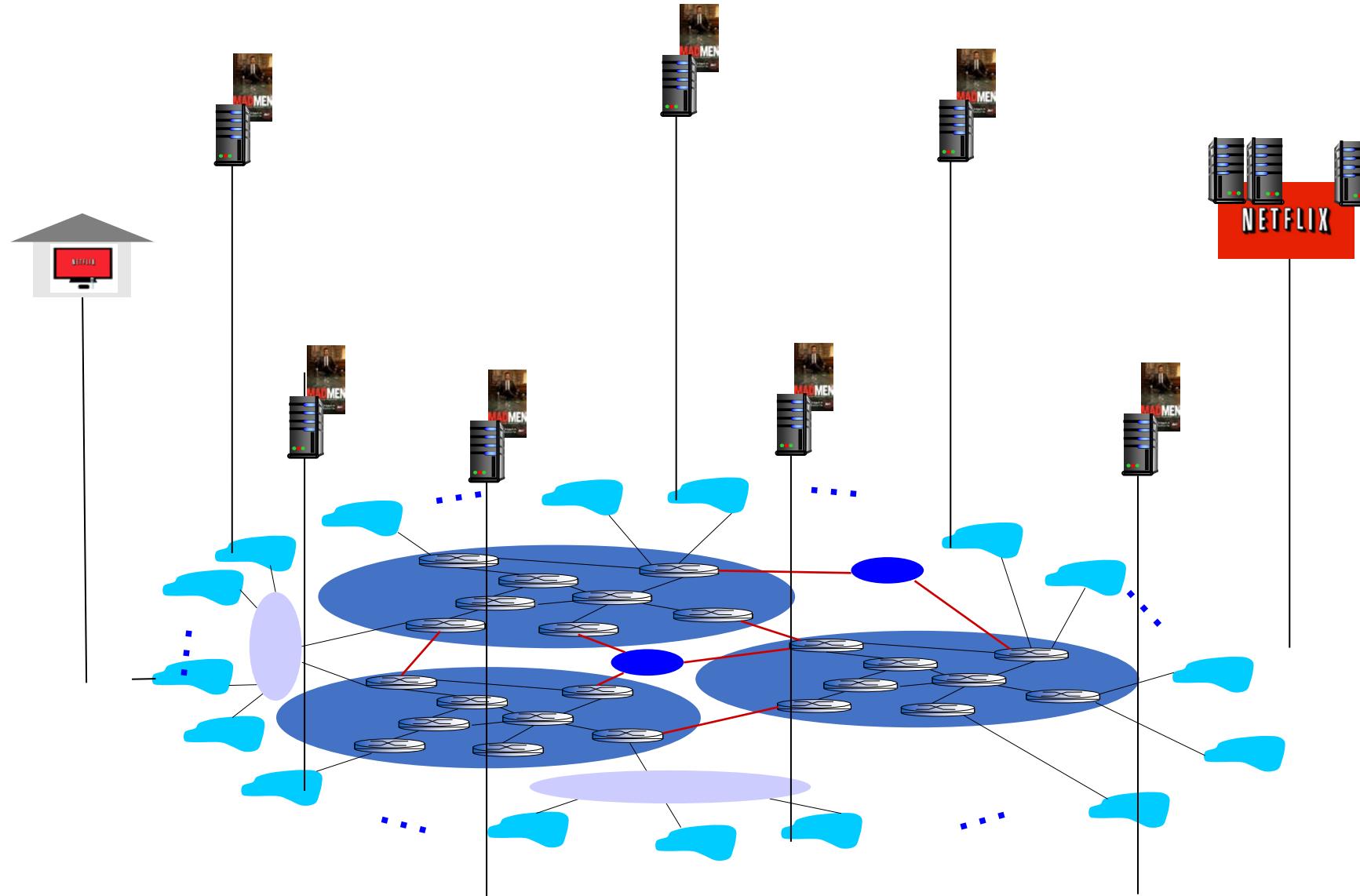


How does Netflix work?

- Netflix: stores copies of content (e.g., MADMEN) at its (worldwide) OpenConnect CDN nodes
- subscriber requests content, service provider returns manifest
 - using manifest, client retrieves content at highest supportable rate
 - may choose different rate or copy if network path congested

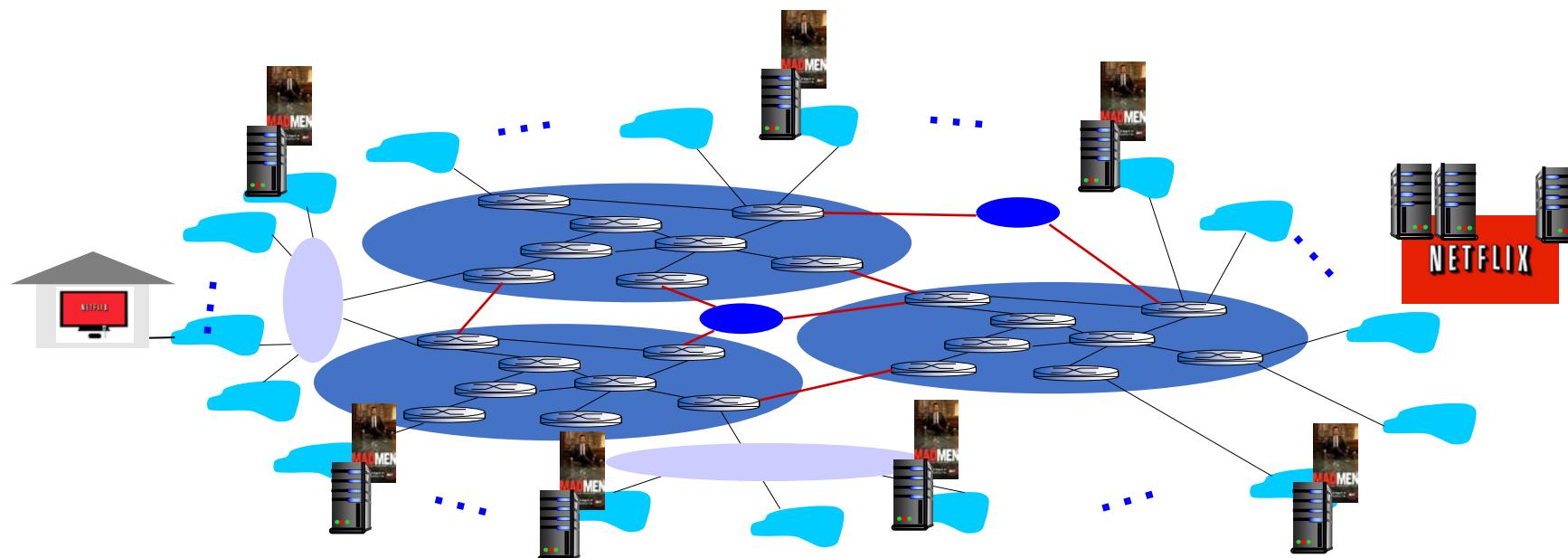


Content distribution networks (CDNs)



Content distribution networks (CDNs)

OTT: “over the top”



That's all for now

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
 - Application level service
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP

