CS 305: Computer Networks

Fall 2022

Lecture 4：Application Layer

Ming Tang

Department of Computer Science and Engineering
Southern University of Science and Technology (SUSTech)

HTTP Outline

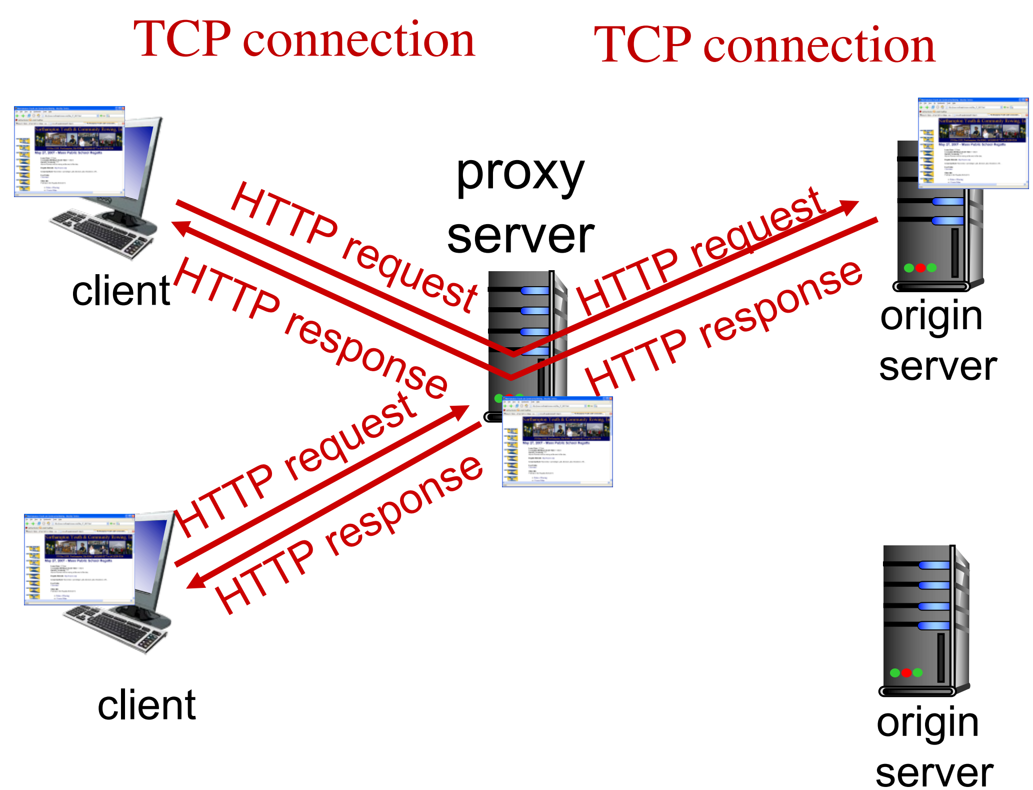
- HTTP Overview
  - HTTP runs over TCP
  - HTTP is stateless
  - Persistent and non-persistent connection
- Request and response messages
- Cookies
- Web caching

Web caches: proxy（代理）server

goal: satisfy client request without involving origin server

Browser sends all HTTP requests to cache

- object in cache: cache returns object
- else cache requests object from origin server, then returns object to client



More about Web caching

- Cache (Proxy server) acts as both client and server
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)

Why Web caching?

- reduce response time for client request (bottleneck bandwidth)
- reduce traffic on an institution's access link
- Internet dense with caches: enables "poor" content providers to effectively deliver content (so too does P2P file sharing)

Caching example:

Assumptions:

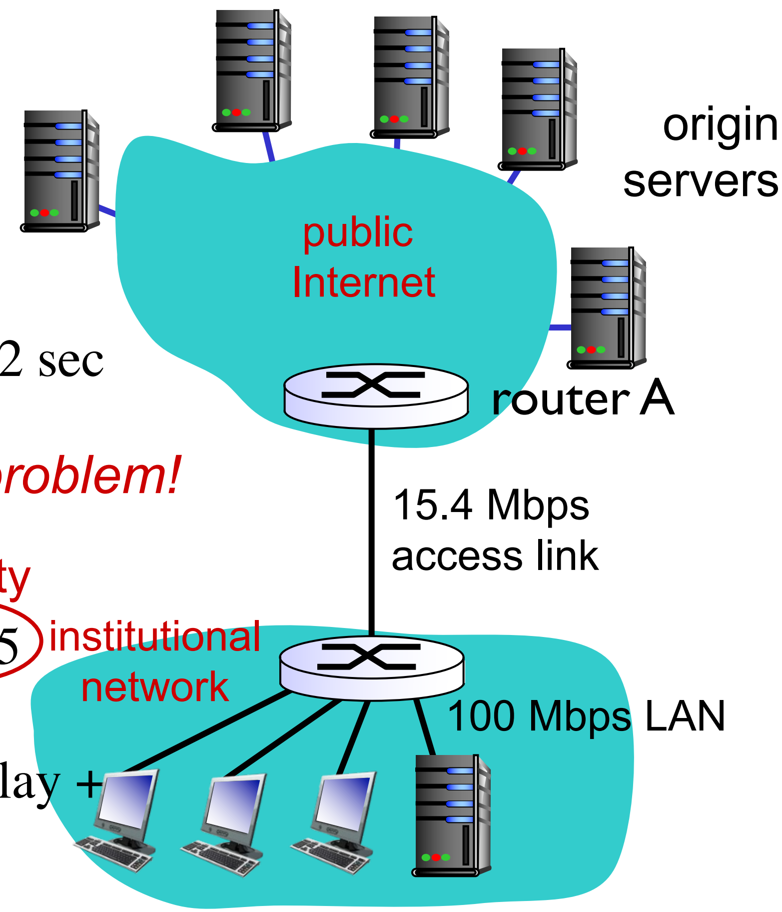
- avg object size: 1M bits
- avg request rate from browsers to origin servers:15 requests/sec
- avg data rate to all browsers: 15 Mbps
- RTT from router A to any origin server: 2 sec
  → "Internet delay"
- access link rate: 15.4 Mbps

problem!

Consequences:

Traffic intensity

- LAN utilization: 15Mbps/100Mbps=0.15
- access link utilization =15/15.4= 0.974
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + milliseconds



Caching example: fatter access link

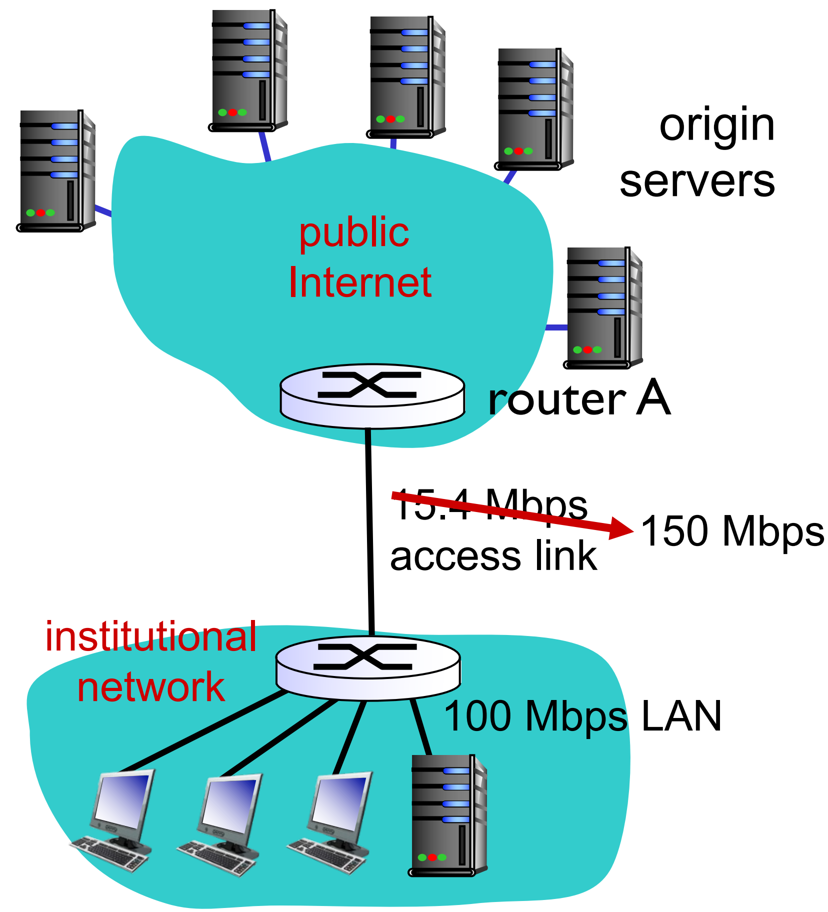
assumptions:

- avg object size: 1M bits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 15 Mbps
- RTT from router A to any origin server: 2 sec
- access link rate: 15.4 Mbps

consequences:

- LAN utilization: 0.15
- access link utilization = ~~0.974~~ 0.1
- total delay = Internet delay + access delay + LAN delay

  = 2 sec + ~~minutes~~ + milliseconds
  milliseconds



Cost: increased access link speed (not cheap!)

Caching example: install local cache

assumptions:

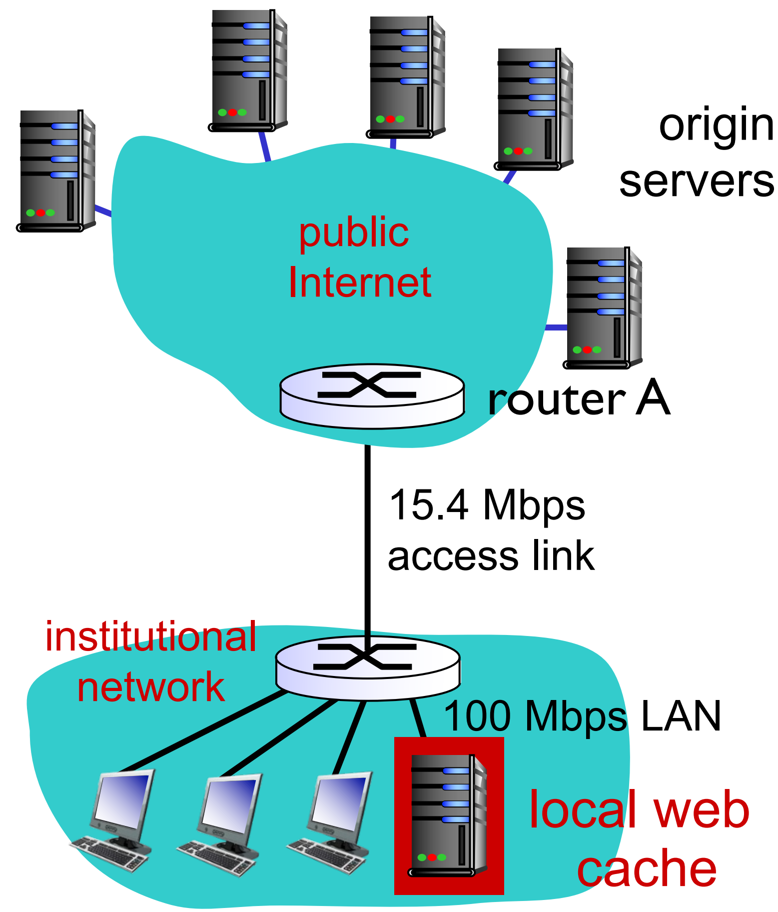
- avg object size: 1M bits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 15 Mbps
- RTT from router A to any origin server: 2 sec
- access link rate: 15.4 Mbps

consequences:

- LAN utilization: 0.15
- access link utilization = ?
- total delay = ?

How to compute link utilization, delay?

Cost: web cache (cheap!)



Hit rates: the fraction of requests that are satisfied by a cache. Typically, 0.2—0.7.

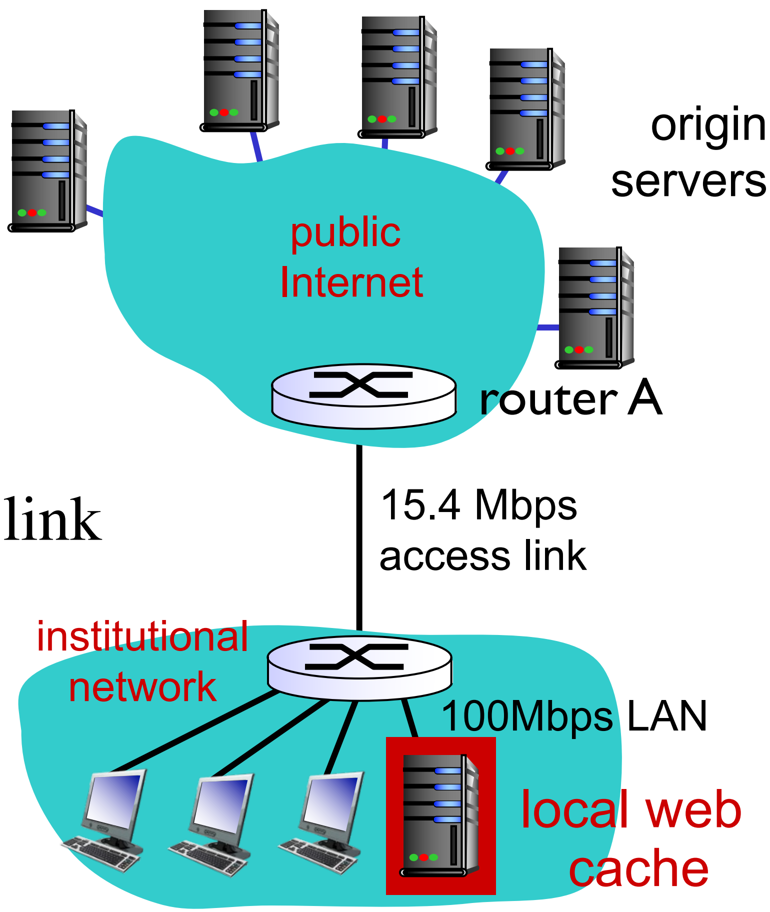
Caching example: install local cache

Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin

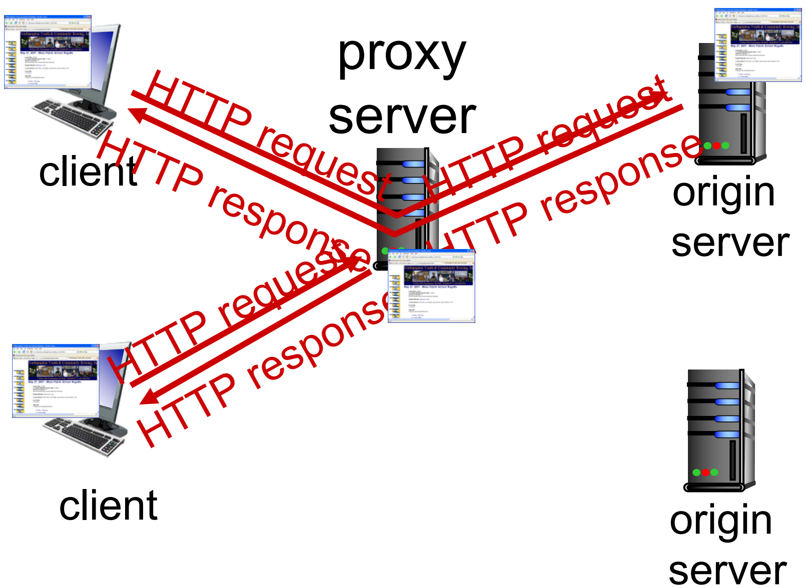
- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link = 0.6*15 Mbps = 9 Mbps
  - utilization = 9/15.4 = 0.58

- Average delay
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs
  - less than with 150 Mbps link (and cheaper too!)



Typically, a traffic intensity less than 0.8 corresponds to a small delay, say, tens of milliseconds

Conditional GET



The copy of an object residing in the cache may be out-of-date:

Conditional GET

- GET method
- If-Modified-Since

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 9 Sep 2015 09:23:24
```

Goal: allows a cache to verify that its objects are up to date

- don't send object if cache has up-to-date cached version
- no object transmission delay
- lower link utilization

Conditional GET

When a browser requests an object via proxy cache:

- *Proxy cache:* specify date of cached copy in HTTP request

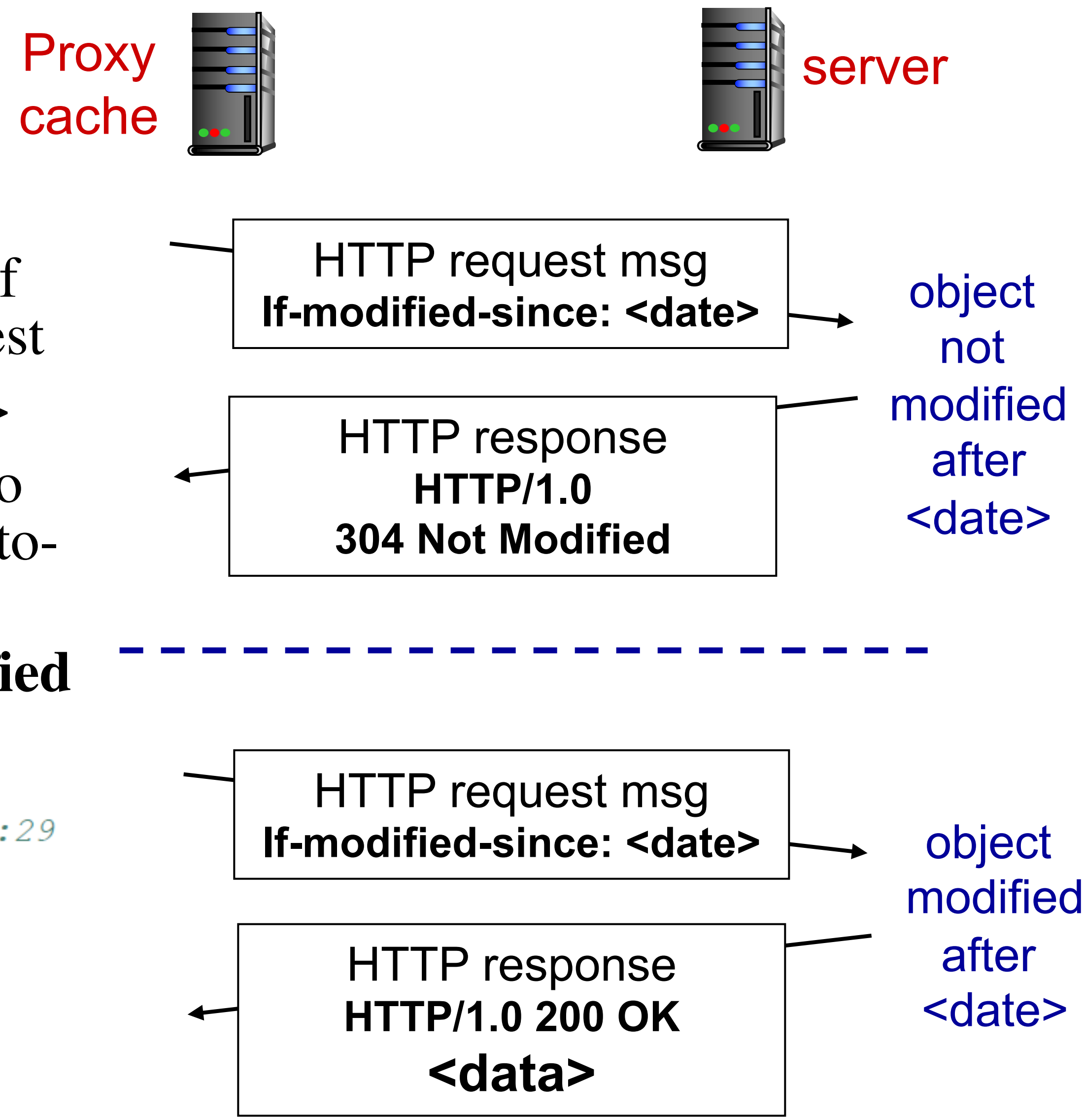
  If-modified-since: <date>

- *Server:* response contains no object if cached copy is up-to-date:

  HTTP/1.0 304 Not Modified

```
HTTP/1.1 304 Not Modified
Date: Sat, 10 Oct 2015 15:39:29
Server: Apache/1.3.0 (Unix)

(empty entity body)
```



HTTP Summary

- HTTP Overview
  - HTTP runs over TCP
  - HTTP is stateless
  - Persistent and non-persistent connection
- Request and response messages
- Cookies
- Web caching

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks

2.7 socket programming with UDP and TCP

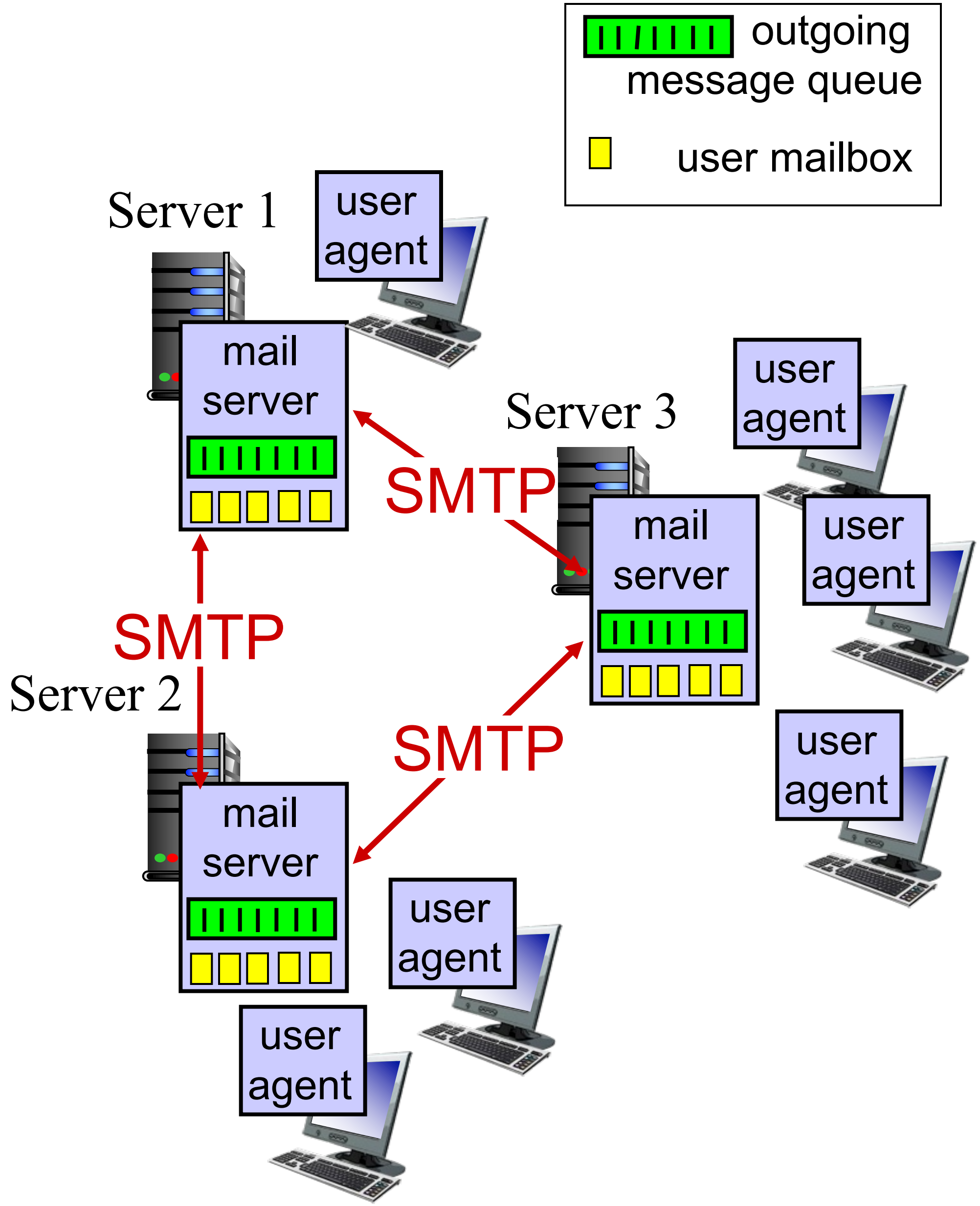
Electronic Mail Overview

- Overview
  - Main components
  - Alice sends an email to Bob
- SMTP
- Mail Message Format
- Mail Access Protocol
  - POP3
  - IMAP
  - HTTP: Web-based Email

Electronic mail

Three major components:

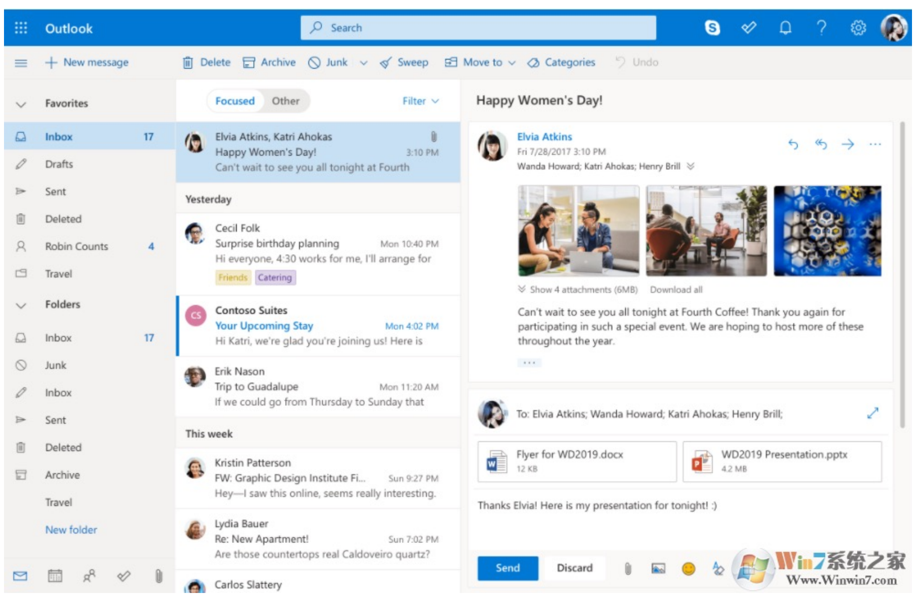
- user agents
- mail servers
- simple mail transfer protocol (SMTP): use TCP

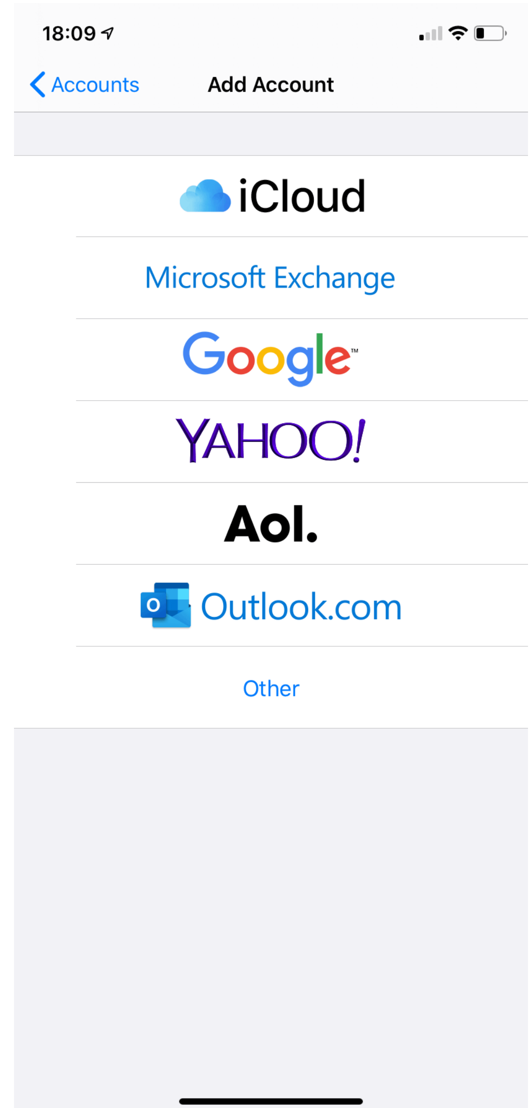


Electronic mail: User Agent

User Agent

- a.k.a. "mail reader"
- Allow users to read, reply to, forward, save and compose messages
- e.g., Outlook, iPhone mail client



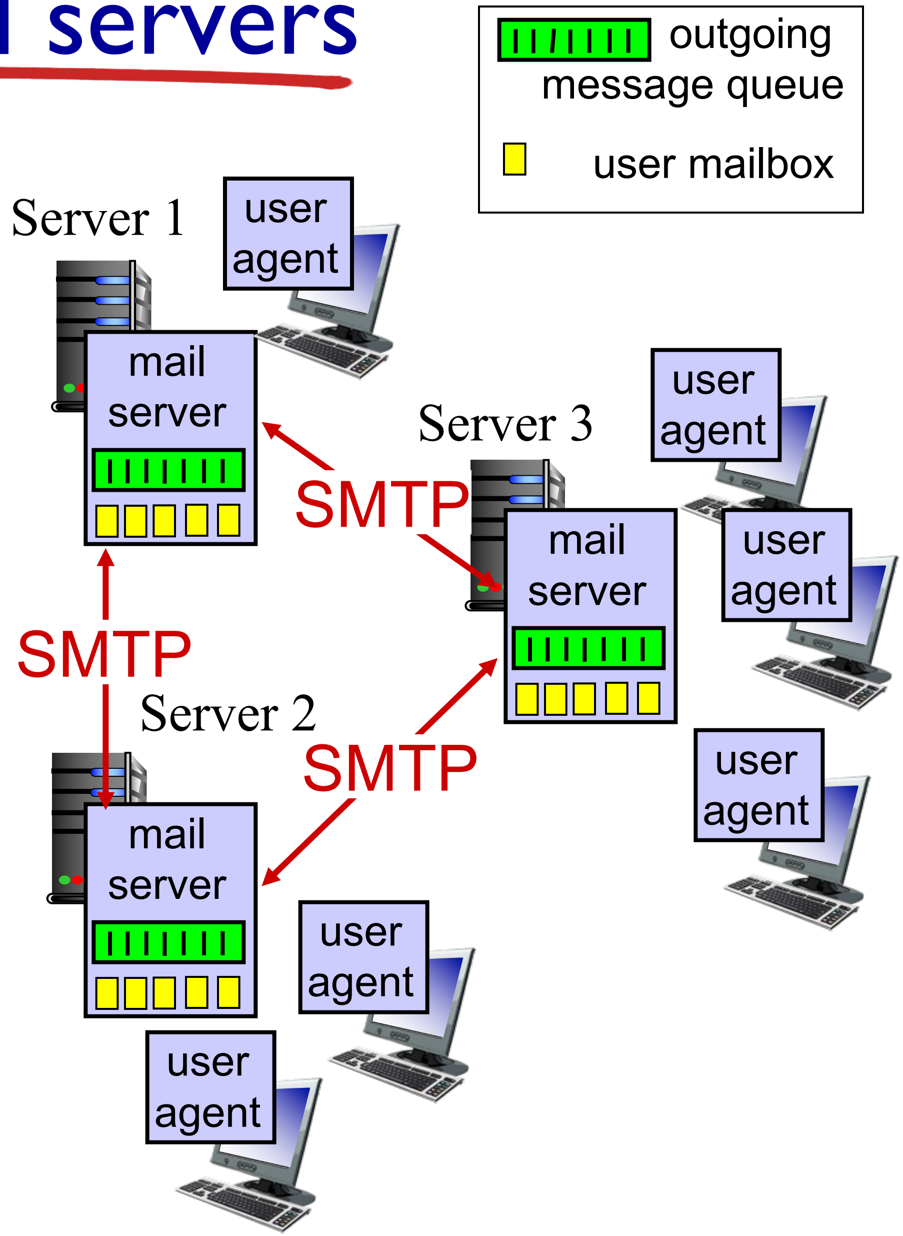


Electronic mail: mail servers

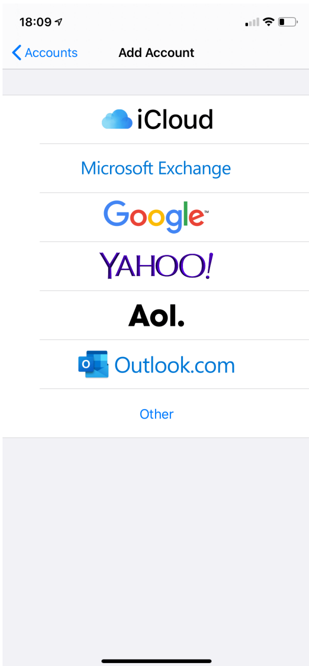
Mail servers:

- Always-on hosts
- *User mailbox* contains outgoing, incoming messages
- *Message queue* of outgoing (to be sent) mail messages
- *Simple Mail Transfer Protocol (SMTP)* <u>between mail servers</u> to send email messages
    - client: sending mail server
    - "server": receiving mail server

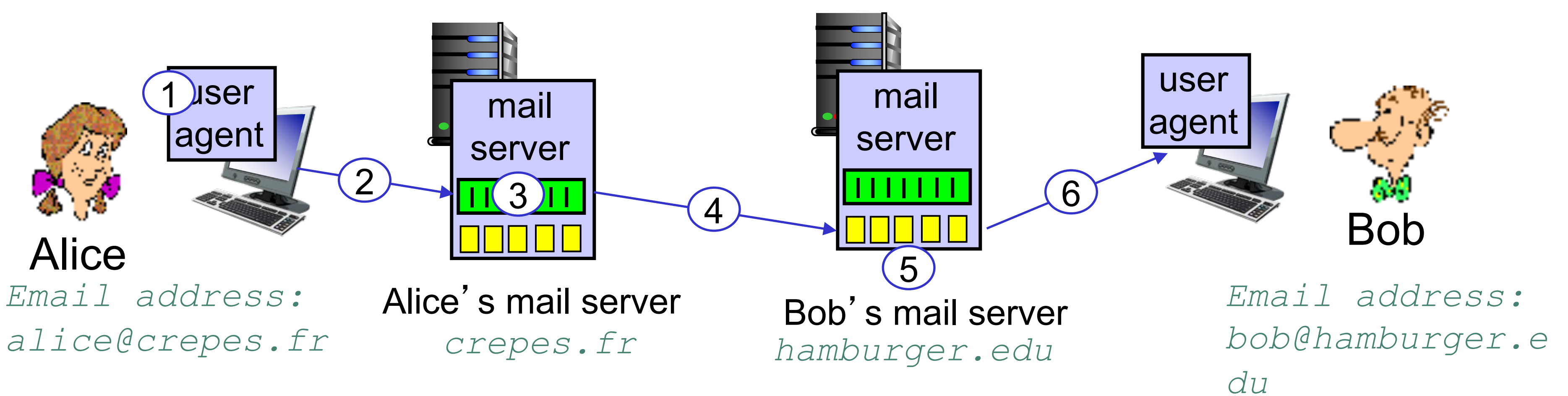
  Both client and server sides of SMTP run on mail server.



Scenario: Alice sends message to Bob



1) Alice uses user agent to compose message "to" `bob@hamburger.edu`
2) Alice's user agent 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

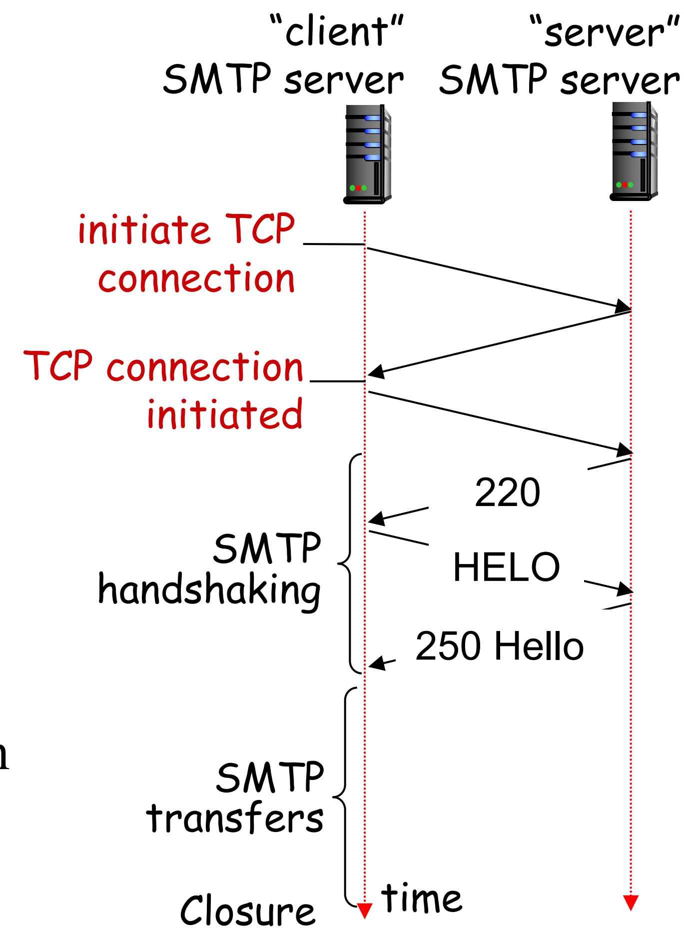


Electronic Mail Overview

- Overview
  - Main components
  - Alice sends an email to Bob
- SMTP
- Mail Message Format
- Mail Access Protocol
  - POP3
  - IMAP
  - HTTP: Web-based Email

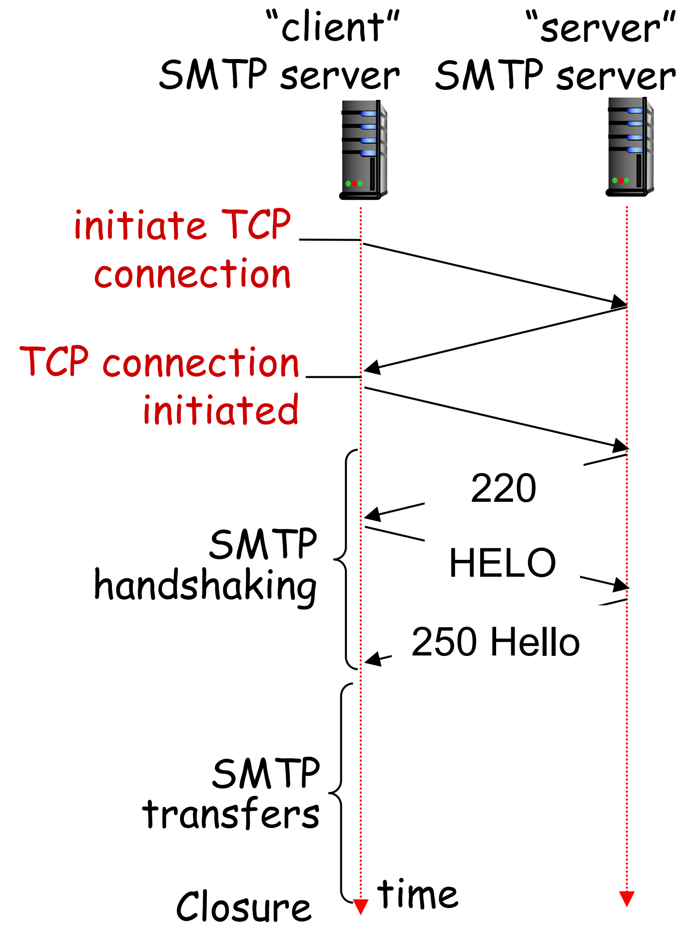
Electronic Mail: SMTP [RFC 2821]

- Uses TCP to reliably transfer email message from client to server, port 25
  - If fail, new attempt after a while (e.g., 30 minutes)
- Direct transfer: sending server to receiving server
  - Direct connection, no intermediate mail server
- Three phases of transfer
  - handshaking (greeting): indicate email address
  - transfer of messages: persistent connection
  - closure



Electronic Mail: SMTP [RFC 2821]

- Two types of messages (like HTTP)
  - commands: text
  - response: status code and phrase
- Entire messages (header & body) must be in ASCII
  - Binary multimedia data → ASCII
  - For HTTP, headers are encoded with ASCII



Sample SMTP interaction

The following are exactly the lines the client (C: crepes.fr) and server (S: hamburger.edu) send after they establishing TCP connections.

commands
response (status code + phrase)

SMTP handshaking

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
```

SMTP transfers

```
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
```

Repeat to send multiple messages

Closure

```
C: QUIT
S: 221 hamburger.edu closing connection
```

SMTP: Closing Observations

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in ASCII
- SMTP server uses CRLF.CRLF to determine end of message

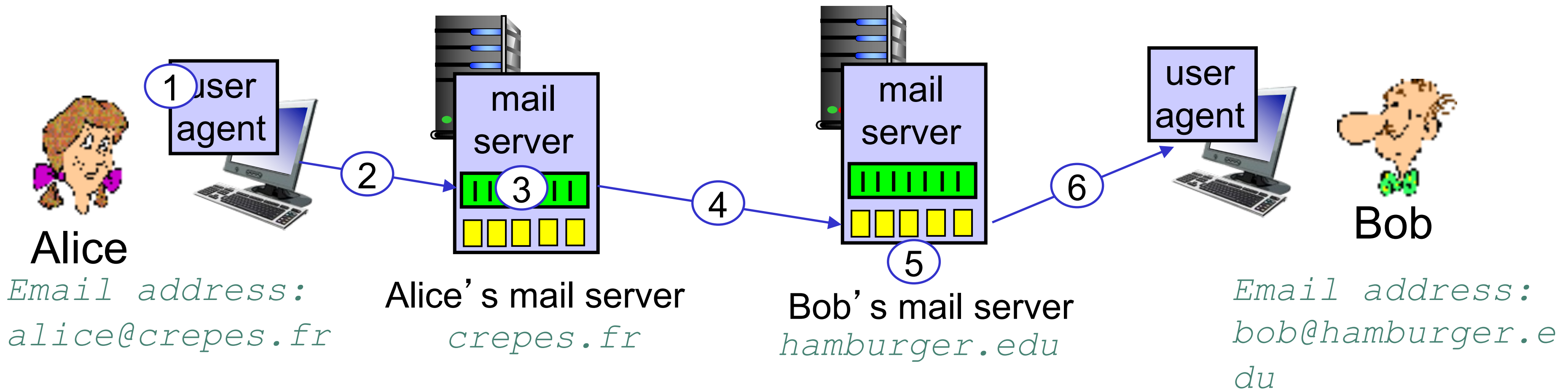
Comparison with HTTP:

- HTTP: pull
- SMTP: push

- HTTP: ASCII in header
- SMTP: ASCII in header and body

- HTTP: each object encapsulated in its own response message
- SMTP: multiple objects sent in one message

Alternative Choices?



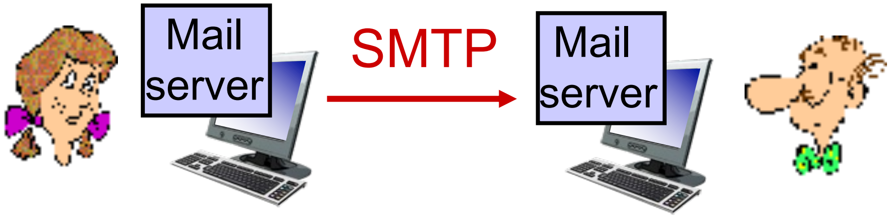
Can we have mail servers directly on user's local PC?

NO

Can we let Alice send to Bob's mail server directly?

NO!

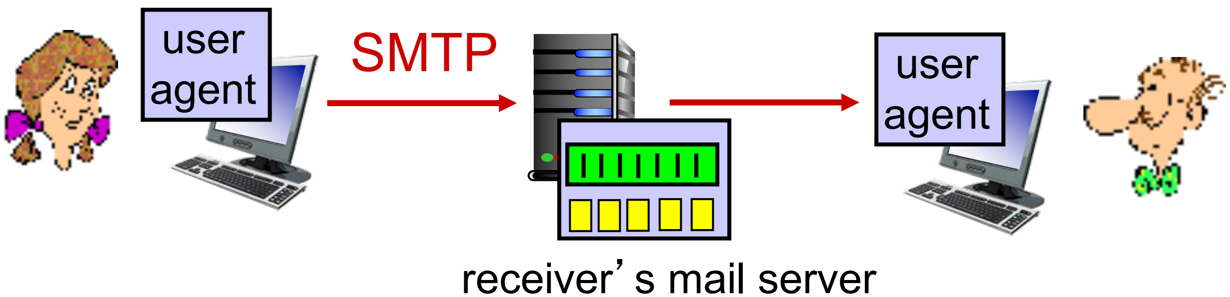
Alternative Choices?



Why not having mail servers directly on user's local PC?

- Recall that a mail server manages mailboxes and runs the client and server sides of SMTP.
- If Bob's mail server were to reside on his local PC, then Bob's PC would have to remain always on in order to receive new mail.

Alternative Choices?



Why not letting Alice send to Bob's mail server directly?

- Bob's mail sever may fail; need to repeatedly send the message until success.

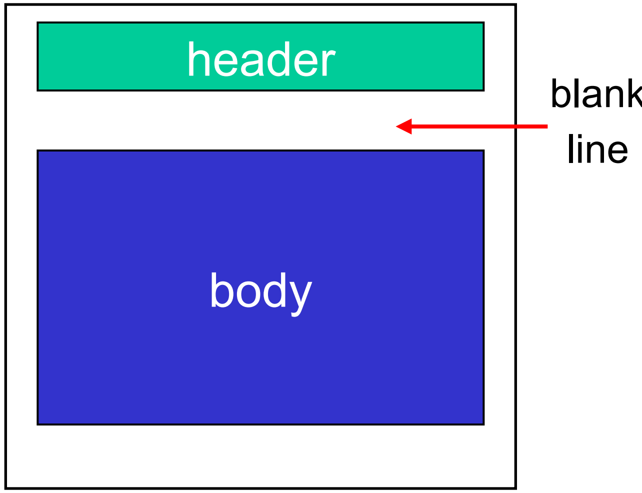
Electronic Mail Overview

- Overview
  - Main components
  - Alice sends an email to Bob
- SMTP
- Mail Message Format
- Mail Access Protocol
  - POP3
  - IMAP
  - HTTP: Web-based Email

Mail message format

Mail message format (RFC 2822) defines *syntax* for e-mail message itself (like HTML)

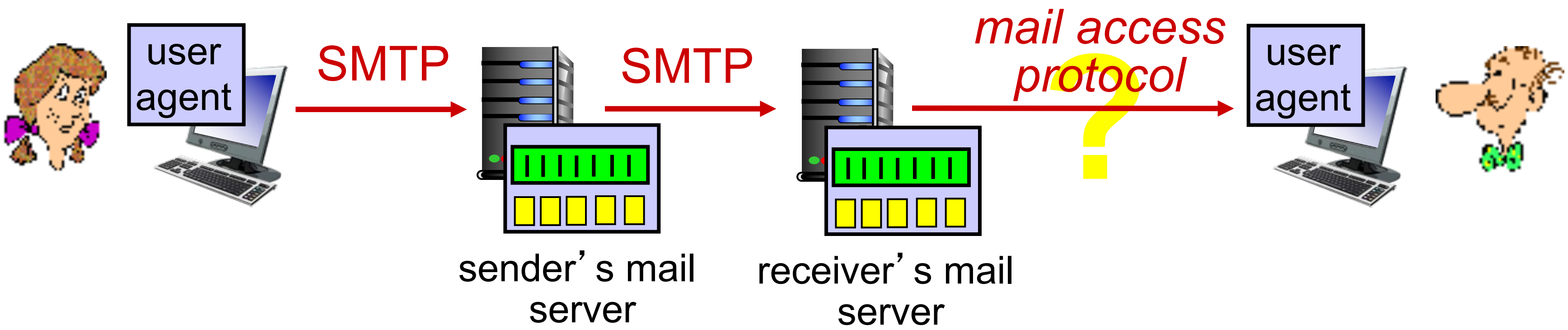
- Header lines, e.g.,
  - To:
  - From:
  - Subject:
  - these lines are part of the message itself, different from SMTP MAIL FROM:, RCPT TO: commands!
- Body: the "message" , ASCII characters only



Electronic Mail Overview

- Overview
  - Main components
  - Alice sends an email to Bob
- SMTP
- Mail Message Format
- Mail Access Protocol
  - POP3
  - IMAP
  - HTTP: Web-based Email

Mail access protocols



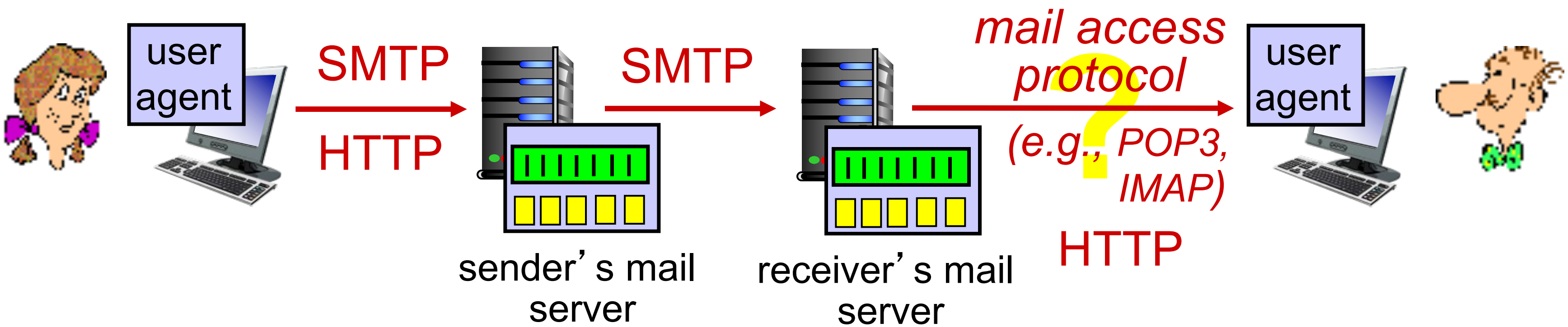
SMTP: delivery to receiver's server

Mail access protocols: How does Bob obtain his message?

SMTP?

No! Because obtaining message is a pull operation.

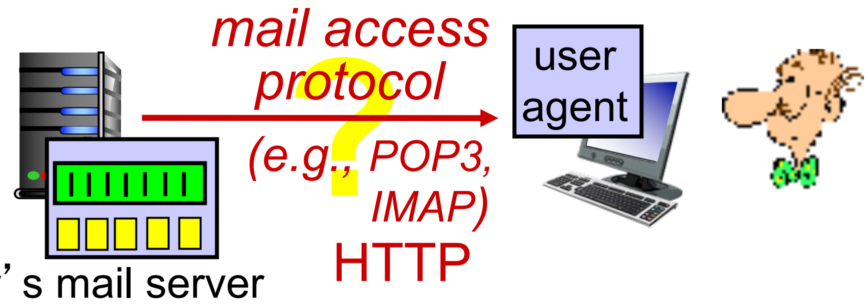
Mail access protocols



Mail access protocol: retrieval from server

- POP3: Post Office Protocol 3: authorization, download
  - TCP, port 110
- IMAP: Internet Mail Access Protocol: more features, including maintain folders, keep user state
- HTTP: gmail, Hotmail, Yahoo! Mail, etc.

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**

Update phase

- After **Quit**, the mail server deletes the messages marked as deletion

```
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
```

Download-and-delete mode

Download-and-keep mode？

POP3 (more) and IMAP

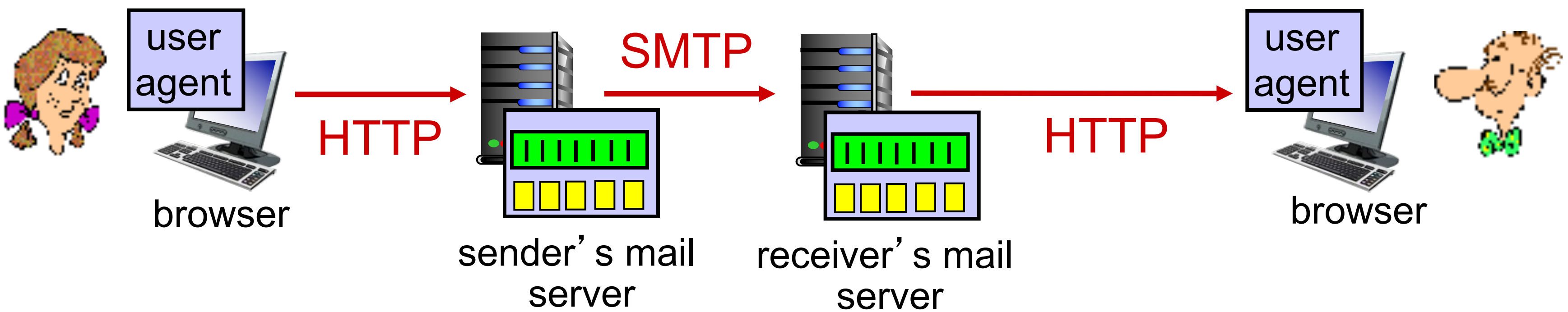
More about POP3

- previous example uses POP3 "download and delete" mode
  - Bob cannot re-read e-mail if he changes client
- POP3 "download-and-keep": reread the message from different machines
- POP3 is stateless across sessions

IMAP

- Maintain a folder hierarchy 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
- Obtain components of messages

Web-based Email



Web-based emails are provided by gmail, Hotmail, Yahoo! Mail, etc.

- The user agent is an ordinary web browser

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks

2.7 socket programming with UDP and TCP

DNS: domain name system

People: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- hostname, e.g., www.yahoo.com - used by humans
- IP address (32 bit) - used for addressing datagrams

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 and name servers communicate to *resolve* names (address/name translation)

DNS Overview

- DNS Services
- DNS Structure
  - Hierarchical structure
  - Iterated and recursive query
- DNS protocol
  - DNS Records
  - Query and reply messages
- Inserting records into DNS

DNS Services

- **hostname to IP address translation**
- **host aliasing**
  - canonical, alias hostnames
  - **www.ibm.com** (alias) is really **servereast.backup2.ibm.com** (canonical)
  - From supplied alias hostname to canonical hostname
- **mail server aliasing**
- **load distribution**
  - replicated Web servers: many IP addresses correspond to one name
  - rotation distributes the traffic (rotate the ordering of IP addresses)

DNS Services

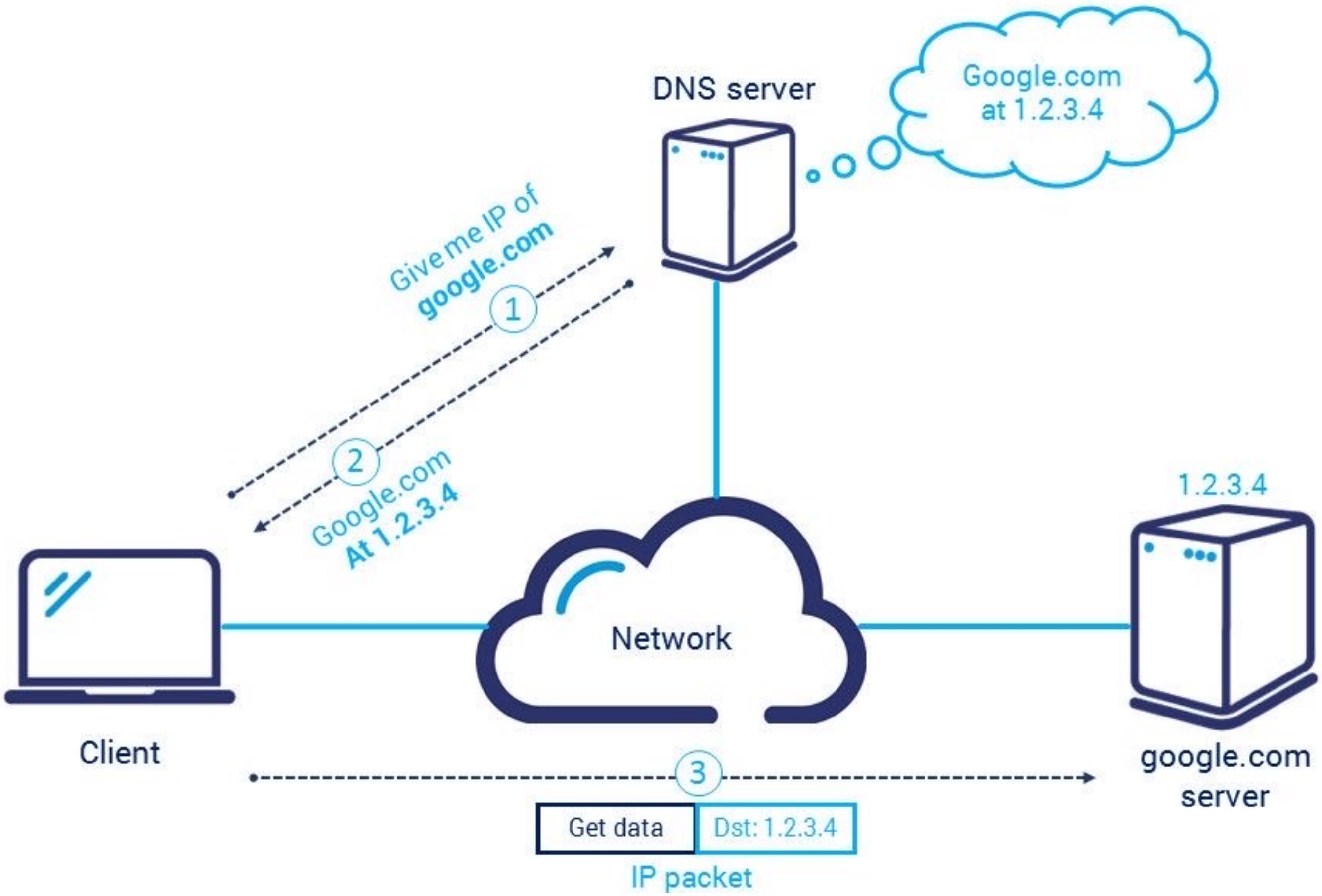
1. An application invokes the client side of DNS
   - specifying the hostname that needs to be translated
2. DNS in the user's host takes over, sending a query message into the network.
   - DNS query and reply messages
   - UDP datagrams to port 53.
3. After a delay, ranging from milliseconds to seconds, DNS in the user's host receives a DNS reply message that provides the desired mapping.
4. The mapping (hostname - IP) is then passed to the invoking application.

Why UDP?

- fast speed
- smaller data packets

DNS Services

From the perspective of the invoking application in the user's host, DNS is a black box providing a simple, straightforward translation service.



DNS Overview

- DNS Services
- DNS Structure
  - Hierarchical structure
  - Iterated and recursive query
- DNS protocol
  - DNS Records
  - Query and reply messages
- Inserting records into DNS

DNS Structure

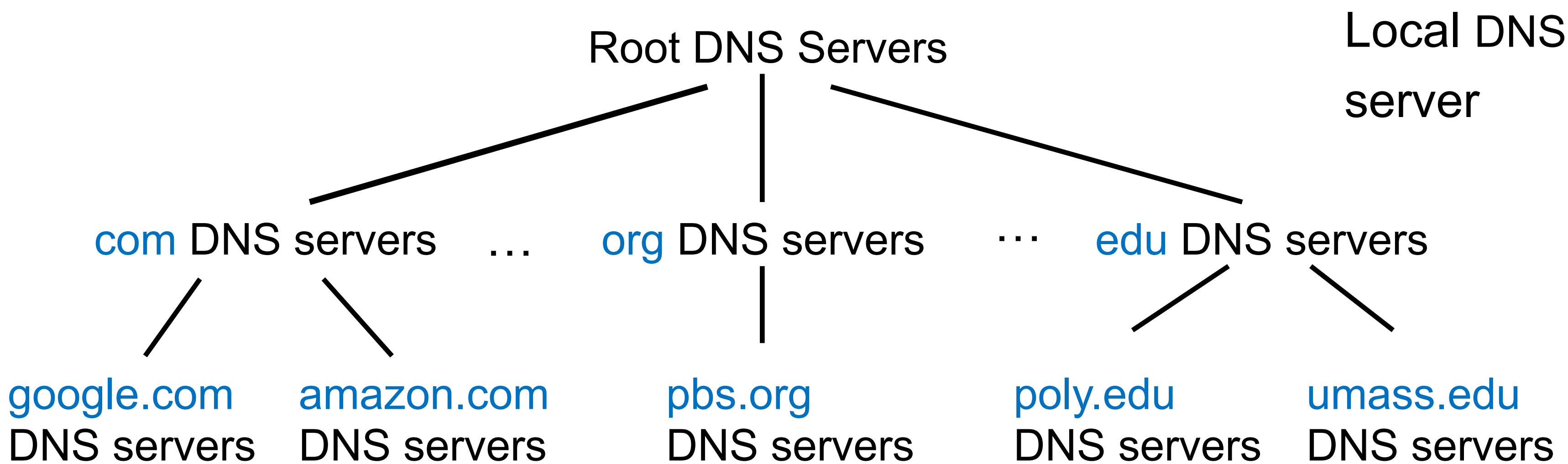
Centralized DNS:
Clients simply direct all queries to the single DNS server, and the DNS server responds directly to the querying clients.

Why not centralize DNS?

- Single point of failure
- Traffic volume
- Distant centralized database
- Maintenance: huge database, update frequently

A: doesn‘t scale!

DNS: a distributed, hierarchical database

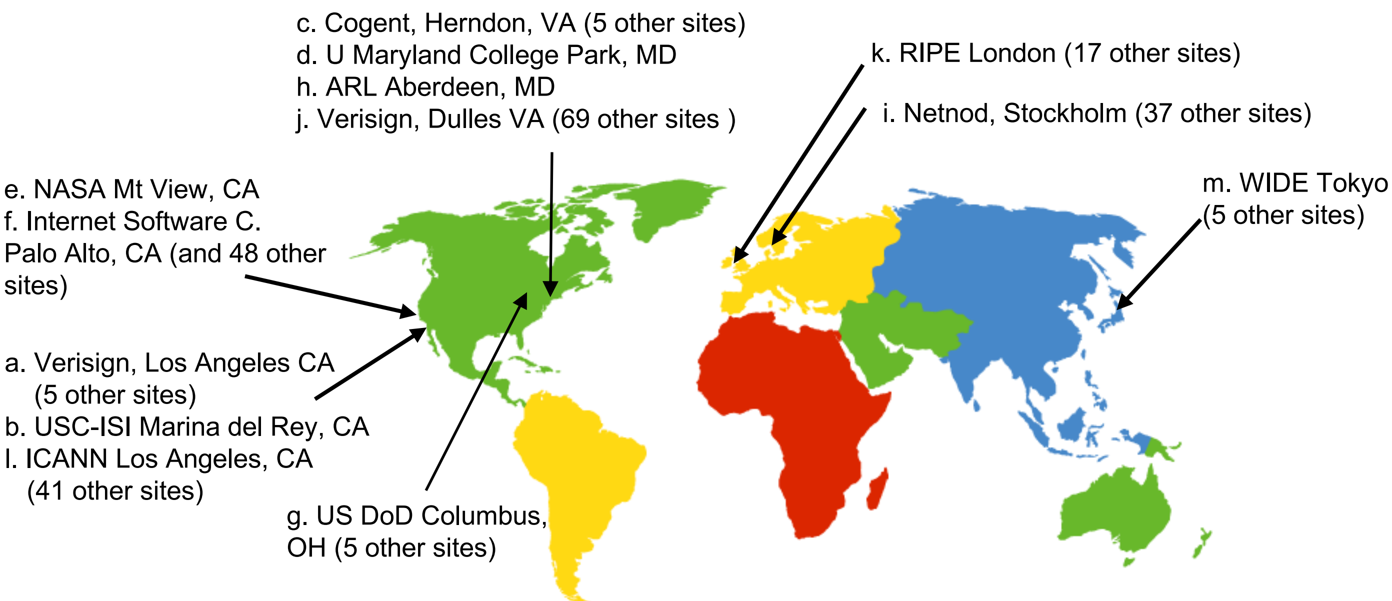


Client wants IP for www.google.com / scholar.google.com:

- Root DNS Servers: find IP address of the .com TLD DNS server
- Top-Level Domain (TLD) DNS: client queries .com DNS server to get google.com authoritative DNS server
- Authoritative DNS servers: client queries google.com DNS server to get IP address for www.google.com / scholar.google.com

DNS: root servers

- Root name server:
  - Provide the IP addresses of the TLD servers



13 logical root name "servers" worldwide

TLD, authoritative servers

Top-level domain (TLD) servers:

- Top-level domains: com, org, net, edu, aero, jobs, museums; top-level country domains: uk, fr, ca, jp
- *Network Solutions* maintains servers 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

Local DNS server

- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one
  - also called "default name server"

When a host connects to an ISP, the ISP provides the IP addresses of one or more of local DNS servers

- A host's local DNS server may be typically "close to" the host

When host makes DNS query, query is sent to local DNS server

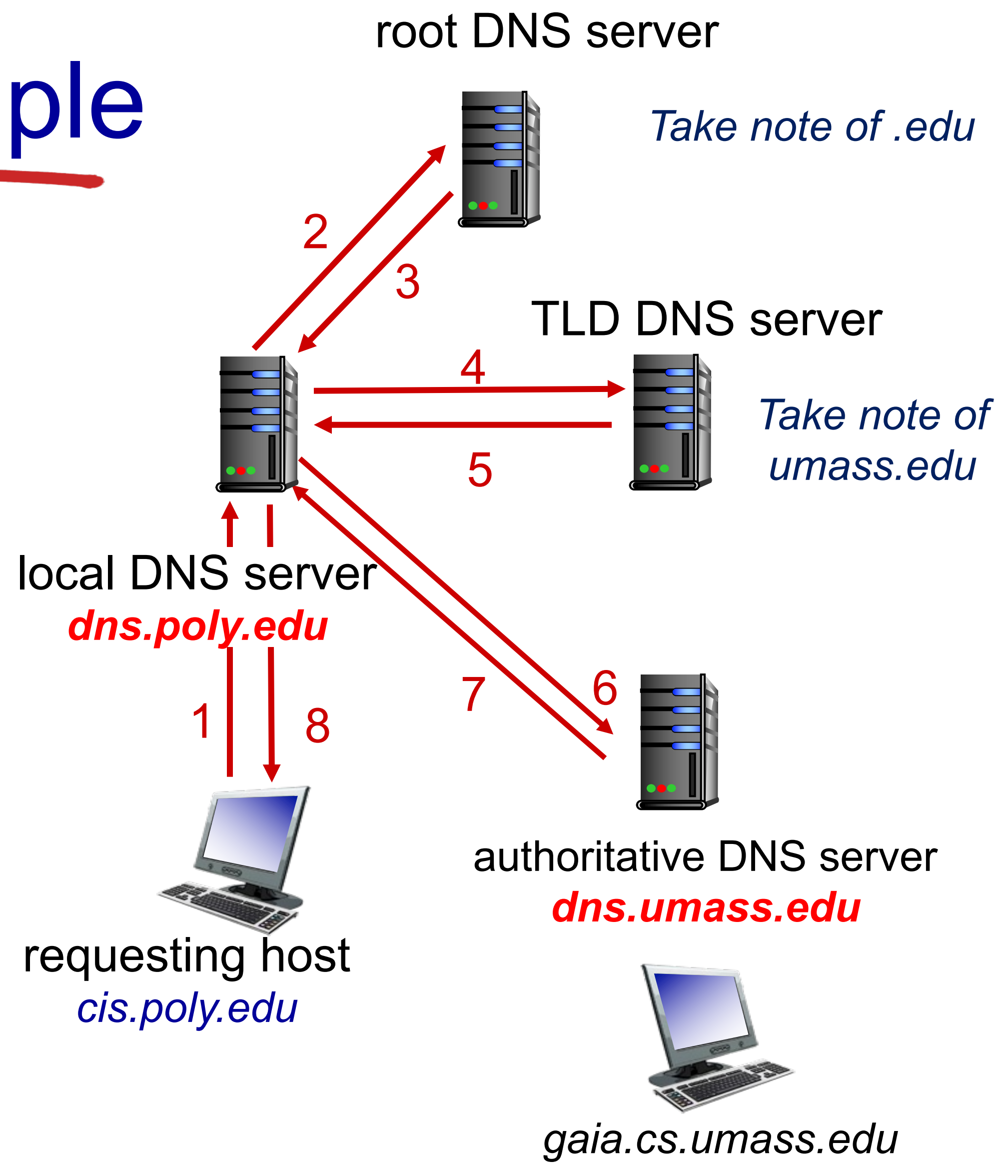
- acts as proxy, forwards query into hierarchy
- has local cache of recent name-to-address translation pairs (but may be out of date!)

DNS name resolution example

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

Iterated query:

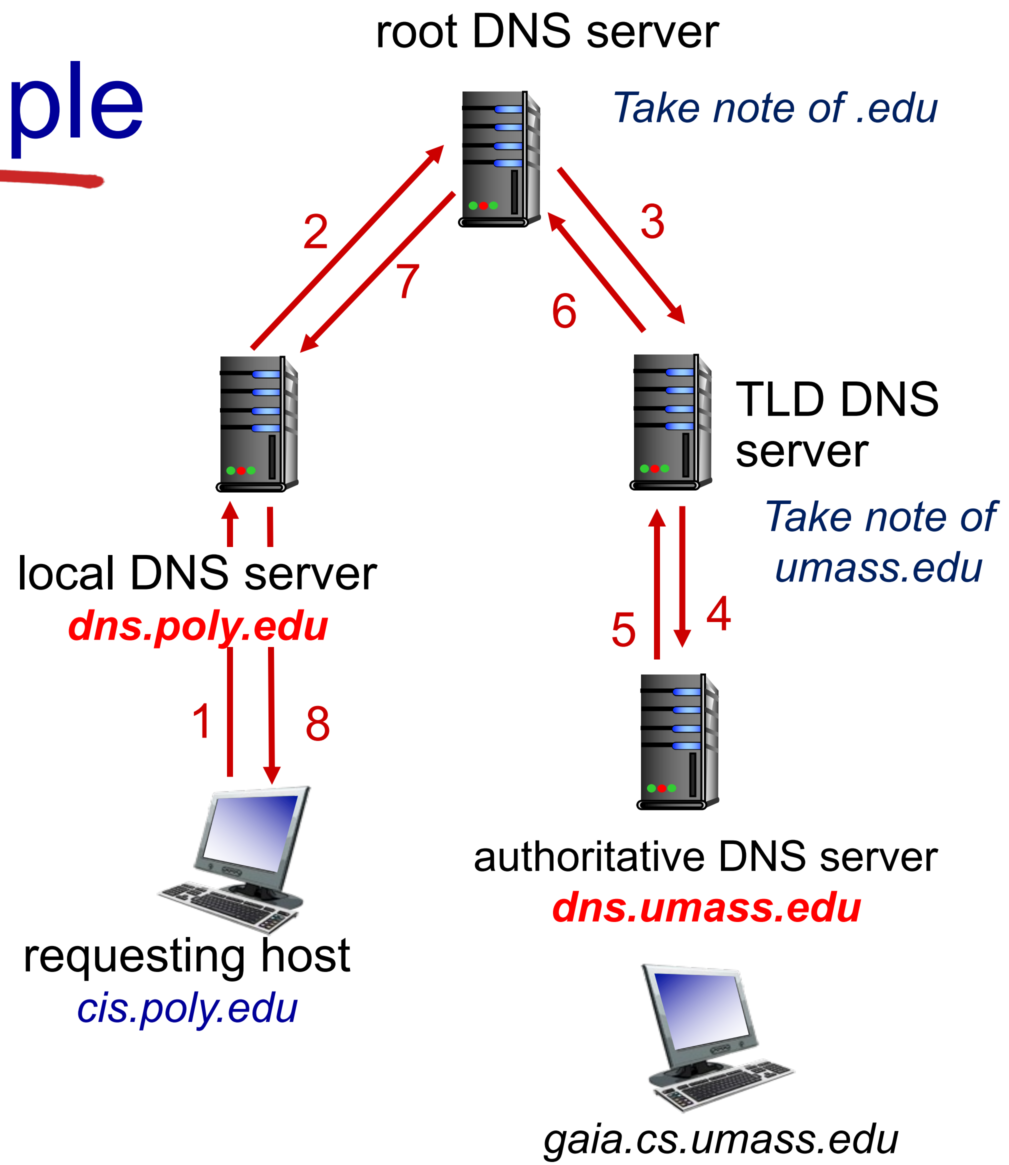
- contacted server replies with the name of another server to contact
- "I don't know this name, but ask this server"



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
  - TLD servers typically cached in local name servers
  - thus root DNS servers not often visited
- Cached entries may be *out-of-date*
  - cache entries timeout (disappear) after some time (e.g., two days)
- Update/notify mechanisms proposed IETF standard
  - RFC 2136

DNS Overview

- DNS Services
- DNS Structure
  - Hierarchical structure
  - Iterated and recursive query
- DNS protocol
  - DNS Records
  - Query and reply messages
- Inserting records into DNS

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 server for this domain (e.g., dns.foo.com)

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 canonical name of the mailserver with **name** (alias name)

DNS records

If a DNS server is authoritative for a particular hostname

- the DNS server will contain a Type A record for the hostname
- (Even if the DNS server is not authoritative, it may contain a Type A record in its cache.)

If a server is not authoritative for a hostname

- the server will contain a Type NS record for the domain that includes the hostname
- it will also contain a Type A record that provides the IP address of the DNS server in the Value field of the NS record.

Example: an .edu TLD server is not authoritative for gaia.cs.umass.edu

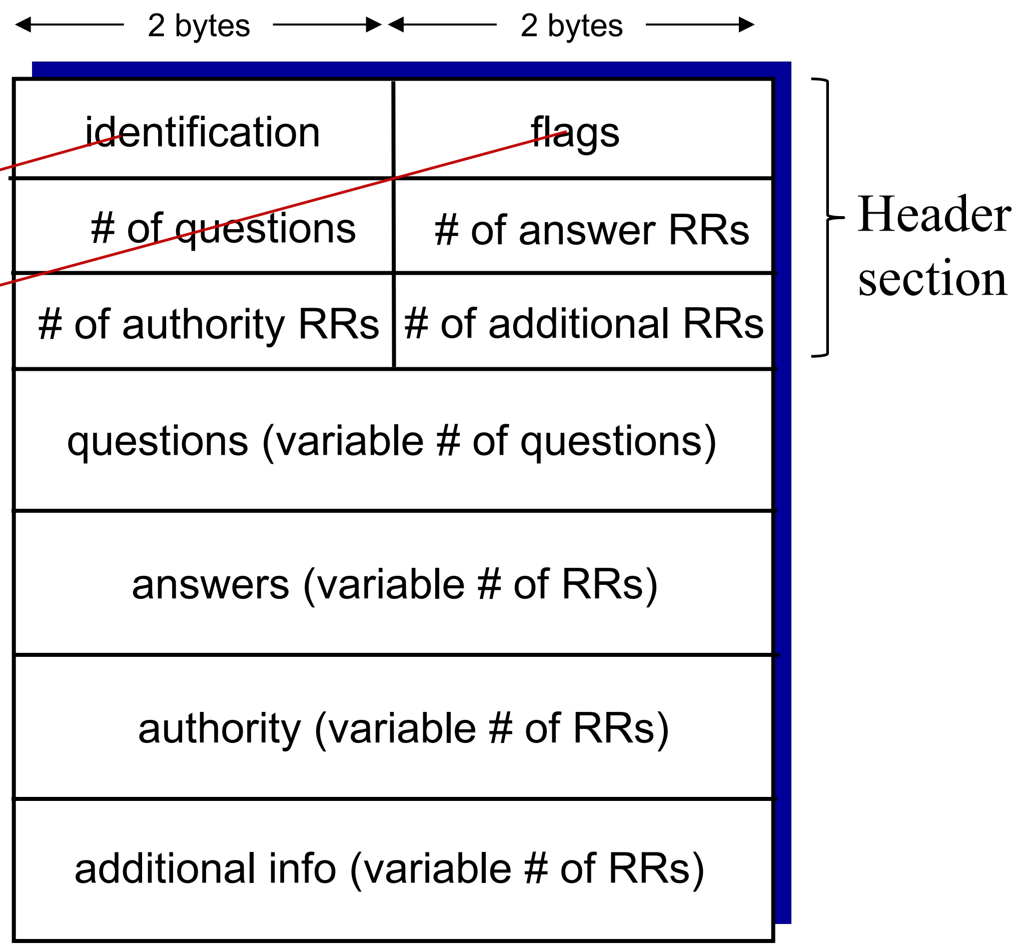
- (umass.edu, dns.umass.edu, NS) .
- (dns.umass.edu, 128.119.40.111, A)

DNS protocol, messages

Query and reply messages, both with same message format

message header

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



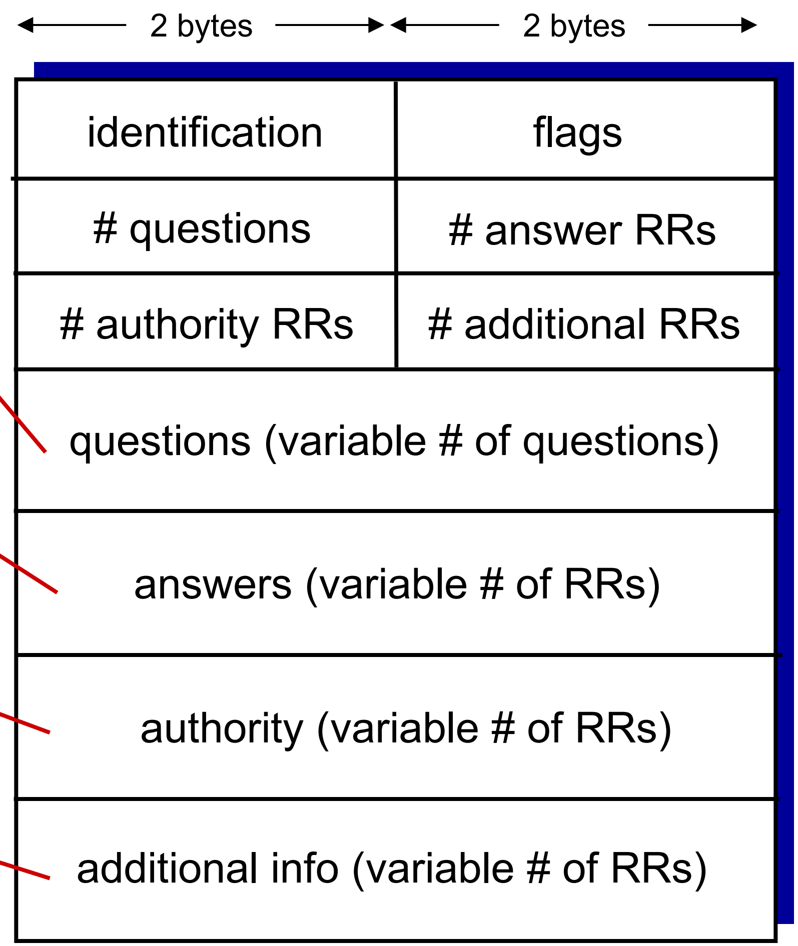
DNS protocol, messages

Name & type fields
(e.g., Type A or Type MX)

RRs in response to query
(a reply can return multiple RRs)

records of other authoritative servers

additional “helpful” info that may be used



DNS protocol, messages

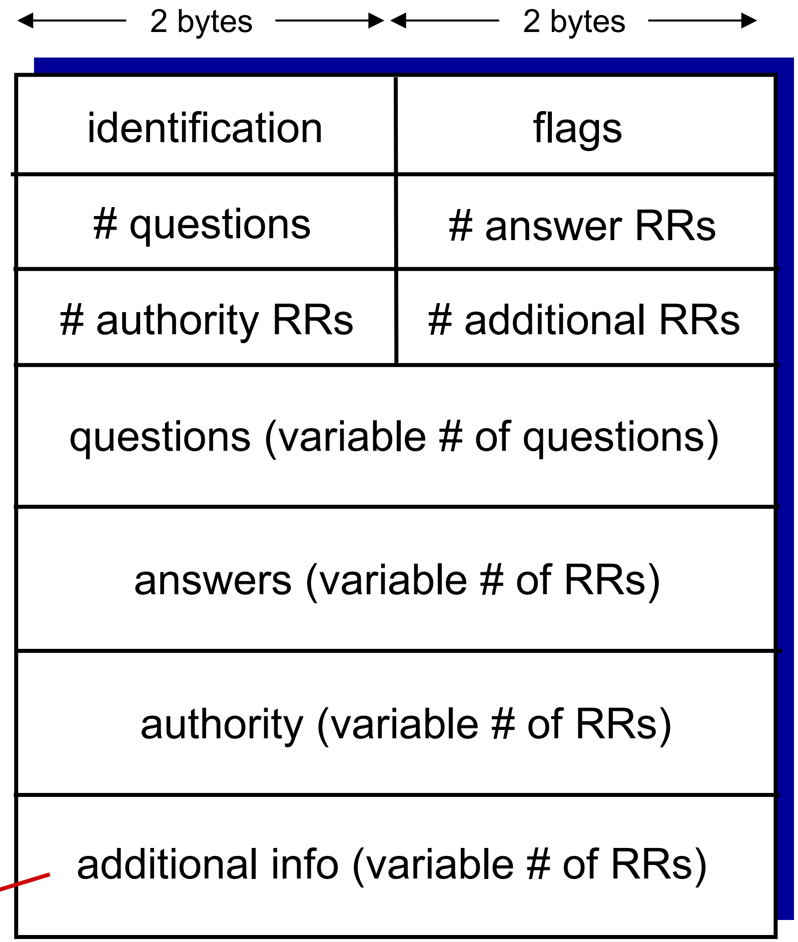
For example, a reply to **an MX query**

Answer section: Type MX

- an RR providing the canonical hostname of a mail server.

Additional section: Type A

- the IP address for the canonical hostname of the mail server.



additional "helpful" info that may be used

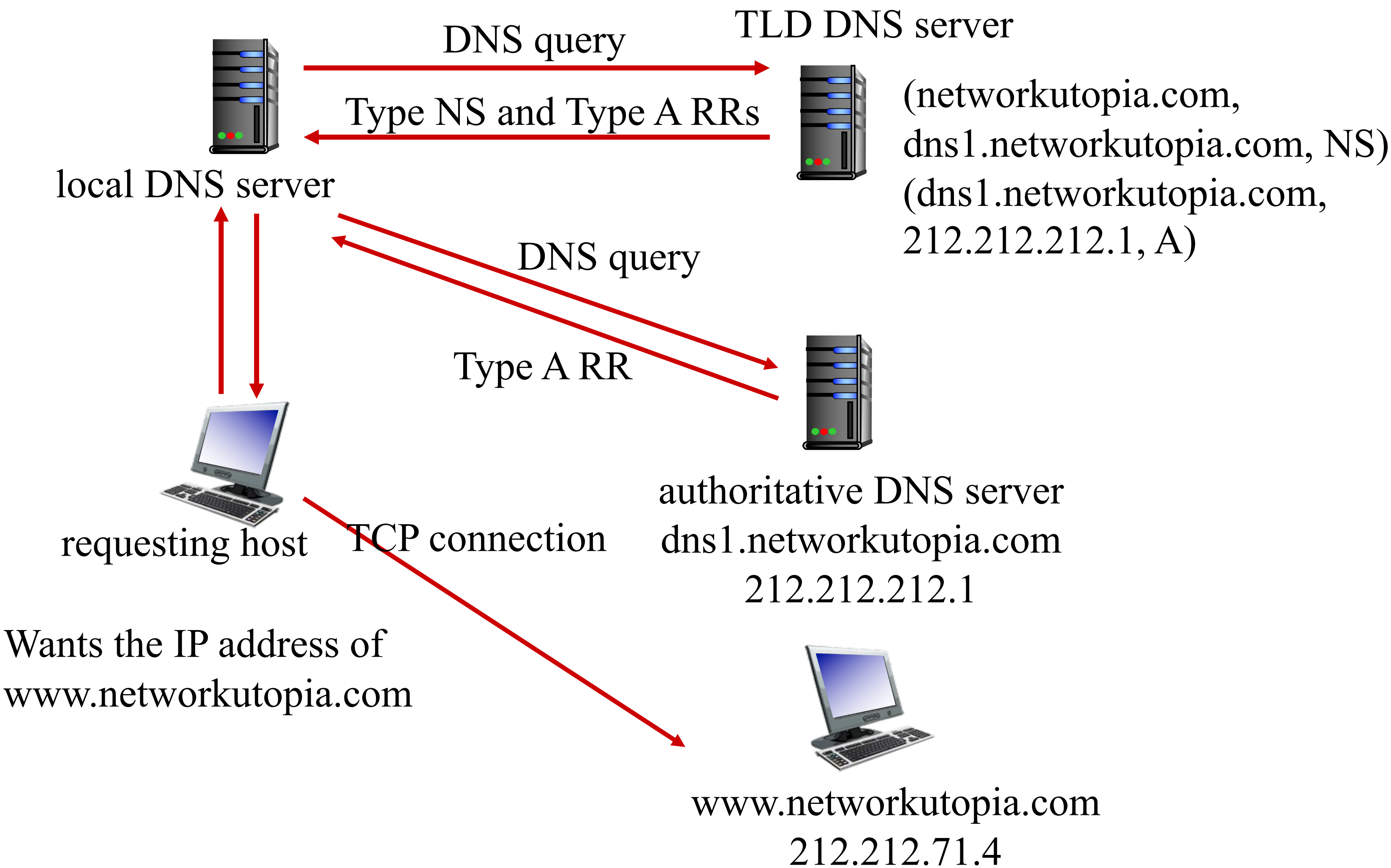
DNS Overview

- DNS Services
- DNS Structure
  - Hierarchical structure
  - Iterated and recursive query
- DNS protocol
  - DNS Records
  - Query and reply messages
- Inserting records into DNS server

Inserting records into DNS

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

Inserting records into DNS



Attacking DNS

Distributed denial-of-service (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 queries; bogus reply
- DNS poisoning
  - Send bogus replies to DNS server

Exploit DNS for DDoS

- target IP
- Redirect an unsuspecting Web user to attack Web site

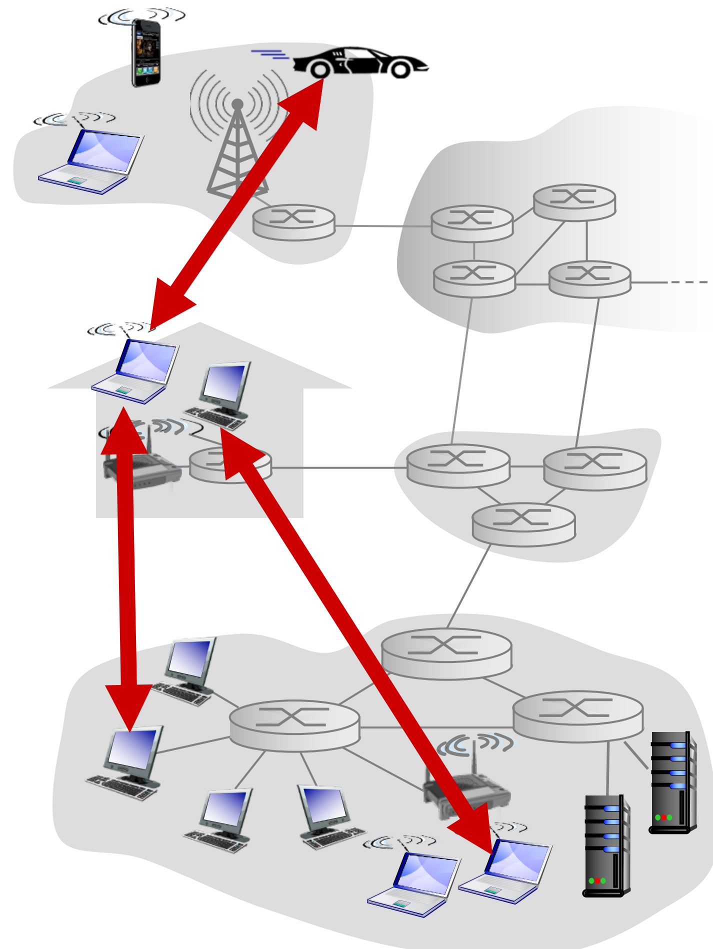
Chapter 2: outline

Pure P2P architecture

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

Examples:

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



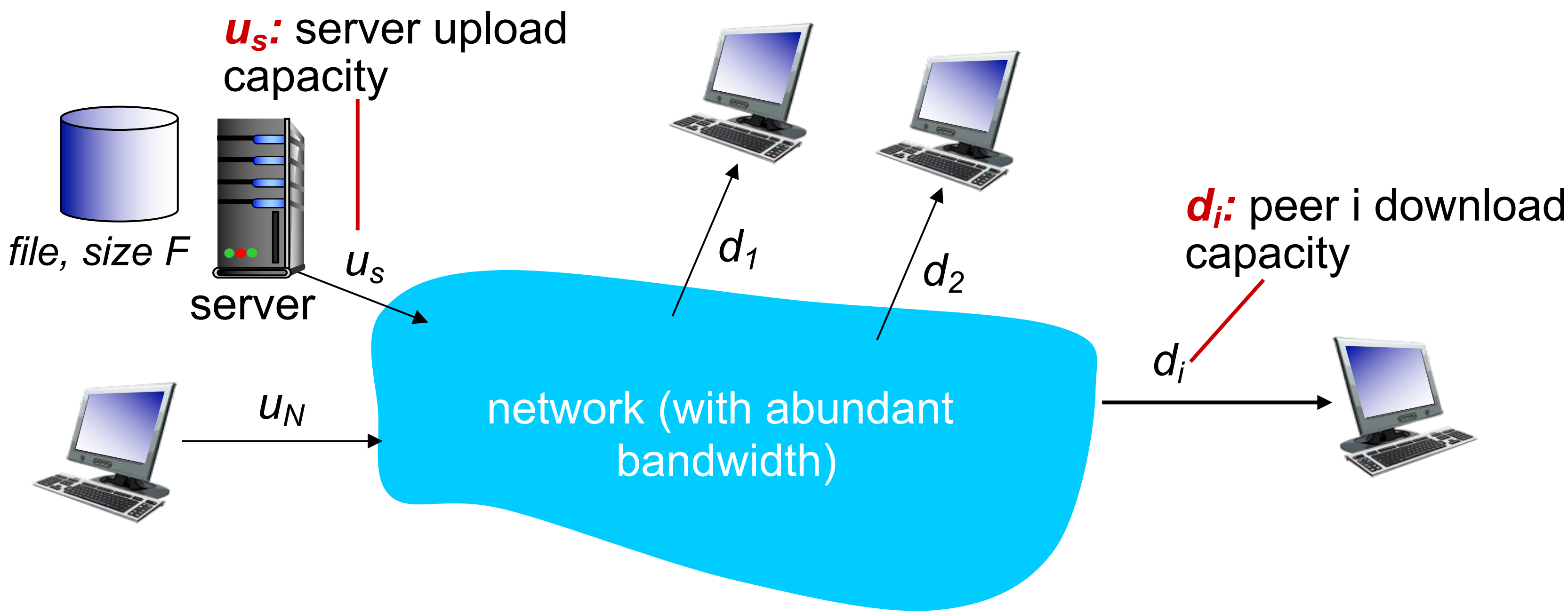
DNS Overview

- P2P vs Client Server
- BitTorrent

File distribution: client-server vs P2P

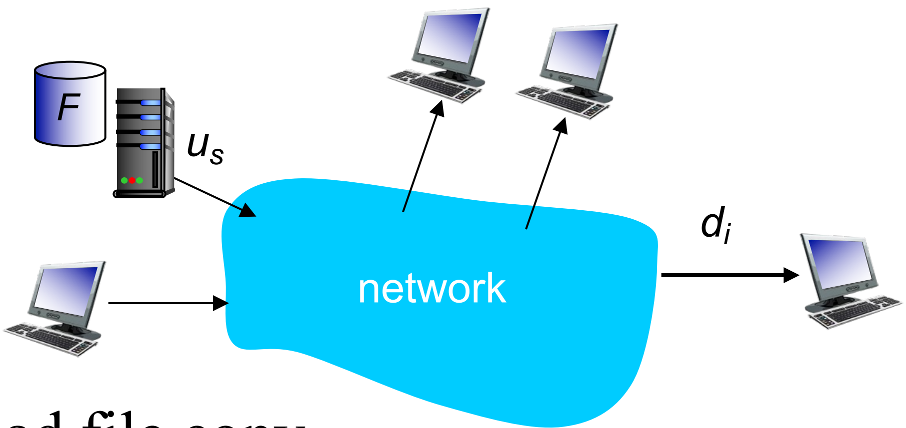
Question: How much time to distribute file (size F) from one server to N *peers*?

- peer upload/download capacity is limited resource
- **Distribution time:** the time it takes to get a copy of the file to all N peers.



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
  - maximum client download time: F/d_{min}

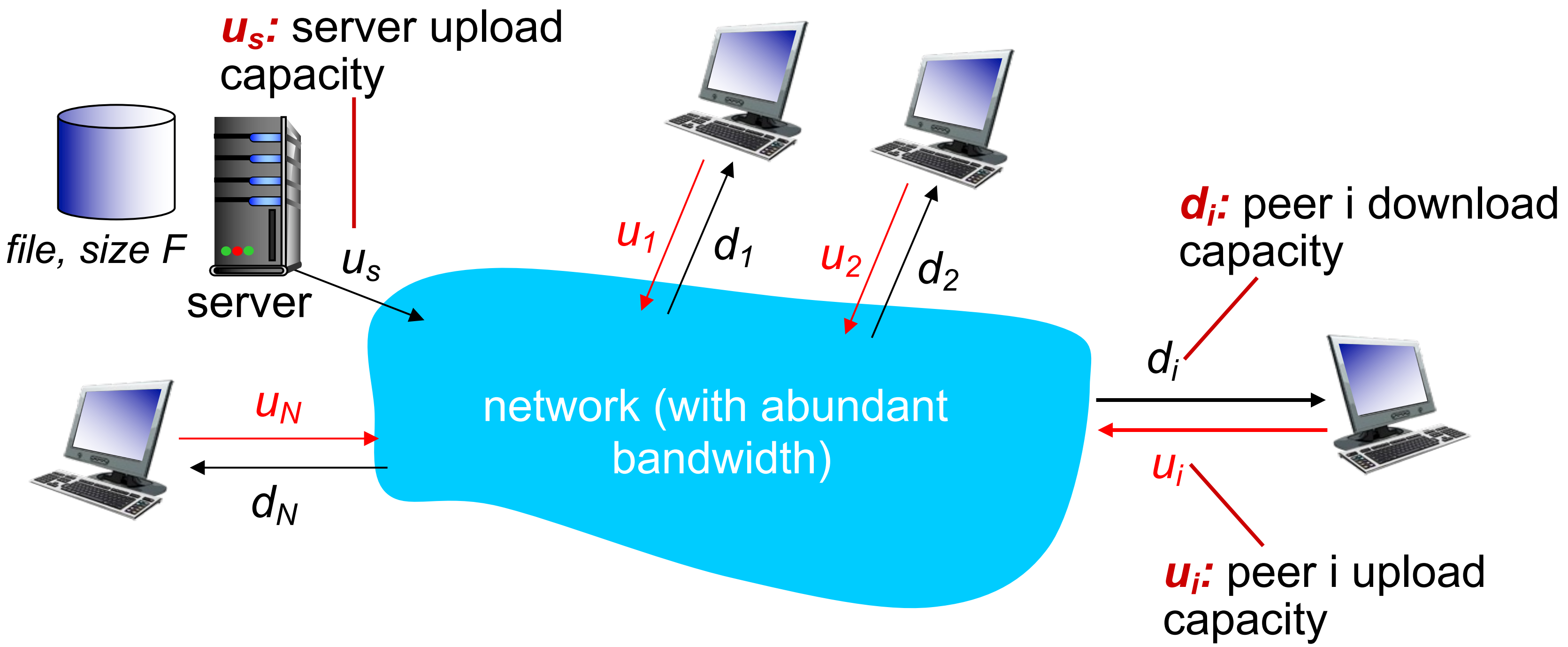
time to distribute F to N clients using client-server approach

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

increases linearly in N

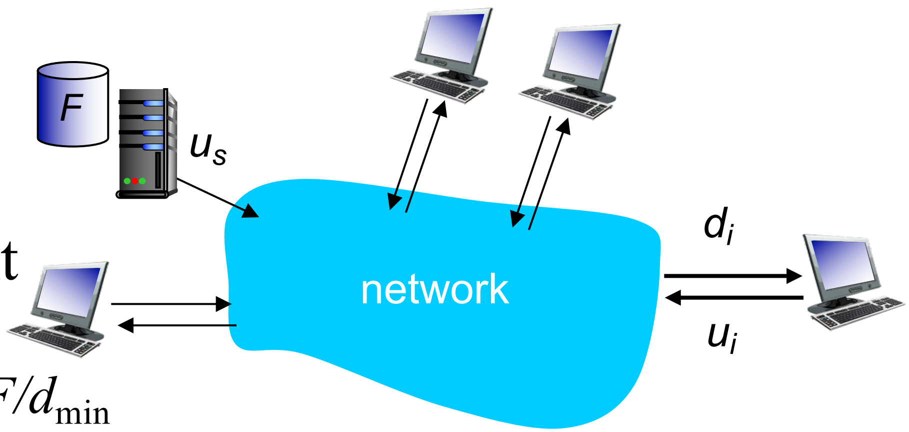
File distribution time: P2P

In P2P model, clients are <u>both downloaders and uploaders</u>.



File distribution time: P2P

- Server transmission: must upload at least one copy
  - time to send one copy: F/u_s
- Client downloading: each client must download file copy
  - maximum client download time: F/d_{min}
- Clients and server: delivering a total of NF bits
  - max upload rate (limiting max download rate) is $u_s + \Sigma u_i$



time to distribute F to N clients using P2P approach

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

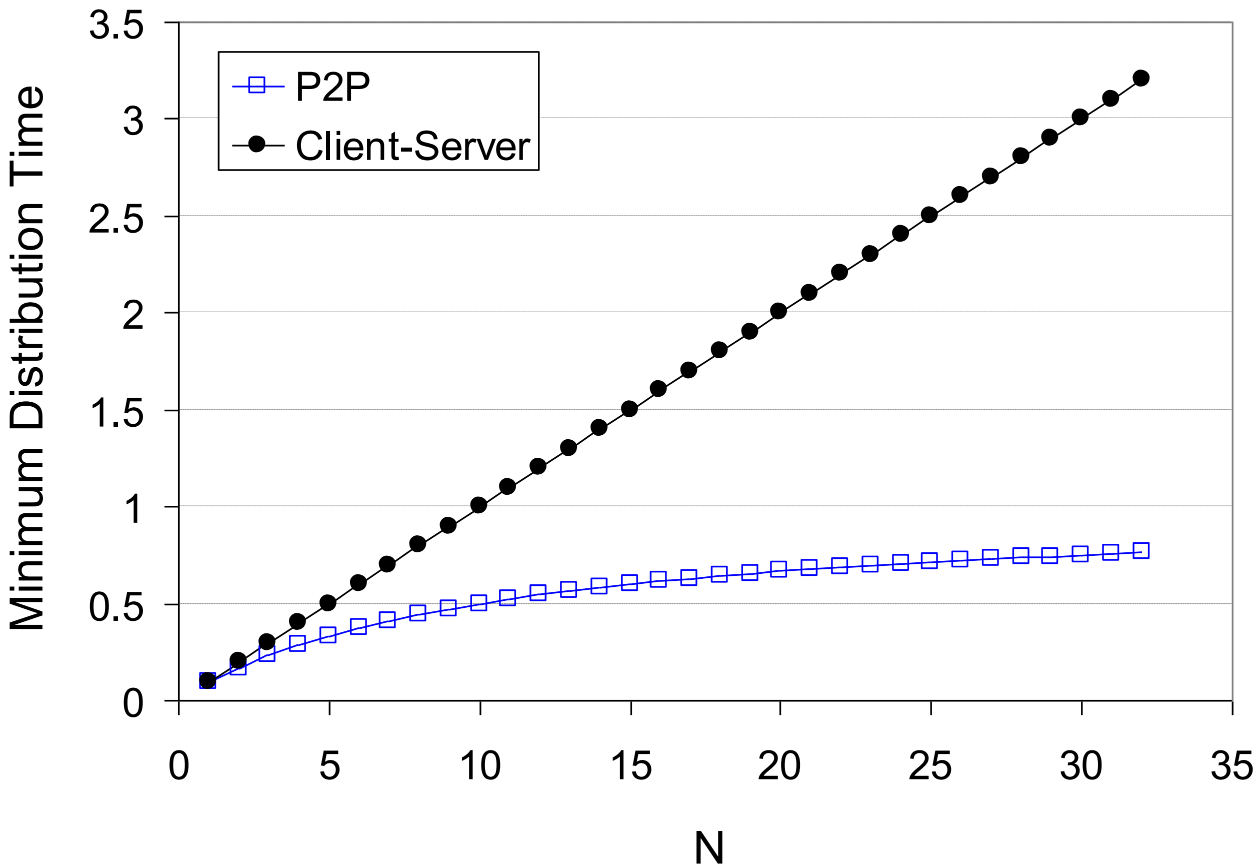
increases linearly in N …

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

If each peer can redistribute a bit as soon as it receives the bit, then there is a scheme that actually achieves this lower bound

Client-server vs. P2P: example

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

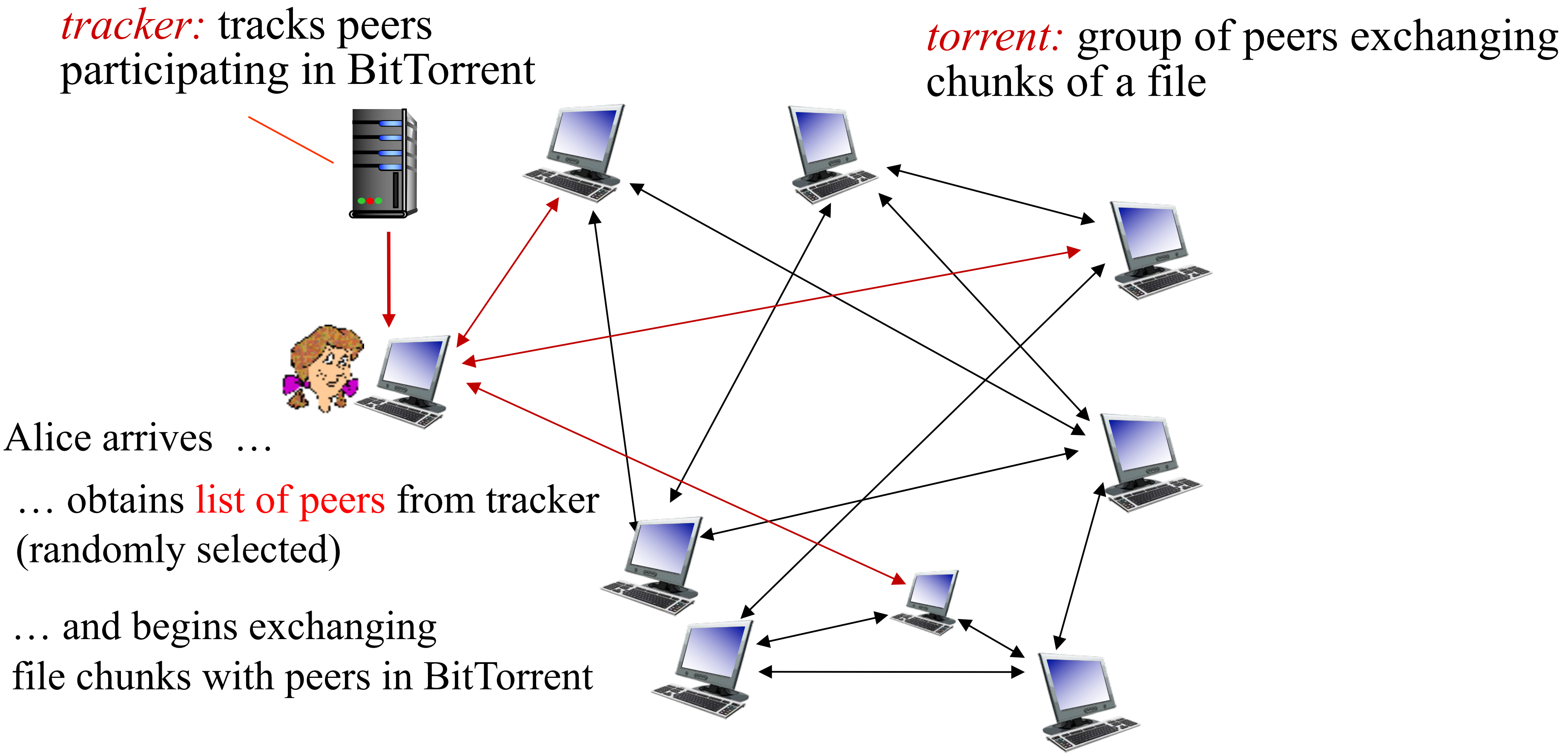


DNS Overview

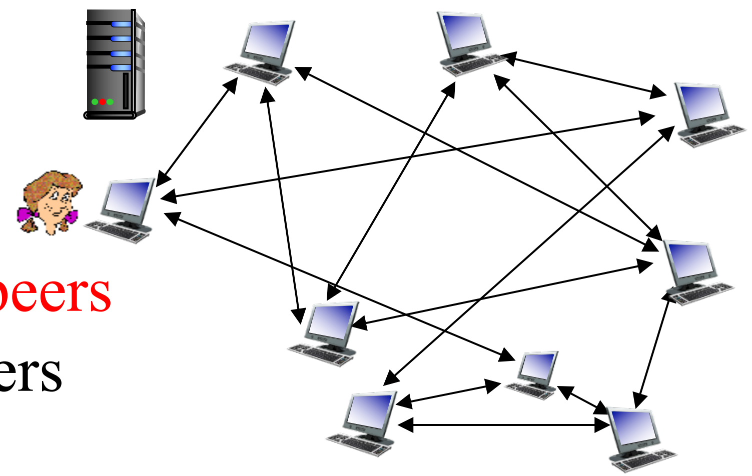
- P2P vs Client Server
- BitTorrent

P2P file distribution: BitTorrent

- File divided into 256Kb chunks
- Peers in BitTorrent send/receive file chunks



P2P file distribution: BitTorrent



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

- While downloading, peer uploads chunks to other peers
  - Peers may leave
  - Peers may come, initiating connections with Alice

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

BitTorrent: requesting, sending file chunks

Q1 : which chunks should she request first from her neighbors?

Q2: to which of her neighbors should she send requested chunks?

requesting chunks:

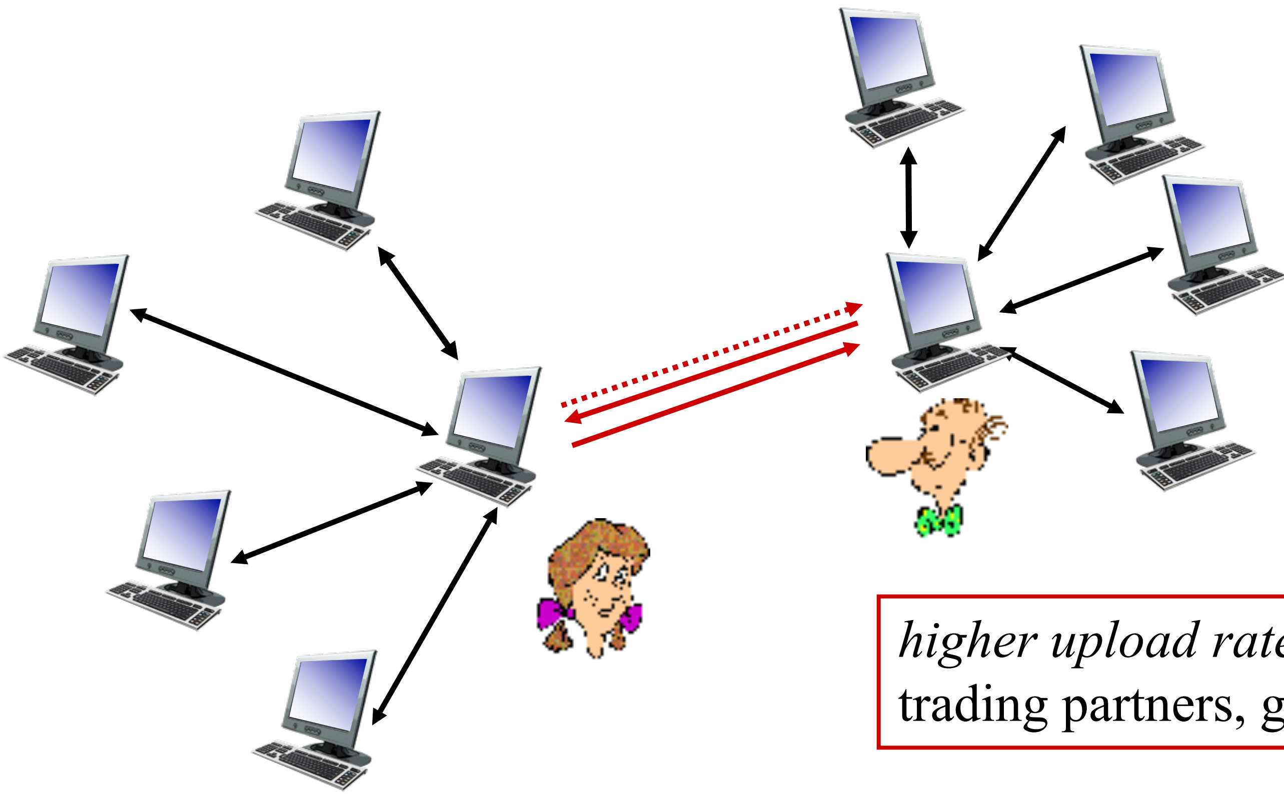
- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each "neighbor" for list of chunks that they have
- Alice requests missing chunks from peers, rarest first

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 every 10 secs
- every 30 secs: randomly select one additional 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



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

Chapter 2: outline