

TLD: 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

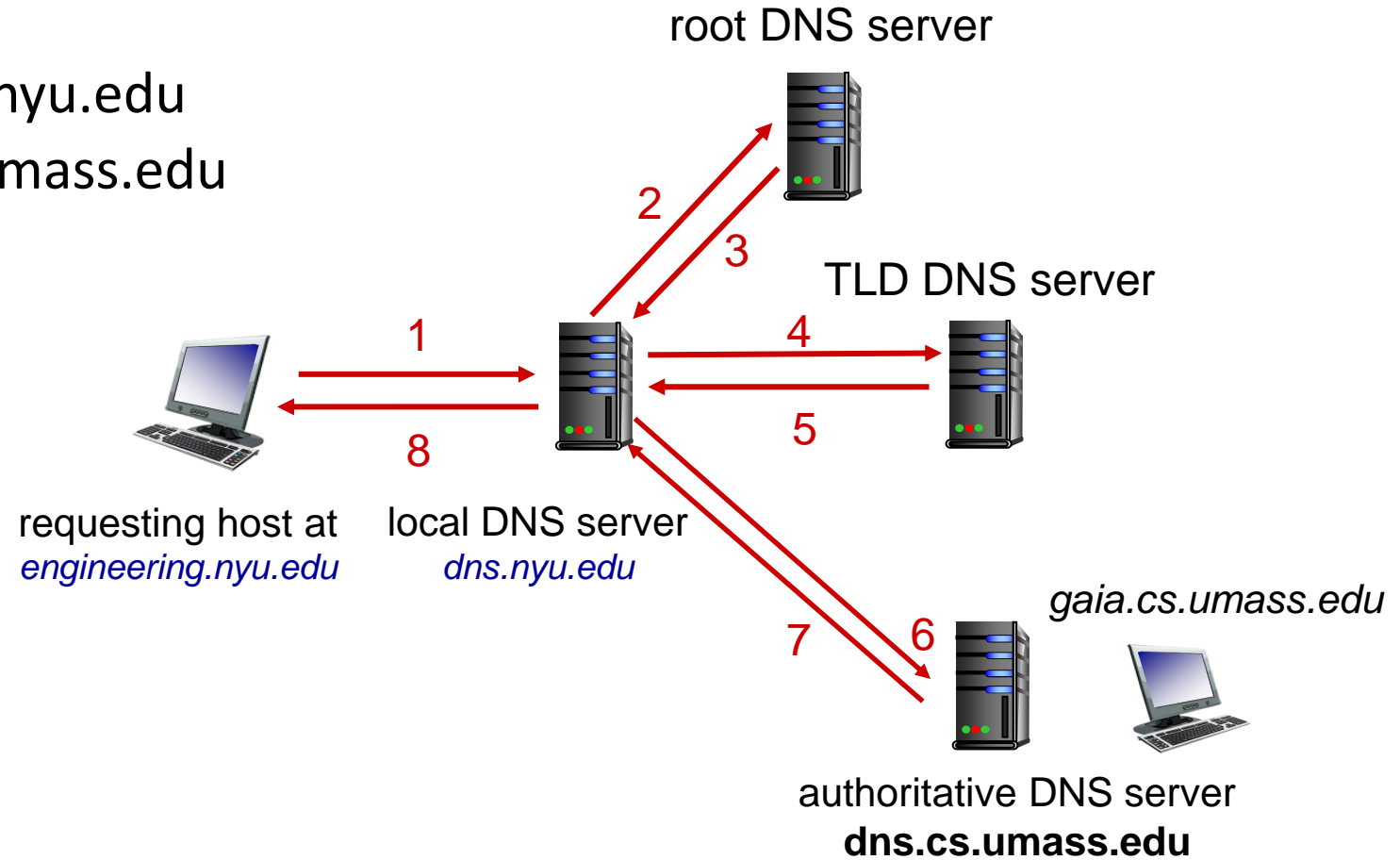
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
 - also called “default name server”
- when host makes DNS query, query is sent to its local DNS server
 - has local cache of recent name-to-address translation pairs (but may be out of date!)
 - acts as proxy, forwards query into hierarchy

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”

