Application-level protocols

Distributed Software Systems

ACKNOWLEDGEMENT: This lecture is based on slides that were made available by the authors of *Computer Networking: A Top Down Approach Featuring the Internet* Jim Kurose, Keith Ross, 2nd edition, Addison Wesley, 2002

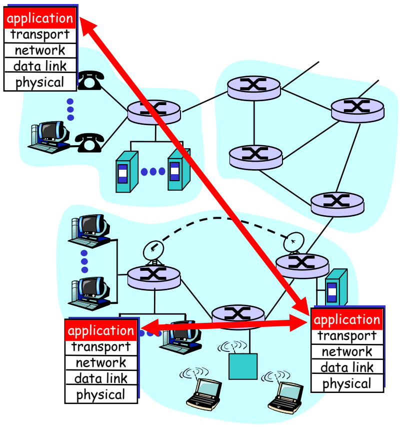
Applications and application-layer protocols

Application: communicating, distributed processes

- running in network hosts in "user space"
- exchange messages to implement app
- e.g., email, file transfer, the Web

Application-layer protocols

- one "piece" of an app
- define messages exchanged by apps and actions taken
- user services provided by lower layer protocols



Network applications: some jargon

- A process is a program that is running within a host.
- Within the same host, two processes communicate with interprocess communication defined by the OS.
- Processes running in different hosts communicate with an application-layer protocol

- A user agent is an interface between the user and the network application.
  - Web:browser
  - E-mail: mail reader
  - streaming audio/video: media player

Client-server paradigm

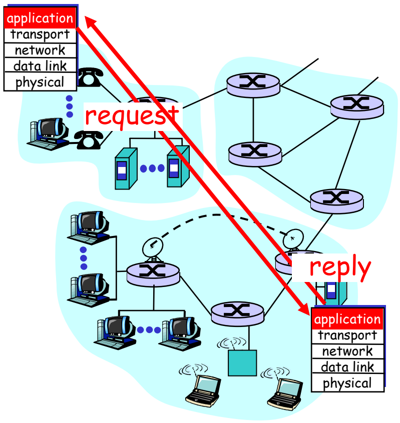
Typical network app has two pieces: *client* and *server*

Client:

- initiates contact with server ("speaks first")
- typically requests service from server,
- for Web, client is implemented in browser; for e-mail, in mail reader

Server:

- provides requested service to client
- e.g., Web server sends requested Web page, mail server delivers e-mail



Application-layer protocols (cont).

API: application programming interface

- defines interface between application and transport layer
- socket: Internet API
  - two processes communicate by sending data into socket, reading data out of socket

Q: how does a process "identify" the other process with which it wants to communicate?

- IP address of host running other process
- "port number" - allows receiving host to determine to which local process the message should be delivered

… lots more on this later.

What transport service does an app need?

Data loss

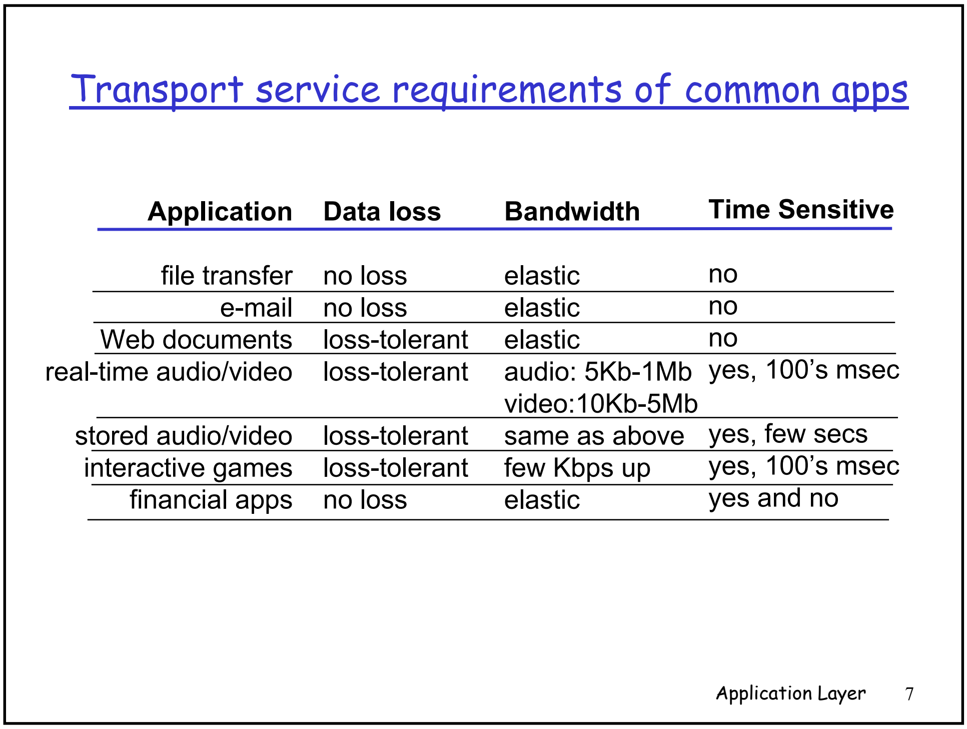
- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, telnet) require 100% reliable data transfer

Bandwidth

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

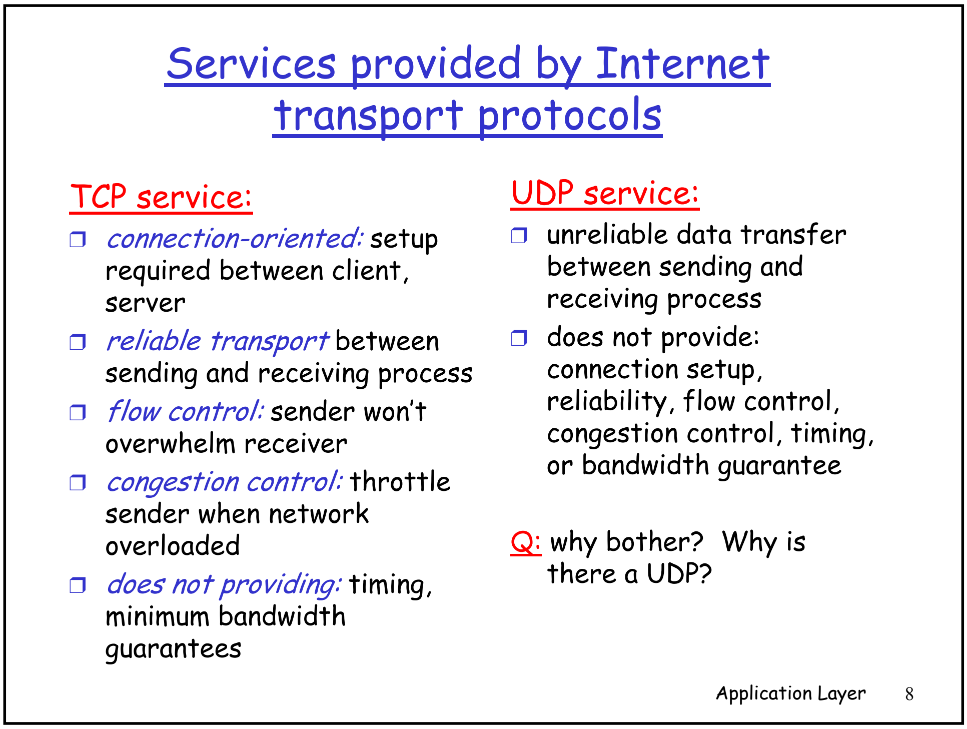
Timing

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



Transport service requirements of common apps

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	loss-tolerant	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kb-1Mb video:10Kb-5Mb	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
financial apps	no loss	elastic	yes and no



Services provided by Internet transport protocols

TCP service:

- *connection-oriented:* setup required between client, server
- *reliable transport* between sending and receiving process
- *flow control:* sender won’t overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not providing:* timing, minimum bandwidth guarantees

UDP service:

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

Q: why bother? Why is there a UDP?

Internet apps: their protocols and transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	smtp [RFC 821]	TCP
remote terminal access	telnet [RFC 854]	TCP
Web	http [RFC 2068]	TCP
file transfer	ftp [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
remote file server	NSF	TCP or UDP
Internet telephony	proprietary (e.g., Vocaltec)	typically UDP

The Web: some jargon

- ❒ Web page:
  - ❍ consists of "objects"
  - ❍ addressed by a URL
- ❒ Most Web pages consist of:
  - ❍ base HTML page, and
  - ❍ several referenced objects.
- ❒ URL has two components: host name and path name:

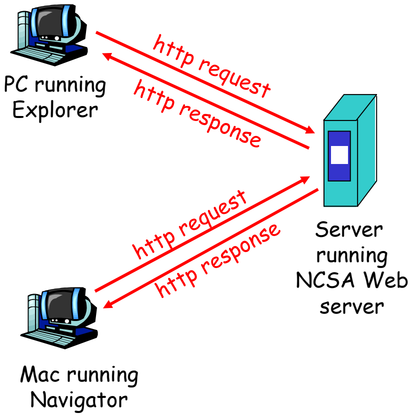
- ❒ User agent for Web is called a browser:
  - ❍ MS Internet Explorer
  - ❍ Netscape Communicator
- ❒ Server for Web is called Web server:
  - ❍ Apache (public domain)
  - ❍ MS Internet Information Server

`www.someSchool.edu/someDept/pic.gif`

The Web: the http protocol

http: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests
- http1.0: RFC 1945
- http1.1: RFC 2068



The http protocol: more

http: TCP transport service:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- http messages (application-layer protocol messages) exchanged between browser (http client) and Web server (http server)
- TCP connection closed

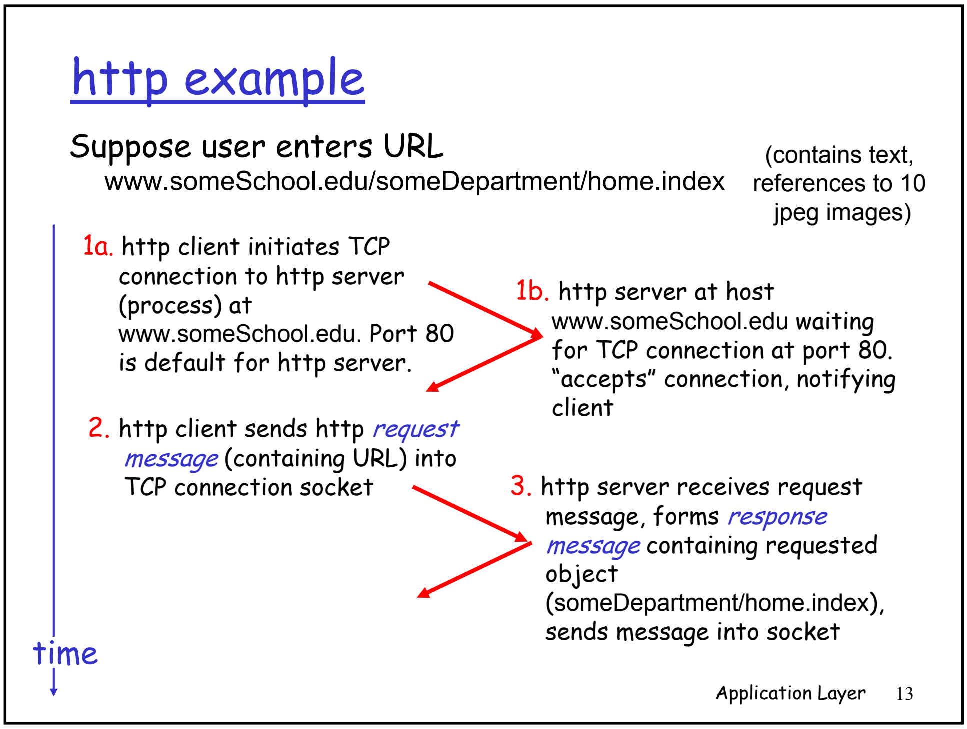
http is "stateless"

- server maintains no information about past client requests

aside

Protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled



http example

Suppose user enters URL

www.someSchool.edu/someDepartment/home.index

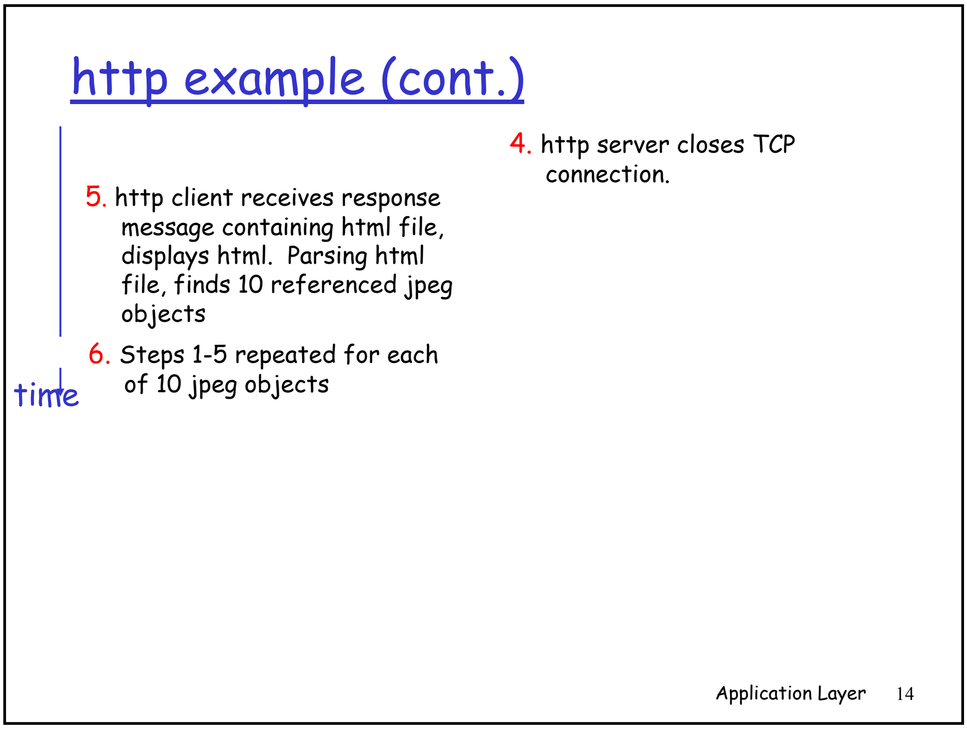
(contains text, references to 10 jpeg images)

1a. http client initiates TCP connection to http server (process) at www.someSchool.edu. Port 80 is default for http server.

1b. http server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. http client sends http *request message* (containing URL) into TCP connection socket

3. http server receives request message, forms *response message* containing requested object (someDepartment/home.index), sends message into socket



http example (cont.)

4. http server closes TCP connection.

5. http client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

Non-persistent and persistent connections

Non-persistent

- HTTP/1.0
- server parses request, responds, and closes TCP connection
- 2 RTTs to fetch each object
- Each object transfer suffers from slow start

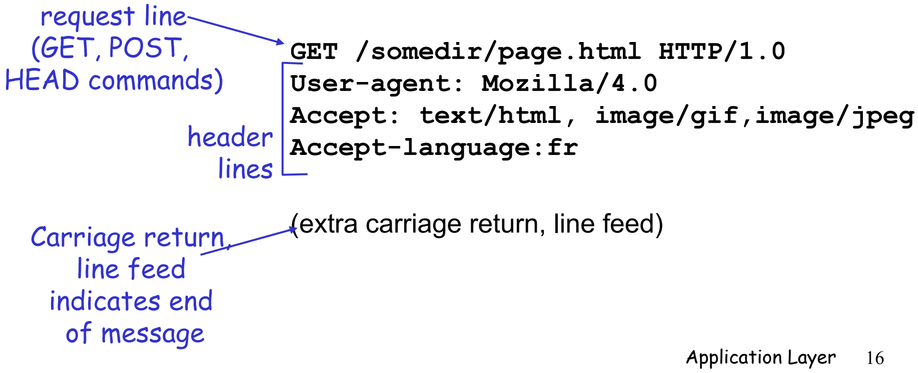
But most 1.0 browsers use parallel TCP connections.

Persistent

- default for HTTP/1.1
- on same TCP connection: server, parses request, responds, parses new request,..
- Client sends requests for all referenced objects as soon as it receives base HTML.
- Fewer RTTs and less slow start.

http message format: request

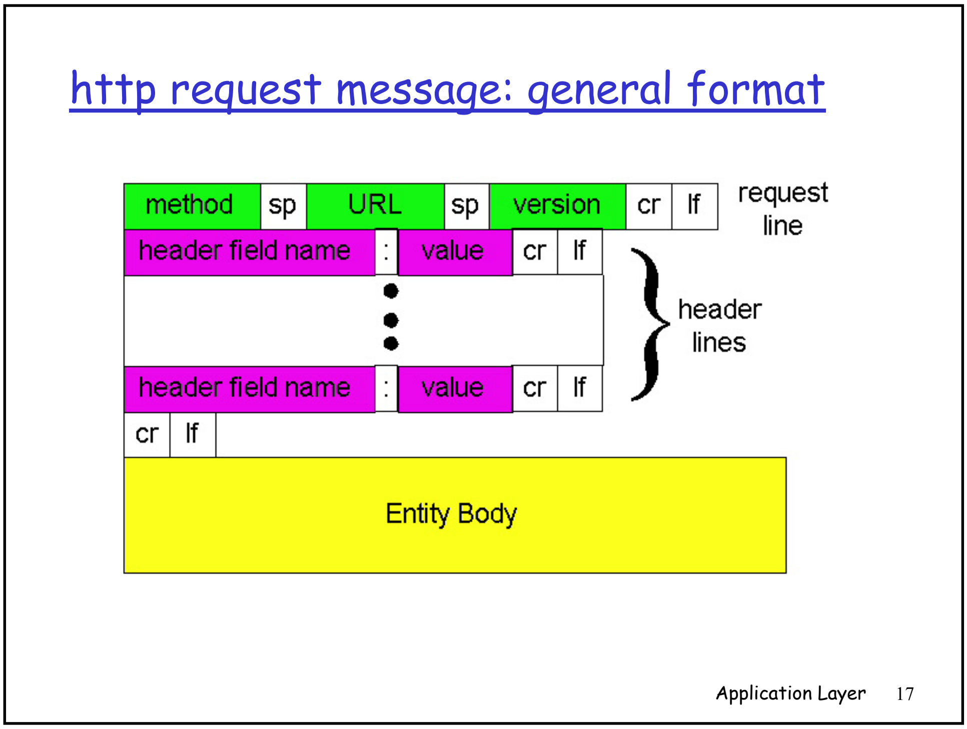
- two types of http messages: *request, response*
- http request message:
  - ASCII (human-readable format)



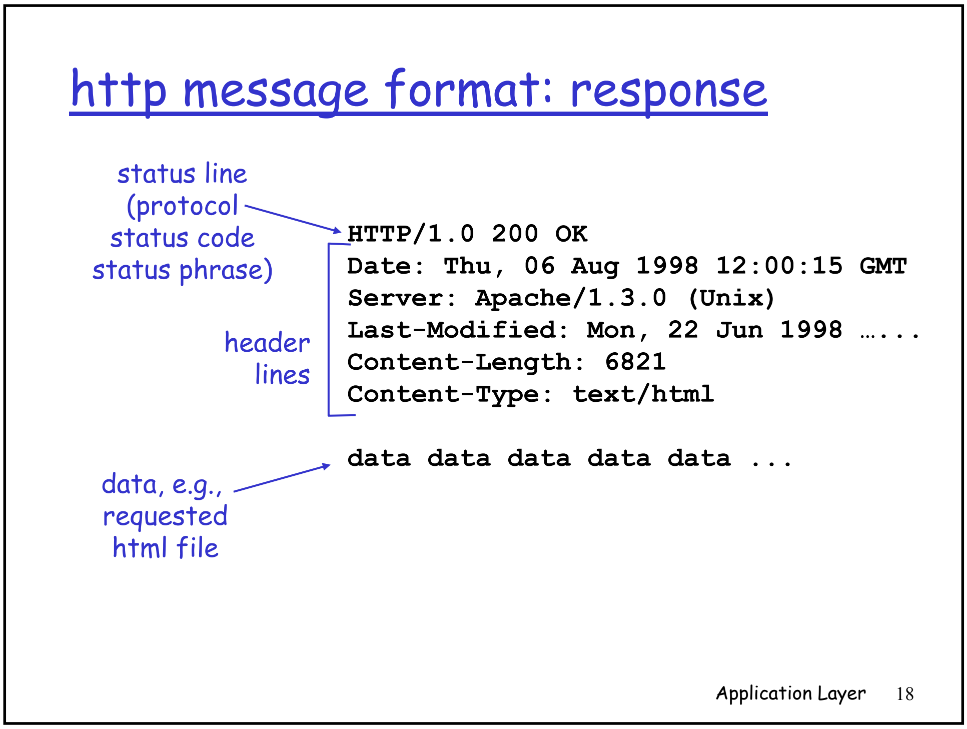
```
GET /somedir/page.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr

(extra carriage return, line feed)
```

request line (GET, POST, HEAD commands)
header lines
Carriage return, line feed indicates end of message



http request message: general format



http message format: response

status line (protocol status code status phrase)
header lines
data, e.g., requested html file

```
HTTP/1.0 200 OK
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

http response status codes

In first line in server->client response message.
A few sample codes:

`200 OK`

- request succeeded, requested object later in this message

`301 Moved Permanently`

- requested object moved, new location specified later in this message (Location:)

`400 Bad Request`

- request message not understood by server

`404 Not Found`

- requested document not found on this server

`505 HTTP Version Not Supported`

Trying out http (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet www.eurecom.fr 80
```

Opens TCP connection to port 80 (default http server port) at www.eurecom.fr. Anything typed in sent to port 80 at www.eurecom.fr

2. Type in a GET http request:

```
GET /~ross/index.html HTTP/1.0
```

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to http server

3. Look at response message sent by http server!

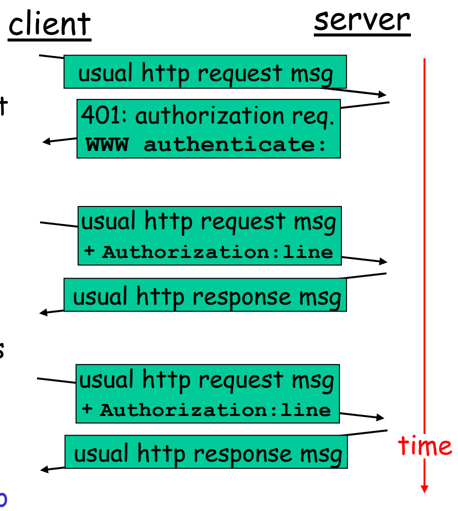
User-server interaction: authentication

Authentication goal: control access to server documents

- stateless: client must present authorization in each request
- authorization: typically name, password
  - `authorization:` header line in request
  - if no authorization presented, server refuses access, sends

    `WWW authenticate:`

    header line in response



client

server

usual http request msg

401: authorization req.
`WWW authenticate:`

usual http request msg
`+ Authorization:line`

usual http response msg

usual http request msg
`+ Authorization:line`

usual http response msg

time

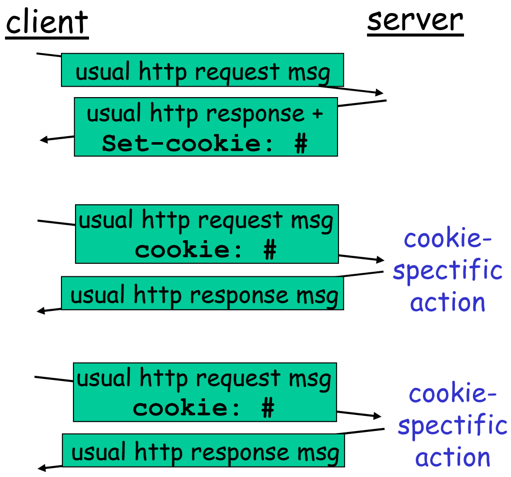
Browser caches name & password so that user does not have to repeatedly enter it.

User-server interaction: cookies

- server sends "cookie" to client in response mst

  `Set-cookie: 1678453`
- client presents cookie in later requests

  `cookie: 1678453`
- server matches presented-cookie with server-stored info
  - authentication
  - remembering user preferences, previous choices



client

server

usual http request msg

usual http response +
`Set-cookie: #`

usual http request msg
`cookie: #`

usual http response msg

cookie-spectific action

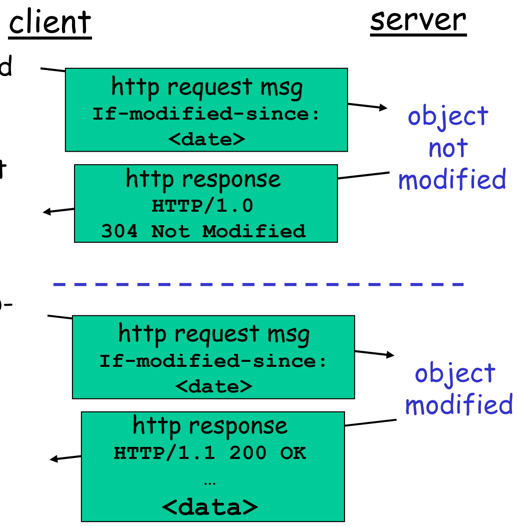
usual http request msg
`cookie: #`

usual http response msg

cookie-spectific action

User-server interaction: conditional GET

- Goal: don't send object if client has up-to-date stored (cached) version
- client: specify date of cached copy in http request
  `If-modified-since: <date>`
- server: response contains no object if cached copy up-to-date:
  `HTTP/1.0 304 Not Modified`



client

server

http request msg
`If-modified-since: <date>`

object not modified

http response
`HTTP/1.0 304 Not Modified`

http request msg
`If-modified-since: <date>`

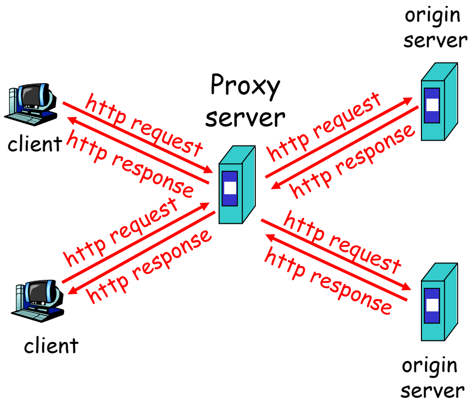
object modified

http response
`HTTP/1.1 200 OK`
…
`<data>`

Web Caches (proxy server)

Goal: satisfy client request without involving origin server

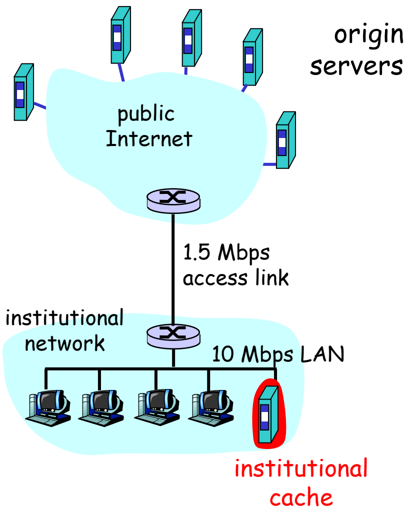
- user sets browser: Web accesses via web cache
- client sends all http requests to web cache
  - if object at web cache, web cache immediately returns object in http response
  - else requests object from origin server, then returns http response to client



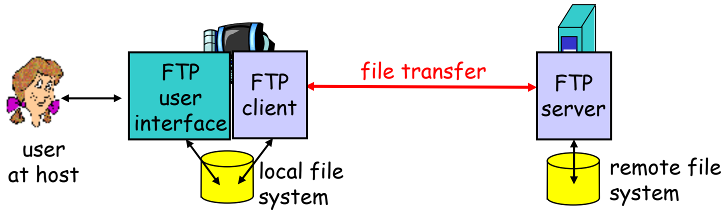
Why Web Caching?

Assume: cache is "close" to client (e.g., in same network)

- smaller response time: cache "closer" to client
- decrease traffic to distant servers
  - link out of institutional/local ISP network often bottleneck



ftp: the file transfer protocol

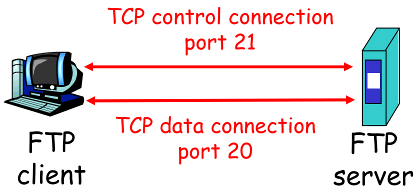


- transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ftp: RFC 959
- ftp server: port 21

ftp: separate control, data connections

- ftp client contacts ftp server at port 21, specifying TCP as transport protocol
- two parallel TCP connections opened:
  - control: exchange commands, responses between client, server.

    "out of band control"
  - data: file data to/from server
- ftp server maintains "state": current directory, earlier authentication



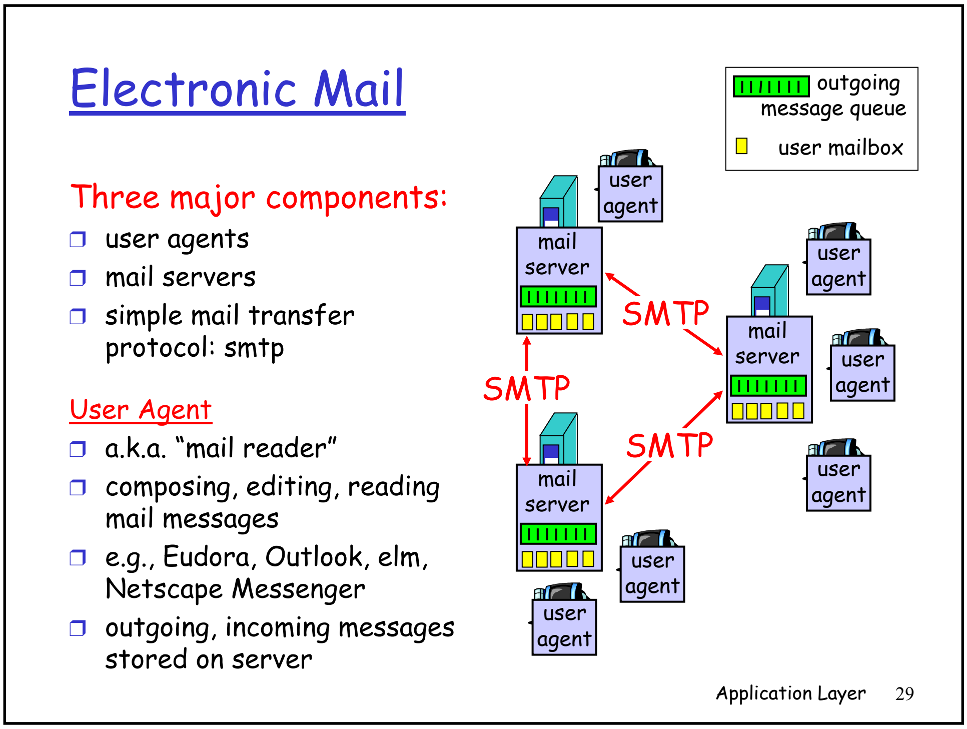
ftp commands, responses

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**



Electronic Mail

Three major components:

- ❐ user agents
- ❐ mail servers
- ❐ simple mail transfer protocol: smtp

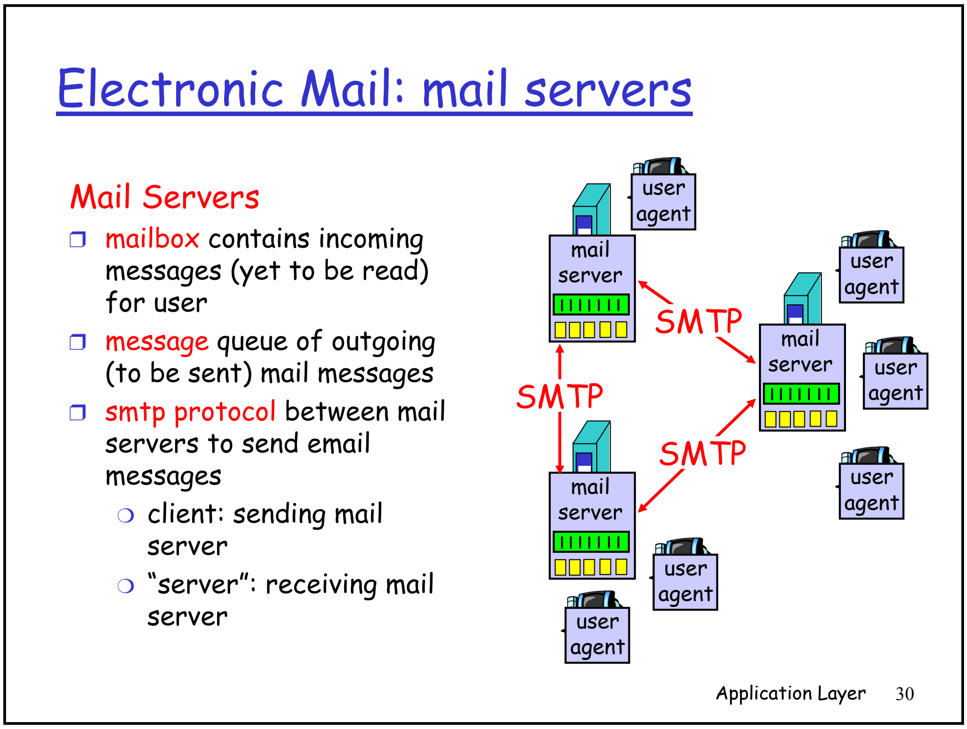
User Agent

- ❐ a.k.a. "mail reader"
- ❐ composing, editing, reading mail messages
- ❐ e.g., Eudora, Outlook, elm, Netscape Messenger
- ❐ outgoing, incoming messages stored on server

🟩 outgoing message queue

🟨 user mailbox

user agent
mail server
SMTP
user agent
mail server
user agent
SMTP
SMTP
user agent
mail server
user agent
user agent

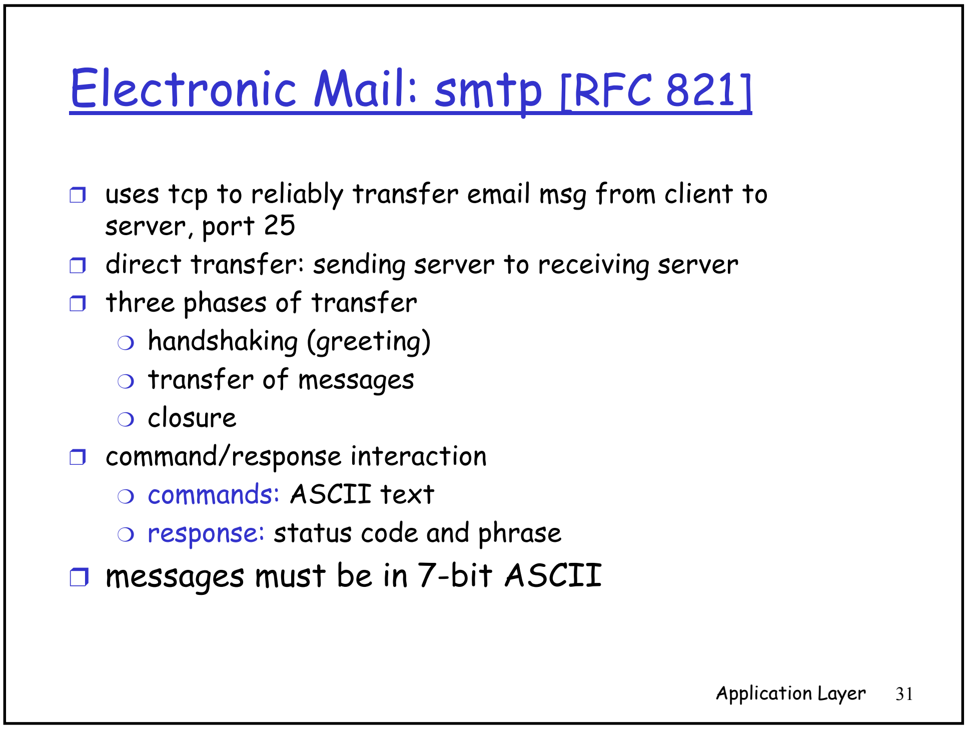


Electronic Mail: mail servers

Mail Servers

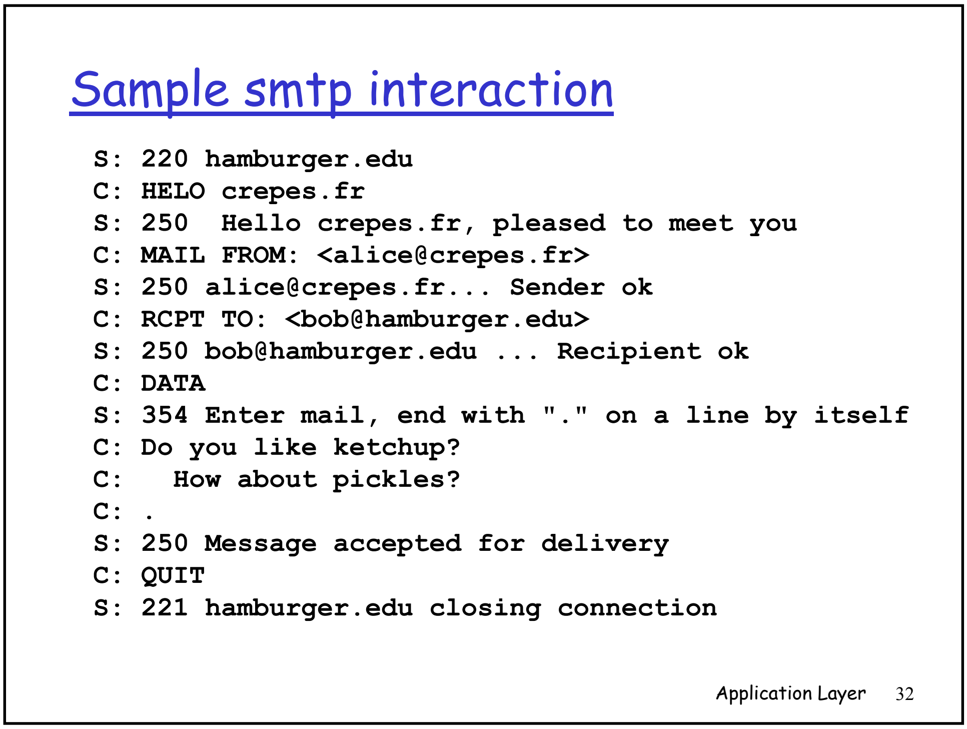
- ❐ mailbox contains incoming messages (yet to be read) for user
- ❐ message queue of outgoing (to be sent) mail messages
- ❐ smtp protocol between mail servers to send email messages
  - ❍ client: sending mail server
  - ❍ "server": receiving mail server

user agent
mail server
SMTP
user agent
mail server
user agent
SMTP
SMTP
user agent
mail server
user agent
user agent



Electronic Mail: smtp [RFC 821]

- uses tcp to reliably transfer email msg 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
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCII



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
```

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
- smtp requires that message (header & body) be in 7-bit ascii
- certain character strings are not permitted in message (e.g., `CRLF.CRLF`). Thus message has to be encoded (usually into either base-64 or quoted printable)
- smtp server uses `CRLF.CRLF` to determine end of message

Comparison with http

- http: pull
- email: push
- both have ASCII command/response interaction, status codes
- http: each object is encapsulated in its own response message
- smtp: multiple objects message sent in a multipart message

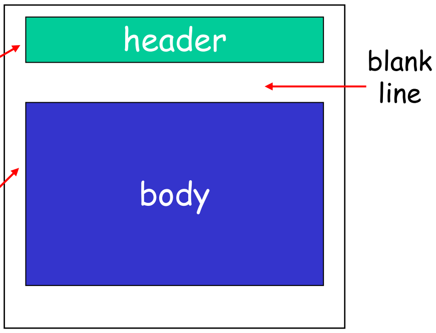
Mail message format

smtp: protocol for exchanging email msgs

RFC 822: standard for text message format:

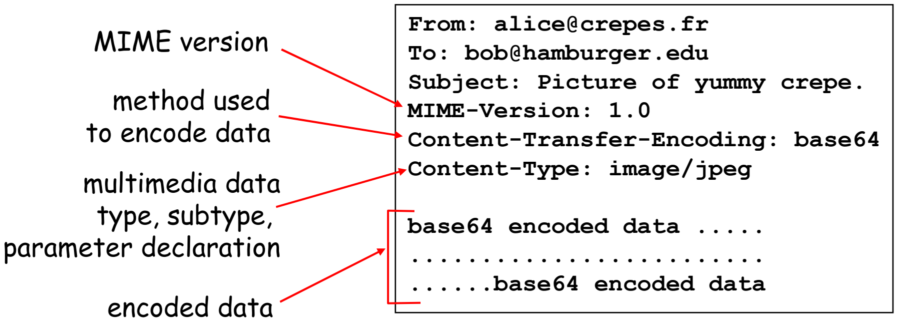
- header lines, e.g.,
  - To:
  - From:
  - Subject:

  different from smtp commands!
- body
  - the "message", ASCII characters only



Message format: multimedia extensions

- MIME: multimedia mail extension, RFC 2045, 2056
- additional lines in msg header declare MIME content type


MIME version
method used to encode data
multimedia data type, subtype, parameter declaration
encoded data

```
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
```

MIME types

`Content-Type: type/subtype; parameters`

Text

- example subtypes: **`plain`**, **`html`**

Image

- example subtypes: **`jpeg`**, **`gif`**

Audio

- exampe subtypes: **`basic`** (8-bit mu-law encoded), **`32kadpcm`** **(32 kbps coding)**

Video

- example subtypes: **`mpeg`**, **`quicktime`**

Application

- other data that must be processed by reader before "viewable"
- example subtypes: **`msword`**, **`octet-stream`**

Multipart Type

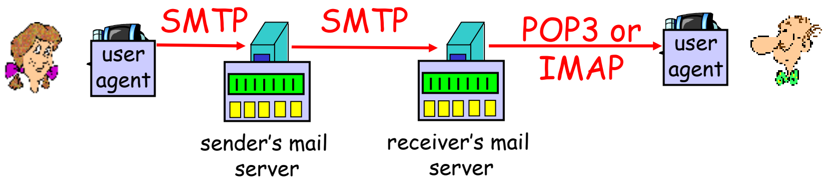
```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789

--98766789
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain

Dear Bob,
Please find a picture of a crepe.
--98766789
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
--98766789--
```

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: Hotmail , Yahoo! Mail, etc.

POP3 protocol

authorization phase

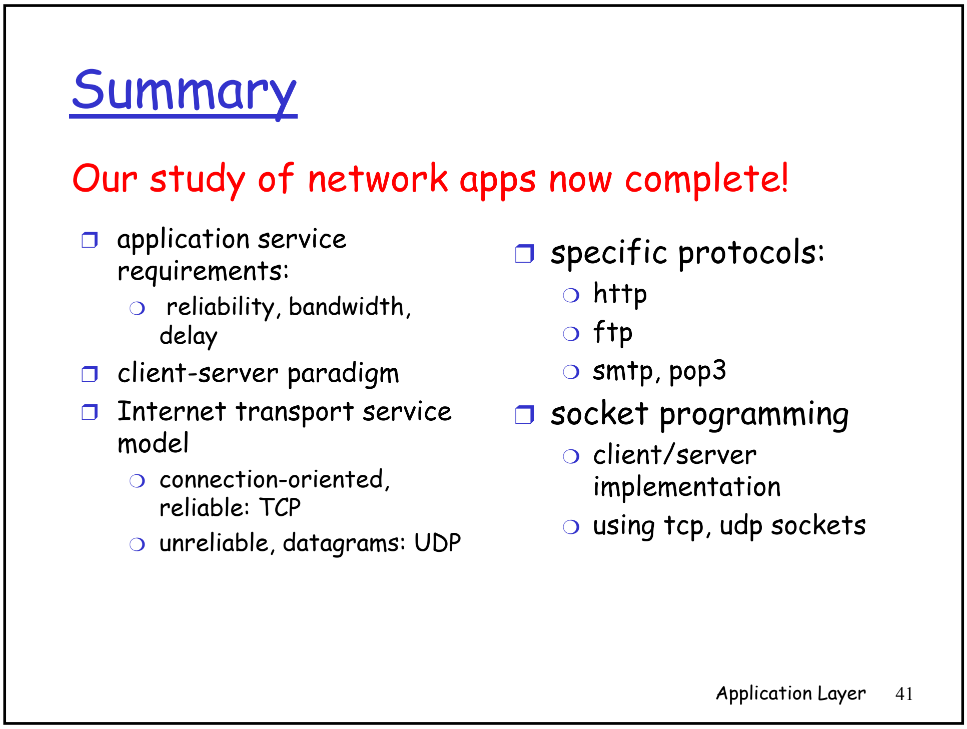
- client commands:
  - **user**: declare username
  - **pass**: password
- server responses
  - **+OK**
  - **-ERR**

transaction phase, client:

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

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

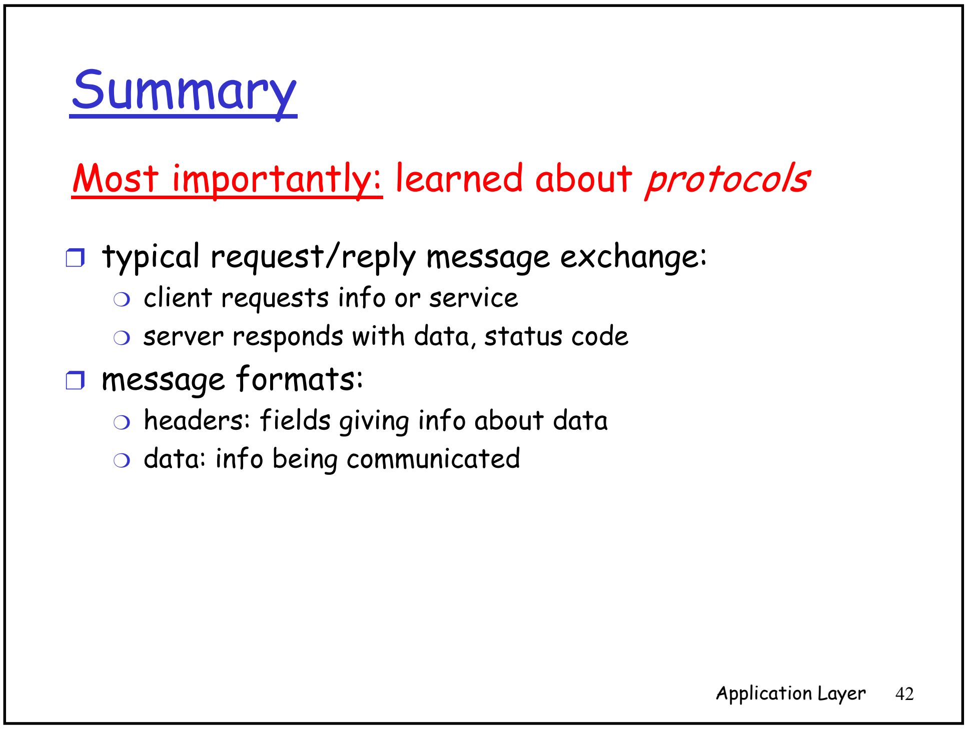
```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```



Summary

Our study of network apps now complete!

- application service requirements:
  - reliability, bandwidth, delay
- client-server paradigm
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP
- specific protocols:
  - http
  - ftp
  - smtp, pop3
- socket programming
  - client/server implementation
  - using tcp, udp sockets



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 being communicated

Client-Server Applications

- The application-layer protocols we have looked at illustrate the choices that arise in the design and implementation of a client-server application
  - choice of transport protocol
  - stateful vs stateless servers
  - in-band vs out-of-band control messages
- Another important design choice is whether the client and especially the server is concurrent or not
- We review these choices in the following slides

Issues in Client design

- Must know or find out the location of the server
- Which protocol to use: reliable or unreliable?
- Blocking (synchronous) request or non-blocking (asynchronous)

Issues in Server Design

- Connection-oriented or connection-less servers
  - TCP or UDP?
- Concurrent or iterative servers: handle multiple requests concurrently or one after the other?
- Stateful or stateless servers
- Multi-protocol, multi-service servers

Connection-less vs connection-oriented servers

- protocol used determines level of reliability
- TCP provides reliable-data delivery
  - verifies that data arrives at other end, retransmits segments that don't
  - checks that data is not corrupted along the way
  - makes sure data arrives in order
  - eliminates duplicate packets
  - provides flow control to make sure sender does not send data faster than receiver can consume it
  - informs both client and server if underlying network becomes inoperable

Connection-less servers

- UDP unreliable – best effort delivery
- UDP relies on application to take whatever actions are necessary for reliability
- UDP used if
  - application protocol designed to handle reliability and delivery errors in an application-specific manner, e.g. audio and video on the internet
  - overhead of TCP connections too much for application
  - multicast

Stateful vs stateless servers

- State ≡ Information that server maintains about the status of ongoing interactions with clients
- Stateful servers
  - state information can help server in performing request faster
  - state information needs to be preserved across (or reconstructed after) crashes
- Stateless servers
  - quicker and more reliable recovery after crashes
  - smaller memory requirements
- Stateless servers: application protocol should have ***idempotent*** operations

Concurrency in servers

- Concurrency needed if several clients and service is expensive
- Operating system support
  - Multiple processes
  - Threads
  - Asynchronous I/O, e.g. using select() system call
- Process/thread pre-allocation for improving performance
- Delayed process/thread allocation