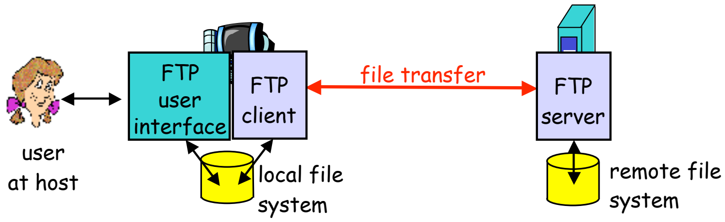
ftp: File Transfer Protocol



- ftp specification: RFC 959 (http://www.ietf.org/rfc/rfc959.txt)

data connection management

ftp commands, responses

over 30 are available

- sent as ASCII text over control conn.
- authentication: user, pass
- file access: e.g. put, get
- file transfer control: mode
- directory: pwd, list, delete
- ftp session: help, stat, abort, quit

Sample commands:

- USER ***username***
- PASS ***password***
- LIST: return list of file in the 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)

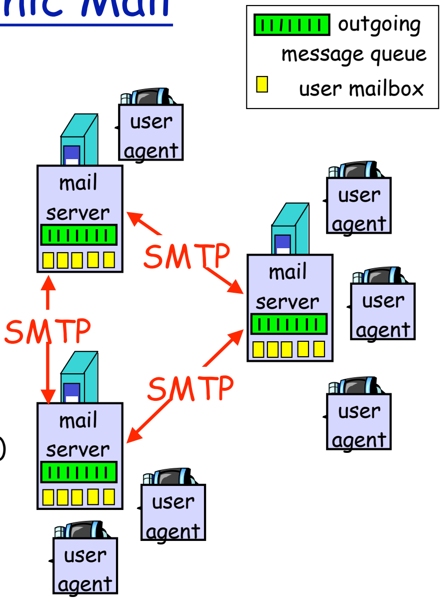
User Agent

- composing, editing, reading mail msgs
  - Eudora, Outlook, elm, Netscape Messenger
- outgoing, incoming messages stored on server

Mail Servers

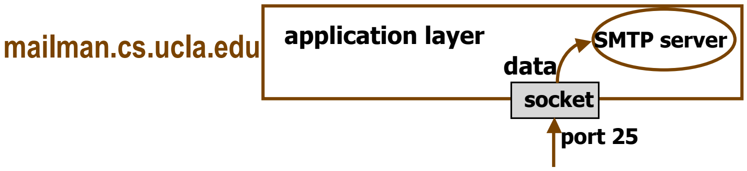
- mailbox contains incoming messages (yet to be read) for user
- message queue of outgoing (to be sent) mail messages

SMTP protocol between mail servers

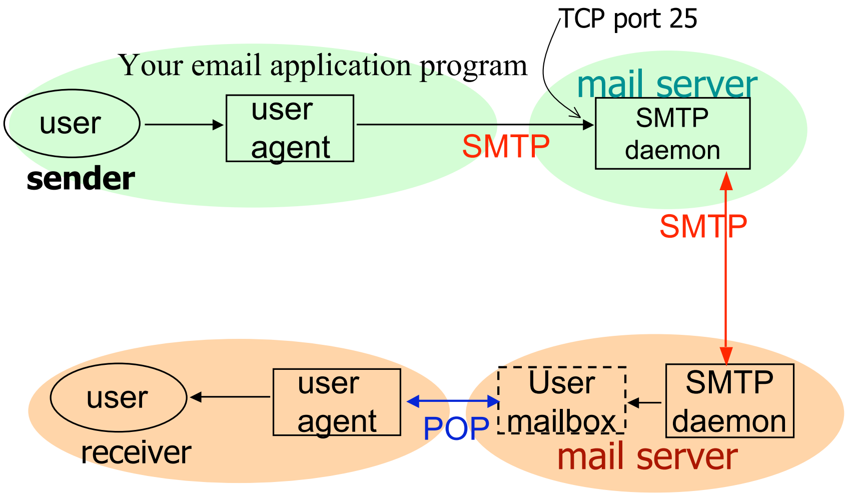


how a sender contacts a SMTP server

- an SMTP server process running on every SMTP server host, waiting for incoming mail
- TCP port# (25) is permanently assigned to SMTP ("well-known port")
- sender opens a TCP connection to the dest.



Email delivery

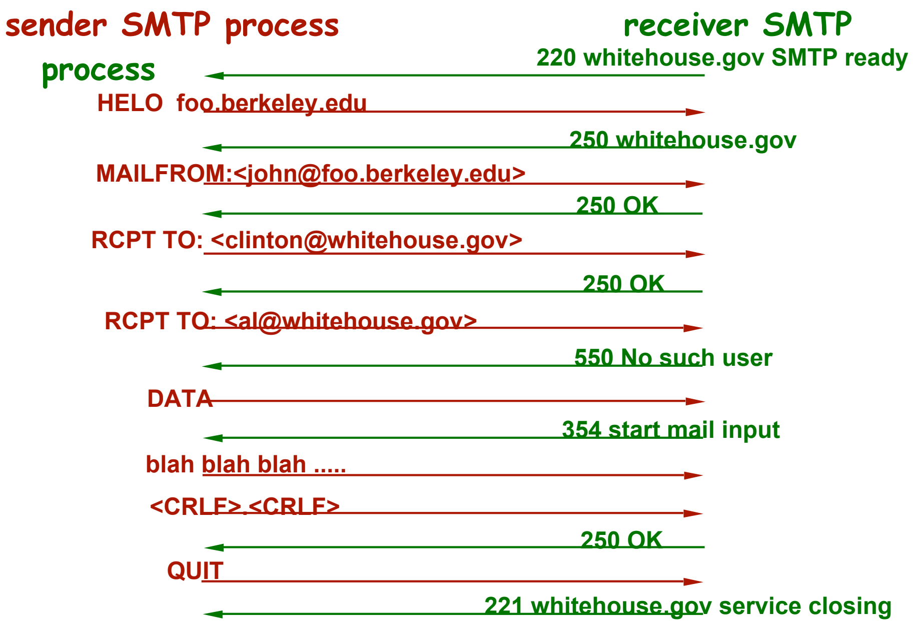


Simple Mail Transfer Protocol [RFC 821]

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
(if more msgs to send, start from "MAIL FROM" again)
C: QUIT
S: 221 hamburger.edu closing connection
```

A typical SMTP message exchange (after the TCP connection setup)



sender SMTP process

receiver SMTP process

220 whitehouse.gov SMTP ready

HELO foo.berkeley.edu

250 whitehouse.gov

MAILFROM:<john@foo.berkeley.edu>

250 OK

RCPT TO: <clinton@whitehouse.gov>

250 OK

RCPT TO: <al@whitehouse.gov>

550 No such user

DATA

354 start mail input

blah blah blah .....

<CRLF>.<CRLF>

250 OK

QUIT

221 whitehouse.gov service closing

Are there some basic rules behind the reply codes?

Code	meaning
220	service ready
221	I'm closing too
250	requested action OK
500	error, command not recognized
550	no such mbox, no action taken

Common practices

1st digit: whether response is good/bad/incomplete
e.g. 2= positive completion, 5=negative completion

2nd digit: encodes responses in specific categories
e.g. 2=connections, 5=mail system (status of the receiver mail system)

3rd digit: a finer gradation of meaning in each category specified by the 2nd digit.

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 body must be encoded if it contains forbidden characters
- 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

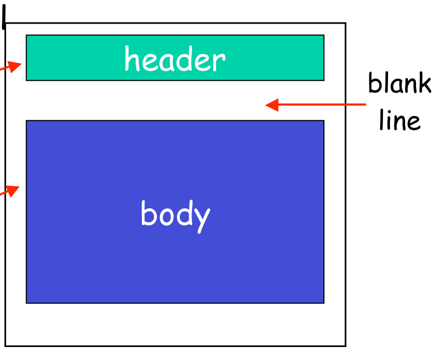
Mail message format

RFC 821: SMTP specification (protocol for exchanging email msgs)

RFC 822: standard for text message format:

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

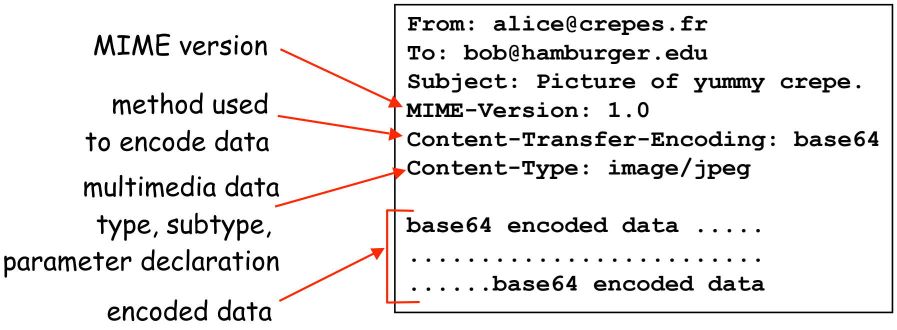
  different from smtp commands!
- body
  - the "message", ASCII characters only



Message format: extension for multimedia

MIME: Multipurpose Internet Mail Extension

- additional lines in msg header declare MIME content type



MIME version

method used to encode data

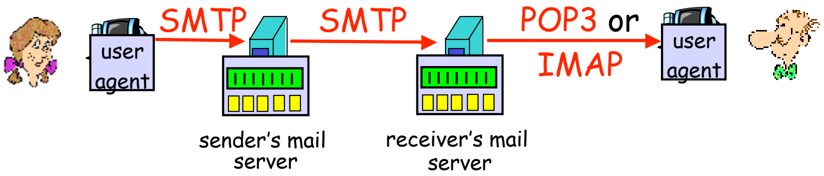
multimedia data type, subtype, parameter declaration

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
```

Mail access protocols



Mail access protocol: retrieval from mail server

- POP: Post Office Protocol [RFC 1939]
  - authorization (agent <-->server) and download
- IMAP: Internet Mail Access Protocol [RFC 1730]
  - more features, such as msg folders on the server
    - more complex implementation
  - 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:

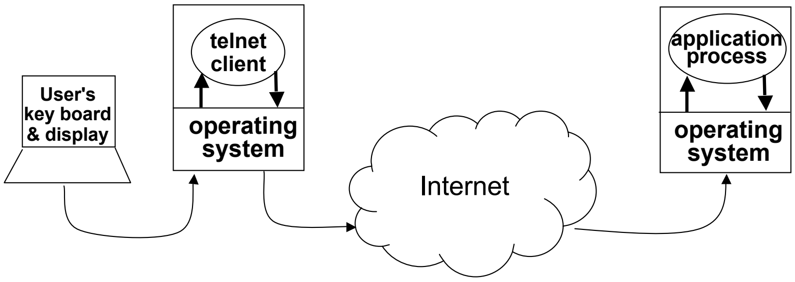
- **list:** list message numbers
- **retr:** retrieve message by number
- **dele:** delete
- **quit**

```
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
```

telnet (RFC854)

- A TCP connection used to transmit data with interspersed TELNET control information
- Client side of the TCP connection initiates a request, the server accepts or rejects the request.
- Telnet server uses port# 23
  - the client side can use any unreserved port.



client-server paradigm

- any program can become a network application client when it needs network services
- servers are special purpose applications dedicated to providing specific service
  - server processes start at system initialization time
- applications at both ends take initiative
  - server application informs local OS that it is ready to take incoming messages
    - wait for incoming messages
    - perform requested service
    - return results
  - client application contacts the server
    - send request
    - wait for reply

identifying servers and services

- each service is assigned a unique well-known port number
- server application process registers with local protocol software with that port #
- a client requests a service by sending request to a specific server host with the well-known port #
- server handles multiple requests concurrently

Chapter 3: Transport Layer

Chapter goals:

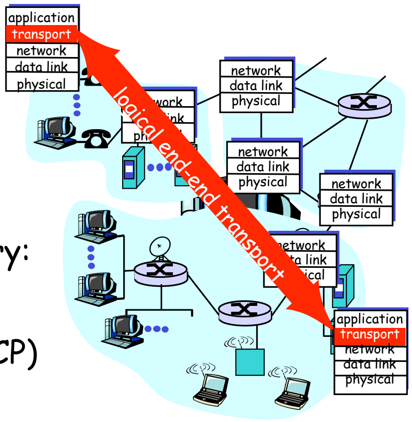
- Principles behind transport layer services:
  - multiplexing/demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation and implementation in the Internet

Chapter Overview:

- transport layer services; multiplexing/demultiplexing
- connectionless transport: UDP
- connection-oriented transport: TCP
  - How to achieve reliable data delivery
- TCP congestion control

Transport services and protocols

- data delivery between app' processes running on different hosts
- transport vs network layer services:



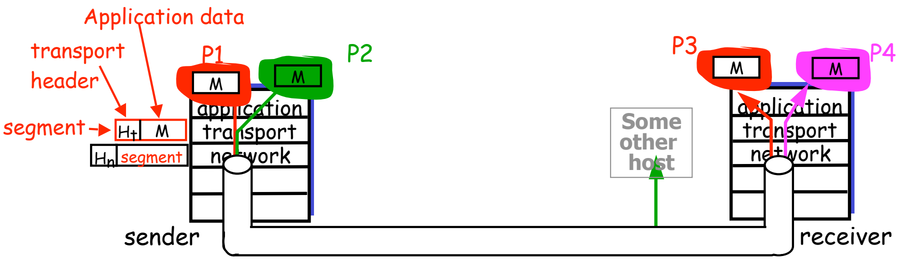
Internet transport services:

- unreliable, unordered delivery: UDP
- reliable, in-order delivery(TCP)

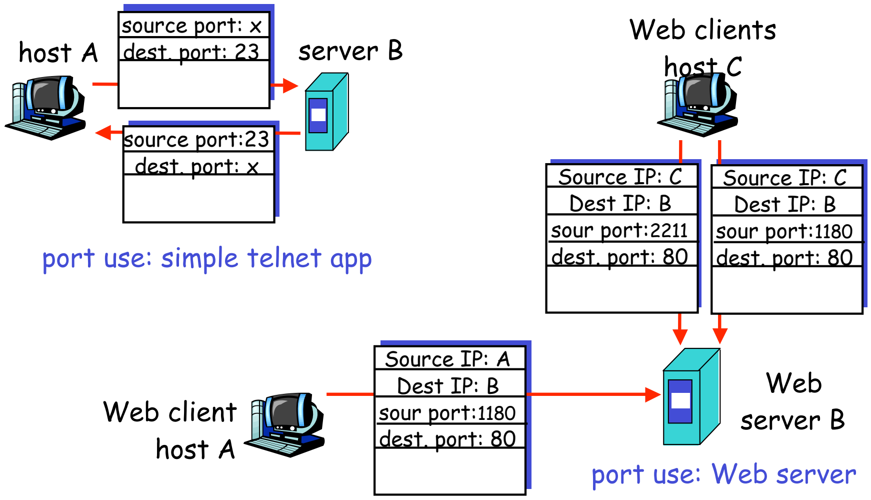
Multiplexing/demultiplexing

Multiplexing
data segments from multiple app processes is sent to lower layer for transmission

Demultiplexing
delivering received data segments to corresponding upper layer protocols/apps



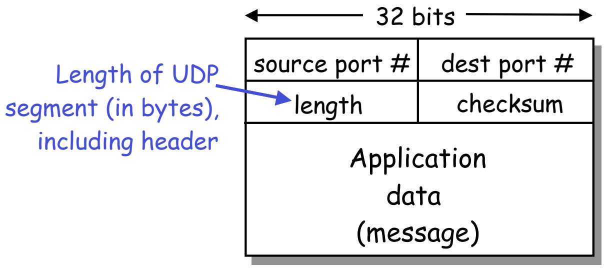
Multiplexing/demultiplexing: examples



UDP: User Datagram Protocol [RFC 768]

- "best effort" service: UDP segments may be lost, or delivered out of order to applications
- connectionless:

UDP format



32 bits

source port #	dest port #
length	checksum
Application data (message)	

Length of UDP segment (in bytes), including header

UDP checksum

Goal: detect bit errors (e.g., flipped bits) in transmitted segment

Sender:

- treat data in the segment as sequence of 16-bit integers
- checksum: addition (1's complement sum) of segment contents
- puts checksum value into UDP checksum field

Receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected

Internet checksum algorithm

- used in IP, TCP, UDP
- sender:
  - consider the data block as 16xn matrix
  - add all data together using 16-bit one's complement arithmetic
  - take the one's complement of the result
- receiver
  - add all bytes together, including the checksum field
  - if sum=0, no bit error

checksum computation: Sample code

```
U_short checksum(u_short *buf, int length)
{
  unsigned long sum = 0;
  if (length % 2) {
    /* pad the data length to be an even number of bytes */
    length += 1;
    }
  length >>= 1;
  while (length--) {
    sum += *buf++;
    if (sum & 0xFFFF0000) {          /*carry occurred, wrap around */
      sum &= 0xFFFF);
      sum++;
    }
  }
return (~sum & 0xFFFF);
}
```