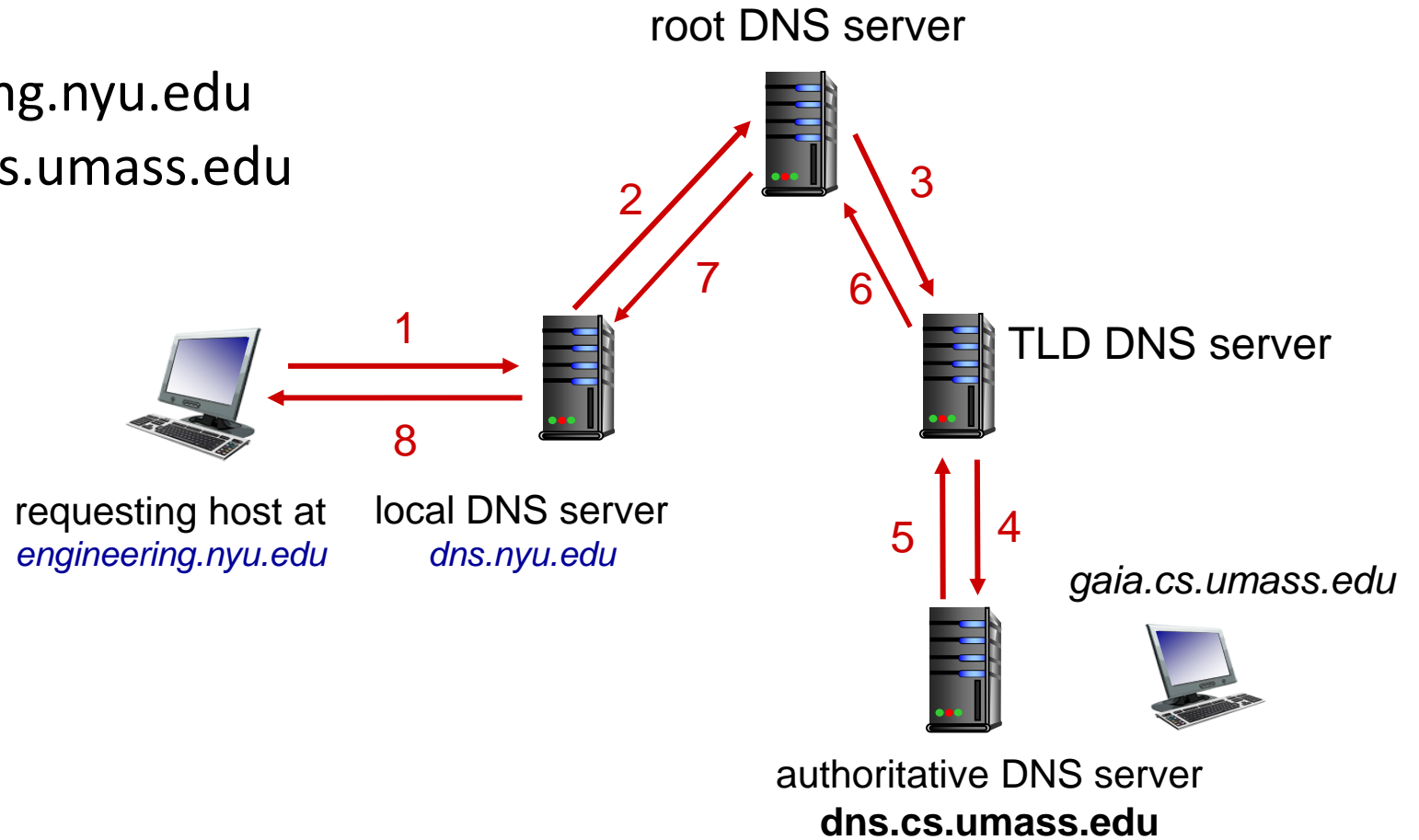


DNS name resolution: recursive query

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

Recursive query:

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



Caching, Updating DNS Records

- once (any) name server learns mapping, it *cached* mapping
 - cache entries timeout (disappear) after some time (TTL)
 - TLD servers typically cached in local name servers
 - thus root name servers not often visited
- cached entries may be *out-of-date* (best-effort name-to-address translation!)
 - if name host changes IP address, may not be known Internet-wide until all TTLs expire!
- update/notify mechanisms proposed IETF standard
 - RFC 2136

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

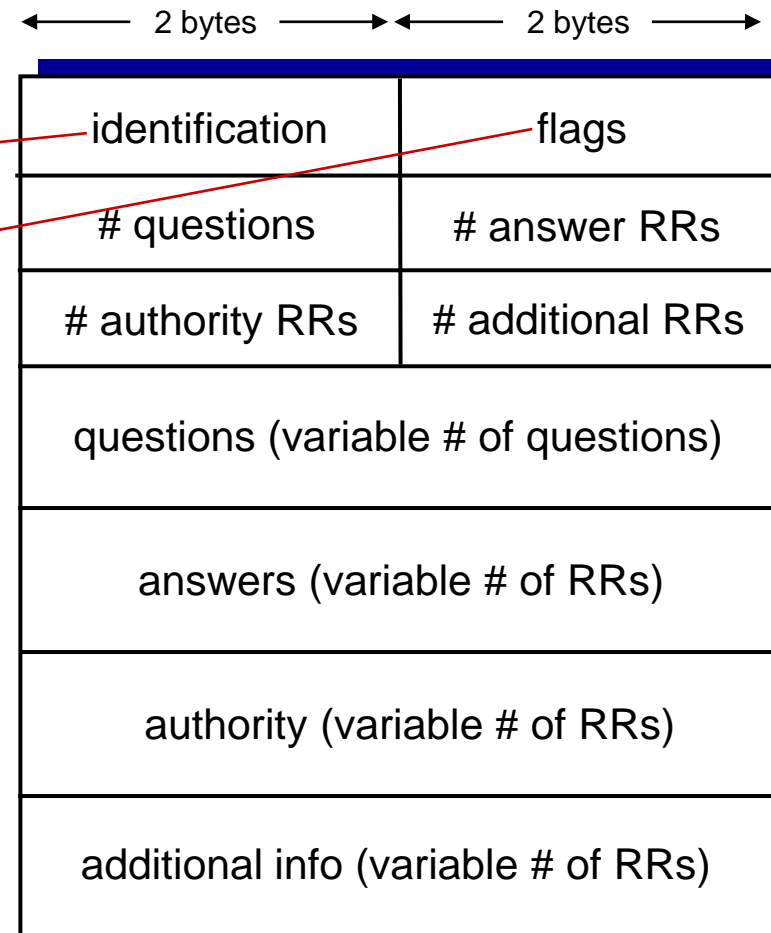
- value is name of mailserver associated with name

DNS protocol messages

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

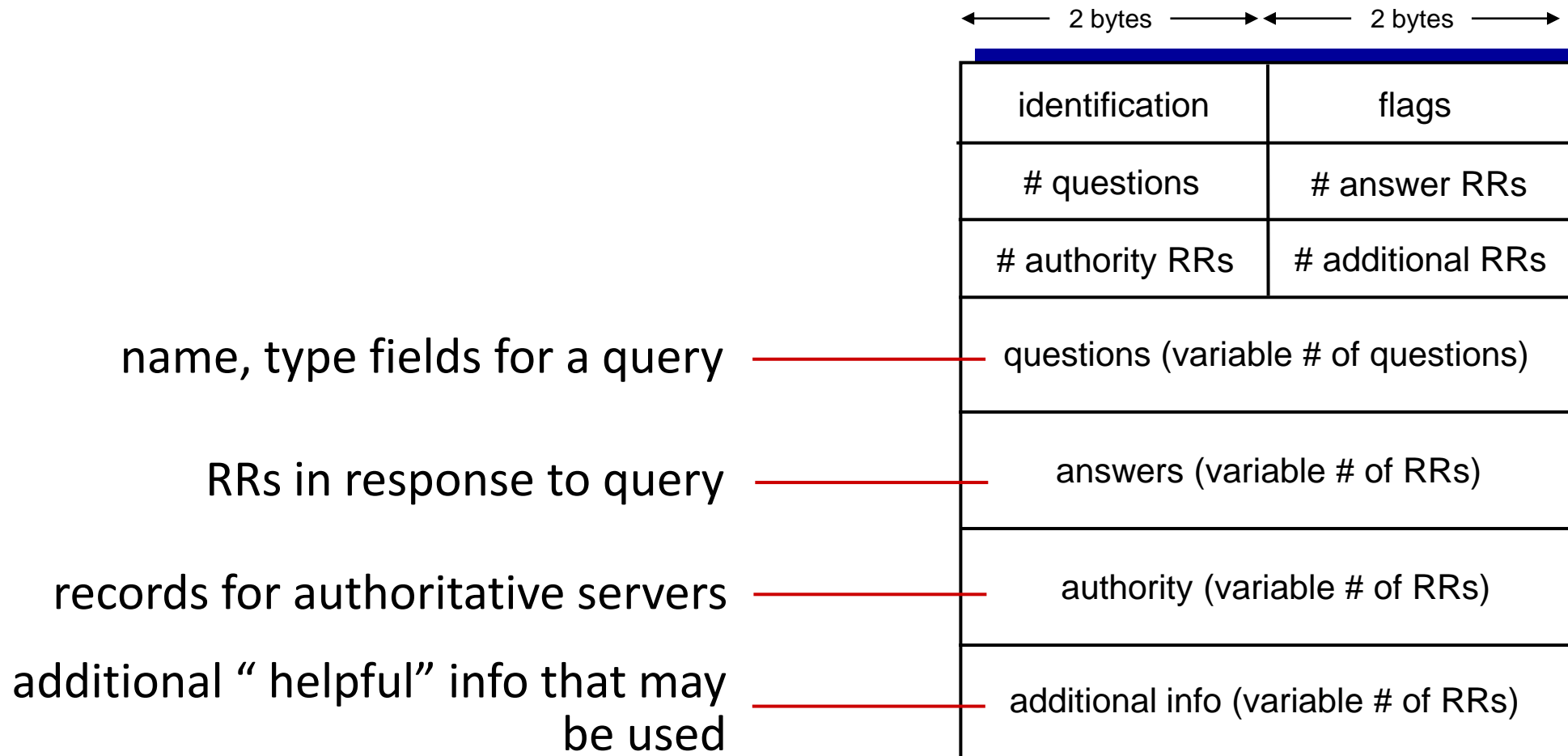
message header:

- **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*:



Inserting records into DNS

Example: new startup “Network Utopia”

- register name networkutopia.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.networkutopia.com`
 - type MX record for `networkutopia.com`

DNS security

DDoS attacks

- 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

Redirect attacks

- man-in-middle
 - intercept DNS queries
- DNS poisoning
 - send bogus replies to DNS server, which caches

Exploit DNS for DDoS

- send queries with spoofed source address: target IP
- requires amplification

DNSSEC
[RFC 4033]

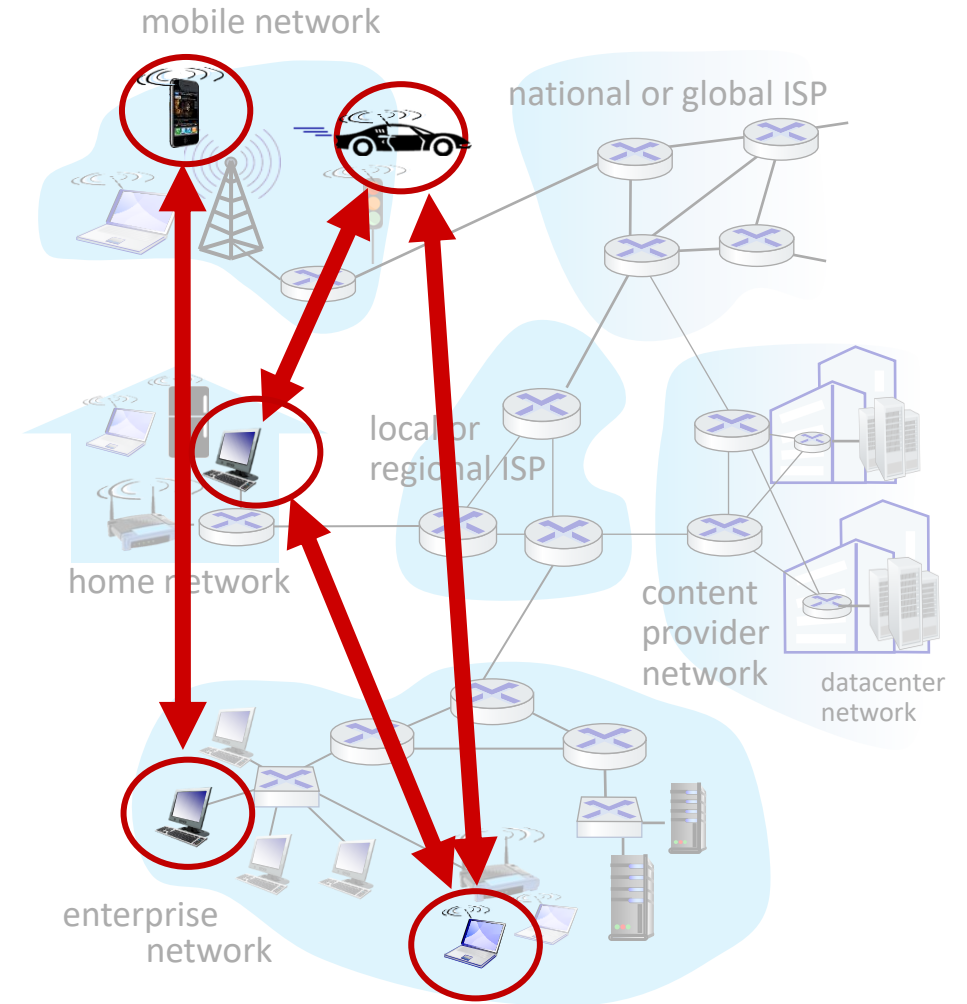
Application Layer: Overview

- 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

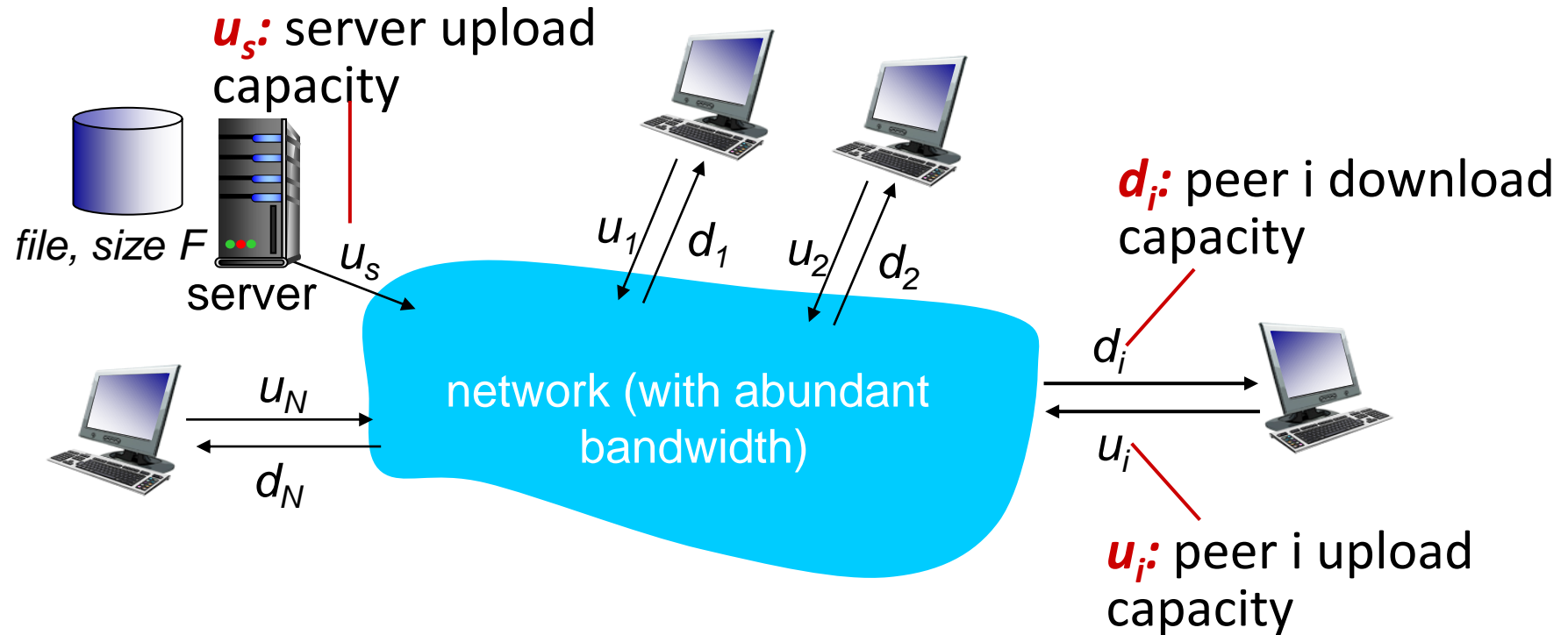
- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
 - *self scalability* – new peers bring new service capacity, and new service demands
- peers are intermittently connected and change IP addresses
 -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



File distribution: client-server vs P2P

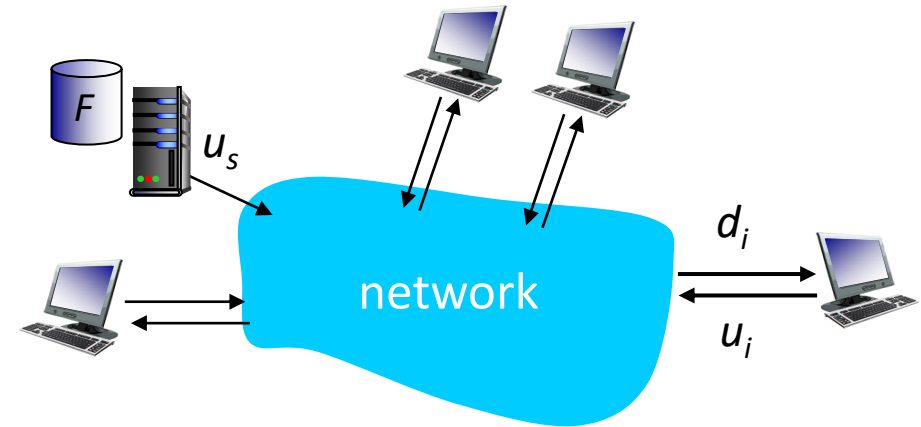
Q: how much time to distribute file (size F) from one server to N peers?

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



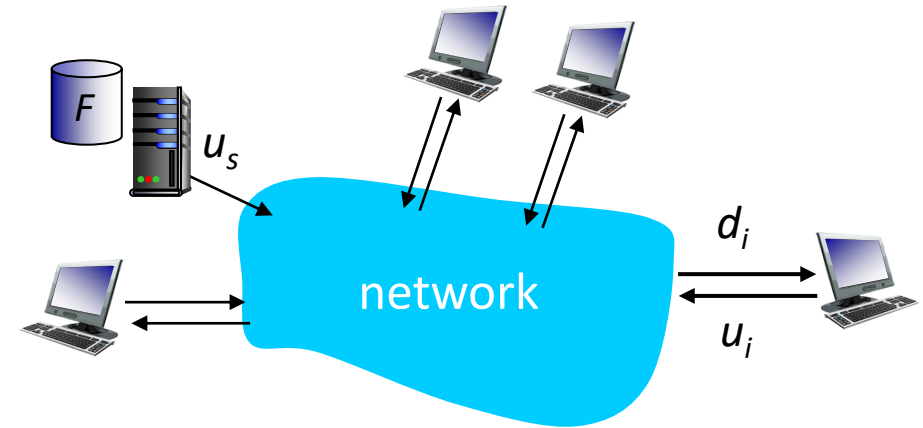
*time to distribute F
to N clients using
client-server approach*

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

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 $u_s + \sum u_i$



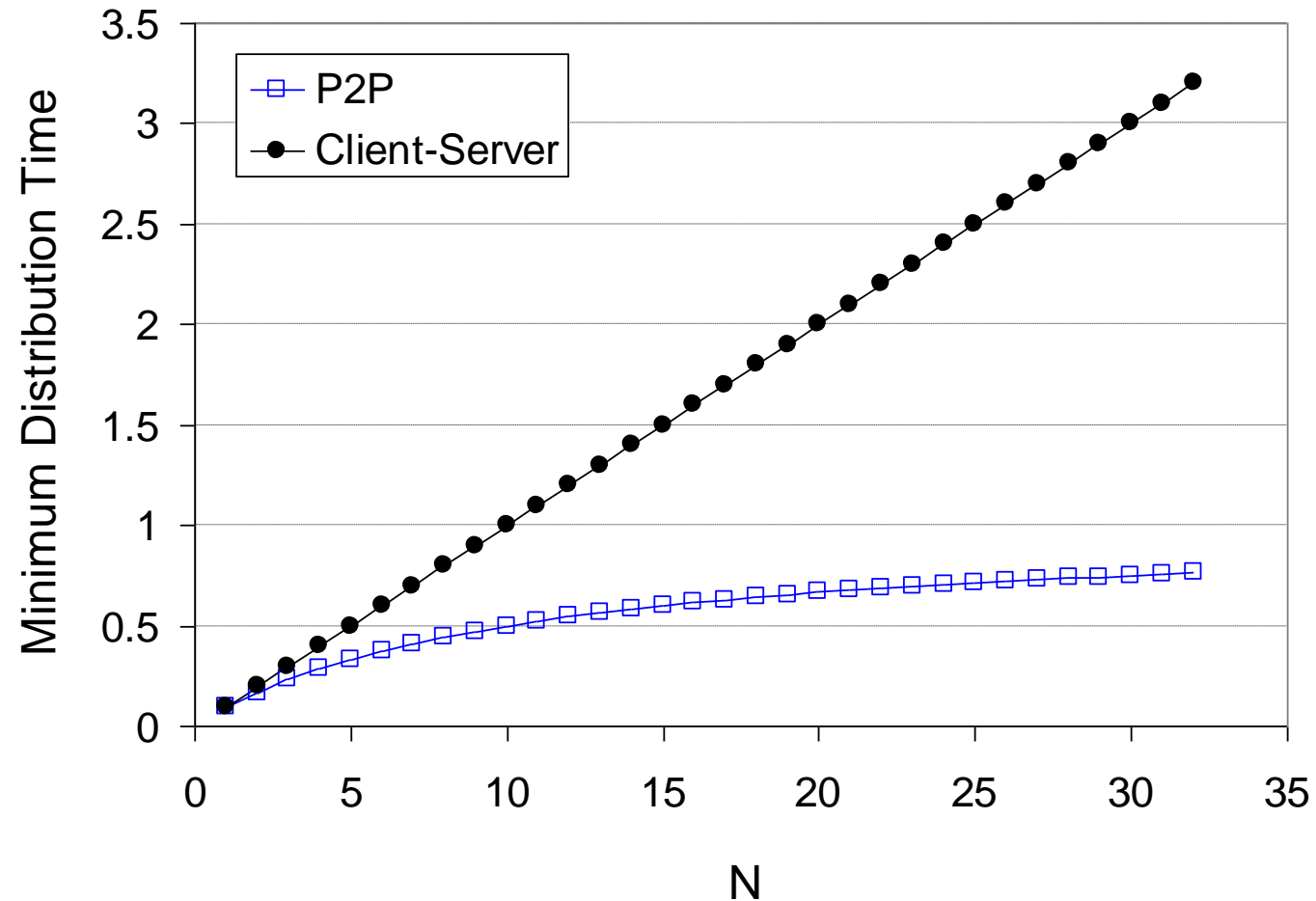
time to distribute F
to N clients using
P2P approach

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

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, $u_s = 10u$, $d_{min} \geq u_s$

