

Chapter 2: Application Layer

Chapter goals:

- r conceptual + implementation aspects of network application protocols
 - m client server paradigm
 - m service models
- r learn about protocols by examining popular application-level protocols

More chapter goals

- r specific protocols:
 - m http
 - m ftp
 - m smtp
 - m pop
 - m dns
- r programming network applications
 - m socket programming

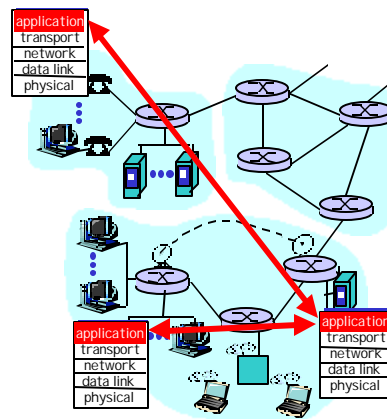
Applications and application-layer protocols

Application: communicating, distributed processes

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

Application-layer protocols

- m one (big) "piece" of a network application
- m define messages exchanged by apps and actions taken
- m user services provided by lower layer protocols
- m e.g., HTTP, SMTP



Network applications: some jargon

- r A **process** is a program that is running within a host.
- r Within the same host, two processes communicate with **interprocess communication** defined by the OS.
- r Processes running in different hosts communicate with an **application-layer protocol**
- r A **user agent** is an interface between the user and the network application.
 - m Web: browser
 - m E-mail: mail reader
 - m streaming audio/video: media player

2: Application Layer 3

Client-server paradigm

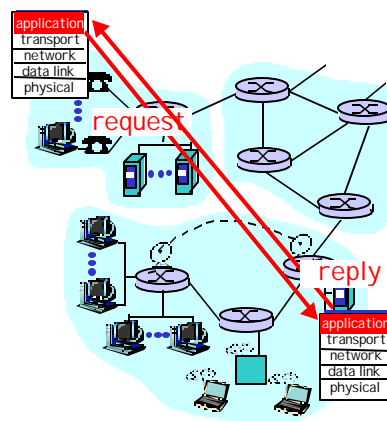
Typical network app has two pieces: *client* and *server*

Client:

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

Server:

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



2: Application Layer 4

Application-layer protocols (cont).

API: application programming interface

- r defines interface between application and transport layer
- r socket: Internet API
 - m 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?

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

... lots more on this later.

2: Application Layer 5

What transport service does an app need?

Data loss

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

Bandwidth

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

Timing

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

2: Application Layer 6

Transport service requirements of common apps

<u>Application</u>	<u>Data loss</u>	<u>Bandwidth</u>	<u>Time Sensitive</u>
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

2: Application Layer 7

Services provided by Internet transport protocols

TCP service:

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

UDP service:

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

Q: why bother? Why is there a UDP?

2: Application Layer 8

Internet apps: their protocols and transport protocols

<u>Application</u>	<u>Application layer protocol</u>	<u>Underlying transport protocol</u>
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	NFS	TCP or UDP
Internet telephony	proprietary (e.g., Vocaltec)	typically UDP

2: Application Layer 9

The Web: some jargon

- r Web page:
 - m consists of "objects"
 - m addressed by a URL
- r Most Web pages consist of:
 - m base HTML file, and
 - m several referenced objects.
- r URL (Uniform Resource Locator) has three parts:
 - protocol, host name (w/port), and path name:
- r User agent for Web is called a browser:
 - m MS Internet Explorer
 - m Netscape Communicator
- r Server for Web is called Web server:
 - m Apache (public domain)
 - m MS Internet Information Server

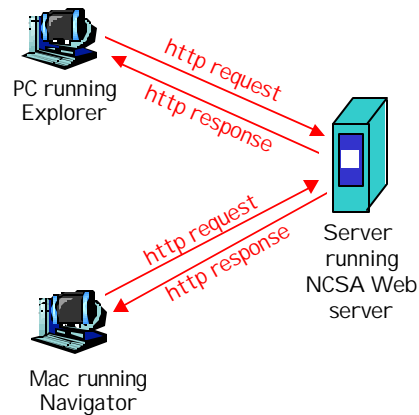
<http://www.someSchool.edu:port/someDept/pic.gif>

2: Application Layer 10

The Web: the http protocol

http: hypertext transfer protocol

- r Web's application layer protocol
- r client/server model
 - m *client*: browser that requests, receives, "displays" Web objects
 - m *server*: Web server sends objects in response to requests
- r http1.0: RFC 1945, May 1996
- r http1.1: RFC 2068, Jan. 1997



2: Application Layer 11

The http protocol: more

http: TCP transport service:

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

http is "stateless"

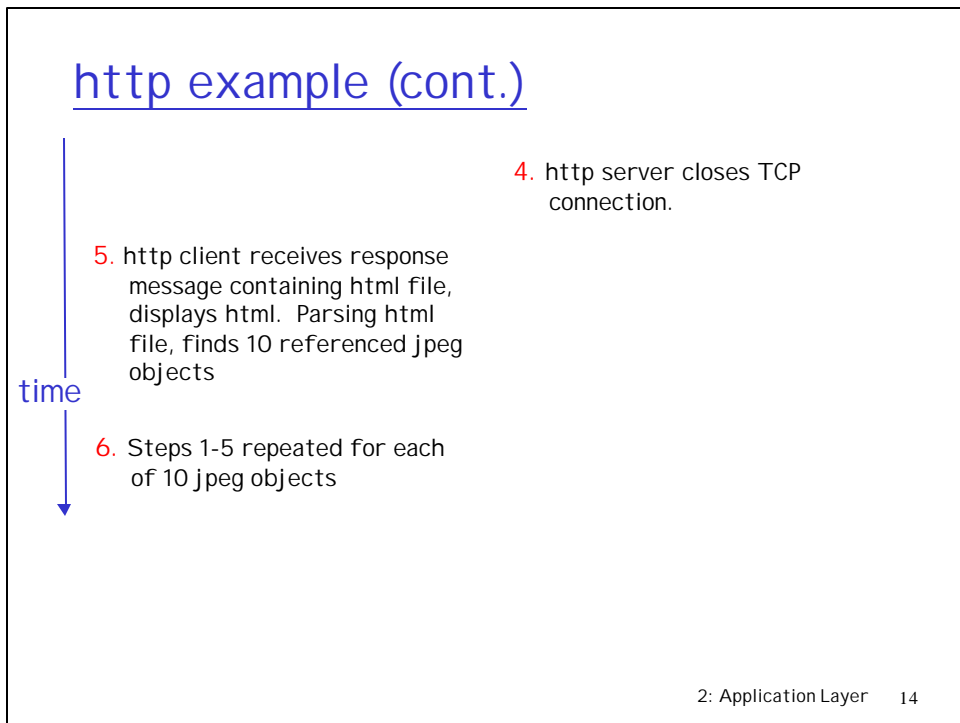
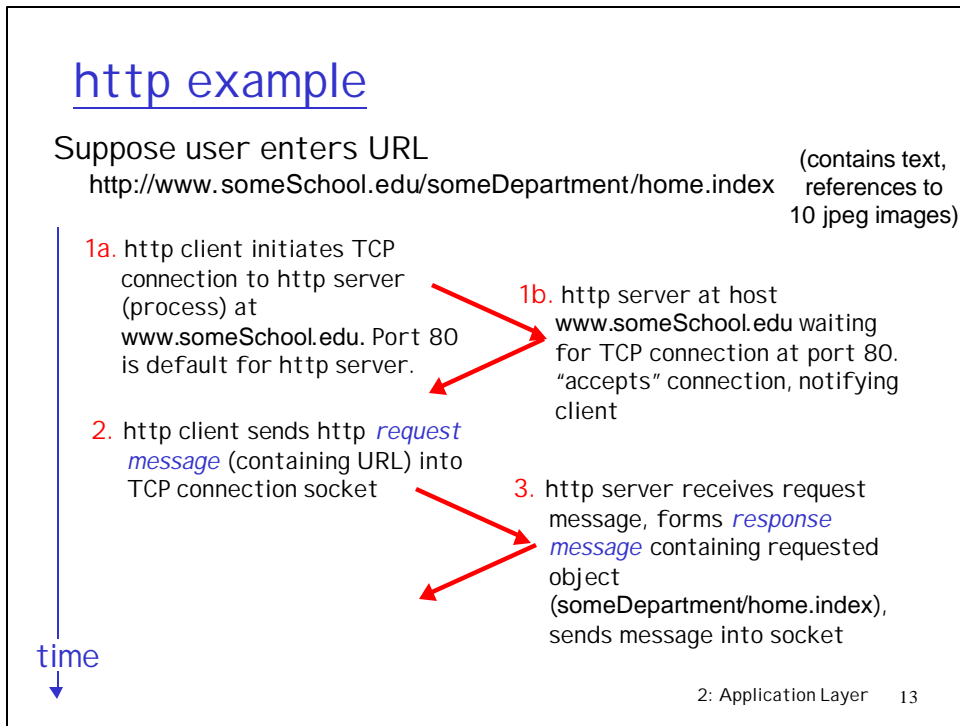
- r server maintains no information about past client requests

—aside

Protocols that maintain "state" are complex!

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

2: Application Layer 12



Non-persistent and persistent connections

Non-persistent

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

But most 1.0 browsers use parallel TCP connections.

Persistent

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

2: Application Layer 15

http message format: request

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

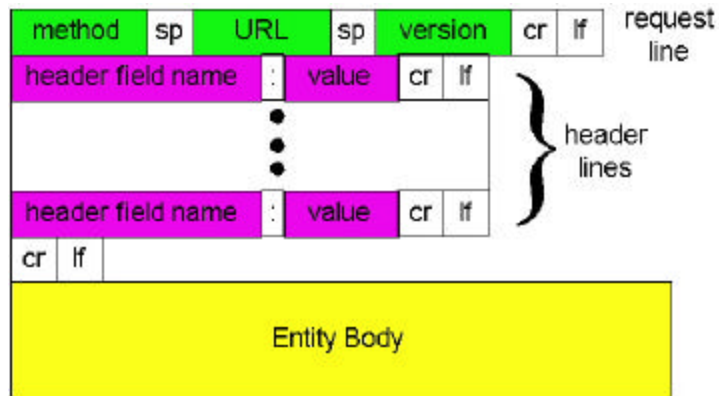
request line (GET, POST, HEAD commands) → GET /somedir/page.html HTTP/1.0

header lines → User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: fr

Carriage return, line feed → (extra carriage return, line feed)
indicates end of message

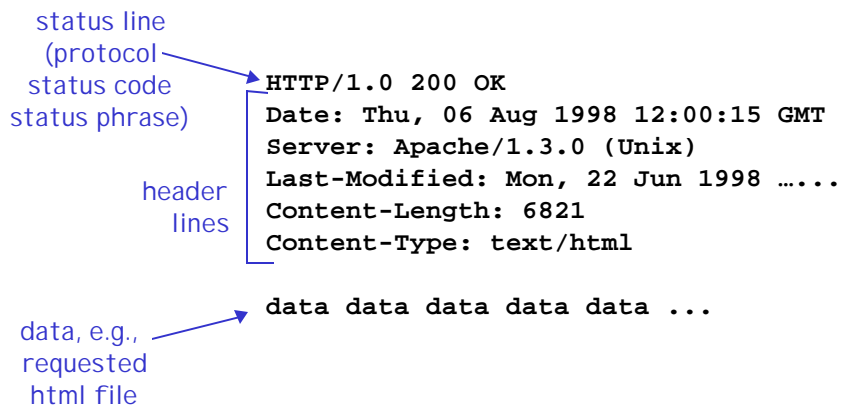
2: Application Layer 16

http request message: general format



2: Application Layer 17

http message format: response



2: Application Layer 18

http response status codes

In first line in server->client response message.

A few sample codes:

200 OK

m request succeeded, requested object later in this message

301 Moved Permanently

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

400 Bad Request

m request message not understood by server

404 Not Found

m requested document not found on this server

505 HTTP Version Not Supported

2: Application Layer 19

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

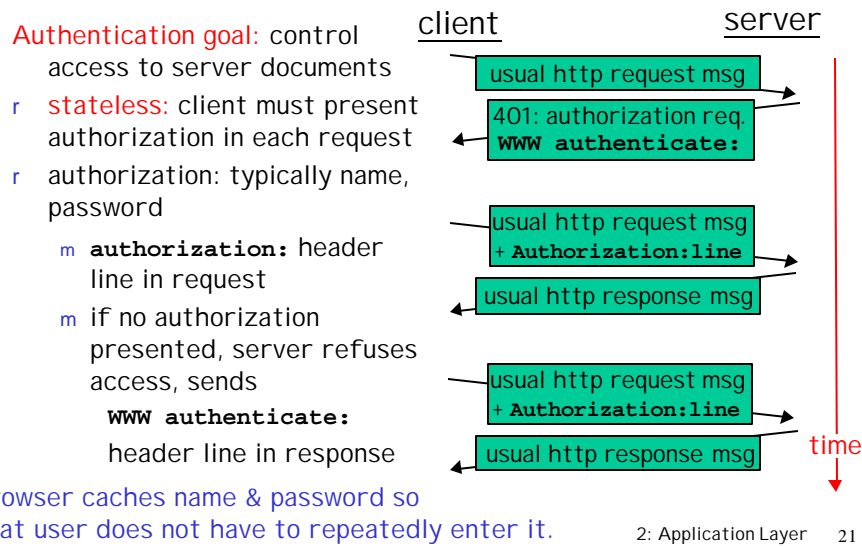
2. Type in a GET http request:

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

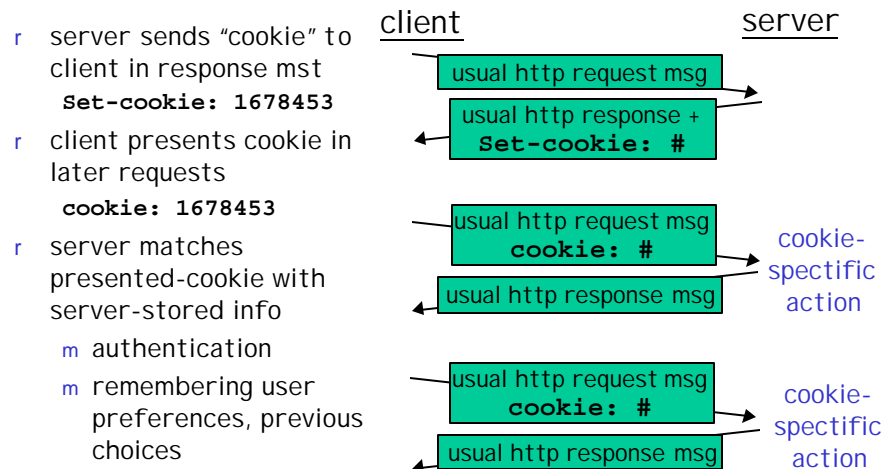
3. Look at response message sent by http server!

2: Application Layer 20

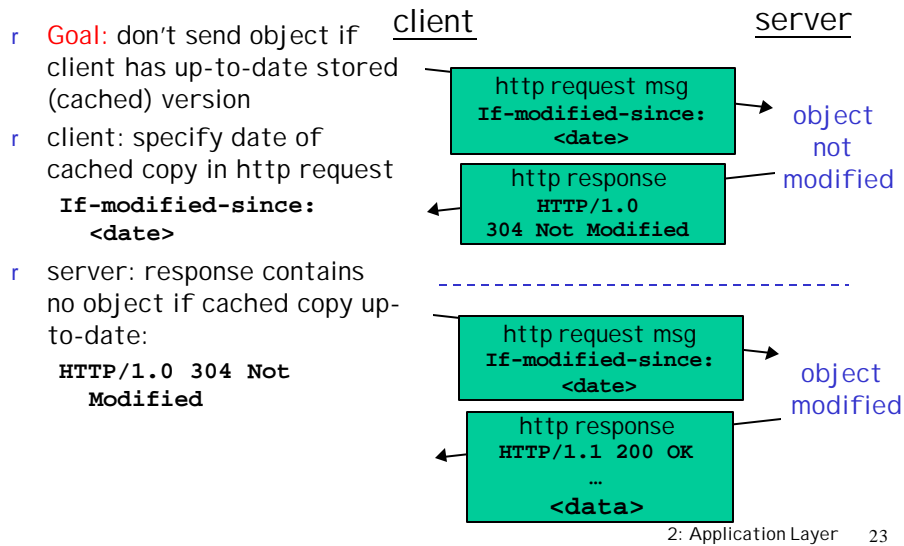
User-server interaction: authentication



User-server interaction: cookies



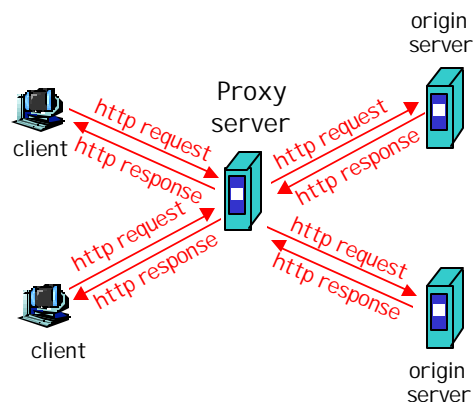
User-server interaction: conditional GET



Web Caches (proxy server)

Goal: satisfy client request without involving origin server

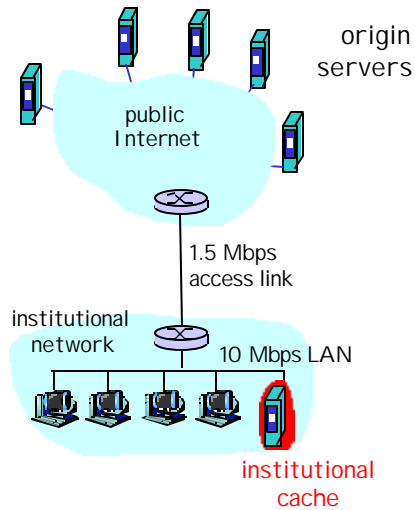
- r user sets browser: Web accesses via web cache
- r client sends all http requests to web cache
 - m if object at web cache, web cache immediately returns object in http response
 - m else requests object from origin server, then returns http response to client



Why Web Caching?

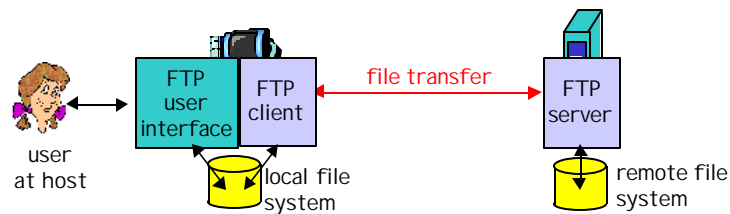
Assume: cache is "close" to client (e.g., in same network)

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



2: Application Layer 25

ftp: the file transfer protocol

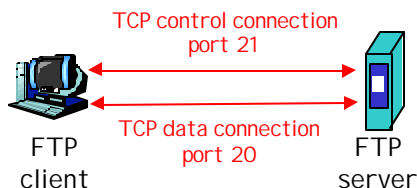


- r transfer file to/from remote host
- r client/server model
 - m *client*: side that initiates transfer (either to/from remote)
 - m *server*: remote host
- r ftp: RFC 959
- r ftp server: port 21

2: Application Layer 26

ftp: separate control, data connections

- r ftp client contacts ftp server at port 21, specifying TCP as transport protocol
- r two parallel TCP connections opened:
 - m **control**: exchange commands, responses between client, server.
"out of band control"
 - m **data**: file data to/from server
- r ftp server maintains "state": current directory, earlier authentication



2: Application Layer 27

ftp commands, responses

Sample commands:

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

Sample return codes

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

2: Application Layer 28

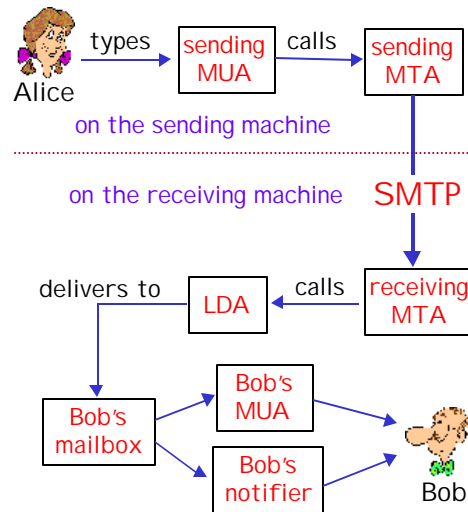
Electronic Mail

Three major components:

- r mail user agents (MUA)
- r mail servers or mail transfer agents (MTA)
- r simple mail transfer protocol: SMTP / ESMTP

User Agent

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

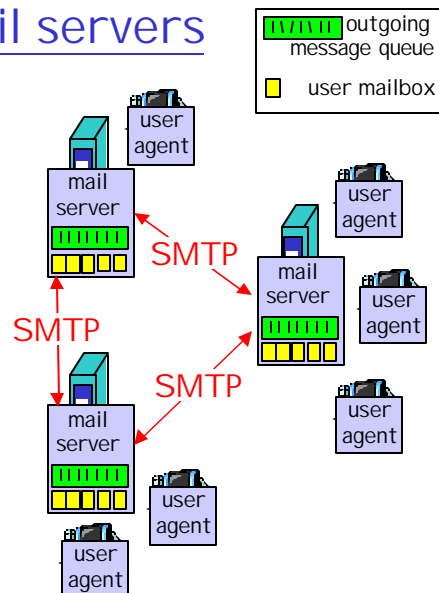


2: Application Layer 29

Electronic Mail: mail servers

Mail Servers

- r **mailbox** contains incoming messages (yet to be read) for user
- r **message** queue of outgoing (to be sent) mail messages
- r **smtp protocol** used between mail servers to send email messages (routing messages)
 - m client: sending mail server
 - m "server": receiving mail server
- r Example: sendmail, smail, qmail, etc.



2: Application Layer 30

Electronic Mail: smtp [RFC 821]

- r uses tcp to reliably transfer email msg from client to server, port 25
- r direct transfer: sending server to receiving server
- r three phases of transfer
 - m handshaking (greeting)
 - m transfer of messages
 - m closure
- r command/response interaction
 - m **commands**: ASCII text
 - m **response**: status code and phrase
- r messages must be in 7-bit ASCII
- r ESMTP [RFC 1869] - SMTP Service Extension: 8-bit data transfer

2: Application Layer 31

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

```

2: Application Layer 32

try smtp interaction for yourself:

- r **telnet servername 25**
 - r see 220 reply from server
 - r enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands
- above lets you send email without using email client (reader)

2: Application Layer 33

smtp: final words

- r smtp uses persistent connections
- r smtp requires that message (header & body) be in 7-bit ascii
- r 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)
- r smtp server uses CRLF.CRLF to determine end of message
- r esmtp can take 8-bit data

Comparison with http

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

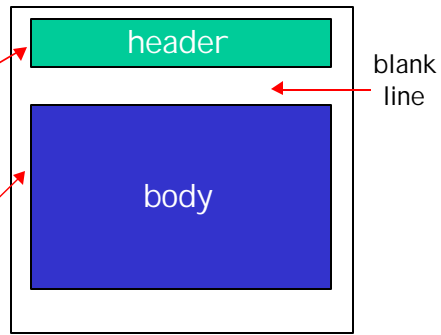
2: Application Layer 34

Mail message format

smtp: protocol for exchanging email msgs

RFC 822: standard for text message format:

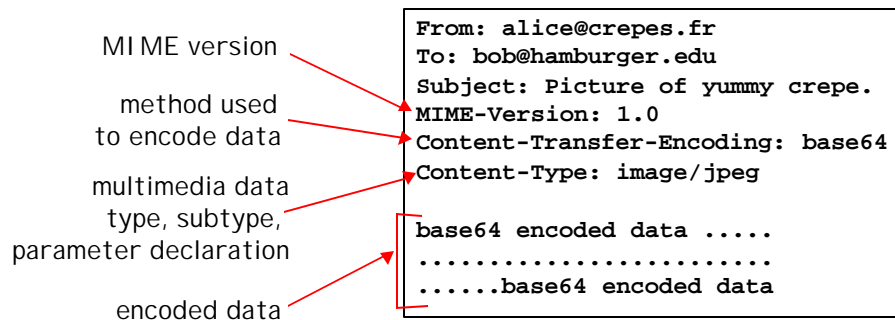
- r header lines
(*keyword: values*), e.g.,
 - m To:
 - m From:
 - m Subject:*different from smtp commands!*
- r body
 - m the "message", ASCII characters only



2: Application Layer 35

Message format: multimedia extensions

- r MIME: Multipurpose Internet Mail Extension, RFCs 2045-2049, especially RFC 2045 and 2046
- r additional lines in msg header declare MIME content type



2: Application Layer 36

MIME types

Content-Type: type/subtype; parameters

Text

r example subtypes: **plain**,
html

Video

r example subtypes: **mpeg**,
quicktime

Image

r example subtypes: **jpeg**,
gif

Application

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

Audio

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

2: Application Layer 37

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
```

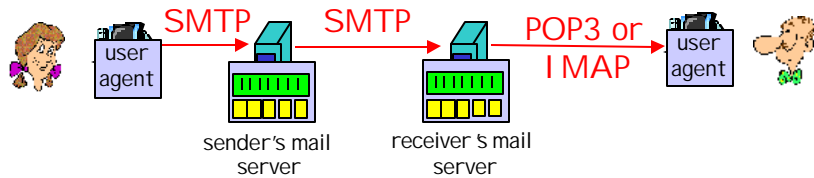
```
Dear Bob,
Please find a picture of a crepe.
```

```
--98766789
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
```

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

2: Application Layer 38

Mail access protocols



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

2: Application Layer 39

POP3 protocol

authorization phase

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

transaction phase, client:

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

2: Application Layer 40