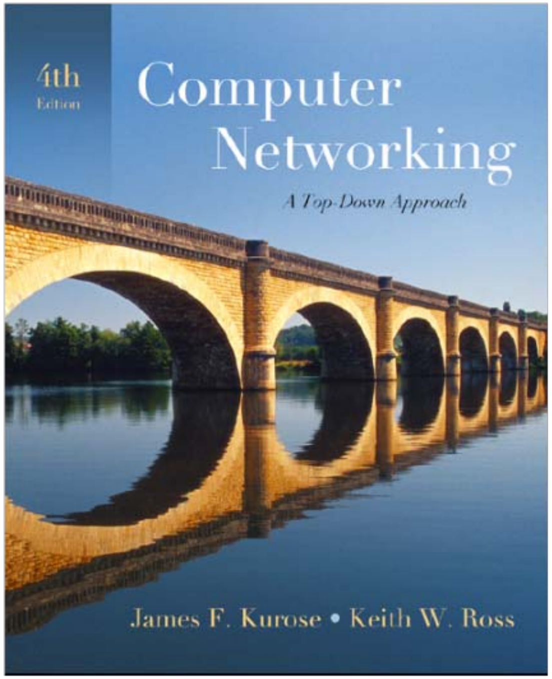
Chapter 2
Application Layer
第二章 應用層



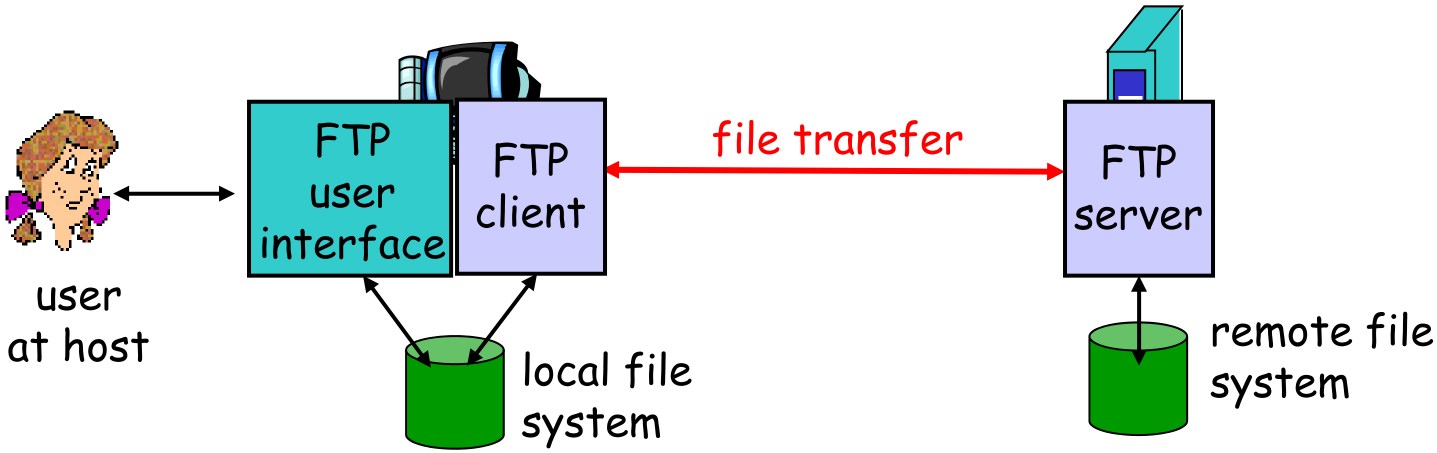
Computer Networking: A Top Down Approach,
4th edition.
Jim Kurose, Keith Ross
Addison-Wesley, July 2007.

Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP 檔案傳輸協定
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP

  電子郵件
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
- 2.9 Building a Web server

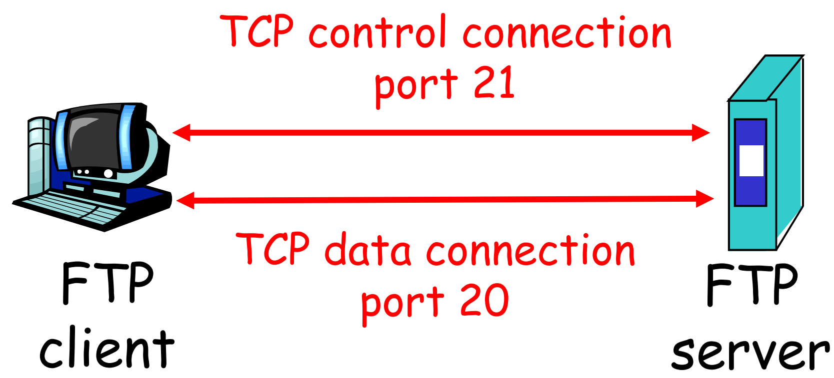
FTP: the file transfer protocol 檔案傳輸協定



- transfer file to/from remote host 資料在本機-遠端間傳輸
- client/server model 主從式架構
  - *client:* side that initiates transfer (either to/from remote) 啟動檔案傳輸的機器（本機）
  - *server:* remote host 遠端
- ftp: RFC 959
- ftp server: port 21 連接埠

FTP: separate control, data connections

- FTP client contacts FTP server at port 21, TCP is transport protocol
- client authorized over control connection 在控制連線上認證
- client browses remote directory by sending commands over control connection. 在控制連線上傳送指令
- when server receives file transfer command, server opens 2nd TCP connection (for file) to client 打開資料連線轉輸
- after transferring one file, server closes data connection. 資料傳輸結束後，結束資料連線



- server opens another TCP data connection to transfer another file. 傳輸另一檔案時，重新建立TCP連結
- control connection: "out of band" 控制及資料傳輸在不同連線上
- FTP server maintains "state": current directory, earlier authentication 維持狀態

FTP commands, responses
FTP指令及回應

Sample commands:

- sent as ASCII text over control channel
- **USER** ***username***
- **PASS** ***password***
- **LIST** return list of file in current directory
- **RETR filename** retrieves (gets) file
- **STOR filename** stores (puts) file onto remote host
- get, send 收、送資料

Sample return codes

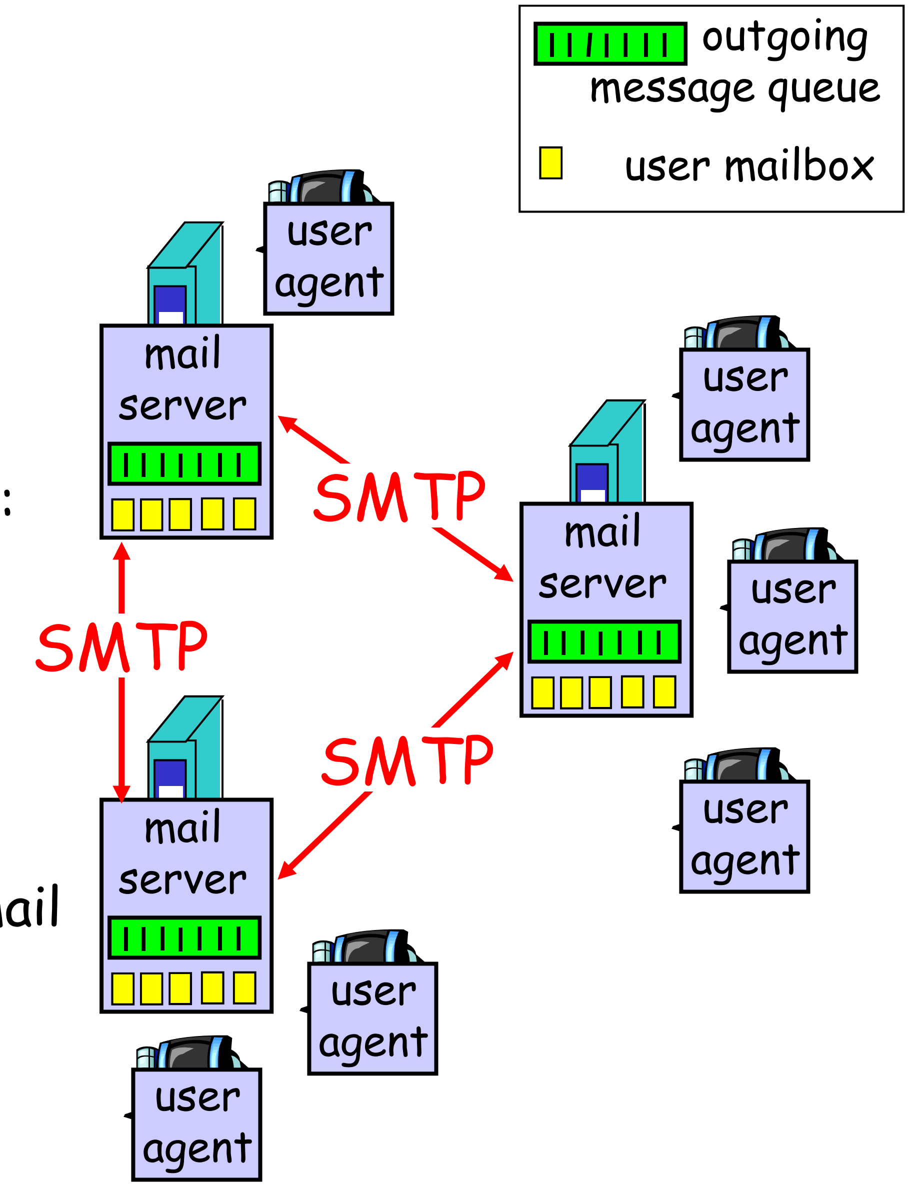
- status code and phrase (as in HTTP)
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can’t open data connection**
- **452 Error writing file**

Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP

  電子郵件
- 2.5 DNS

Electronic Mail



Three major components:

- user agents 使用代理程式
- mail servers 郵件伺服器
- simple mail transfer protocol: SMTP 郵件傳輸協定

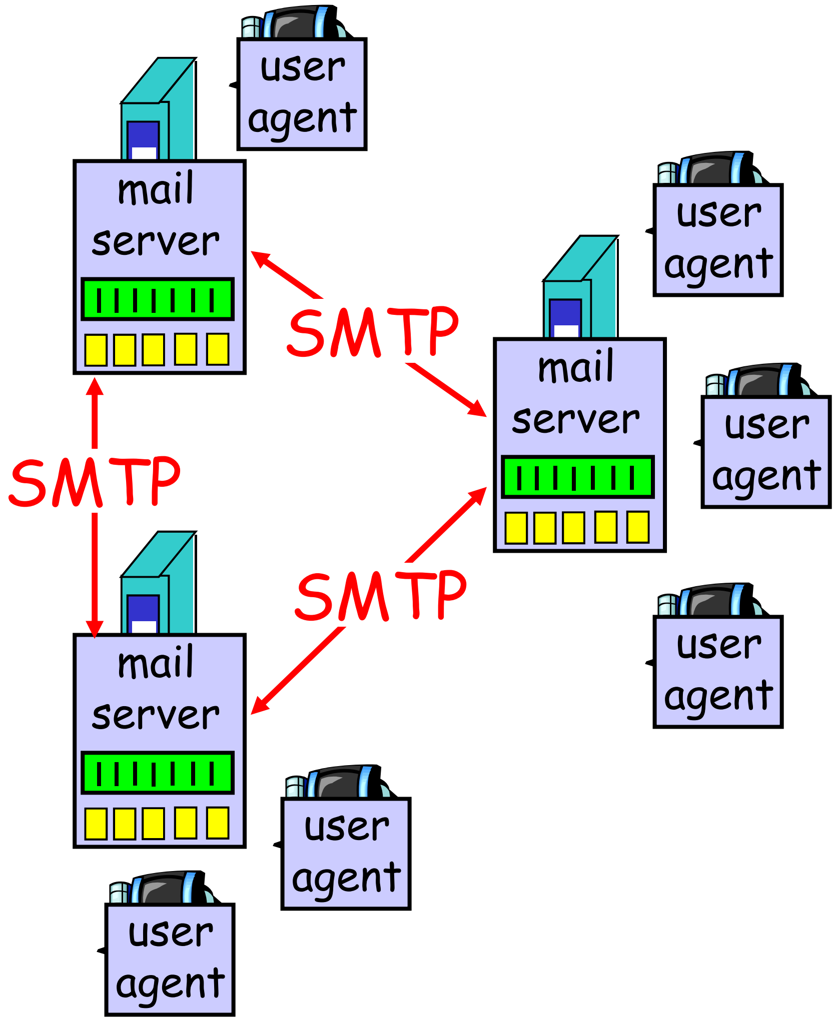
User Agent 使用者代理程式

- a.k.a. "mail reader" 郵件程式
- composing, editing, reading mail messages 寫信、讀信
- e.g., Eudora, Outlook, elm, Mozilla Thunderbird
- outgoing, incoming messages stored on server 信件存在伺服器

Electronic Mail: mail servers

Mail Servers 郵件伺服器

- **mailbox** contains incoming messages for user 信箱
- **message queue** of outgoing (to be sent) mail messages
- **SMTP protocol** between mail servers to send email messages
  - client: sending mail server 送信的server
  - "server": receiving mail server 收信的server



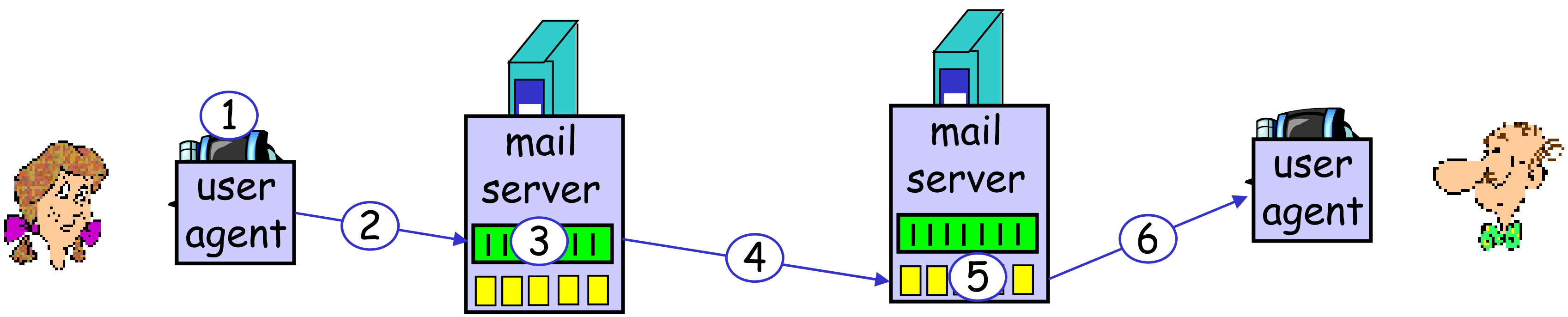
Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25 使用TCP連線，連接埠號號為25
- direct transfer: sending server to receiving server 直接連線，不透過其它主機
- three phases of transfer 傳輸三步驟
  - handshaking (greeting) 握手（建立連線）
  - transfer of messages 傳輸資料
  - closure 結束
- command/response interaction
  - commands: ASCII text 指令
  - response: status code and phrase 回應
- messages must be in 7-bit ASCII（7-bit編碼）

Scenario: Alice sends message to Bob

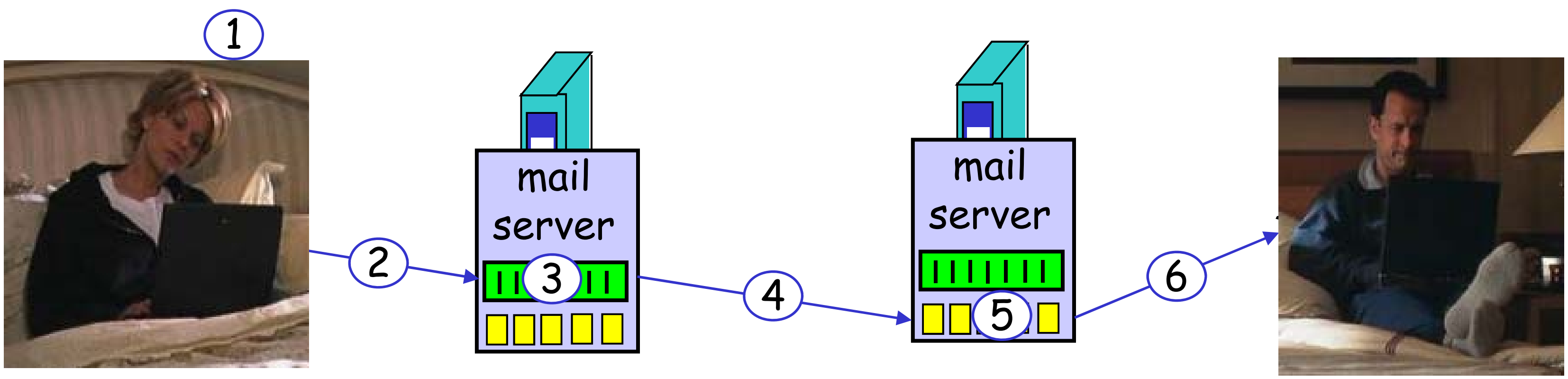
Alice傳送電子郵件給Bob的過程

1) Alice uses UA to compose message and "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



Scenario: Kathleen sends message to Joe

1) Kathleen uses UA to compose message and "to" `joe@fox.com`
2) Kathleen's UA sends message to her mail server; message placed in message queue
3) Client side of SMTP opens TCP connection with Joe's mail server
4) SMTP client sends Kathleen's message over the TCP connection
5) Joe's mail server places the message in Joe's mailbox
6) Joe 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
```

S: SMTP伺服器端 C: SMTP用戶端

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 email client (reader)

SMTP: final words

- SMTP uses persistent connections 使用持續性連線
- SMTP requires message (header & body) to be in 7-bit ASCII
  以 7-bit ASCII編碼
- SMTP server uses CRLF.CRLF to determine end of message
  以CRLF.CRLF結束訊息

Comparison with HTTP:

- HTTP: pull 取得式協定
- SMTP: push 送出式協定
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

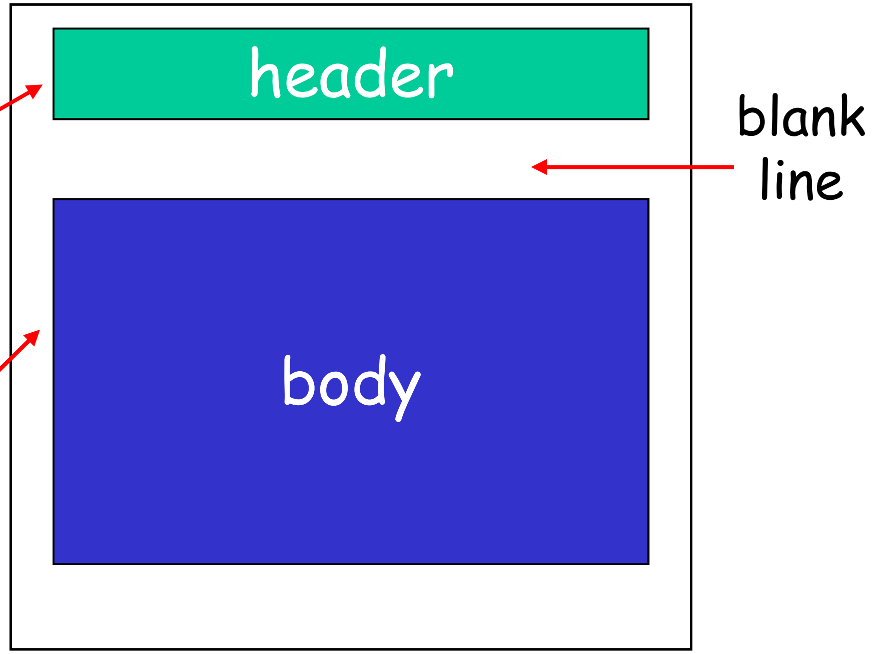
Mail message format 郵件格式

SMTP: protocol for exchanging email msgs

RFC 822: standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:

  different from SMTP commands!
- body 訊息內容
  - the "message", ASCII characters only



Message format: multimedia extensions 多媒體郵件

- MIME: multimedia mail extension, RFC 2045, 2056
- additional lines in msg header declare MIME content type
  在表頭增加資訊告知此為多媒體資訊，並指出多媒體型態

MIME version 版本

method used to encode data 編碼方式

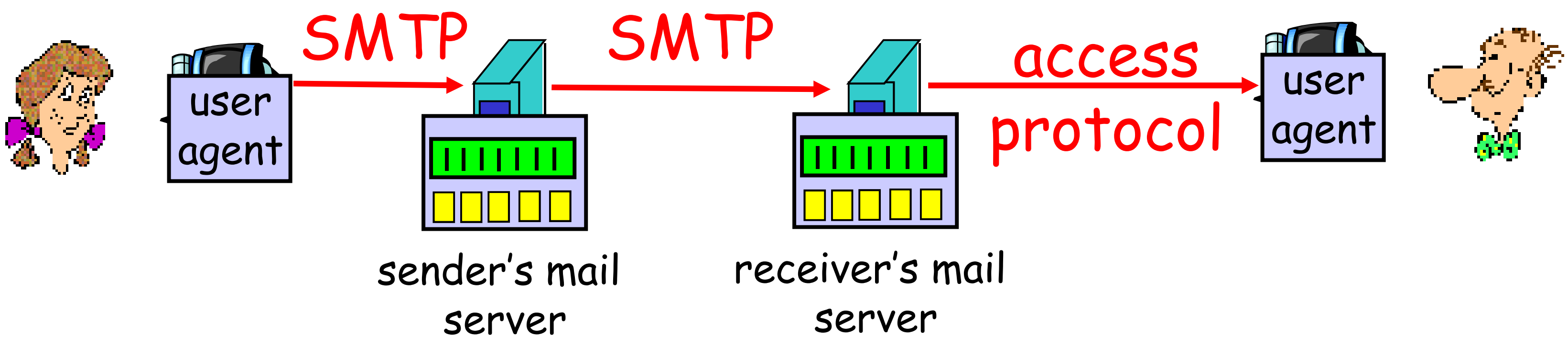
multimedia data type, subtype, parameter declaration 多媒體格式

encoded data 編碼後資料

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
```

Mail access protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) and download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - more features (more complex)
    - manipulation of stored msgs on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

POP3 protocol

authorization phase

- client commands:
  - **user**: declare username
  - **pass**: password
- server responses
  - **+OK**
  - **-ERR**

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

transaction phase, client:

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

```
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 (more) and IMAP

More about POP3

- Previous example uses "download and delete" mode.
- Bob cannot re-read e-mail if he changes client
- "Download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

IMAP

- Keep all messages in one place: the server
- Allows user to organize messages in folders
- IMAP keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name

Chapter 2: Application layer

- 2.1 Principles of network applications 應用層原理
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P Applications
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP

Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
- 2.9 Building a Web server

DNS: Domain Name System

People: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- "name", e.g., ww.yahoo.com - used by humans

Q: map between IP addresses and name ?

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol* host, routers, name servers to communicate to *resolve* names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network's "edge"

DNS

DNS services

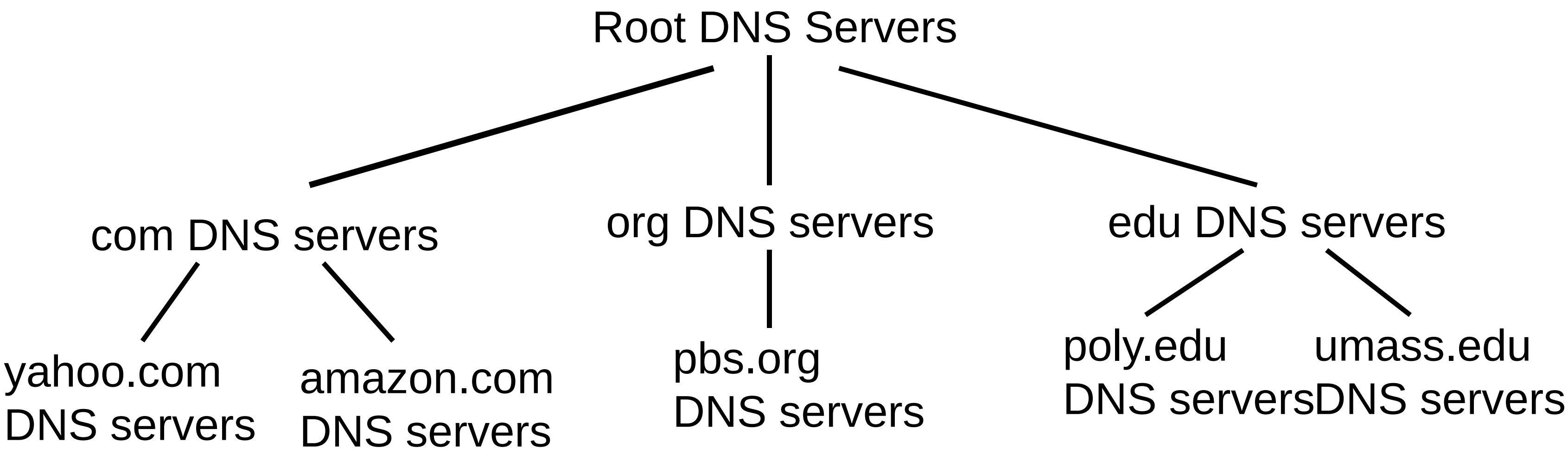
- hostname to IP address translation
- host aliasing
  - Canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?

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

doesn't *scale!*

Distributed, Hierarchical Database

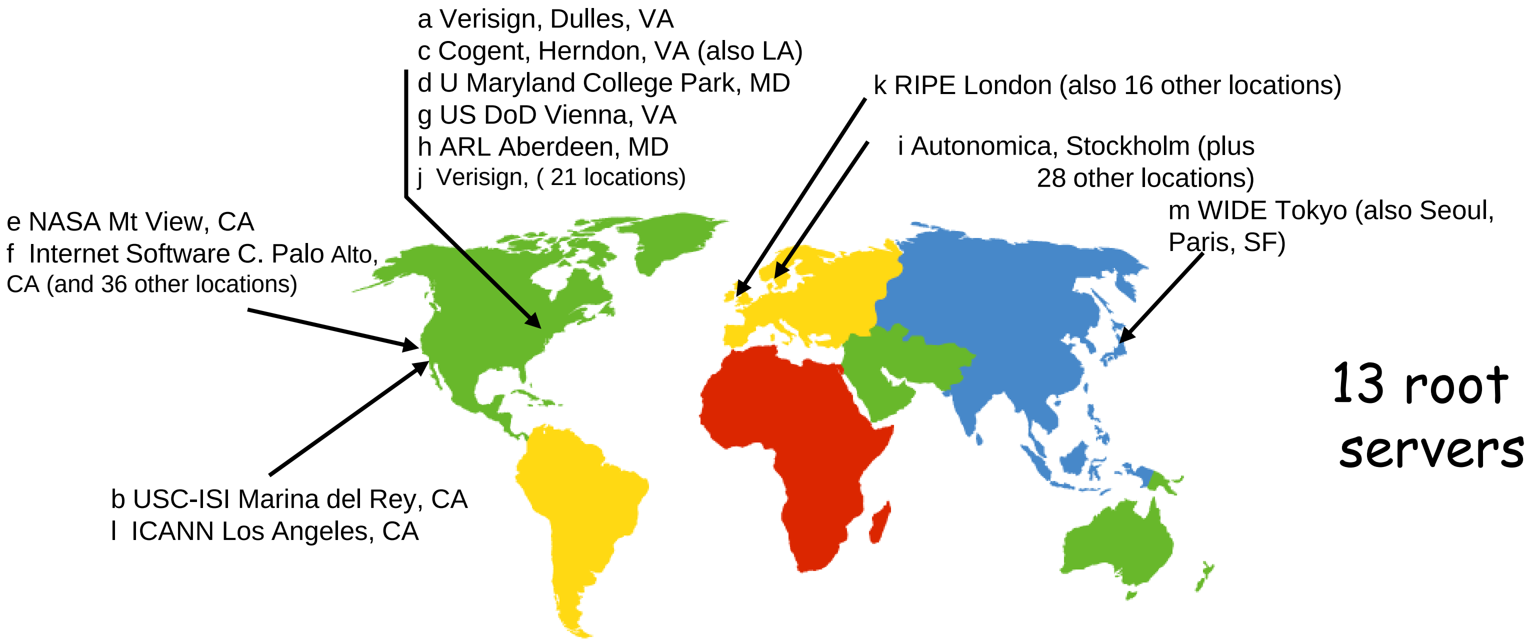


Client wants IP for www.amazon.com; 1st approx:

- client queries a 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

- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server



13 root name servers worldwide

TLD and Authoritative Servers

- Top-level domain (TLD) servers:
  - responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
  - Network Solutions maintains servers for com TLD
  - Educause for edu TLD
- Authoritative DNS servers:
  - organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web, mail).
  - can be maintained by organization or service provider

Local Name Server

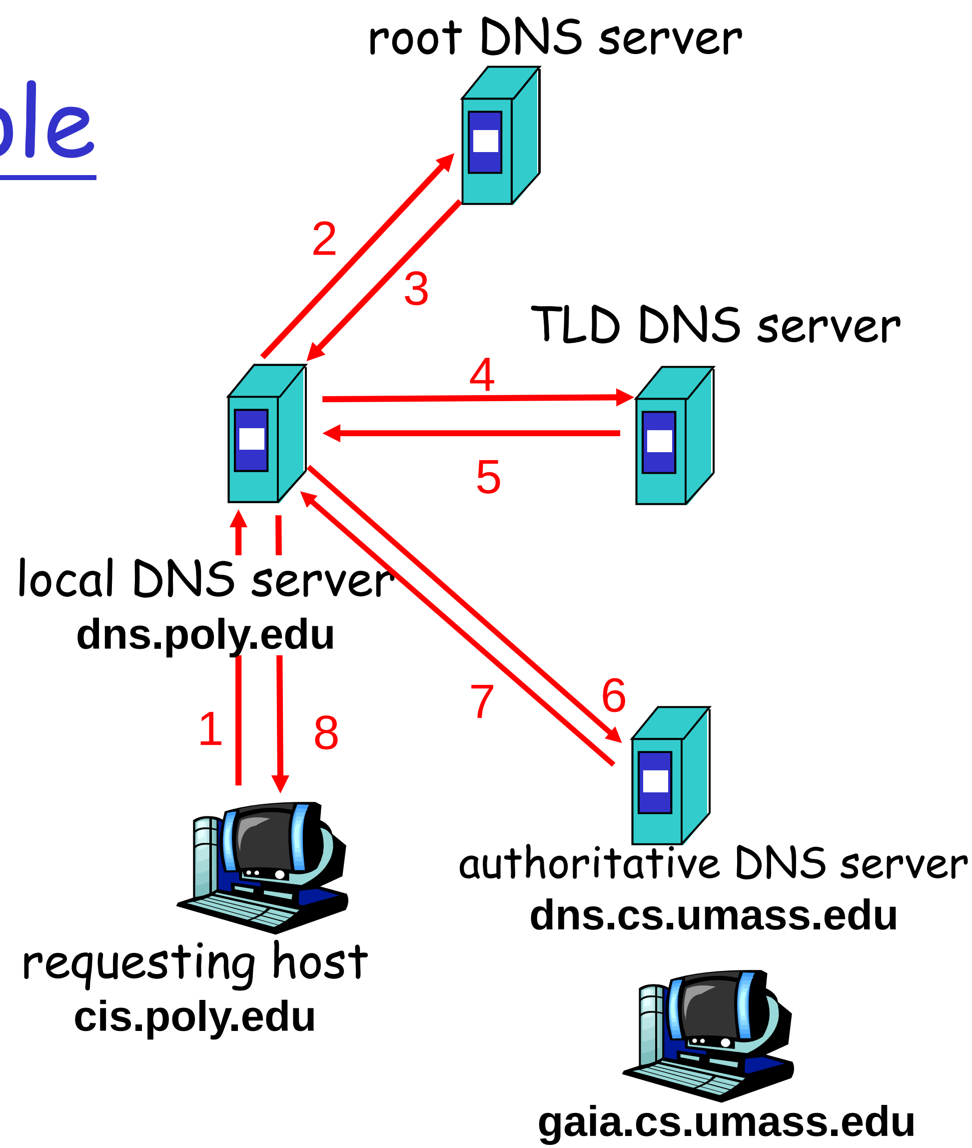
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one.
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - acts as proxy, forwards query into hierarchy

DNS name resolution example

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

iterated query:

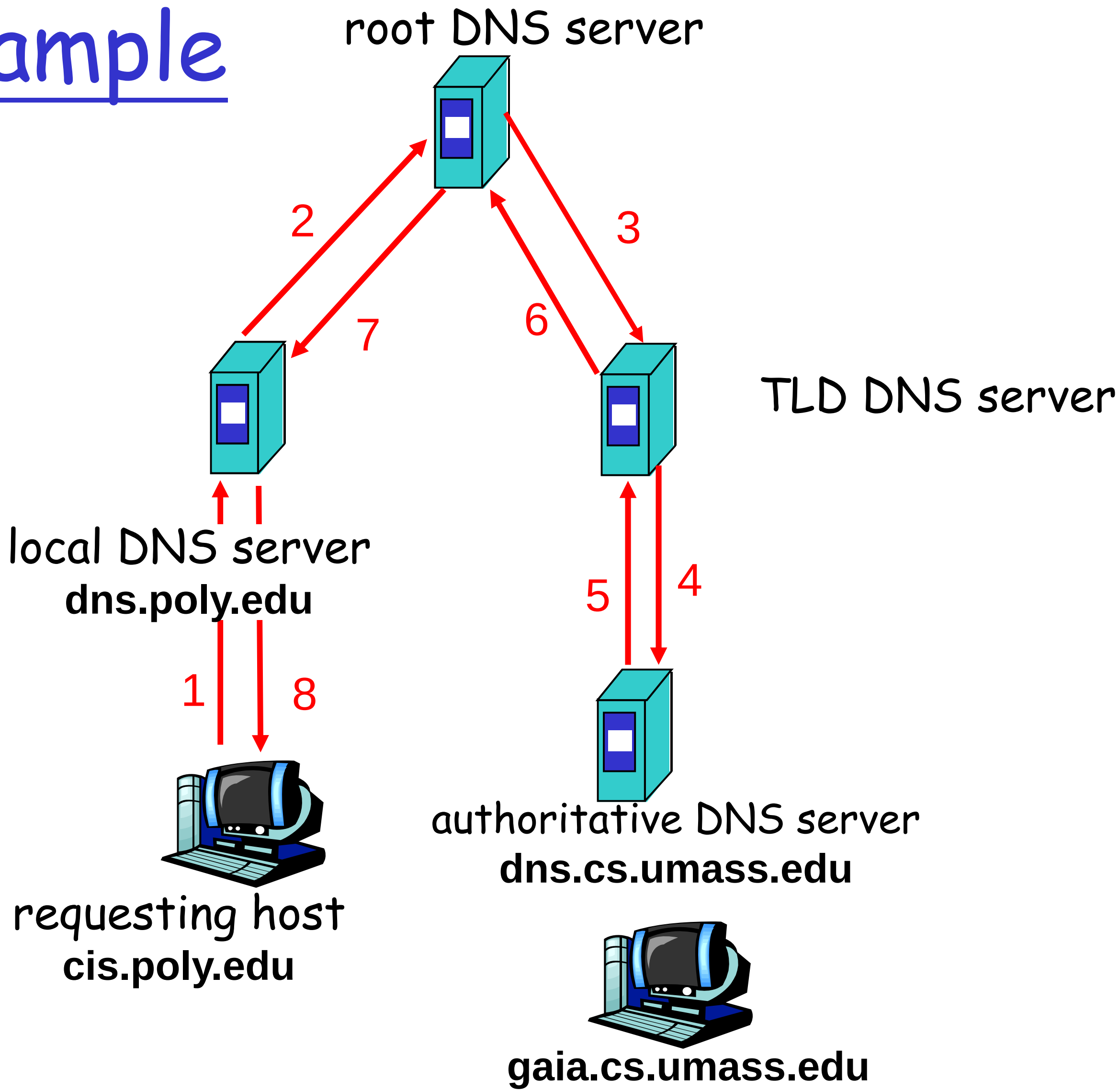
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example

recursive query:

- puts burden of name resolution on contacted name server
- heavy load?



DNS: caching and updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time
  - TLD servers typically cached in local name servers
    - Thus root name servers not often visited
- update/notify mechanisms under design by IETF
  - RFC 2136
  - http://www.ietf.org/html.charters/dnsind-charter.html

DNS records

DNS: distributed db storing resource records (RR)

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

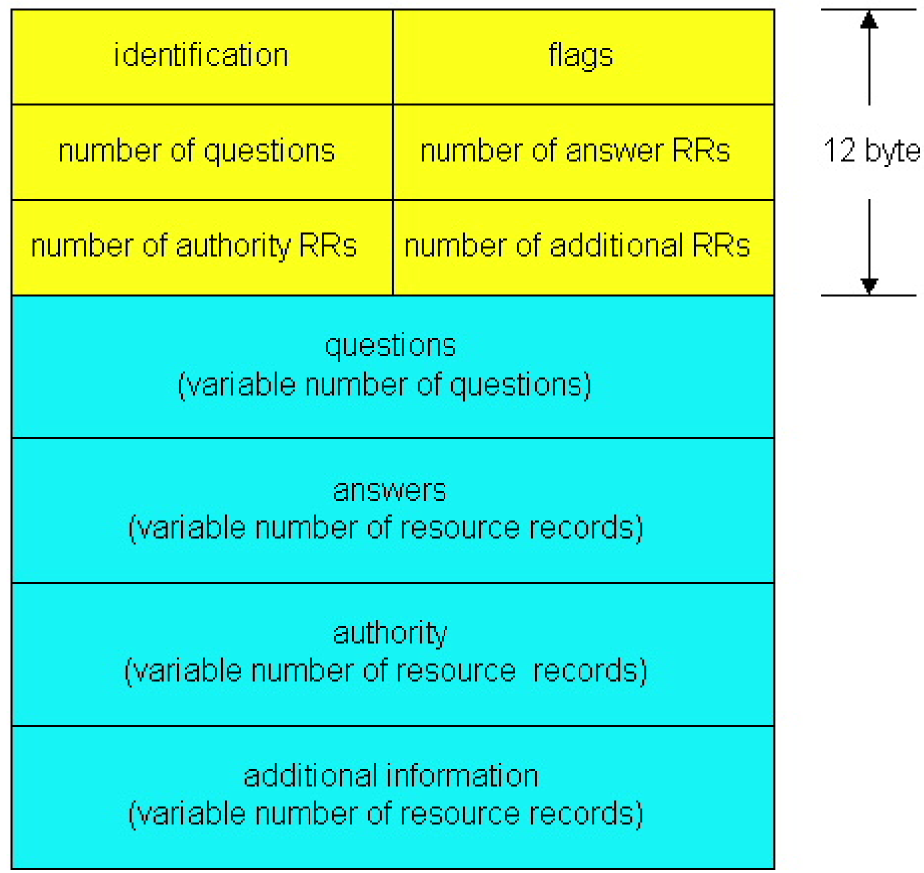
- Type=A
  - **name** is hostname
  - **value** is IP address
- Type=NS
  - **name** is domain (e.g. foo.com)
  - **value** is hostname of authoritative name server for this domain
- Type=CNAME
  - **name** is alias name for some "canonical" (the real) name
    `www.ibm.com` is really `servereast.backup2.ibm.com`
  - **value** is canonical name
- Type=MX
  - **value** is name of mailserver associated with **name**

DNS protocol, messages

DNS protocol : *query* and *reply* messages, both with same *message format*

msg header

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



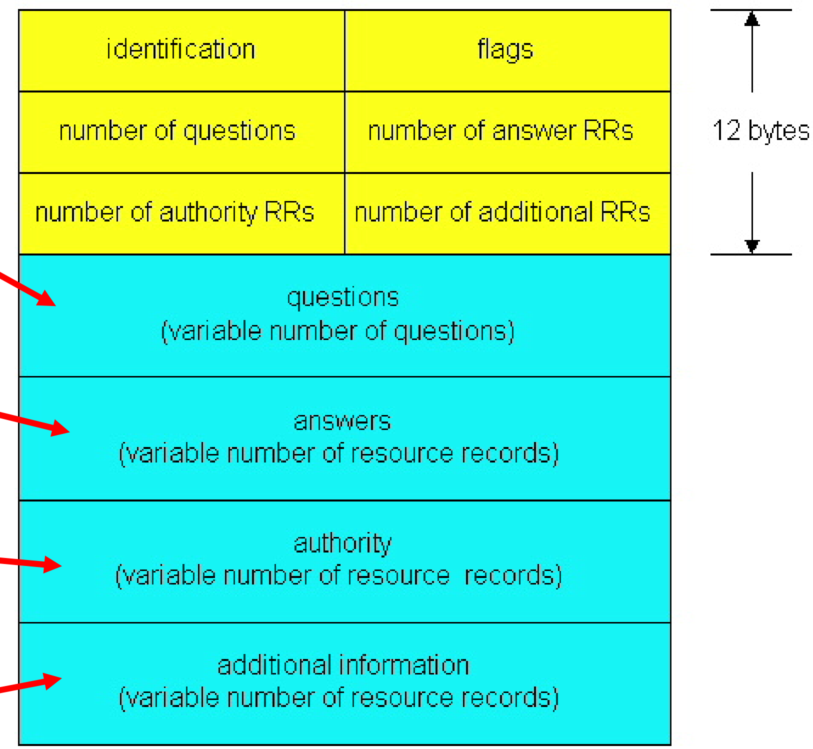
DNS protocol, messages

Name, type fields for a query

RRs in response to query

records for authoritative servers

additional "helpful" info that may be used



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 name 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)
```

- create authoritative server Type A record for www.networkuptopia.com; Type MX record for networkutopia.com
- How do people get IP address of your Web site?

Chapter 2: Application layer

- 2.1 Principles of network applications
  - app architectures
  - app requirements
- 2.2 Web and HTTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
- 2.9 Building a Web server

P2P file sharing

Example

- Alice runs P2P client application on her notebook computer
- intermittently connects to Internet; gets new IP address for each connection
- asks for "Hey Jude"
- application displays other peers that have copy of Hey Jude.
- Alice chooses one of the peers, Bob.
- file is copied from Bob's PC to Alice's notebook: HTTP
- while Alice downloads, other users uploading from Alice.
- Alice's peer is both a Web client and a transient Web server.

All peers are servers = highly scalable!

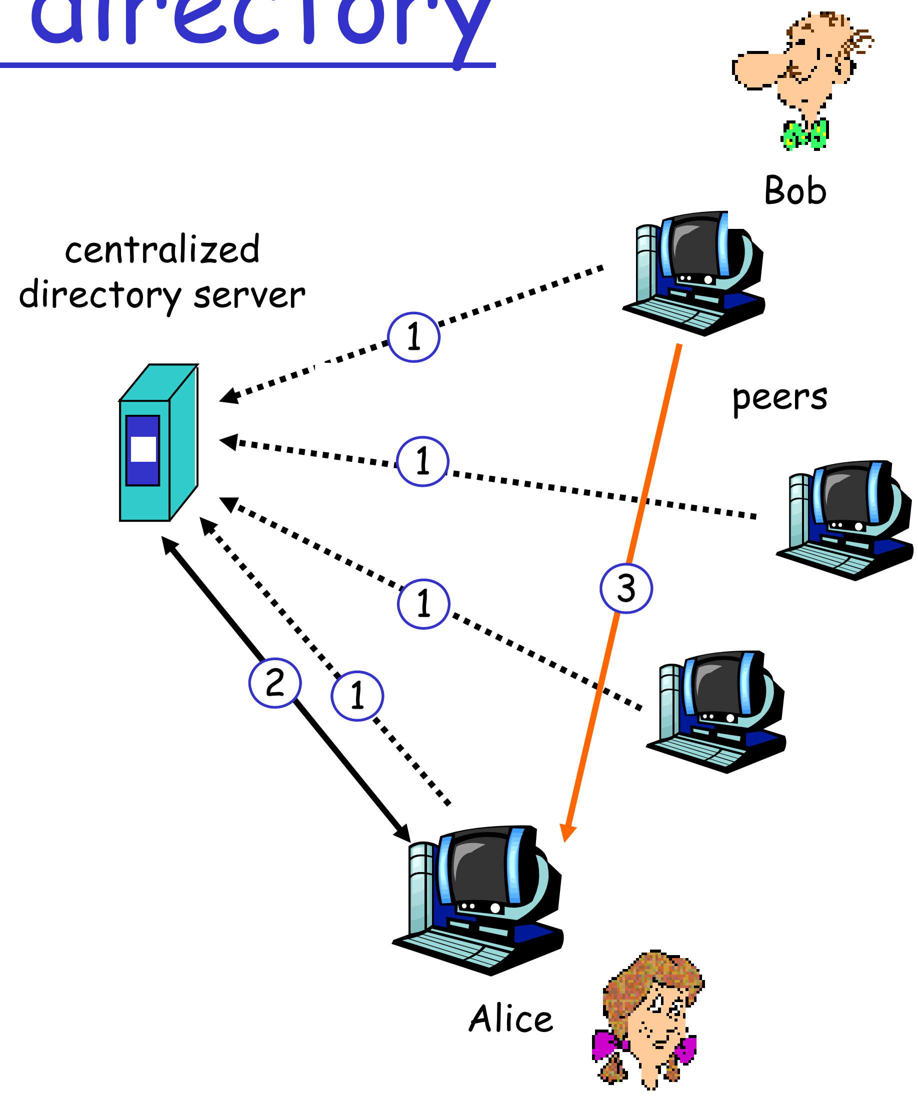
P2P: centralized directory

original "Napster" design

1) when peer connects, it informs central server:
   - IP address
   - content

2) Alice queries for "Hey Jude"

3) Alice requests file from Bob



P2P: problems with centralized directory

- single point of failure
- performance bottleneck
- copyright infringement: "target" of lawsuit is obvious

file transfer is decentralized, but locating content is highly centralized

Query flooding: Gnutella

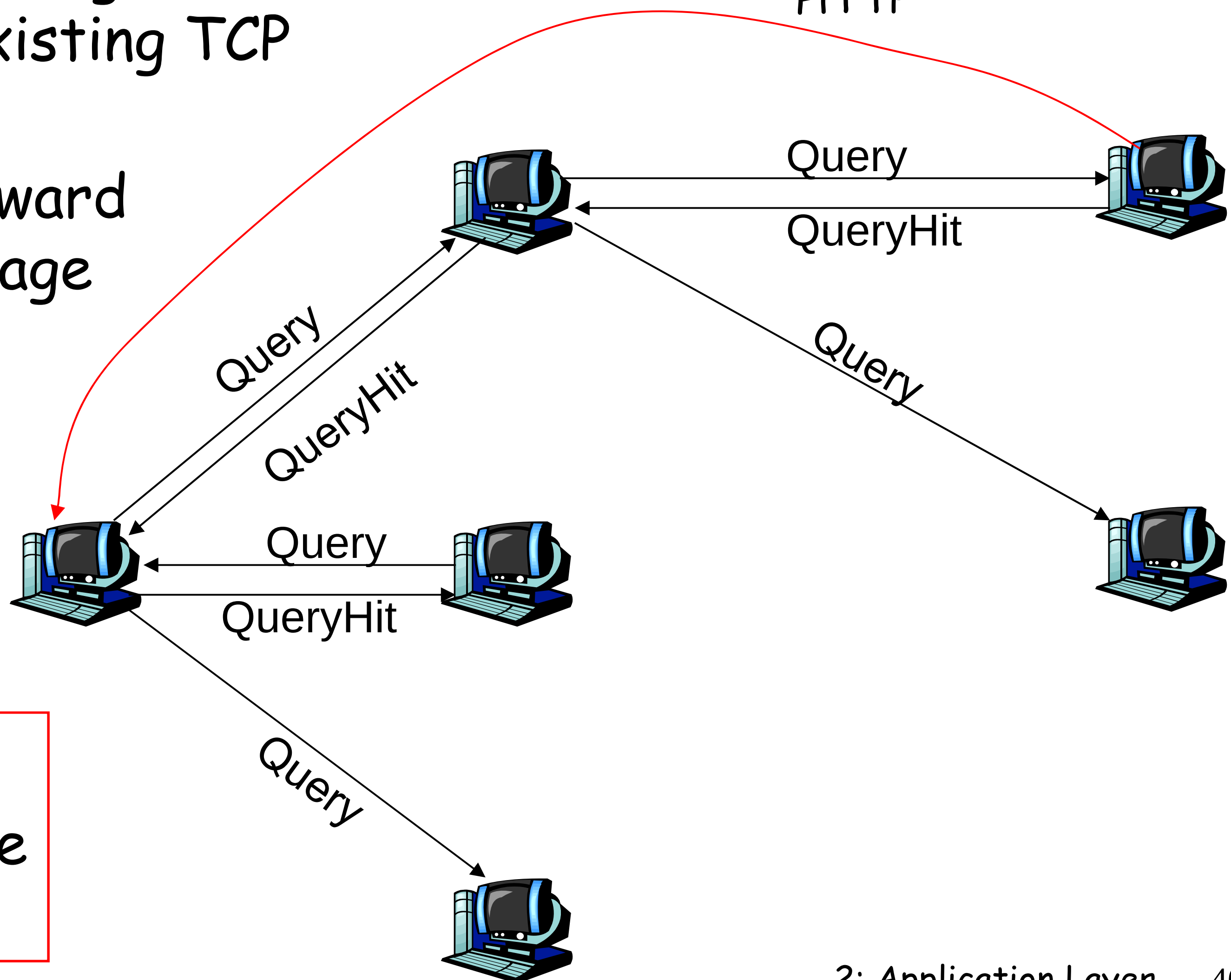
- fully distributed
  - no central server
- public domain protocol
- many Gnutella clients implementing protocol

overlay network: graph

- edge between peer X and Y if there's a TCP connection
- all active peers and edges form overlay net
- edge: virtual (*not* physical) link
- given peer typically connected with < 10 overlay neighbors

Gnutella: protocol

- Query message sent over existing TCP connections
- peers forward Query message
- QueryHit sent over reverse path



Scalability:
limited scope flooding

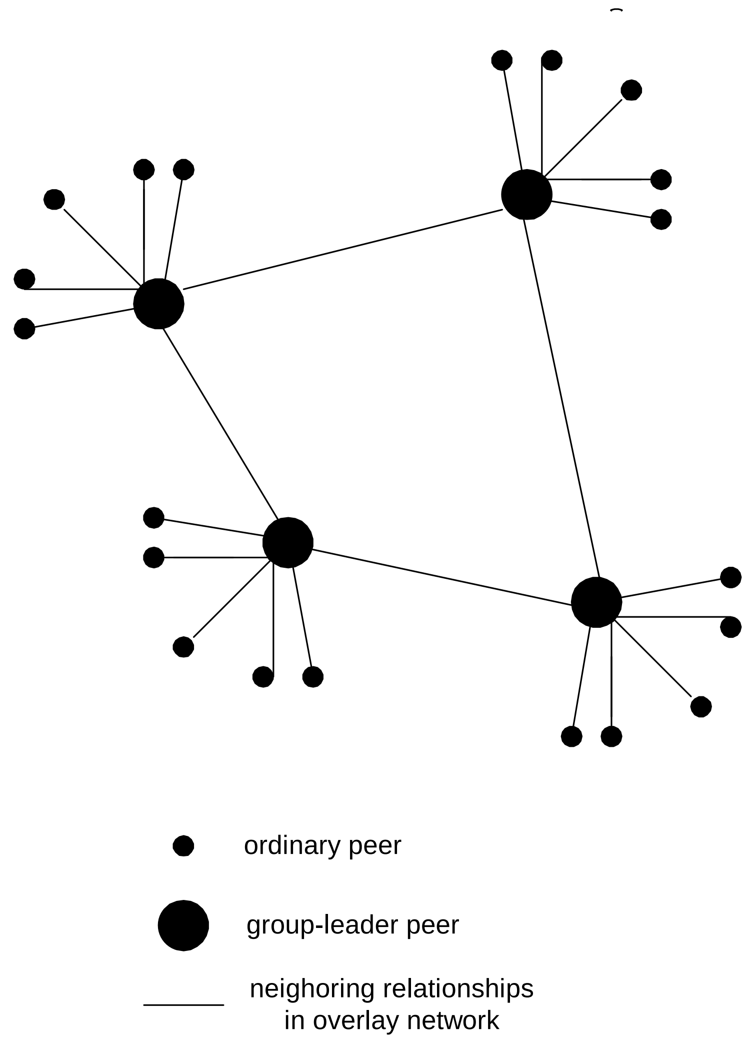
Gnutella: Peer joining

1. joining peer Alice must find another peer in Gnutella network: use list of candidate peers
2. Alice sequentially attempts TCP connections with candidate peers until connection setup with Bob
3. *Flooding:* Alice sends Ping message to Bob; Bob forwards Ping message to his overlay neighbors (who then forward to their neighbors….)
   - peers receiving Ping message respond to Alice with Pong message
4. Alice receives many Pong messages, and can then setup additional TCP connections

Peer leaving: see homework problem!

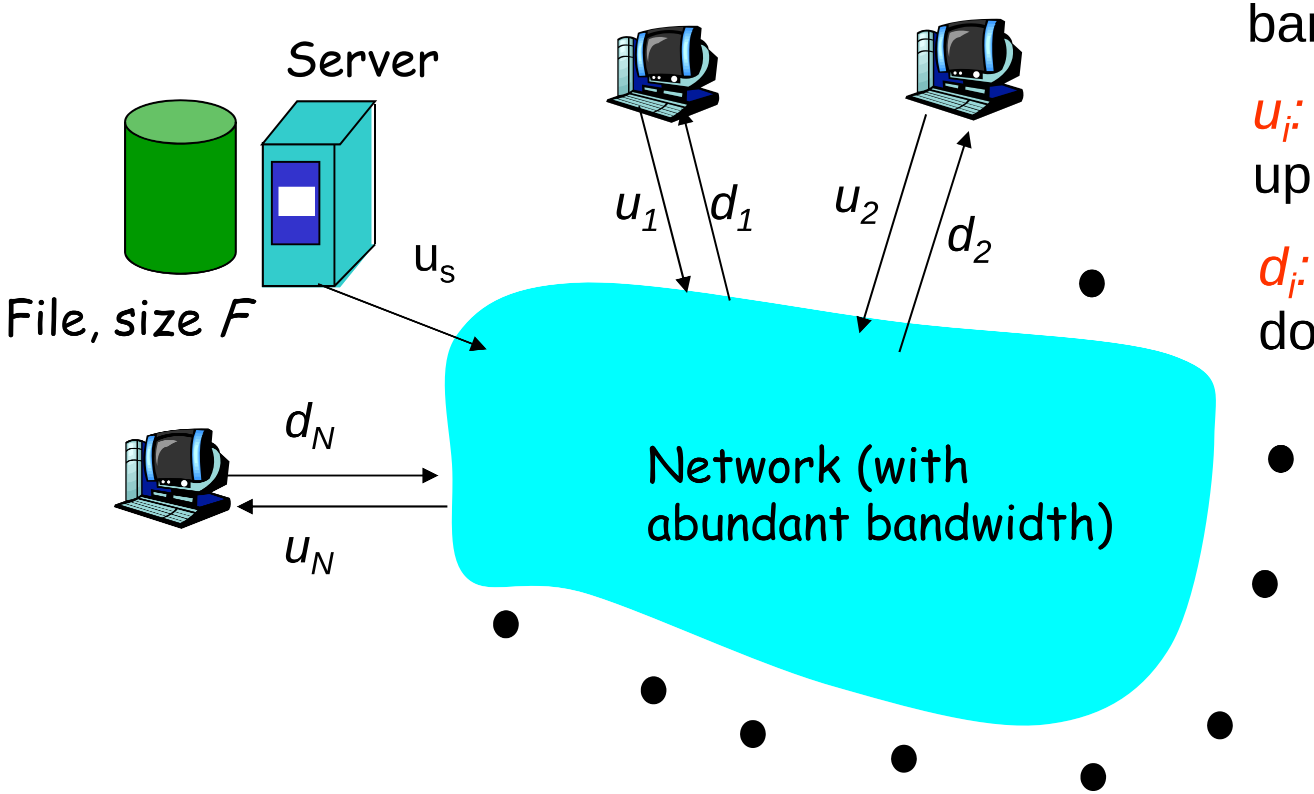
Hierarchical Overlay

- between centralized index, query flooding approaches
- each peer is either a *group leader* or assigned to a group leader.
  - TCP connection between peer and its group leader.
  - TCP connections between some pairs of group leaders.
- group leader tracks content in its children



Comparing Client-server, P2P architectures

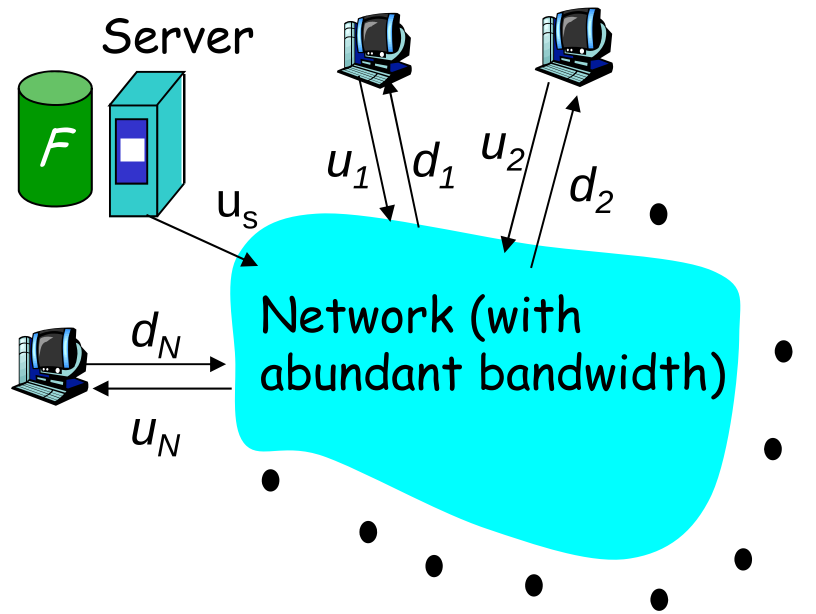
Question : How much time distribute file initially at one server to *N* other computers?



- u_s: server upload bandwidth
- u_i: client/peer i upload bandwidth
- d_i: client/peer i download bandwidth

Client-server: file distribution time

- server sequentially sends N copies:
  - NF/u_s time
- client i takes F/d_i time to download

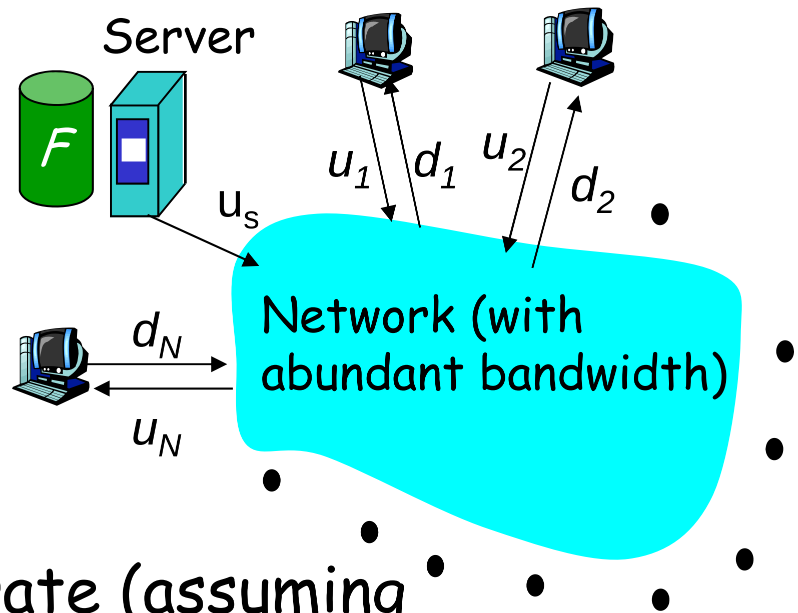


Time to distribute F to N clients using client/server approach $= d_{cs} = \max_i \{ NF/u_s, F/\min(d_i) \}$

increases linearly in N (for large N)

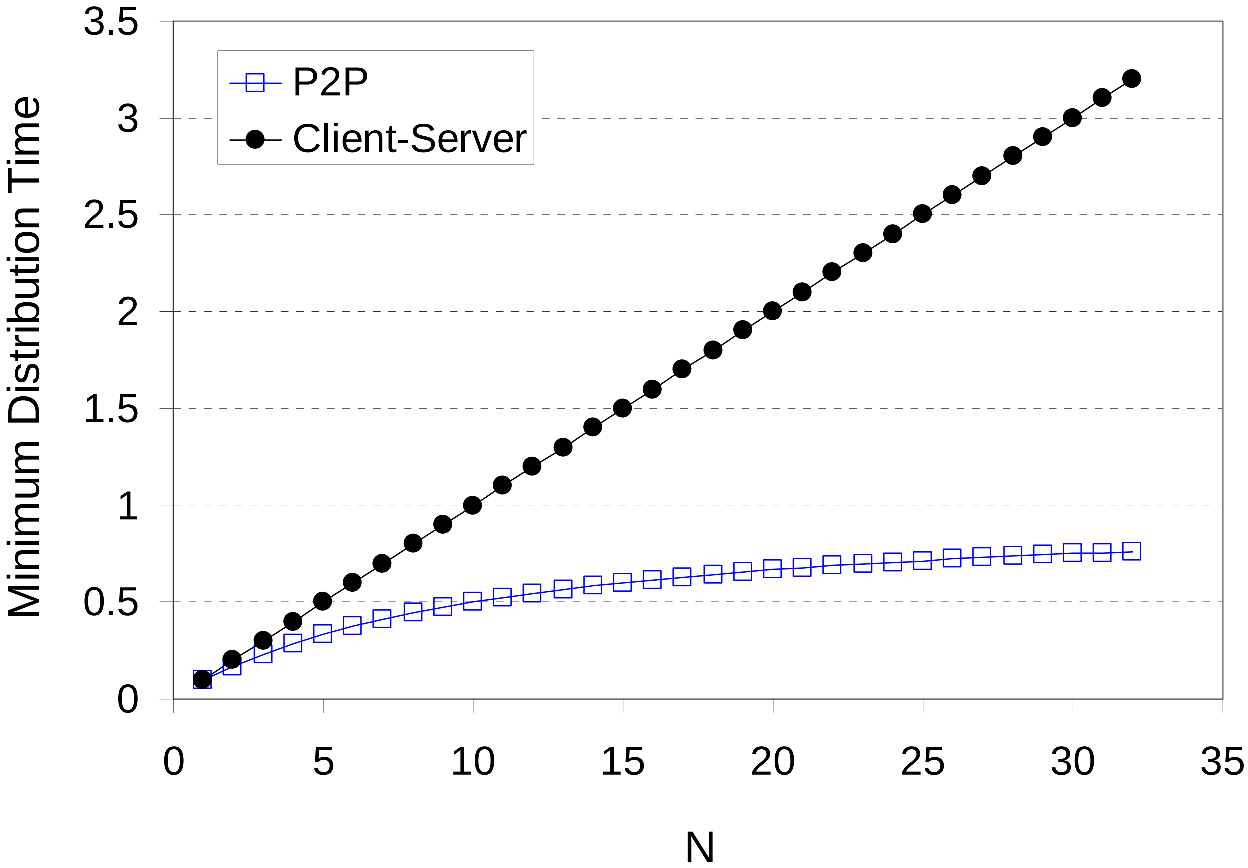
P2P: file distribution time

- server must send one copy: F/u_s time
- client i takes F/d_i time to download
- NF bits must be downloaded (aggregate)
  - fastest possible upload rate (assuming all nodes sending file chunks to same peer): $u_s + \sum_{i=1,N} u_i$



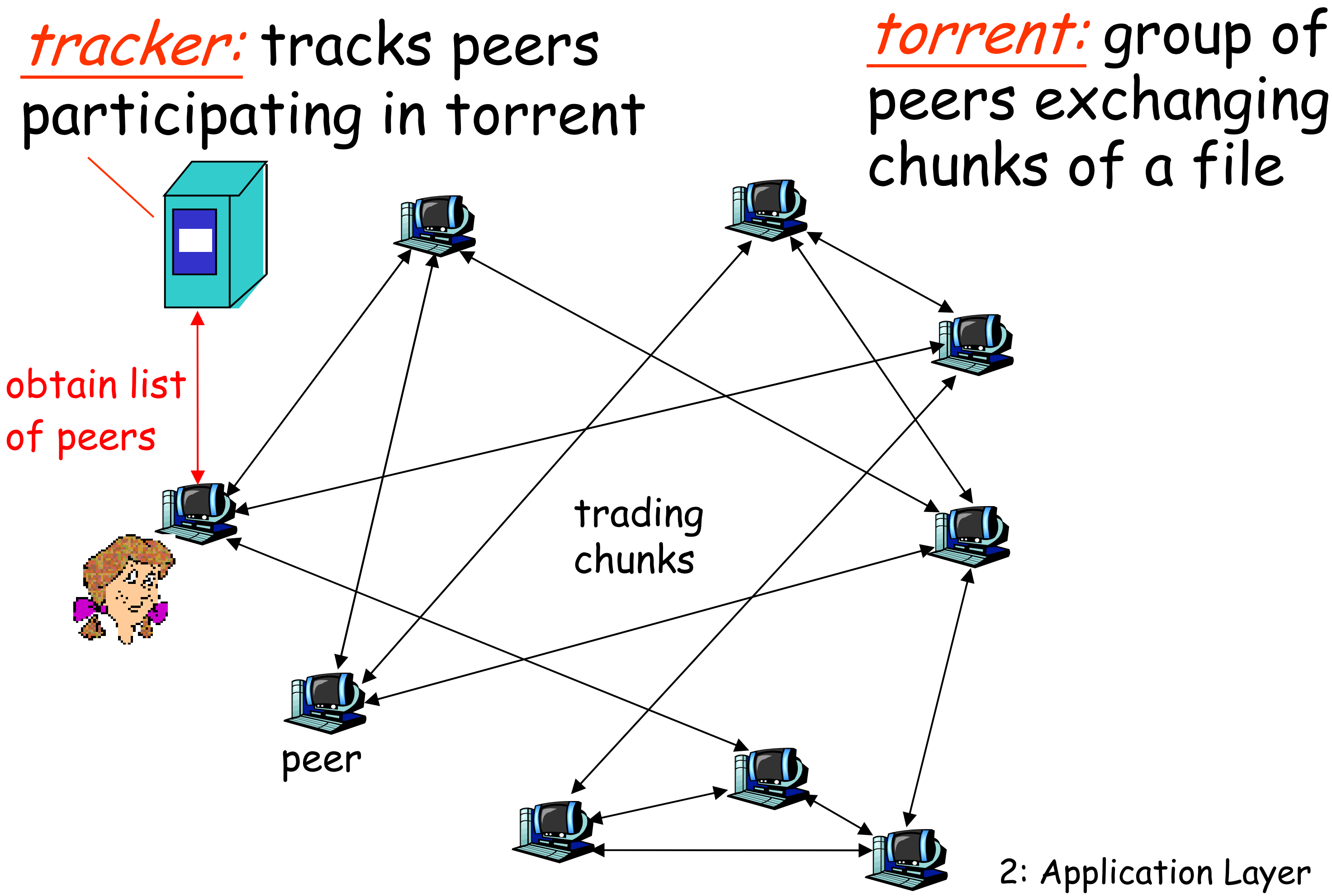
$$d_{P2P} = \max \left\{ F/u_s,\ F/\min_i(d_i),\ NF/(u_s + \sum_{i=1,N} u_i) \right\}$$

Comparing Client-server, P2P architectures

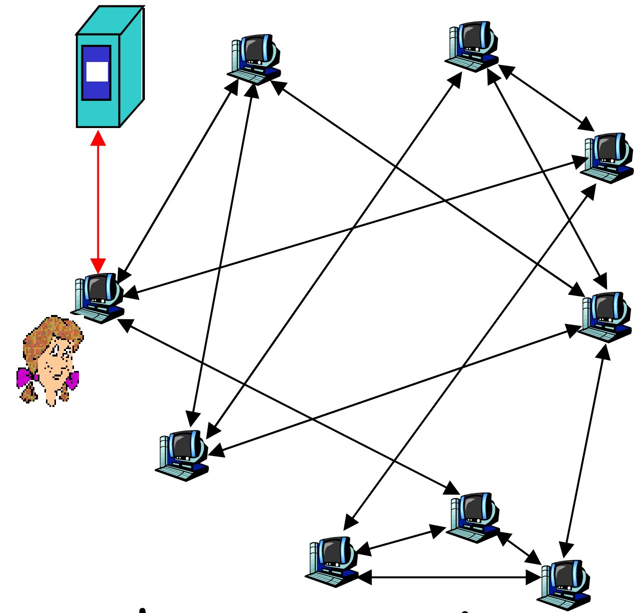


P2P Case Study: BitTorrent

- P2P file distribution



BitTorrent (1)



- file divided into 256KB *chunks*.
- peer joining torrent:
  - has no chunks, but will accumulate them over time
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers.
- peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain

BitTorrent (2)

Pulling Chunks

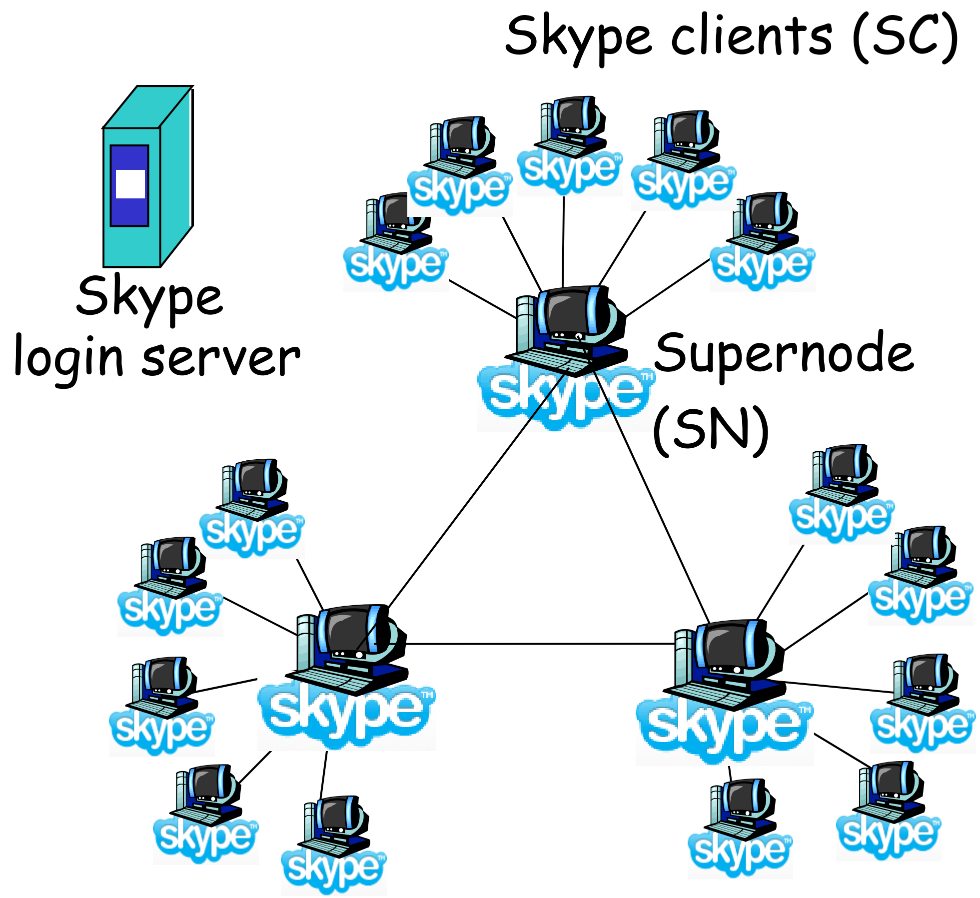
- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice issues requests for her missing chunks
  - rarest first

Sending Chunks: tit-for-tat

- Alice sends chunks to four neighbors currently sending her chunks *at the highest rate*
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - newly chosen peer may join top 4

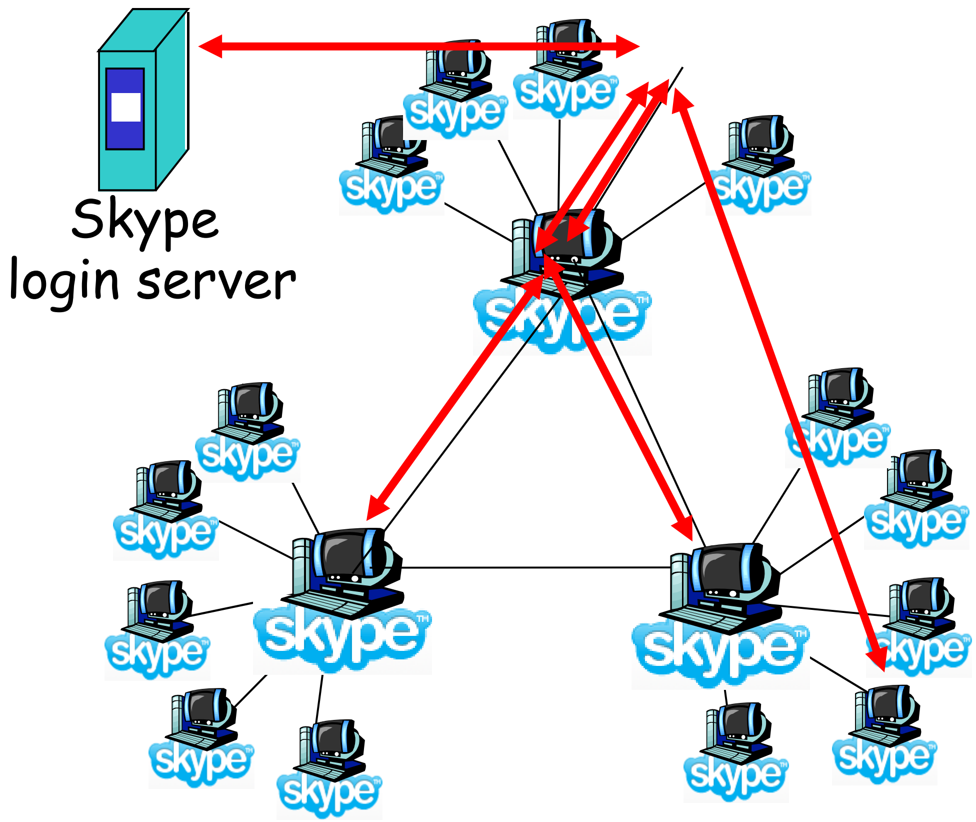
P2P Case study: Skype

- P2P (pc-to-pc, pc-to-phone, phone-to-pc) Voice-Over-IP (VoIP) application
  - also IM
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay



Skype: making a call

- User starts Skype
- SC registers with SN
  - list of bootstrap SNs
- SC logs in (authenticate)
- Call: SC contacts SN will callee ID
  - SN contacts other SNs (unknown protocol, maybe flooding) to find addr of callee; returns addr to SC
- SC directly contacts callee, overTCP



Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP