Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

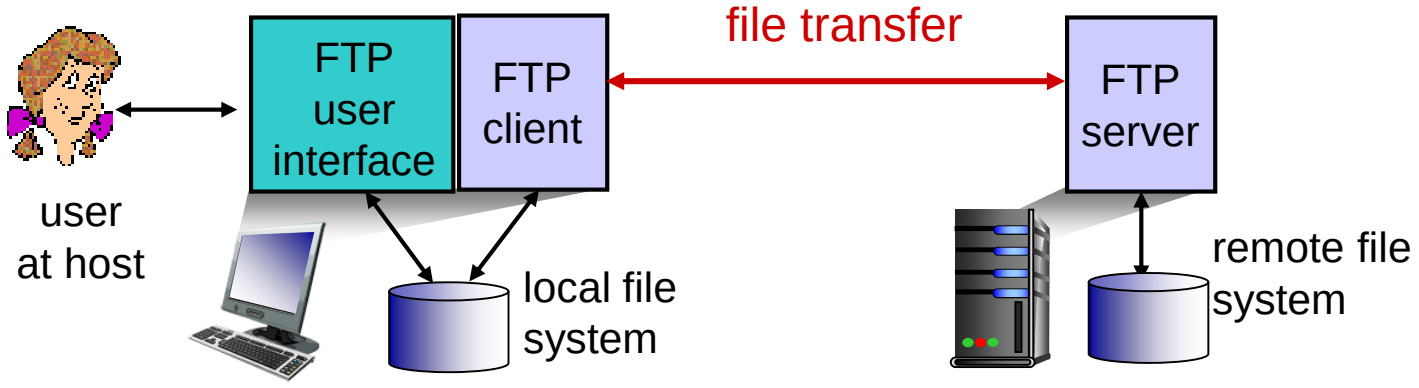
2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

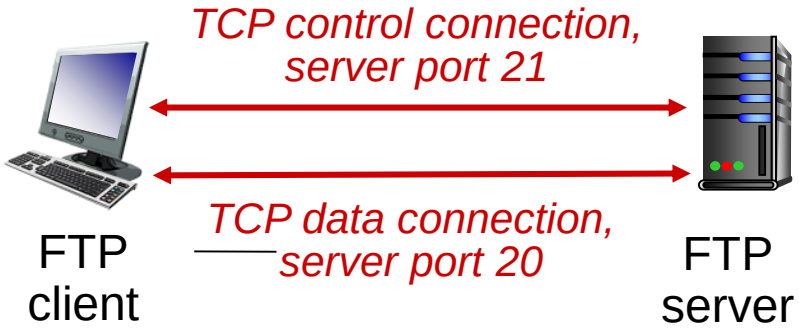
FTP: the file transfer protocol



- 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, using TCP
- client authorized over control connection
- client browses remote directory, sends commands over control connection
- when server receives file transfer command, *server* opens 2^{nd} 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

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`**

SFTP Demo

Demo of sftp from flip.engr.oregonstate.edu

- Commands:
  - pwd
  - lpwd
  - put
  - get

FTP doesn't cut it nowadays

totally insecure

- SFTP is the encrypted version
- Normally on port 22
  - Same port as SSH, because it IS using SSH

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

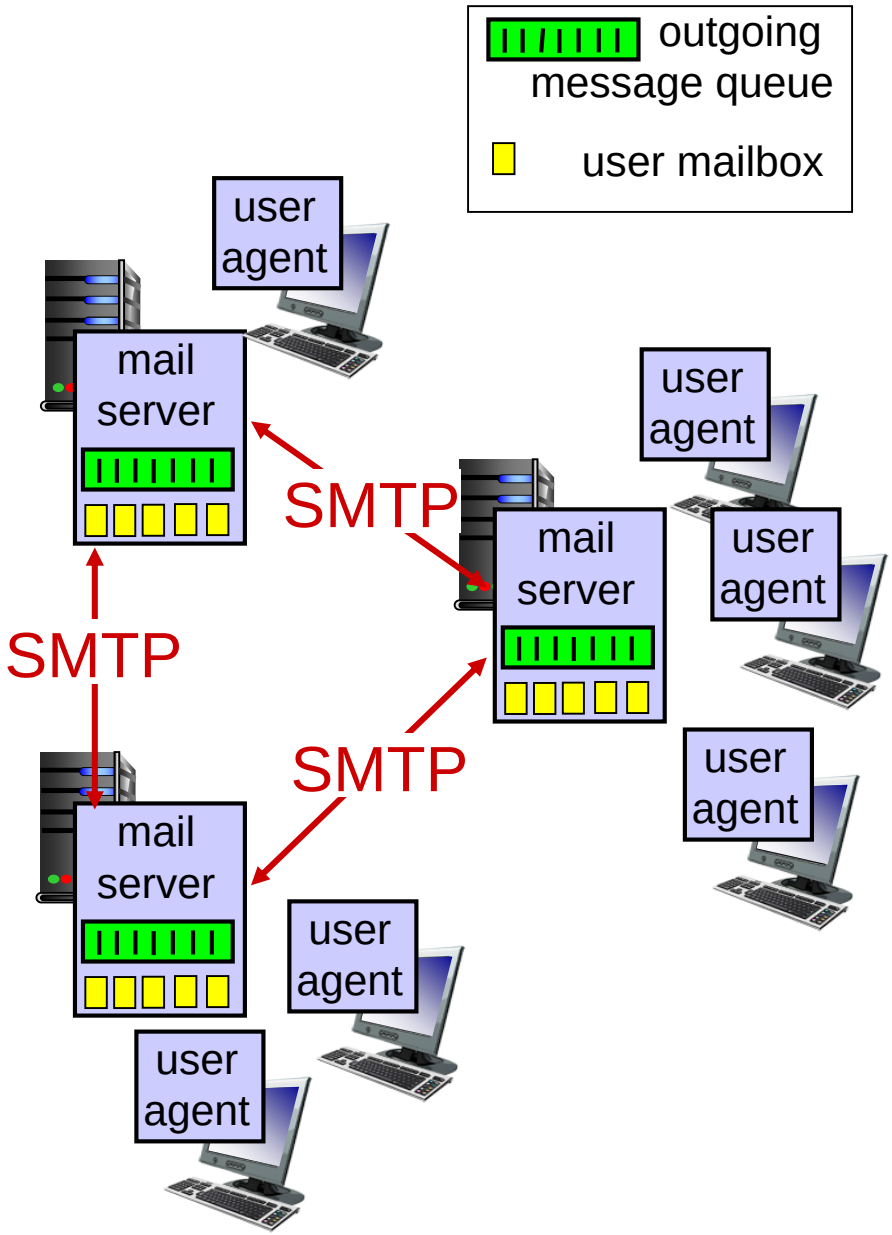
Electronic mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

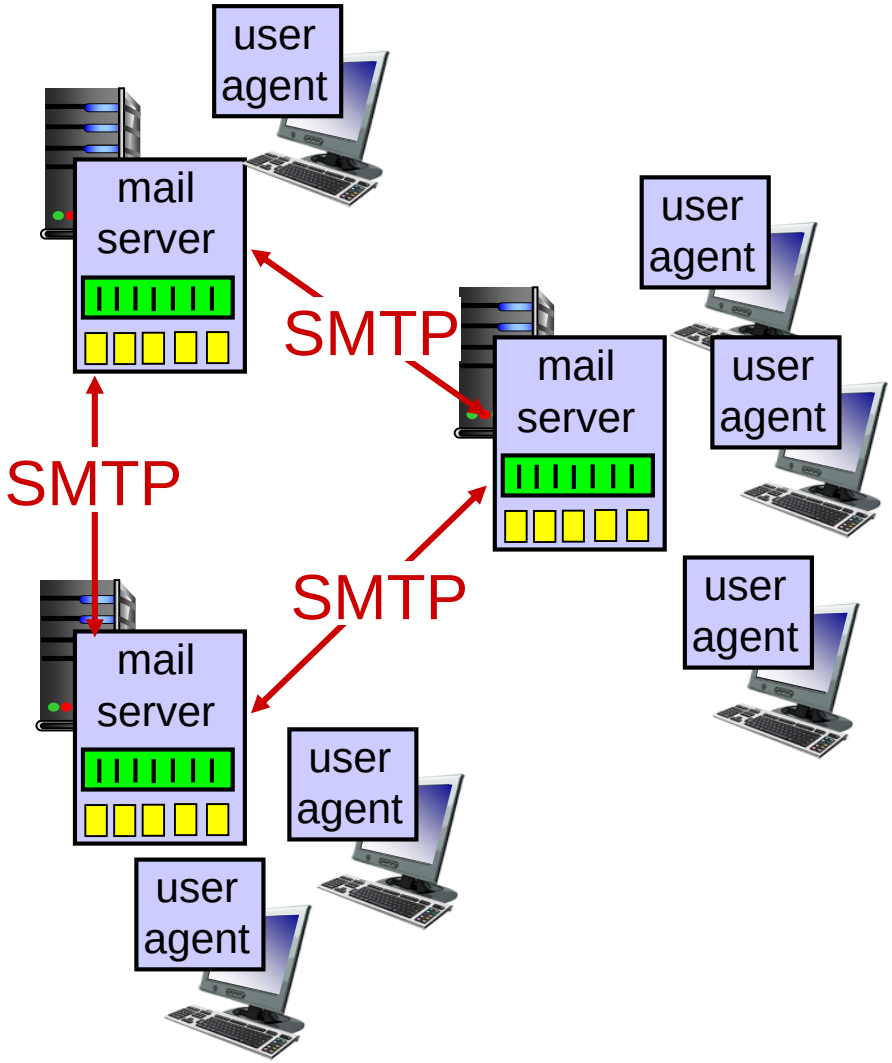
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



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

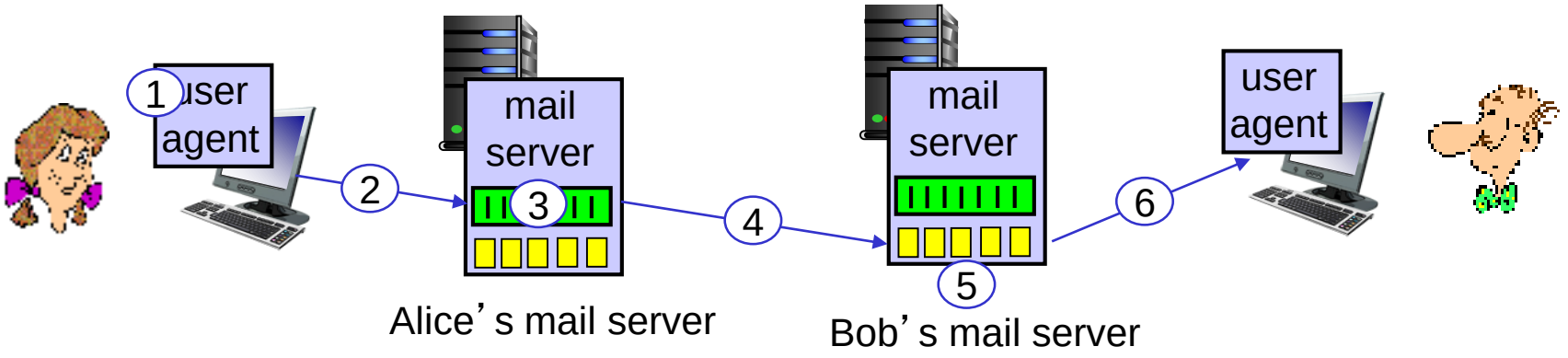


Electronic Mail: SMTP [RFC 2821]

- 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 ASCI

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



Sample SMTP interaction – no security

```
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
```

Sample SMTP interaction – w/ security

```
# First step Generate an authentication key

$ perl -MMIME::Base64 -e 'print encode_base64("\000myemail\@gmail.com\000mypassword")'
AG5pY2UudHJ5QGdtYWlsLmNvbQBub2l0c25vdG15cGFzc3dvcmQ=
```

Sample SMTP interaction – w/ security

```
$ openssl s_client -connect smtp.gmail.com:465 -crlf -ign_eof
         [... lots of openssl output ...]
220 mx.google.com ESMTP m46sm11546481eeh.9
EHLO localhost
250-mx.google.com at your service, [1.2.3.4]
250-SIZE 35882577
250-8BITMIME
250-AUTH LOGIN PLAIN XOAUTH
250 ENHANCEDSTATUSCODES
AUTH PLAIN AG5pY2UudHJ5QGdtYWlsLmNvbQBub2l0c25vdG15cGFzc3dvcmQ=
235 2.7.0 Accepted
MAIL FROM: <gryphius-demo@gmail.com>
250 2.1.0 OK m46sm11546481eeh.9
rcpt to: <somepoorguy@example.com>
250 2.1.5 OK m46sm11546481eeh.9
DATA
354  Go ahead m46sm11546481eeh.9
Subject: it works

yay!
.
250 2.0.0 OK 1339757532 m46sm11546481eeh.9
quit
221 2.0.0 closing connection m46sm11546481eeh.9
read:errno=0
```

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

comparison with HTTP:

- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

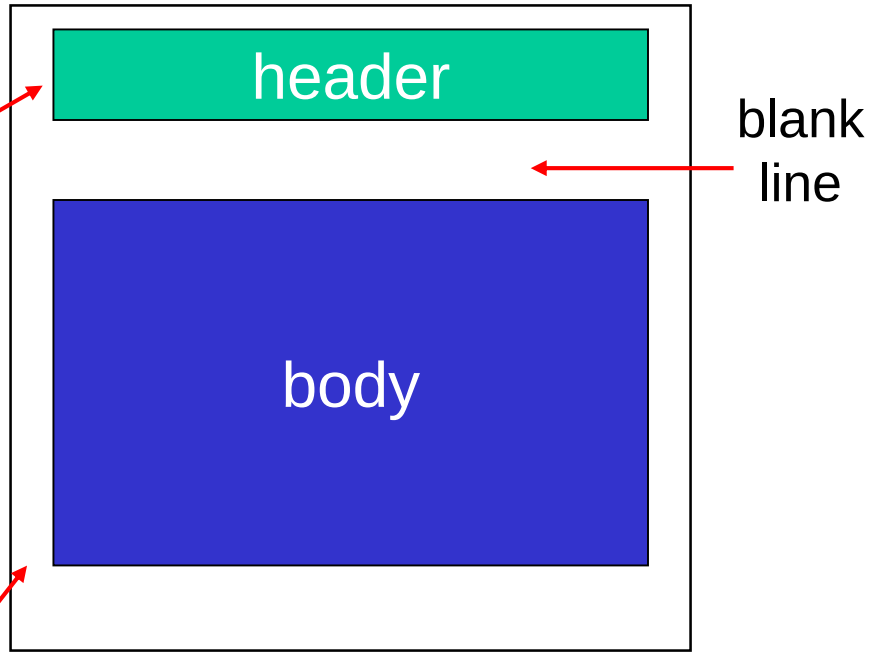
Mail message format

SMTP: protocol for exchanging email msgs

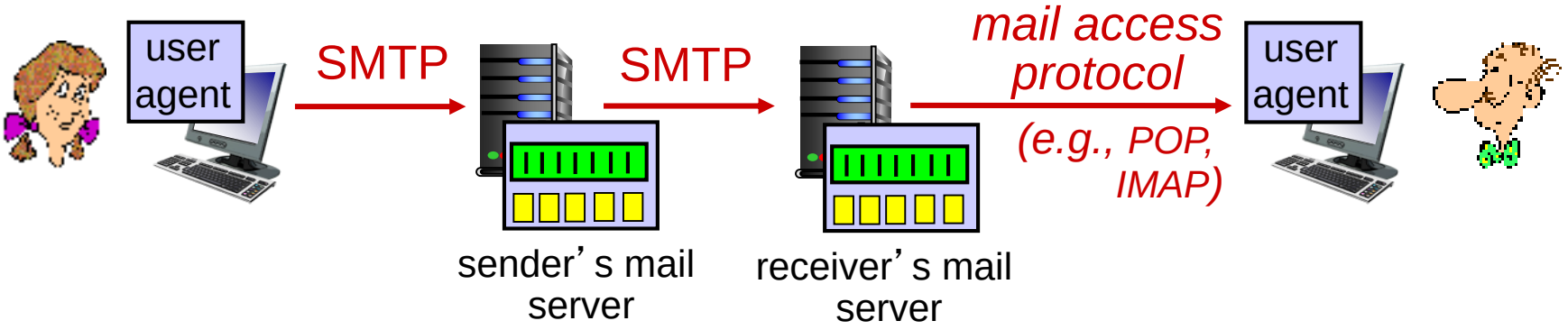
RFC 822: standard for text message format:

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

  different from SMTP MAIL FROM, RCPT TO: commands!
- Body: the "message"
  - ASCII characters only



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.
  - Exchange: Microsoft's implementation, de facto for business

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
```

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

Microsoft Exchange

winning

- Uses proprietary protocol called MAPI
- Managed by Microsoft Exchange Server
- Includes (near-instant) synchronization of calendar, email, contacts, etc. across many devices at once
- De facto business email system

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP