

CSCE 463/612

Networks and Distributed Processing

Spring 2017

Application Layer II

Dmitri Loguinov

Texas A&M University

February 7, 2017

Original slides copyright © 1996-2004 J.F Kurose and K.W. Ross

Chapter 2: Roadmap

2.1 Principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 Electronic Mail

- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P file sharing

2.7 Socket programming with TCP

2.8 Socket programming with UDP

2.9 Building a Web server

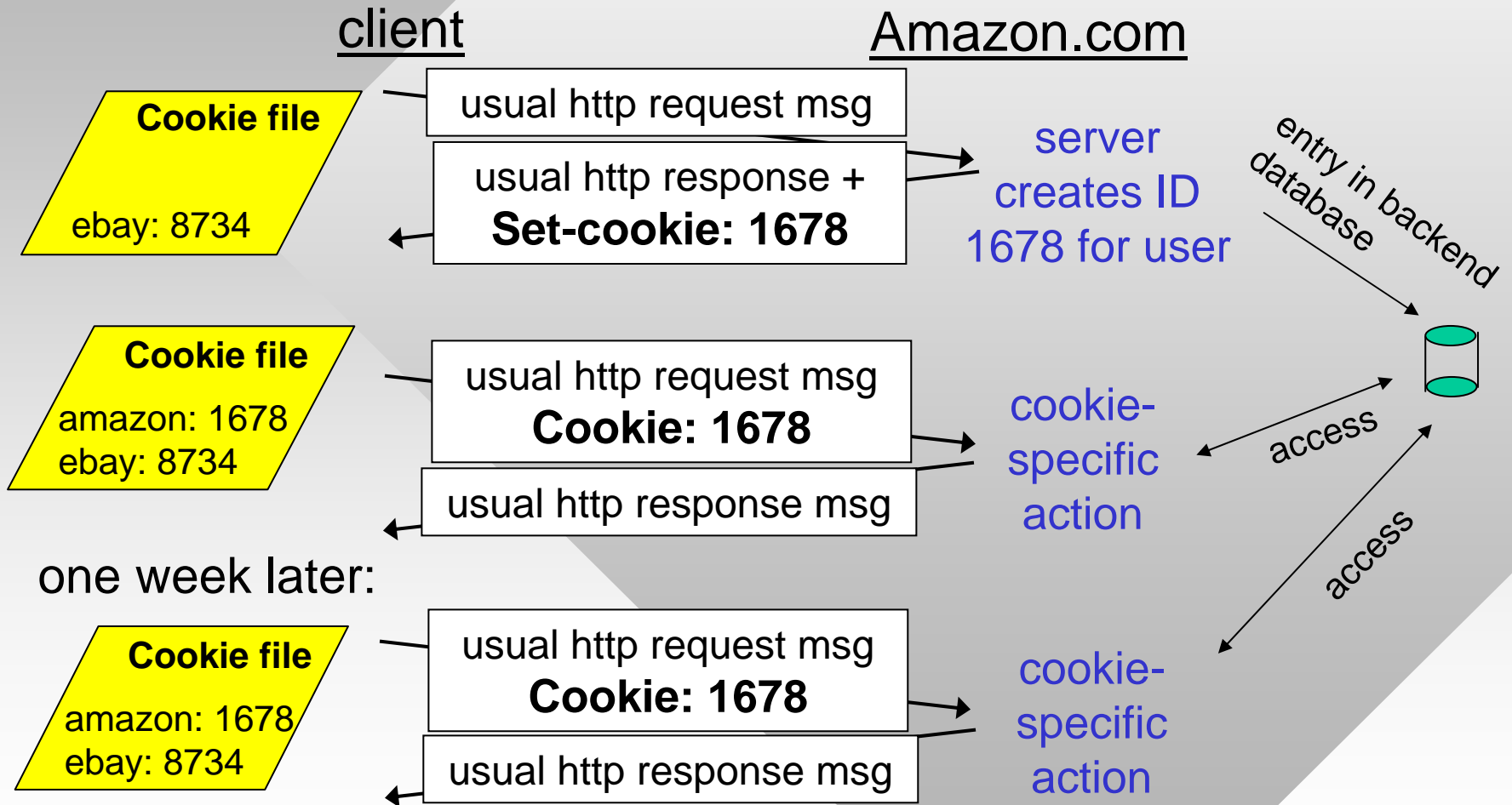
User-Server State: Cookies

- User visits the same web site multiple times
 - Doesn't want to type password or make selections each time
- Website remembers info about the user
 - Pages viewed, items bought, credit cards used
 - Zip code and cable channels (tvguide.com)
 - Weather.com
 - csnet.cse.tamu.edu

Four components:

- **Cookie header** line in the HTTP response message
- **Cookie file** kept on user's host and managed by user's browser
- **Cookie header** line in HTTP request message
- Back-end **database** at Web site

Cookies: Keeping State



Cookie Example

```
telnet irl.cs.tamu.edu 80
GET / HTTP/1.0
```

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 18 Sep 2007 18:47:25 GMT
Server: Microsoft-IIS/6.0
```

```
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
```

```
Content-Length: 6916
Content-Type: text/html
```

```
Set-Cookie: ASPSESSIONIDACSRQCTQ=PIGHLBAAJICJONABJFINMLOA;
path=/
```

```
Cache-control: private
```

cookies expire when browser is closed; optionally, the Expires directive allows extending this into the future

path prefix where cookie is valid

cookie value

shared caching not allowed

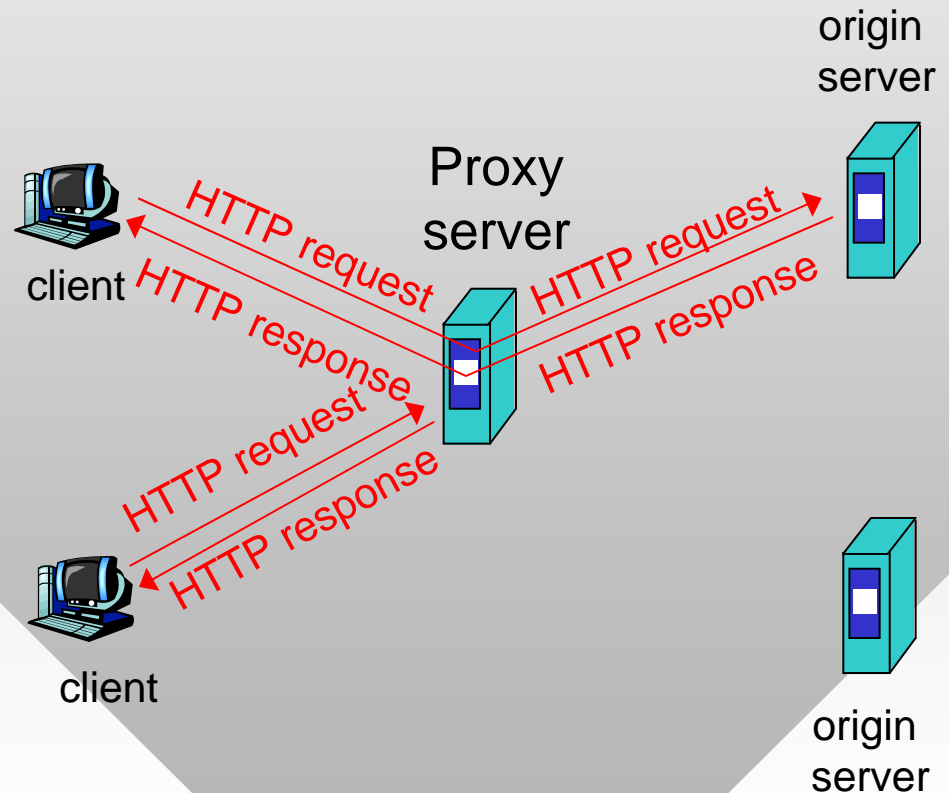
Cookies (continued)

- Cookie file location is browser-dependent
 - For example, Internet Explorer:
C:\Users*<user>*\AppData\Roaming\Microsoft\Windows\Cookies
 - Impersonation is possible by copying or intercepting your cookies (through sniffing and malicious scripting)
- Other privacy issues
 - Websites accumulate data about users (form input, actions), share this information with others
 - Hacked websites leak this even further
- **Incognito** browsing mode starts with no cookies
 - New cookies are accepted and kept until browser is closed

Web Caches (Proxy Server)

Goal: satisfy client request without involving origin server

- Browser explicitly sends requests via cache or cache intercepts all outgoing HTTP traffic
 - Object in cache: cache returns object
 - Else cache requests object from origin server, then returns object to client



More About Web Caching

- Cache acts as both client and server
- Typically cache is installed by your ISP, university, or company at some network border

Why web caching?

- Reduce response time for client request
- Reduce traffic on the access link

Why web caching (cont'd)

- Reduce load on the servers and allows them to scale to a larger number of users
- Increase security – cached pages can be scanned for viruses before user download is allowed
- Filter URLs to prevent undesirable destinations

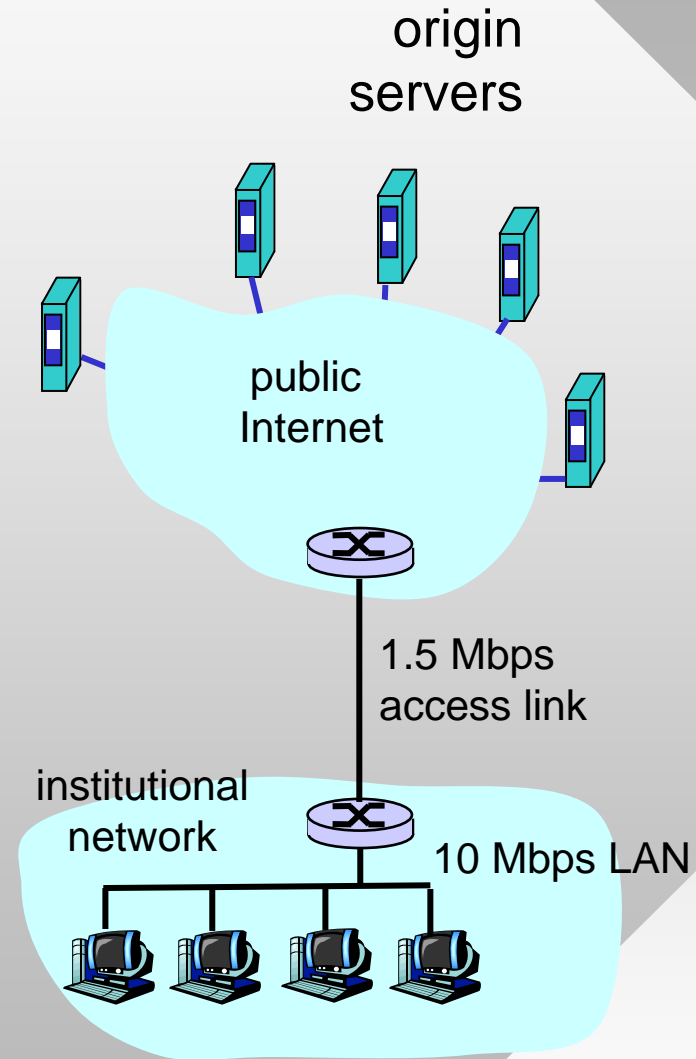
Caching Example

Assumptions

- Average object size = 100,000 bits
- Average request rate from institution's browsers to origin servers = 15/sec
- Delay from ISP router to any origin server and back to router = 2 sec

Consequences

- Utilization on LAN = 15%
- Utilization on access link = 100%
- Total delay = Internet delay + access delay + LAN delay =
= 2 sec + access queuing delay + milliseconds



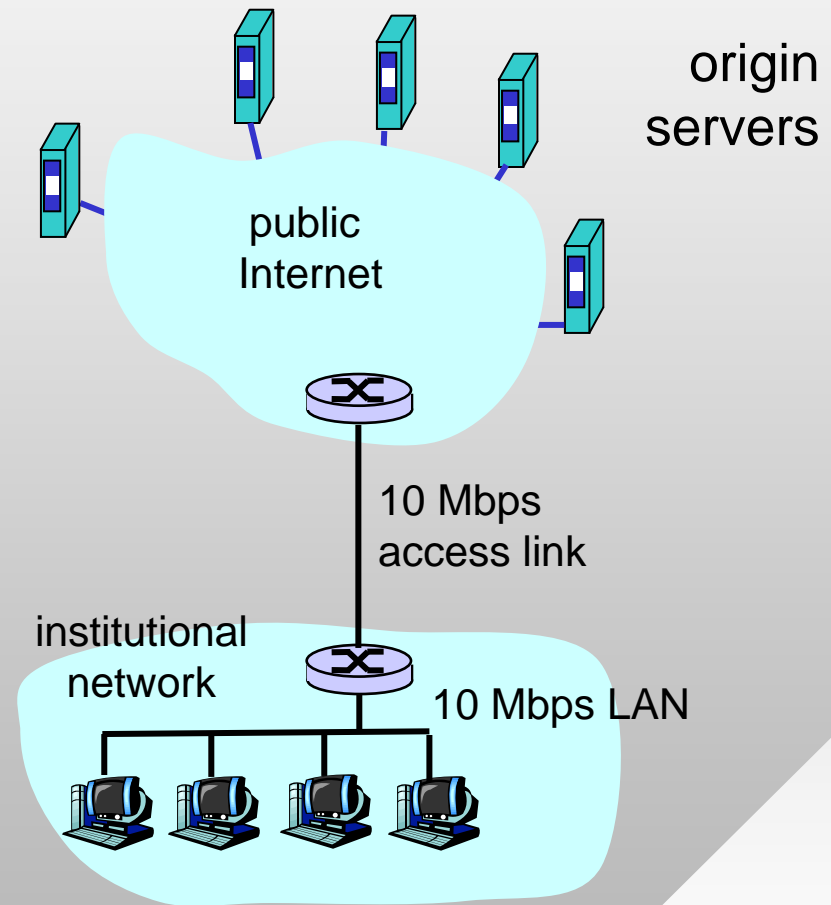
Caching Example (cont)

Possible solution

- Increase bandwidth of access link to, say, 10 Mbps

Consequences

- Utilization on LAN = 15%
- Utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay
= 2 sec + msec + msec
- Often a costly upgrade



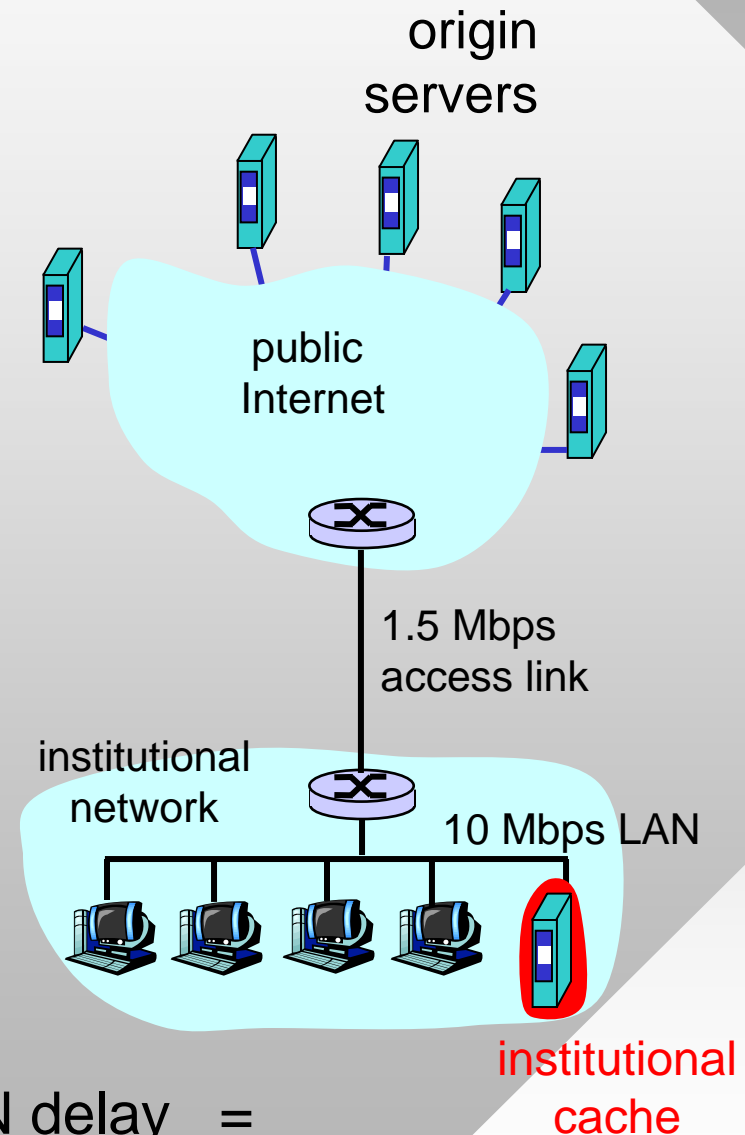
Caching Example (cont)

Install cache

- Suppose hit rate is 40%

Consequences

- 40% of requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- Utilization of access link reduced to 60%, resulting in small queuing delays
- Total average delay =
Internet delay + access delay + LAN delay =
 $0.6 * 2.0 \text{ secs} + \text{msecs} \approx 1.2 \text{ secs}$



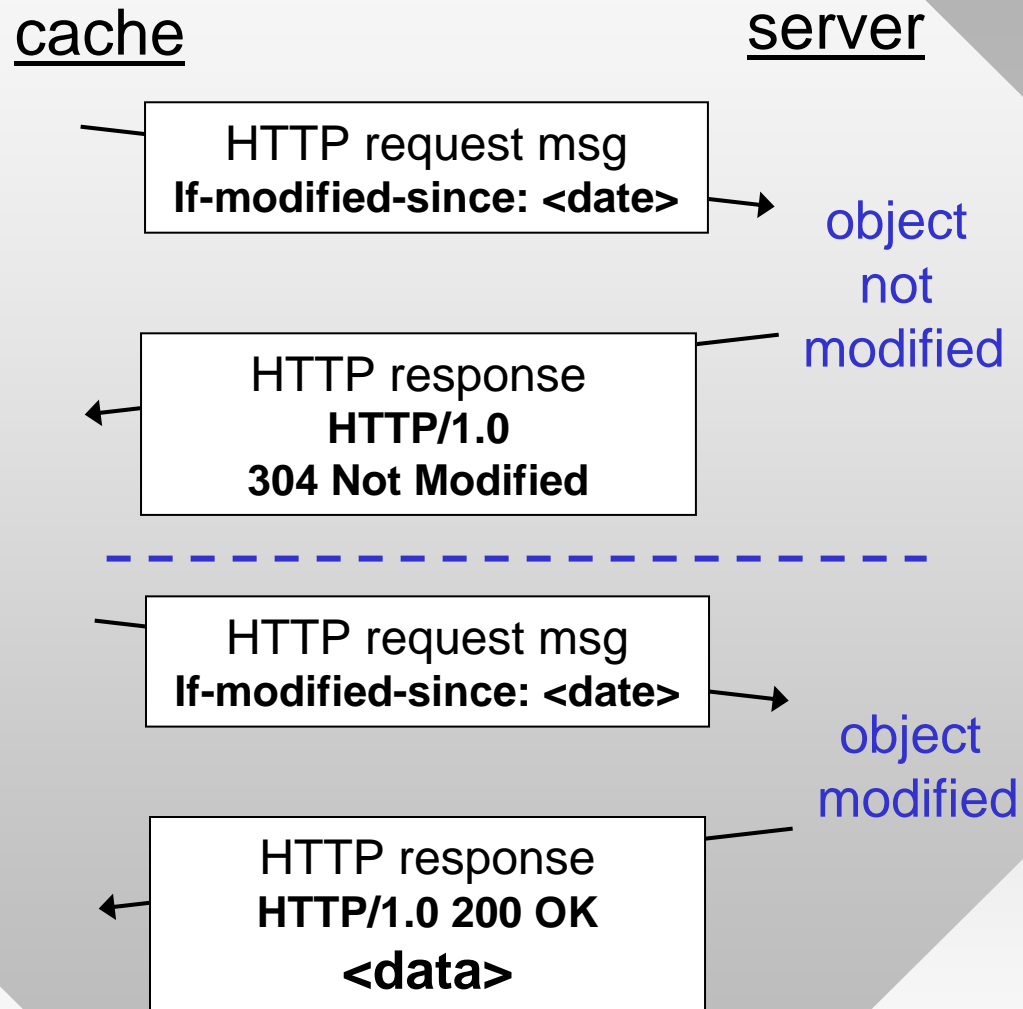
Conditional GET

- **Goal:** don't send object if cache has up-to-date cached version

- Cache: specify date of cached copy in HTTP request
- Server: response contains no object if cached copy is up-to-date

- Expires field in header

- Server may provide date when content expires
- Expires: Sat, 01 Oct 2011 16:00:00 GMT



Chapter 2: Roadmap

2.1 Principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 Electronic Mail

- SMTP, POP3, IMAP

2.5 DNS

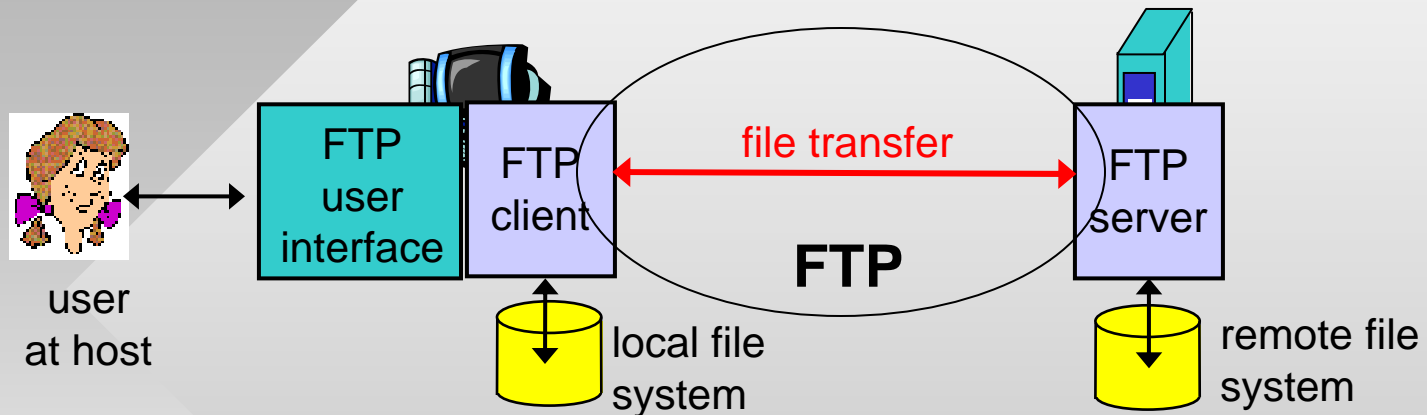
2.6 P2P file sharing

2.7 Socket programming with TCP

2.8 Socket programming with UDP

2.9 Building a Web server

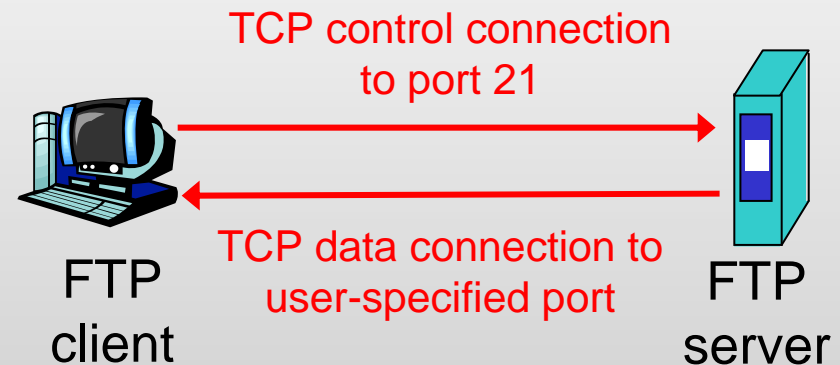
FTP: The File Transfer Protocol



- Transfer file to/from remote host
- Client/server model
 - *client*: initiates the transfer (either to/from remote)
 - *server*: accepts connection on remote host
- FTP: RFC 959 (1985)
- FTP server: port 21

FTP: Separate Control, Data Connections

- FTP client contacts server on port 21 to open **control** connection
 - Obtains authorization over this channel
 - Sends commands for file transfer and/or directory listing
- **Active** mode (default): server opens **data** connection to the client
 - One for each command



- **Passive** mode:
 - Data connection opened by client
 - Useful when client is behind a firewall
- After transferring object, sender closes **data** connection

FTP Commands, Responses

Sample commands:

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

Connection management

- Active mode (PORT)
 - PORT tells the server to which <IP, port> to issue a connect
 - Third party IP is OK in theory, DDoS possibility
- Passive mode (PASV)
 - PASV forces the server to open a new socket to which the client can connect

FTP Example

commands do not work until
user is authorized

```
telnet ftp.gnu.org 21
```

```
220 GNU FTP server ready
```

```
HELO
```

```
530 Please login with USER and PASS
```

```
USER anonymous
```

```
230-Due to U.S. Export Regulations, all cryptographic software on this
```

```
230-site is subject to the following legal notice:
```

```
230-   This site includes publicly available encryption source code
```

```
...
```

```
230 Login successful.
```

```
PORT 128,194,135,66,10,5
```

specifies IP 128.194.135.66
and port number 2565

```
----- passive example
```

```
PASV
```

```
227 Entering Passive Mode (140,186,70,20,154,15)
```

status code

IP 140.186.70.20 and port 39439

Chapter 2: Roadmap

2.1 Principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 Electronic Mail

- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P file sharing

2.7 Socket programming with TCP

2.8 Socket programming with UDP

2.9 Building a Web server

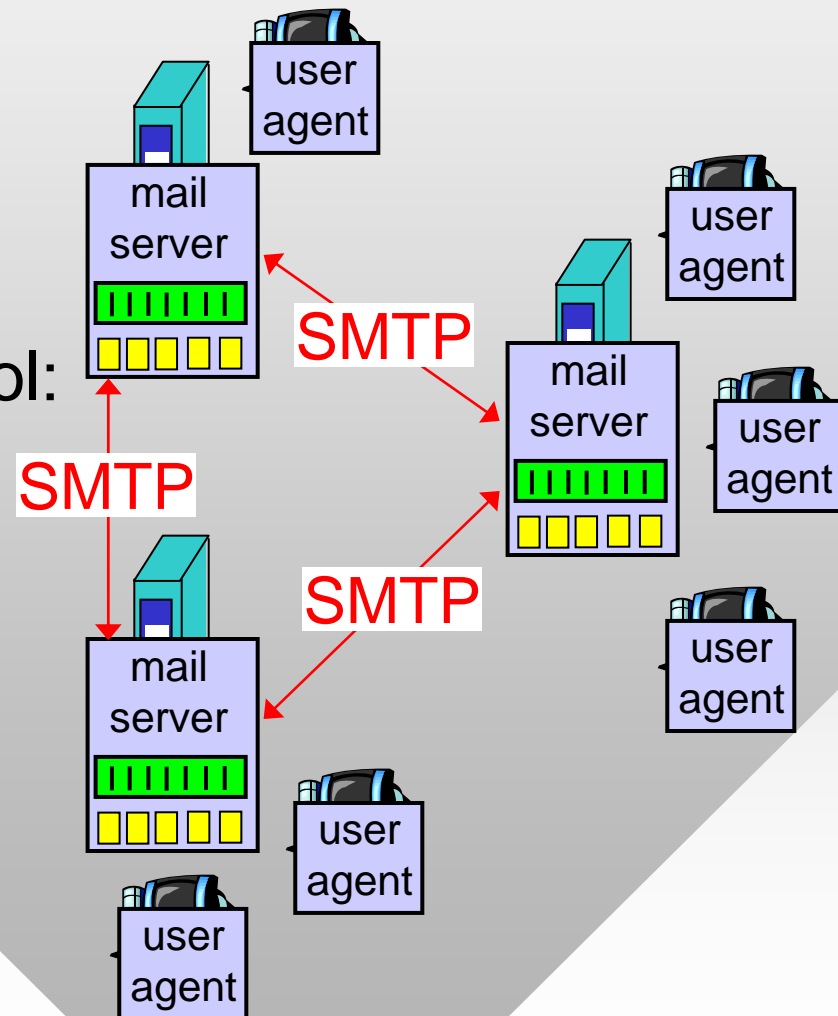
Electronic Mail

Three major components:

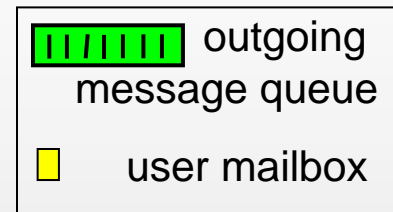
- User agents
- Mail servers
- Simple mail transfer protocol: SMTP

User Agent (UA)

- Mail reader – composing, editing, reading mail messages
 - pine, outlook, elm, thunderbird, iphone



Electronic Mail: Mail Servers

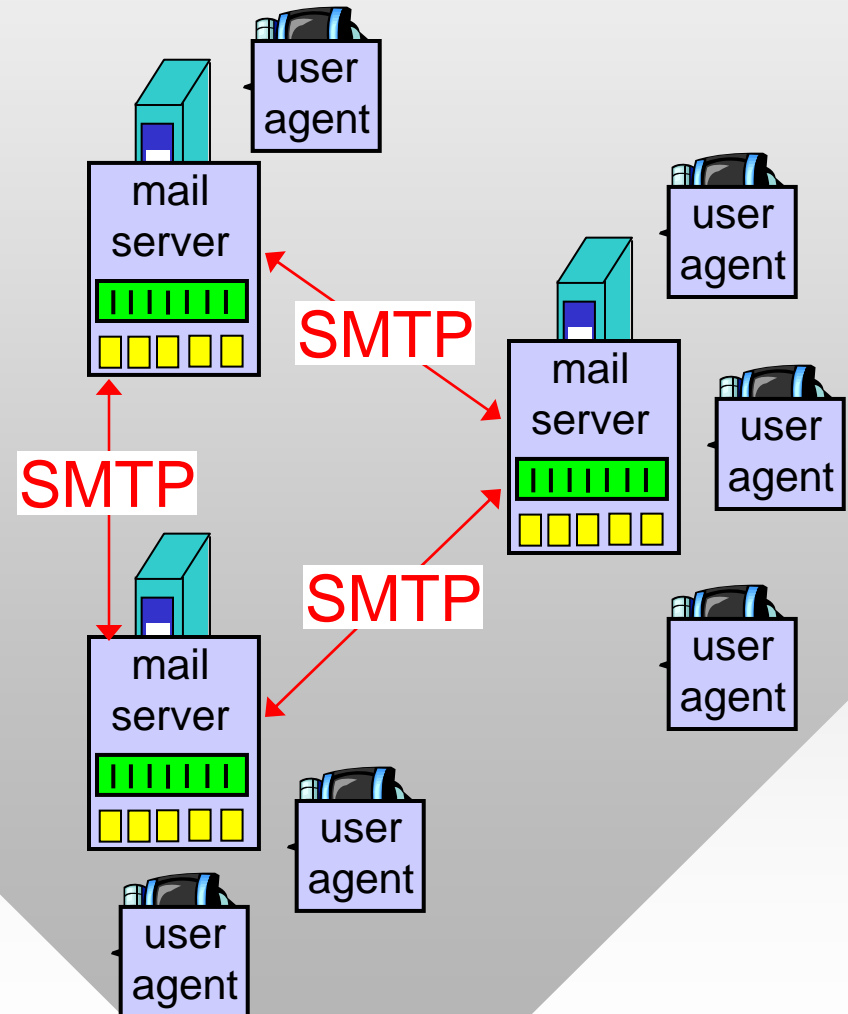


Mail Servers

- Message queue of outgoing (to be sent) mail messages
- Mailbox contains incoming messages for user

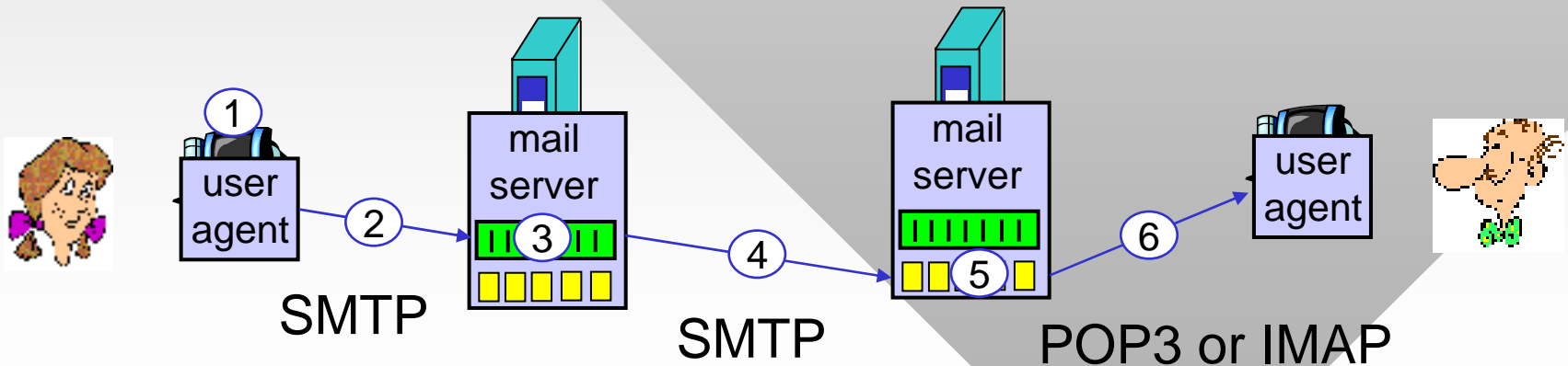
SMTP protocol

- Used by mail servers to send email messages
 - Client: sending mail server
 - Server: receiving mail server



Scenario: Alice Sends Message to Bob

- 1) Alice uses UA to compose message and “to” bob@someschool.edu
- 2) Alice’s UA sends message to her mail server
- 3) Message accepted and placed in outgoing queue
- 4) SMTP client sends message to Bob’s server
- 5) Bob’s mail server places the message in Bob’s mailbox
- 6) Bob invokes his user agent to read message



Electronic Mail: SMTP [RFC 821, 974, 1869, 2821]

- Original RFC in 1982, latest version in 2001
- Uses TCP to reliably transfer email message from client to server, **port 25**
- Three phases of transfer
 - SMTP handshake (greeting)
 - Transfer of messages
 - Closure
- Command/response interaction
 - **Commands:** ASCII text separated by `\r\n`
 - **Response:** status code and phrase (one line)

Sample SMTP Interaction

```
telnet mail.cs.tamu.edu 25
```

```
220 pine.cs.tamu.edu ESMTP Sendmail 8.12.9/8.12.9;  
Mon, 20 Sep 2004 15:52:57 -0500 (CDT)
```

```
HELO viper.cs.tamu.edu
```

```
250 pine.cs.tamu.edu Hello irl-viper.cs.tamu.edu  
[128.194.135.66], pleased to meet you
```

```
MAIL FROM:<dmitri@cs.tamu.edu>
```

```
250 2.1.0 <dmitri@cs.tamu.edu>... Sender ok
```

```
RCPT TO:<dmitri@cs.tamu.edu>
```

```
250 2.1.5 <dmitri@cs.tamu.edu>... Recipient ok
```

```
DATA
```

```
354 Enter mail, end with "." on a line by itself
```

```
Hello
```

```
Blah-blah-blah
```

```
.
```

```
250 2.0.0 i8KKqvvk027391 Message accepted for delivery
```

```
QUIT
```

```
221 2.0.0 pine.cs.tamu.edu closing connection
```

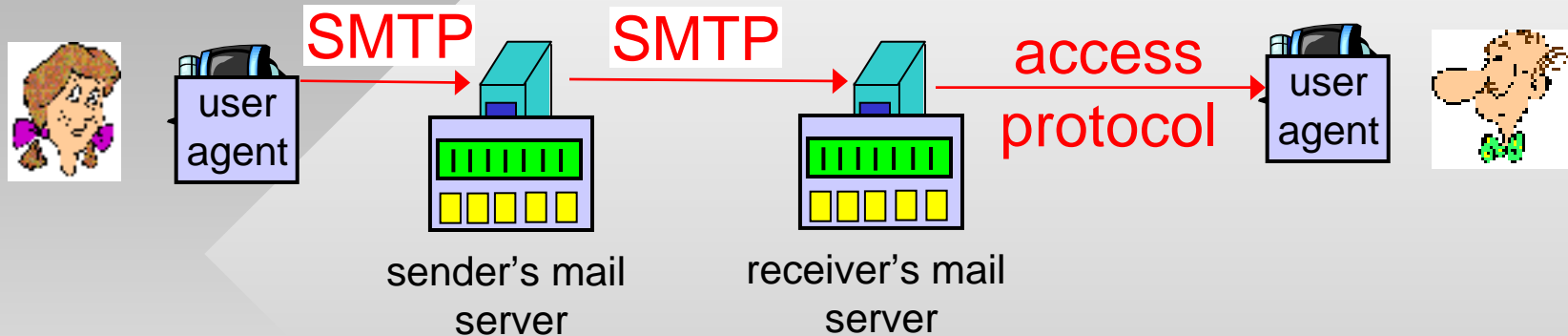
SMTP: Final Words

- SMTP uses non-pipelined persistent connections
- SMTP requires message (header & body) in 7-bit ASCII (codes < 128)
 - Additional restrictions may exist for the line length
- SMTP server uses `\r\n. \r\n` to determine the end of message
 - Solution: UA inserts a dot in front of all lines already starting with a dot

Comparison with HTTP:

- HTTP: pull, SMTP: push
- Both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own request/response msg
- SMTP: multiple objects sent in one msg separated by special tokens

Mail Access Protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
 - POP3: Post Office Protocol v3 [RFC 1939] – port 110
 - Authorization (agent <-->server) and download
 - IMAP: Internet Mail Access Protocol [RFC 1730] – port 143
 - More features (more complex)
 - Manipulation of stored messages on server
 - HTTP: Hotmail, Yahoo!, Gmail, etc.

POP3 Protocol

- Server responses

- +OK
- -ERR

Authorization phase

- Client commands:

- user: declare username
- pass: password

Transaction phase, client:

- list: list message #s
- retr: retrieve message by number
- dele: delete
- quit

```
telnet mail.cs.tamu.edu 110
+OK POP3 server ready
user bob
+OK
pass hungry
+OK user successfully logged on
```

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

POP3 (More) and IMAP

More about POP3

- Example used “download and delete” mode
- “Download-and-keep”
 - Multiple copies of message on different clients
- POP3 is **stateless** across sessions
 - Server assigns unique IDs to each message
 - Command UIDL lists IDs
 - UA determines new messages by remembering IDs of downloaded email

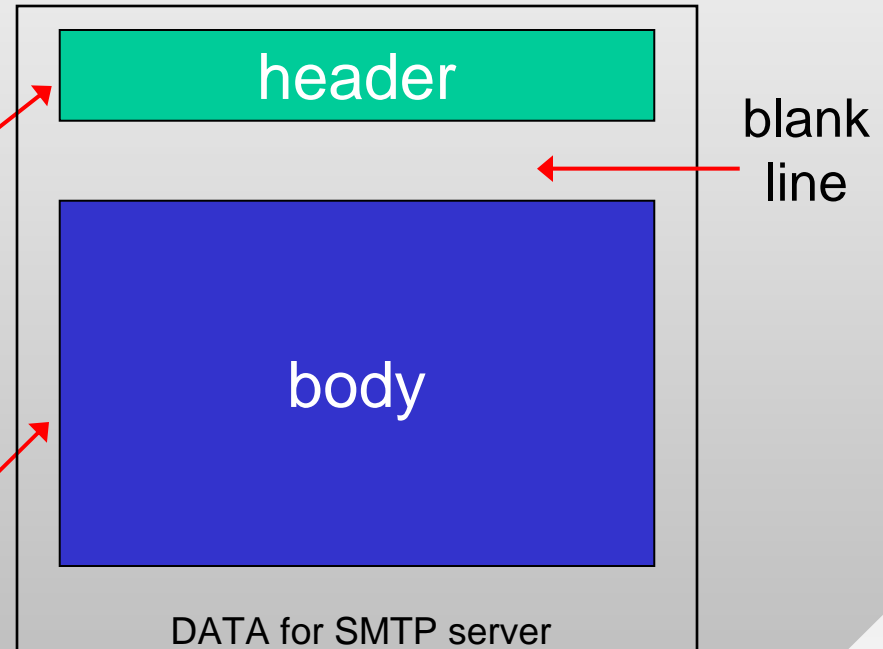
IMAP

- Keeps message status (folder membership, read/unread, flagged, replied to) at the server: **stateful** protocol
- More feature for the user, but more computationally expensive for the server

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”, 7-bit ASCII characters only



Message Format: MIME

- MIME: Multipurpose Internet Mail Extensions, RFCs 2045, 2056 (1996)
 - Additional lines in 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: vacation pics
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
RSAxNjAxOTQvTiAxNC9UIDkyNzg0OS9
IIFsgNT'cwIDQ2N10+Pg1lbnRvYmoNIC
```

6-bit encoding using A-Z, a-z, 0-9, +, /

Message Format: MIME 2

- Multiple objects separated by a specific boundary

```
Content-Type: multipart/mixed;  
  boundary="-----=_NextPart_000_0074_01C6DB4C.731EBEB0"
```

This is a multi-part message in MIME format.

```
-----=_NextPart_000_0074_01C6DB4C.731EBEB0  
Content-Type: text/plain;charset="iso-8859-1"  
Content-Transfer-Encoding: 7bit
```

Some text message here...

```
-----=_NextPart_000_0074_01C6DB4C.731EBEB0  
Content-Type: application/pdf;name="9-18-06.pdf"  
Content-Transfer-Encoding: base64
```