

Week 3: application layer

- principles of network applications
- FTP
- electronic mail
 - SMTP, POP3, IMAP
- P2P applications

1

Week3: application layer

our goals:

- ❖ conceptual, implementation aspects of network application protocols
 - transport-layer service models
 - client-server paradigm
 - peer-to-peer paradigm
- ❖ learn about protocols by examining popular application-level protocols
 - FTP
 - SMTP / POP3 / IMAP

2

Some network apps

- ❖ e-mail
- ❖ web
- ❖ text messaging
- ❖ remote login
- ❖ P2P file sharing
- ❖ multi-user network games
- ❖ streaming stored video (YouTube, Hulu, Netflix)
- ❖ voice over IP (e.g., Skype)
- ❖ real-time video conferencing
- ❖ social networking
- ❖ search
- ❖ ...
- ❖ ...

3

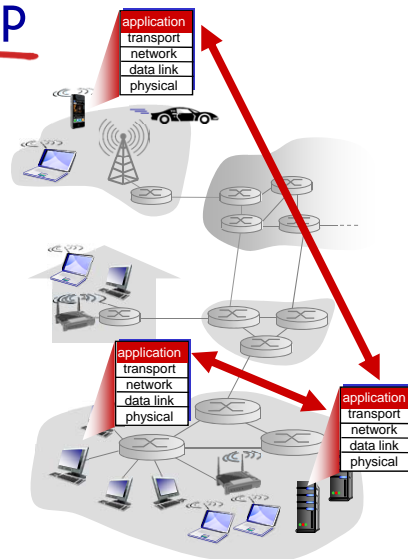
Creating a network app

write programs that:

- ❖ run on (different) *end systems*
- ❖ communicate over network
- ❖ e.g., web server software communicates with browser software

no need to write software for network-core devices

- ❖ network-core devices do not run user applications
- ❖ applications on end systems allows for rapid app development, propagation



4

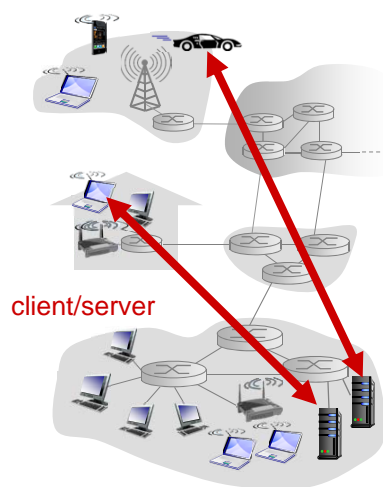
Application architectures

possible structure of applications:

- ❖ client-server
- ❖ peer-to-peer (P2P)

5

Client-server architecture



server:

- ❖ always-on host
- ❖ permanent IP address
- ❖ data centers for scaling

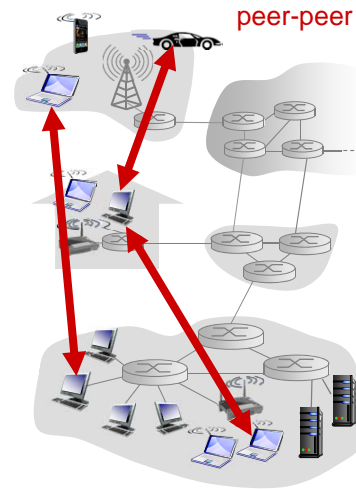
clients:

- ❖ communicate with server
- ❖ may be intermittently connected
- ❖ may have dynamic IP addresses
- ❖ do not communicate directly with each other

6

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, as well as new service demands
- ❖ peers are intermittently connected and change IP addresses
 - complex management



7

Processes communicating

process: program running within a host

- ❖ within same host, two processes communicate using *inter-process communication* (defined by OS)
- ❖ processes in different hosts communicate by exchanging *messages*

clients, servers

client process: process that initiates communication

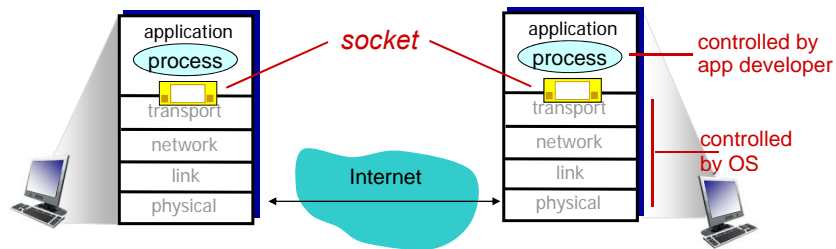
server process: process that waits to be contacted

- ❖ aside: applications with P2P architectures have client processes & server processes

8

Sockets

- ❖ process sends/receives messages to/from its **socket**
- ❖ socket analogous to door
 - sending process shoves message out door
 - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



9

Addressing processes

- ❖ to receive messages, process must have **identifier**
- ❖ host device has unique 32-bit IP address
- ❖ **Q:** does IP address of host on which process runs suffice for identifying the process?
 - **A:** no, many processes can be running on same host
- ❖ **identifier** includes both **IP address** and **port numbers** associated with process on host.
- ❖ example port numbers:
 - HTTP server: 80
 - mail server: 25
- ❖ to send HTTP message to `gaia.cs.umass.edu` web server:
 - **IP address:** 128.119.245.12
 - **port number:** 80
- ❖ more shortly...

10

App-layer protocol defines

- ❖ **types of messages exchanged,**
 - e.g., request, response
 - ❖ **message syntax:**
 - what fields in messages & how fields are delineated
 - ❖ **message semantics**
 - meaning of information in fields
 - ❖ **rules** for when and how processes send & respond to messages
- open protocols:**
 - ❖ defined in RFCs
 - ❖ allows for interoperability
 - ❖ e.g., HTTP, SMTP
 - proprietary protocols:**
 - ❖ e.g., Skype

11

What transport service does an app need?

data integrity

- ❖ some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- ❖ other apps (e.g., audio) can tolerate some loss

timing

- ❖ some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”

throughput

- ❖ some apps (e.g., multimedia) require minimum amount of throughput to be “effective”
- ❖ other apps (“elastic apps”) make use of whatever throughput they get

security

- ❖ encryption, data integrity, ...

12

Transport service requirements: common apps

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video: 10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
text messaging	no loss	elastic	yes and no

13

Internet transport protocols services

TCP service:

- ❖ *reliable transport* between sending and receiving process
- ❖ *flow control*: sender won't overwhelm receiver
- ❖ *congestion control*: throttle sender when network overloaded
- ❖ *does not provide*: timing, minimum throughput guarantee, security
- ❖ *connection-oriented*: setup required between client and server processes

UDP service:

- ❖ *unreliable data transfer* between sending and receiving process
- ❖ *does not provide*: reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup,

Q: why bother? Why is there a UDP?

14

Internet apps: application, transport protocols

<u>application</u>	<u>application layer protocol</u>	<u>underlying transport protocol</u>
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

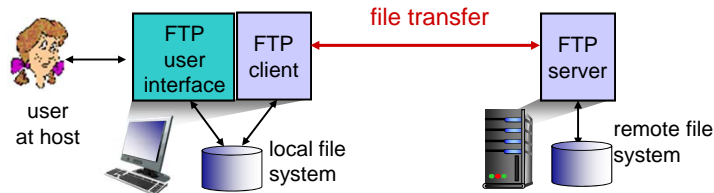
15

Week 3: roadmap

- principles of network applications
- **FTP**
- electronic mail
 - SMTP, POP3, IMAP
- P2P applications

16

FTP: the file transfer protocol

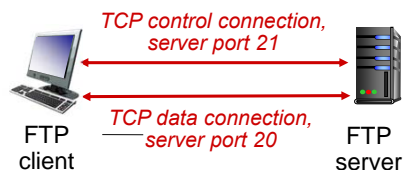


- ❖ transfer file to/from remote host
- ❖ client/server model
 - *client*: side that initiates transfer (either to/from remote)
 - *server*: remote host
- ❖ ftp: RFC 959
- ❖ ftp server: port 21

17

FTP: separate control, data connections

- ❖ FTP client contacts FTP server at port 21, using TCP
- ❖ client authorized over control connection
- ❖ client browses remote directory, sends commands over control connection
- ❖ when server receives file transfer command, *server* opens 2nd TCP data connection (for file) to client
- ❖ after transferring one file, server closes data connection
- ❖ server opens another TCP data connection to transfer another file
- ❖ control connection: *“out of band”*
- ❖ FTP server maintains “state”: current directory, earlier authentication



18

FTP commands, responses

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

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**

19

Week 3: roadmap

- principles of network applications
- FTP
- **electronic mail**
 - **SMTP, POP3, IMAP**
- P2P applications

20

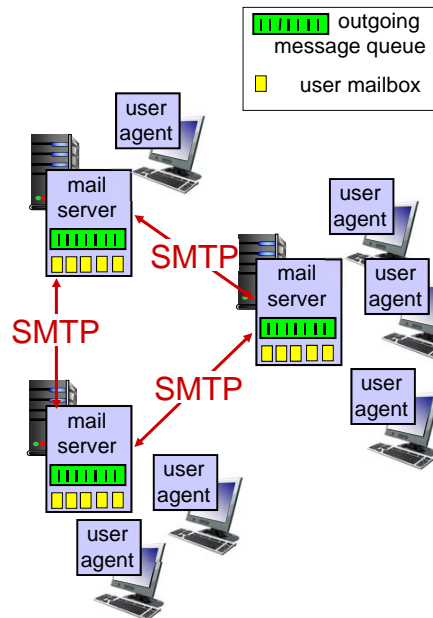
Electronic mail

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, Thunderbird, iPhone mail client
- ❖ outgoing, incoming messages stored on server

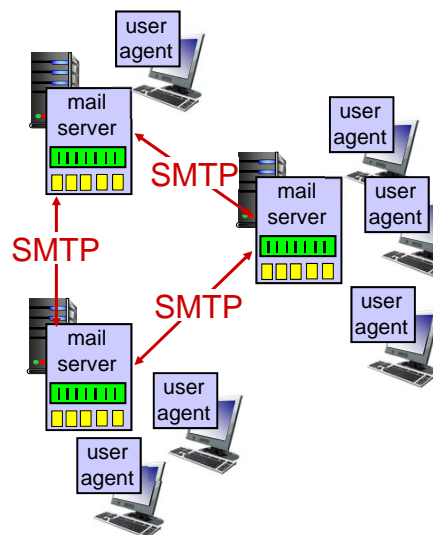


21

Electronic 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



22

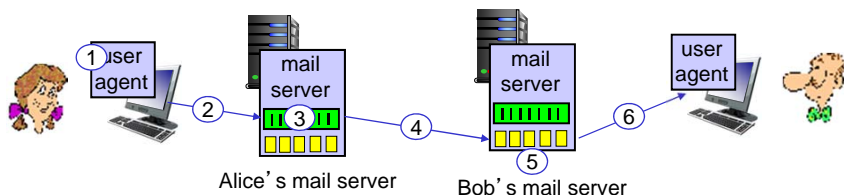
Electronic Mail: SMTP [RFC 2821]

- ❖ 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, FTP)
 - **commands:** ASCII text
 - **response:** status code and phrase
- ❖ messages must be in 7-bit ASCII

23

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



24

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
```

25

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)

26

SMTP: final words

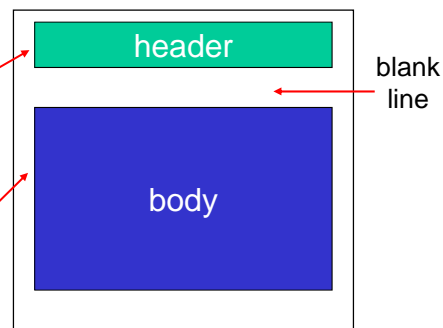
- ❖ SMTP uses persistent connections
- ❖ SMTP requires message (header & body) to be in 7-bit ASCII
- ❖ SMTP server uses CRLF . CRLF to determine end of message
- ❖ SMTP: push
- ❖ have ASCII command/response interaction, status codes
- ❖ SMTP: multiple objects sent in multipart msg

27

Mail message format

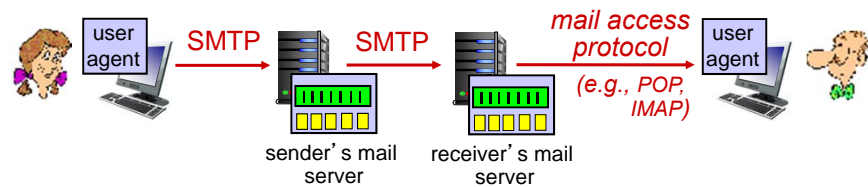
SMTP: protocol for exchanging email msgs
RFC 822: standard for text message format:

- ❖ header lines, e.g.,
 - To:
 - From:
 - Subject:
- different* from SMTP MAIL FROM, RCPT TO: commands!
- ❖ Body: the “message”
 - ASCII characters only



28

Mail access protocols



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

29

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
  
```

30

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”: 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

31

Week 3: application layer

- principles of network applications
- FTP
- electronic mail
 - SMTP, POP3, IMAP
- **P2P applications**

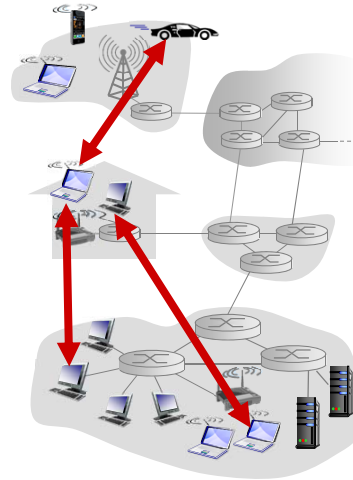
32

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)

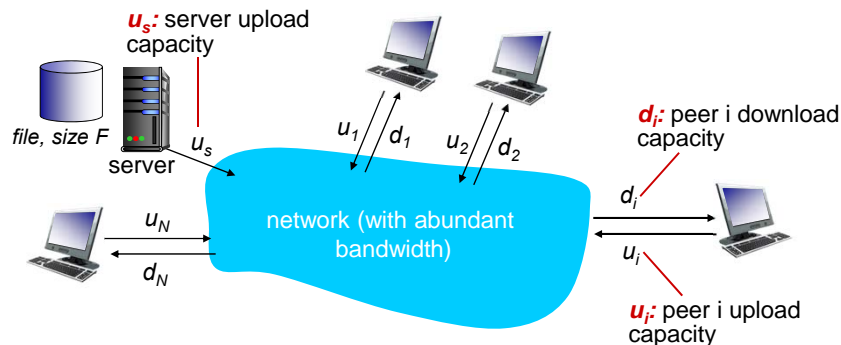


33

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



34

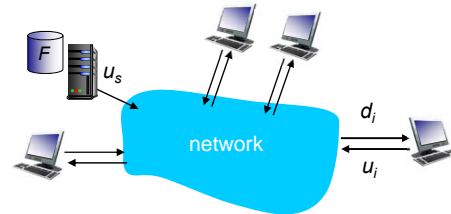
File distribution time: client-server

- ❖ **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}



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

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

increases linearly in N

35

File distribution time: P2P

- ❖ **server transmission:** must upload at least one copy

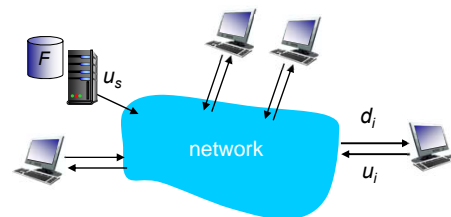
- time to send one copy: F/u_s

- ❖ **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 $u_s + \sum u_i$



time to distribute F
to N clients using
P2P approach

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

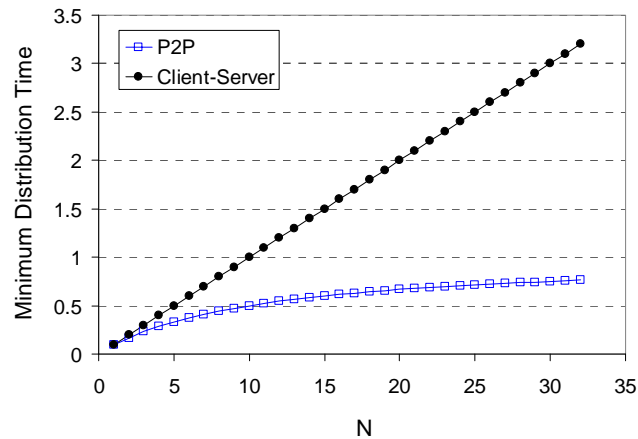
increases linearly in N ...

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

36

Client-server vs. P2P: example

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



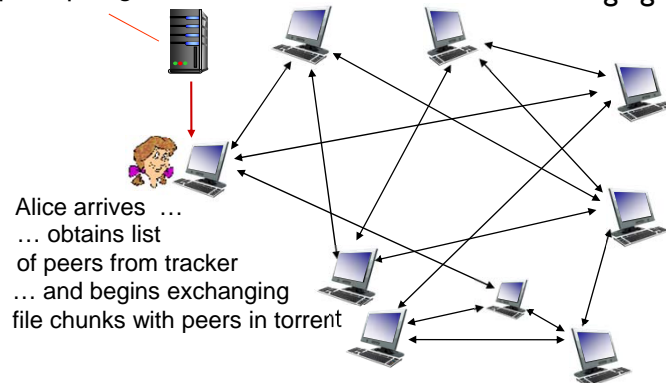
37

P2P file distribution: BitTorrent

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

tracker: tracks peers participating in torrent

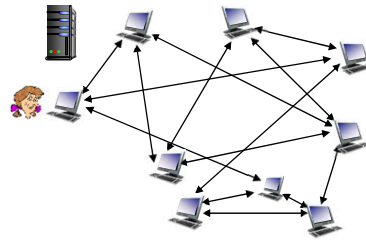
torrent: group of peers exchanging chunks of a file



38

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



39

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 every 10 secs
- ❖ every 30 secs: randomly select another peer, starts sending chunks
 - “optimistically unchoke” this peer
 - newly chosen peer may join top 4

40

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

