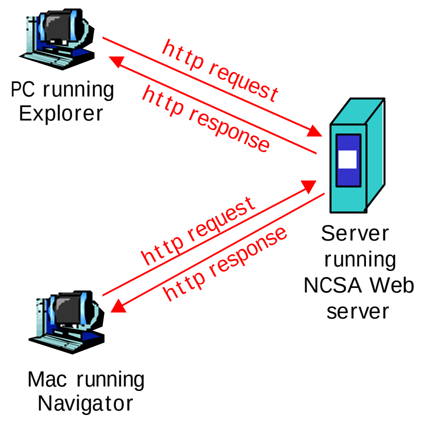
Internet apps: their protocols and transport protocols

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

WWW: the http protocol

http: hypertext transfer protocol

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



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

time

2: Application Layer 11

http example (cont.)

4. http server closes TCP connection.

5. http client receives response message containing html file, displays html. Parsing html file, findis10 referenced jpeg objects

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

time

- non-persistent connection: one object in each TCP connection
  - some browsers create multiple TCP connections *simultaneously* - one per object
- persistent connection: multiple objects transferred within one TCP connection

2: Application Layer 12

http message format: request

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

request line (GET, POST, HEAD commands)

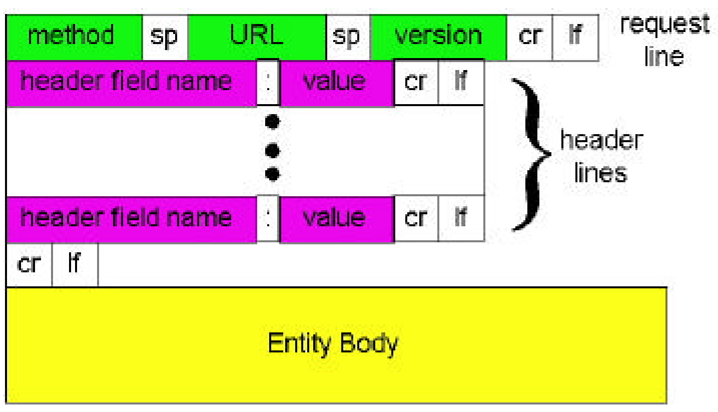
```
GET /somedir/page.html HTTP/1.1
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr
```

header lines

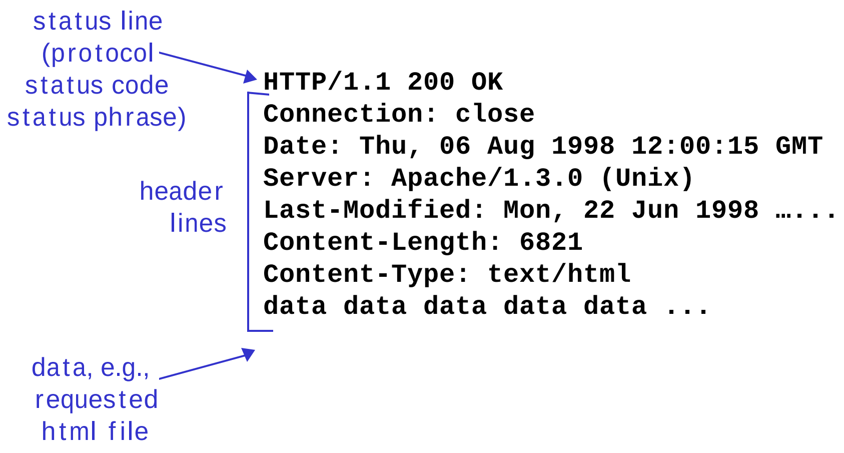
(extra carriage return, line feed)

Carriage return, line feed indicates end of message

http request message: general format



http message format: reply



```
HTTP/1.1 200 OK
Connection: close
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 ...
```

Trying out http (client side) for yourself

1. Telnet to your favorite WWW 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

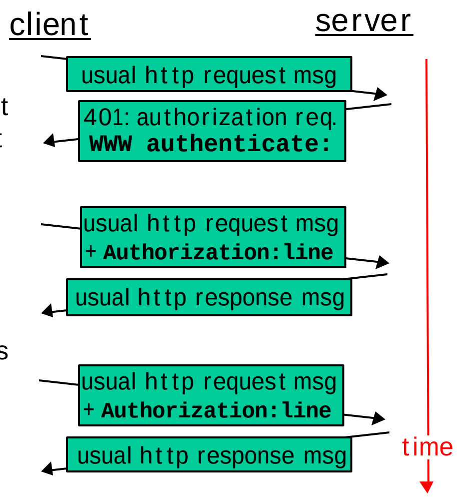
Authentication goal: control access to server documents

r stateless: client must present authorization in each request

r authorization: typically name, password

  m **authorization:** header line in request

  m if no authorization presented, server refuses access, sends
  WWW authenticate:
  header line in response



User-server interaction: cookies

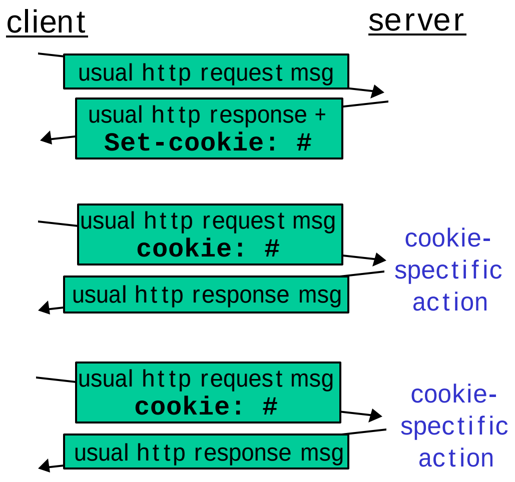
r server sends “cookie” to client in response
Set-cookie: #

r client present cookie in later requests
cookie: #

r server matches presented-cookie with server-stored cookies

  m authentication

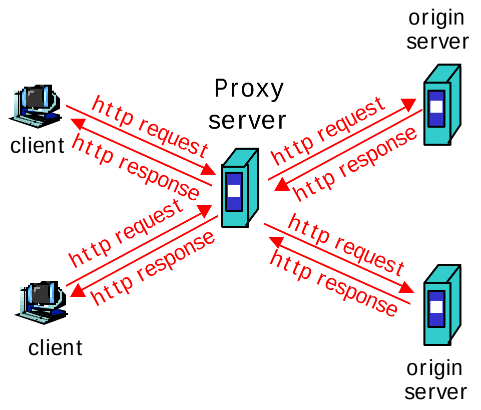
  m remembering user preferences, previous choices



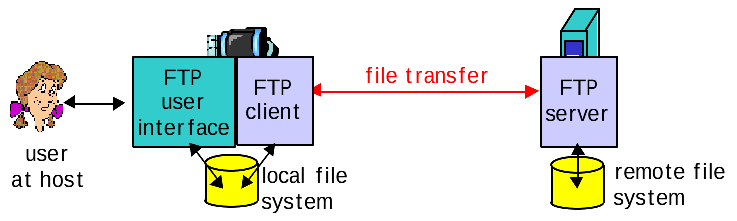
Web Caches (proxy server)

Goal: satisfy client request without involving origin server

- user sets browser: WWW 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



ftp: the file transfer protocol

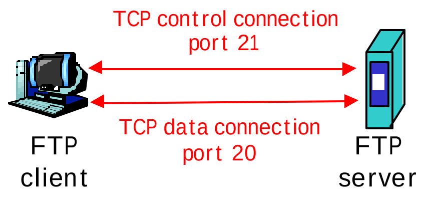


- transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ftp: RFC 959
- ftp server: port 21

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



2: Application Layer 24

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**

2: Application Layer 25

Electronic Mail

Three major components:

r user agents

r mail servers

r simple mail transfer protocol: smtp

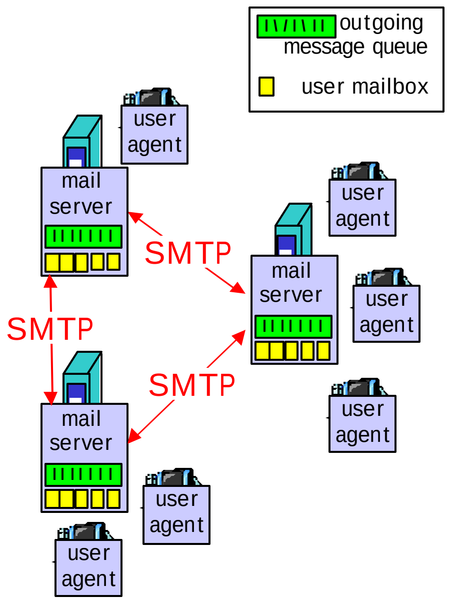
User Agent

r a.k.a. “mail reader”

r composing, editing, reading mail messages

r e.g., Eudora, pine, elm, Netscape Messenger

r outgoing, incoming messages stored on server



2: Application Layer 26

Electronic Mail: mail servers

Mail Servers

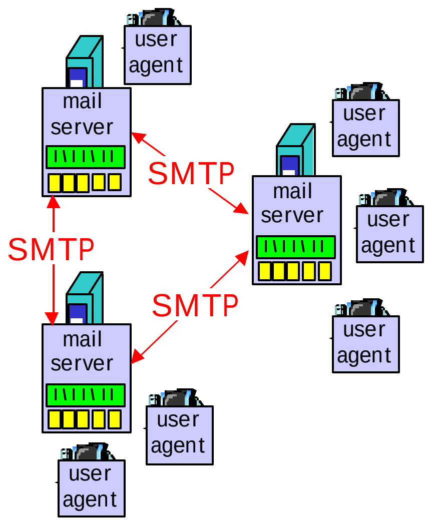
r mailbox contains incoming messages (yet ot be read) for user

r message queue of outgoing (to be sent) mail messages

r smtp protocol between mail server to send email messages

m client: sending mail server

m “server”: receiving mail server



2: Application Layer 27

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
  - closure
- command/response interaction
  - commands: ASCI text
  - response: status code and phrase

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
```

smtp: final words

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)

Comparison with http

- http: pull
- email: push
- both have ASCII command/response interaction, status codes
- http: multiple objects in file sent in separate connections
- smtp: multiple message parts sent in one connection

2: Application Layer 30

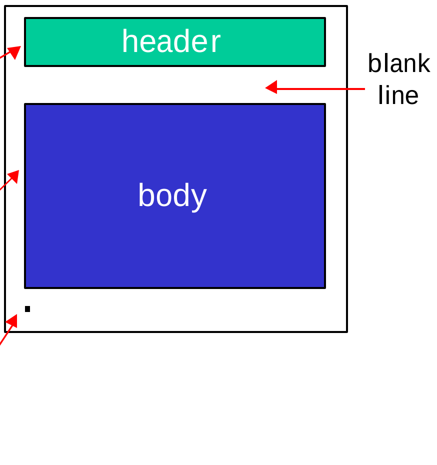
Mail message format

smtp: protocol for exchanging email msgs

RFC 822: standard for text message format:

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

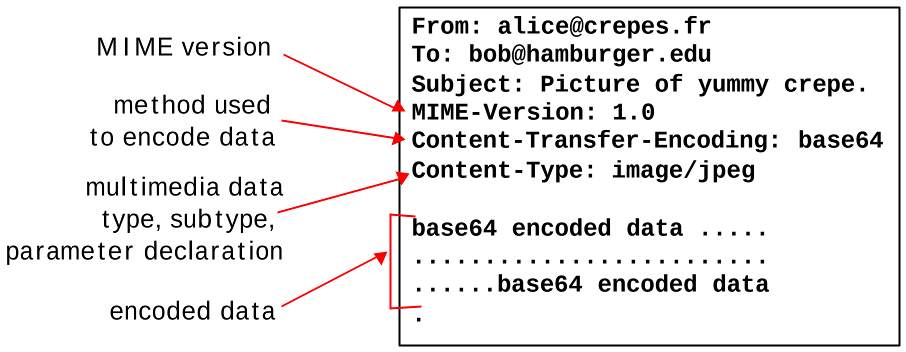
  different from smtp commands!
- body
  - the "message", ASCII characters only
- line containing only `.'



2: Application Layer 31

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

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

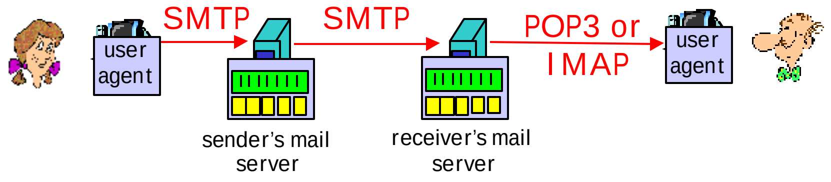
Image
- example subtypes: **jpeg, gif**

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**

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

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
```