The eMail, P2P applications, CDN

CE 352, Computer Networks

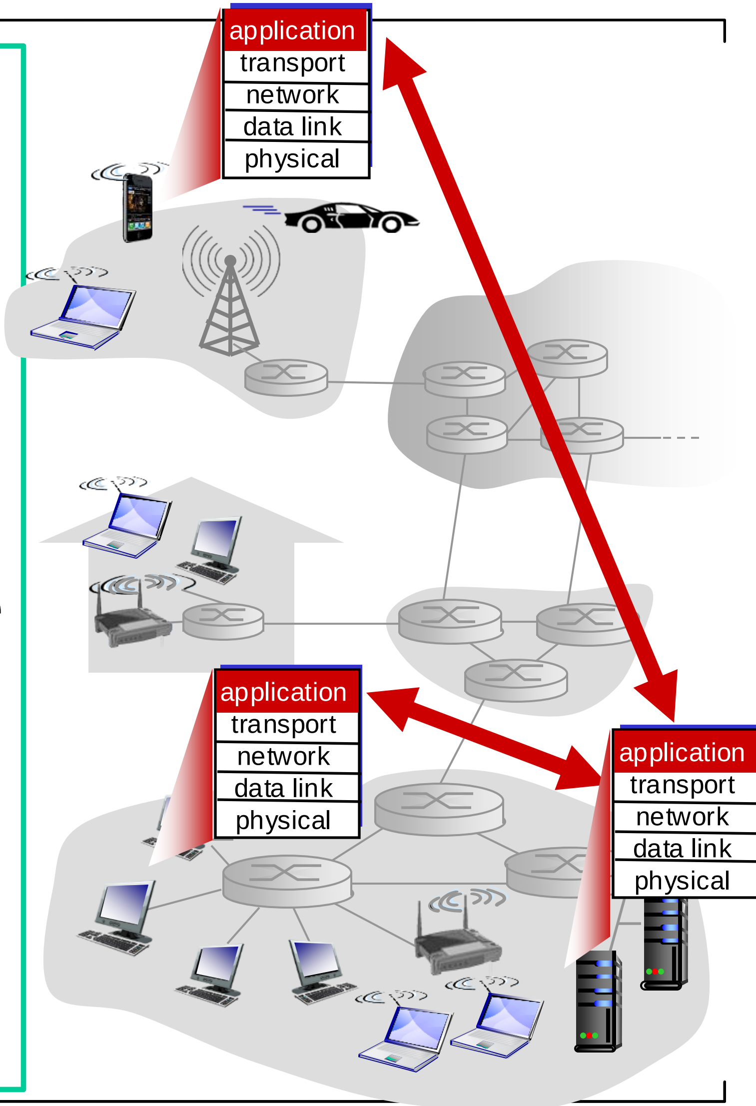
Salem Al-Agtash

Lecture 6

Slides are adapted from Computer Networking: A Top Down Approach, 7th Edition © J.F Kurose and K.W. Ross

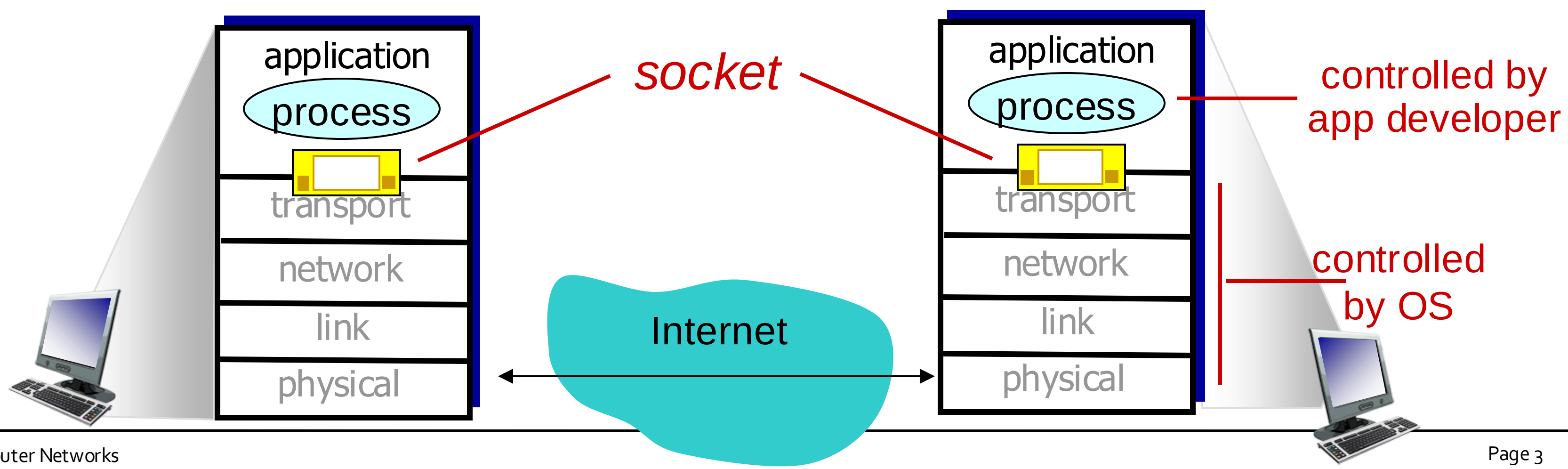
Recap (Applications on the Network)

- End-end system programs
- Architecture
  - Client – Server
  - Peer-to-peer (P2P)
- no need to write software for network-core devices
- Examples:
  - Web, e-mail, text messaging, remote login, file transfer
  - social networking, multi-user network games
  - VoIP, streaming stored video (YouTube, Hulu, Netflix)



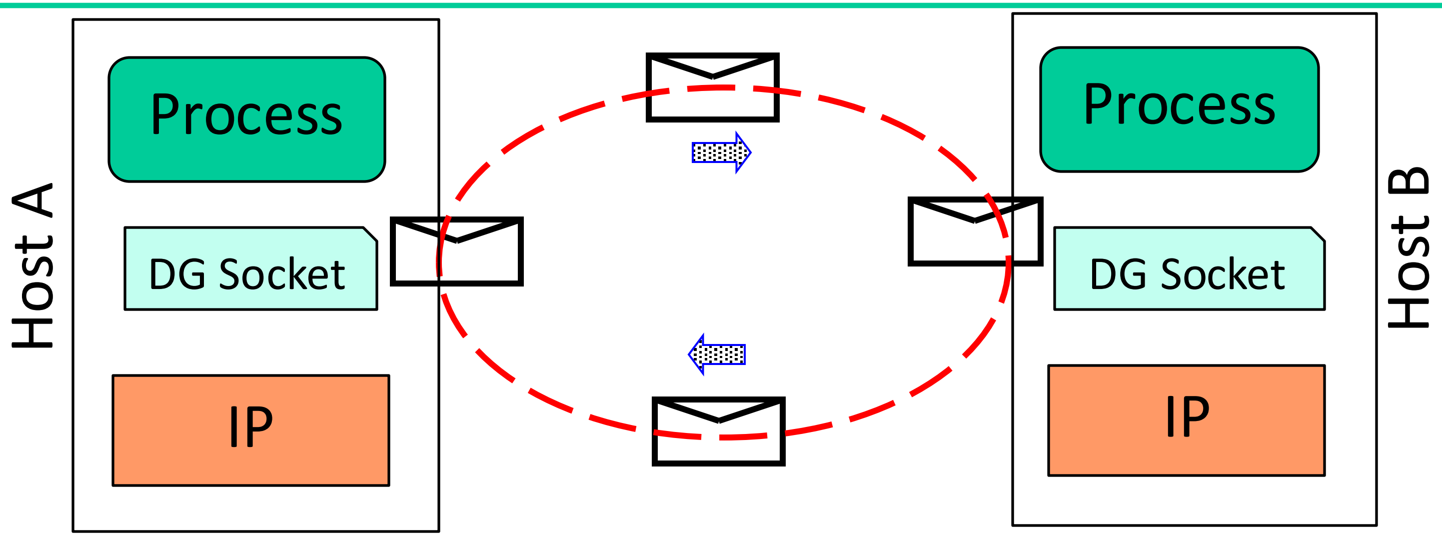
Recap (process communication)

- Process: Program in Execution
  - Same hosts: processes communicate using IPC defined by OS. e.g. Pipes, Shared Memory, Message Queues
  - Different hosts: processes communicate by exchanging messages. e.g. Client-Server, P2P
- Socket: Process sends/ receives messages via socket (IP + Port)
  - Sending process shoves message out door and relies on transport infrastructure to deliver message to socket at receiving process



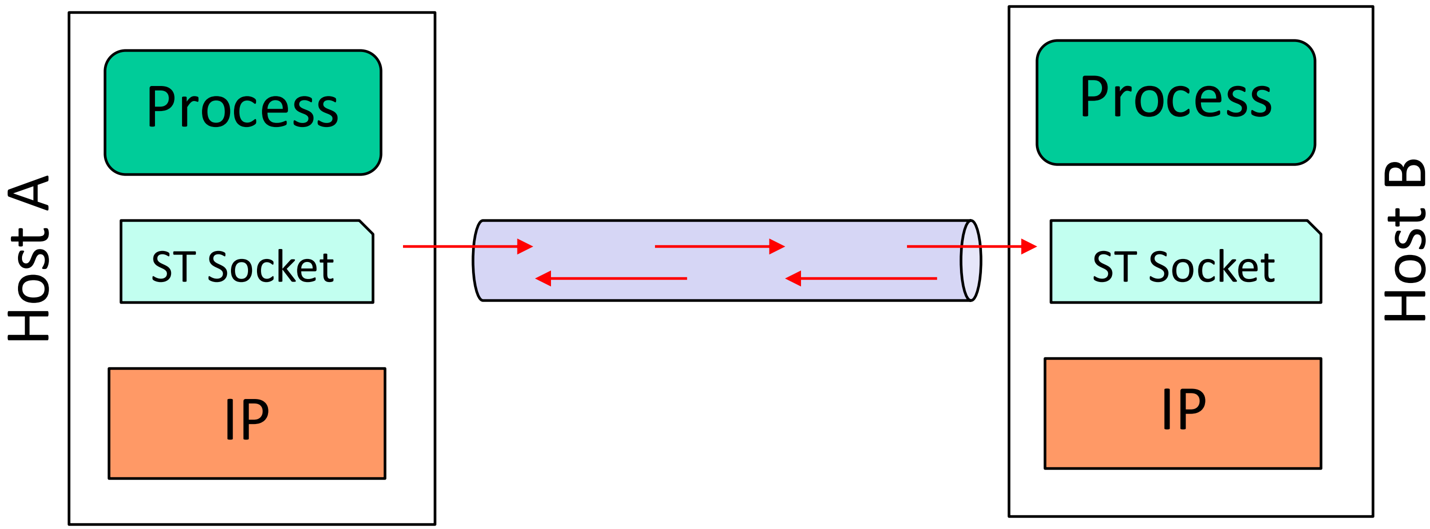
Datagram socket: UDP)

- UDP: no "connection" between client & server
  - no handshaking before sending data
  - sender explicitly attaches IP destination address and port # to each packet
  - receiver extracts sender IP address and port# from received packet
- UDP: transmitted data may be lost or received out-of-order
- Application viewpoint: UDP provides *unreliable* transfer of groups of bytes ("datagrams") between client and server



Stream socket: TCP

- TCP: "connection" between client & server
  - Sever creates socket and begins to listen
  - Client contacts server by creating TCP socket, specifying IP/port of server
  - Server creates thread to communication with particular client
- Application viewpoint: TCP provides *reliable* in-order byte stream transfer ("pipe") between client and server



System calls

Fill in IP and Port
- `struct sockaddr_in servAddr, clientAddr;`

Create a socket
- `socket(AF_INET,SOCK_STREAM,0);`

Bind the socket
- `bind(sockfd,(..)&servAddr, sizeof(servAddr))`

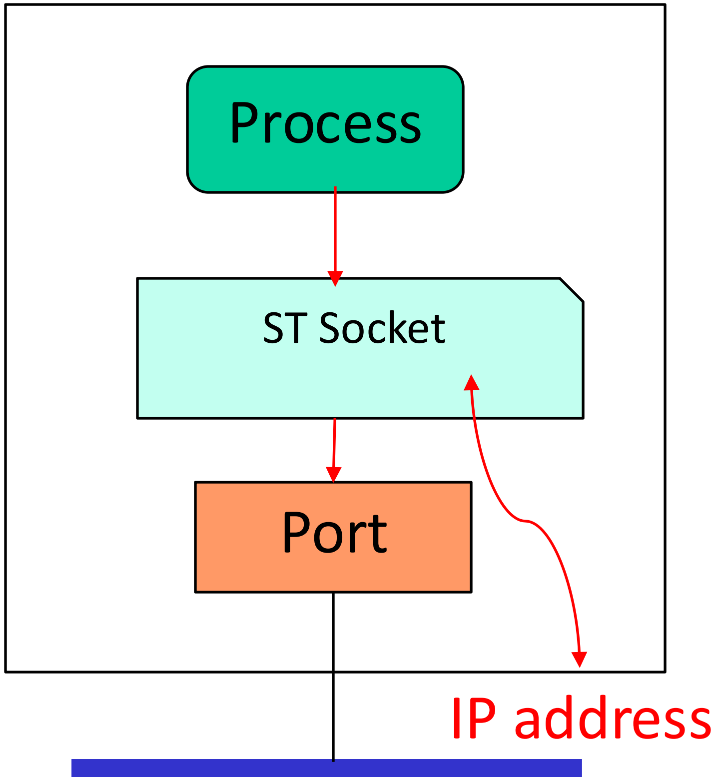
Server listens for connections
- `listen(sockfd,n);`

Client connects to a server
- `connect(sockfd,(..sockaddr*)&servAddr, sizeof(servAddr));`

Sever accepts connection
- `accept(sockfd,(struct sockaddr *)&clientAddr,sizeof(clientAddr));`

Read/write, send/receive

Binding address to socket



Recap (WWW)

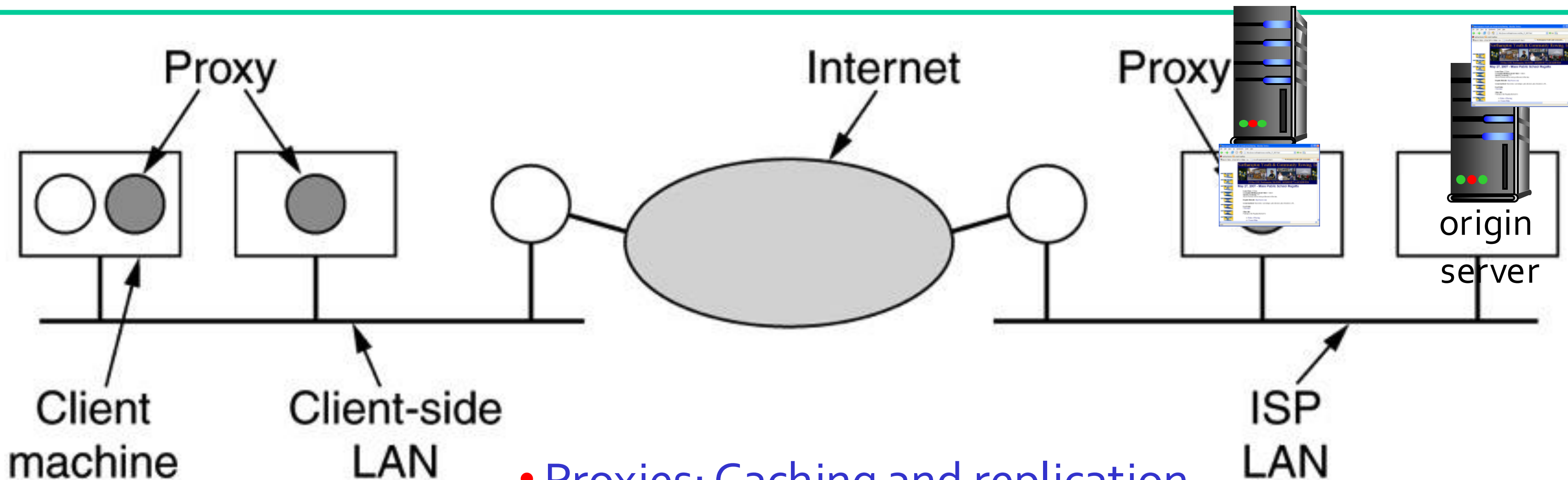
- Distributed database of "pages" linked through HTTP
  - HTTP/0.9, 1.0, 1.1, 2.0
  - Persistent and non-persistent HTTP, request, response
  - DNS (UDP) and HTTP (TCP)
  - HTML and Dynamic web pages

- Web components
  - Infrastructure:
    - Clients, Servers
  - Content:
    - URL: naming content
    - HTML: formatting content
  - Protocol for exchanging information: HTTP, HTTPS

- Cookies and web caches (proxy server)

Recap (Web caches (proxy server))

- Web users
  - Availability and fast downloads
- Web content providers
  - More users, cost-effective infrastructure, and non-congested network



- Proxies: Caching and replication
- Content delivery networks (CDN): economies of scale

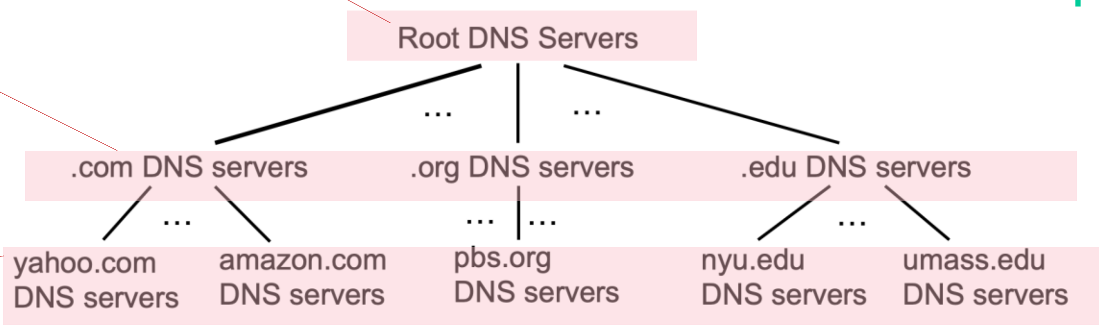
Recap (DNS: a distributed, hierarchical database

13 root servers (labeled A – M):

- contacts authoritative name server if name mapping not known
- returns mapping to local name server

top-level domain (TLD) servers:

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

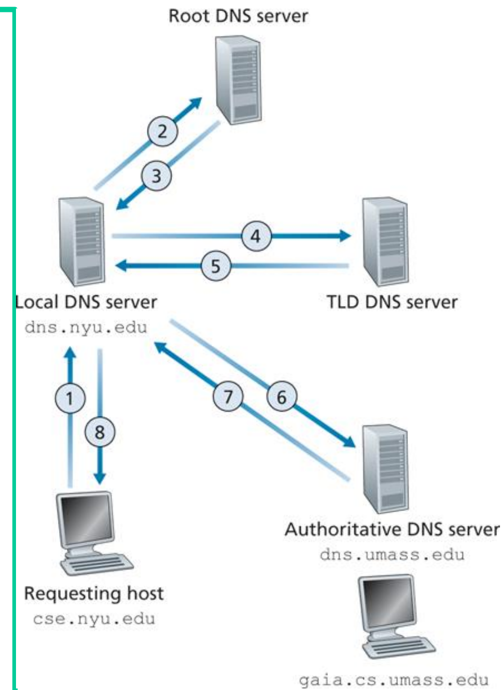


authoritative DNS servers:

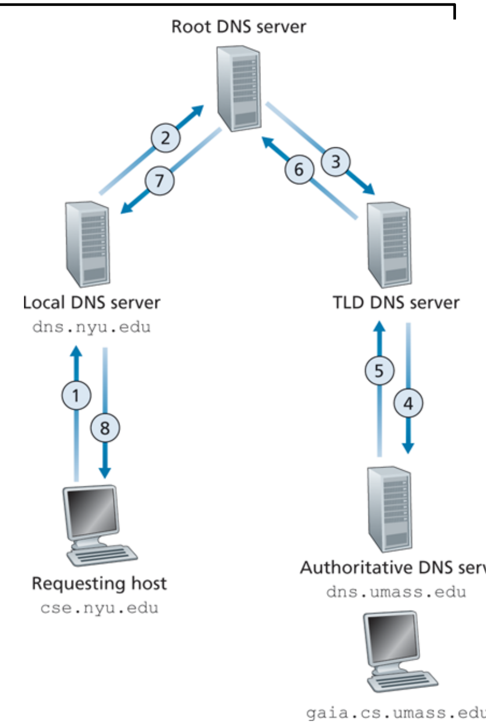
- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

Recap (DNS)

- Client--server on UDP Port 53
- DNS hierarchy
  - Root DNS server
  - TLD DNS server
  - Local/ authoritative DNS server
- DNS (BIND, Windows DNS)
  - Hostname → IP
  - Host and mail server aliasing
  - Load distribution
- DNS records
  - Cashing (root server not often visited)
- DNS attacks
  - DDoS and DNS poisoning



Iterative query

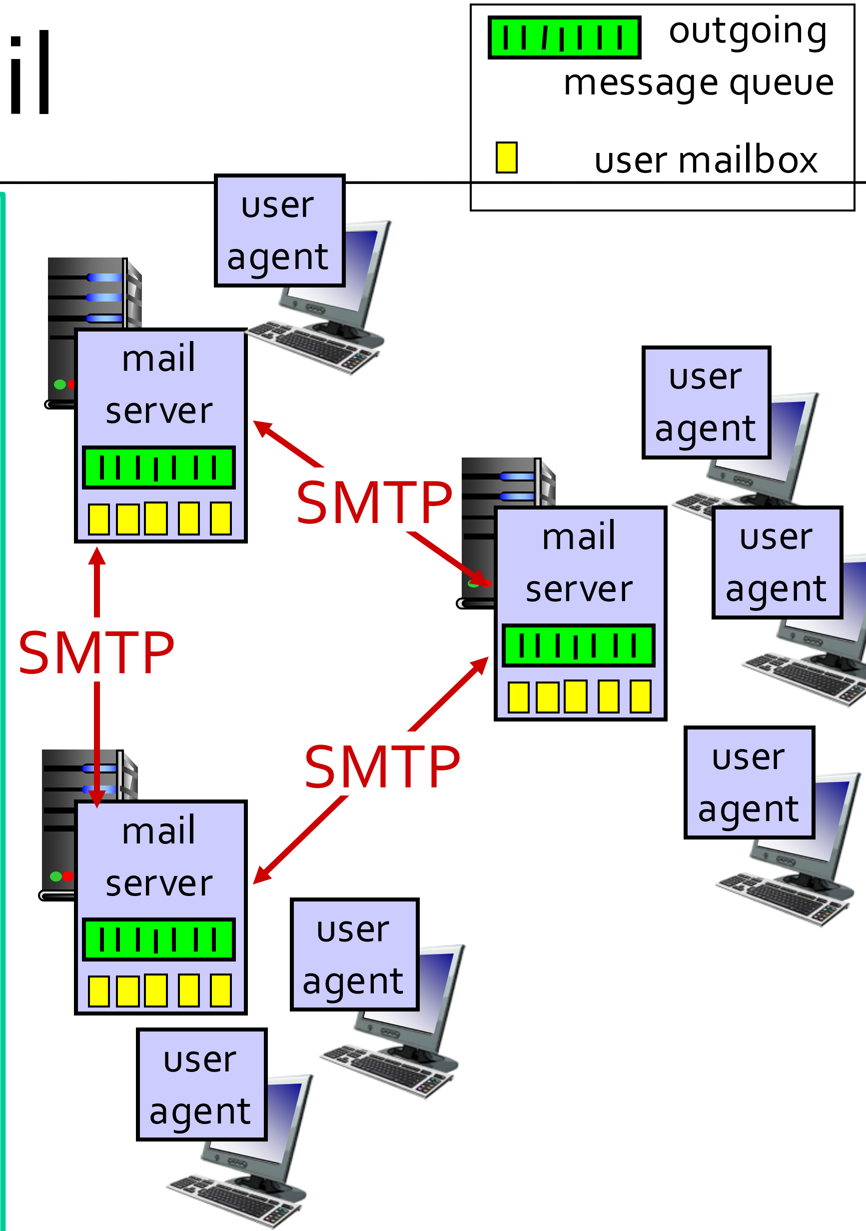


Recursive query

- Today
  - electronic mail: SMTP, POP3, IMAP
  - P2P applications
  - video streaming and content distribution networks (CDNs)

Component of eMail

- *User Agent* a.k.a. "mail reader"
  - composing, editing, reading mail messages
  - e.g., Outlook, Thunderbird, Mail
  - outgoing, incoming messages stored on server
- Mail servers:
  - *mailbox* contains incoming messages for user
  - *message queue* of outgoing (to be sent) mail messages
- *SMTP protocol*
  - between mail servers
  - client: sending mail server
  - "server": receiving mail server



SMTP [RFC 2821]

uses TCP to reliably transfer email message from client to server, port 25

direct transfer: sending server to receiving server

three phases of transfer

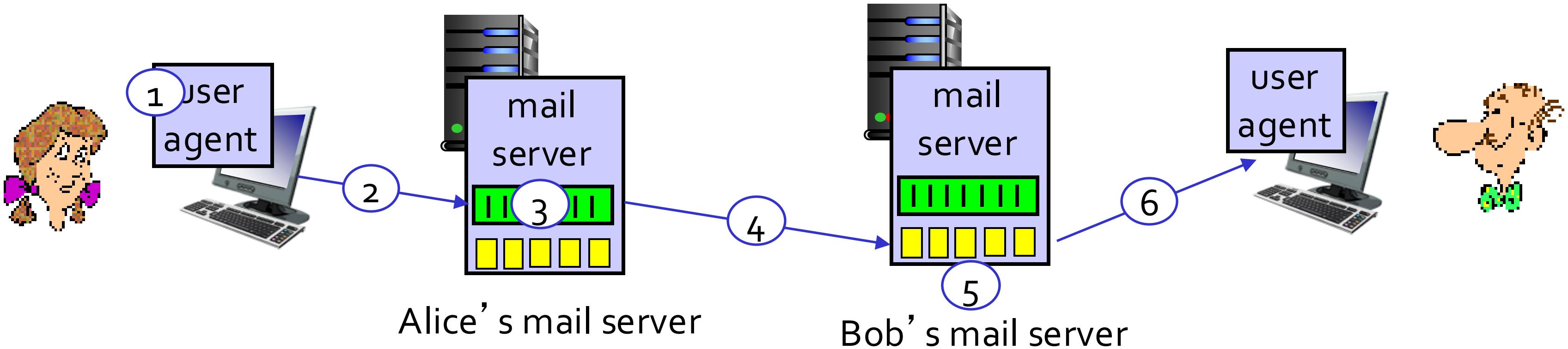
- handshaking (greeting)
- transfer of messages
- closure

command/response interaction (like HTTP)

- commands: ASCII text
- response: status code and phrase

Scenario: Alice sends message to Bob

1) Alice uses UA to compose message "to" bob@someschool.edu
2) Alice's UA sends message to her mail server; message placed in message queue
3) client side of SMTP opens TCP connection with Bob's mail server
4) SMTP client sends Alice's message over the TCP connection
5) Bob's mail server places the message in Bob's mailbox
6) Bob invokes his user agent to read message



Sample SMTP interaction

S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection

- *TRY Yourself*

`telnet servername 25`

see 220 reply from server

enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

Sample SMTP interaction

```
Connection closed by foreign host.
ENGR-L-00688:~ salagtash$ telnet mail.smtp2go.com 2525
Trying 173.255.233.87...
Connected to mail.smtp2go.com.
Escape character is '^]'.
220 mail.smtp2go.com ESMTP Exim 4.94.2-S2G Wed, 19 Jan 2022 00:38:43 +0000
EHLO
250-mail.smtp2go.com Hello  [162.229.186.16]
250-SIZE 52428800
250-8BITMIME
250-DSN
250-PIPELINING
250-PIPE_CONNECT
250-AUTH CRAM-MD5 PLAIN LOGIN
250-CHUNKING
250-STARTTLS
250-PRDR
250-SMTPUTF8
250 HELP
HELO
250 mail.smtp2go.com Hello  [162.229.186.16]
MAIL FROM:<alagtash@gmail.com>
250 OK
```

- *TRY Yourself*

`telnet servername 25`

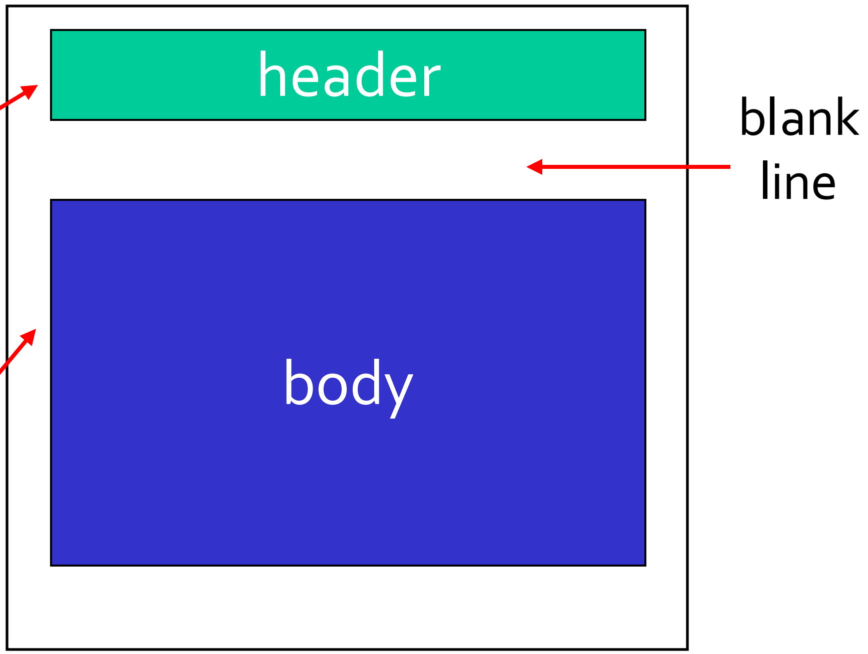
see 220 reply from server

enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

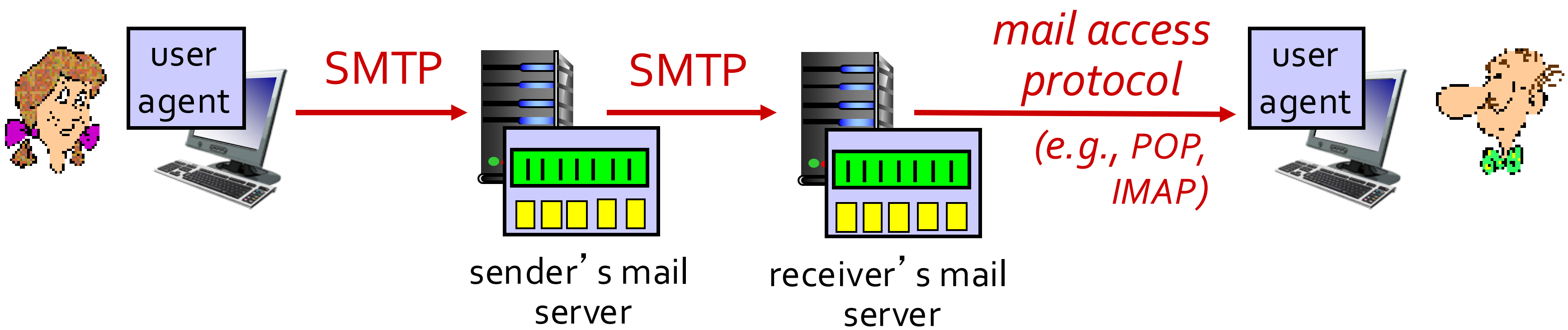
above lets you send email without using email client (reader)

Mail message format

- SMTP: protocol for exchanging email messages
- RFC 822: standard for text message format:
  - header lines, e.g.,
    - From: alice@crepes.fr
    - To: bob@hamburger.edu
    - Subject: Searching for the meaning of life.
- Body: the "message"
- ASCII characters only



Mail access protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]: authorization, download – port 110
  - IMAP: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored messages on server - 143
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

Mail settings Network Solutions

POP / IMAP	pop3
Incoming server	mail.[domain].com
Incoming port	995
SSl (security) incoming	ssl
Outgoing server	smtp.[domain].com
Outgoing port	587
Requires sign-in	yes

POP3 protocol

authorization phase

client commands:

- **user:** declare username
- **pass:** password

server responses

- **+OK**
- **-ERR**

transaction phase

client:

- **list:** list message numbers
- **retr:** retrieve message by number
- **dele:** delete
- **quit**

S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off

POP3 and IMAP

more about POP3

previous example uses POP3 "download and delete" mode

- Bob cannot re-read e-mail if he changes client

POP3 "download-and-keep": copies of messages on different clients

POP3 is stateless across sessions

IMAP

keeps all messages in one place: at server

allows user to organize messages in folders

keeps user state across sessions:

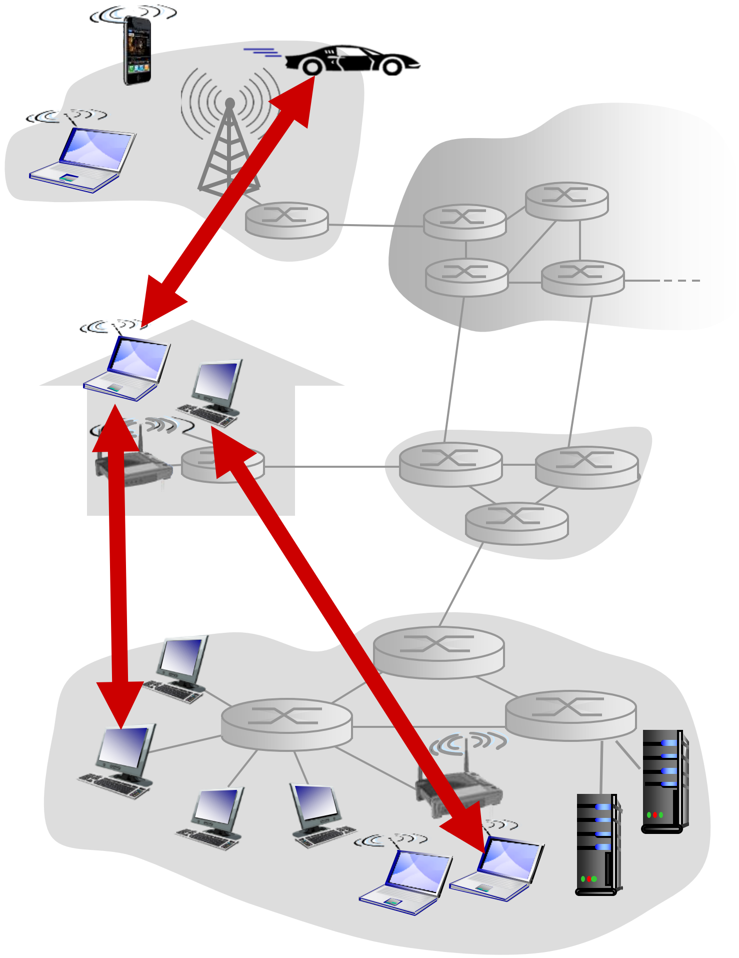
- names of folders and mappings between message IDs and folder name

Pure P2P architecture

- *not* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

examples:

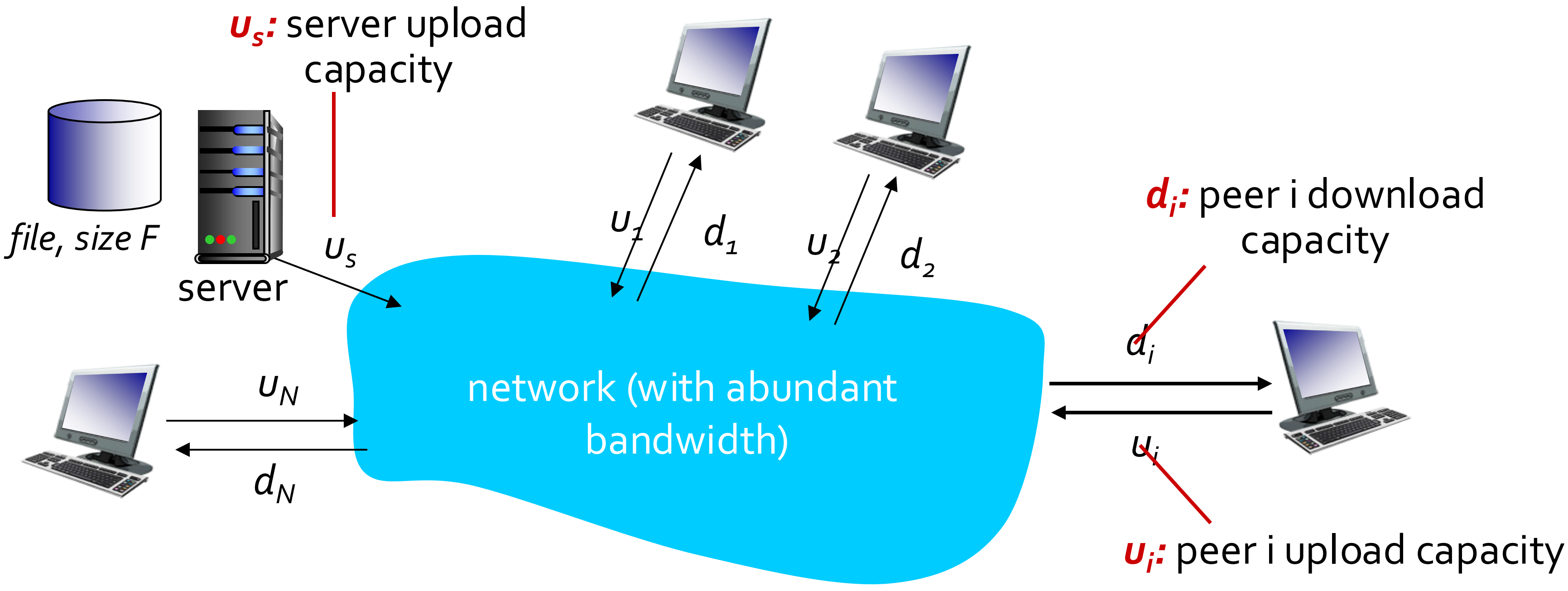
- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)



File distribution: client-server vs P2P

Time to distribute file (size *F*) from one server to *N* *peers*?

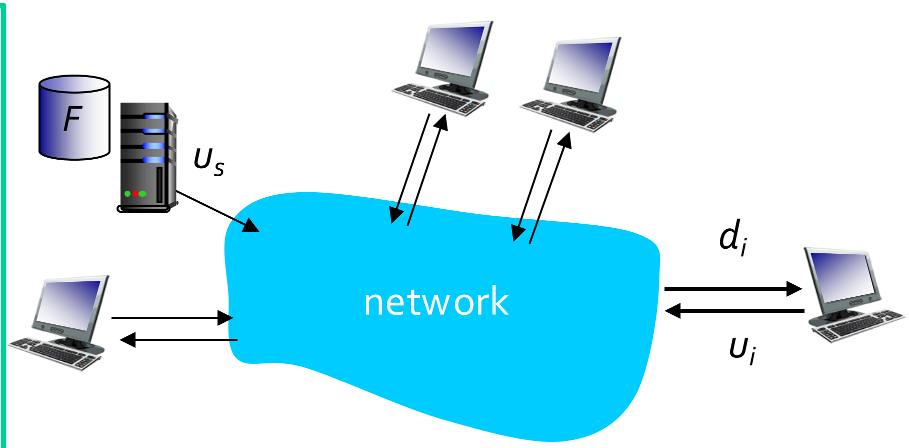
- peer upload/download
- capacity is limited resource



File distribution time: client-server

server transmission: must sequentially send (upload) *N* file copies:

- time to send one copy: ?
- time to send *N* copies: ?
- *client:* each client must download file copy
- d_{min} = min client download rate
- client download time (min d): ?

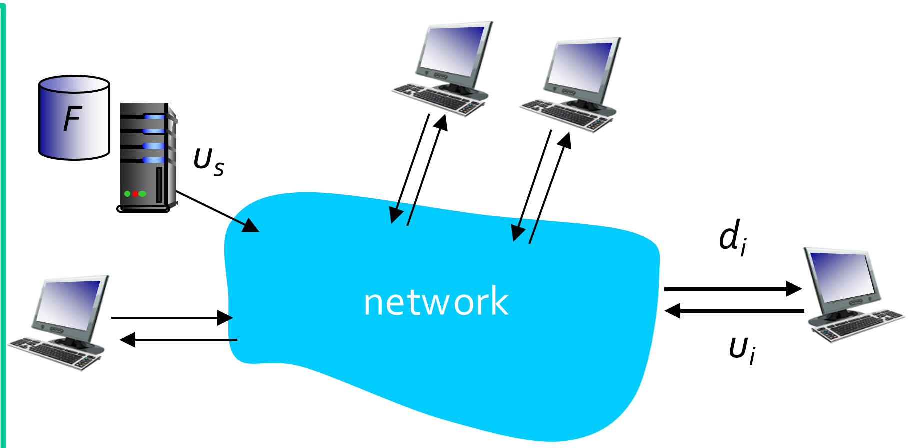


time to distribute F to N clients using client-server approach ?

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
- client download time (min d): 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

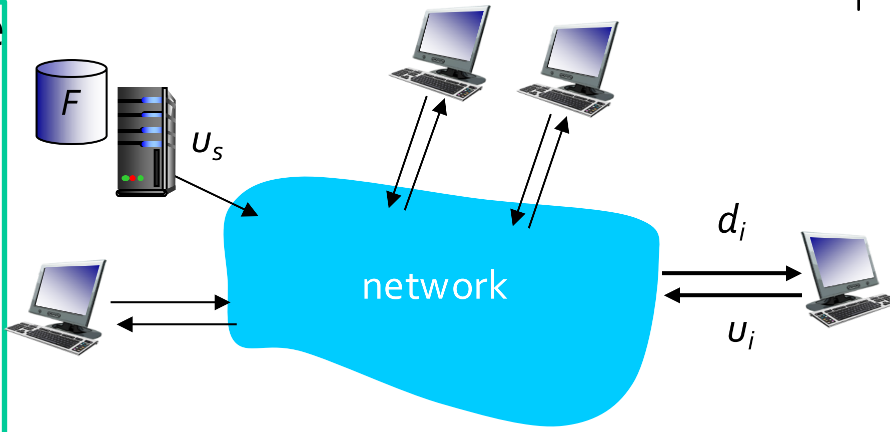
- time to send one copy: *?*

client: each client must download file copy

- client download time with min d: *?*

clients: as aggregate must upload *NF* bits

- max upload rate: *?*
- min distrib time: *?*



time to distribute F to N clients using P2P approach *?*

File distribution time: P2P

server transmission: must upload at least one copy

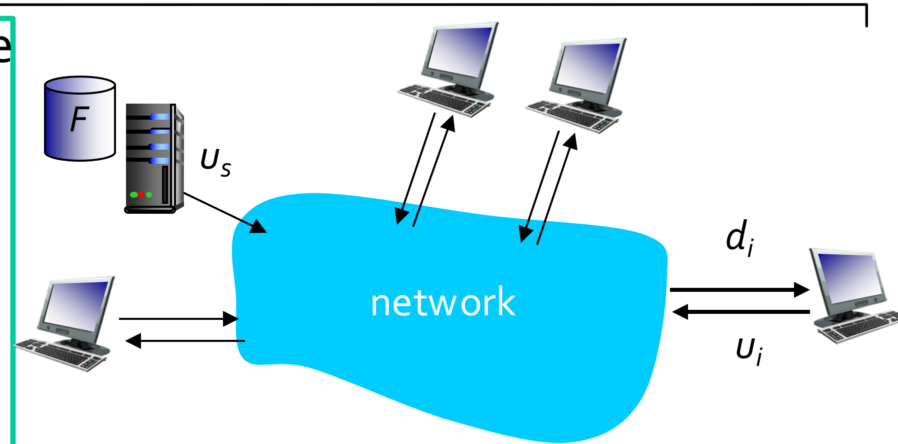
- time to send one copy: F/u_s

client: each client must download file copy

- client download time with mid d: F/d_{min}

clients: as aggregate must upload *NF* bits

- max upload rate: $u_s + Sum(u_i)$
- min distrib time: $NF/[u_s + Sum(u_i)]$



time to distribute F to N clients using P2P approach

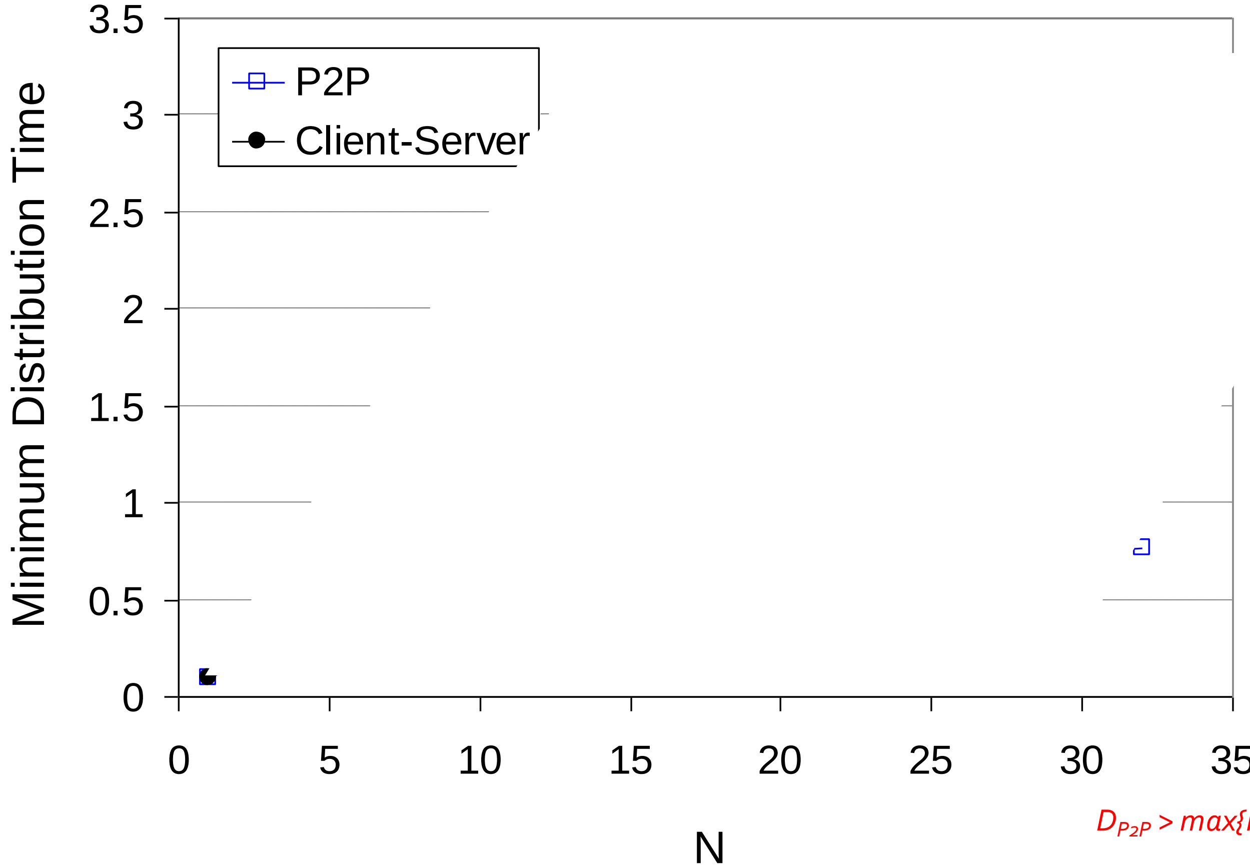
$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + Sum\,u_i)\}$$

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

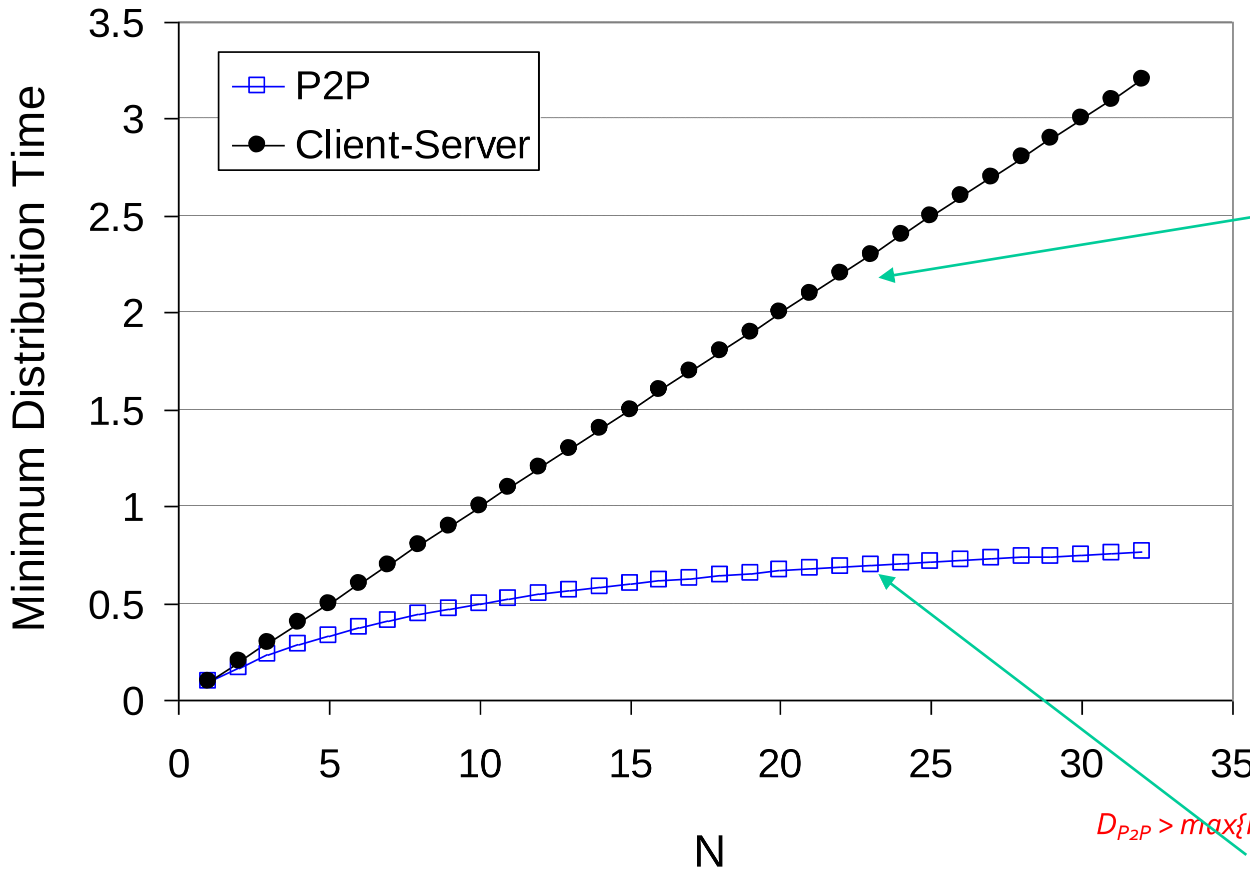


$D_{c\text{-}s} \geq \max\{NF/u_s, F/d_{min}\}$
$= \max(N/10, 1/10)$
$= N/10$

$D_{P2P} > \max\{F/u_s, F/d_{min}, NF/(u_s + \text{Sum}\,u_i)\}$
$= N/(N+10)$

Client-server vs. P2P: example

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



$D_{c\text{-}s} \geq \max\{NF/u_s, F/d_{min}\}$
$= \max(N/10, 1/10)$
$= N/10$

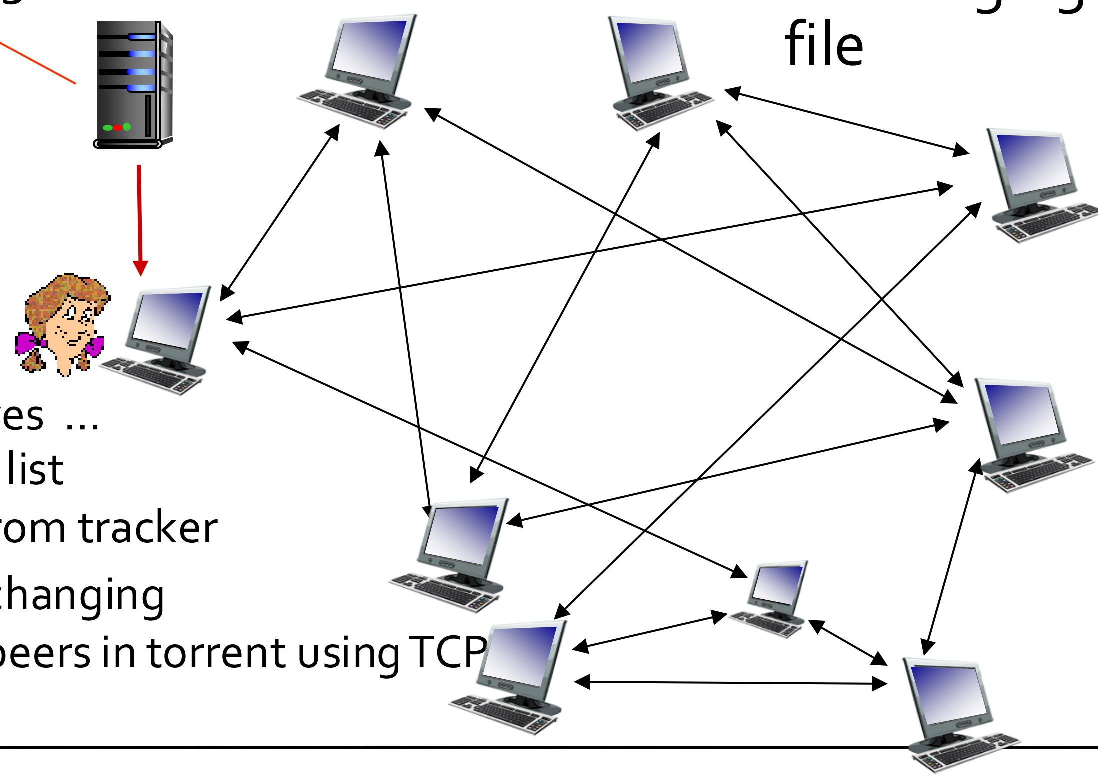
$D_{P2P} > \max\{F/u_s, F/d_{min}, NF/(u_s + \mathrm{Sum}\,u_i)\}$
$= N/(N+10)$

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

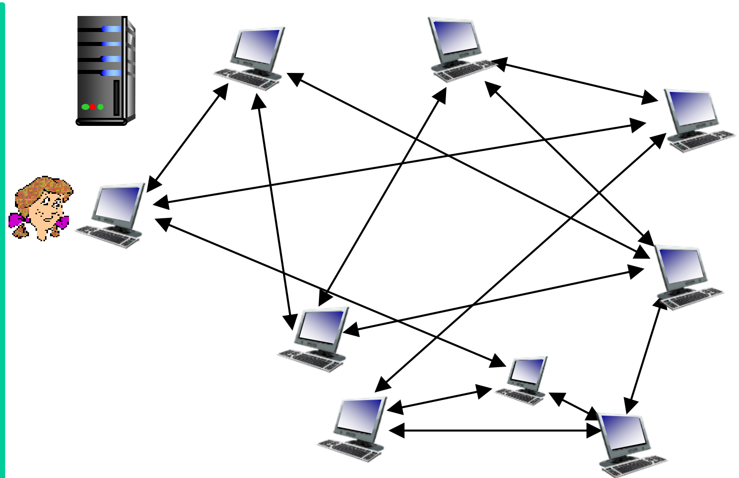


Alice arrives …
… obtains list
of peers from tracker
… and begins exchanging
file chunks with peers in torrent using TCP

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”)

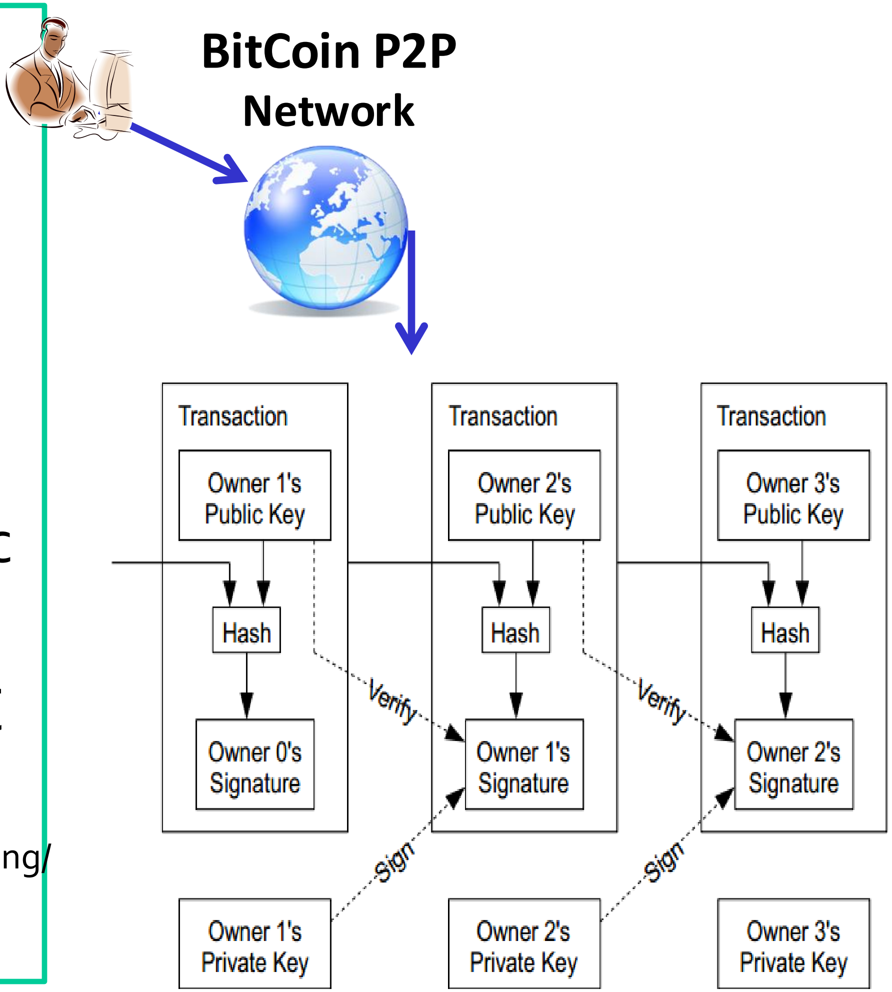


- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn:* peers may come and go
- once peer has entire file, it may (selfishly) leave or (voluntarily) remain in torrent

P2P Bitcoin mining

- Peer-to-peer computer process used to secure and verify bitcoin transactions
- A distributed ledger that tracks fund transfers among accounts
- BitCoin transfer: Sign(Previous transaction + New owner's public key)
- Mining one bitcoin with just a PC now takes millions of years!

http://www.techtangerine.com/2017/09/22/bitcoin-pc-mining/



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
- 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 to encode image
  - spatial (within image)
  - temporal (from one image to next)
- CBR: (constant bit rate) for video encoding
- VBR: (variable bit rate) for video encoding
- examples:
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, < 1 Mbps)
  - Low-quality (100Kbps), HD (3Mbps), 4K (10 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)

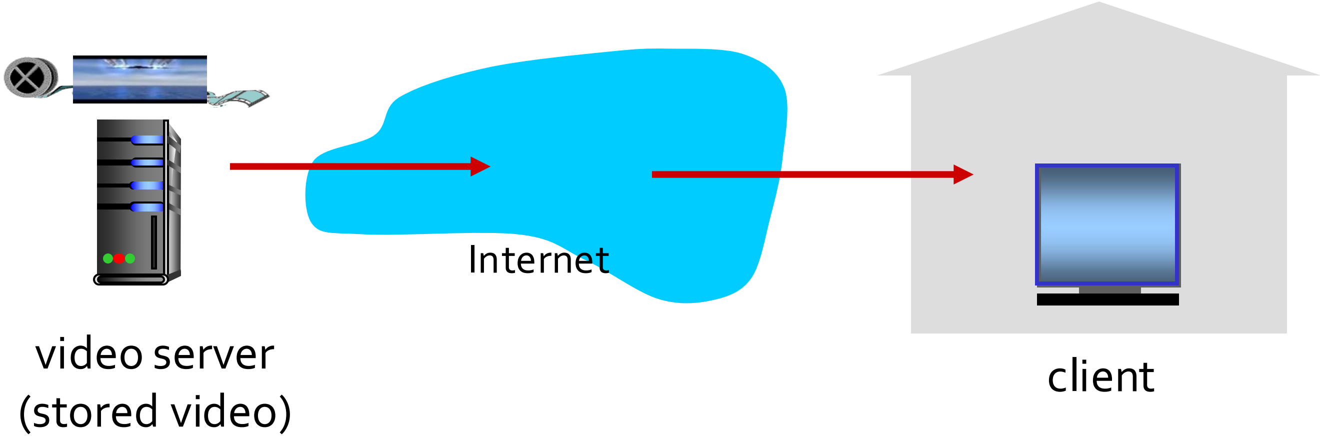
frame *i*

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

frame *i+1*

Streaming stored video:

- HTTP streaming: all clients receive same encoding of video
- But, clients have different amount of bandwidth available
- Solution: videos are encoded in different bit-rate versions, different quality videos. So clients dynamically requests chunks of video segments based on bandwidth availability -> DASH



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

Streaming multimedia: DASH

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 time (depending on bandwidth)

"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)

Content distribution networks

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*

CDN: Content distribution networks

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)*. Two placement scenarios:

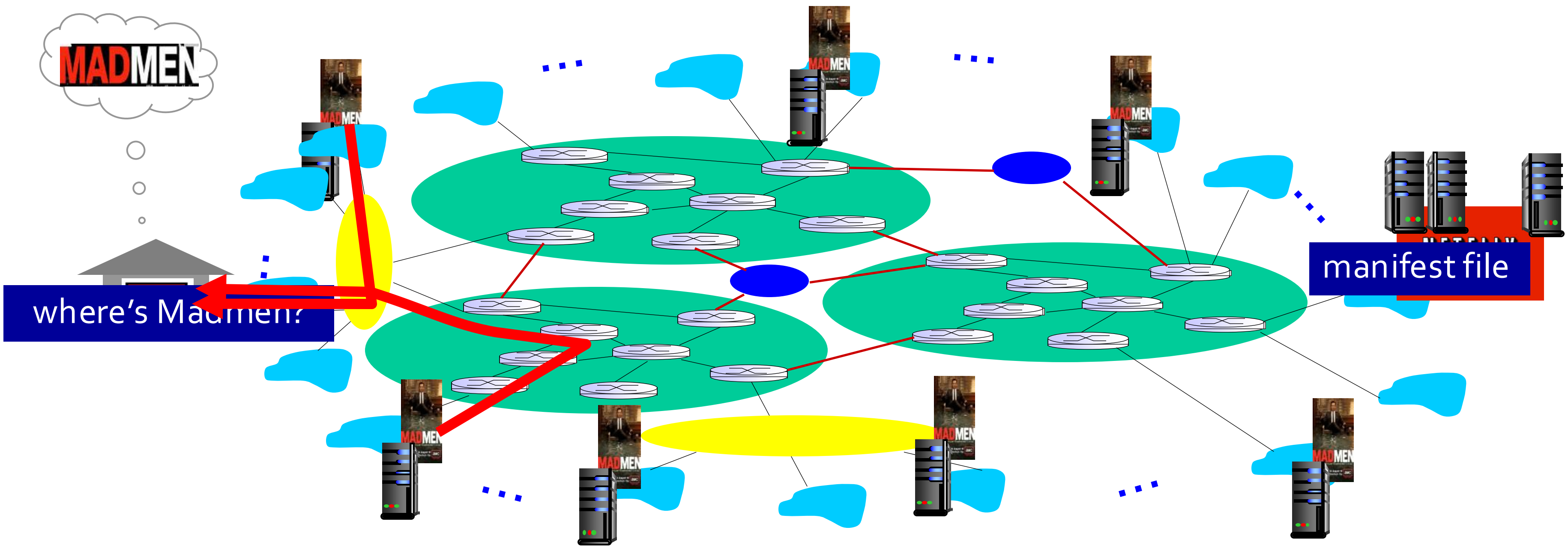
- *enter deep:* push CDN servers deep into many access networks
  - close to users
  - used by Akamai, 1700 locations
- *bring home:* smaller number (10's) of larger clusters in POPs near (but not within) access networks -> similar to web caching
  - used by Limelight

Google's network infrastructure

Three tiers of server clusters (2011, 2016 data):

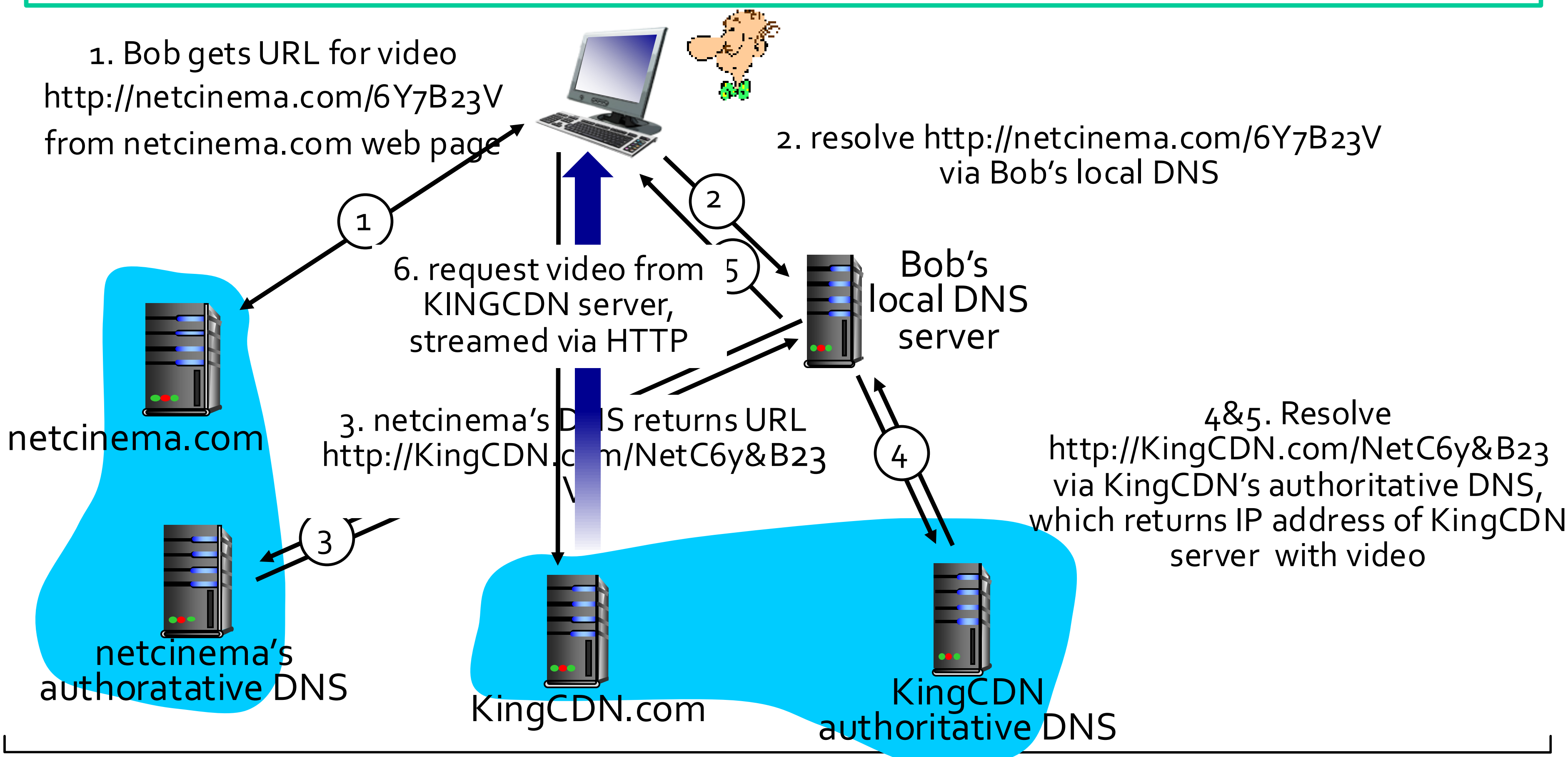
- 14 "mega data centers," each having ~ 100,000 servers, responsible for serving dynamic (and often personalized) content, including search results and Gmail messages.
  - 8 in North America,
  - 4 in Europe
  - 2 in Asia
- ~50 clusters in IXPs scattered throughout the world, each having 100–500 servers, responsible for serving static content, including YouTube videos.
- 100s of "enter-deep" clusters located within an access ISP, each having 10s of servers, responsible to perform TCP splitting and serve static content

Netflix stores copies of MadMen on CDN nodes. Subscriber → near CDN

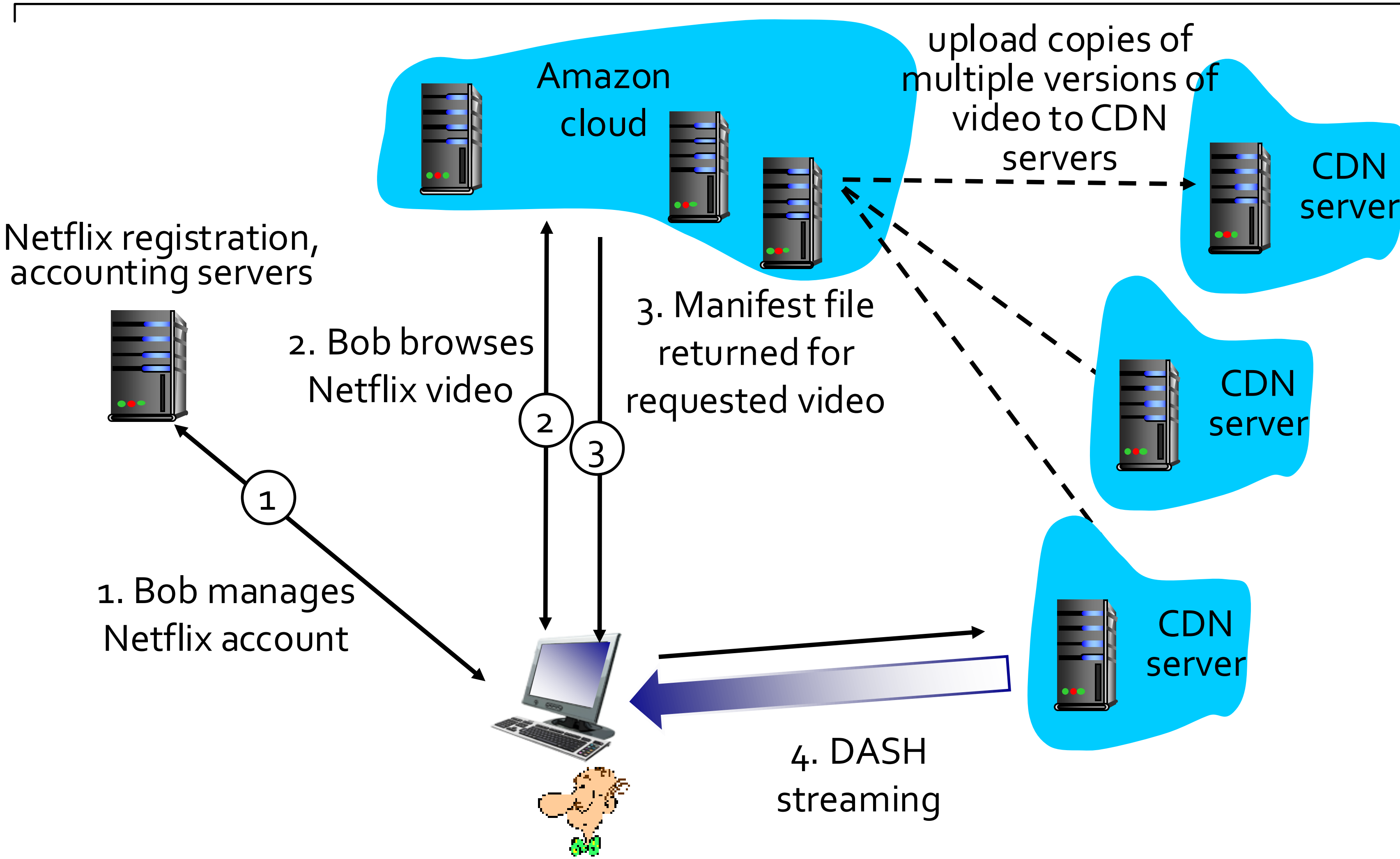


CDN content access: a closer look

- Bob (client) requests video http://netcinema.com/6Y7B23V
  - video stored in CDN at http://KingCDN.com/NetC6y&B23V



Netflix


Amazon cloud
upload copies of multiple versions of video to CDN servers
CDN server
CDN server
CDN server
Netflix registration, accounting servers
2. Bob browses Netflix video
3. Manifest file returned for requested video
2
3
1
1. Bob manages Netflix account
4. DASH streaming

Summary

Today:

- The eMail: SMTP, POP3
- P2P application architecture: BitTorrent
- Video streaming, CDNs

Canvas discussion:

- Reflection
- Exit ticket

Next time:

- read 3.1, 3.2, 3.3 and 3.4 of K&R (Transport layer)
- follow on Canvas! material and announcements

Any questions?