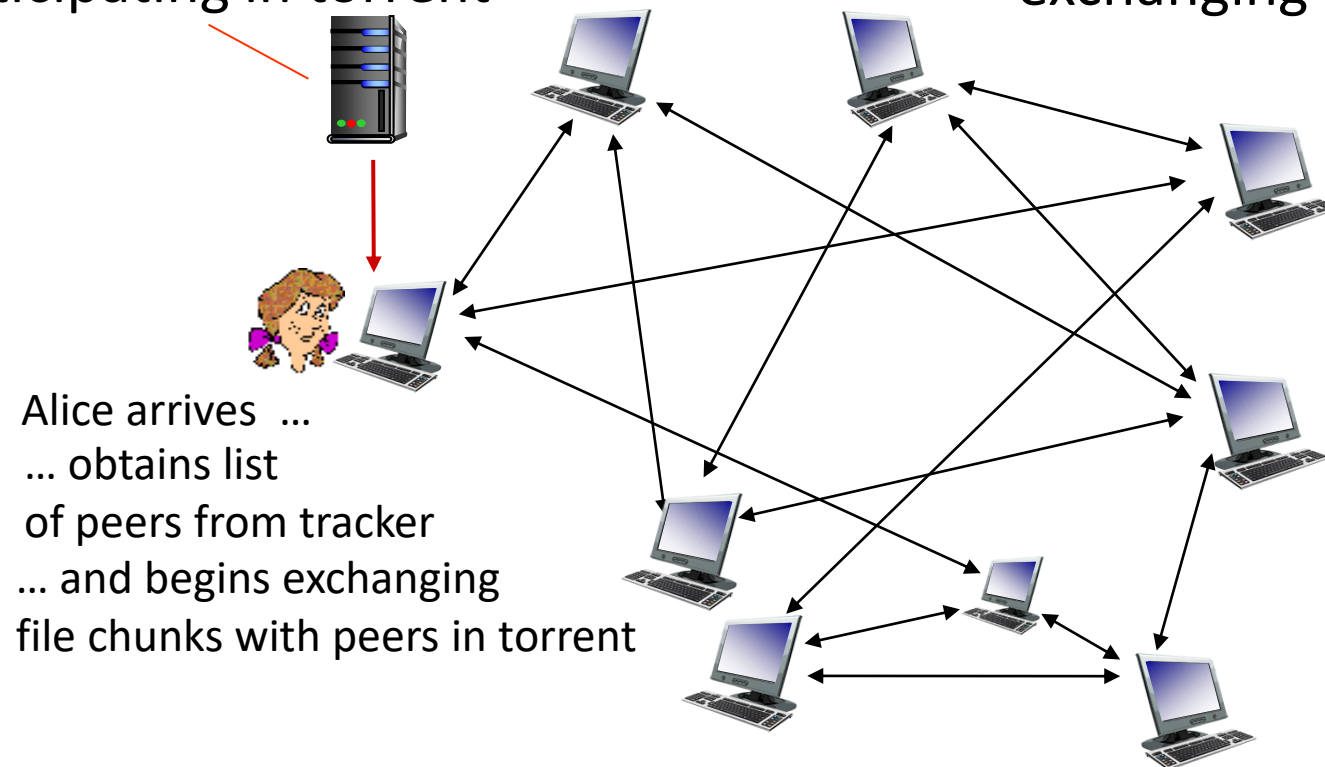


P2P file distribution: BitTorrent

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

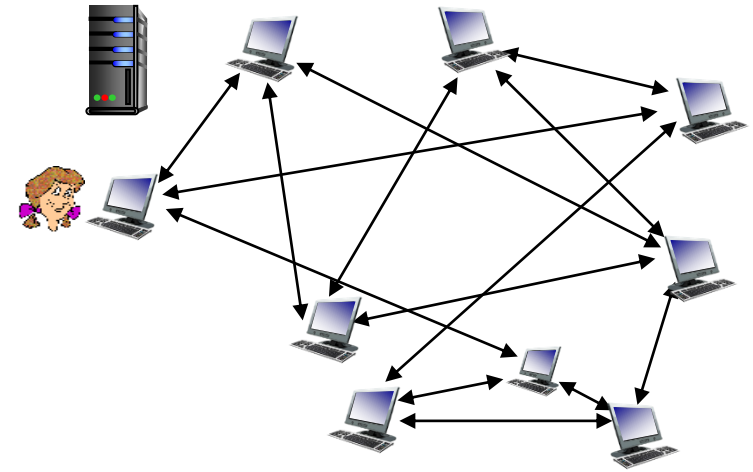
tracker: tracks peers participating in torrent

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
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent



BitTorrent: requesting, sending file chunks

Requesting chunks:

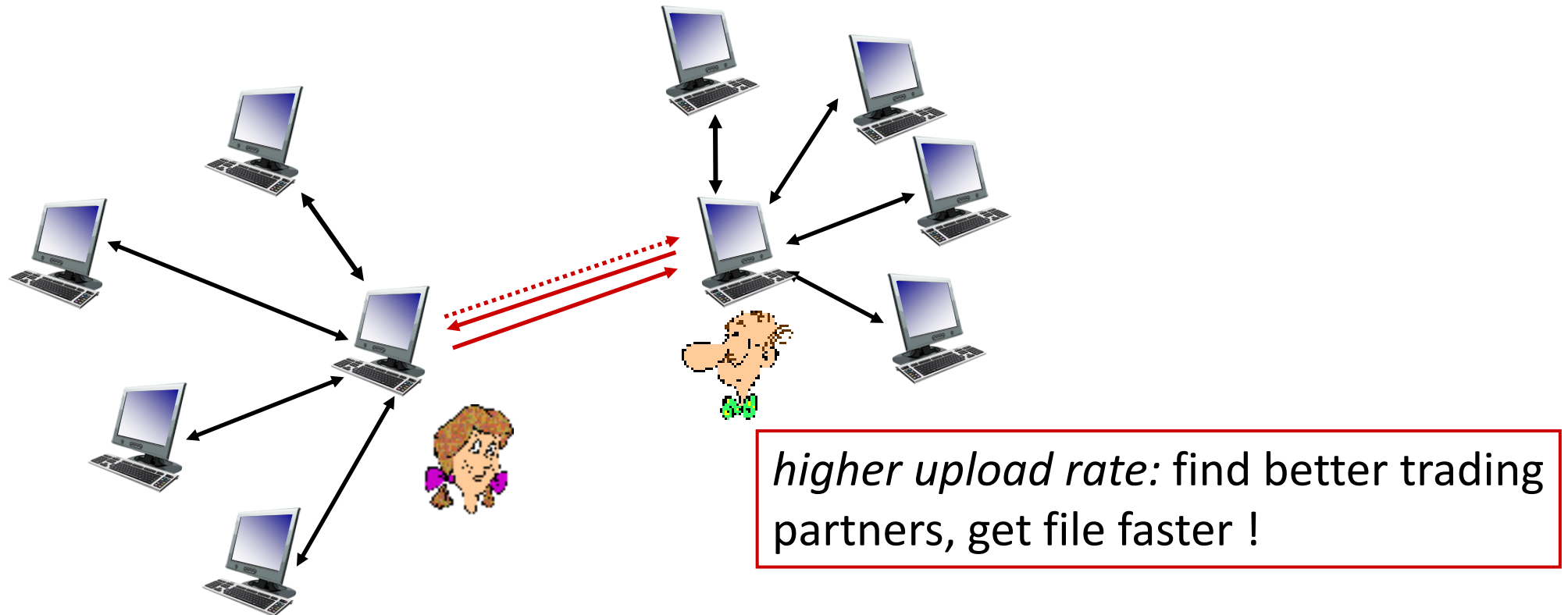
- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first

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

- (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



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: context

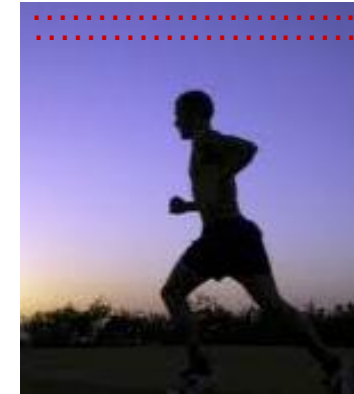
- stream video traffic: major consumer of Internet bandwidth
 - Netflix, YouTube, Amazon Prime: 80% of residential ISP traffic (2020)
- challenge: scale - how to reach ~1B users?
 - single mega-video server won't work (why?)
- challenge: heterogeneity
 - different users have different capabilities (e.g., 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
- 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

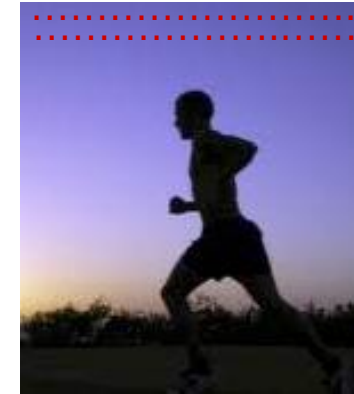


frame $i+1$

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

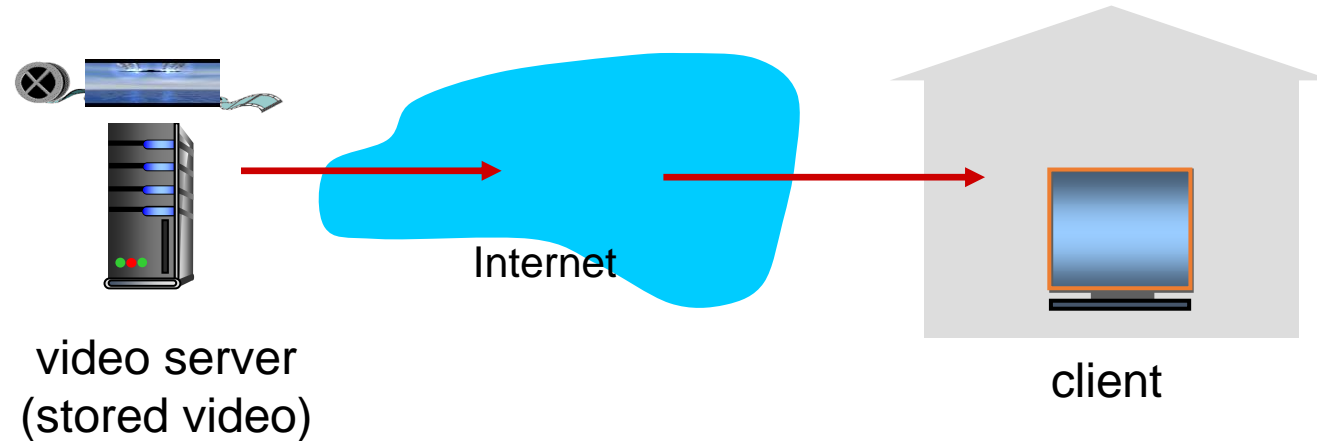
temporal coding example: instead of sending complete frame at $i+1$, send only differences from frame i



frame $i+1$

Streaming stored video

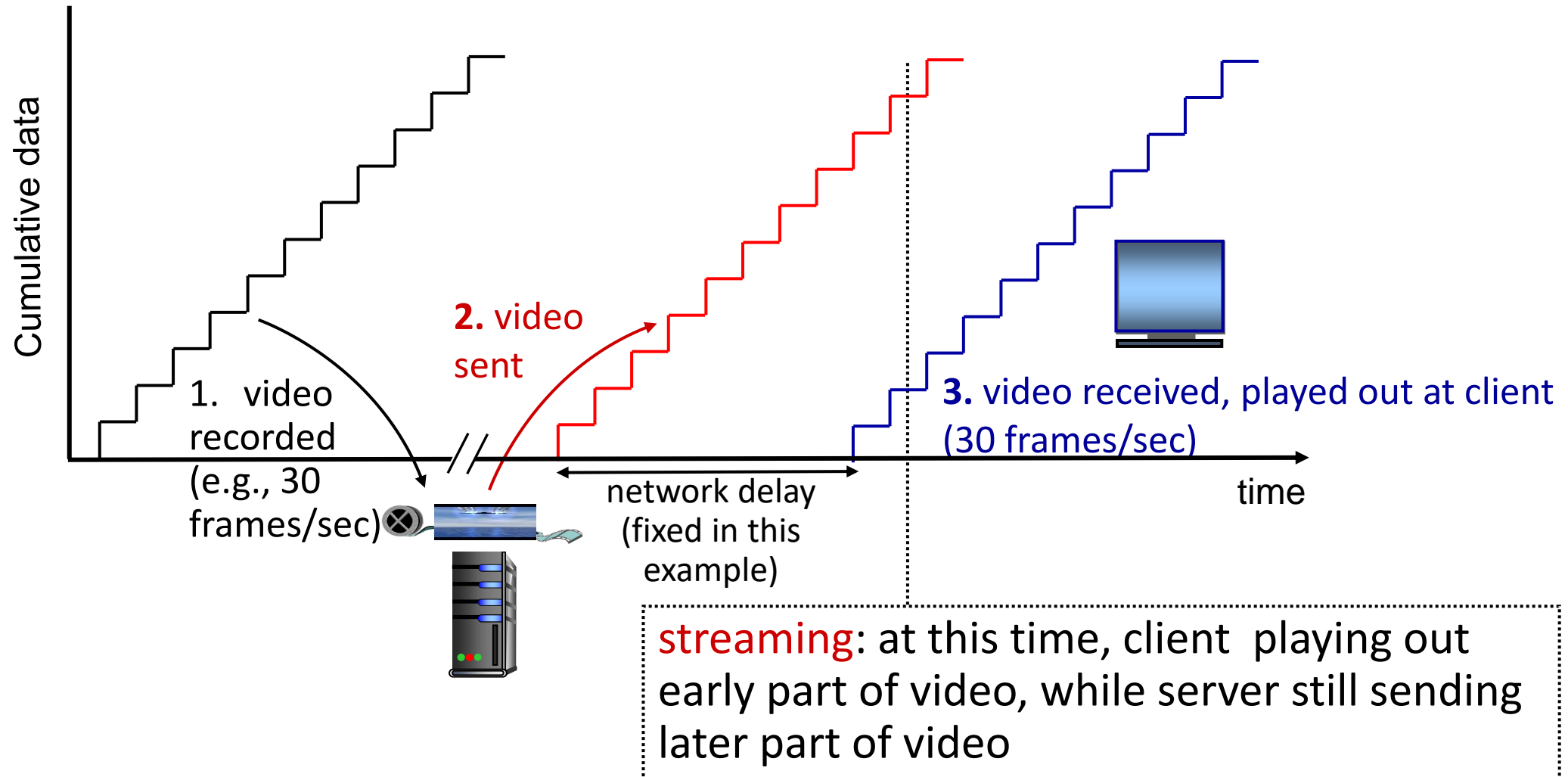
simple scenario:



Main challenges:

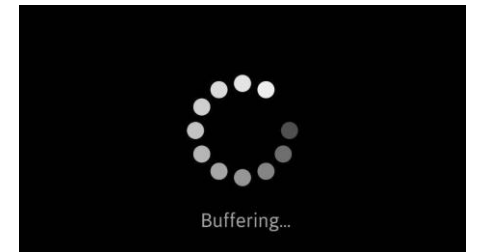
- server-to-client bandwidth will *vary* over time, with changing network congestion levels (in house, in access network, in network core, at video server)
- packet loss and delay due to congestion will delay playout, or result in poor video quality

