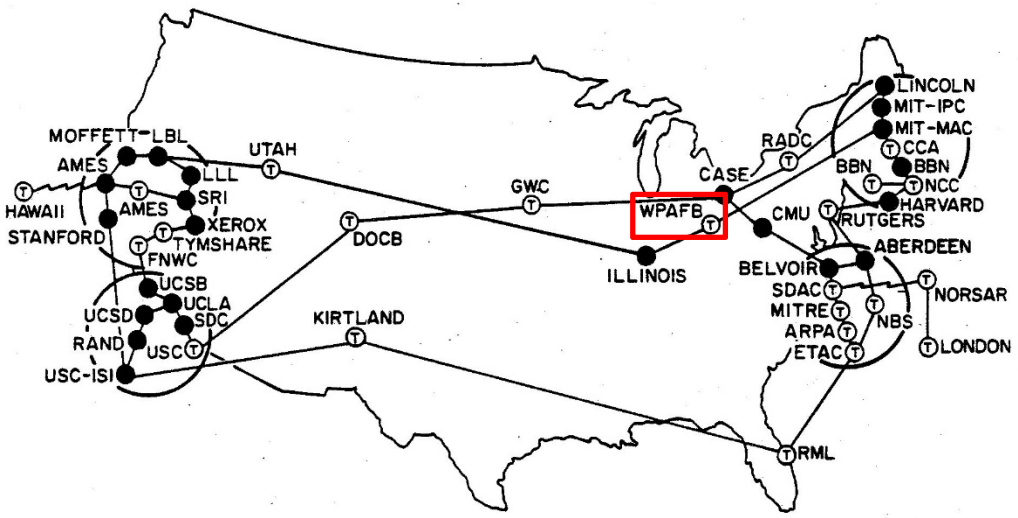
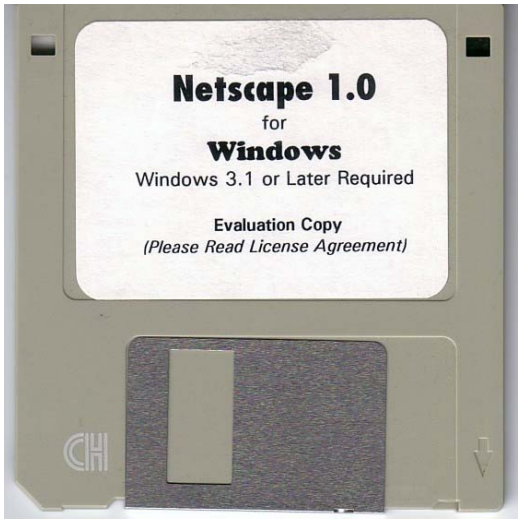


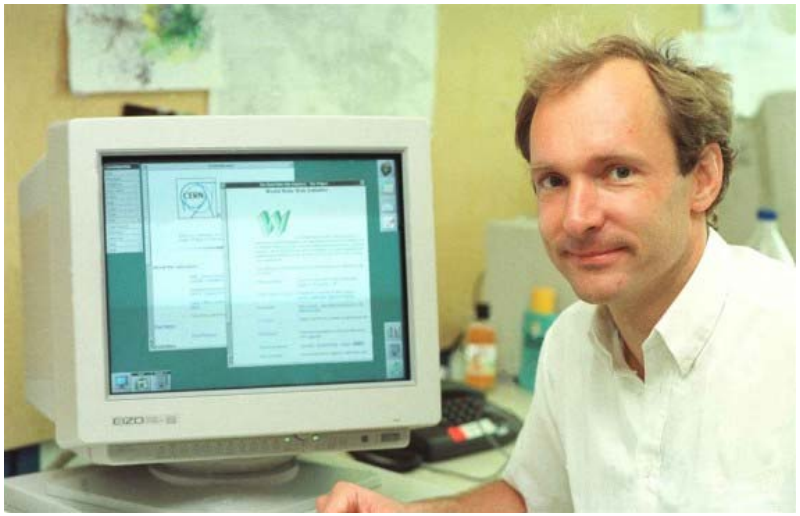
ARPANET 1974
http://bpastudio.csudh.edu/fac/lpress/history/arpamaps/

CSCE 560 Introduction to Computer Networking

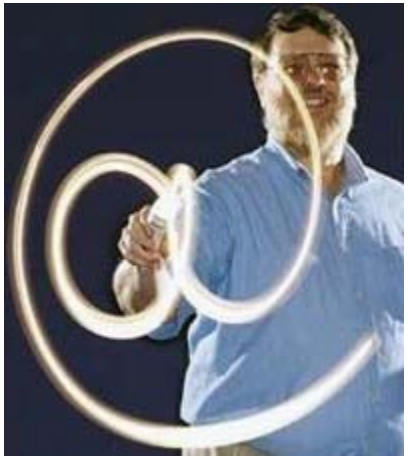
Dr. Barry Mullins
AFIT/ENG
Bldg 642, Room 209
255-3636 x7979



http://www.computerhistory.org/



Sir Tim Berners-Lee – "inventor" of the WWW
The world's first website, at CERN,
went online August 6, 1991



Ray Tomlinson
inventor of email 1971

Chapter 2: Application Layer

Our goals:

- Conceptual, implementation aspects of network **application** protocols
  - Transport-layer services
  - Client-server paradigm
  - Peer-to-peer paradigm
- Examine popular application-level protocols
  - HTTP
    - HyperText Transfer Protocol
  - FTP
    - File Transfer Protocol
  - SMTP / POP3 / IMAP
    - Simple Mail Transfer Protocol
    - Post Office Protocol
    - Internet Message Access Protocol
  - DNS
    - Domain Name System
- Create network applications
  - Socket API - application program interface

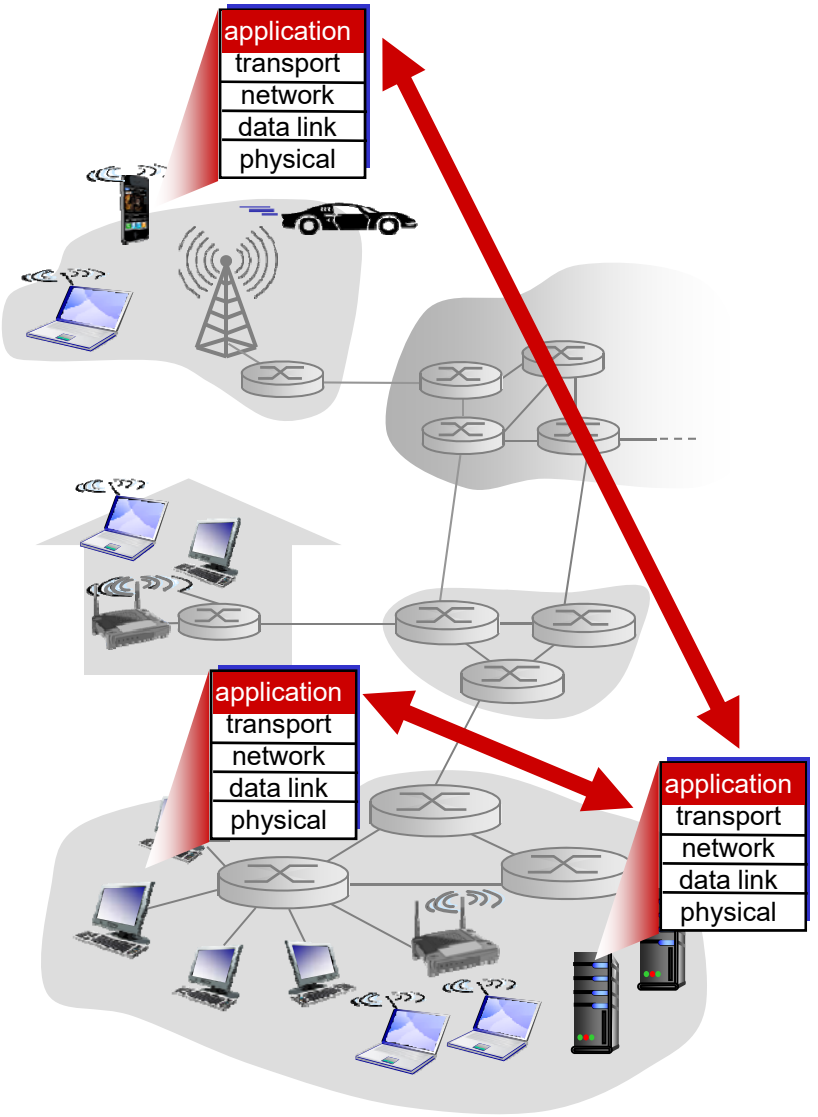
Creating a Network App

Write programs that

- Run on different end systems
- Communicate over a network
  - Web server software communicates with browser software

Very little software written for devices in network core

- Network core devices do not run user applications



Chapter 2: Application Layer

- 2.1 Principles of Network Applications
- 2.2 Web and HTTP
- FTP
- 2.3 Electronic Mail
  - SMTP, POP3, IMAP
- 2.4 DNS
- 2.5 P2P Applications
- 2.6 Video Streaming and Content Distribution Networks
- 2.7 Socket Programming: Creating Network Applications

Communicating Processes

Process: program (e.g., Chrome browser) running within a host

- Within same host, two processes communicate using interprocess communication (IPC) (defined by OS)
- Processes in different hosts communicate by exchanging messages

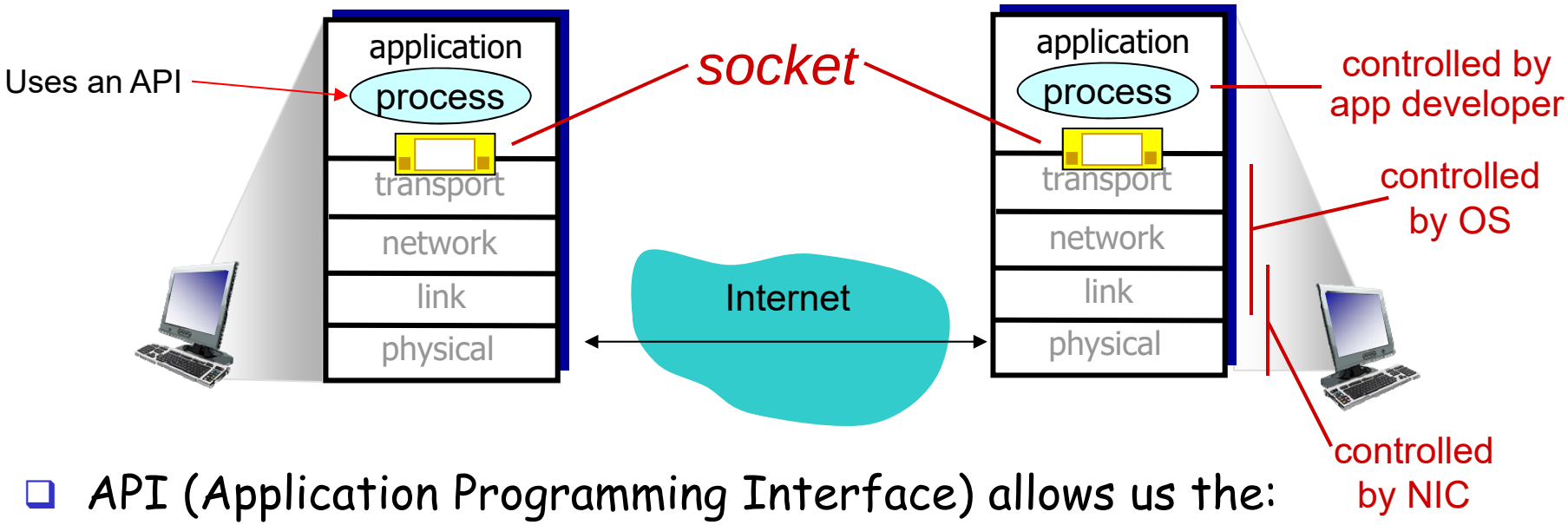
Client process: process that initiates communication

Server process: process that waits to be contacted

Fun fact: Applications with P2P architectures have client processes AND server processes

Sockets

- Process sends/receives messages to/from its socket
- Socket analogous to a door
  - Sending process shoves message out door
  - Sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



- API (Application Programming Interface) allows us the:
  1. choice of transport protocol
  2. ability to select a few parameters

Addressing Processes

- For a process to receive messages, it must have an identifier
- A host has a unique 32-bit IP address

Q: Does the IP address of the host on which the process runs suffice for identifying the process?

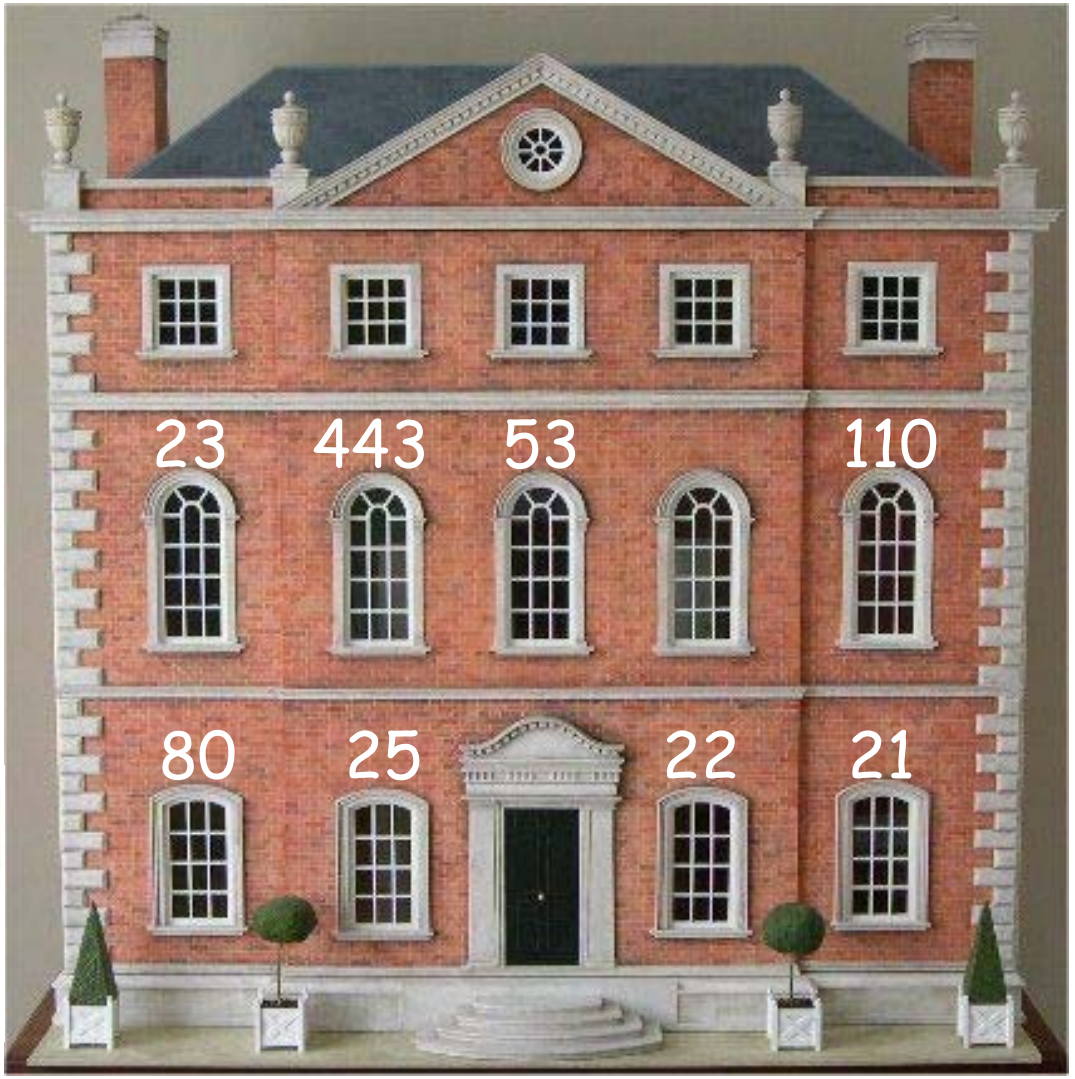
A: No, *many* processes can be running on same host

- Identifier includes both the IP address and port numbers associated with the process on the host
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- To send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - Port number: 80
- More on this later

Preview – IP and Port Numbers

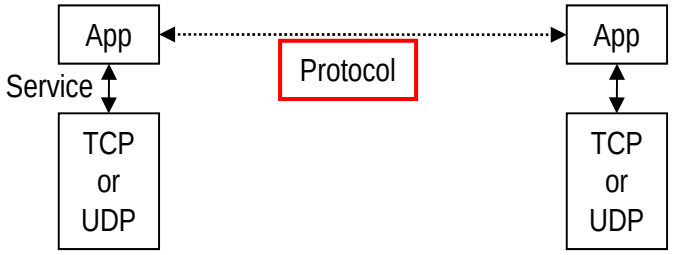
IP address → server : Postal address → house

Port number → application (web, email) : Window number → person



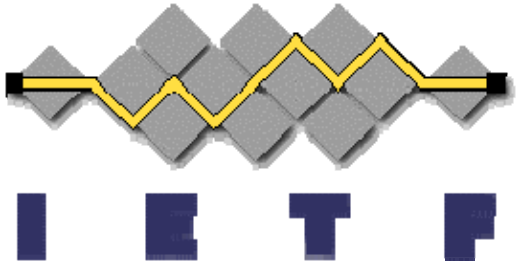
Application Layer Protocol Defines:

- Types of messages exchanged
  - Request & response messages
  - Control messages
- Message syntax
  - What fields are in messages & how fields are delineated
- Message semantics
  - Meaning of information in fields
- Rules for when and how processes send & respond to messages



Public-domain protocols:

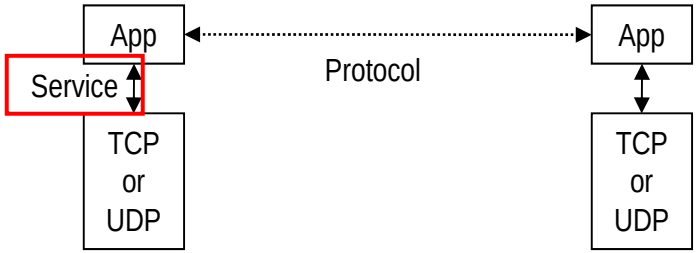
- Defined in RFCs
- Facilitates interoperability
- e.g., HTTP, SMTP
- Internet Engineering Task Force
  - www.ietf.org/rfc/
  - "The goal of the IETF is to make the Internet work better."



Proprietary protocols:

- e.g., Skype

What Transport Service Does An App Need?



Data Integrity

- Some apps can tolerate some loss
  - Audio/Video
  - Interactive games
- Other apps require 100% reliable data transfer
  - File transfer
  - Web transactions

Timing

- Some apps require low delay to be "effective"
  - Internet telephony
  - Interactive games

Throughput

- Some apps require a minimum amount of throughput to be "effective"
  - Multimedia
- Other apps make use of whatever bandwidth they get → "elastic apps"
  - File transfer
  - Web traffic
  - Email

Internet Transport Protocol Services

TCP service:

- *Connection-oriented:* setup required between client and server processes
- *Reliable transport* between sending and receiving process
- *Flow control:* sender won't overwhelm receiver
- *Congestion control:* throttle sender when network overloaded
- *Does not provide:*
  - Timing
  - Minimum bandwidth guarantees

UDP service:

- Unreliable data transfer between sending and receiving process
- Does not provide:
  - Connection setup
  - Reliability
  - Flow control
  - Congestion control
  - Timing
  - Bandwidth guarantee

Q: Why bother?
Why is there a UDP?

Internet Apps: Application, Transport Protocols

Application	Application layer protocol	Underlying transport protocol
Email	SMTP [RFC 2821]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
File transfer	FTP [RFC 959]	TCP
Streaming multimedia	HTTP (e.g., YouTube) RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, Proprietary (e.g., Skype)	TCP or UDP
Domain Name System	DNS [RFC 1035 + many more]	UDP primarily

RTP – Real-time Transport Protocol

SIP – Session Initiation Protocol

Chapter 2: Application Layer

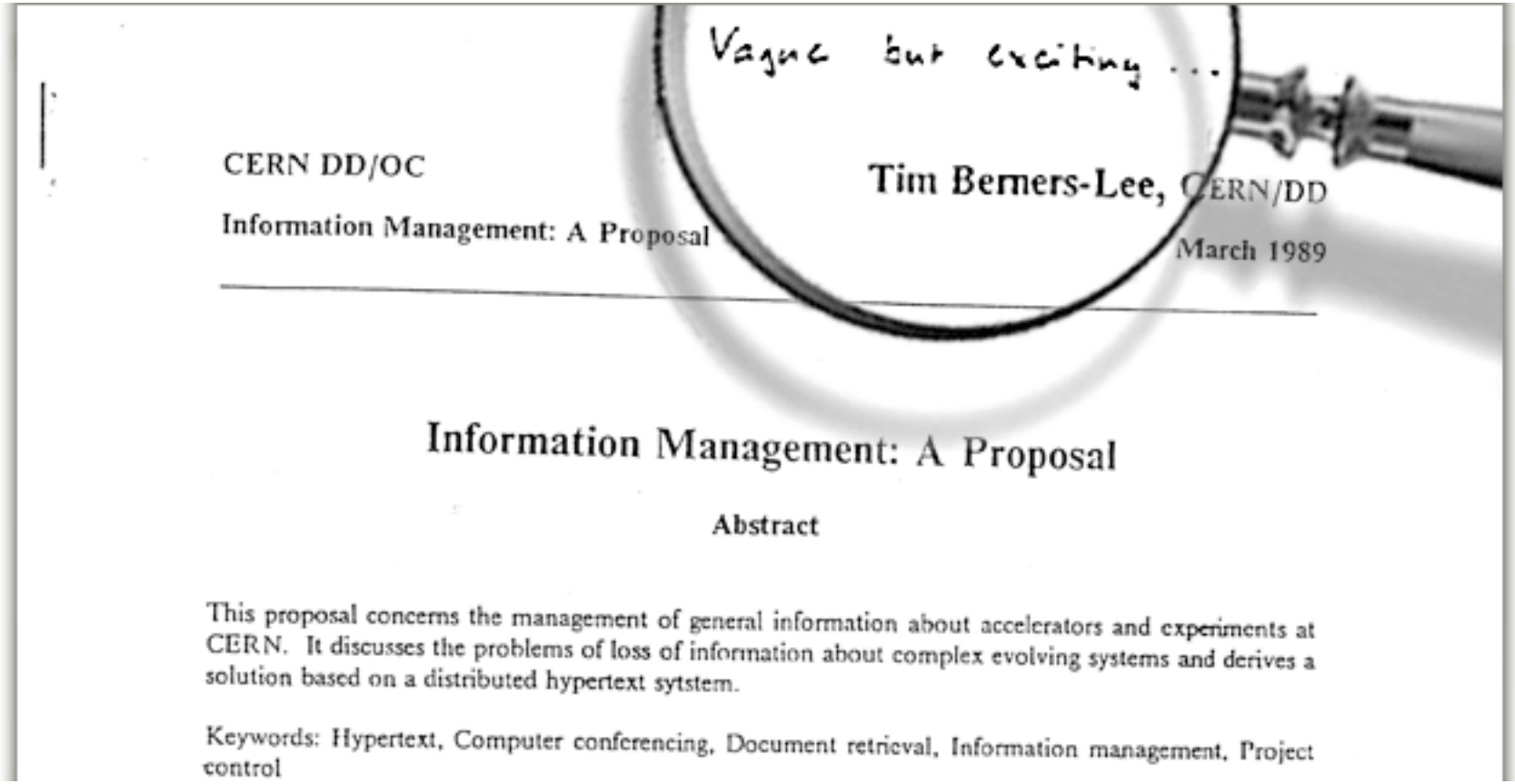
- 2.1 Principles of Network Applications
- 2.2 Web and HTTP
- FTP
- 2.3 Electronic Mail
  - SMTP, POP3, IMAP
- 2.4 DNS
- 2.5 P2P Applications
- 2.6 Video Streaming and Content Distribution Networks
- 2.7 Socket Programming: Creating Network Applications

Ever have the feeling your “job title” understates your role?

Sir Timothy John "Tim" Berners-Lee is best known as the inventor of the World Wide Web

"Vague but exciting …"

In March 1989, Tim Berners-Lee submitted a proposal for an information management system to his boss, Mike Sendall. "*Vague, but exciting*", were the words that Sendall wrote on the proposal, allowing Berners-Lee to continue.

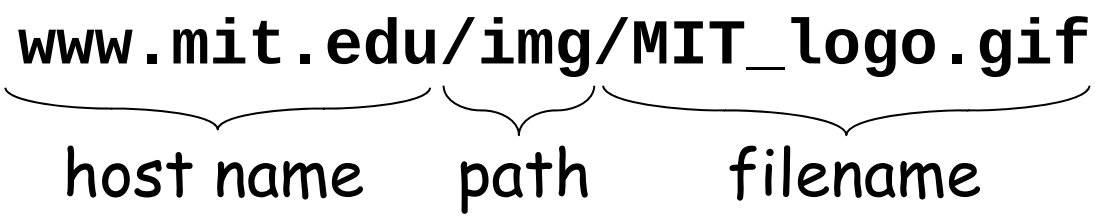


http://info.cern.ch/Proposal.html

Web and HTTP

First some jargon

- Web page consists of objects
  - Object can be HTML file, JPEG image, Java applet, audio file, …
  - An object can be ANY file
- Web page consists of base HTML file which includes several referenced objects
  - Each object is addressable by a URL
    - Uniform Resource Locator
- Example URL:



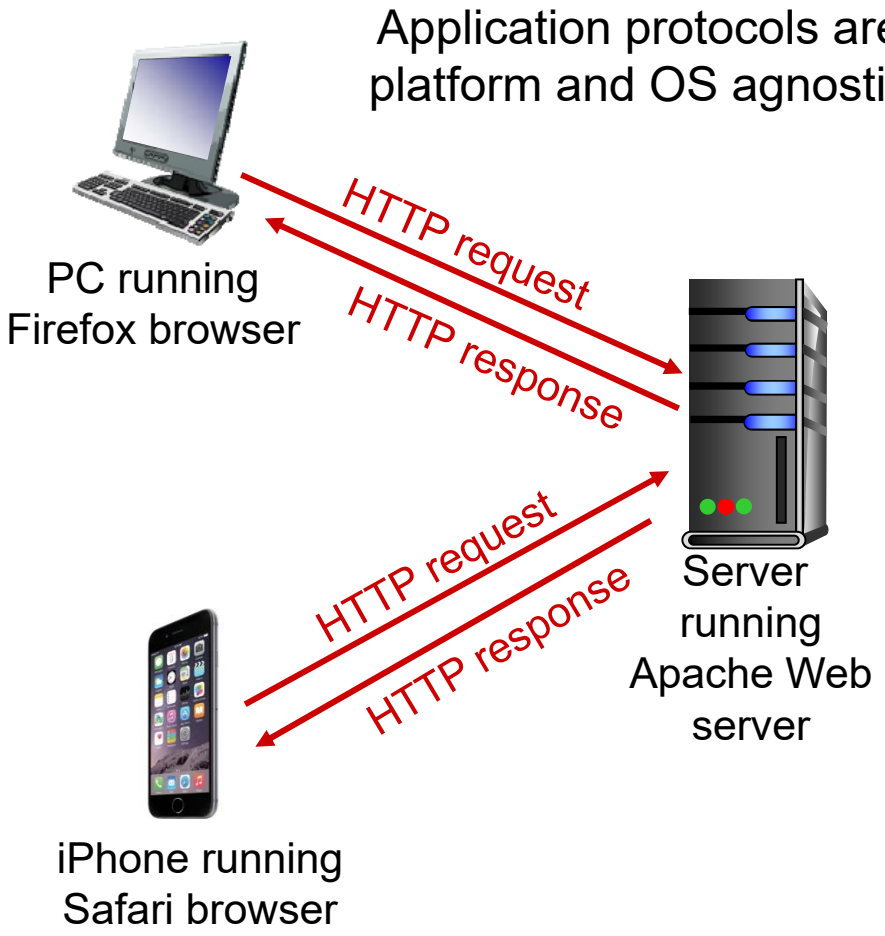
HTTP: Hypertext Transfer Protocol

- Web's application layer protocol

Client/Server model

- *Client:* browser that requests, receives, and "displays" Web objects
- *Server:* Web server sends objects in response to requests

- HTTP documented in
  - HTTP 1.0: RFC 1945
  - HTTP 1.1: RFC 2616

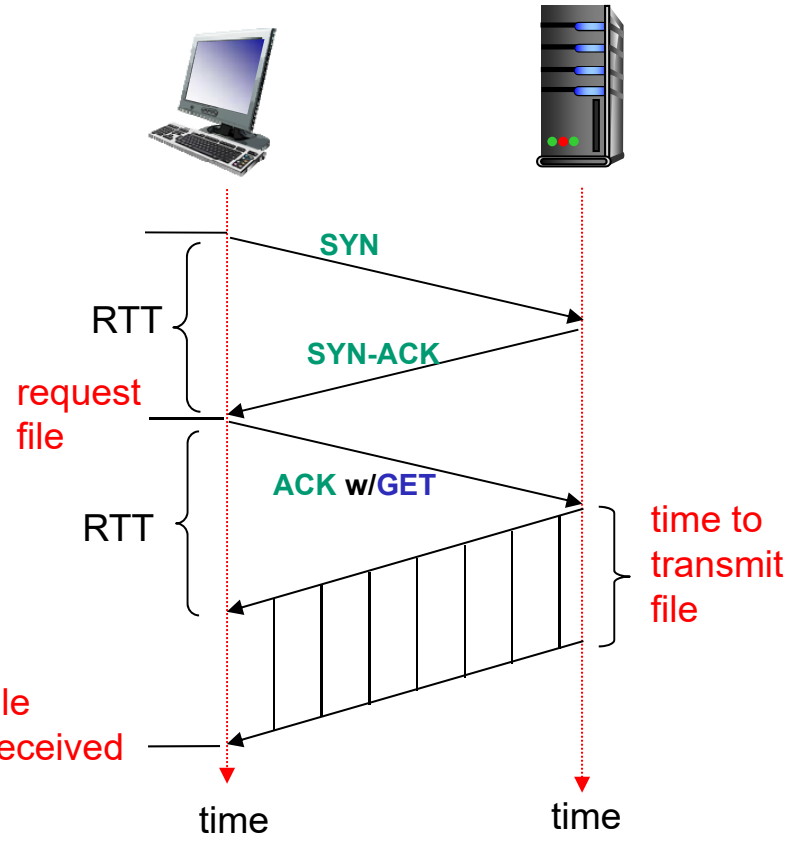


HTTP Overview

Uses TCP:

- Client initiates TCP connection (creates socket) to server on port 80
  - Sends a "SYN" packet
- Server accepts TCP connection from client
  - Sends a "SYN-ACK" packet
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed (not shown on figure)

RTT: Time required for a small packet to travel from client to server and back



File transmission time:

Total = 2RTT + file transmit time

HTTP Connections

Non-persistent HTTP

- At most only **one** object is sent over a TCP connection
- HTTP/1.0 uses non-persistent HTTP

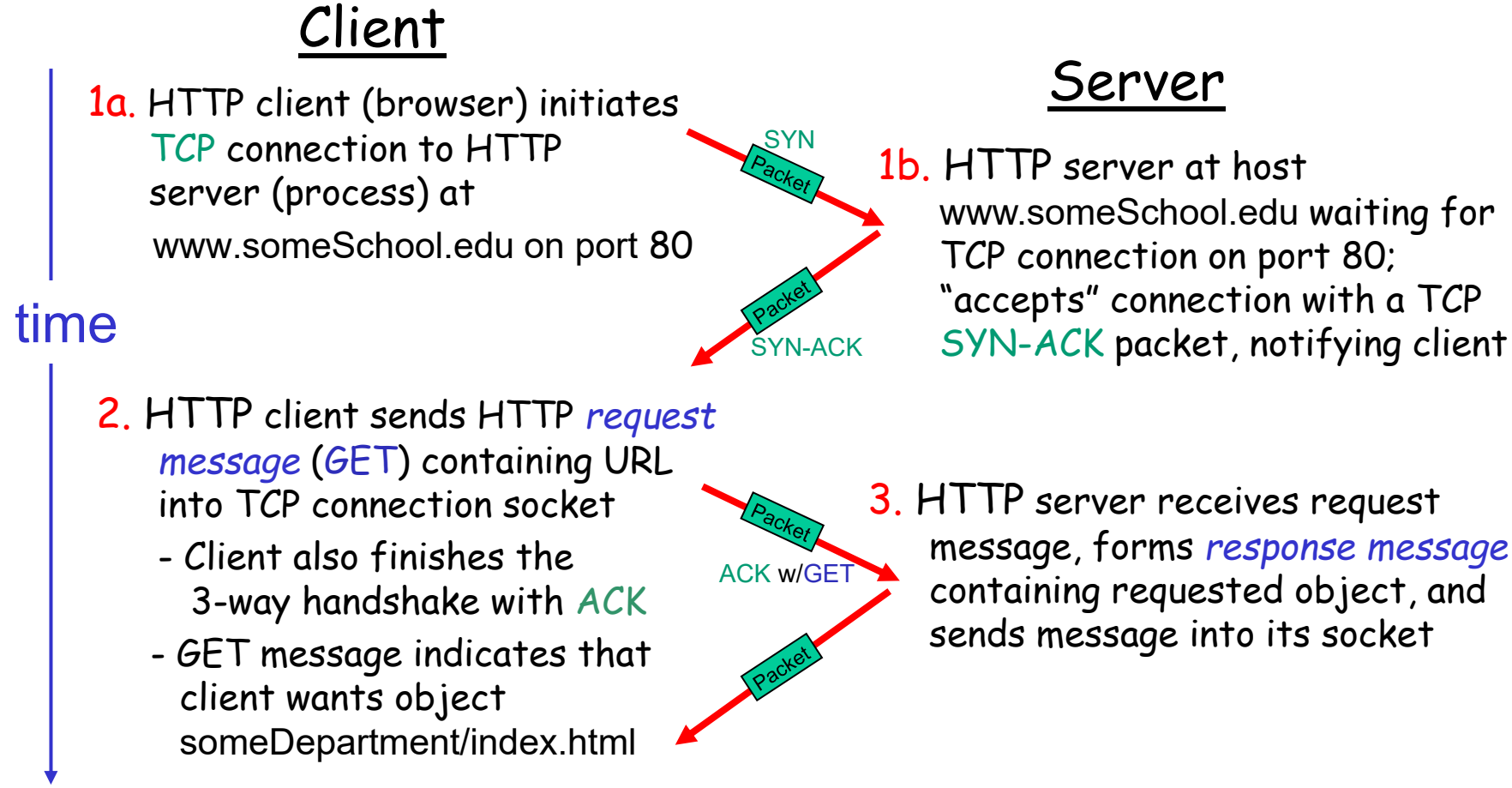
Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server
- HTTP/1.1 uses persistent connections by default

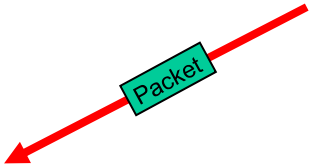
Non-persistent HTTP

Suppose user enters URL
`www.someSchool.edu/someDepartment/index.html`

(contains base html and references to 10 jpeg images)



Non-persistent HTTP (cont.)



4. HTTP server starts to close TCP connection

5. HTTP client receives response message containing html file, displays html
   - Client parses html file, finds 10 referenced jpeg objects

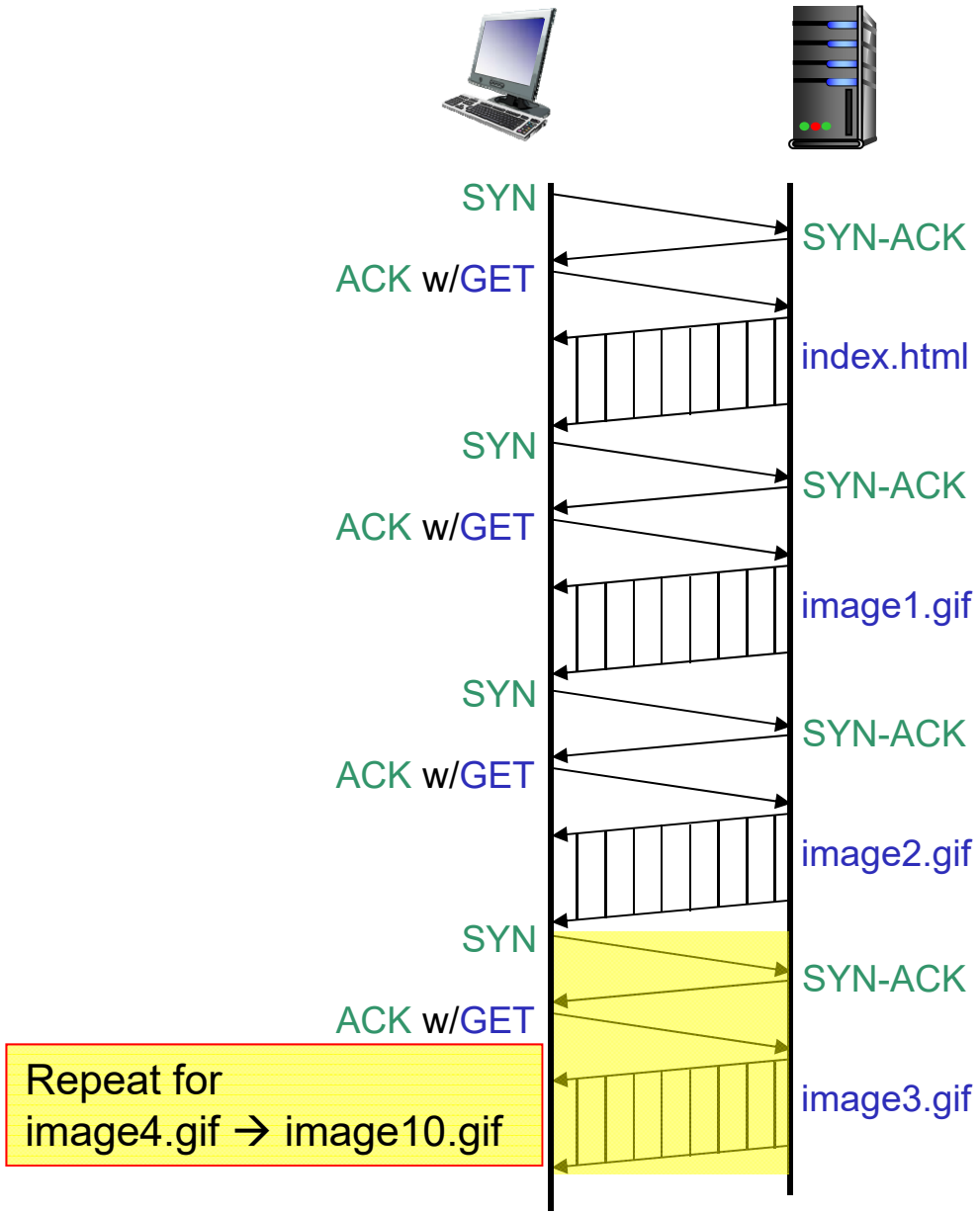
6. Steps 1-5 repeated for **each** of 10 jpeg objects

time

Visualizing Non-persistent Connections

Non-persistent HTTP issues:

- Requires 2 RTTs per object
- OS overhead for *each* TCP connection
  - TCP buffers and variables



Persistent HTTP

Persistent HTTP

- Server leaves TCP connection open after sending response
- Subsequent HTTP messages between same client/server sent over open connection

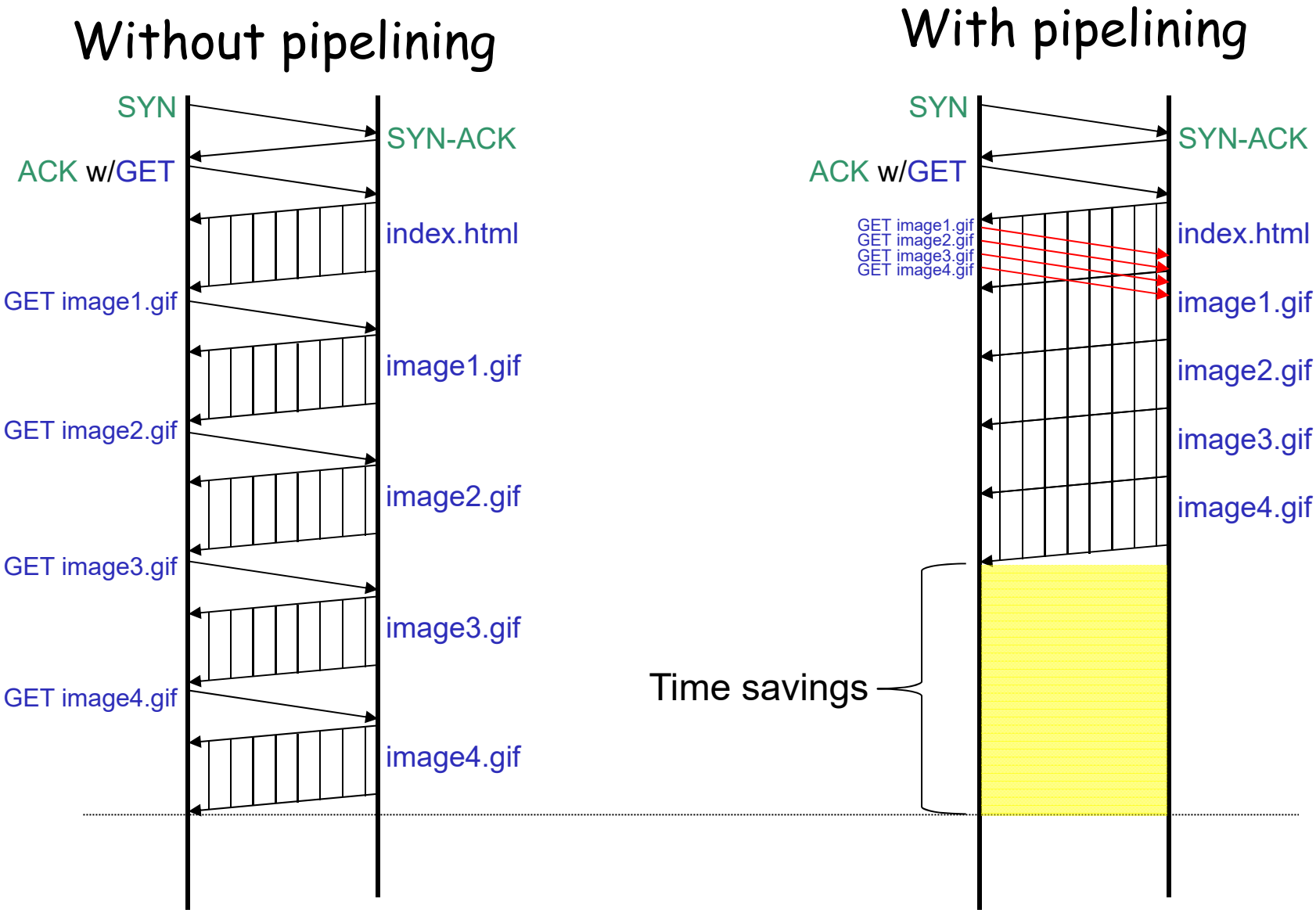
Persistent ***without*** pipelining:

- Client issues new request only when previous response has been received
- One RTT for each referenced object

Persistent ***with*** pipelining:

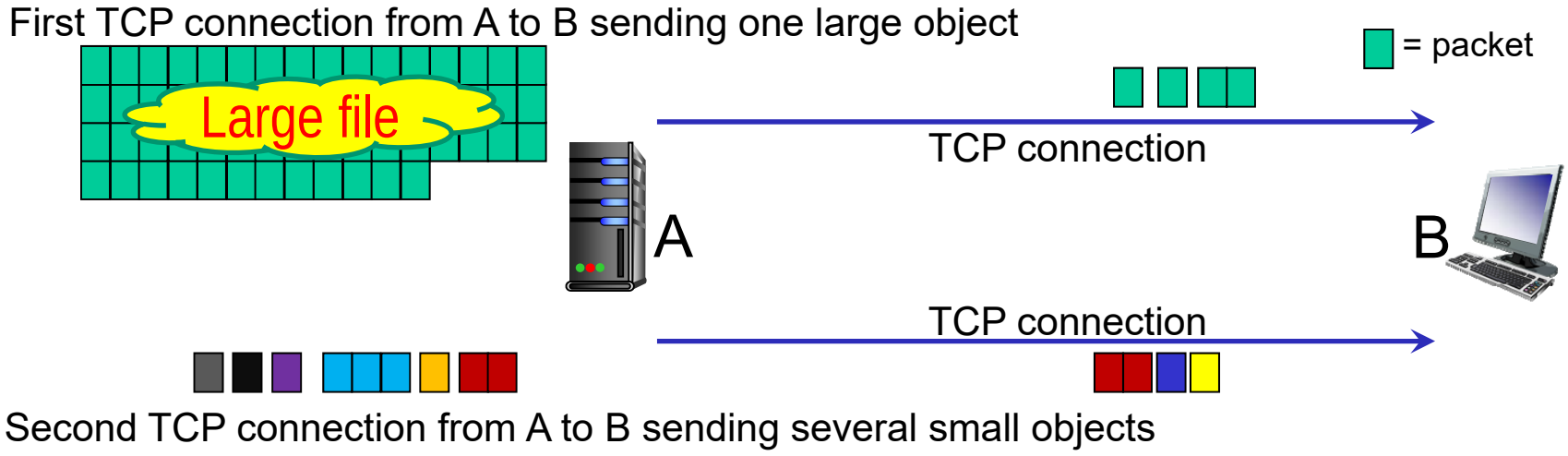
- Default in HTTP/1.1
- Client sends requests as soon as it encounters a referenced object
- Specification requires that the server respond to the requests in the order in which they are received
  - Must use FIFO
  - As a result, one large object can back up many smaller objects.
    - Known as *head of line blocking*

Visualizing Pipelining – Persistent Connections



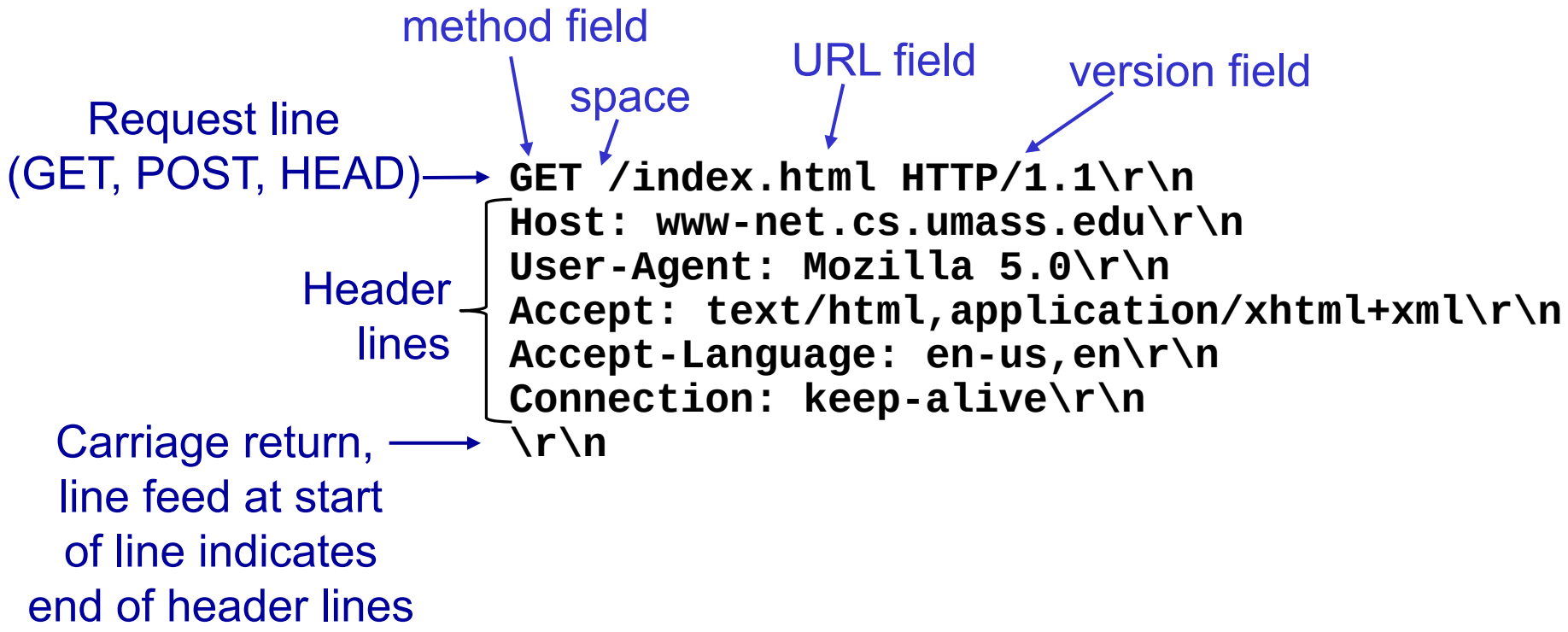
Multiple Parallel TCP Connections

- As an alternative to, or in addition to, pipelining, many clients can open more than one TCP session (connection) to the server and then make requests through each connection
- This mitigates head of line blocking as shown below
- The HTTP 1.1 standard specifies that each client should open no more than two connections to any one server
  - In practice, this is basically ignored



HTTP Request Message

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



User agent: a client application implementing a network protocol

HTTP `User-Agent` Trickery

- Since 1990s, most major browsers use
  - **`User-Agent: Mozilla/5.0`**
  - Code name for the now-defunct Netscape Web browser
  - Netscape was viewed as the cutting-edge browser of its day

- Browsers pretended to be Netscape to trick Web servers into sending it the most sophisticated version of their site even though most of them aren't derived from Mozilla software
  - Internet Explorer
  - Chrome
  - Firefox
  - Torch
- Microsoft Edge → **`User-Agent: Microsoft-CryptoAPI/10.0`**

http://www.washingtonpost.com/blogs/the-switch/wp/2013/09/23/this-hacker-might-seem-shady-but-throwing-him-in-jail-is-bad-for-everyone/

Uploading Form Input – Get Method

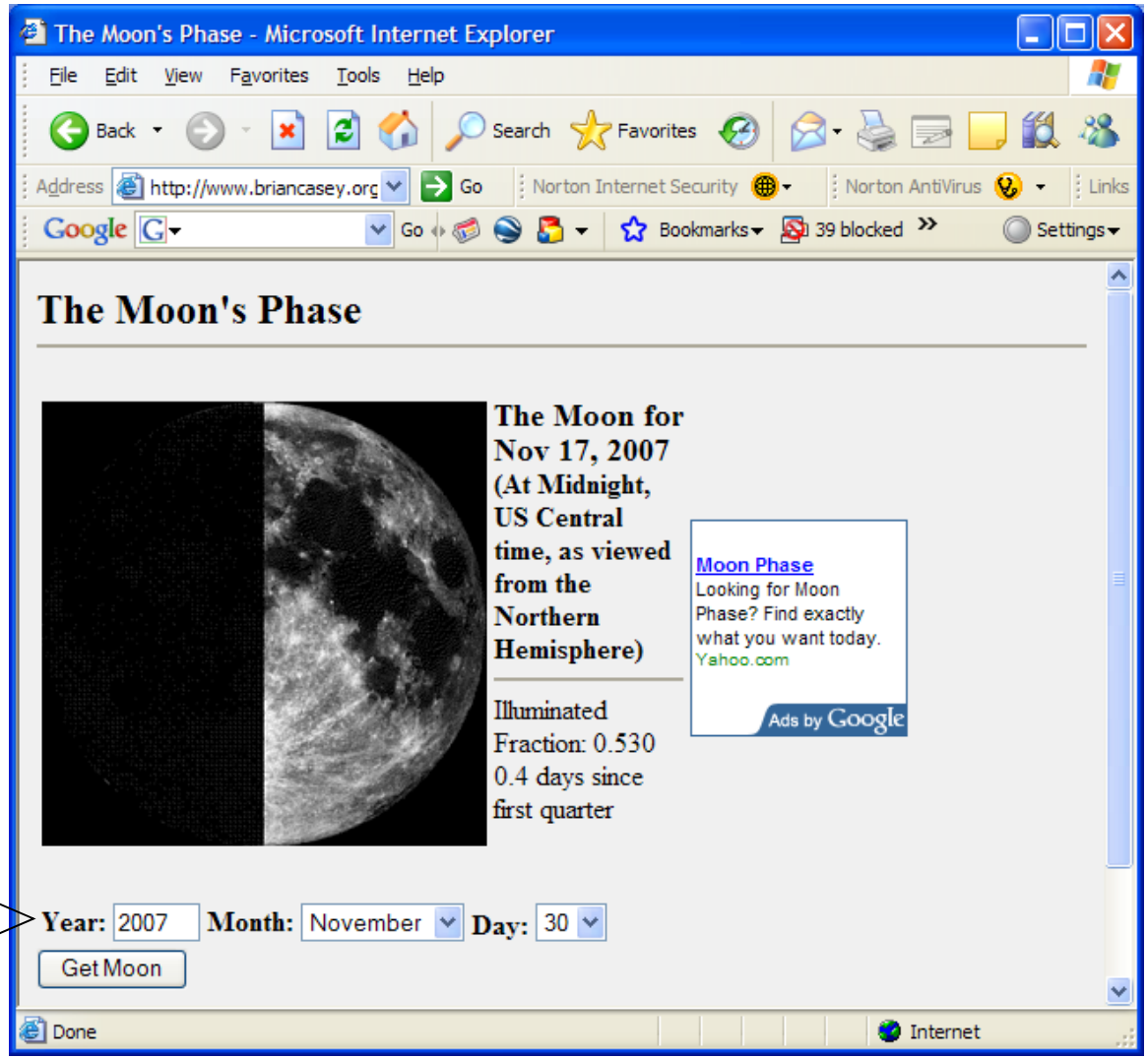
- Input is uploaded in URL field of request line:

```
GET http://www.mapquest.com/maps?address=2950+Hobson+Way&zipcode=45433 HTTP/1.1
<<snipped some header information>>
Host: www.mapquest.com
<CRLF>
```

Uploading Form Input – Post Method

- A web page may include form input
  - Address to locate
- Input is uploaded to server in body of request
  - aka entity body

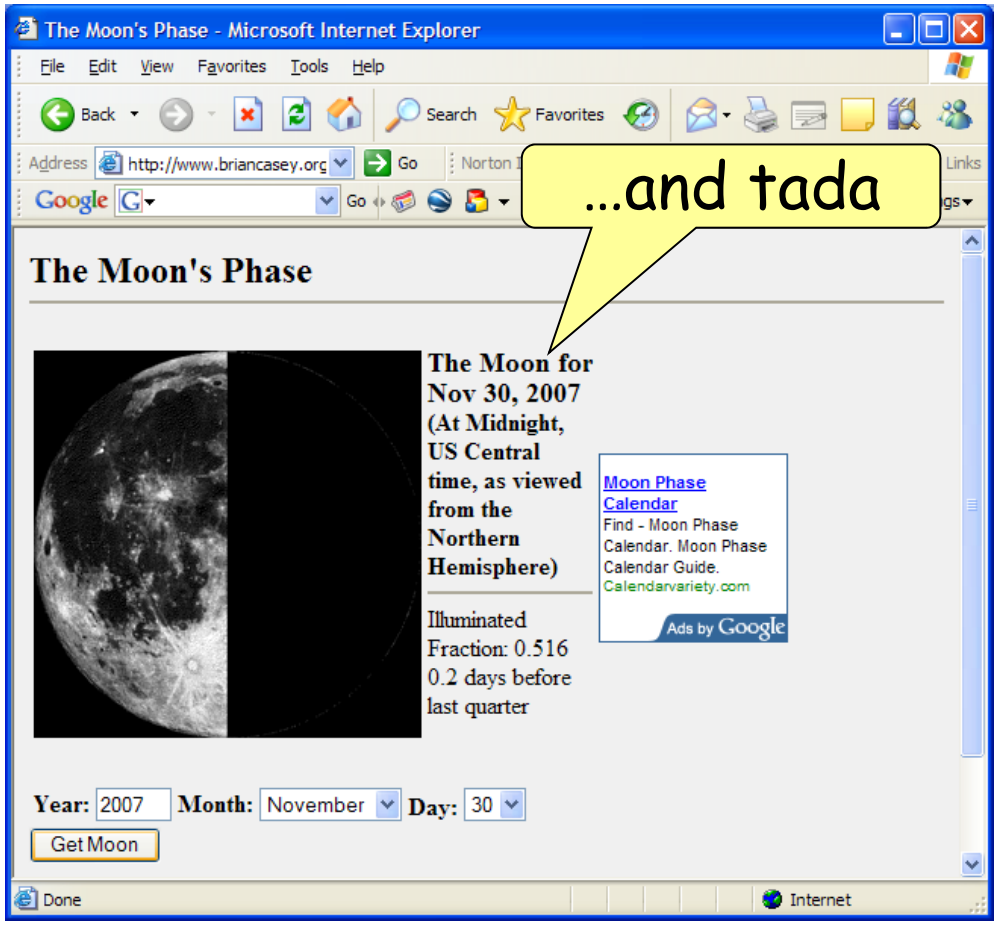
- Currently viewing 17 Nov 07
- Want to see 30 Nov 07 …



Uploading Form Input - Post Method

```
POST http://www.briancasey.org/artifacts/astro/moon.cgi HTTP/1.1
Content-Length: 25
Host: www.briancasey.org
<<snipped some header information>>
<CRLF>
year=2007&month=11&day=30
```

25 characters



Method Types

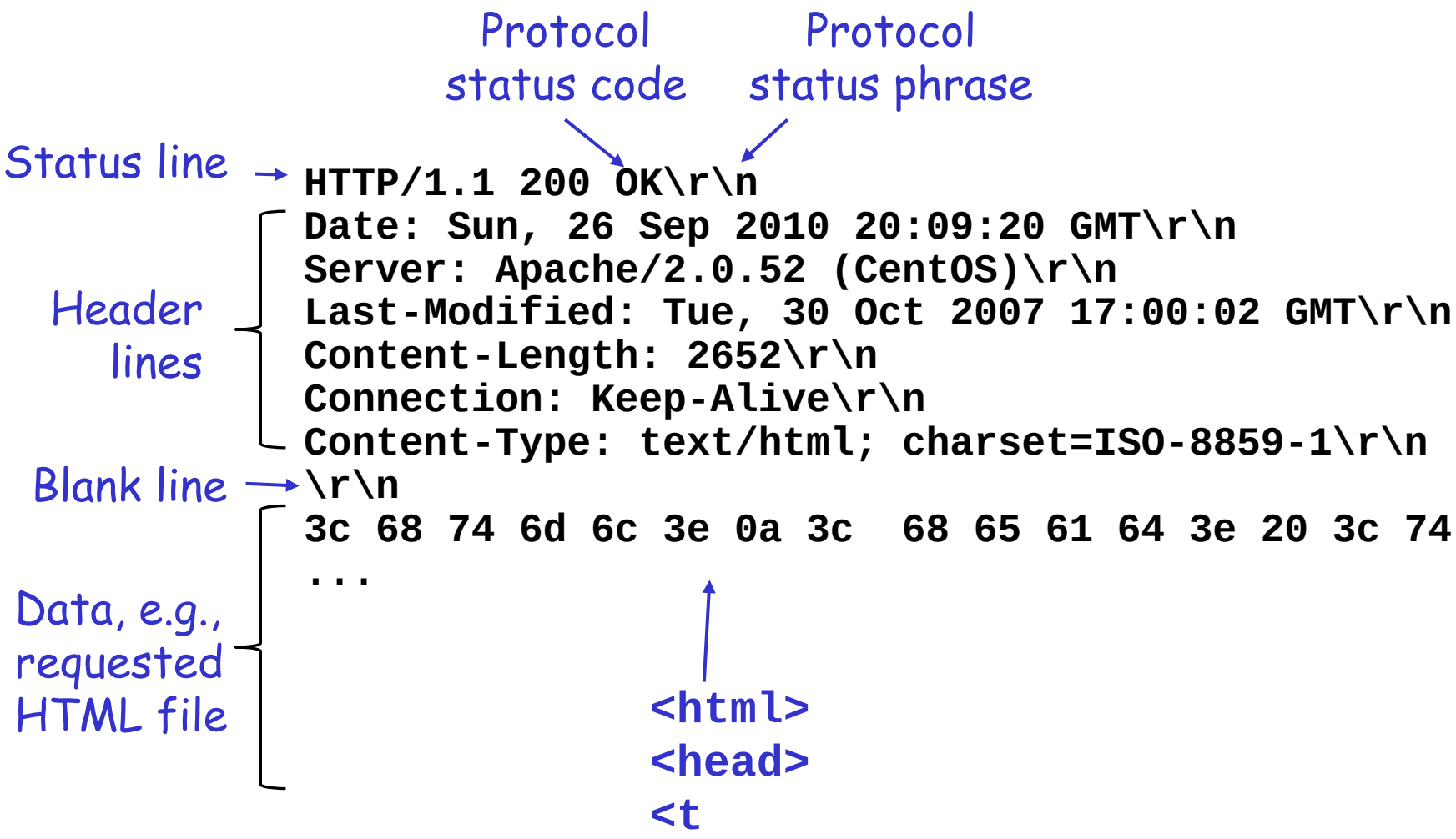
HTTP/1.0

- GET
- POST
- HEAD
  - Asks server to leave requested object out of response
  - Used for debugging

HTTP/1.1

- GET, POST, HEAD
- PUT
  - Uploads file in entity body to path specified in URL field
- DELETE
  - Deletes file specified in the URL field

HTTP Response Message



HTTP Response Status Codes

1xx Intermediate Status
2xx Successful Response
3xx Redirects
4xx Request Errors
5xx Server Errors

In first line of server->client response message

A few sample codes:

200 OK

- Request succeeded, requested object later in this message

301 Moved Permanently

- Requested object moved, new location specified later in this message (Location:)

400 Bad Request

- Request message not understood by server

505 HTTP Version Not Supported

More exhaustive list
http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

HTTP Response Status Codes

1xx Intermediate Status
2xx Successful Response
3xx Redirects
4xx Request Errors
5xx Server Errors

404 Not Found

- Requested document not found on this server

Flavor not found?!?!

HTTP Response Status Codes

404 Not Found

2018 Quidditch
National Championship

Manual HTTP (GET)

Open a command shell and type (or paste) the RED text

Notice the server does not echo the characters

```
> telnet gaia.cs.umass.edu 80
Trying 128.119.245.12...
Connected to gaia.cs.umass.edu.
Escape character is '^]'.
```

Start telnet session to Web service (port 80) on gaia.cs.umass.edu

May not see this notice

Request line

```
GET http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html HTTP/1.0  [ENTER]
Connection: keep-alive [ENTER]
[ENTER]
```

Header line

Carriage return, line feed at start of line indicates end of header lines

Empty entity body

```
HTTP/1.1 200 OK
```

Status line

```
Date: Mon, 12 Oct 2015 13:31:14 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.1e-fips PHP/5.4.16 mod_perl/2.0.9dev Perl/v5.16.3
Last-Modified: Mon, 12 Oct 2015 05:59:01 GMT
ETag: "173-521e206aa287e"
Accept-Ranges: bytes
Content-Length: 371
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

Response Header

Blank line – separates header from data

Data (Webpage)

```
<html>

Congratulations again!  Now you've downloaded the file lab2-2.html. <br>
This file's last modification date will not change.  <p>
Thus  if you download this multiple times on your browser, a complete copy <br>
will only be sent once by the server due to the inclusion of the IN-MODIFIED-SINCE<br>
field in your browser's HTTP GET request to the server.

</html>
```

Manual HTTP (POST)

Notice the server does not echo the characters

Open a command shell and type (or paste) the RED text

```
> telnet www.briancasey.org 80
Trying 192.155.194.19...
Connected to www.briancasey.org.
Escape character is '^]'.
```

Start telnet session to Web service (port 80) on www.briancasey.org

May not see this notice

Request line

```
POST http://www.briancasey.org/artifacts/astro/moon.cgi HTTP/1.1  [ENTER]
Content-Length: 25 [ENTER]
Host: www.briancasey.org [ENTER]
[ENTER]
```

Header lines

Carriage return, line feed at start of line indicates end of header lines

```
year=2007&month=11&day=30
```

Entity body

```
HTTP/1.1 200 OK
```

Status line

```
Date: Mon, 12 Oct 2015 13:51:07 GMT
Server: Apache/2.2.15 (CentOS)
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=ISO-8859-1
```

Response Header

Blank line – separates header from data

```
<html>
<head> <title> The Moon's Phase </title> </head>
<body bgcolor="#F0F0F0">

<<snipped>>
```

Data (Webpage)

Web Server Scripting

- A URL may refer to a static web page or a server-side script
  - Script is just a program that is run in response to a HTTP request
- Server-side scripts create "dynamic" web pages based on the input

- Common Gateway Interface (CGI)
  - Standard argument passing convention between web server and clients
- CGI scripts may be written in any language
  - PHP, Python, Perl, Ruby, sh, csh, Java
- CGI scripts are commonly used to produce responses to Web page form input from client browsers

- **`http://www.briancasey.org/artifacts/astro/moon.cgi`**

Client Side Web Page Scripts/Programs

- Web pages may also contain scripts or programs within the HTML code to be run on the client

- Unlike server scripts, web page scripts and programs run on the browser machine's processor, not on the server's processor

- Examples:
  - Javascript
  - VBScript
  - Java applets

- **http://media.pearsoncmg.com/aw/aw_kurose_network_2/applets/message/messagesegmentation.html**

HTTP is "Stateless"

- Server maintains no information about past client requests

Protocols that maintain "state" are complex!

- Past history (state) must be maintained
- If server/client crashes, their views of "state" may be inconsistent and must be reconciled

How to keep "state"

- Protocol endpoints
  - Maintain state at sender/receiver over multiple transactions

- **Cookies**: HTTP headers carry state

Cookies

Most major Web sites use cookies

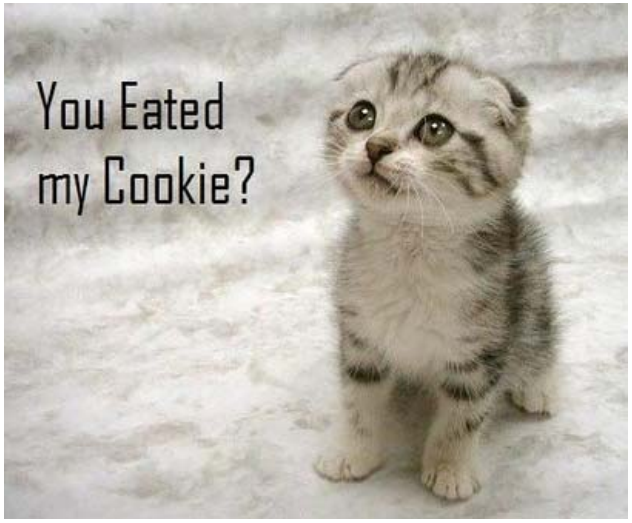
What cookies can bring:

- Authorization
- Shopping carts
- Recommendations
- User session state
  - Web email

aside

Cookies and privacy:

- Cookies permit sites to learn a lot about you
- You may supply name and email to sites
- Search engines use redirection & cookies to learn even more



Providing State Information: Cookies

Four components:

1) cookie header line of HTTP *response* message
2) cookie header line in HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) back-end database at Web site

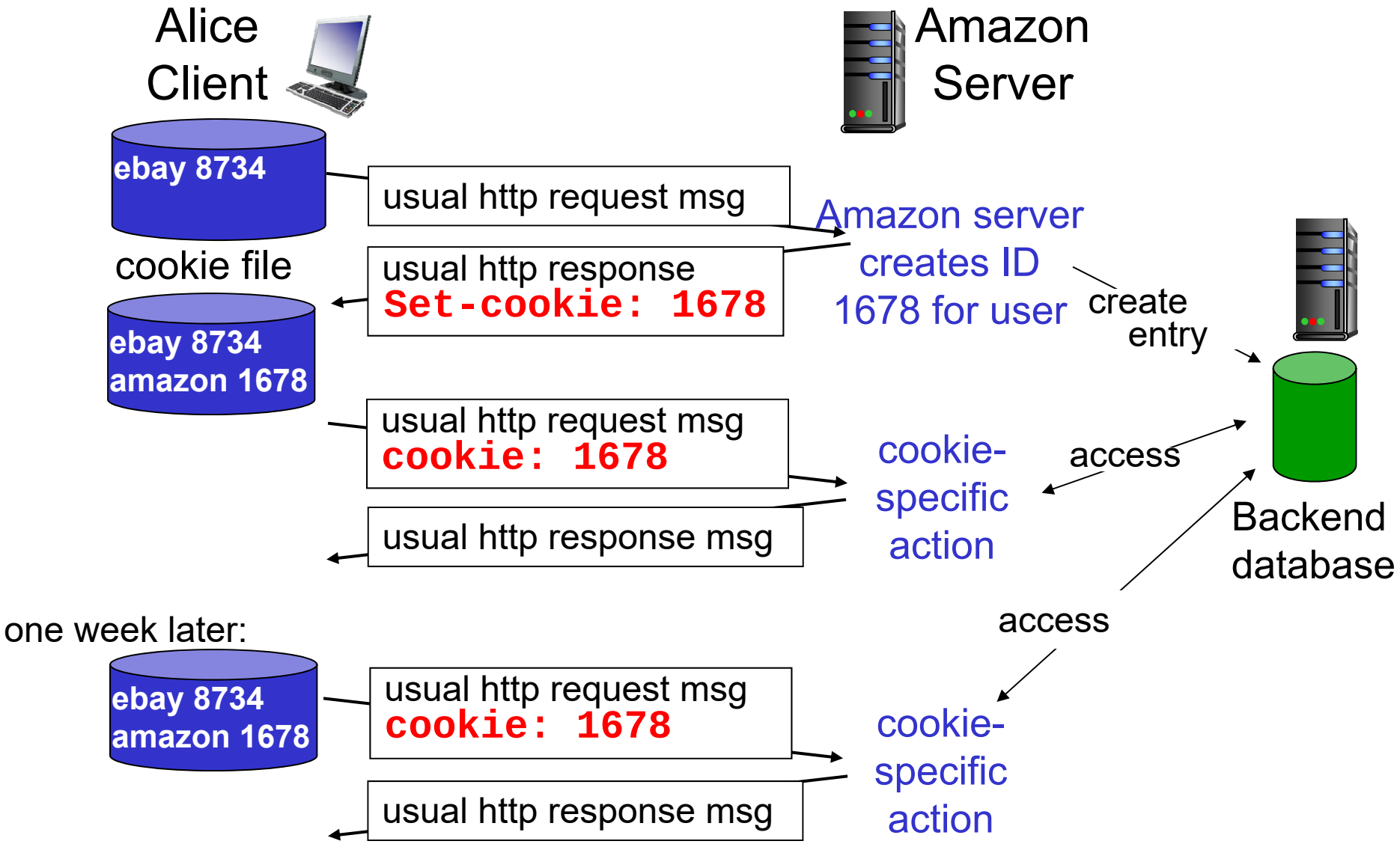
Example:

- Alice accesses Internet always from same PC
- She visits a specific e-commerce site for first time
- When initial HTTP requests arrives at site, site creates
  - a unique ID and
  - an entry in backend database for ID

"A cookie is a well-known computer science term that is used when describing an opaque piece of data held by an intermediary."
http://cookiecentral.com/faq/#1.2

Windows 10 cookies folder can be displayed using
Press the Windows key and R together. Type shell:cookies and click OK.

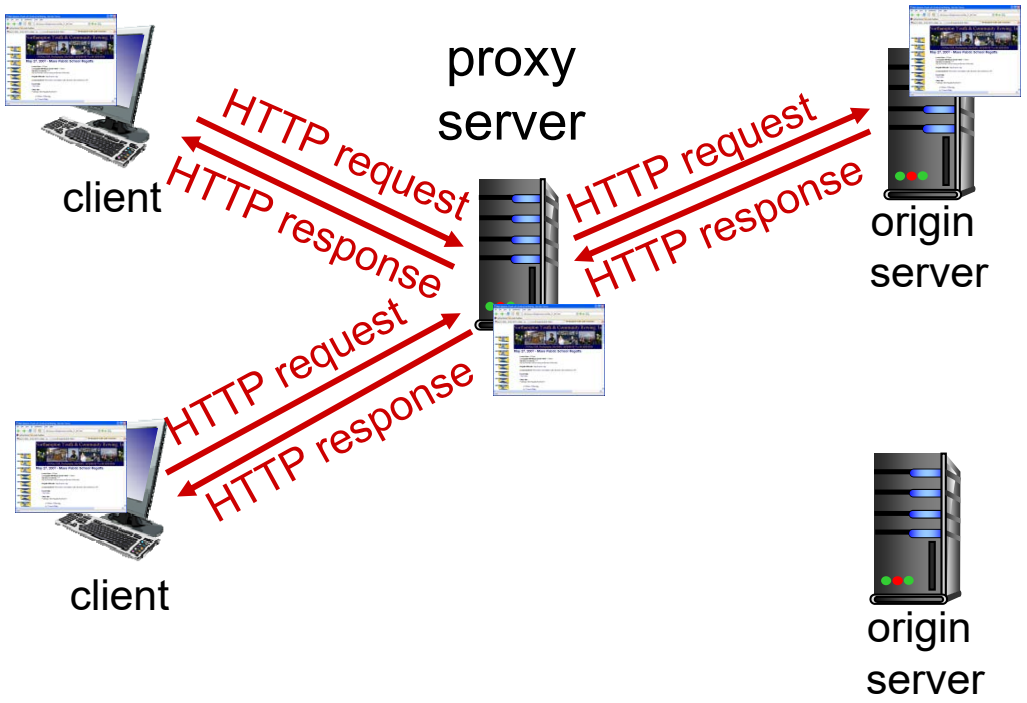
Cookies: Keeping "State"



Web Caches (Proxy Server)

Goal: satisfy client requests without involving origin server

- User starts browser: Web accesses via proxy
- Browser sends all HTTP requests to proxy
  - If object in cache: cache returns object if "fresh"
  - else proxy requests object from origin server, then returns object to client
    - Keeps a copy in cache



Fresh: Object has an expiry time or other age-controlling header set and is still within the fresh period

Expires: 31 Dec 2024 14:19:41 GMT

More About Web Caching

- Cache acts as both client and server
  - Server for original requesting client
  - Client to origin server
- Typically cache installed by ISP
  - University, company, residential ISP

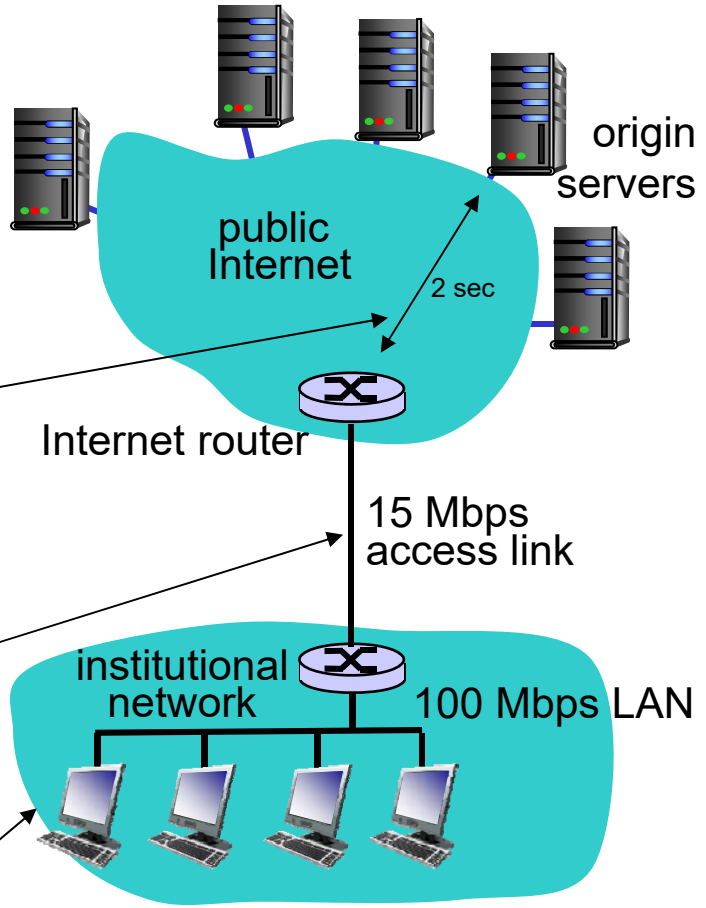
Why Web caching?

- Reduce response time for client request
- Reduce traffic on an institution's access link
- Reduce web traffic on the Internet

Caching Example - Baseline

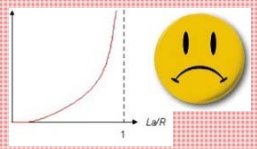
Assumptions

- Average object size = 1,000,000 bits
- Avg. request rate from institution's browsers to origin servers = 15/sec
  - 15Mbps required
- RTT from Internet router to any origin server and back to router = 2 sec (Internet delay)



Consequences

- Utilization on access link 15Mbps/15Mbps = 100%
  - Traffic intensity = 1



- LAN Utilization = 15Mbps/100Mbps → 15%

- Total delay = Internet delay + access delay (delay between routers) + LAN delay
  = 2 sec + **minutes** + milliseconds

Caching Example – Increase Bandwidth

Possible solution

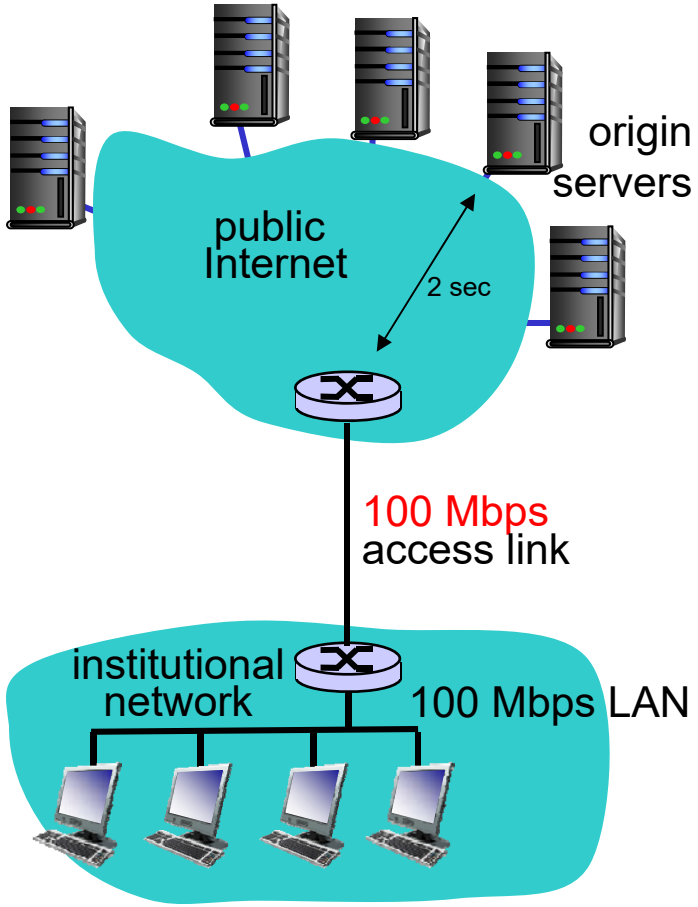
- Increase bandwidth of access link to **100** Mbps

Consequences

- Utilization on LAN = 15%
- Utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay

  = 2 sec + msecs + msecs

- **Often a costly upgrade**



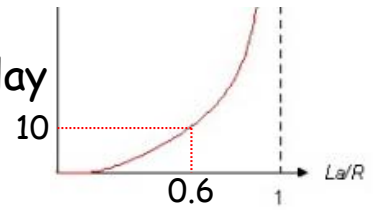
Caching Example – Install a Cache

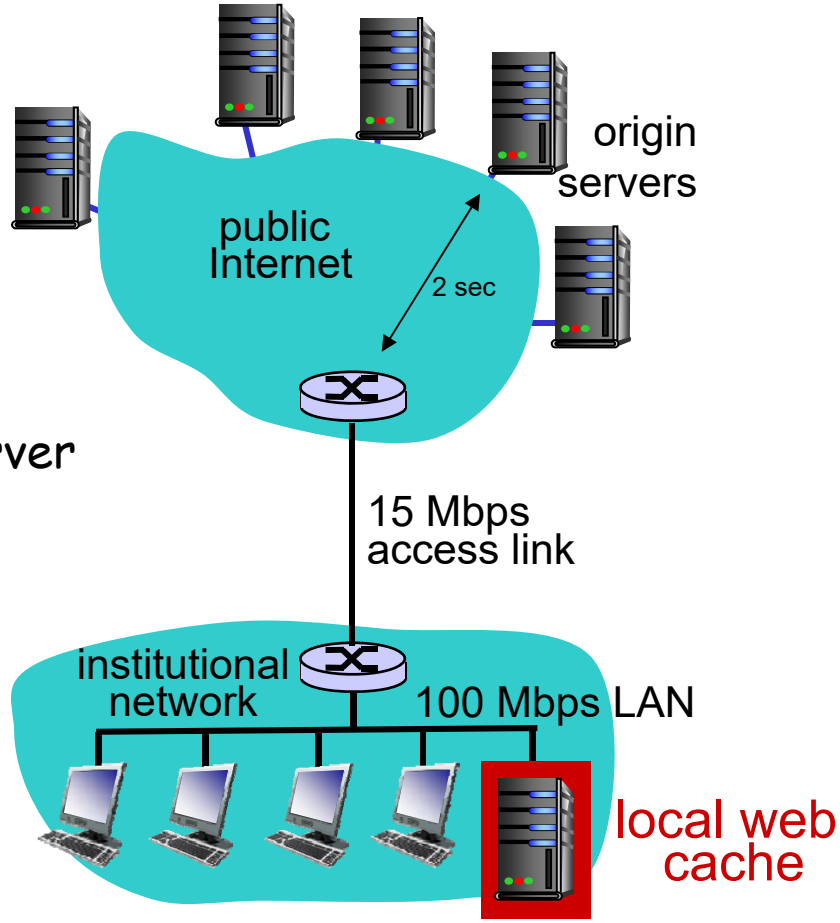
Install a cache

- Suppose hit rate is 0.4

Consequence

- 40% requests are fresh and can be satisfied almost immediately (10 msec)
  - 60% requests satisfied by origin server
- Access link utilization:
  - 60% of requests use access link
  - Data rate over access link = 0.6*(15)(1,000,000)Mbps = 9 Mbps
  - With 0.6 utilization
    - Assume 10 msec delay





Expires: 31 Dec 2024 14:19:41 GMT

- Total avg delay =
  - 0.6 * (delay from origin servers) + 0.4 * (delay when satisfied from cache)
  - 0.6 * (2.01) + 0.4 * (10 msec)
  - 1.21 secs

Should everything be cached?

Bank transactions?

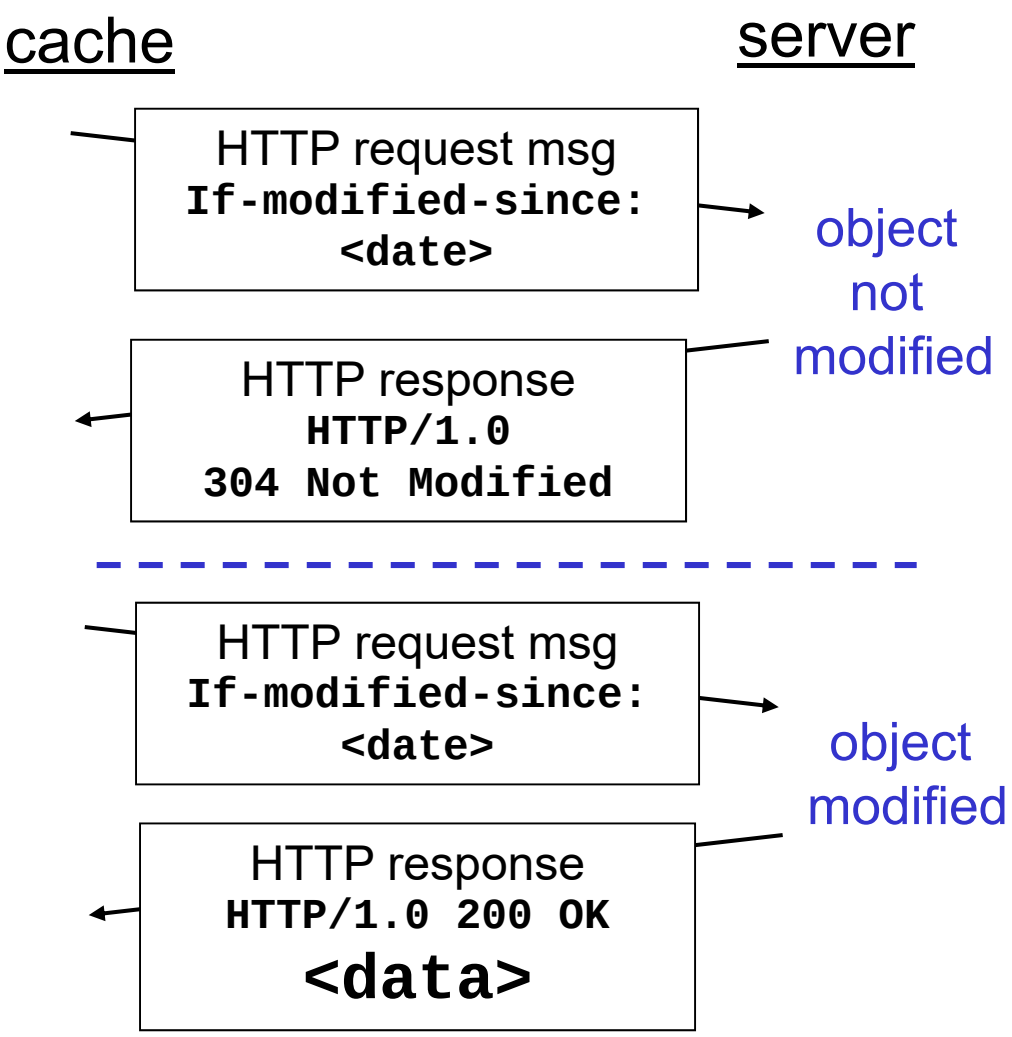
Conditional GET

- Q: How do we know the cached object is the latest version (i.e., "fresh") if it has "expired"?
- A: Use a conditional GET to validate the object

- Goal: don't send object if cache already has up-to-date version
- Cache: specify date of cached copy in HTTP request

  `If-modified-since: <date>`
- Server: response contains no object if cached copy is up-to-date:

  `HTTP/1.0 304 Not Modified`



Conditional Get Example

1. First cache request

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
```

2. First server response

```
HTTP/1.1 200 OK
Date: Mon, 7 Jul 2003 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Wed, 2 Jul 2003
09:23:24
Content-Type: image/gif

(data data data …)
```

3. Cache saves Last-Modified time of the object

4. Subsequent cache request

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 2 Jul 2003 09:23:24
```

5. Subsequent server response

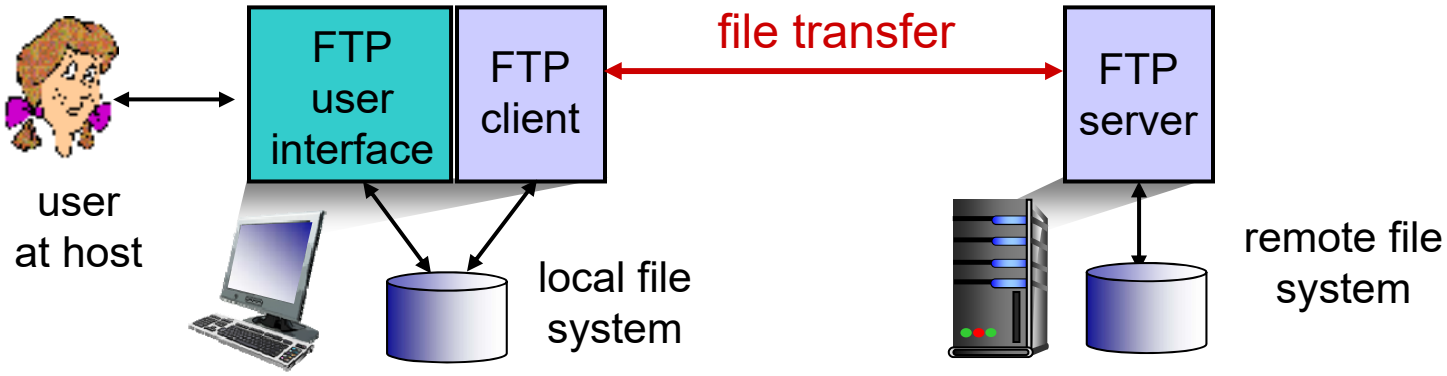
```
HTTP/1.1 304 Not Modified
Date: Mon, 14 Jul 2003 15:39:29
Server: Apache/1.3.0 (Unix)

(empty entity body …)
```

Chapter 2: Application Layer

- 2.1 Principles of Network Applications
- 2.2 Web and HTTP
- FTP
- 2.3 Electronic Mail
  - SMTP, POP3, IMAP
- 2.4 DNS
- 2.5 P2P Applications
- 2.6 Video Streaming and Content Distribution Networks
- 2.7 Socket Programming: Creating Network Applications

FTP: File Transfer Protocol



- Transfer file to/from remote host
- Client/Server model across TCP
  - Client: side that initiates transfer
  - Server: remote host
- FTP: RFC 959
- FTP server listens on port 21

FTP Commands, Responses

Sample commands:

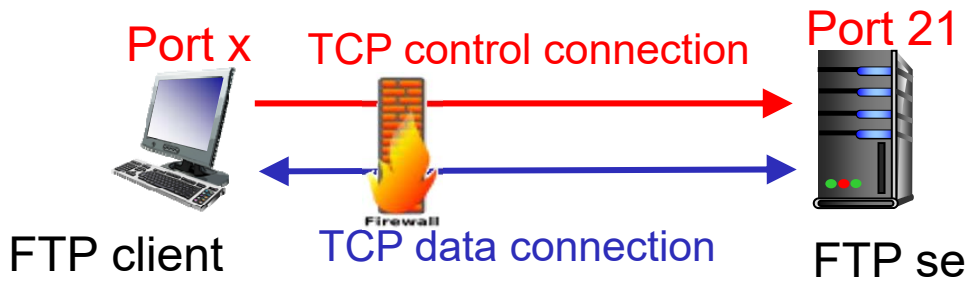
- Sent as ASCII text over control channel
- **USER *username***
- **PASS *password***
- **LIST / LS** return list of files in current directory
- **GET filename** retrieves a file
- **PUT filename** sends a file to the remote host (server)

Sample return codes

- Status code and phrase (as in HTTP)
- Sent as ASCII text over control channel
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

FTP: Separate Control, Data Connections

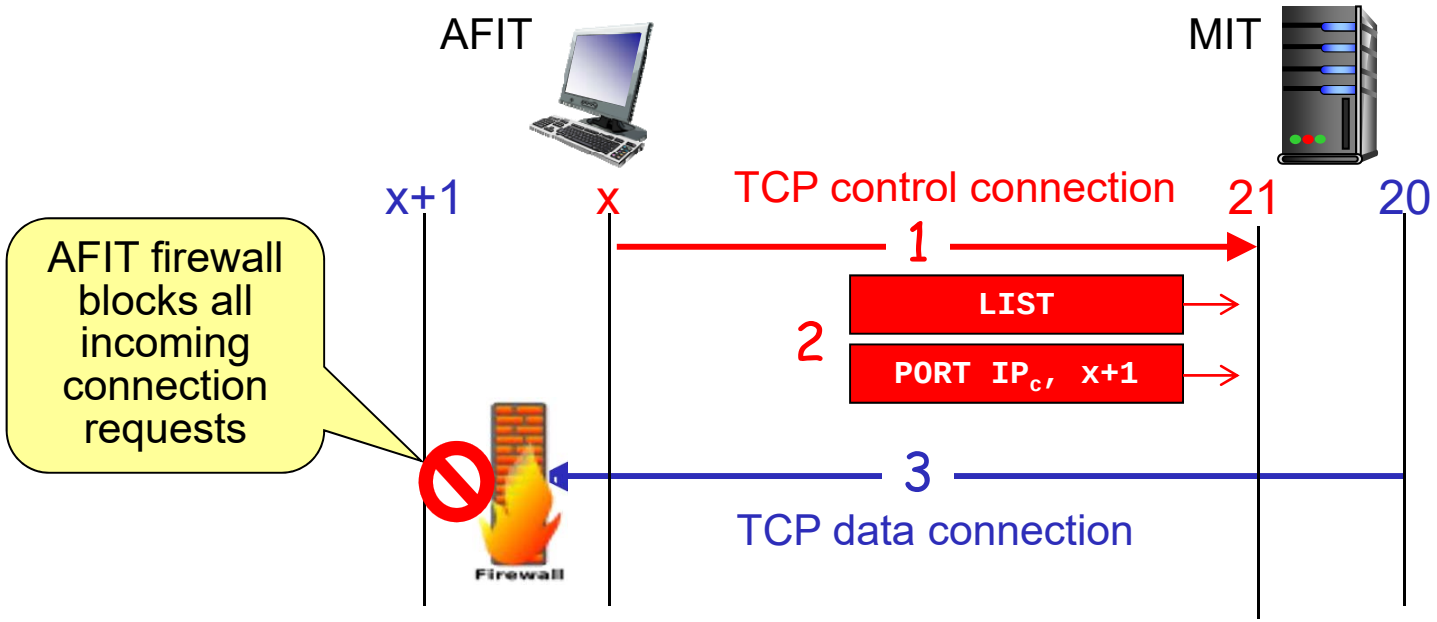
- Uses TWO connections
- **Control** connection is used to
  - Obtain authorization (username and password)
  - Browse remote directories on server
  - Control connection is called "out of band"
- **Data** connection is used to transfer data including directory listings
  - FTP can run in active or passive mode, which determines how the data connection is established
  - Use passive mode if incoming connections are blocked by firewall



Server maintains state information

Active FTP (Server → Client)

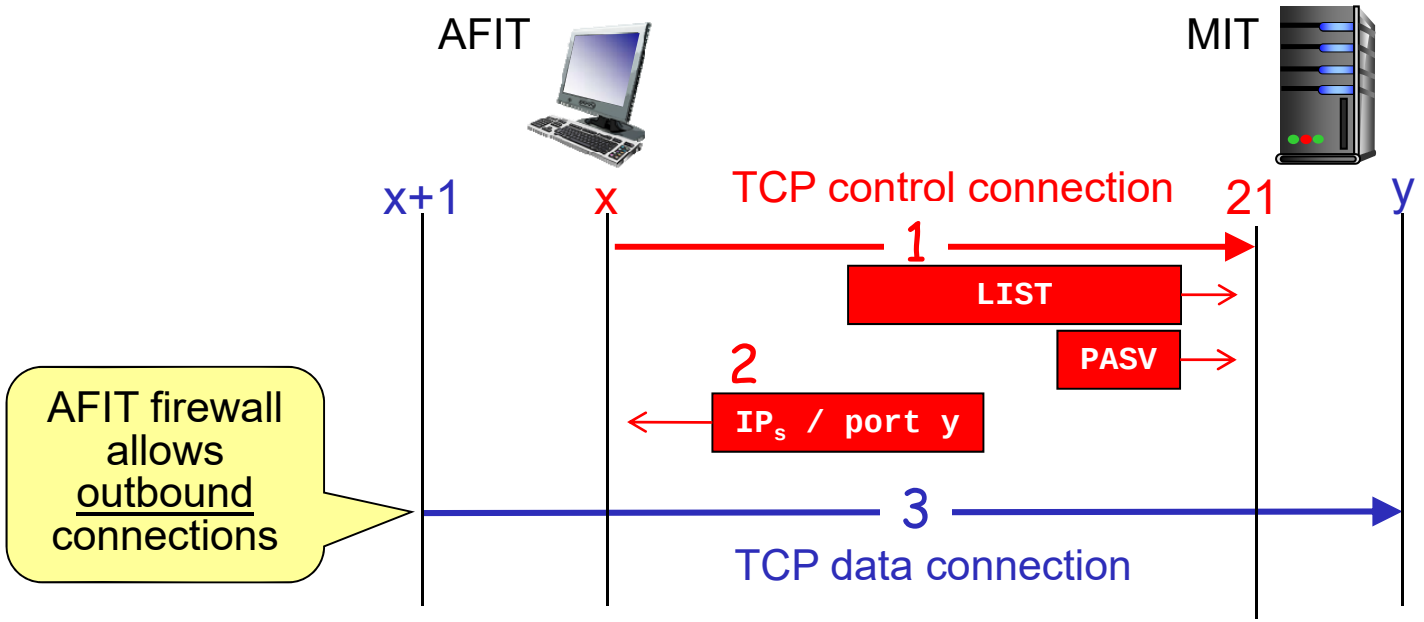
1. AFIT client (port x: x>1023) connects to FTP server (MIT) on TCP port 21
2. When client sends a command for a file transfer / directory listing (**LIST**) …
   - Client starts listening on port x+1 and sends **PORT** command across command connection to server to provide client's IP address (IP_c) and listening port (x+1)
3. **Server** opens a TCP data connection from port 20 to port x+1 on client
   - After transferring one file/listing, server closes data connection
4. Subsequent file transfers/listings → repeat steps 2 and 3 with different client ports (i.e., not x+1)



Passive FTP (Client → Server)

Most Web browsers (which act as FTP clients) use passive FTP by default

1. AFIT client opens two ports (port x and x+1; x > 1023) and connects the first port (x) to FTP server (MIT) on TCP port 21
2. When client sends a command for a file transfer / directory listing (**LIST**) …
   - Client sends **PASV** and waits for server response
   - Server responds with its IP address (IP_s) / listening port (y: y>1023)
3. **Client** then opens a TCP data connection from port x+1 to port y on IP_s
4. Subsequent file transfers/listings → repeat steps 2 and 3 with different client ports (i.e., not x+1)



Chapter 2: Application Layer

- 2.1 Principles of Network Applications
- 2.2 Web and HTTP
- FTP
- 2.3 Electronic Mail
  - SMTP, POP3, IMAP
- 2.4 DNS

- 2.5 P2P Applications
- 2.6 Video Streaming and Content Distribution Networks
- 2.7 Socket Programming: Creating Network Applications

Email

- Email is transferred from one host to "another" using the Simple Mail Transfer Protocol (SMTP)
  - SMTP is a delivery protocol only

- Like HTTP, SMTP has a similar set of ASCII commands and replies to transfer messages between machines

- Recipient of SMTP traffic must always be on
  - ✓ SMTP traffic can flow from:
    - ✓ Sending host → dedicated email server
    - ✓ Dedicated email server → another dedicated email server
  - ✗ SMTP traffic cannot flow from:
    - ✗ Dedicated email server → a receiving host
      - We use POP or IMAP for this

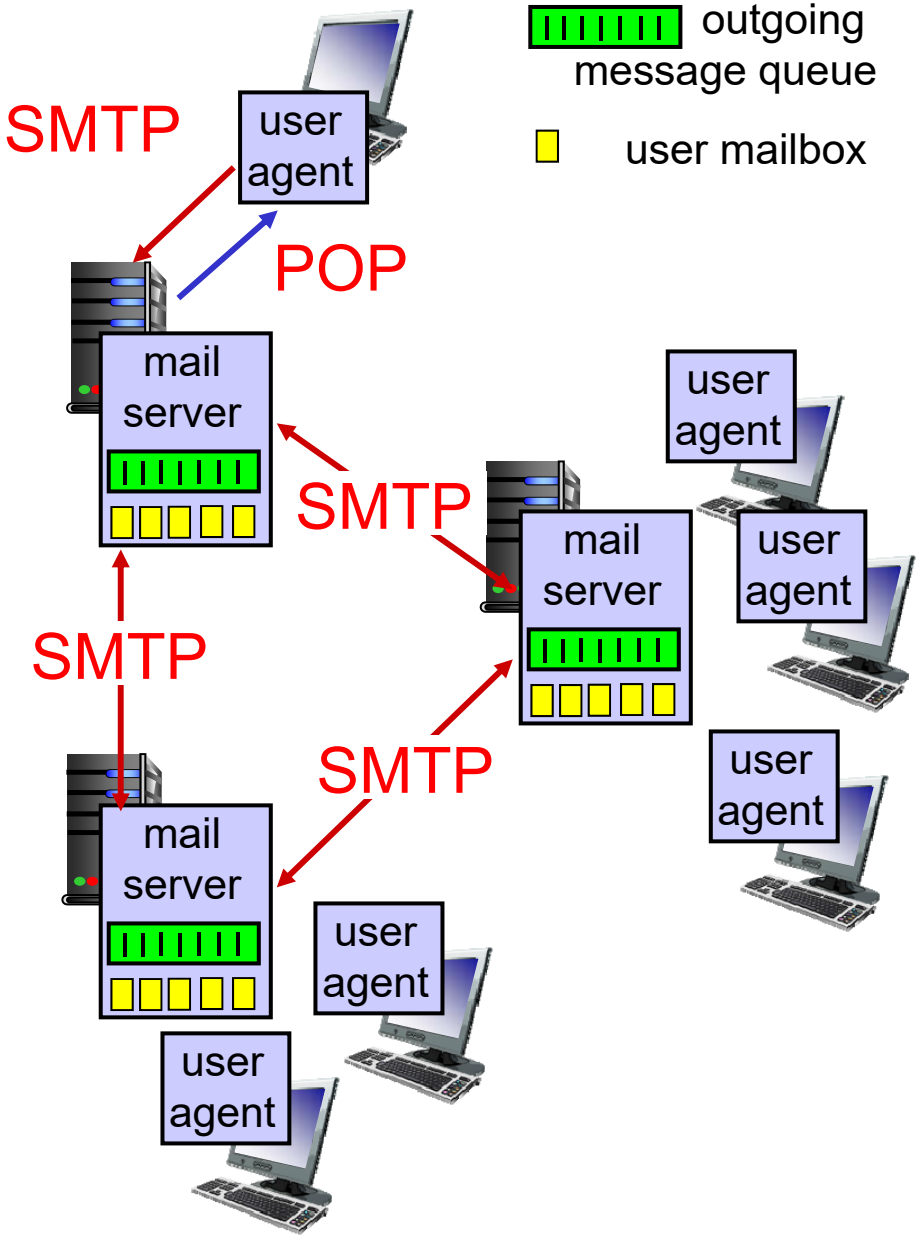
Electronic Mail

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, Mail (Linux)
- Outgoing, incoming messages stored on server



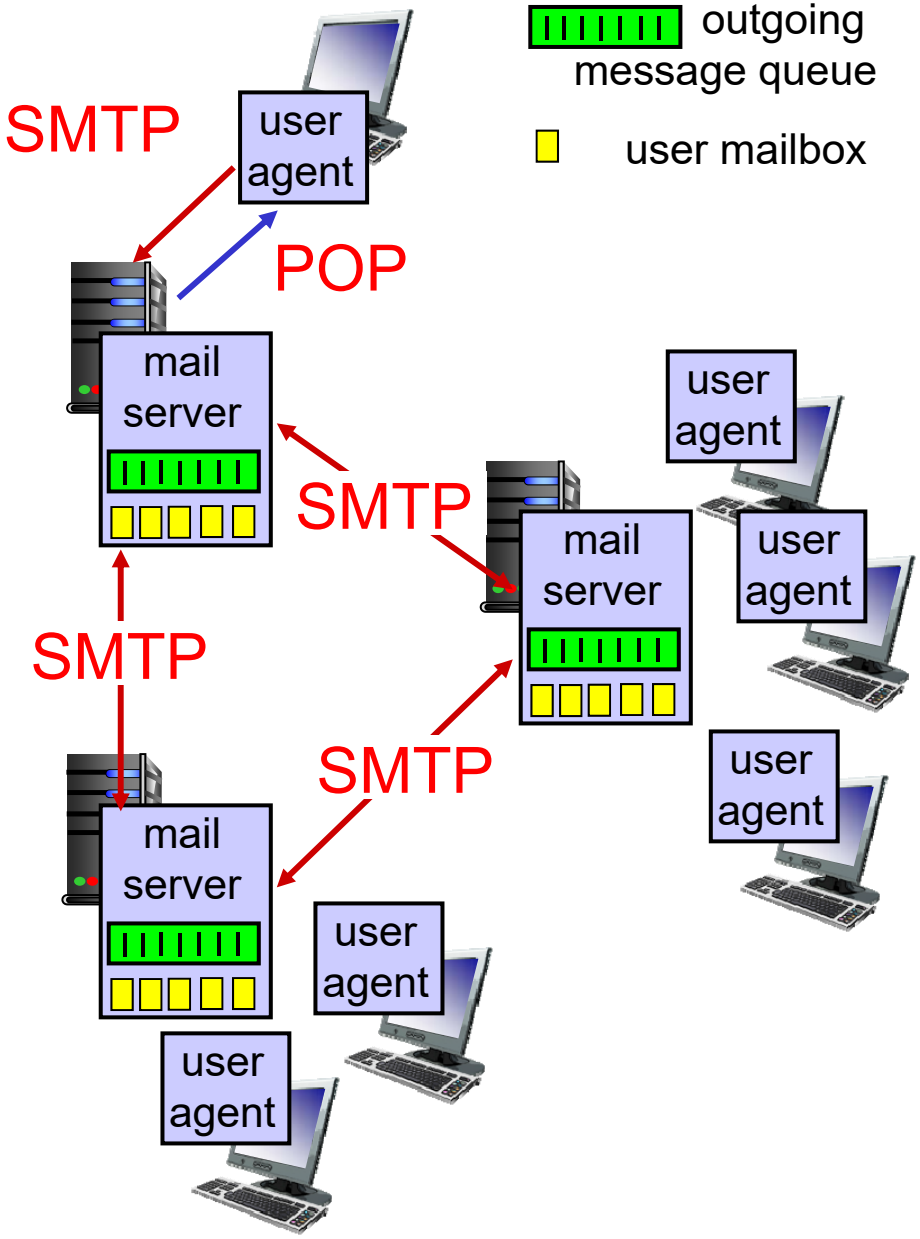
Electronic Mail

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": receiving mail

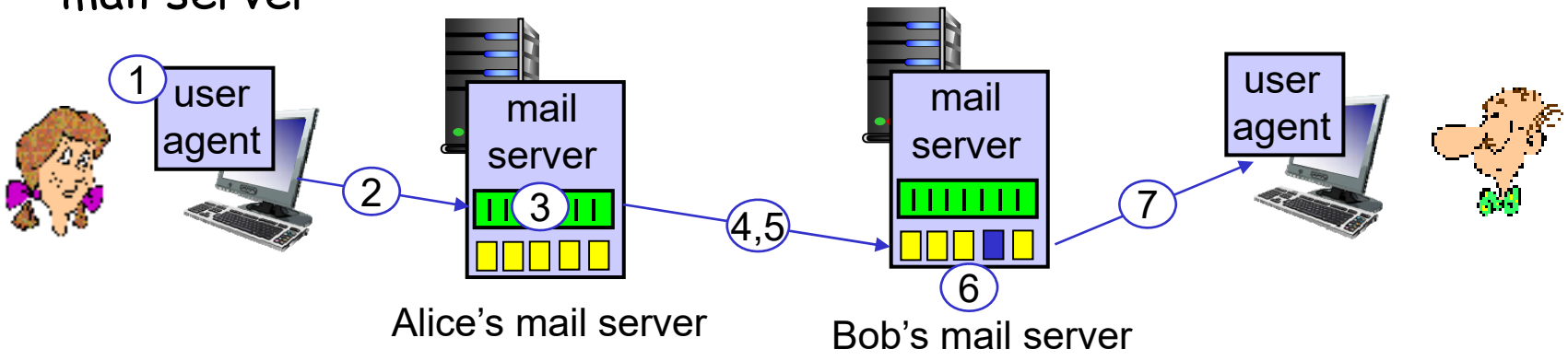


Electronic Mail: SMTP [RFC 822/2821]

- Uses TCP to transfer email message
  - From client to server port 25
  - From sending server to receiving server (direct transfer)
- Three phases of transfer
  - Handshaking (greeting)
  - Transfer of messages
  - Closure
- Command/Response interaction
  - Commands: ASCII text
  - Response: ASCII status code and phrase
- Messages must be in 7-bit ASCII

Scenario: Alice Sends Message to Bob

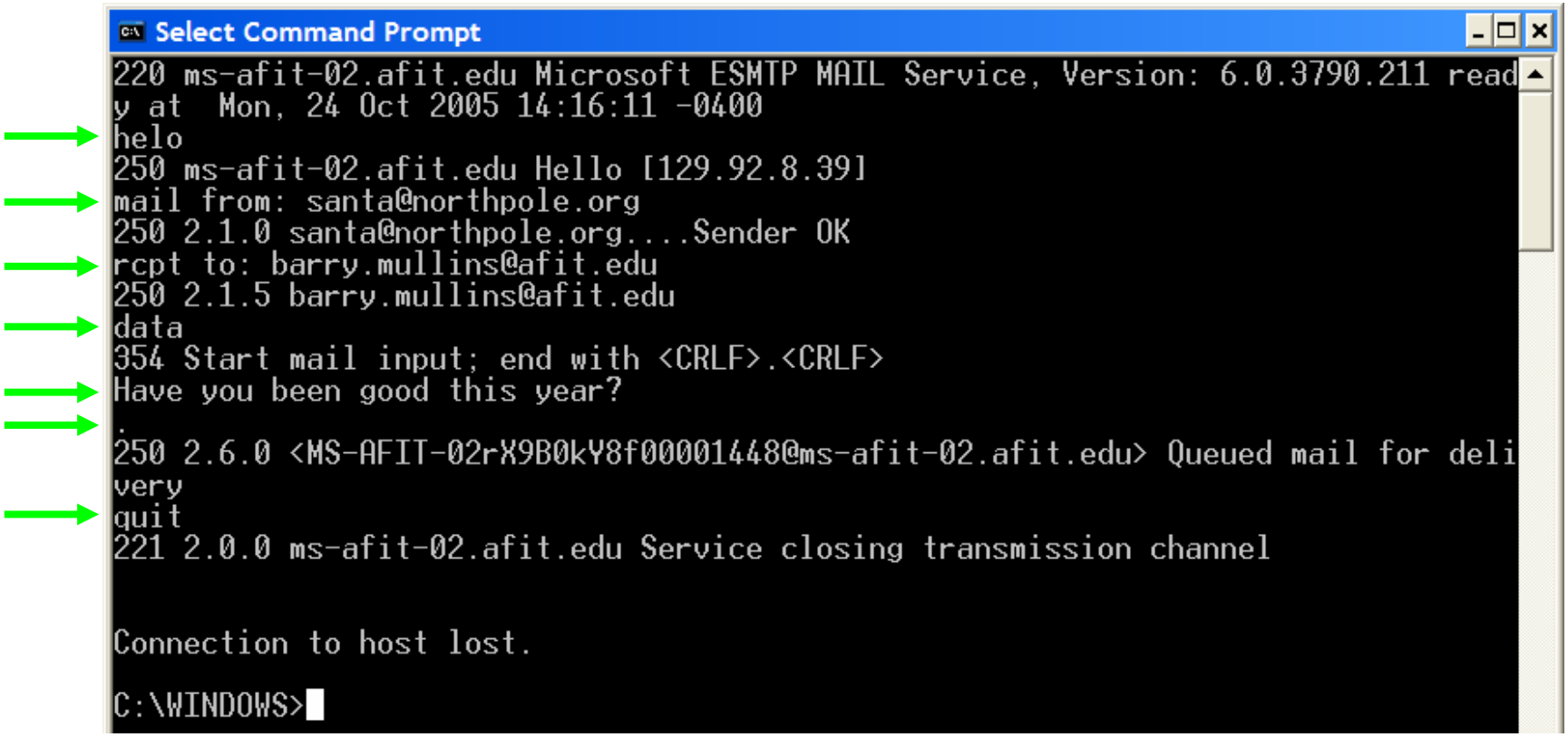
1) Alice uses UA to compose message "to" **bob@someschool.edu**
2) Alice's UA sends message to her mail server (SMTP or HTTP)
3) Message placed in message queue
4) Client side of SMTP opens TCP connection with Bob's mail server
5) SMTP client sends Alice's message over the TCP connection
6) Bob's mail server places the message in Bob's mailbox
7) Bob invokes his user agent to read message (POP, HTTP, IMAP)



Sample SMTP Interaction

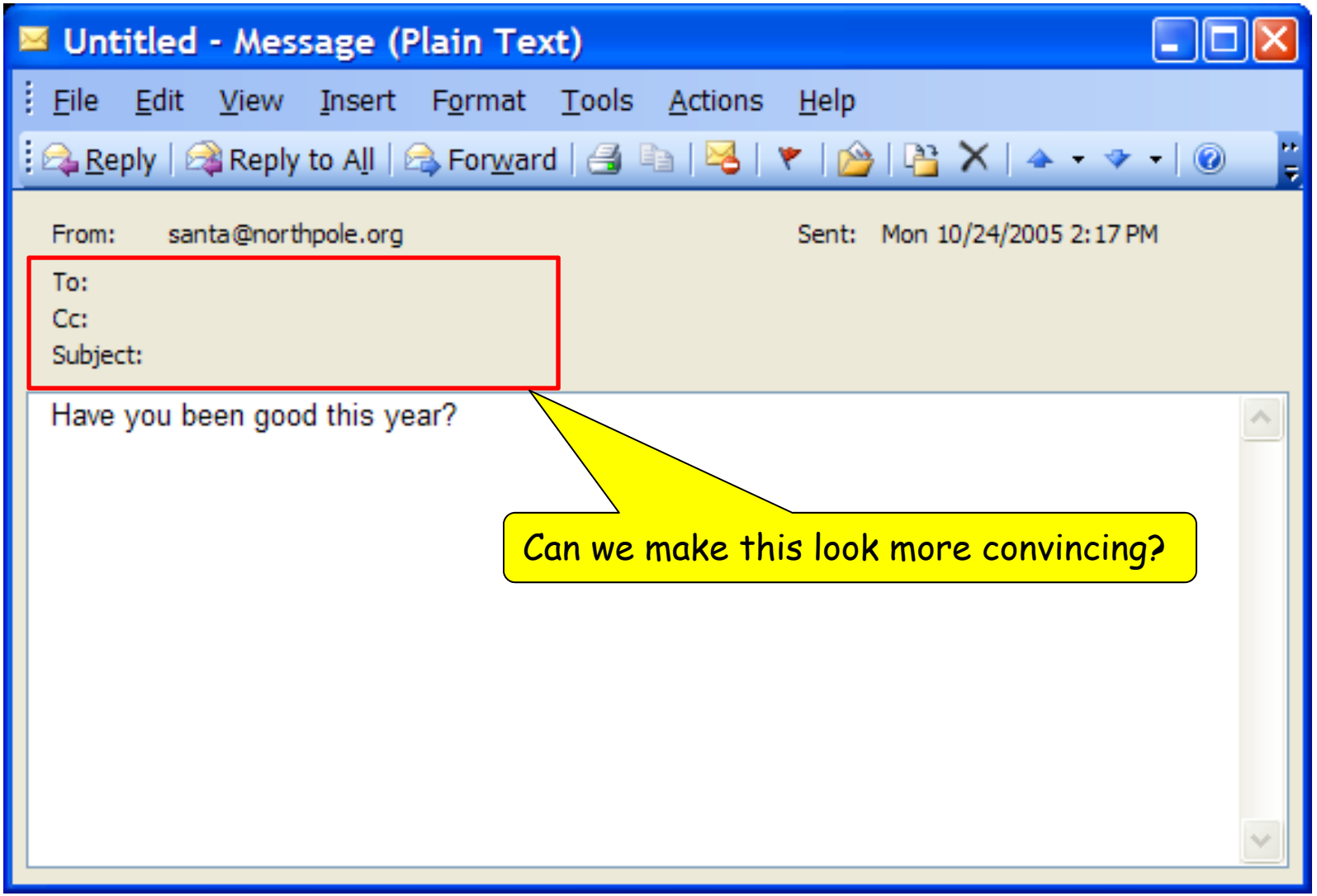
Assumes you are connected to the AFIT network (i.e., VPN)

- **telnet servername 25**
  - **telnet ms-afit-02 25**



Above lets you send email without using email client (reader)

Sample SMTP Interaction



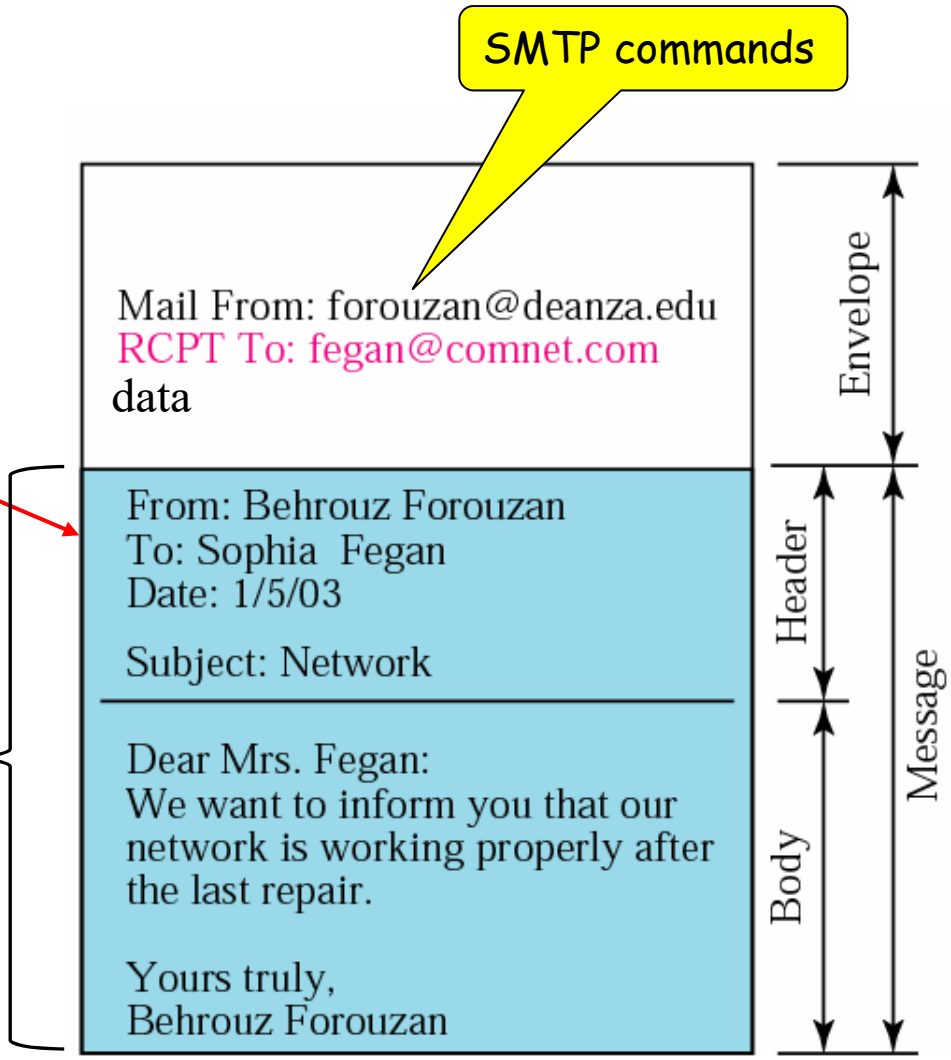
Mail Message Format

RFC 822

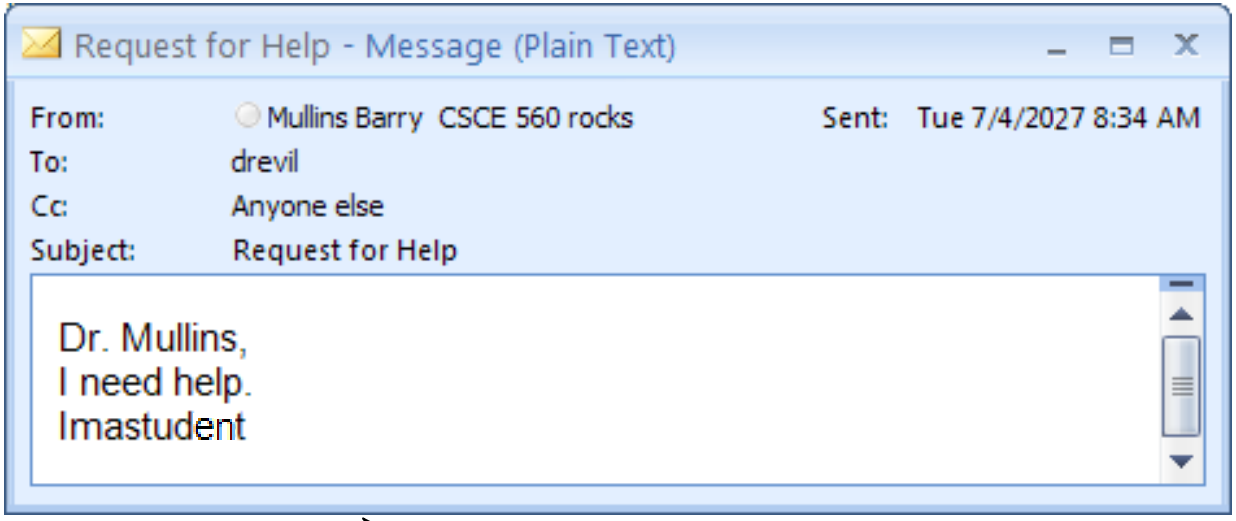
standard for text message format:

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

  different from SMTP commands!
- "Message"
  - ASCII characters only



An Example Mail Session




```
telnet ms-afit-02  25
helo
mail from: anything
rcpt to: barry.mullins@afit.edu
data
to: drevil
from: Mullins Barry CSCE 560 rocks
subject: Request for Help
cc: Anyone else
date: 4 Jul 27 08:34 EDT
Dr. Mullins,
I need help.
Imastudent
.<CR><LF>
```

A period followed by the Enter key

Now you do it!

Your assignment is to send me a spoofed email. Include your real name in the email body. Also include the word "spoofed" somewhere in the body. If you don't, I won't know the email is from you and the department head may get involved.

Don't ask; it's a long story. ☺

http://www.e-eeasy.com/SMTPServerList.aspx

SMTP: Comparison with HTTP

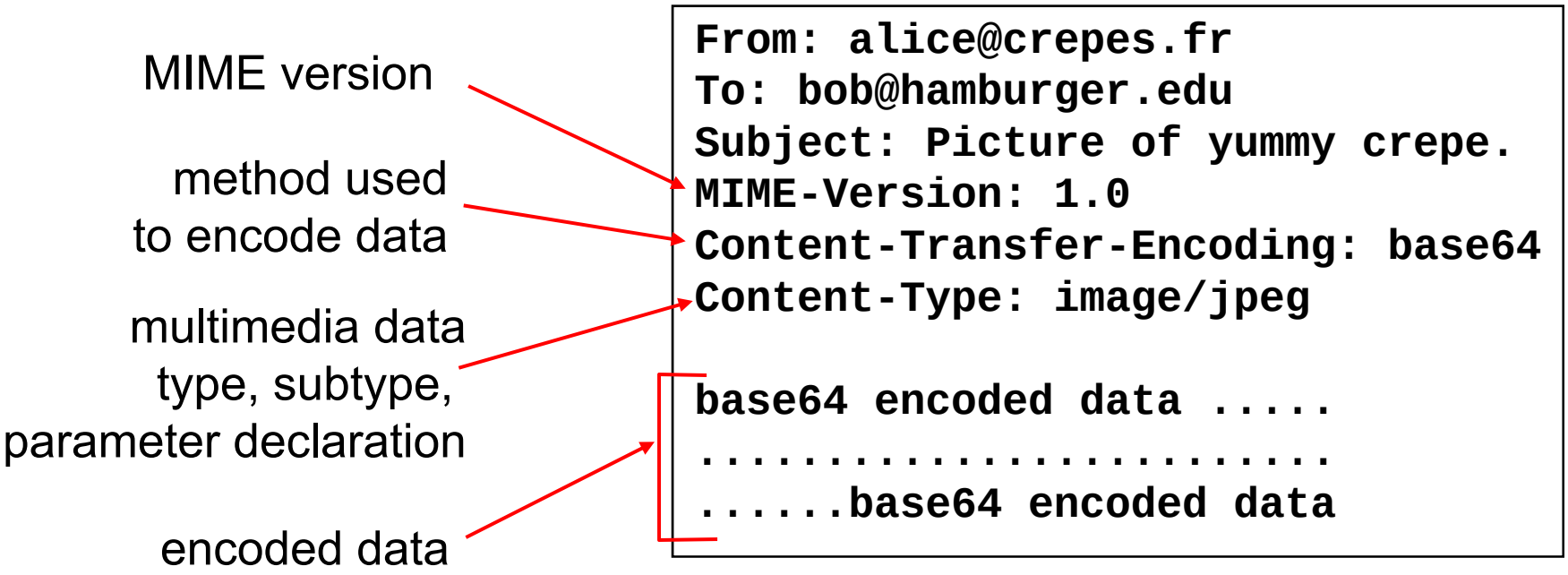
- HTTP: pull
- SMTP: push

- Both have ASCII command/response interaction, status codes

- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

Message Format: Multimedia Extensions

- Multipurpose Internet Mail Extensions (MIME)
  - Multimedia mail extension, RFC 2045, 2046
- Additional lines in msg header declare MIME content type



Base64 Acceptable Characters

- Need to send only printable characters that can be represented using 7 bits ($0\text{-}127_{10}$ or $00\text{-}7F_{16}$) and avoid "special" chars $0\text{-}1F_{16}$

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

https://en.wikipedia.org/wiki/Base64

Base64 Index Table

- Arrange these characters in an index table

Index	Char	Index	Char	Index	Char	Index	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

https://en.wikipedia.org/wiki/Base64

Base64 Example

- Text to encode: **Man**
- Encode using ASCII table
- Convert to binary
- Group into 6 bits and convert to decimal
- Look up decimal number in index table
- Send 7-bit printable, non-special ASCII character

77 4D M

84 54 T

19	T
20	U
21	V
22	W

Source	Text (ASCII)	M								a								n								
	Octets	77 (0x4d)								97 (0x61)								110 (0x6e)								
Bits		0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0	
Base64 encoded	Sextets	19								22				5				46								
	Character	T								W				F				u								
	Octets	84 (0x54)								87 (0x57)				70 (0x46)				117 (0x75)								

MIME Types

`Content-Type: type/subtype; parameters`

Text

- Example subtypes:
  - **`plain, html`**

Image

- Example subtypes:
  - **`jpeg, gif`**

Audio

- Example subtypes:
  - **`basic`** (8-bit mu-law encoded)
  - **`32kadpcm`** (32 kbps coding)

Video

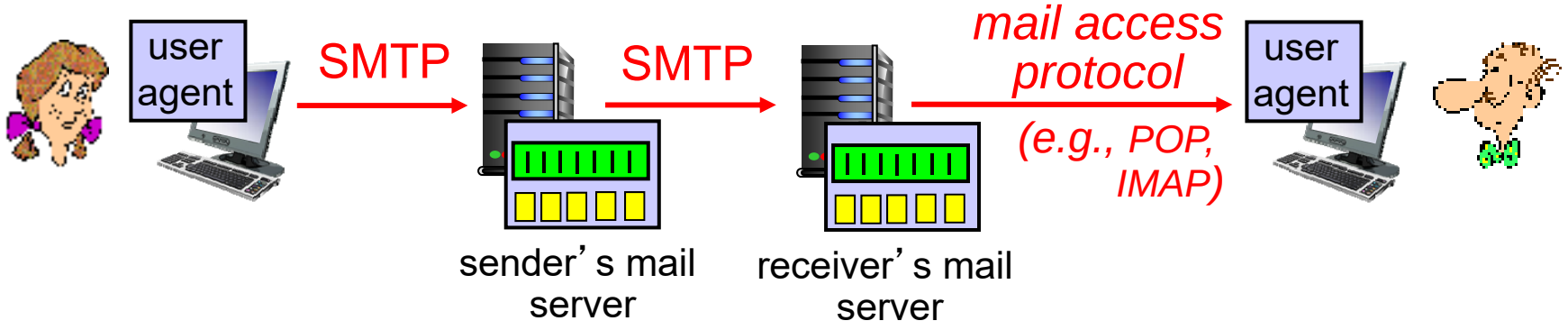
- Example subtypes:
  - **`mpeg, quicktime`**

Application

- Other data that must be processed by reader before "viewable"
- Example subtypes:
  - **`msword, octet-stream`**

Binary file

Mail Access Protocols



- Mail access protocol: retrieve mail from server
  - POP: Post Office Protocol [RFC 1939]
    - Authorization (agent <--> server)
    - Download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - More features (more complex)
    - Manipulation of stored msgs on server
  - HTTP: gmail, 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**

Update phase

- Server deletes marked messages

```
telnet mailserver 110
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

Sent in plain text!

```
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

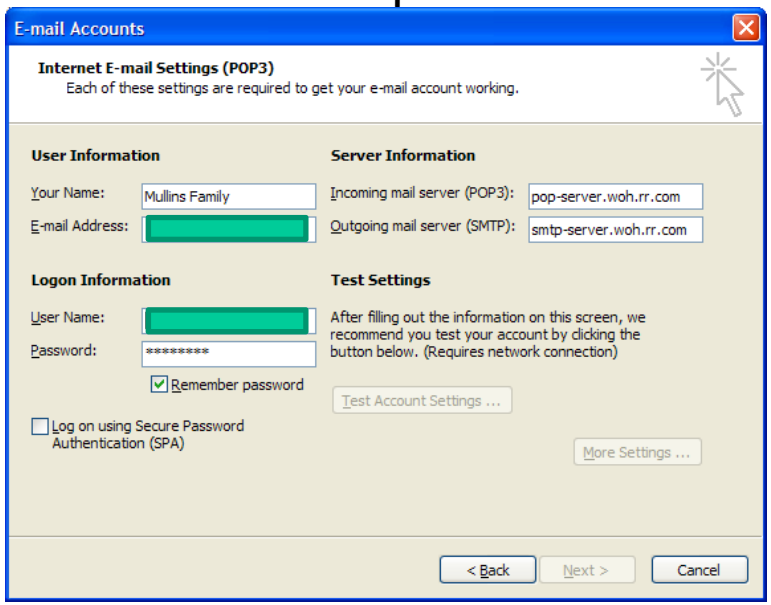
More about POP3

- Previous example uses "download-and-delete" mode
- Bob cannot re-read email if he changes client
- "Download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions
- Server listens on port 110

- Use SMTP to **send** mail to your mail server
- Use POP to **download** mail from the mail server

IMAP

- Keep all messages in 1 place: server
- Allows user to organize messages in folders
- IMAP keeps user state across sessions:
  - Names of folders and mappings between message IDs and folder name
- Server listens on port 143



IMAP - Internet Message Access Protocol

Chapter 2: Application Layer

- 2.1 Principles of Network Applications
- 2.2 Web and HTTP
- FTP
- 2.3 Electronic Mail
  - SMTP, POP3, IMAP
- 2.4 DNS
- 2.5 P2P Applications
- 2.6 Video Streaming and Content Distribution Networks
- 2.7 Socket Programming: Creating Network Applications

Domain Name System (DNS) Motivation

- Problem statement:
  - Internet hosts and routers identified by:
    - 32-bit IP address (e.g., 129.92.6.150) – up to 12 digits
      - Used for addressing datagrams
  - Average brain can easily remember 7 digits
  - We need an easy way to remember IP addresses or provide a mapping between IP addresses and host names

- Solution:
  - Use alphanumeric names (e.g., www.yahoo.com) to refer to hosts
  - Add a distributed, hierarchical protocol (called DNS) to map between alphanumeric host names and binary IP addresses

  - We call this *Address Resolution* →

DNS: Domain Name System

Domain Name System is a:

- Distributed database implemented in a hierarchy of many name servers
- Application-layer protocol that allows hosts to query the distributed database to resolve names (address/name translation)

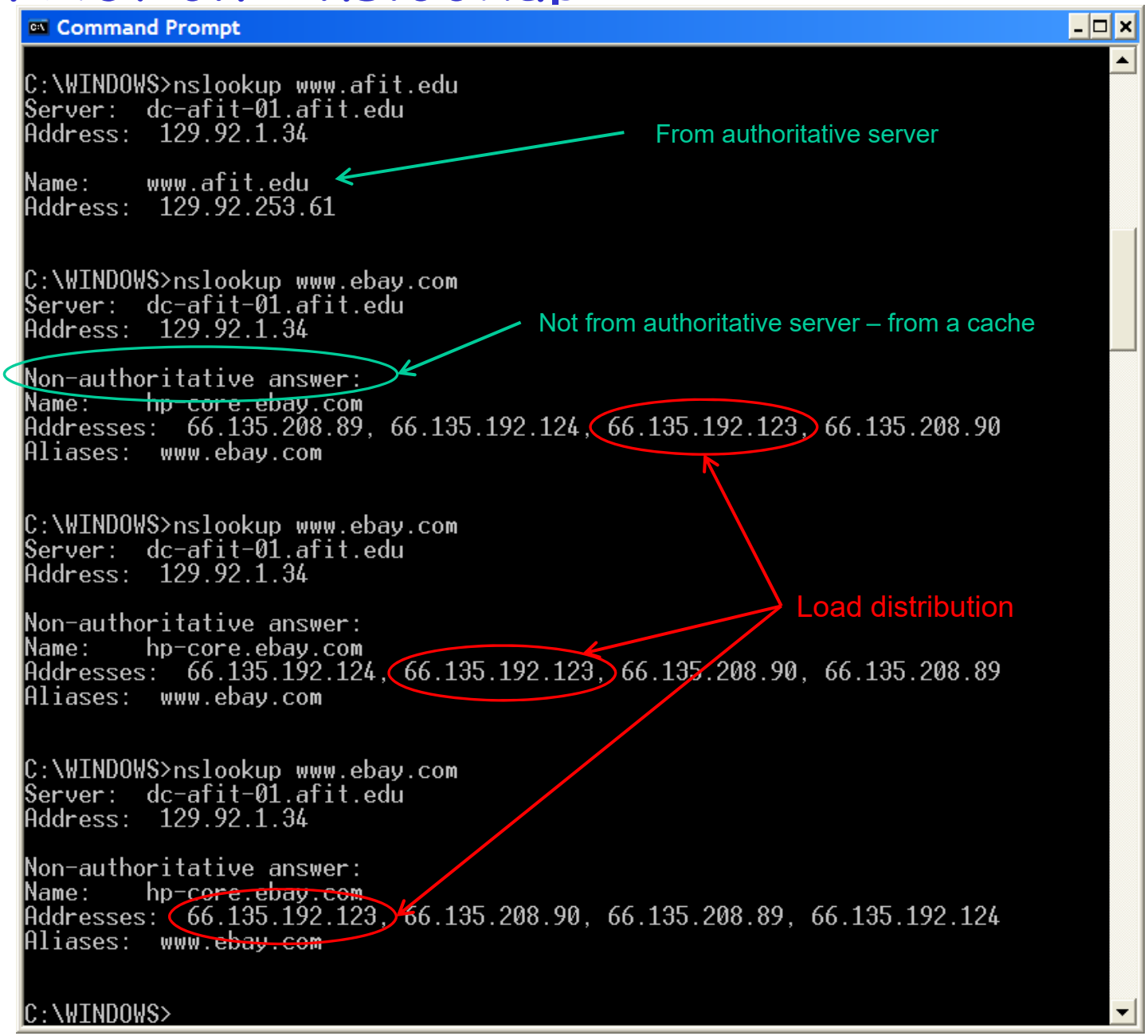
- Interesting note: A core Internet function is implemented as an application-layer protocol
  - Complexity pushed to network's "edge"

- Address resolution adds additional delay to an application
  - Must find correct IP address before setting up TCP connection

DNS Services

- Hostname to IP address translation
- Host aliasing
  - Canonical and alias names
  - Relay1.west-coast.enterprise.com = enterprise.com
  - Relay1.west-coast.enterprise.com = www.enterprise.com
- Mail server aliasing
  - We prefer bob@hotmail.com instead of bob@relay1.west-coast.hotmail.com
- Load distribution
  - Replicated web servers: set of IP addresses for one canonical name
  - DNS responds with the list of IP addresses but rotates order with each request

DNS in Action - nslookup



DNS – Basically How Does It Work?

- Your application needs to setup a connection with a server (www.mit.edu)
- Since your app does not already know the IP address of the destination, your app invokes DNS
  - Requests an IP address for a host name (www.mit.edu)
- DNS on your host
  - Checks HOSTS file for static entry
    - **`74.125.228.116  google.com`**
    - **`127.0.0.1  annoyingadsite.com`**
  - If not in HOSTS file, checks DNS cache on host
    - **`ipconfig /displaydns`**

C:\Windows\System32\drivers\etc
/etc/hosts

- If no hit on host, sends a query "into the network" using UDP port 53
  - Waits ms to seconds for the DNS response
- IP address is passed to calling application and used to initiate connection to the server

DNS

- Where is this "into the network"?
  - A single centralized DNS server?

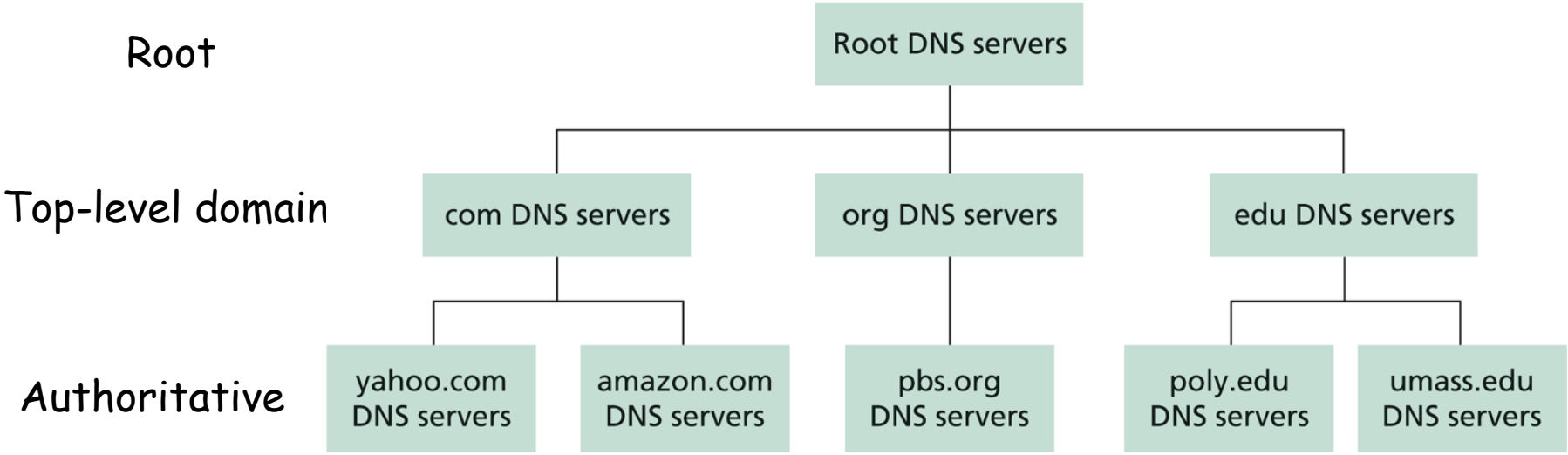
NO!!

Why not centralize DNS?

- Single point of failure
- Traffic volume
- Distant centralized database
  - Distance does matter – remember propagation delay?
- Maintenance
  - Database management would be unreal

- Doesn't *scale!*

Distributed, Hierarchical Database



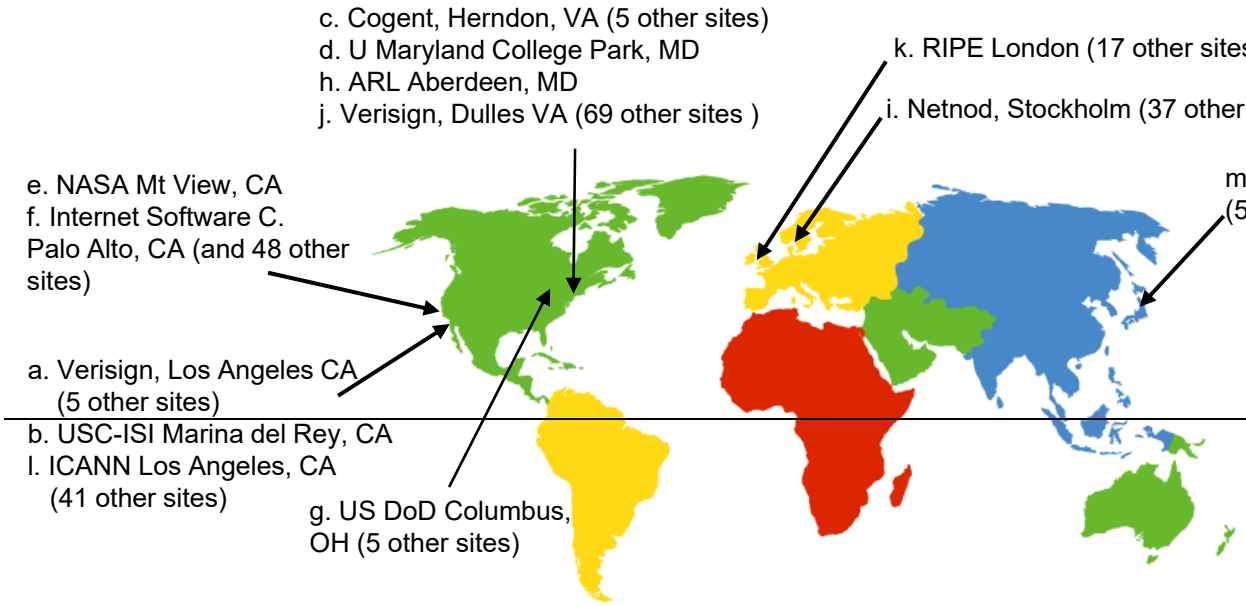
- No single DNS server has all the mappings!!

Client wants IP address for www.amazon.com; 1st approx:

- Client queries a root server to find .com DNS server
- Client queries .com DNS server to get amazon.com DNS server
- Client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: Root Name Servers

- ICANN: Internet Corporation for Assigned Names and Numbers
  - Manages DNS root servers
  - IP addresses → https://www.iana.org/domains/root/servers
  - Names → **`k.root-servers.net`**

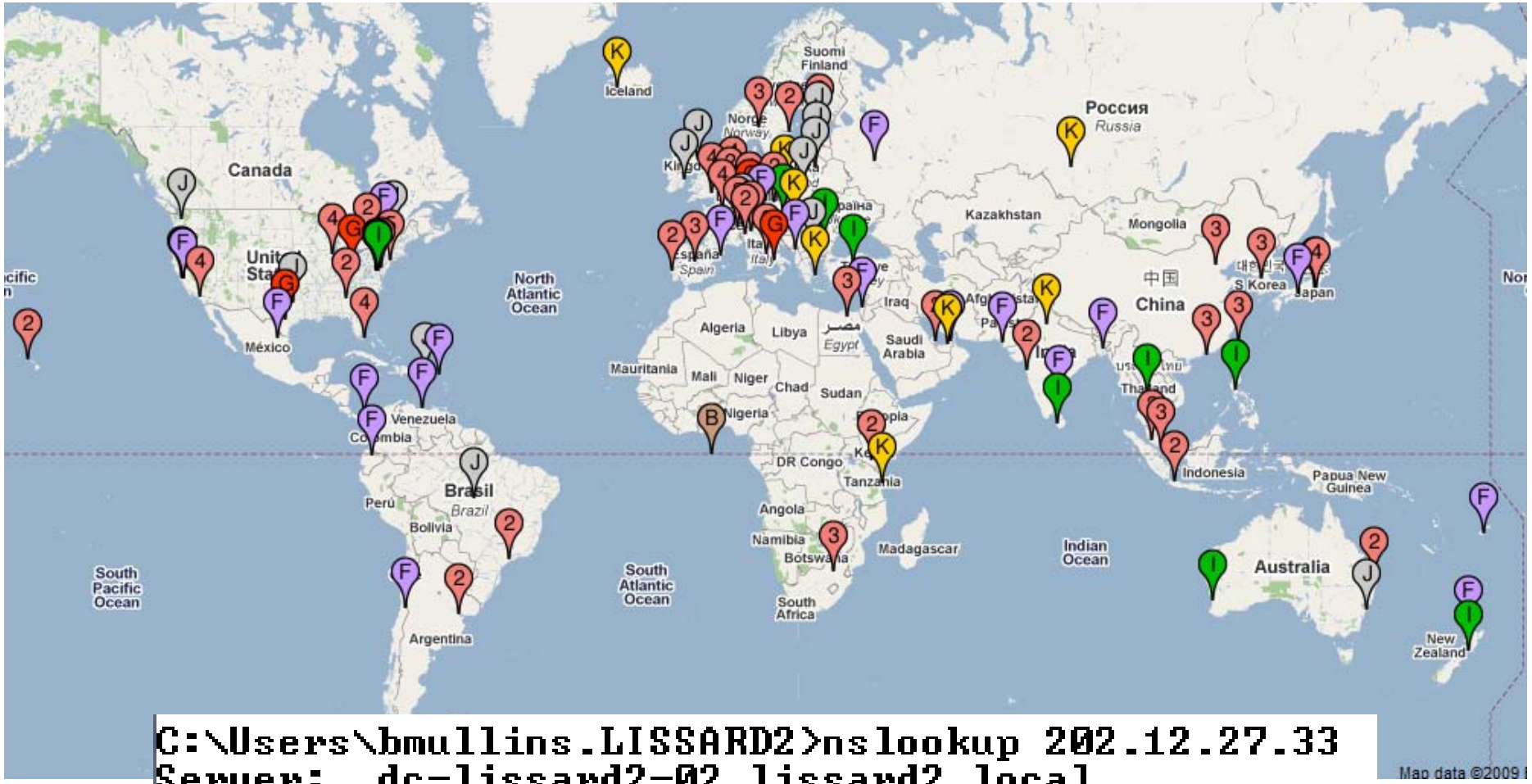


13 root name servers worldwide

Looks like the southern hemisphere is just out of luck…

Remember propagation delay?

DNS: Root Name Servers Replication




```
C:\Users\bmullins.LISSARD2>nslookup 202.12.27.33
Server:  dc-lissard2-02.lissard2.local
Address:  10.1.2.2

Name:    m.root-servers.net
Address:  202.12.27.33
```

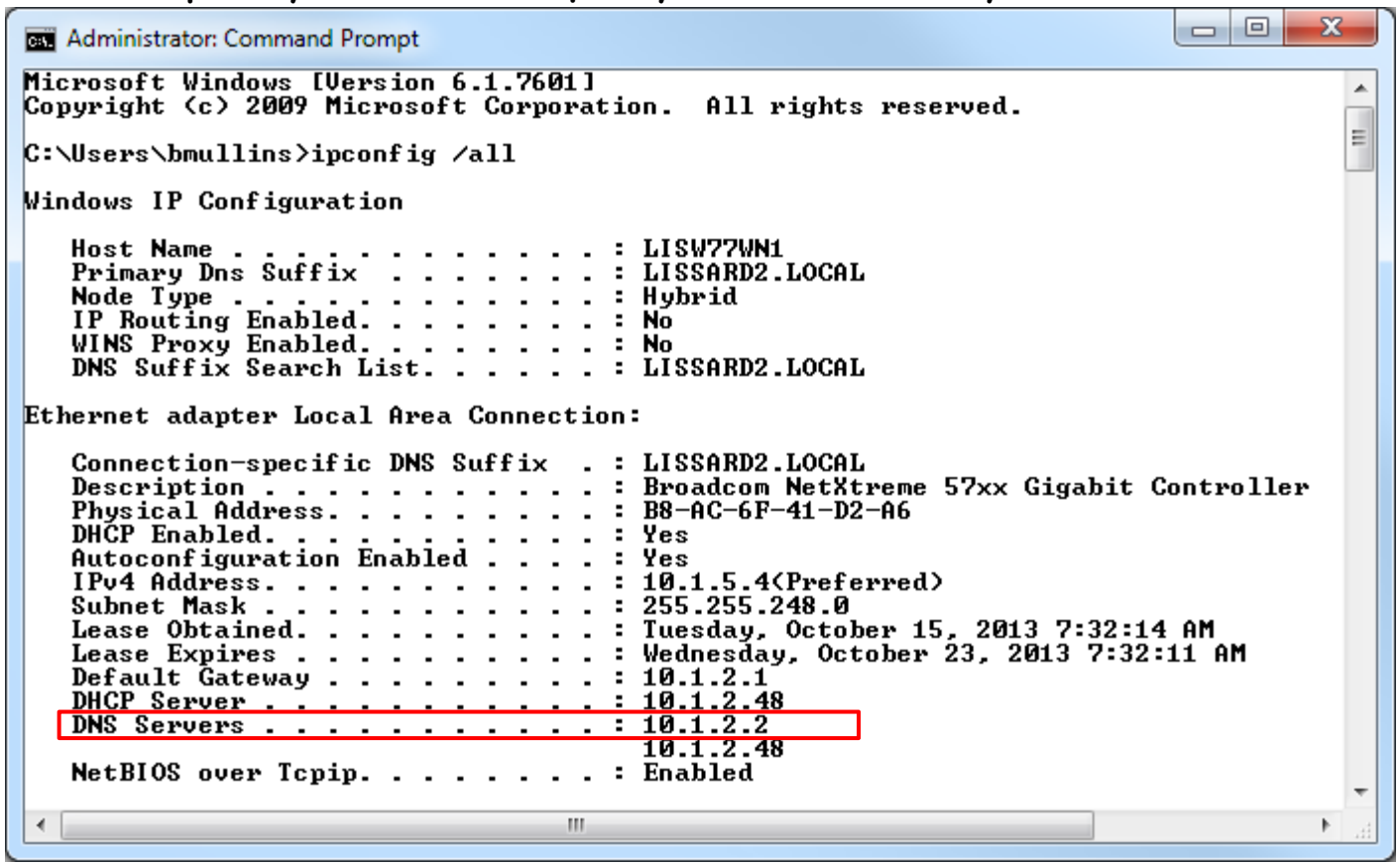
TLD and Authoritative Servers

- Top-level domain (TLD) servers:
  - Responsible for .com, .org, .net, .edu, .mil, .gov, etc, and all top-level country domains .uk, .fr, .ca, .jp
  - Network Solutions maintains TLD servers for .com
  - Educause maintains .edu TLD
  - As of June 2011, ICANN will allow any TLD → .barrymullins
    - At that time, only cost $185,000 ☺

- Authoritative DNS servers:
  - Organization's DNS servers provide authoritative hostname to IP mappings for organization's servers (e.g., Web and mail)
  - Can be maintained by organization or service provider

Local Name (DNS) Server

- Does not strictly belong within the hierarchy
- Each ISP (residential ISP, company, university) has one though
  - Also called "default name server"
- When host makes a DNS query, query is sent to its local DNS server
  - Server acts as a proxy, forwards query into hierarchy

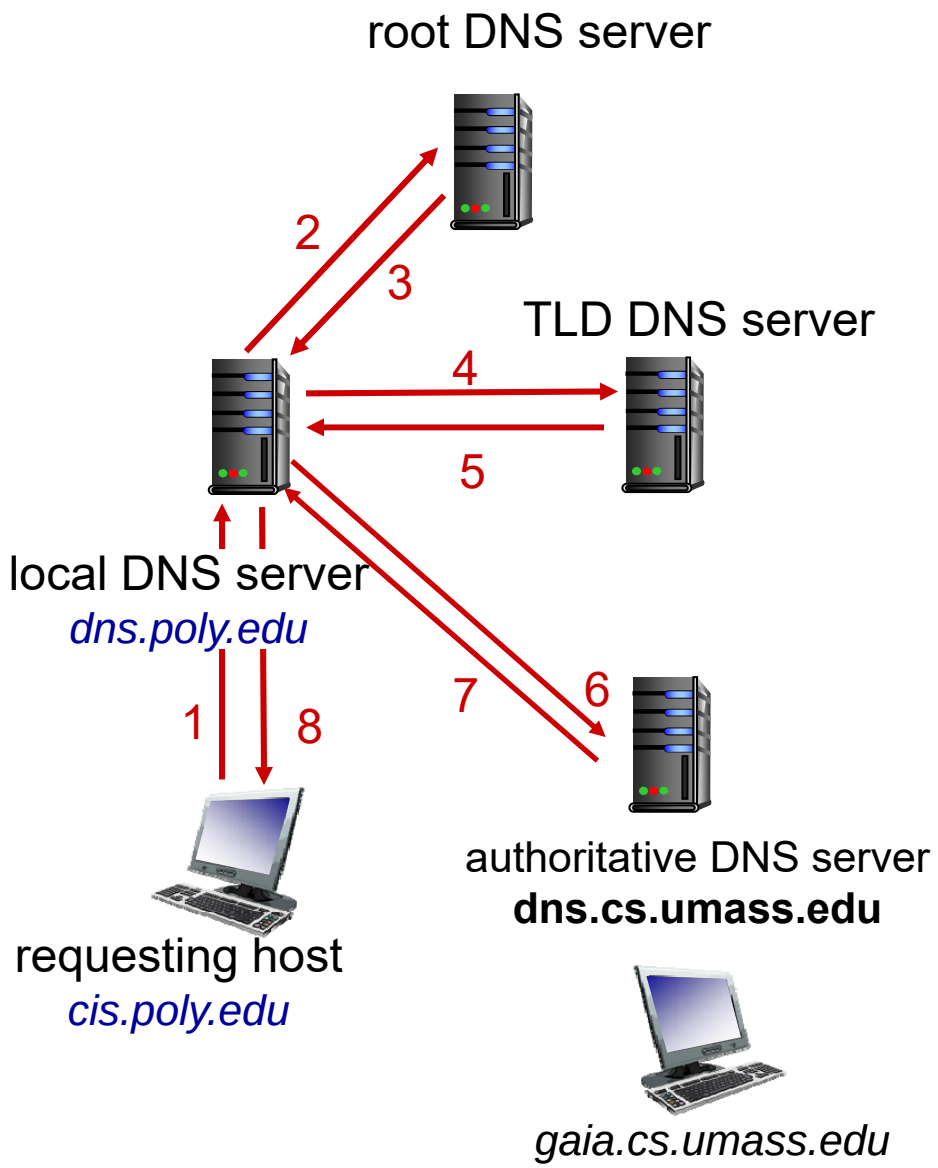


Example

- Host at cis.poly.edu wants IP address for gaia.cs.umass.edu

Iterated query:

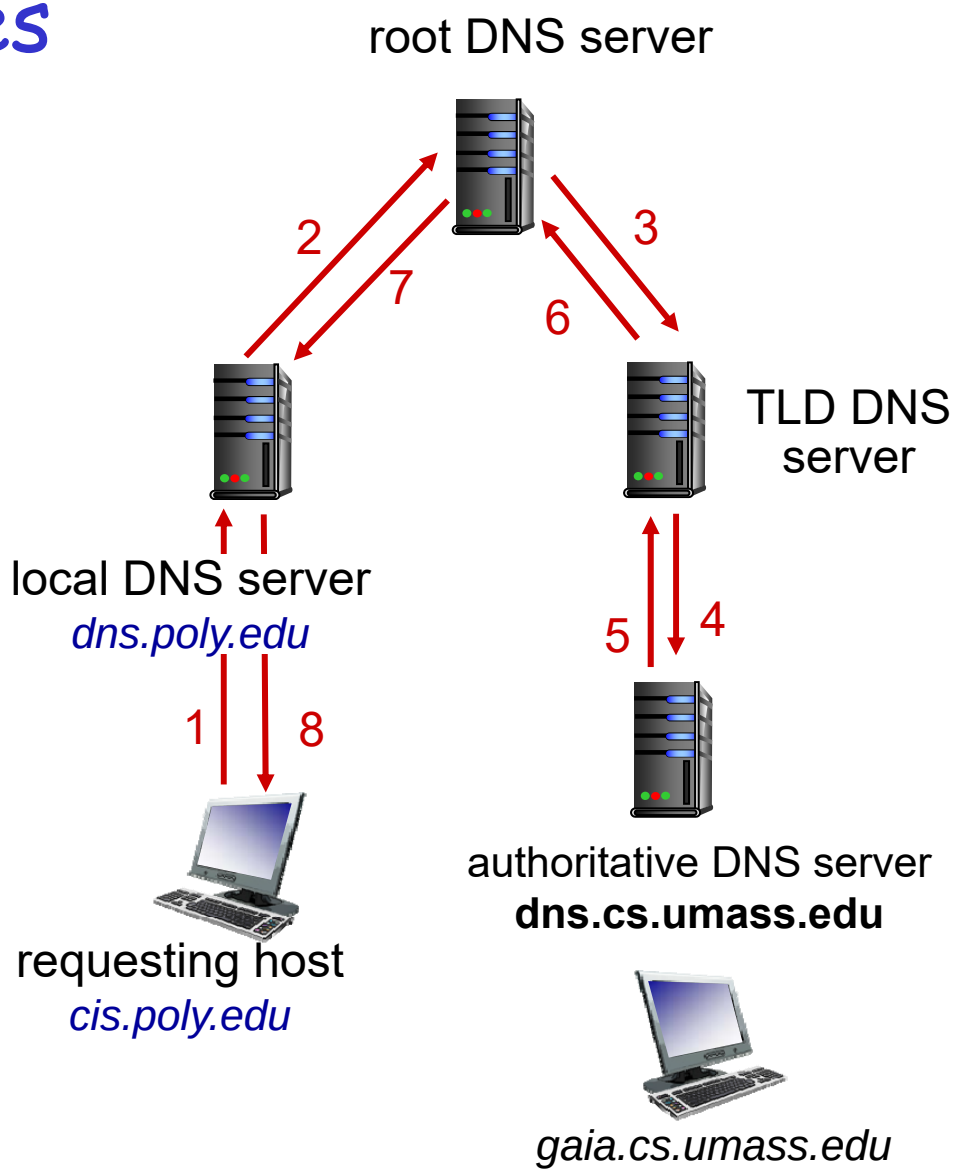
- Contacted server replies with name of server to contact
  - "I don't know this name, but ask this server"
- In practice, use this pattern
  - One recursive
    - (1,8)
  - and subsequent iterative
    - 2 through 7



Recursive Queries

Recursive query:

- Query asks server to obtain mapping on its behalf
- Puts burden of name resolution on contacted name server
- Heavy load?



DNS: Caching and Updating Records

- Once (any) name server learns mapping, it *caches* mapping
  - Cache entries timeout (disappear) after some time (TTL)
    - Time varies greatly
  - TLD servers typically cached in local name servers
    - Thus root name servers not often visited

- Your host also maintains a DNS cache
  - **`ipconfig /displaydns`**
- Can delete current cache
  - **`ipconfig /flushdns`**

DNS Records

DNS: distributed db storing resource records (RR)

RR format: **(name, value, type, ttl)**

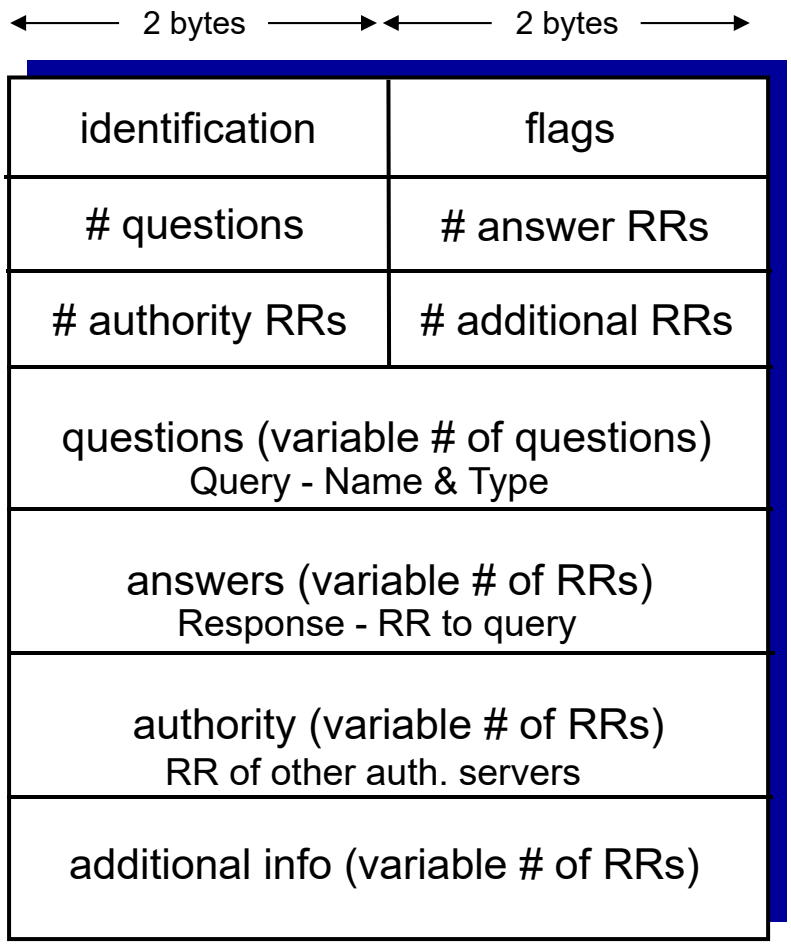
- Type=A
  - **name** is hostname
  - **value** is IPv4 address
  - (relay1.bar.foo.com, 145.37.93.126, A)
  - IPv6 uses a *AAAA* record
- Type=NS
  - **name** is domain (e.g., foo.com)
  - **value** is IP address of authoritative name server for this domain
  - (foo.com, dns.foo.com, NS)
- Type=CNAME
  - **name** is alias name for some "canonical" (the real) name
    www.ibm.com is really
    servereast.backup2.ibm.com
  - **value** is canonical name
  - (foo.com, relay1.bar.foo.com, CNAME)
- Type=MX
  - **value** is name of mailserver associated with **name**
  - (foo.com, mail.bar.foo.com, MX)

DNS Protocol, Messages

Query & reply messages both have the same message format

msg header

- Identification:
  - 16 bit # for query
  - reply to query uses same #
- Flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



DNS Protocol, Message Example

```
C:\Users\bmullins>nslookup www.mit.edu
Server:  dc-lissard2-02.lissard2.local
Address:  10.1.2.2

Non-authoritative answer:
Name:    e9566.b.akamaiedge.net
Address:  23.212.28.78
Aliases:  www.mit.edu
          www.mit.edu.edgekey.net
```

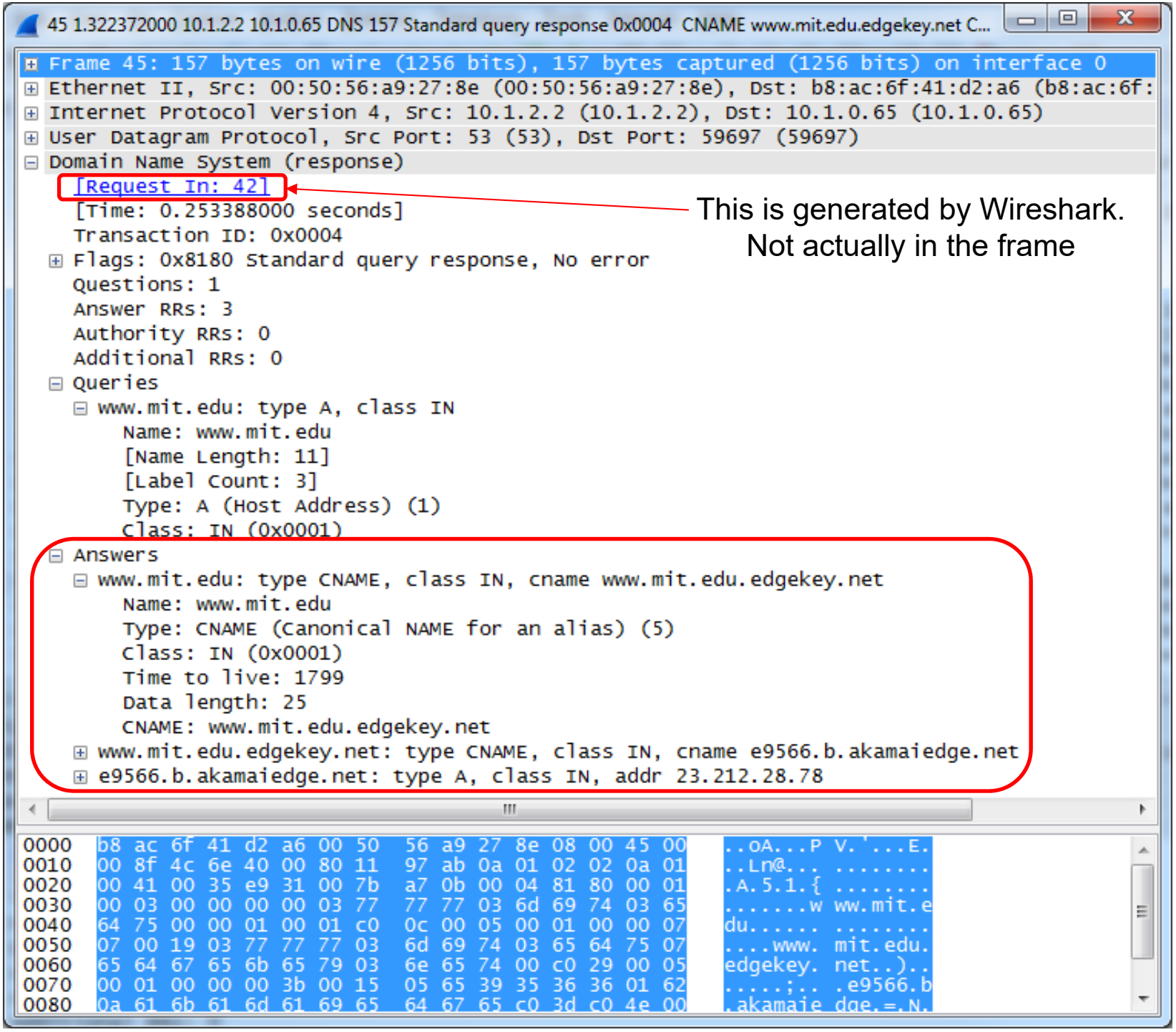
Next slide

42 1.068984000 10.1.0.65 10.1.2.2 DNS 71 Standard query 0x0004 A www.mit.edu

```
Frame 42: 71 bytes on wire (568 bits), 71 bytes captured (568 bits) on interface 0
Ethernet II, Src: b8:ac:6f:41:d2:a6 (b8:ac:6f:41:d2:a6), Dst: 00:50:56:a9:27:8e (00:50:
Internet Protocol Version 4, Src: 10.1.0.65 (10.1.0.65), Dst: 10.1.2.2 (10.1.2.2)
User Datagram Protocol, Src Port: 59697 (59697), Dst Port: 53 (53)
Domain Name System (query)
    [Response In: 45]
    Transaction ID: 0x0004
  Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  Queries
    www.mit.edu: type A, class IN
        Name: www.mit.edu
        [Name Length: 11]
        [Label Count: 3]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
```

```
0000  00 50 56 a9 27 8e b8 ac  6f 41 d2 a6 08 00 45 00   .PV.'... oA....E.
0010  00 39 17 80 00 00 80 11  00 00 0a 01 00 41 0a 01   .9...... .....A..
0020  02 02 e9 31 00 35 00 25  16 7b 00 04 01 00 00 01   ...1.5.% .{......
0030  00 00 00 00 00 00 03 77  77 77 03 6d 69 74 03 65   .......w ww.mit.e
0040  64 75 00 00 01 00 01                              du.....
```

DNS Protocol, Message Example



This is generated by Wireshark.
Not actually in the frame

Inserting Records into DNS

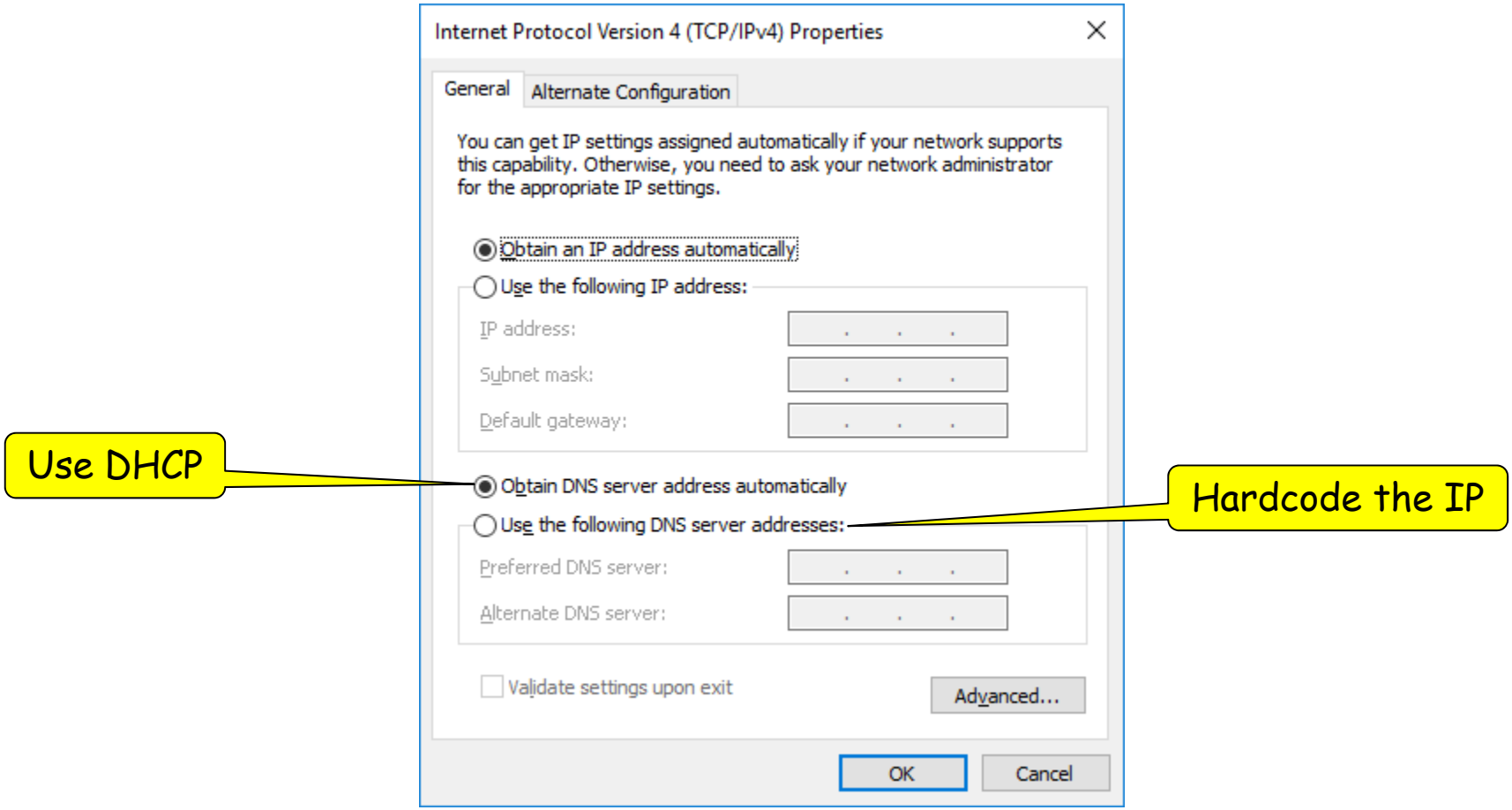
- Example: just created startup company "Network Utopia"
- Register name **networkuptopia.com** at a DNS registrar (e.g., GoDaddy, Network Solutions)
  - Need to provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
  - Registrar inserts two RRs into the .com TLD servers:

(networkutopia.com, dns1.networkutopia.com, NS) ← alias
(dns1.networkutopia.com, 212.212.212.1, A) ← mapping

- Put in authoritative server
  - Type A record for www.networkuptopia.com
  - Type MX record for networkutopia.com mail server

Bootstrapping DNS

- How does a host contact the name server if all it has is the name and no IP address?
  - IP address of at least 1 name server must be given a priori
    - Hardcode the IP or use DHCP



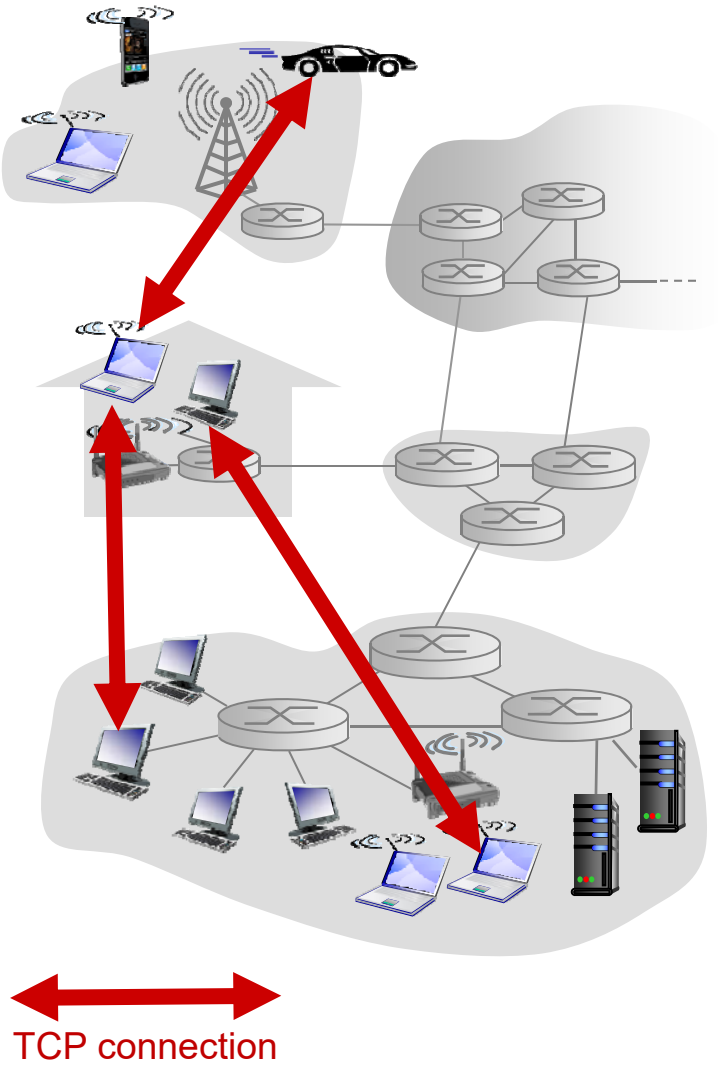
Chapter 2: Application Layer

- 2.1 Principles of Network Applications
- 2.2 Web and HTTP
- FTP
- 2.3 Electronic Mail
  - SMTP, POP3, IMAP
- 2.4 DNS
- 2.5 P2P Applications
- 2.6 Video Streaming and Content Distribution Networks
- 2.7 Socket Programming: Creating Network Applications

Pure P2P Architecture

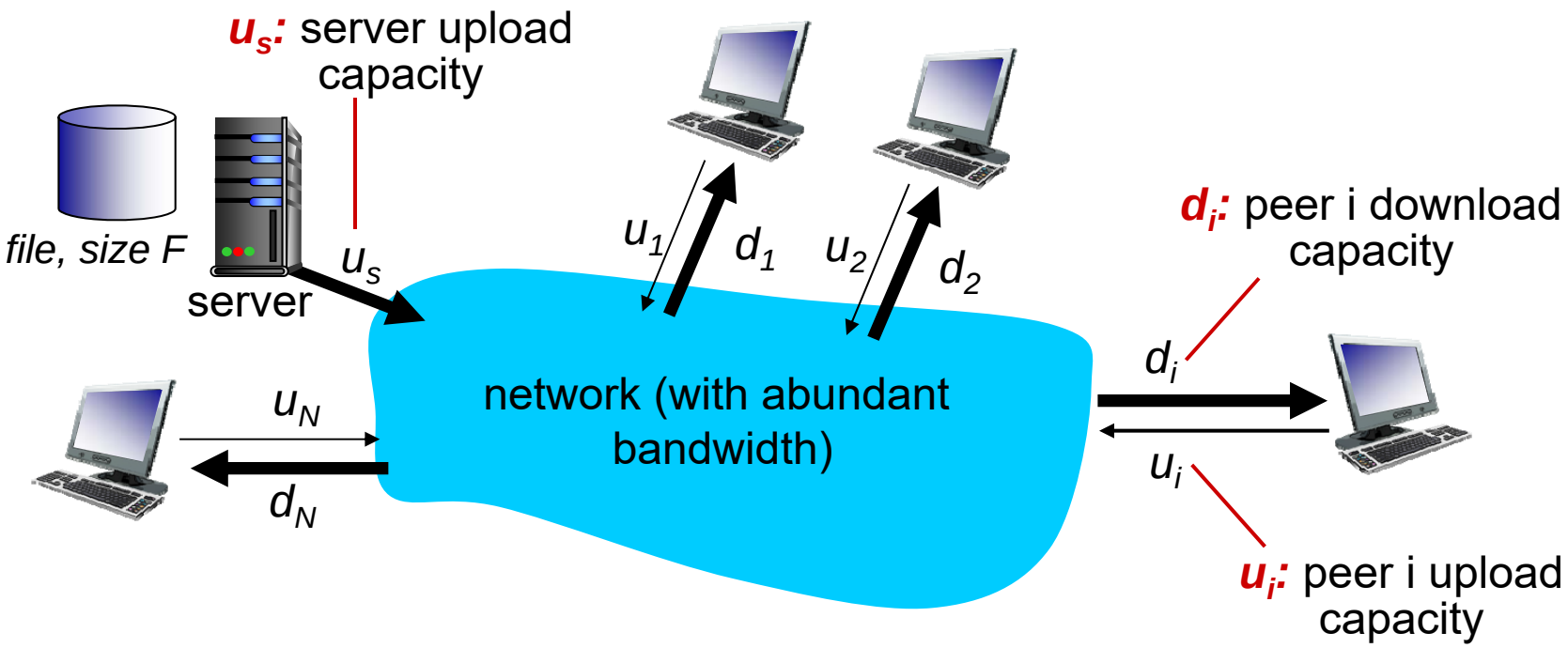
- No always-on server
- Arbitrary end systems directly communicate
- Peers are intermittently connected and change IP addresses
- Examples:
  - File distribution (BitTorrent)
  - VoIP (Skype)

Highly scalable



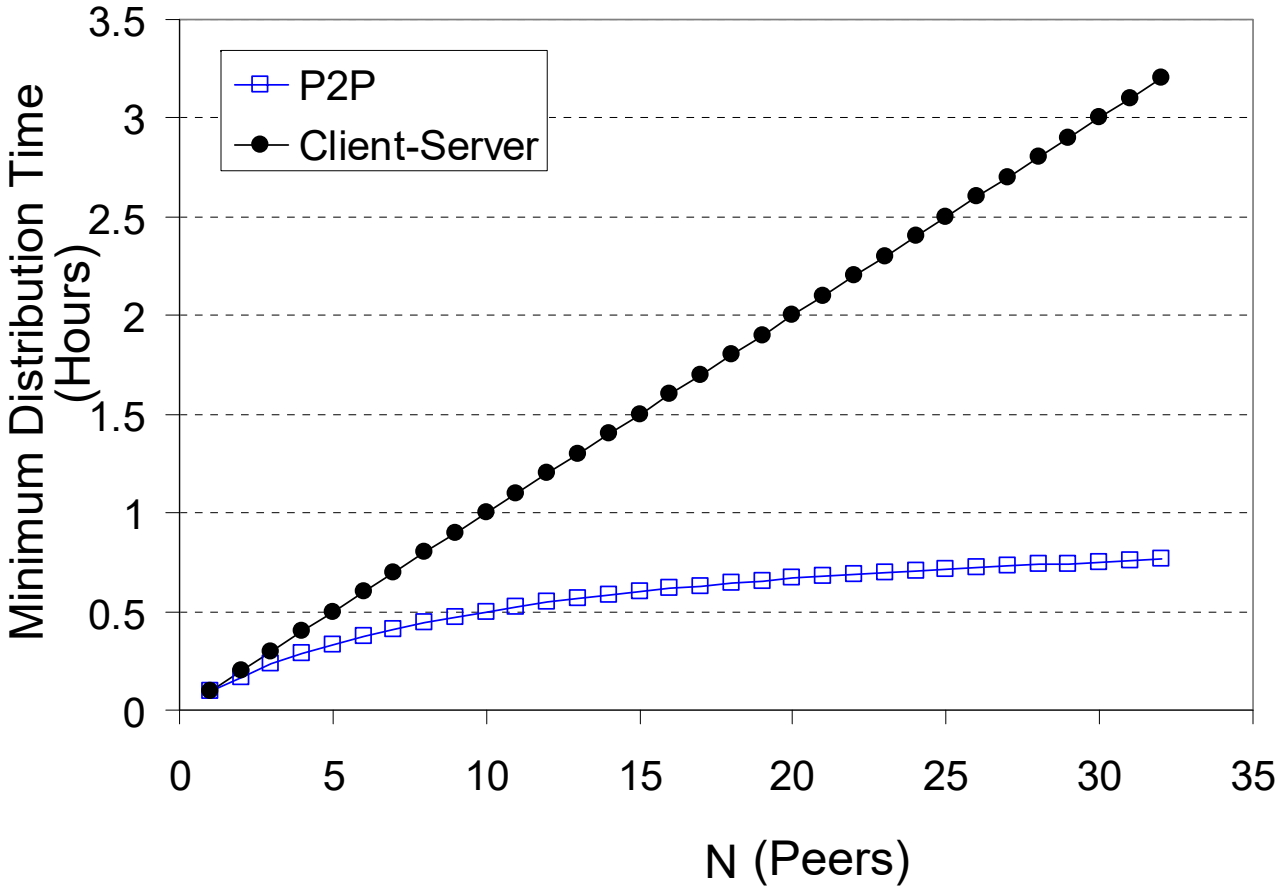
File distribution: Client-Server vs. P2P

- How much time is required to distribute file (size F) from one server to N peers?
  - Peer upload/download capacity is limited resource



Client-server vs. P2P: Example

- Client upload rate: u, Assume: Upload file from client: F/u = 1 hour
- Server upload rate: u_s = 10u → file upload in 0.1 hour (6 min)
- Time to get file to all N peers



P2P File Sharing

Example

- Alice runs P2P client app on her computer
- Intermittently connects to Internet
  - New IP address for each connection
- Asks for "The Wall"
- Application displays other peers that have copy of "The Wall"

- Alice chooses one peer → Bob
- File is copied from Bob's PC to Alice's notebook using TCP
- While Alice downloads, she also uploads to other users
- Alice's peer is both a Web client and a transient Web server

All peers are servers = highly scalable!

BitTorrent: Multiple Uploaders

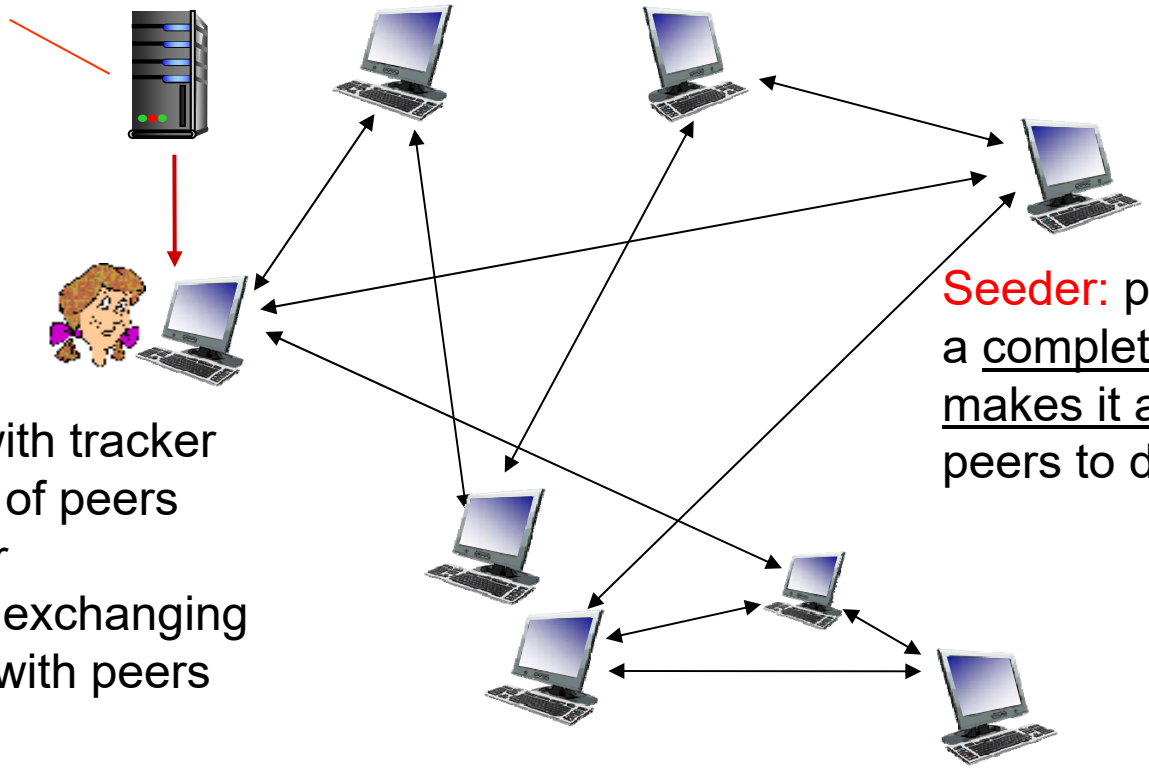
- Designed in 1999 by Bram Cohen to take advantage of high download speeds while mitigating slow upload rates
- Aggregation of many slow upload speeds = FAST DOWNLOADS!
- Users who download pieces of files automatically upload pieces they already have

P2P File Distribution: BitTorrent

- File divided into 256KB chunks
- Peers in torrent send/receive file chunks

Tracker: tracks peers participating in torrent

Torrent or swarm: group of peers exchanging chunks of a file



Alice arrives
- Registers with tracker
- Obtains list of peers from tracker
- and begins exchanging file chunks with peers in torrent

Seeder: peer that possesses a complete copy of a file and makes it available for other peers to download

P2P File Distribution: BitTorrent

- Peer joining torrent:
  - Registers with tracker to get list of peers, connects to subset of peers ("neighbors")
  - Initially has no chunks, but accumulates them over time from other peers
- While downloading, peer uploads chunks to other peers
- Peer may change peers with whom it exchanges chunks
  - Churn: peers may come and go
- Once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

BitTorrent: Requesting, Sending Chunks

Requesting chunks:

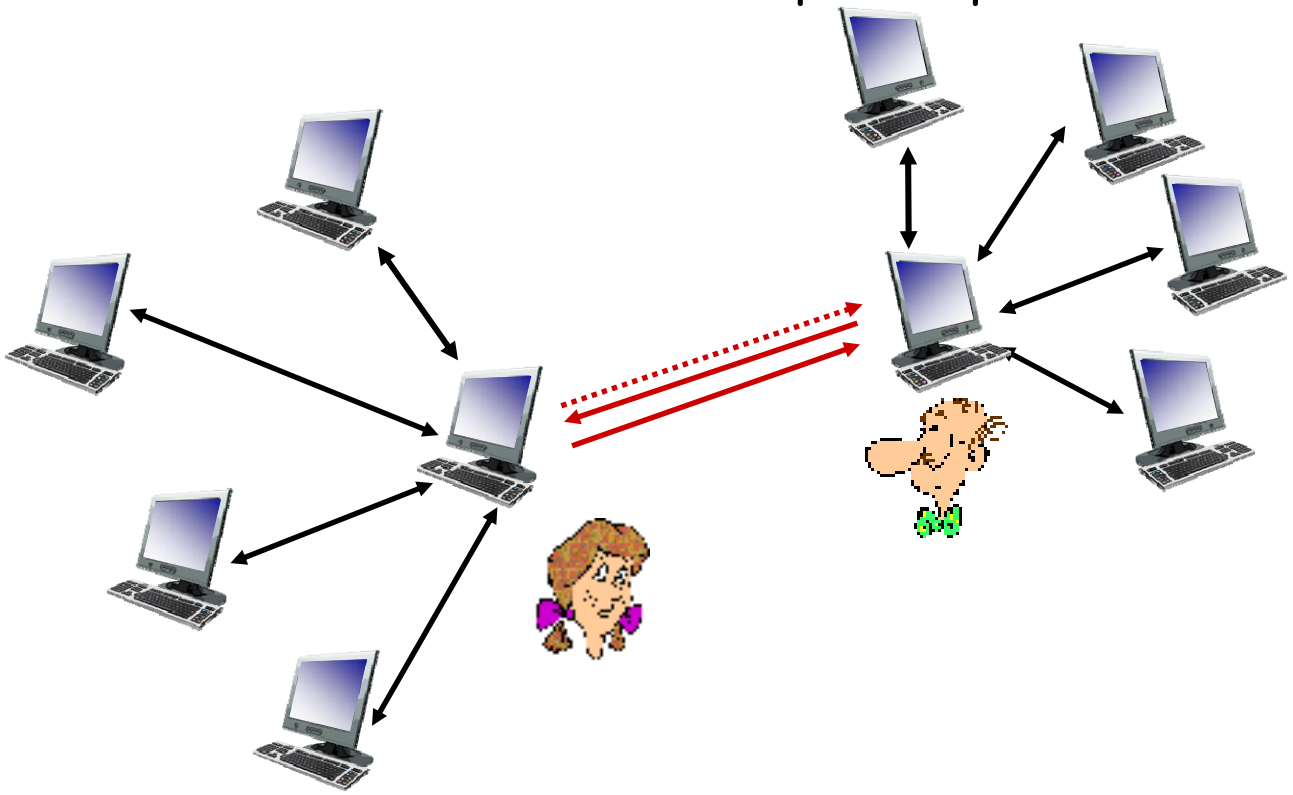
- At any given time, different peers have different subsets of file chunks
- Periodically, Alice asks each peer for list of chunks they have
- Alice requests missing chunks from peers, rarest first

Sending chunks: tit-for-tat

- Alice sends chunks to those four peers currently sending her chunks at highest rate
  - Other peers are choked by Alice (do not receive chunks from her)
  - Re-evaluate top 4 every 10 secs

BitTorrent: tit-for-tat

- Every 30 secs: Alice randomly selects another peer, starts sending chunks
  - Alice "optimistically unchokes" Bob
  - Alice becomes one of Bob's top-four providers; Bob reciprocates
  - Bob becomes one of Alice's top-four providers



Chapter 2: Application Layer

- 2.1 Principles of Network Applications
- 2.2 Web and HTTP
- FTP
- 2.3 Electronic Mail
  - SMTP, POP3, IMAP
- 2.4 DNS
- 2.5 P2P Applications
- 2.6 Video Streaming and Content Distribution Networks
- 2.7 Socket Programming: Creating Network Applications

Socket Programming

Goal: Learn how to build client/server applications that communicate using sockets

Socket API

- Socket is explicitly
  - created,
  - used,
  - then released by apps
- Client/Server paradigm

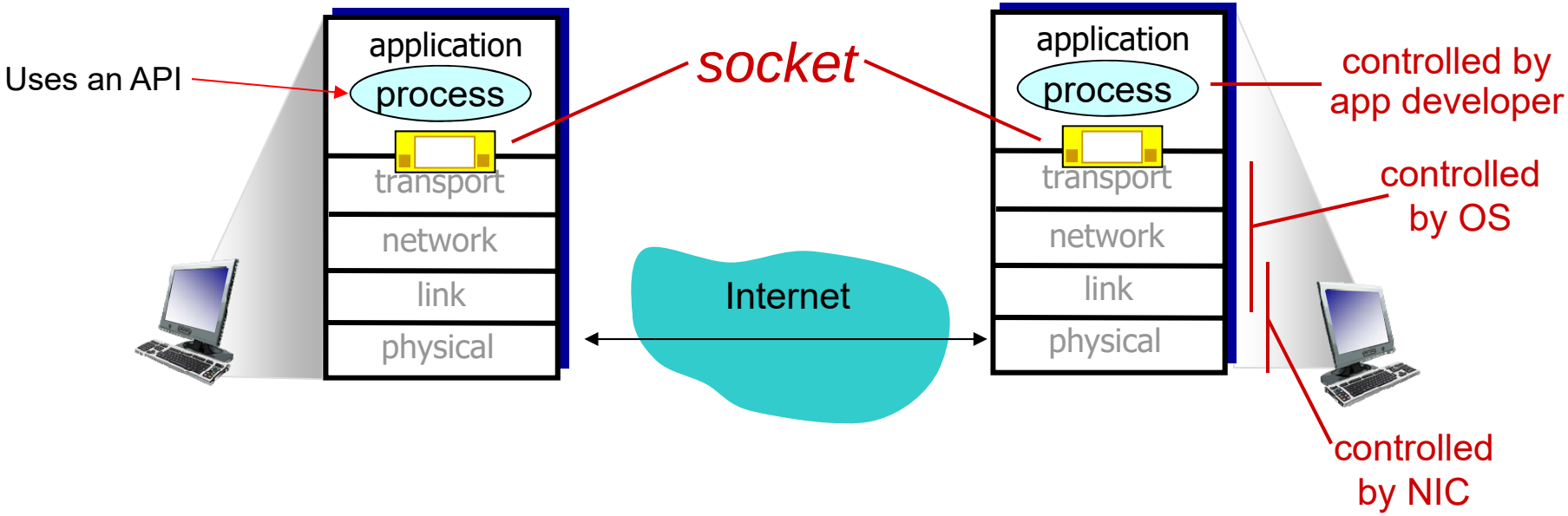
socket

A *host-local*, *application-created*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another application process

Socket Programming

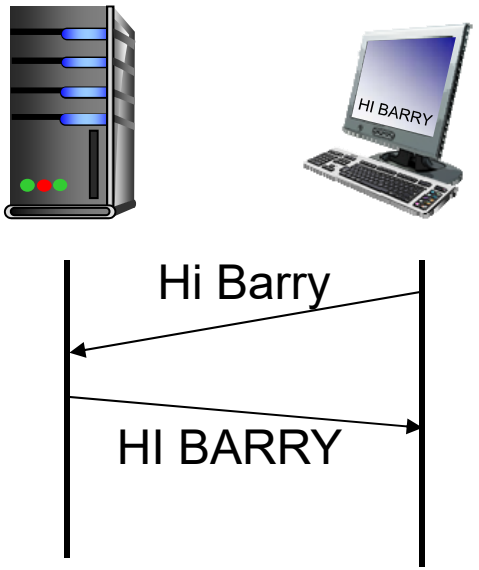
Socket: a door between application process and end-to-end transport protocol (UDP or TCP)

- Two socket types for two transport services:
  - UDP: unreliable datagram
  - TCP: reliable, byte stream-oriented



Application Example

- Client reads a line of characters (data) from its keyboard and sends the data to the server
- Server receives the data and converts characters to uppercase
- Server sends the modified data to the client
- Client receives the modified data and displays the line on its screen



Socket Programming With UDP

UDP: transmitted data may be lost or received out-of-order

UDP: no "connection" between client & server

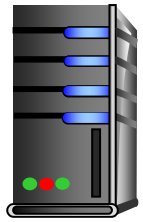
- No handshaking before sending data
- Sender application explicitly attaches to each packet
  - IP destination address
  - Port #
- Rcvr extracts sender IP address and port # from packet and uses them to reply to the sender

Application viewpoint:

- UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server

Client/Server Socket Interaction: UDP

Server (running on serverIP)



create socket, port = x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)

wait for incoming datagram

read datagram from serverSocket
extract IP, port

create datagram with client IP address, port number; send to serverSocket

Client

create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)

create datagram with serverIP and port=x; send datagram via clientSocket

read datagram from clientSocket

close
clientSocket

AF_INET - Address Family used for the socket you're creating (e.g., Internet Protocol address-INET)

Example App: UDP Server

Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print ('The server is ready to receive')
while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

```
netstat -nao | find /i "listening"
```

create UDP socket

bind socket to local port number 12000

loop forever

Read from UDP socket into message, getting client's address (client IP and port)

send upper case string back to this client

Python uses indentation to delineate blocks of code

Notepad++ can parse Python code and automatically display code blocks

Example App: UDP Client

Python UDPClient

include Python's socket library

```
from socket import *
```

Change to the name or IP address of the server

```
serverName = 'hostname'
```

Network layer - IP

Transport layer - UDP

```
serverPort = 12000
```

create UDP socket for client

```
clientSocket = socket(AF_INET, SOCK_DGRAM)
```

get user keyboard input

```
message = input('Input lowercase sentence:')
```

Attach server name, port to message; send into socket

```
clientSocket.sendto(message,(serverName, serverPort))
```

read reply characters from socket into string. serverAddress contains server IP and port

```
modifiedMessage, serverAddress =
                                clientSocket.recvfrom(2048)
```

print out received string

```
print modifiedMessage
```

close socket

```
clientSocket.close()
```

Addressing

- For TCP/IP socket communication, generally need 4 parameters:
  - Source Identifier (32-bit IP address)
  - Source Port (16-bit)
  - Destination Identifier (32-bit IP address)
  - Destination Port (16-bit)

- UDP uses only Destination (IP+Port) for demultiplexing
- TCP uses Source + Destination
  - (quadruple: Src IP, Src Port, Dest IP, Dest Port)

Socket Programming with TCP

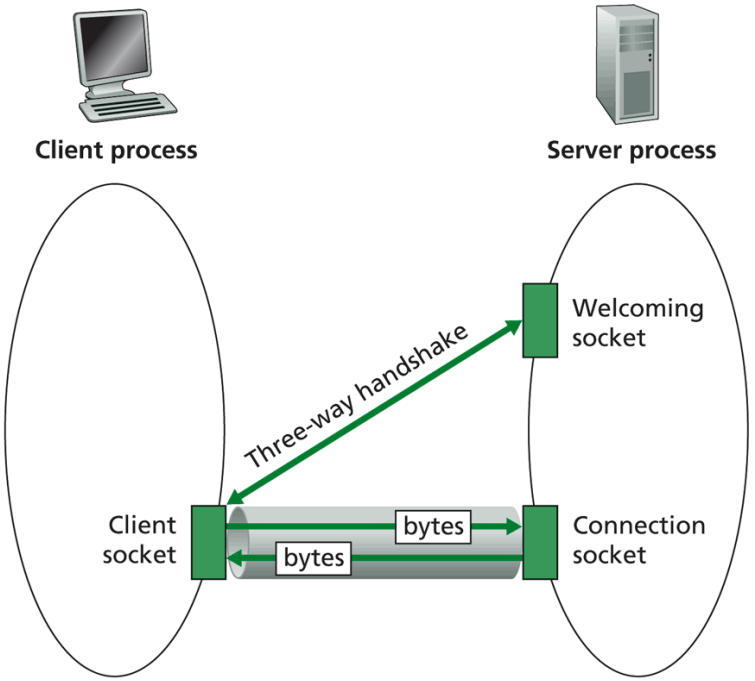
Client must contact server

- Server process must first be running
- Server must have created socket (door) that welcomes client's contact

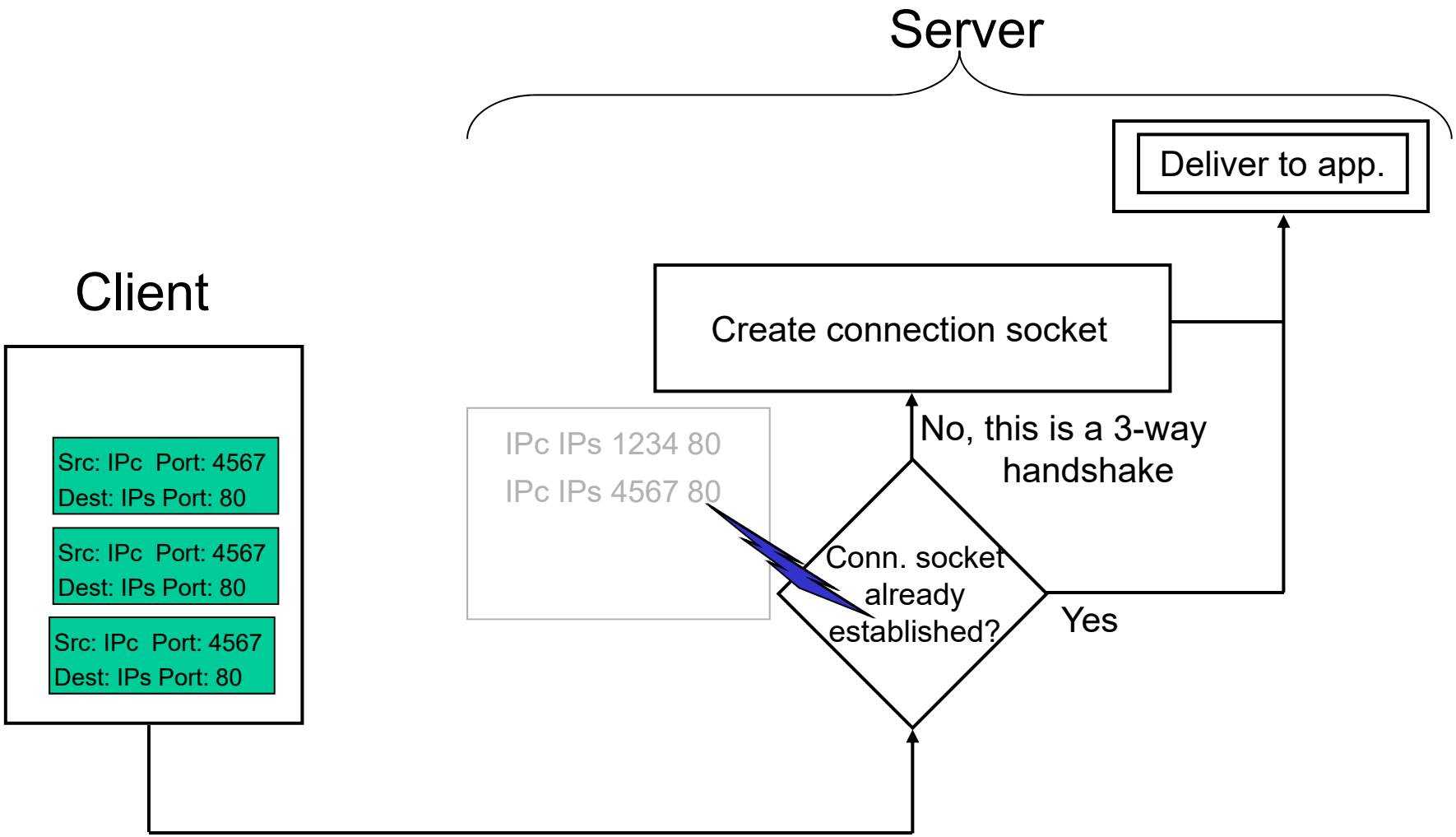
Client contacts server by:

- Creating client TCP socket
- Specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

- When contacted by client, server TCP **creates new connection socket** for server process to communicate with client
  - Allows server to talk with multiple clients
  - Source port numbers used to distinguish clients



Socket Programming with TCP



Client/Server Socket Interaction: TCP

Server (running on **hostid**)

Client

create socket for incoming request on port=**x**:
serverSocket = socket()

Welcoming socket

wait for incoming connection request
connectionSocket = serverSocket.accept()

Connection socket

TCP connection setup

create socket, connect to **hostid**, port=**x**
clientSocket = socket()

send request using clientSocket

read request from connectionSocket

write reply to connectionSocket

read reply from clientSocket

close connectionSocket

close clientSocket

Example App: TCP Server

Python TCPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
while 1:
    print ('The server is ready to receive')
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

server waits on accept() for incoming requests, new socket created on return

read bytes from socket (not an address as in UDP)

close connection to this client (but *not* welcoming socket)

Welcoming socket

Connection socket

Example App: TCP Client

Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server: ', modifiedSentence
sentence = input('Press enter to exit ...')
clientSocket.close()
```

Change to the name or IP address of the server

Transport layer - TCP

create TCP socket for server, remote port 12000

No need to attach server name, port

I added so you can see the server's response