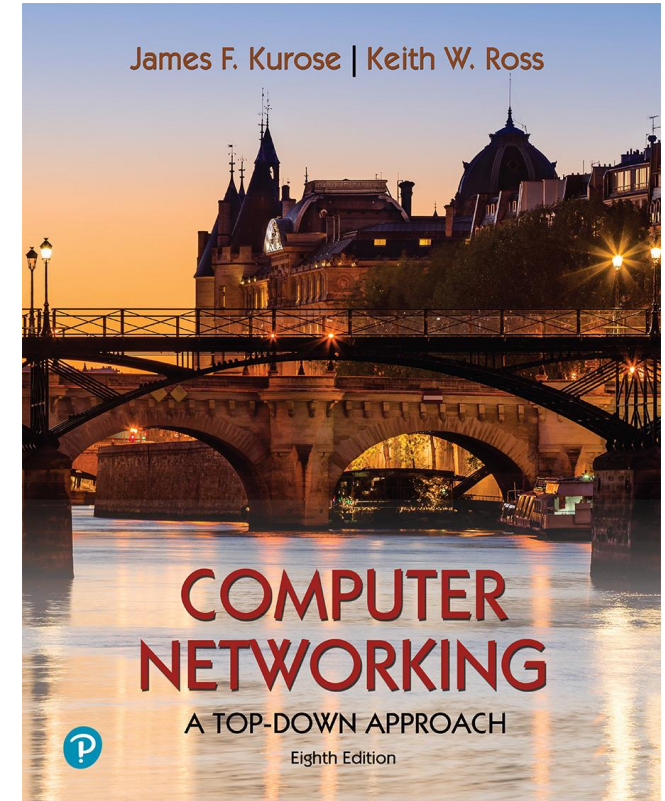# Chapter 2
Application Layer

Computer Networking: A Top-Down Approach

8th edition   n
Jim Kurose, Keith Ross
Pearson, 2020

Application layer: overview

- Principles of network applications

- Web and HTTP

- **E-mail, SMTP, IMAP**

- The Domain Name System DNS

- P2P applications

- video streaming and content distribution networks
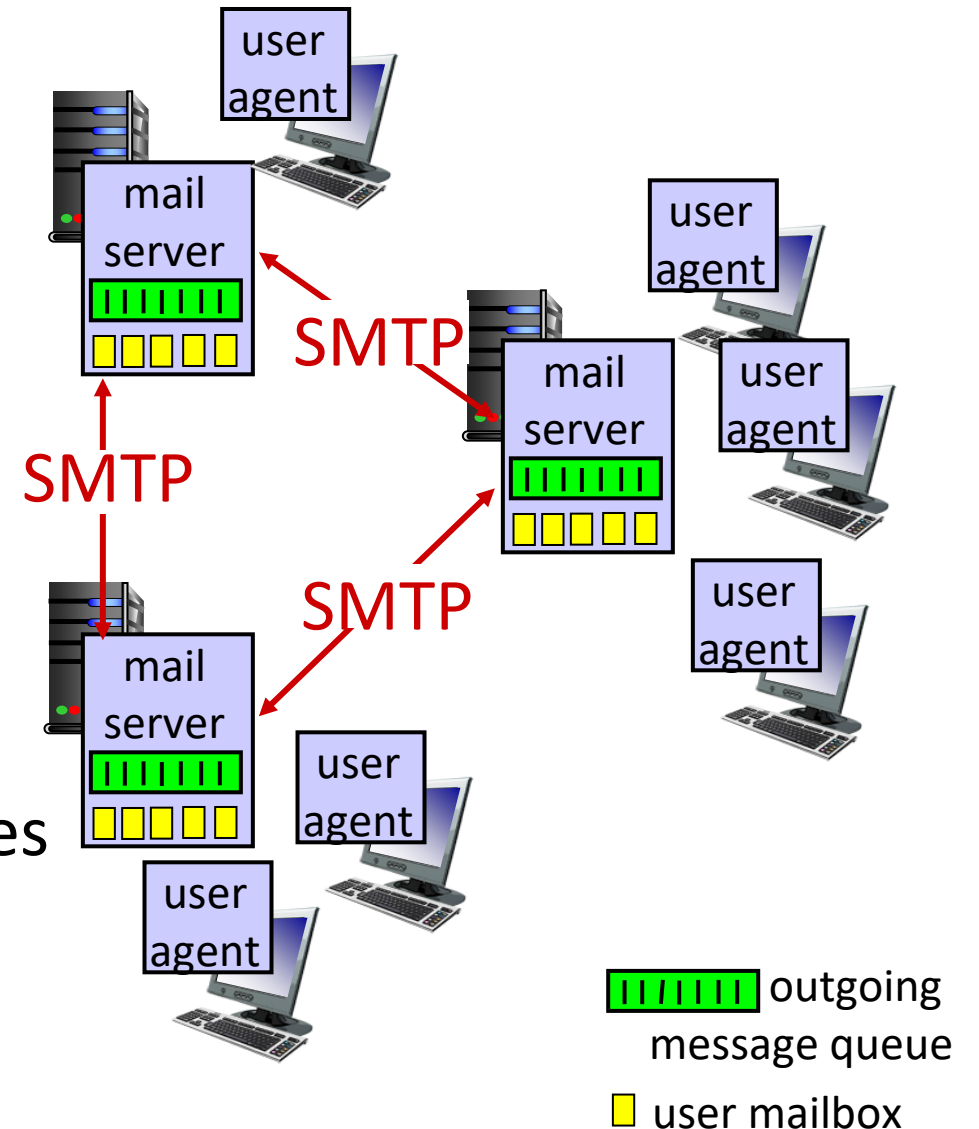
- socket programming with UDP and TCP

E-mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, iPhone mail client
- outgoing, incoming messages stored on server

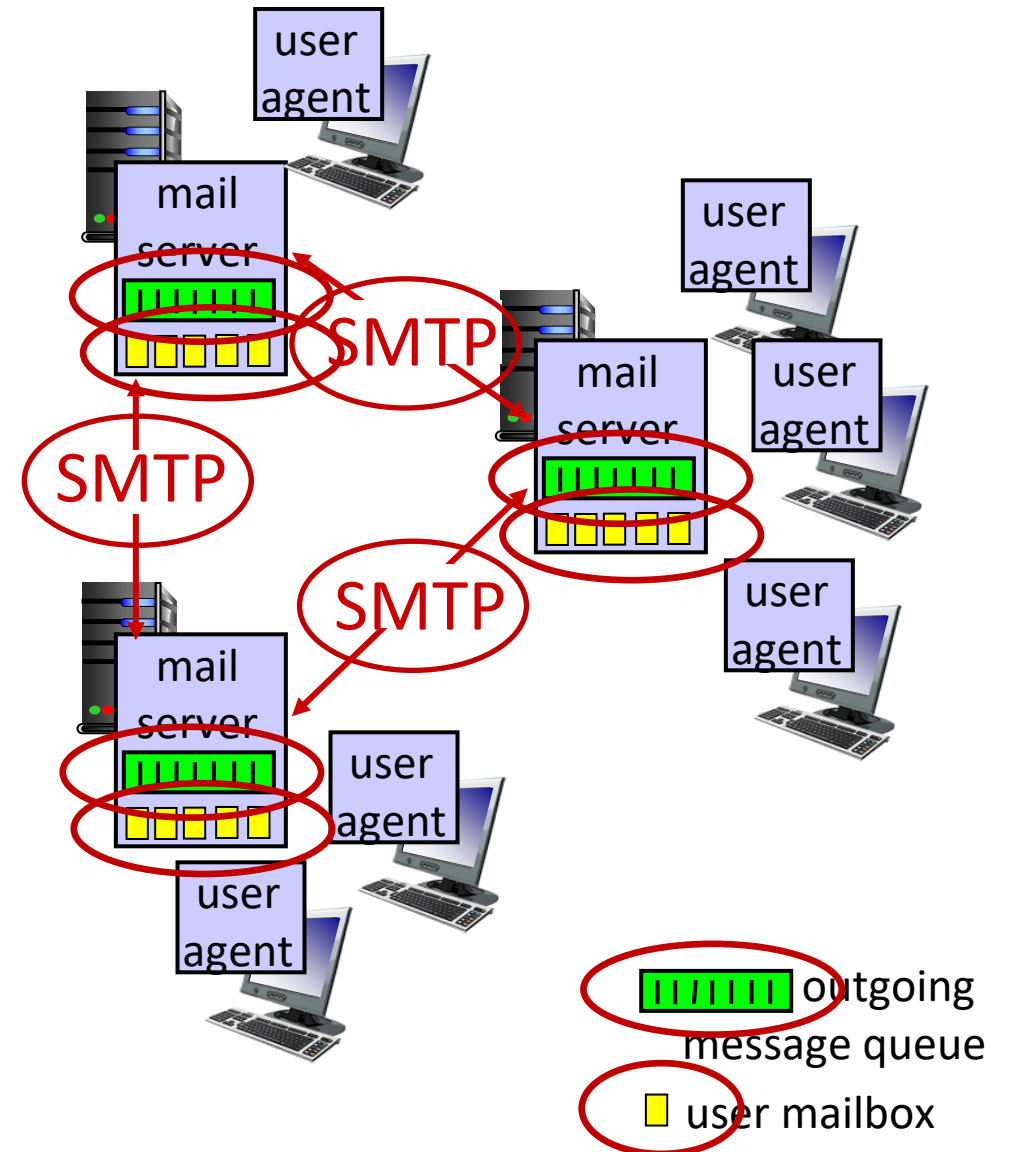
outgoing message queue

user mailbox

E-mail: mail servers

mail servers:

- *mailbox* contains incoming messages for user
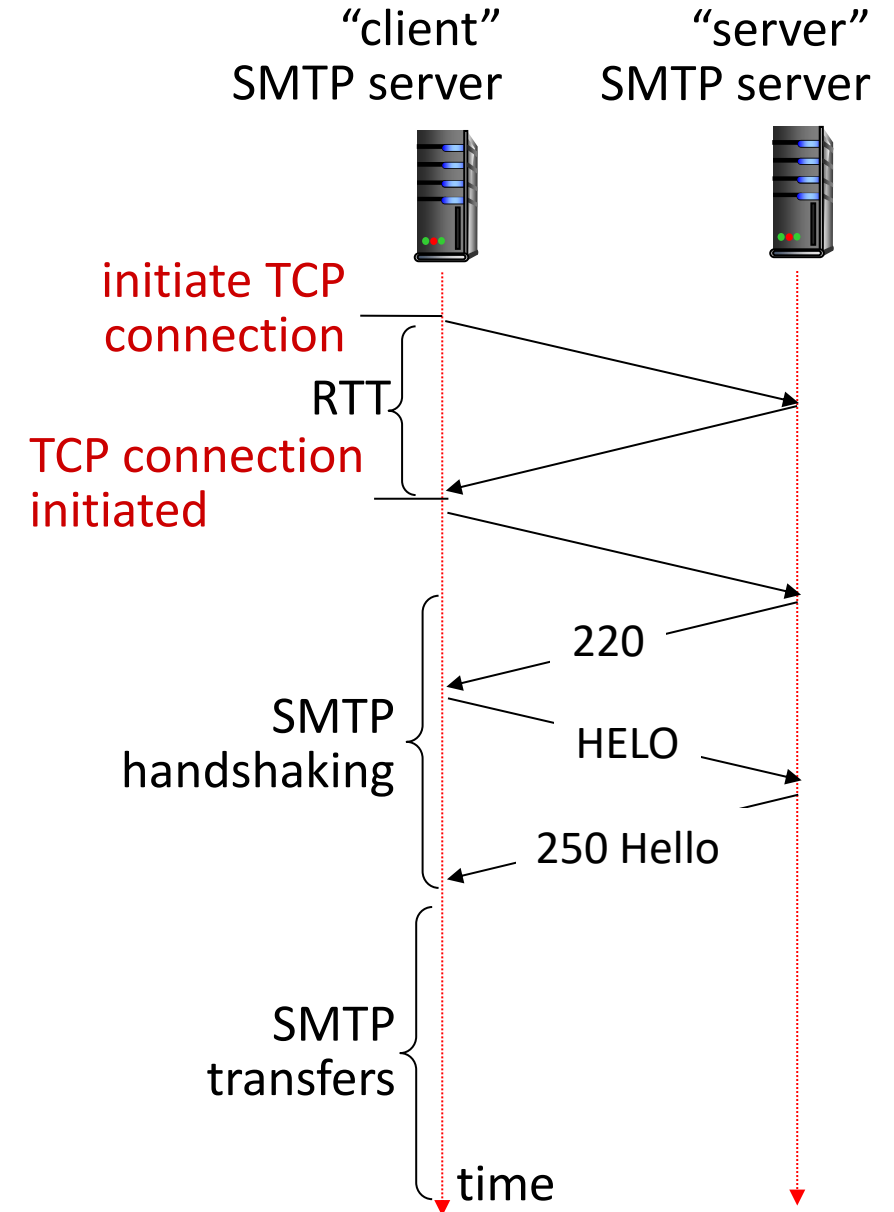- *message queue* of outgoing (to be sent) mail messages

SMTP protocol between mail servers to send email messages

- client: sending mail server
- "server": receiving mail server

SMTP RFC (5321)

- uses TCP to reliably transfer email message from client (mail server initiating connection) to server, port 25
  - direct transfer: sending server (acting like client) to receiving server
- three phases of transfer
  - SMTP handshaking (greeting)
  - SMTP transfer of messages
  - SMTP closure
- command/response interaction (like HTTP)
  - commands: ASCII text
  - response: status code and phrase

"client"
SMTP server

"server"
SMTP server

initiate TCP connection

RTT

TCP connection initiated

220

SMTP handshaking

HELO

250 Hello

SMTP transfers

time

Scenario: Alice sends e-mail 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 using SMTP; message placed in message queue

3) client side of SMTP at mail server 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

Alice's mail server

Bob's mail server
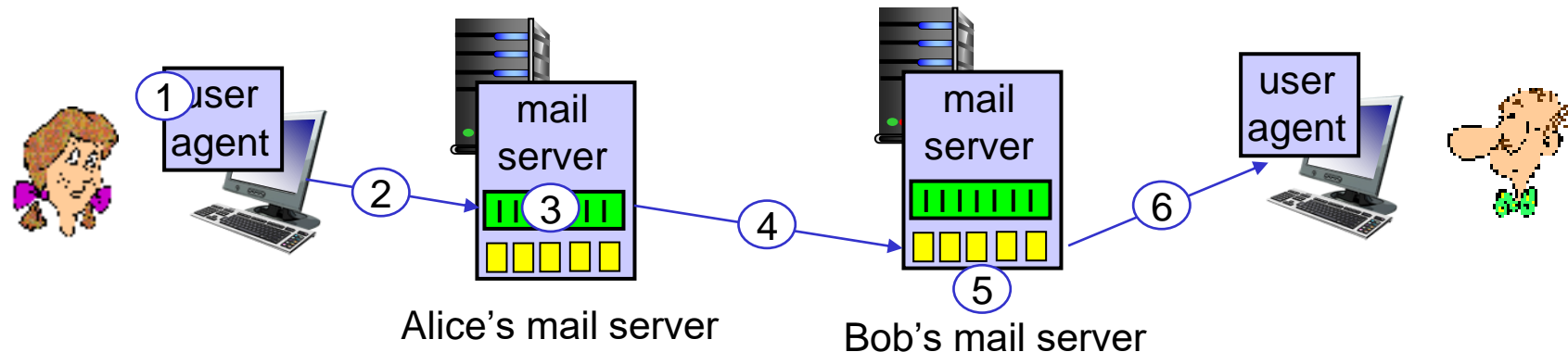
Sample SMTP interaction

```
S: 220 hamburger.edu
```

SMTP: observations

comparison with HTTP:

- HTTP: client pull

- SMTP: client push

- both have ASCII command/response interaction, status codes

- HTTP: each object encapsulated in its own response message

- SMTP: multiple objects sent in multipart message

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses CRLF.CRLF to determine end of message

Mail message format

SMTP: protocol for exchanging e-mail messages, defined in RFC 5321 (like RFC 7231 defines HTTP)

RFC 2822 defines *syntax* for e-mail message itself (like HTML defines syntax for web documents)

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

  these lines, within the body of the email message area different from SMTP MAIL FROM:, RCPT TO: commands!

- Body: the "message" , ASCII characters only

header

blank line

body
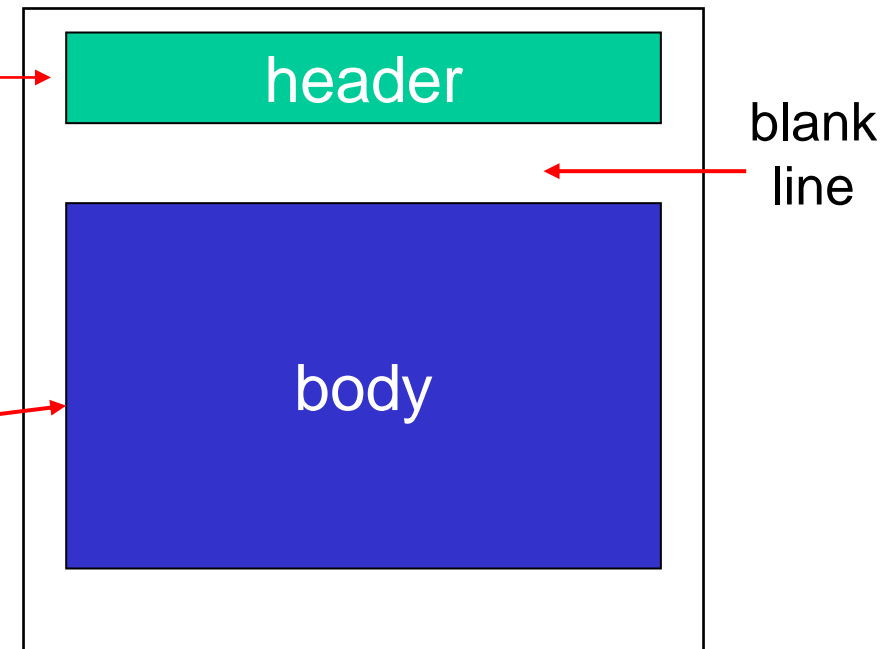
Retrieving email: mail access protocols
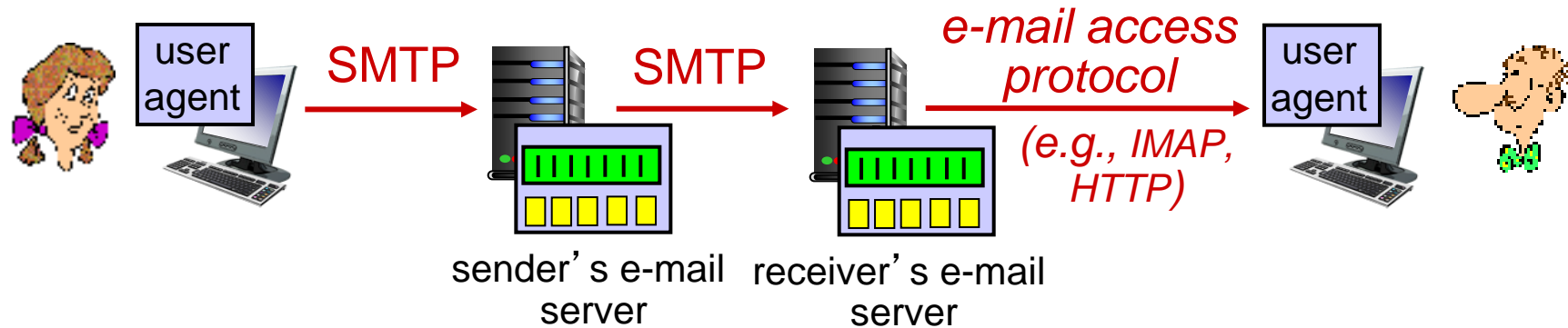
- **SMTP:** delivery/storage of e-mail messages to receiver's server

- mail access protocol: retrieval from server
  - **IMAP:** Internet Mail Access Protocol [RFC 3501]: messages stored on server, IMAP provides retrieval, deletion, folders of stored messages on server

- **HTTP:** gmail, Hotmail, Yahoo!Mail, etc. provides web-based interface on top of STMP (to send), IMAP (or POP) to retrieve e-mail messages

Application Layer: Overview

- Principles of network applications

- Web and HTTP

- E-mail, SMTP, IMAP

- The Domain Name System DNS

- **P2P applications**

- video streaming and content distribution networks

- socket programming with UDP and TCP
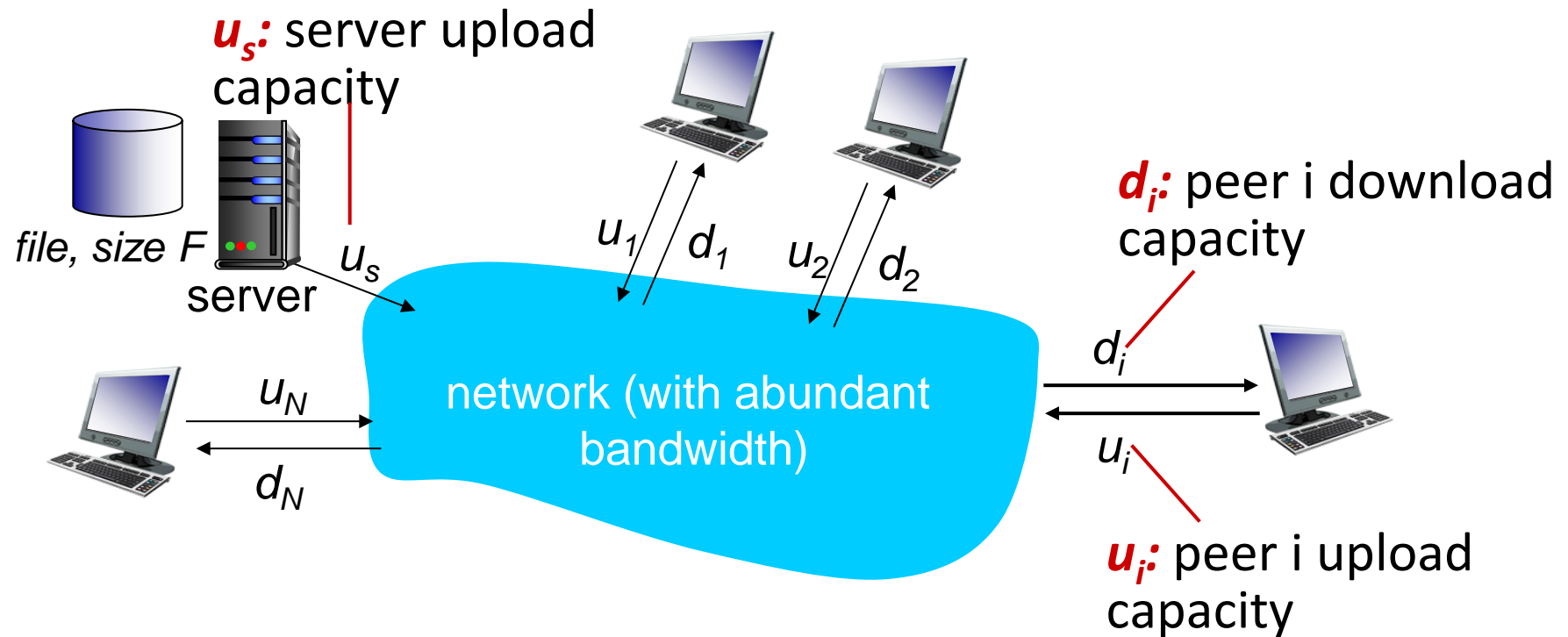
Peer-to-peer (P2P) architecture

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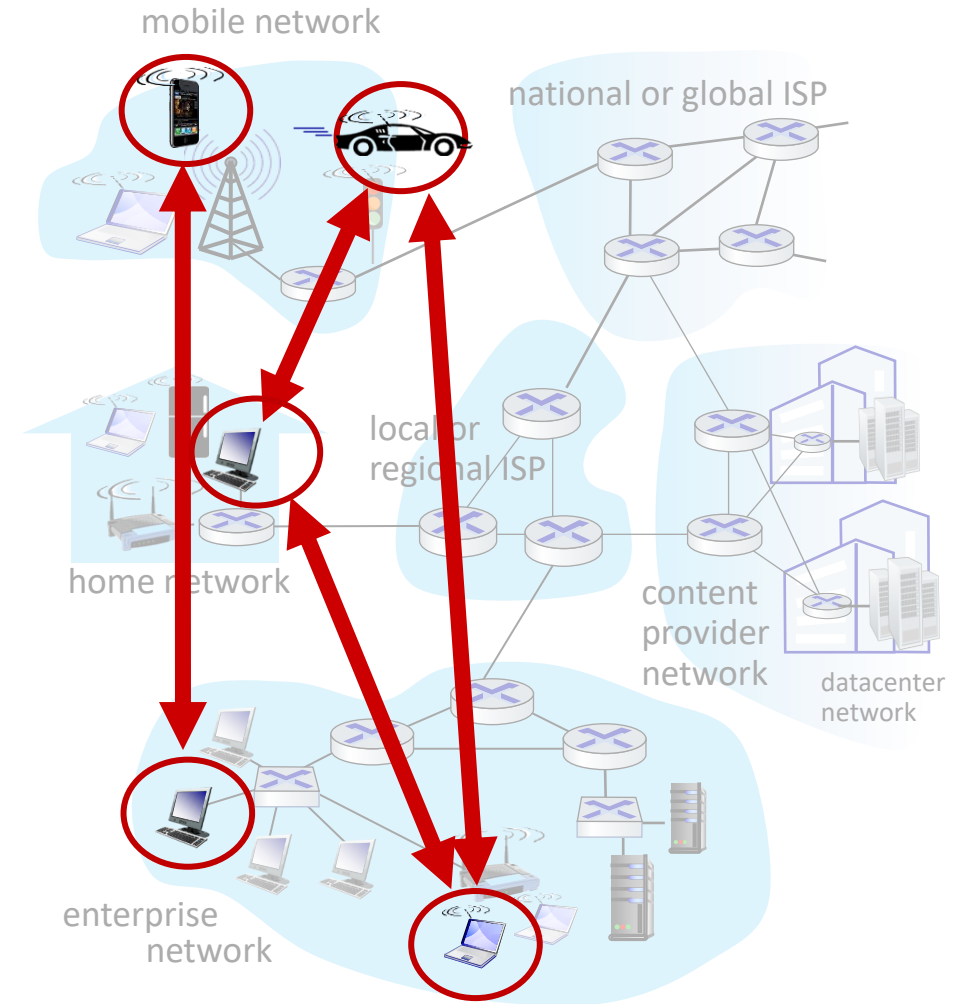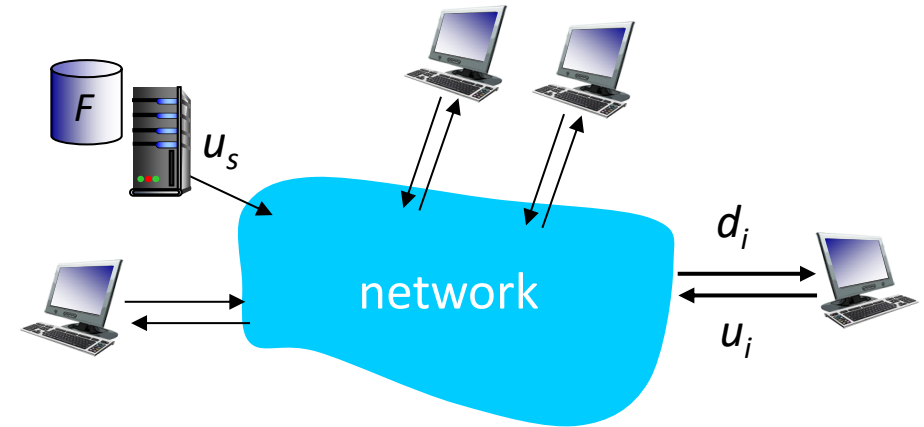
File distribution: client-server vs P2P

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

- peer upload/download capacity is limited resource

u_s: server upload capacity

file, size F

server

u_s

u_1 d_1 u_2 d_2

u_N

d_N

network (with abundant bandwidth)

d_i

u_i

d_i: peer i download capacity

u_i: peer i upload capacity

File distribution time: client-server

- *server transmission:* must sequentially send (upload) N file copies:
  - time to send one copy: F/u_s
  - time to send N copies: NF/u_s

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

$$
\boxed{
\begin{array}{l}
\textit{time to distribute } F \\
\textit{to N clients using} \\
\textit{client-server approach}
\end{array}
\quad D_{c\text{-}s} \geq max\{NF/u_s,F/d_{min}\}
}
$$

increases linearly in N

File distribution time: P2P

- *server transmission:* must upload at least one copy:
  - time to send one copy: F/u_s

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

- *clients:* as aggregate must download NF bits
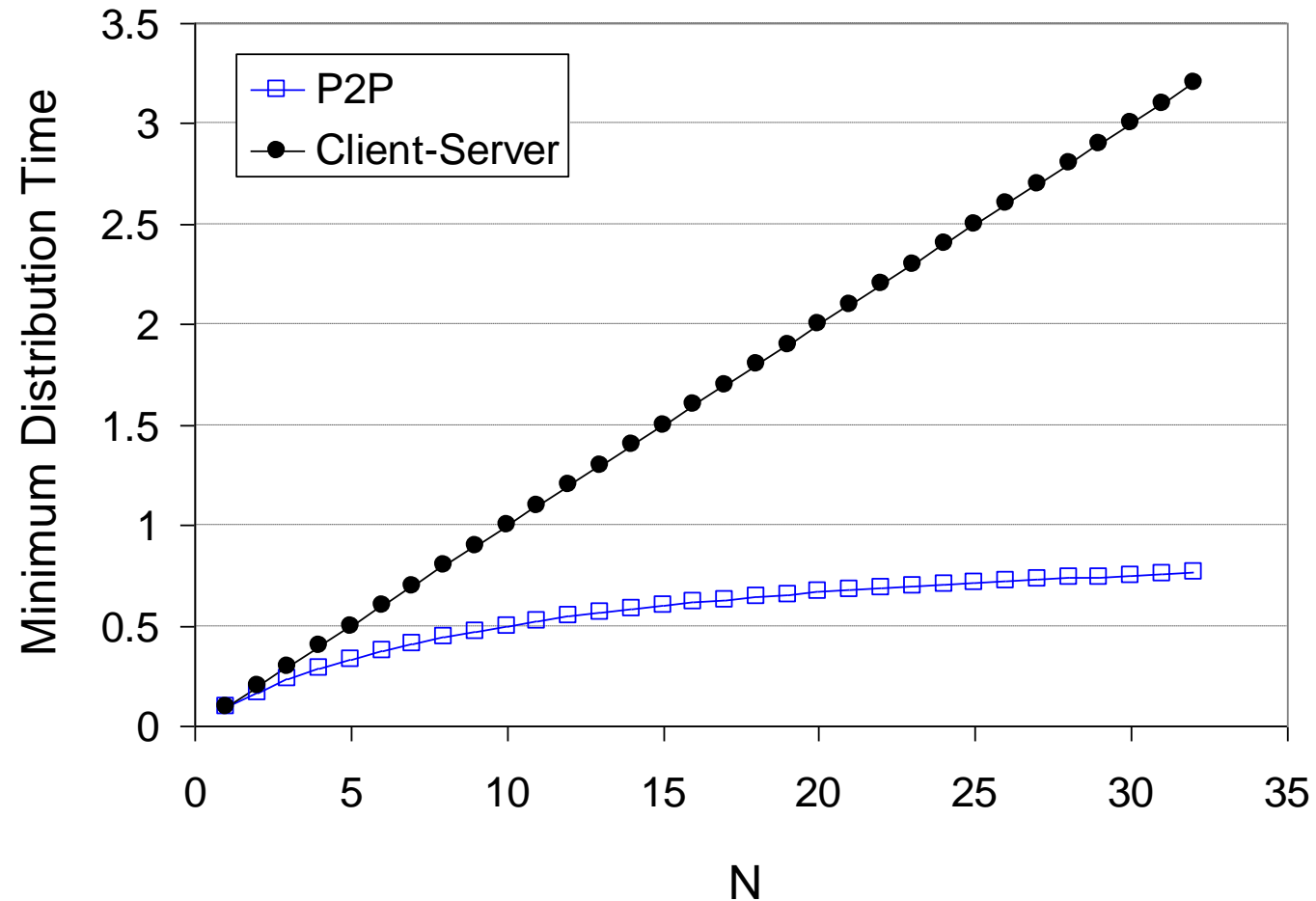 - max upload rate (limiting max download rate) is $u_s + \Sigma u_i$
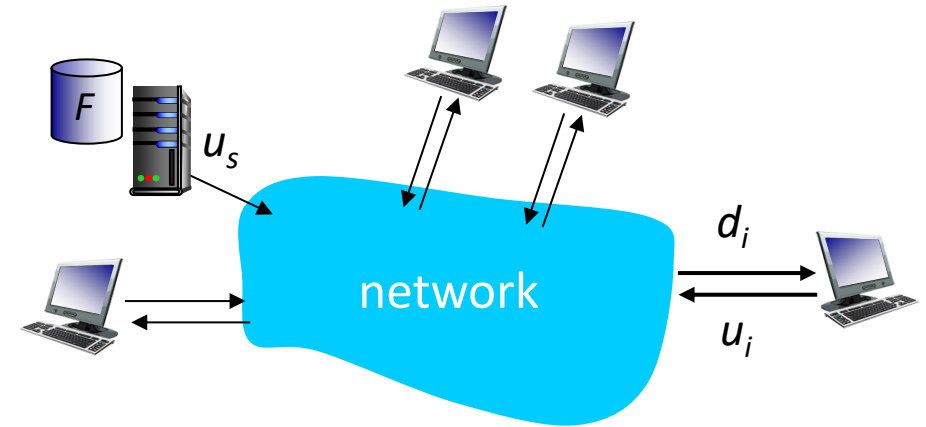
time to distribute F to N clients using P2P approach

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

increases linearly in N …

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

Client-server vs. P2P: example

client upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$
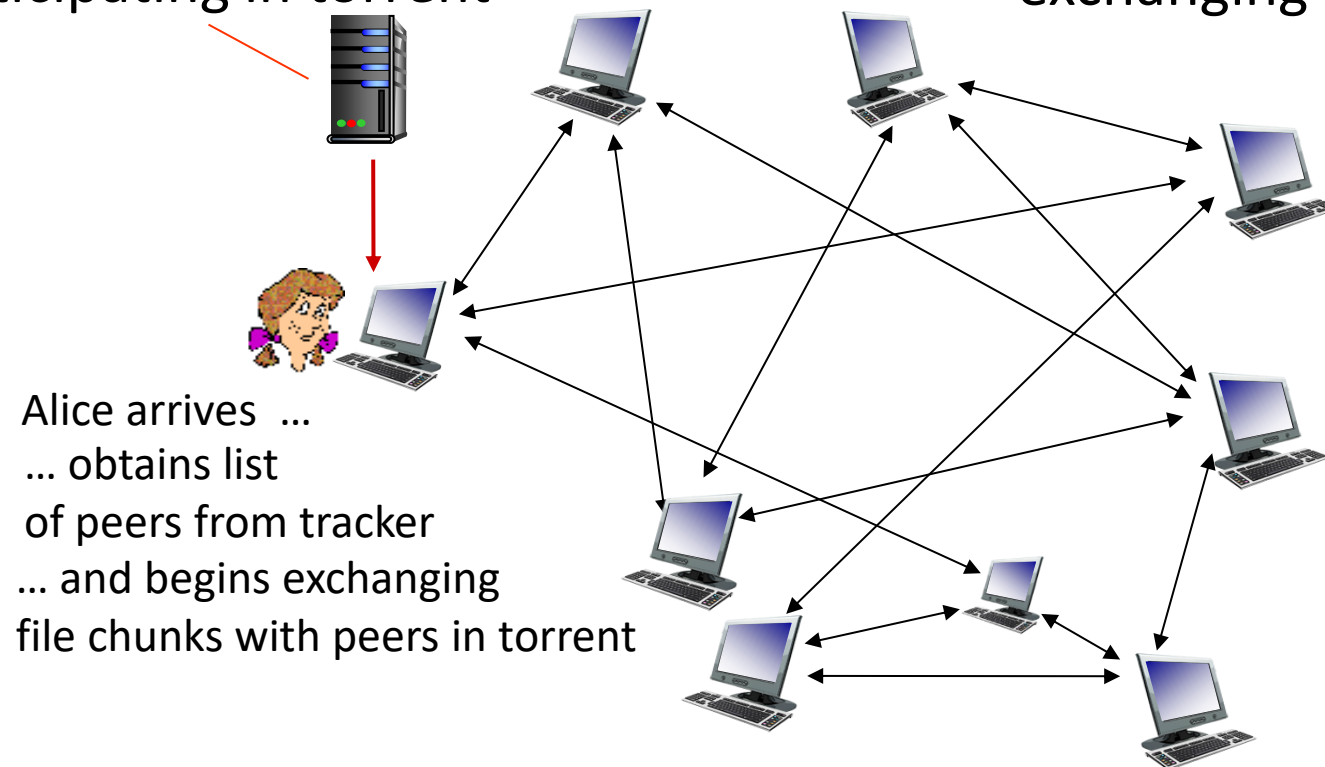
P2P file distribution: BitTorrent

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

tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file

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

P2P file distribution: BitTorrent

- peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn:* peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent
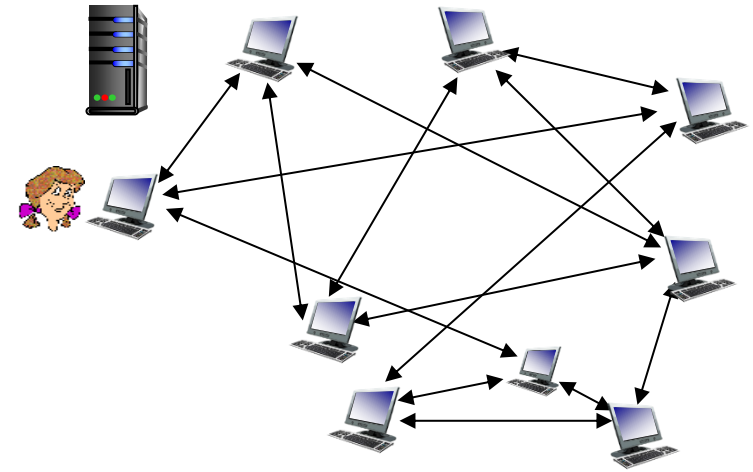
# BitTorrent: requesting, sending file chunks

Requesting chunks:

- at any given time, different peers have different subsets of file chunks

- periodically, Alice asks each peer for list of chunks that they have

- Alice requests missing chunks from peers, rarest first

Sending chunks: tit-for-tat

- Alice sends chunks to those four peers currently sending her chunks *at highest rate*
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every10 secs

- every 30 secs: randomly select another peer, starts sending chunks
  - "optimistically unchoke" this peer
  - newly chosen peer may join top 4

BitTorrent: tit-for-tat

(1) Alice "optimistically unchokes" Bob

(2) Alice becomes one of Bob's top-four providers; Bob reciprocates

(3) Bob becomes one of Alice's top-four providers
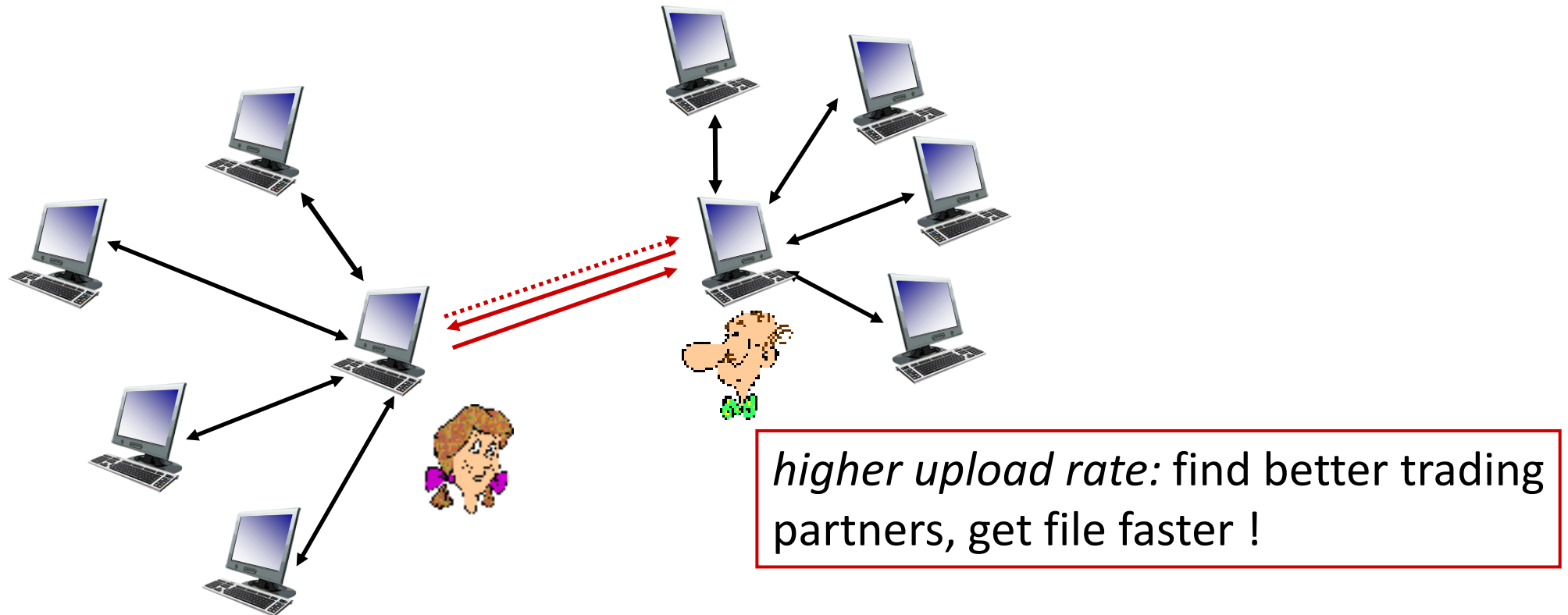
higher upload rate: find better trading partners, get file faster !

Application layer: overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS

- P2P applications
- **video streaming and content distribution networks**
- socket programming with UDP and TCP

Video Streaming and CDNs: context

- stream video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube, Amazon Prime: 80% of residential ISP traffic (2020)
- *challenge:* scale - how to reach ~1B users?
- *challenge:* heterogeneity
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- *solution:* distributed, application-level infrastructure

Multimedia: video

- video: sequence of images displayed at constant rate
  - e.g., 24 images/sec
- digital image: array of pixels
  - each pixel represented by bits
- coding: use redundancy *within* and *between* images to decrease # bits used to encode image
  - spatial (within image)
  - temporal (from one image to next)

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) and *number of repeated values (*N)
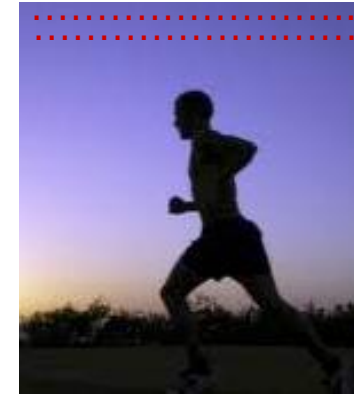
frame *i*

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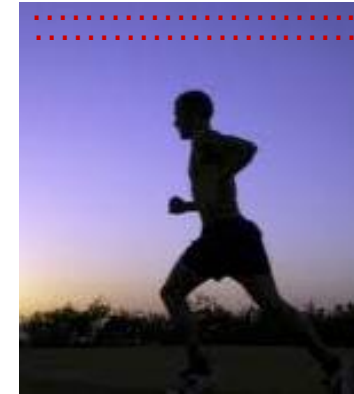
frame *i+1*

Multimedia: video

- CBR: (constant bit rate): video encoding rate fixed

- VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes

- examples:
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, 64Kbps – 12 Mbps)

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) and *number of repeated values* (*N*)

frame *i*

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

frame *i+1*