Streaming stored video

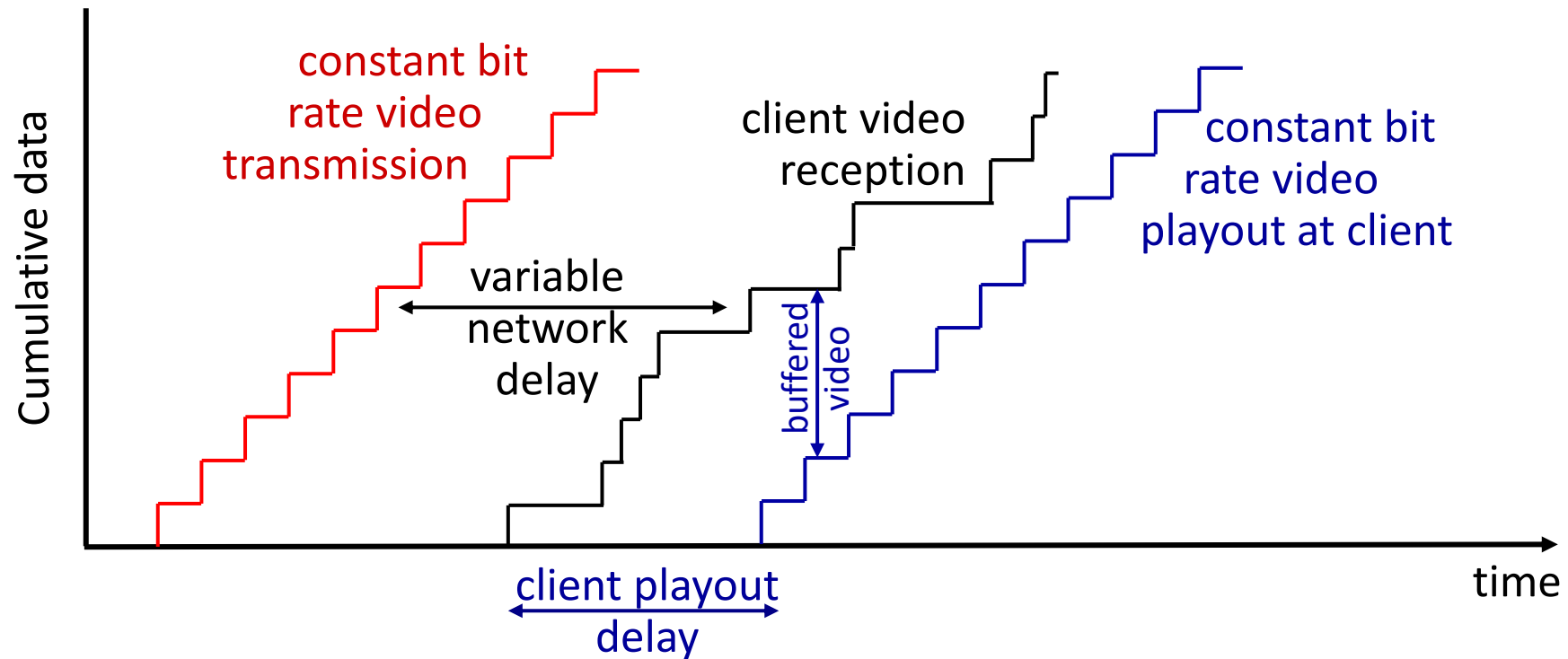


Streaming stored video: challenges

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



Streaming stored video: playout buffering



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

Streaming multimedia: DASH

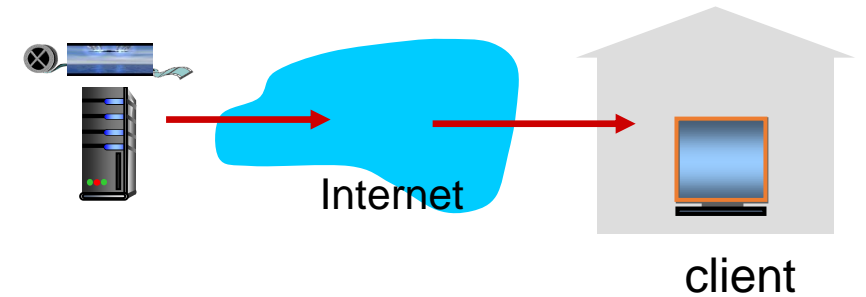
■ *DASH*: *D*ynamic, *A*daptive *S*teaming over *H*TTP

■ *server*:

- divides video file into multiple chunks
- each chunk stored, encoded at different rates
- *manifest file*: provides URLs for different chunks

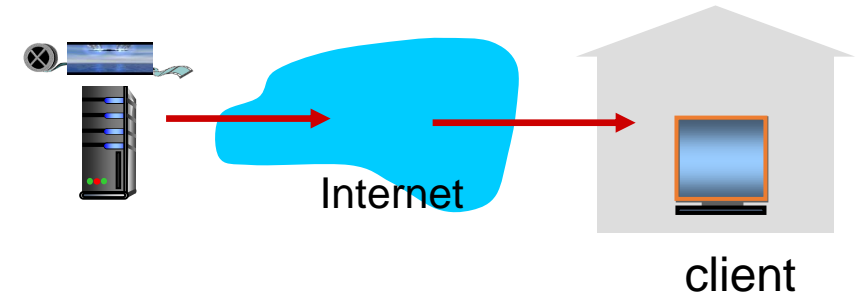
■ *client*:

- 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 at different points in time (depending on available bandwidth at time)



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?
- *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 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*)
 - *enter deep*: push CDN servers deep into many access networks
 - close to users
 - Akamai: 240,000 servers deployed in more than 120 countries (2015)
 - *bring home*: smaller number (10's) of larger clusters in POPs near (but not within) access networks
 - used by Limelight



Content distribution networks (CDNs)

- CDN: stores copies of content at CDN nodes
 - e.g. Netflix stores copies of MadMen
- subscriber requests content from CDN
 - directed to nearby copy, retrieves content
 - may choose different copy if network path congested

