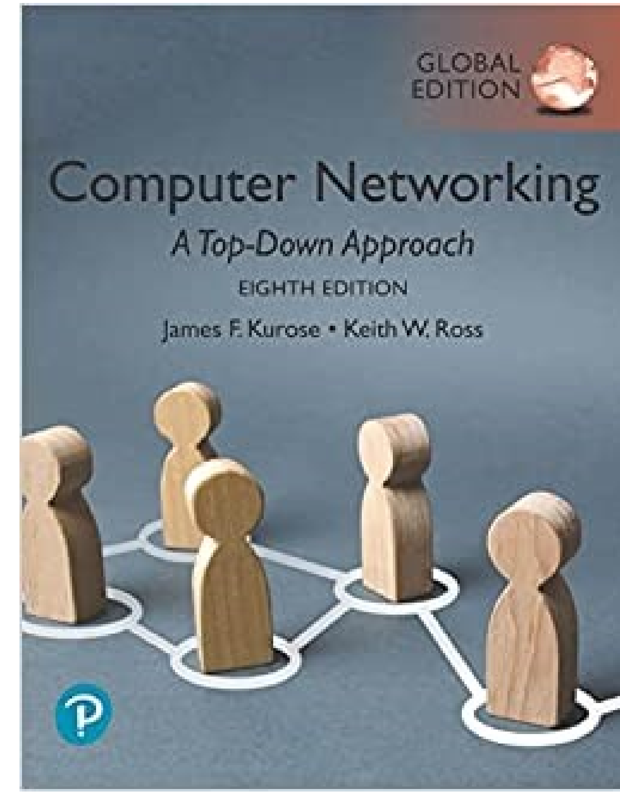
Chapter 2
Application Layer – part 2

School of Computing
Gachon Univ.
Joon Yoo



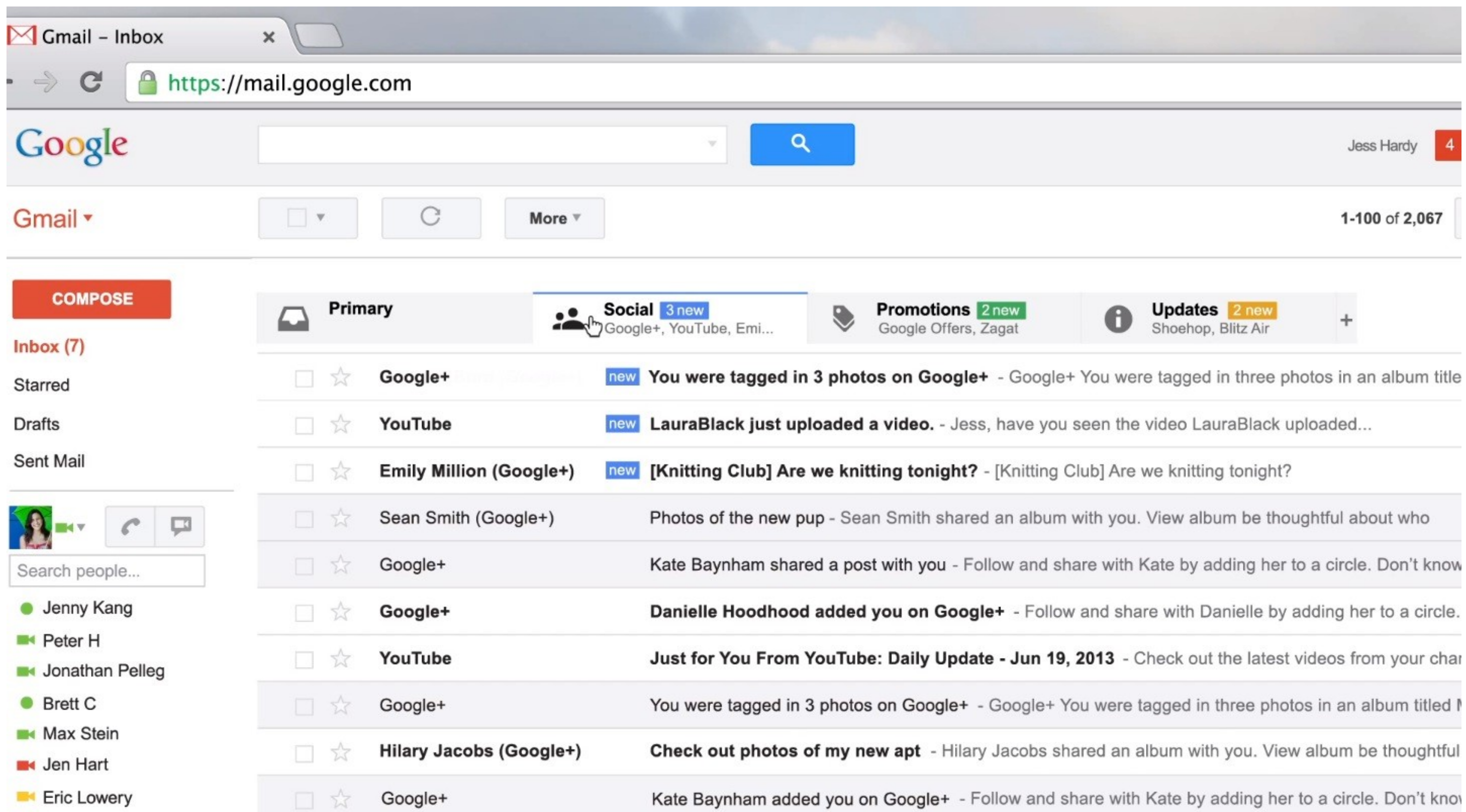
Computer Networking: A Top-Down Approach
8th edition (Global edition)
Jim Kurose, Keith Ross
Pearson, 2021

Many slides from J.F Kurose and K.W. Ross

가천대학교
Gachon University

Question

How can you receive your E-mail?



Chapter 2: outline

2.1 principles of network applications

- app architectures
- app requirements

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.6 Video Streaming and CDN

2.7 Socket Programming

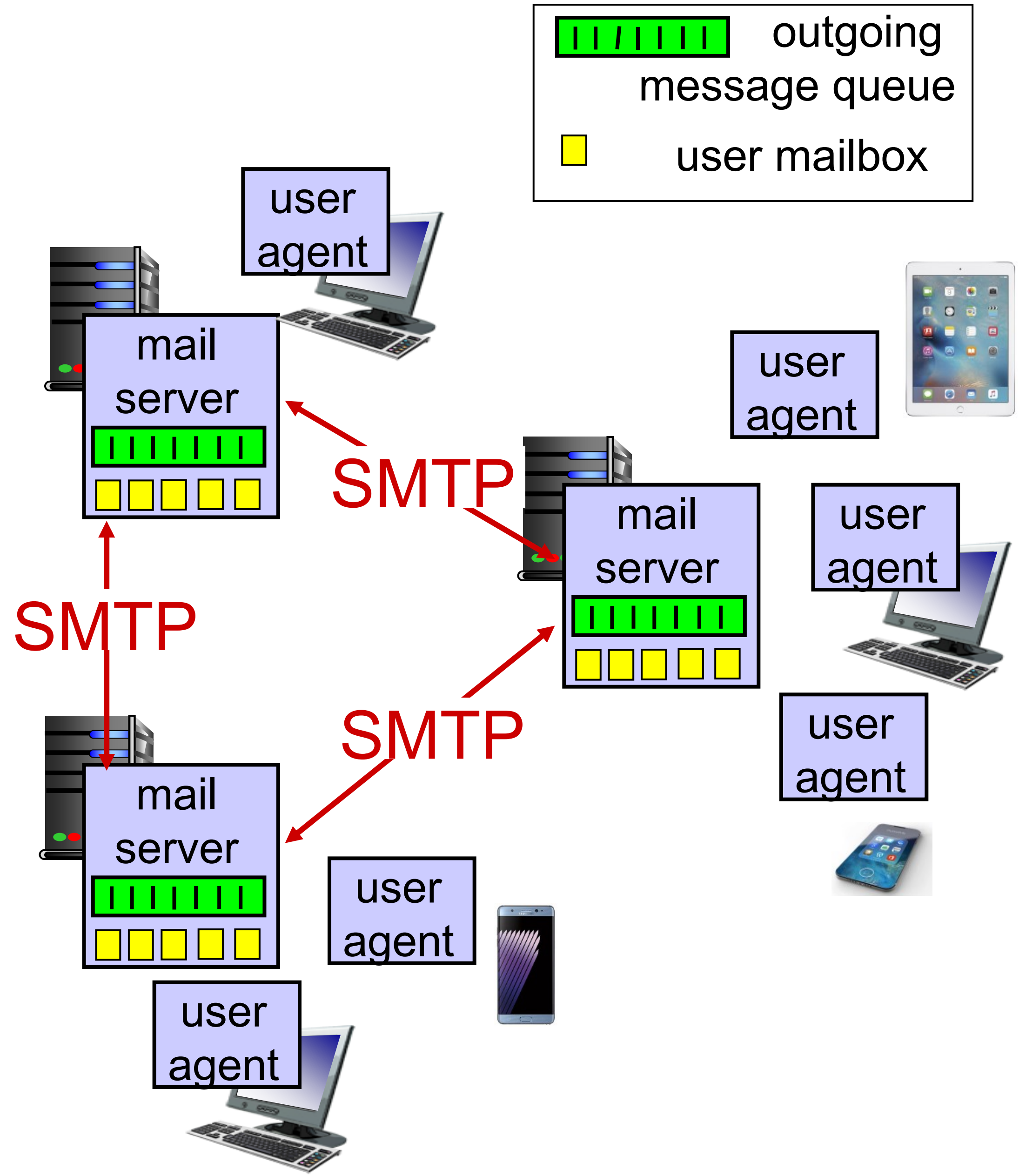
Electronic Mail

- Electronic Mail (E-mail) has been around since the beginning of the Internet – and remains one of the most important and utilized application
- E-mail is an *asynchronous* communication
  - People can send out messages when it is convenient for them, no need to coordinate other people's schedules
  - c.f., HTTP is *synchronous*: server-client must immediately communicate

Electronic mail

Three major components:

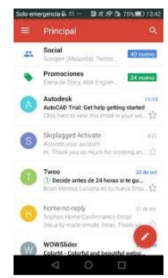
- user agents (UA)
- mail servers
- simple mail transfer protocol (SMTP) – application layer protocol

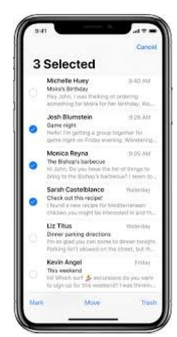


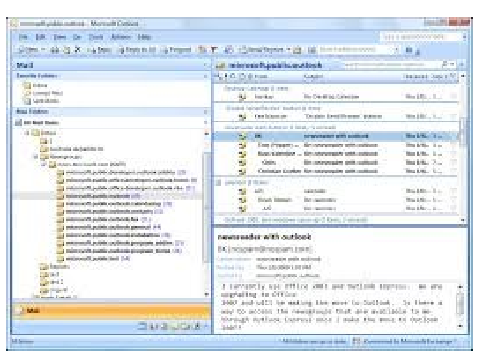
Electronic mail

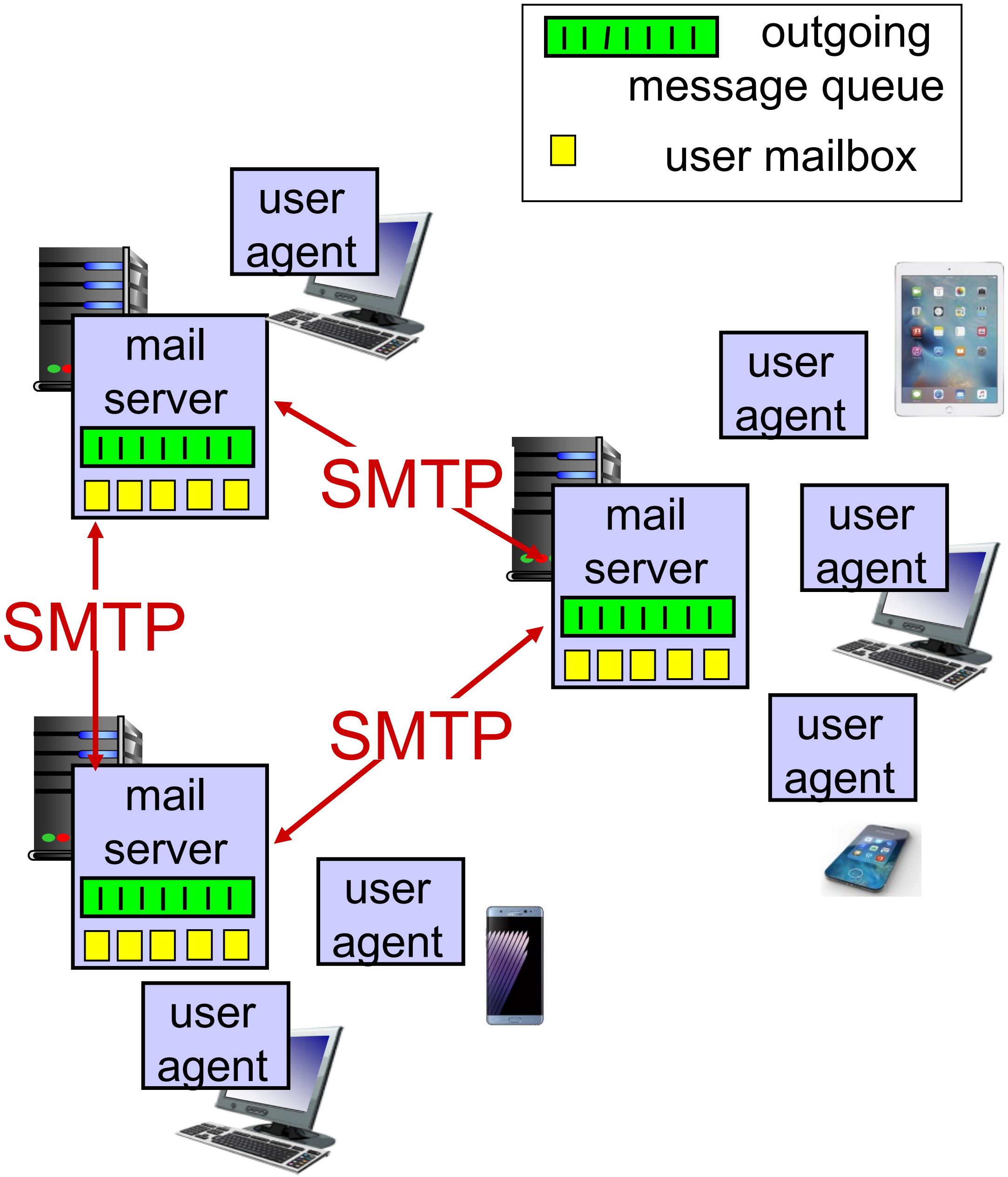
User Agent (UA)

- allows users to read, reply to, forward, save, and compose messages
  - Android Gmail client
  - iPhone mail client
  - MS Outlook



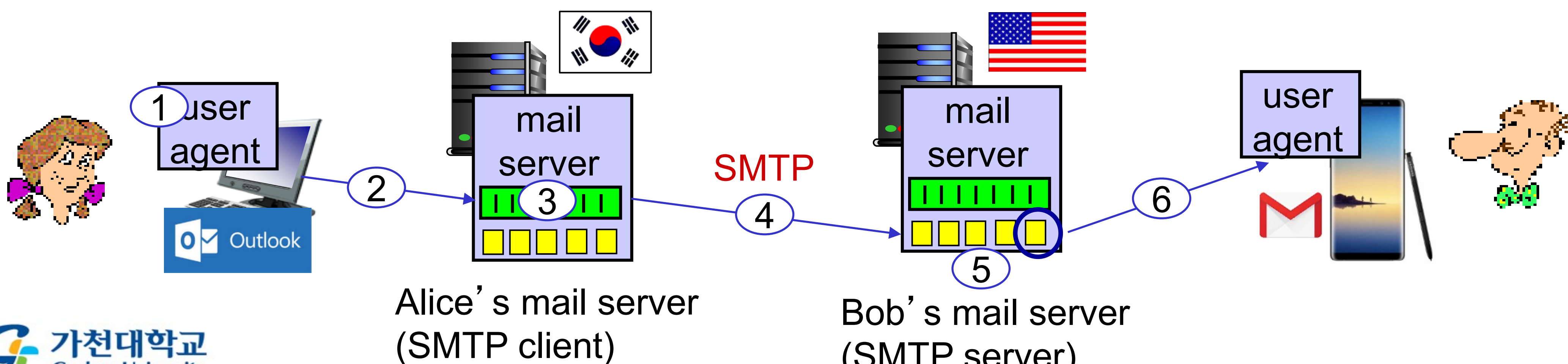






Scenario: Alice sends message to Bob

1) Alice uses **UA** to compose e-mail 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) When Bob wants to read a message, his **UA** retrieves message from his mailbox

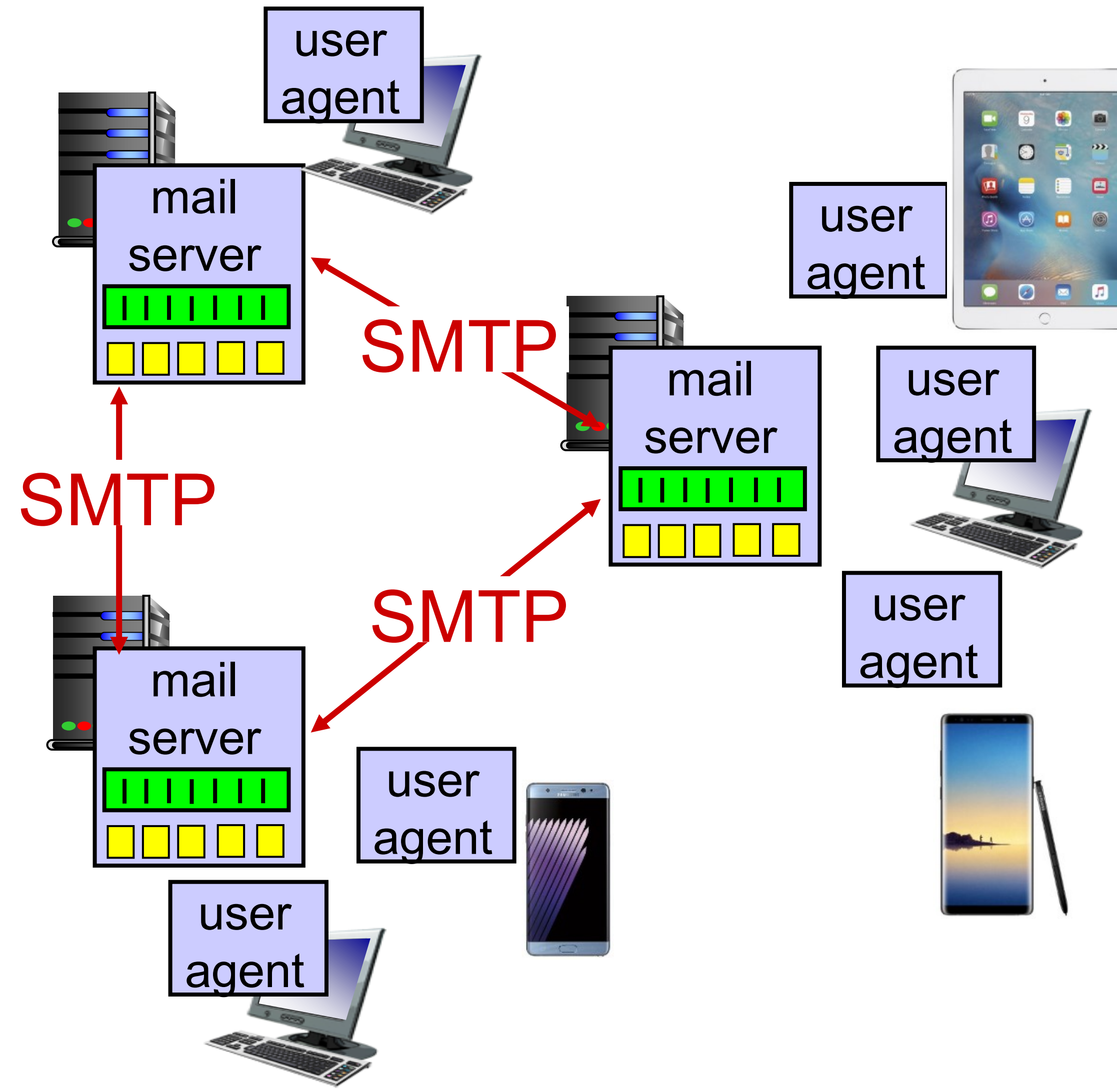


가천대학교 Gachon University

Electronic mail: mail servers

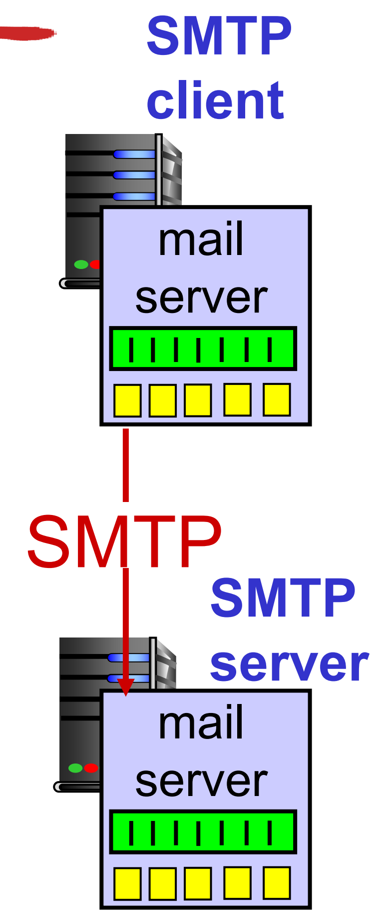
mail servers:

- *message queue* of outgoing (to be sent) mail messages
- *mailbox* contains incoming messages for user
- *SMTP protocol* between mail servers to send/receive email messages
  - Simple Mail Transfer Protocol (SMTP)



Mail Server-to-server : SMTP

- direct transfer: sending server (SMTP client) to receiving server (SMTP server)
- uses ______ transport protocol to reliably transfer email message from client to server, port 25
  - If the server is down, the client tries again later



SMTP: Comparison with HTTP

- HTTP: transfer files from Web server to Web client (browser)
  - **Pull protocol**: HTTP client pulls the information from server
  - TCP connection is initiated by machine that wants to receive
- SMTP: transfer files from one mail server to another mail server
  - **Push protocol**: sending mail server pushes the file to the receiving mail server
  - TCP connection is initiated by machine that wants to send
- SMTP requires message (header & body) to be in **7-bit ASCIIs**
  - Need to encode all binary multimedia data into ASCII before sending over SMTP (No such restriction in HTTP)
    - Image (sender) → 7-bit ASCII text (in SMTP msg) → Image (receiver)
  - This made sense in early 80s when transmission capacity was scarce, so all messages were text – but now it is archaic

Try SMTP interaction for yourself:

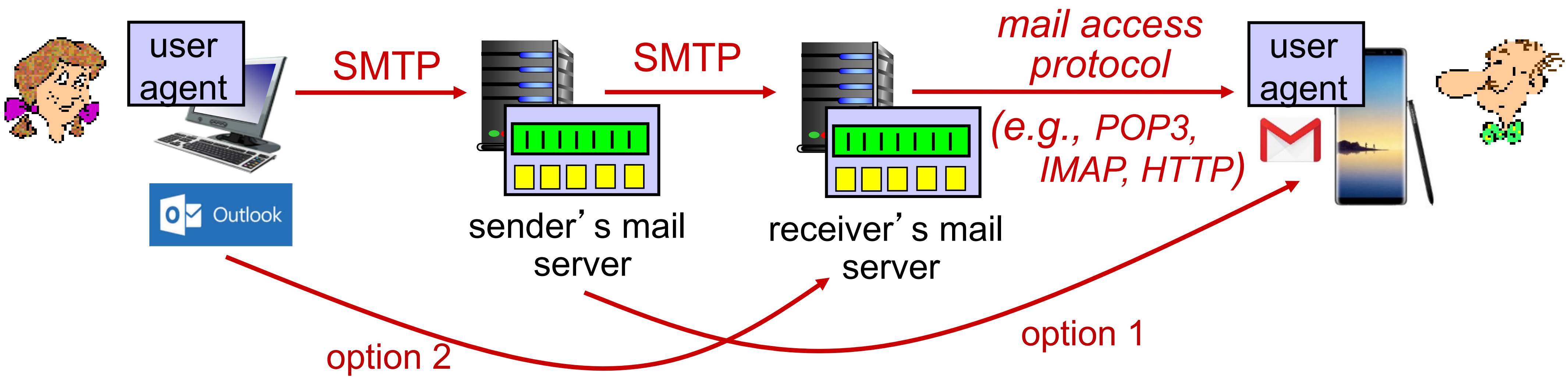
telnet <servername> 25

- see 220 reply from server
- enter HELO, MAIL FROM:, RCPT TO:, DATA, QUIT commands

above lets you send email without using e-mail client (reader)

Note: this will only work if <servername> allows telnet connections to port 25 (this is becoming increasingly rare because of security concerns)

Mail access protocols



- 3 step procedure: UA → mail server → mail server → UA
- Why not take option 1?
- Why not take option 2?
- How does Bob's UA retrieve mail from mailbox?

POP3, IMAP & Web mail

POP3 - Post Office Protocol [RFC 1939]

- Transfer mail from recipient's mail server to user agent (client)
- POP3 uses "download and delete" mode

IMAP - Internet Mail Access Protocol [RFC 1730]

- keeps all messages in one place: at server – doesn't delete

Web-based E-mail

- User agent is Web browser and communicate with mailbox via HTTP (rather than SMTP, POP3, or IMAP)
- The mail server still uses SMTP to send/receive messages to/from other mail servers
- e.g., Gmail, NAVER mail, Gachon E-mail

Chapter 2: outline

2.1 principles of network applications

- app architectures
- app requirements

2.2 Web and HTTP

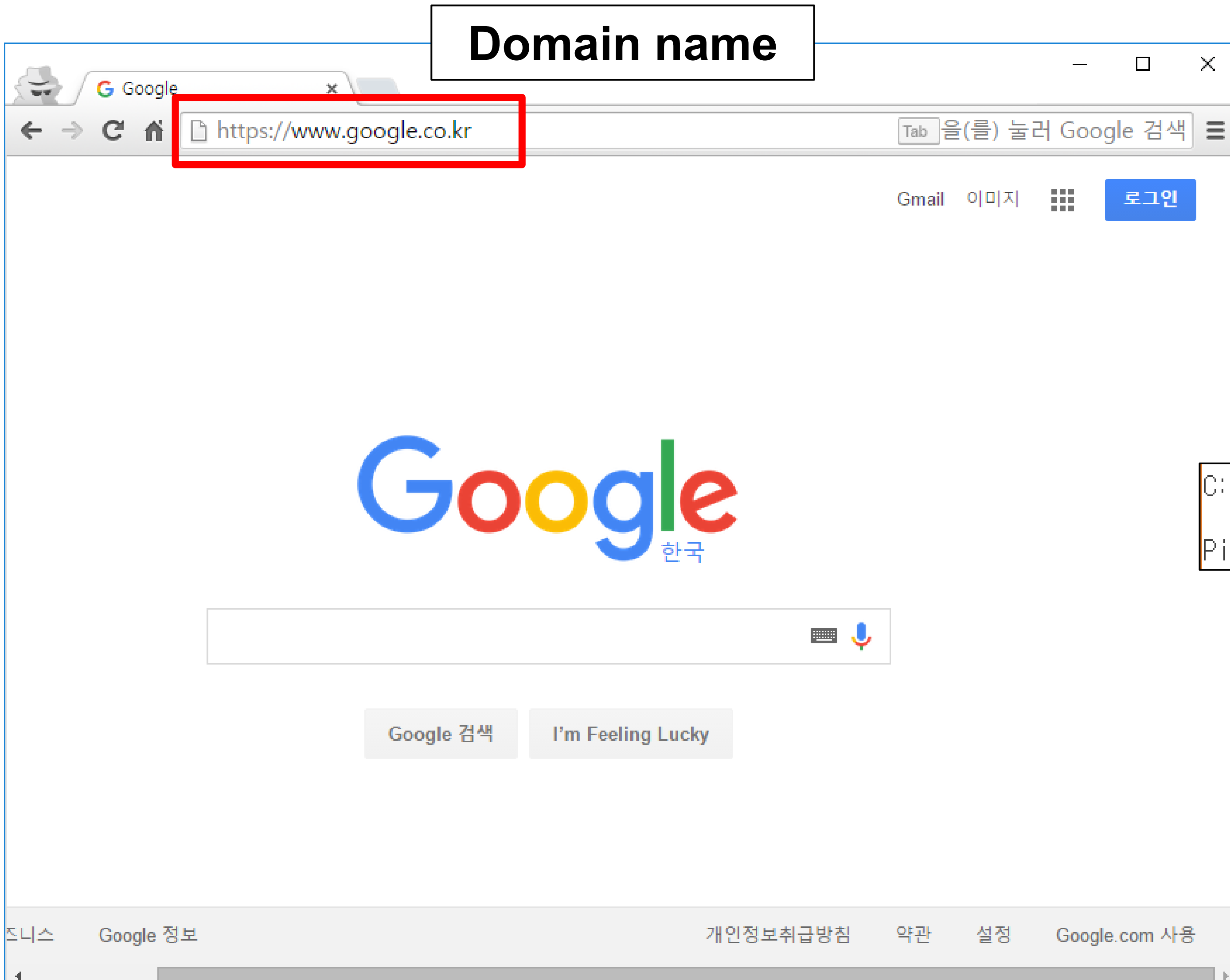
2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.6 Video Streaming and CDN

2.7 Socket Programming



But, Google server's actual address is an **IP address**!

```
C:₩Users₩jyoo>ping google.co.kr

Ping google.co.kr [59.18.46.113] 32바이트 데이터 사용:
```

DNS: domain name system

people: many identifiers:

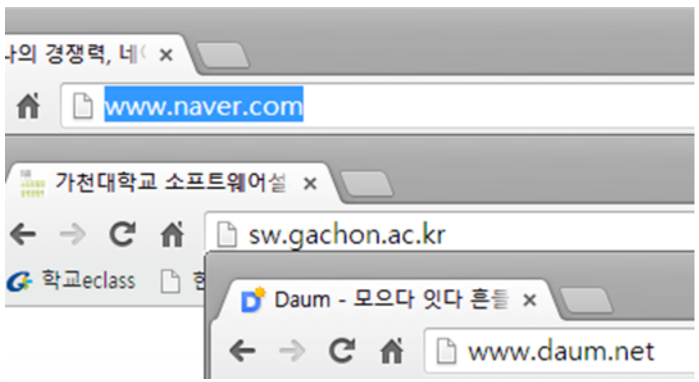
- SSN (≈주민번호), driver's license#, passport #, …

Internet hosts, routers:

- **IP address** or **host name** (e.g., www.yahoo.com)
- People prefer ___ and routers prefer ____

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

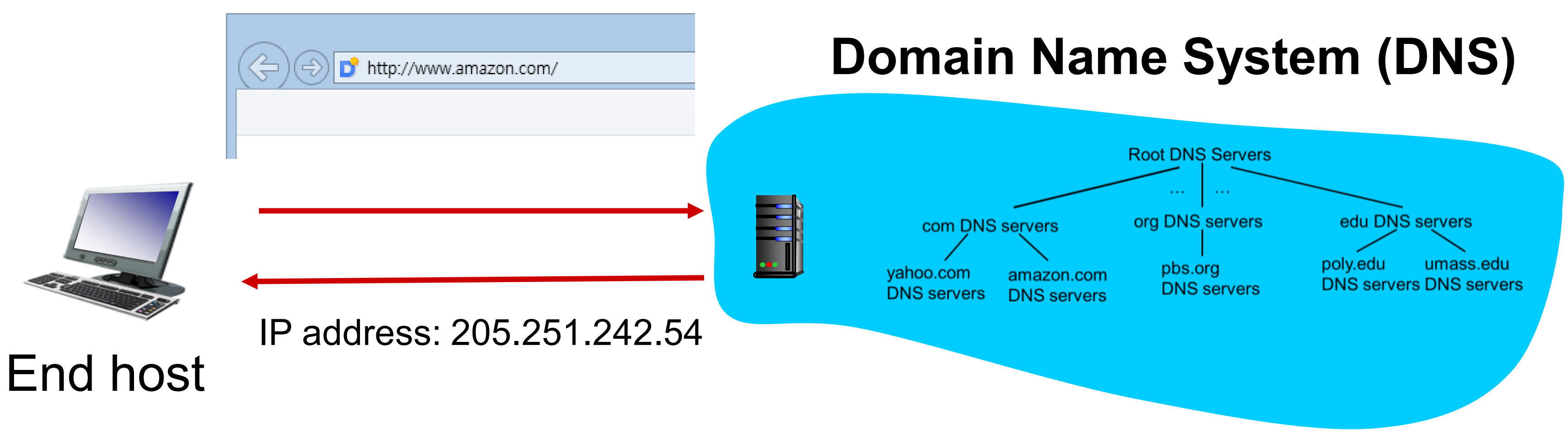
A: DNS



DNS: domain name system

Domain Name System (DNS):

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol:* hosts, name servers employ DNS to *translate* host names into IP addresses



DNS: distributed vs. central

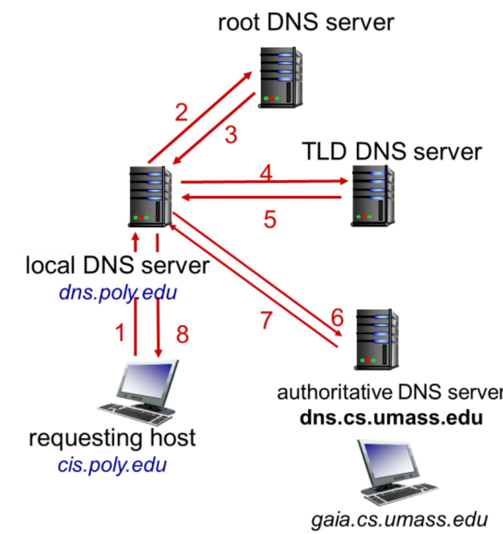
Single centralized server

- Ask one DNS server to translate name to IP address
- Very simple!

Then, why not single centralize DNS server?

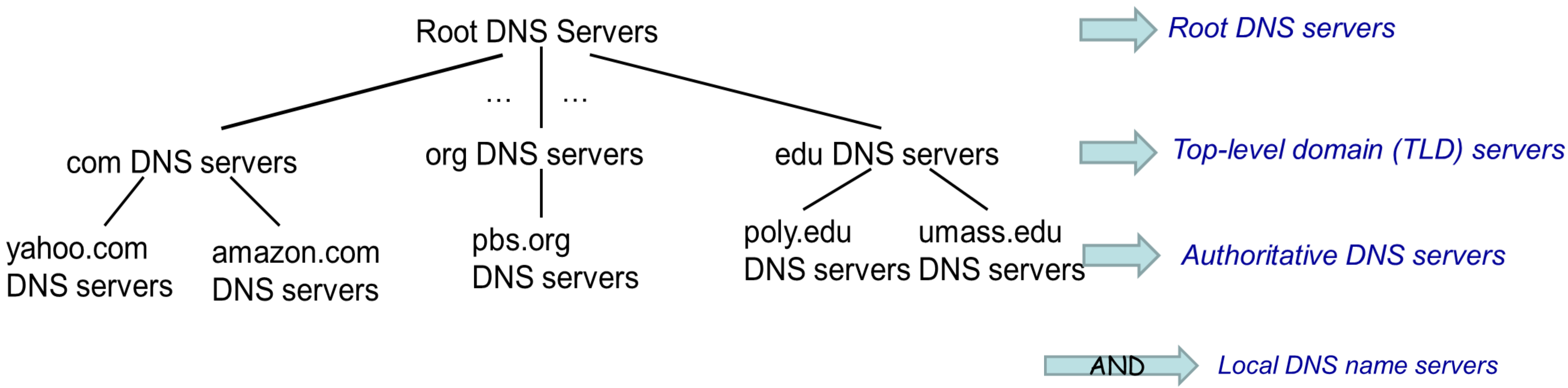
- single point of failure
- traffic volume
- distant centralized database
- maintenance

vs.



A: *doesn't scale!*

DNS: a **distributed, hierarchical** database



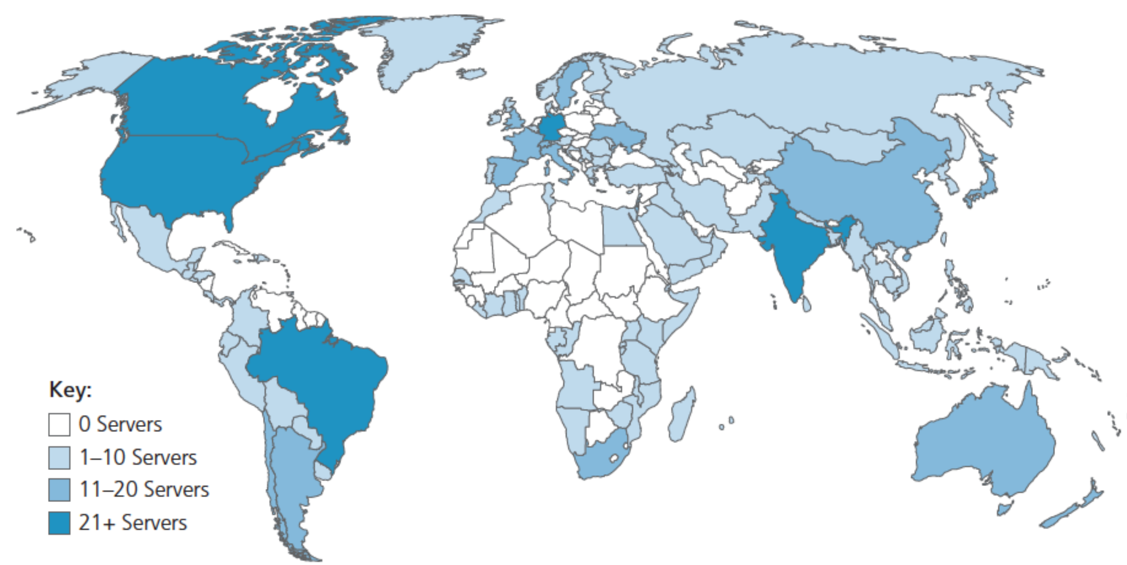
client wants IP address for `www.amazon.com` *; 1st approx:*

- client queries **root server** to find TLD (`.com`) DNS server
- client queries `.com` DNS server to get authoritative (`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
- *incredibly important* Internet function
  - Internet couldn't function without it!
- 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)



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.: `kr`, `uk`, `fr`, `ca`, `jp`

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
- e.g., google, NAVER, Gachon

Local DNS name server

- each ISP (residential ISP, company, university) has one (or more)
  - also called "default name server"

```
ipconfig/all
```

```
IPv4 주소 . . . . . . . . . . : 121.135.107.150(기본 설정)
서브넷 마스크 . . . . . . . . : 255.255.255.0
임대 시작 날짜. . . . . . . . : 2014년 9월 21일 일요일 오후 1:59:47
임대 만료 날짜. . . . . . . . : 2014년 9월 21일 일요일 오후 11:11:20
기본 게이트웨이 . . . . . . . : 121.135.107.1
DHCP 서버 . . . . . . . . . . : 121.137.7.58
DHCPv6 IAID . . . . . . . . . : 199754650
DHCPv6 클라이언트 DUID. . . . : 00-01-00-01-18-39-90-22-E8-03-9A-66-87-44
DNS 서버. . . . . . . . . . . : 168.126.63.1
                                168.126.63.2
Tcpip를 통한 NetBIOS. . . . . : 사용
```

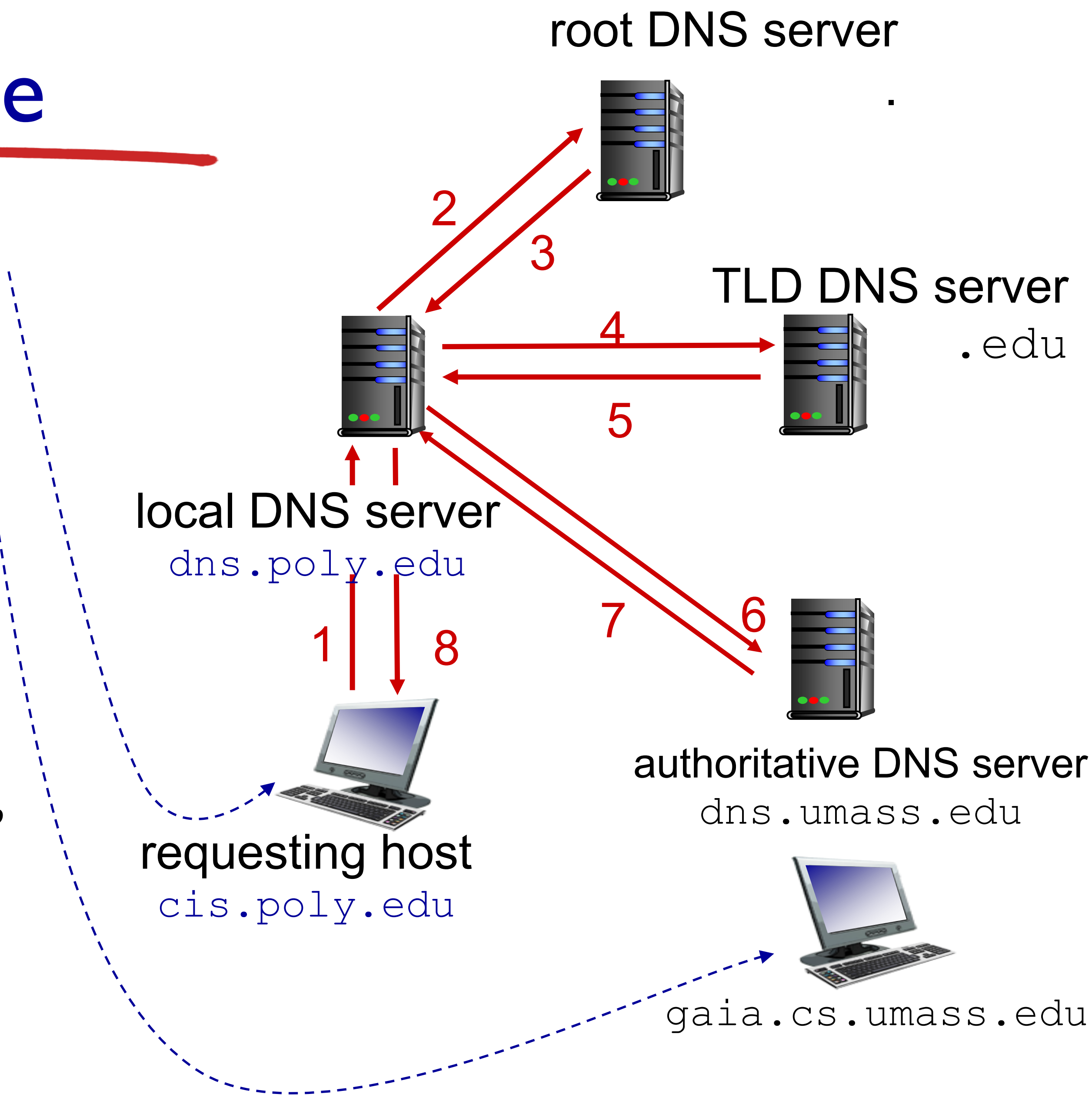
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

DNS name resolution example

- host at `cis.poly.edu` wants IP address for `gaia.cs.umass.edu`

iterated query:

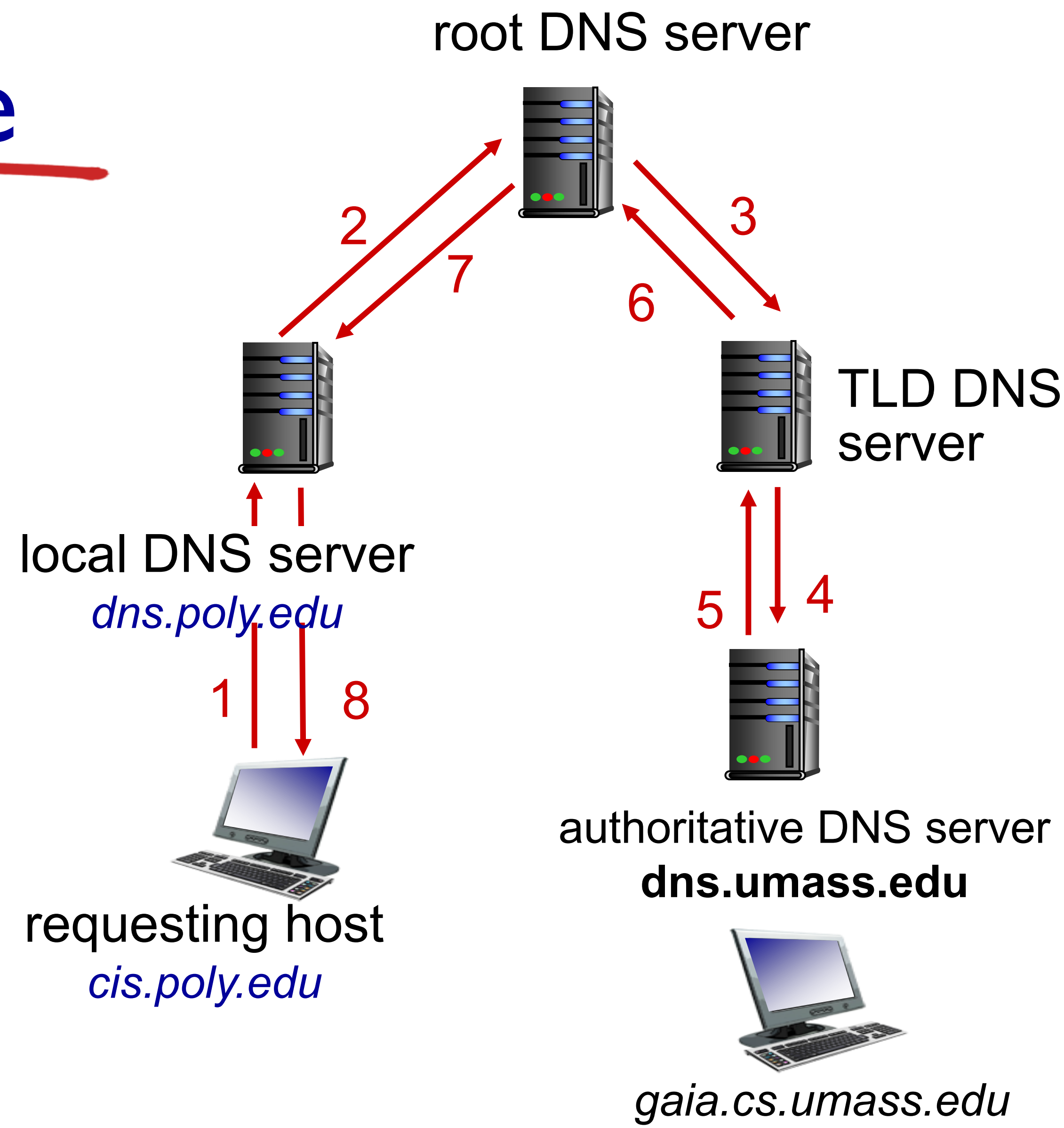
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example

recursive query:

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



DNS: caching, updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time-to-live (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

Question

How can you watch a YouTube video without buffering?

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 electronic mail
- SMTP, POP3, IMAP

2.4 DNS

2.6 Video Streaming and CDN

2.7 Socket Programming

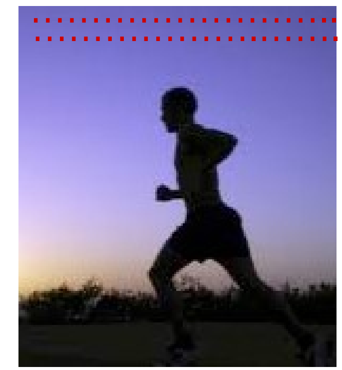
Video Streaming and CDNs: context

- 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 (or fps)
- 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*

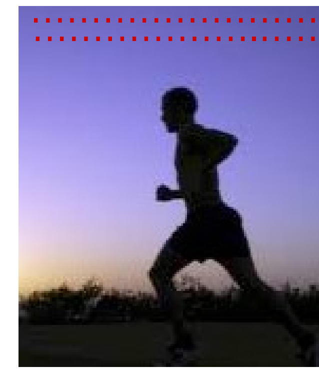
temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

frame *i+1*

Multimedia: video

- 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)



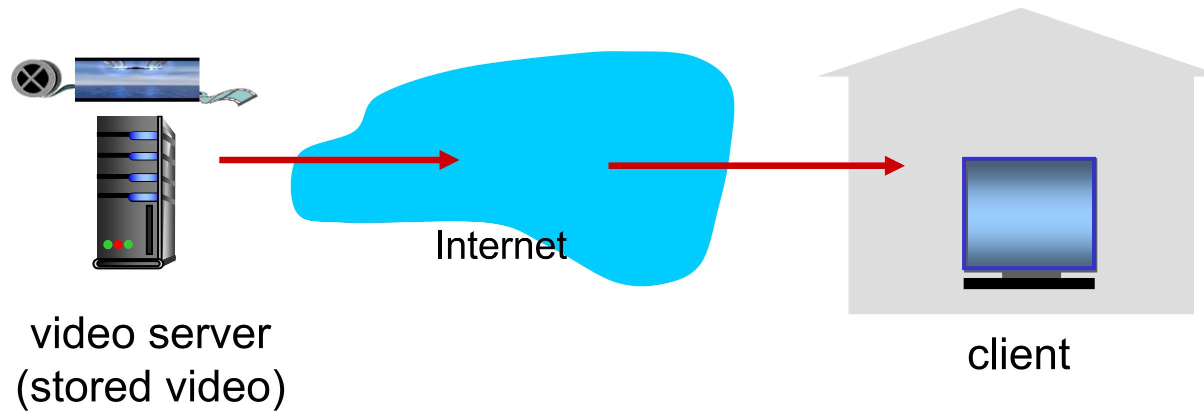
frame *i*

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

frame *i+1*

Streaming stored video

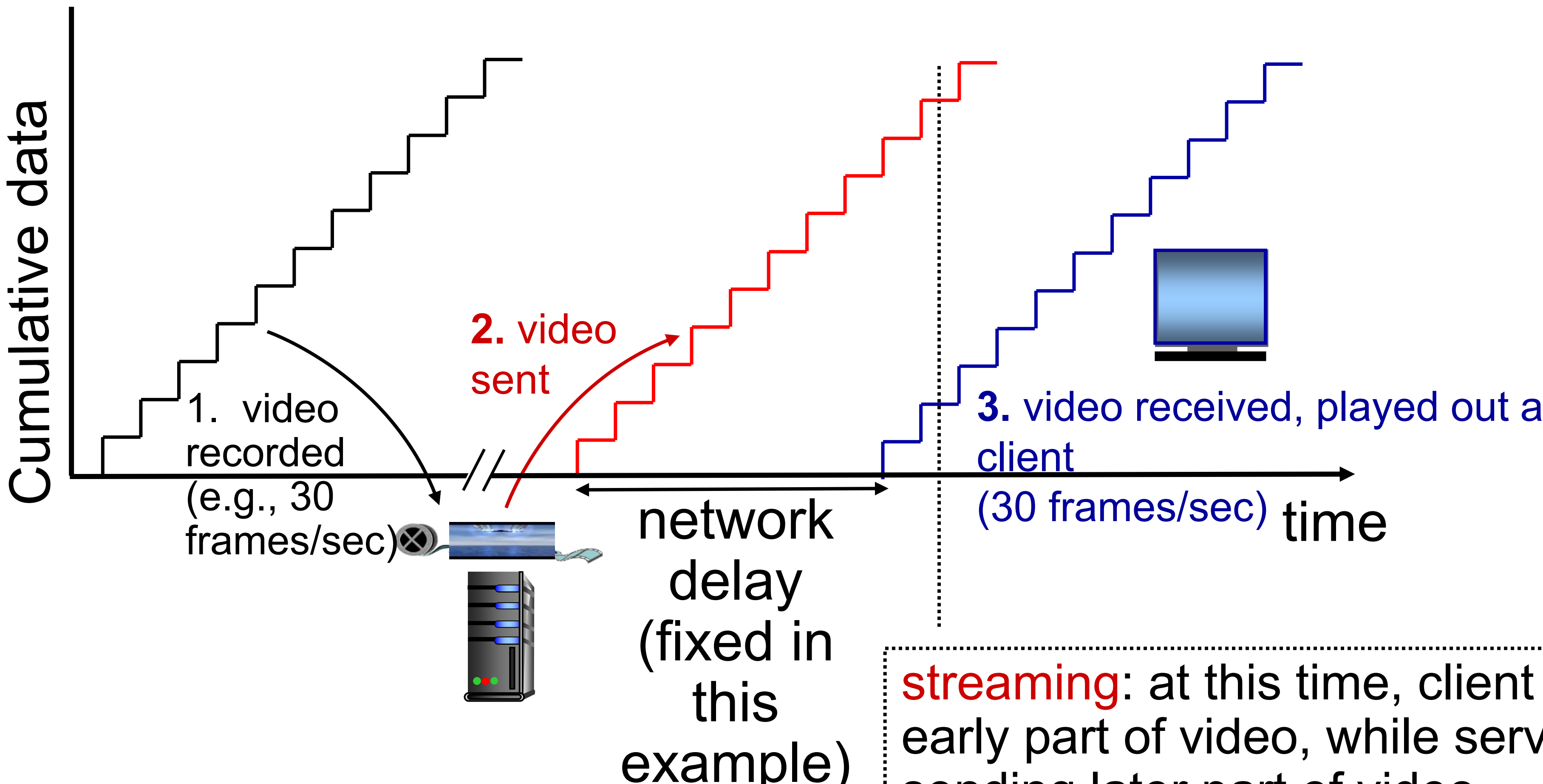
simple scenario:



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

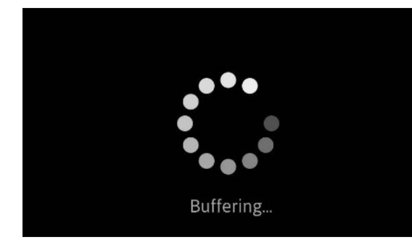
Streaming stored Video



streaming: at this time, client playing out early part of video, while server still sending later part of video

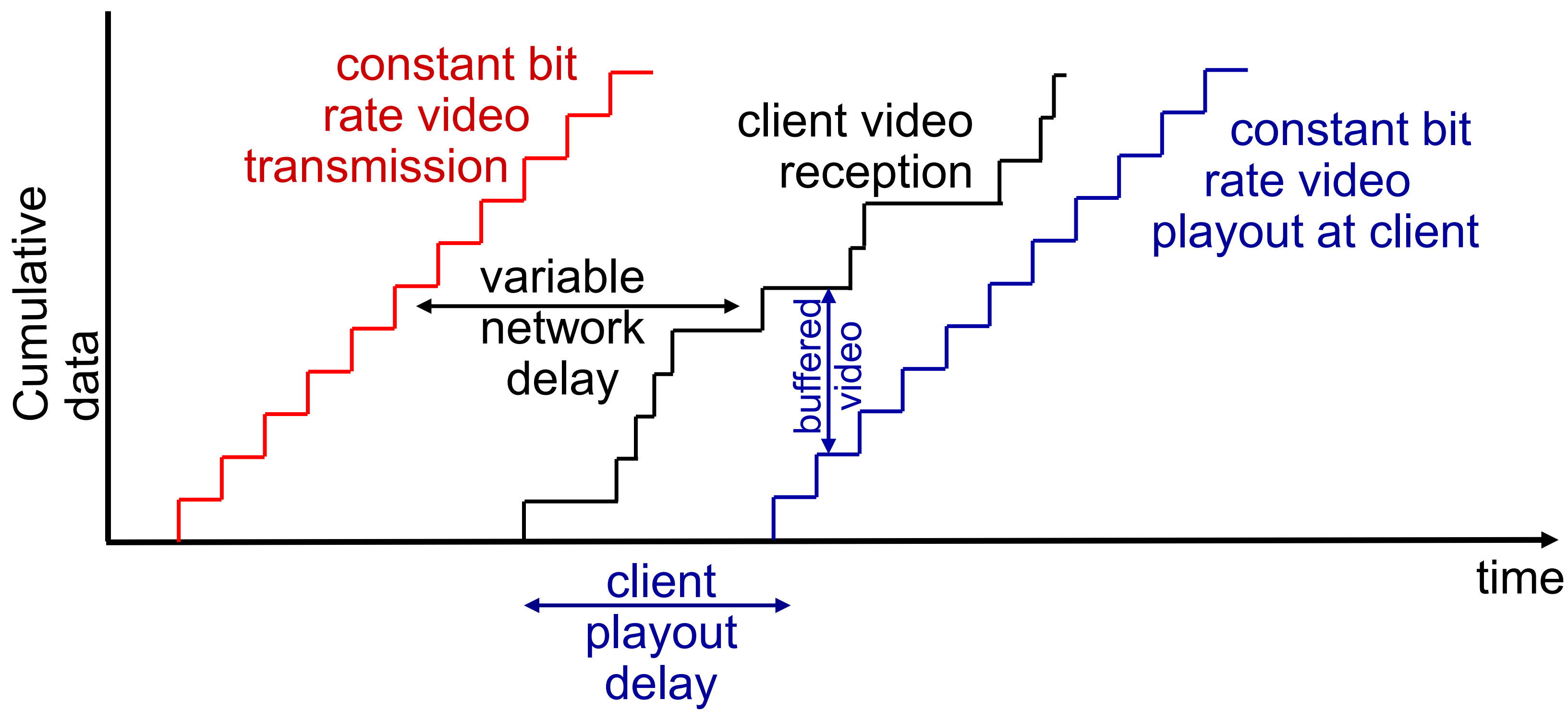
Streaming stored Video: challenges

- 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

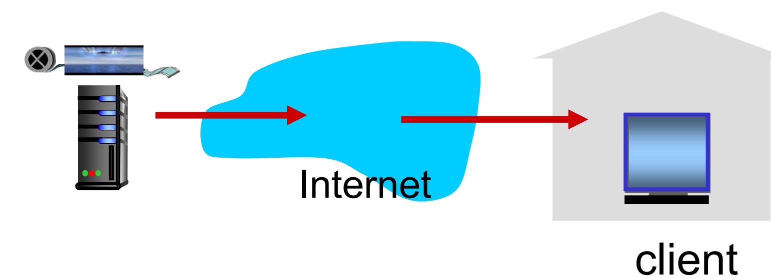
Streaming stored video: playout buffering



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

Streaming multimedia: DASH

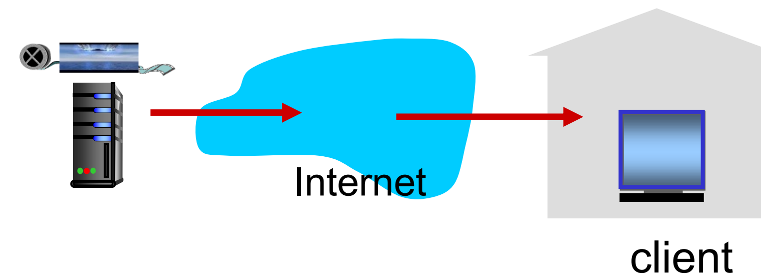
- *DASH: D*ynamic, *A*daptive *S*treaming over *H*TTP
- *server:*
  - divides video file into multiple chunks
  - each chunk stored, encoded at different rates
  - *manifest file:* provides URLs for different chunks



- *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 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

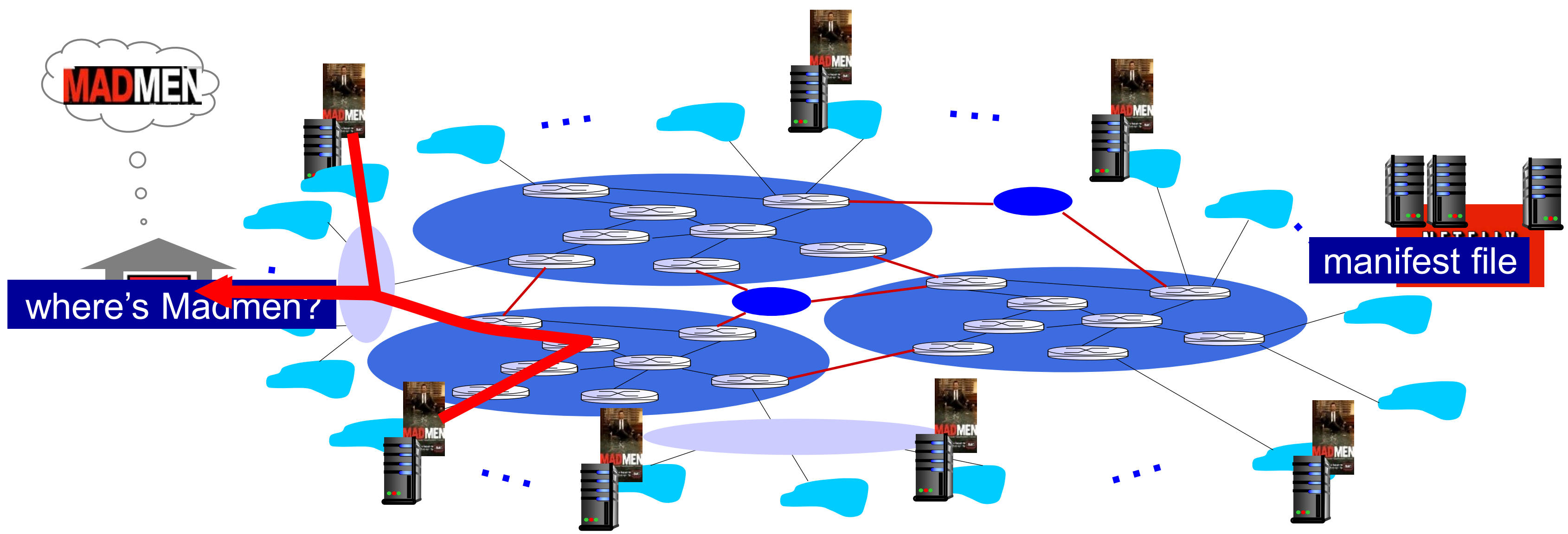
….quite simply: this solution *doesn't scale*

Content distribution networks (CDNs)

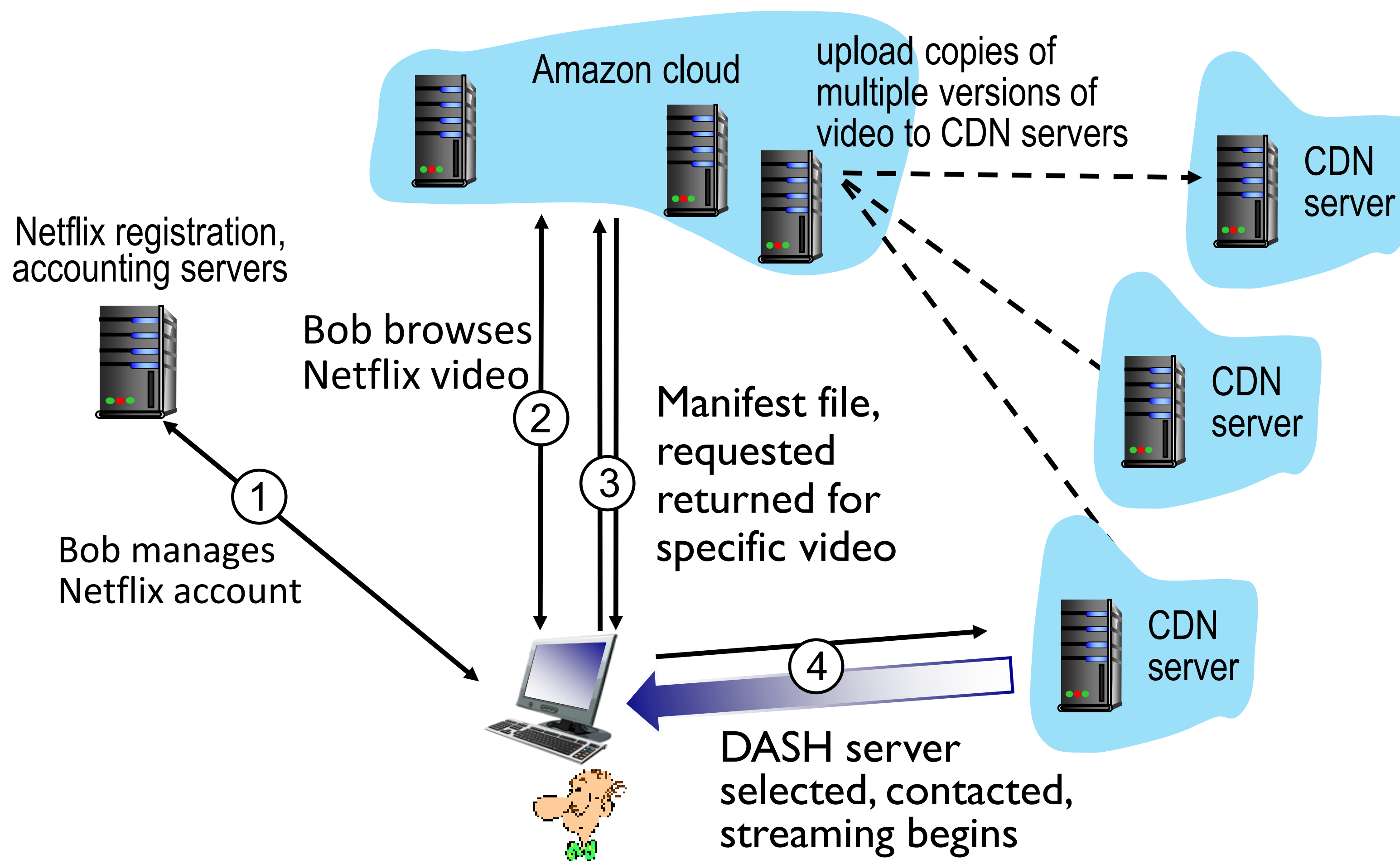
- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?
- option 2: store/serve multiple copies of videos at multiple geographically distributed sites *(CDN)*
  - *enter deep:* push CDN servers deep into many access networks
    - close to users
    - Akamai: 240,000 servers deployed in more than 120 countries (2015)
  - *bring home:* smaller number (10's) of larger clusters 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



Case study: Netflix



Chapter 2: outline

2.1 principles of network applications

- app architectures
- app requirements

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.6 Video Streaming and CDN

2.7 Socket Programming

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:

- centralized vs. decentralized
- stateless vs. stateful
- scalability
- reliable vs. unreliable message transfer
- "complexity at network edge"