# CSCE 463/612
# Networks and Distributed Processing
# Fall 2020

## Application Layer

Dmitri Loguinov
Texas A&M University

September 3, 2020

# Updates

- URLs to try the parser on →

```
http://x.com/path:900
http://x.com?script:900/
http://x.com?script/
http://x.com:8800?script:/
```

- Quiz next time (entire class), variation on problems 5-33 at the end of chapter 1
  - More questions based on my programming tutorial (pointers, bits ops, debugging, Windows datatypes)
- Examine this fragment:

```
#define HUGE 10000000         // 10 MB
char buf [HUGE], *ptr = buf;
while((bytes = recv (sock, ptr, 100, 0)) != 0)
          ptr += bytes;

*ptr = NULL;
len = ptr – buf;
```

- Issues include
  - Inefficient recv
  - Buffer overflow when page exceeds 10 MB
  - Deadlock on errors

# Robots.txt

- Websites are crawled by many automated programs
  - This potentially consumes large volumes of traffic
- Besides bandwidth, concerns arise about protected or human-only portions of websites
  - Shopping carts, registration pages, posting into forums
- Webmasters need a mechanism to indicate prohibited path prefixes within their sites
  - These are given in /robots.txt

```
User-agent: *
Disallow: /search
Disallow: /sdch
Disallow: /groups
Disallow: /images
Disallow: /catalogs
Allow: /catalogs/about
Allow: /catalogs/p?
Disallow: /catalogues
```

- Directives are parsed in order, until first match
  - Algorithm has become ambiguous in recent years: Google crawlers use the longest-prefix match

# Robots.txt 2

- Despite being around since 1994, robots.txt is not a standard, but rather a suggestion on politeness
  - See http://robotstxt.org
- Extensions to robots.txt (even less official)
  - Crawl-delay specifies the # of seconds between visits
  - Sitemap points to an XML file that lists all available documents
  - Wildcards in directory paths (* and $ = ends with)

```
User-agent: *
Disallow: /*.asp$
Disallow: /sdch/*.php
Crawl-delay: 64
Sitemap: http://www.google.com/sitemaps_webmasters.xml
```

- How often should robots.txt be reloaded?
  - Original spec doesn’t say; Google uses 1 day by default

# Chapter 2: Roadmap

2.1 Principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 Electronic Mail

- SMTP, POP3, IMAP

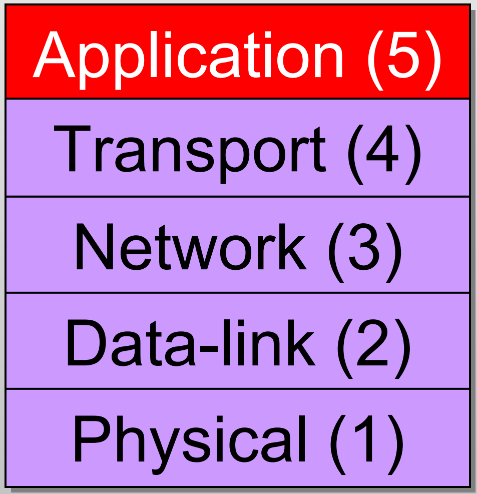
2.5 DNS

2.6 P2P file sharing

2.7 Socket programming with TCP

2.8 Socket programming with UDP

2.9 Building a Web server



Application (5)

Transport (4)

Network (3)

Data-link (2)

Physical (1)

# Some Network Applications

- E-mail
- Remote login
- Web
- Instant messaging
- P2P file sharing
- Multi-user network games
- Streaming video
- Internet telephone
- Thermostat
- House alarm

- Real-time video conferencing
- Massively parallel computing
- Phones, tablets
- Internet fridge, TV

Mon, Jan 3, 2011 06:09 PM

Take out the garbage tonight

Do your homework

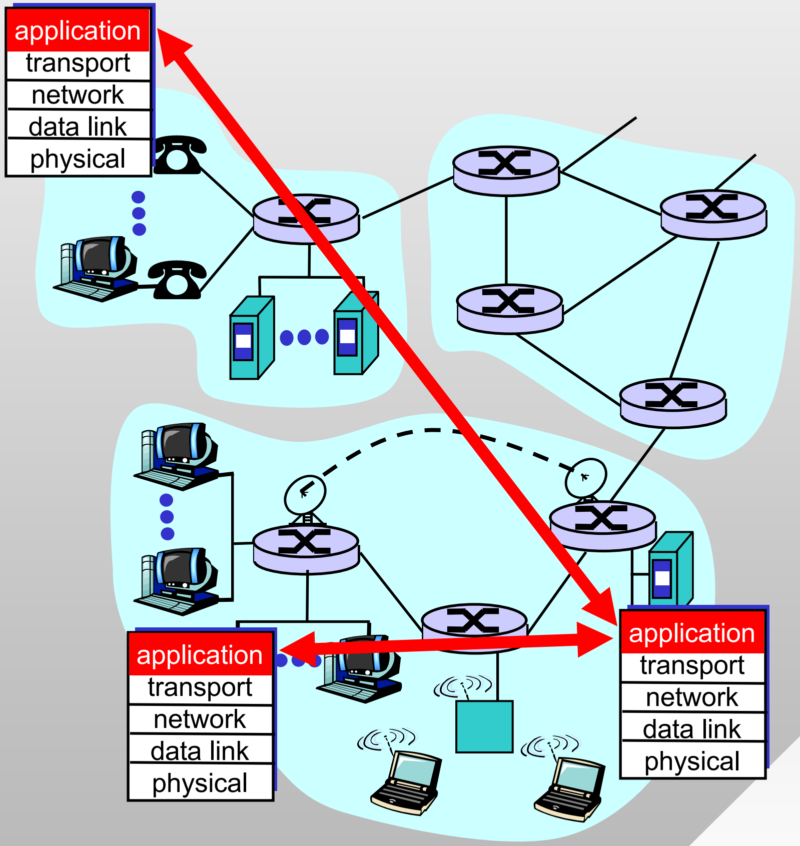
# Creating a Network Application

Programs that

- Usually interact with user
- Communicate over a network
- E.g., Web server software communicates with browser software

No software written for devices in network core

- Network core devices do not function at app layer
- This design allows for rapid application development



# 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

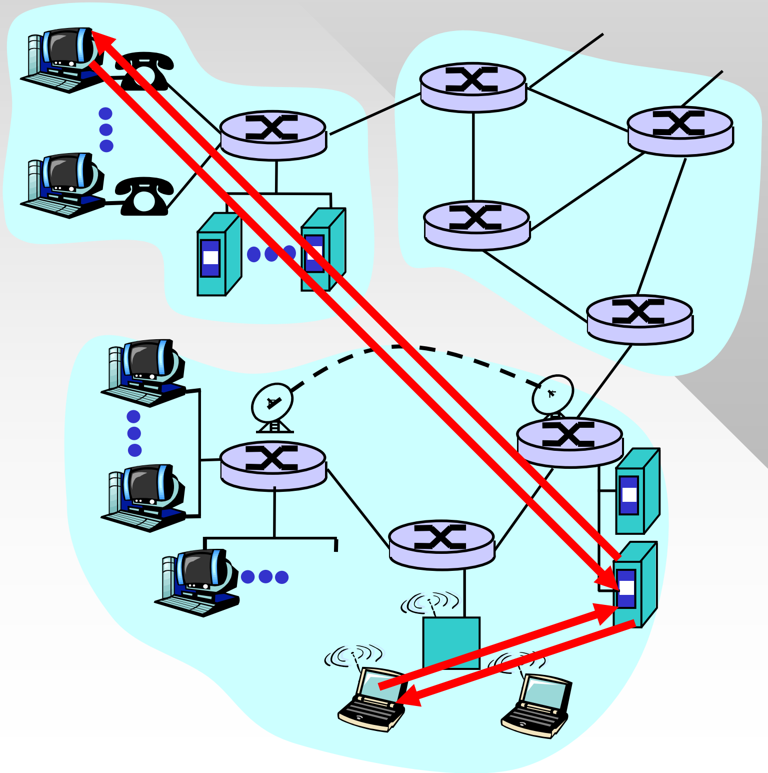
2.7 Socket programming with TCP

2.8 Socket programming with UDP

2.9 Building a Web server

# Communication Principles

- Three architectures
  - Client-server
  - Peer-to-peer (P2P)
  - Hybrid



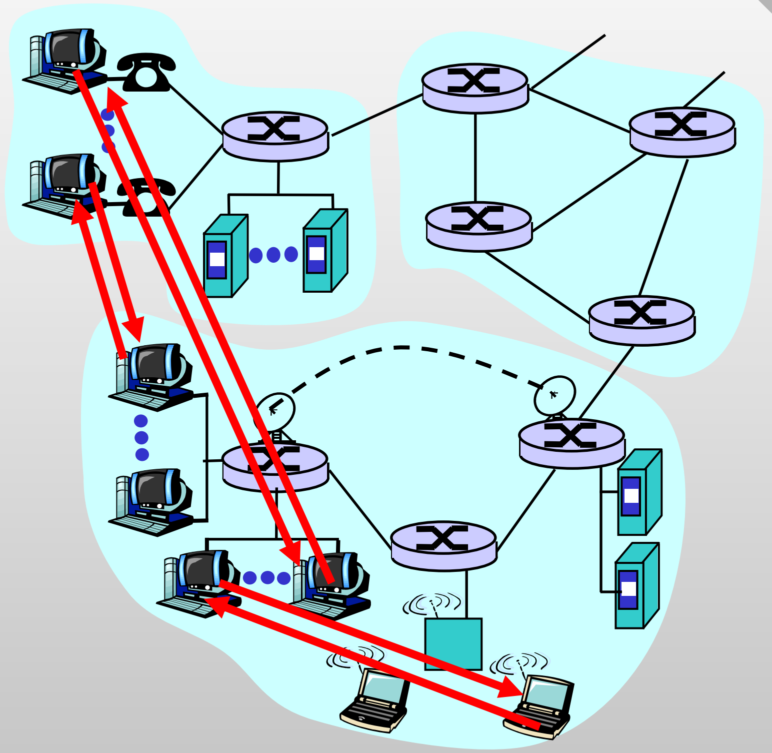
Server:

- An always-on host
- Permanent IP address or hostname
- Server farms for scaling

Clients:

- May be intermittently connected
- May have dynamic IP addresses and hostnames
- Do not communicate directly with each other, only talk to servers

# P2P Architecture



- No always-on server
- Arbitrary end systems directly communicate
- Peers are intermittently connected and change IP addresses/hostnames
- Example: Gnutella
  - Distributed graph between users over TCP connections
- Highly scalable: assume 6M users with 1GB of shared data and 500 Kbps upstream bandwidth
  - 6 PB of storage, 3 Tbps bandwidth for free
- Downside – difficult to provide reliable service

# Hybrid Architecture

Napster

- File transfer P2P, but search is centralized
  - Peers register content at central server
  - Peers query same central server to locate content

Instant messaging

- Login and chatrooms are centralized
  - User registers its IP address with central server when it comes online
  - User contacts central server to find IP addresses of buddies or participate in chatrooms
- Chatting between two users is P2P
  - E.g., Skype relays data through other live peers

# Process Communication

- Process: program running within a host
- Within same host, two processes communicate using inter-process communication (semaphore, mutex, pipe, shared memory)
- Processes in different hosts communicate by exchanging messages

- Client: process that initiates communication
- Server: process that waits to be contacted

- Applications with P2P architecture act as both client & server

# 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

# Web and HTTP

## Terminology

- Web page consists of a base HTML-file that may include references to external objects
  - Examples of objects: JPEG image, Java applet, audio file, video stream, or flash animation
- Each object is addressable by a URL (Uniform Resource Locator) with the HTTP scheme

```
http://[user:pass@]host[:port][/path][?query][#fragment]
```

  - Username/password not used often anymore
  - Fragement specifies portion of HTML for browser to jump to
  - Query provides input arguments to scripts

# HTTP Overview

- HTTP: HyperText Transfer Protocol
  - HTTP 1.0: RFC 1945 (1996)
  - HTTP 1.1: RFC 2068 (1997), RFC 2616 (1999)
  - HTTP 2: RFC 7540 (May 2015)
- Nonpersistent HTTP
  - At most one object is sent over a TCP connection
  - HTTP/1.0 must use nonpersistent HTTP
- Persistent HTTP
  - Multiple objects sent over single TCP connection
  - HTTP/1.1 uses persistent connections by default
  - Field “Connection: close” overrides this behavior

# Nonpersistent HTTP

(contains text, references to 10 jpeg images)

Suppose user enters URL
`www.tamu.edu/someDepartment/home.html`

1a. Client initiates TCP connection to server process at www.tamu.edu using port 80

1b. Server at host www.tamu.edu waiting for TCP connection on port 80 accepts connection, notifies client

2. Client sends HTTP *request message* (containing URL) into TCP socket. Message indicates object /someDepartment/home.html

3. Server receives request, forms *response message* containing requested object, and sends message into its socket

time

# Nonpersistent HTTP (Cont.)

4. Server closes TCP connection

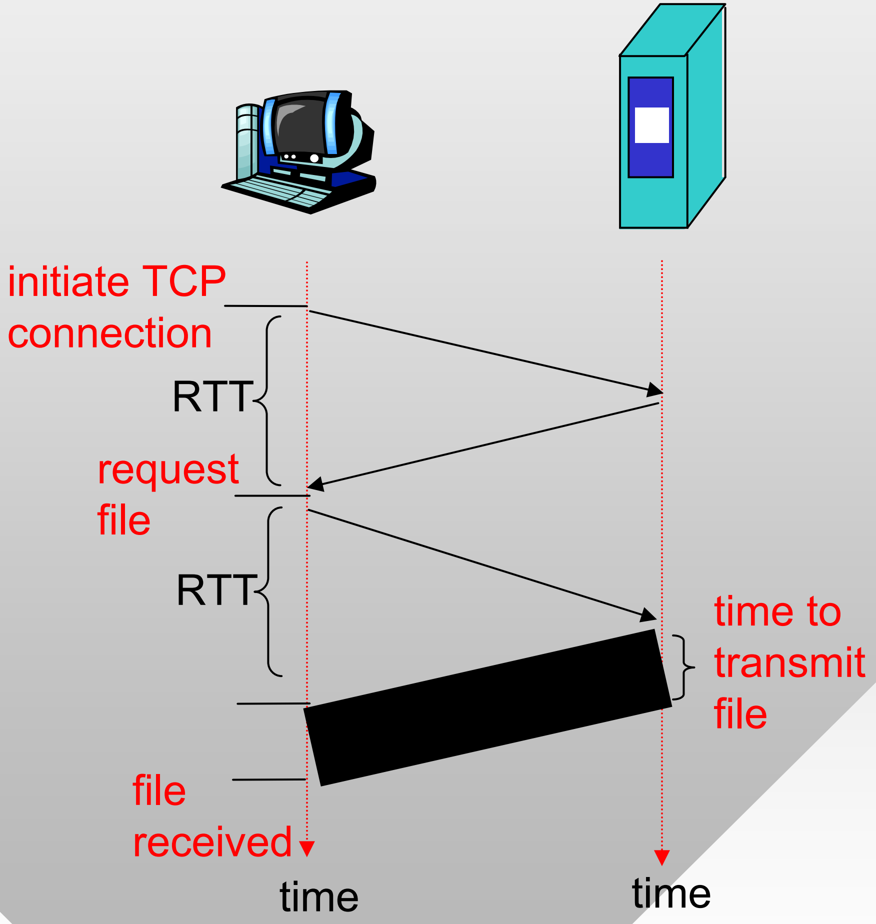
5. Client receives response message containing the html file, displays html. Parsing html file, finds 10 referenced jpeg objects

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

# Response Time Modeling

- RTT (Round-Trip Time):
  - Delay for a small packet to travel from client to server and back
- Response time:
  - One RTT to initiate TCP connection
  - One RTT for HTTP request and first few bytes of HTTP response to return
  - File transmission time

total = 2RTT + file load time



# Persistent HTTP

HTTP/2 allows out-of-order replies, fragmentation of objects, and prioritization

## Nonpersistent HTTP issues:

- Requires two RTTs per object
- OS must work and allocate host resources for each TCP connection
- Workaround: browsers open parallel TCP connections to fetch referenced objects

## Persistent  HTTP

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

## Persistent without pipelining:

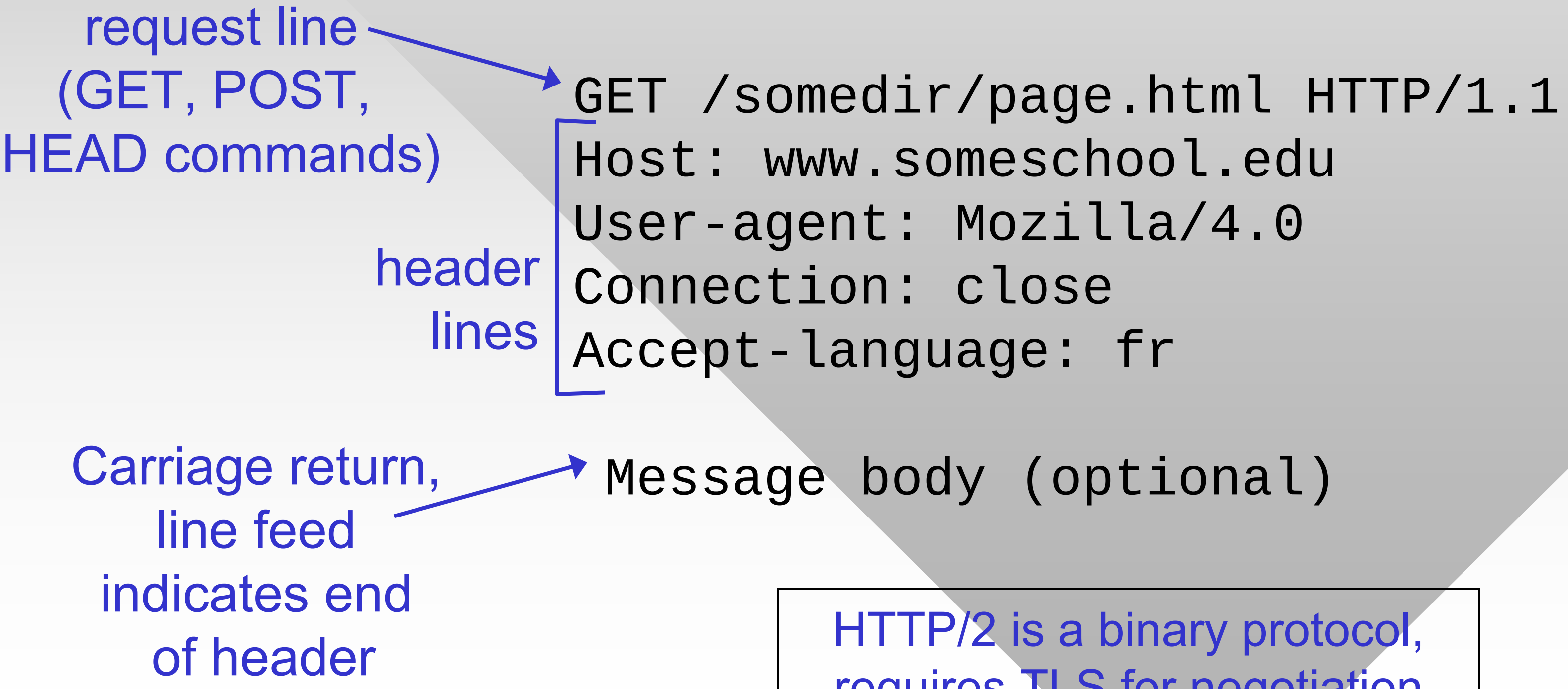
- Client issues new request only when previous response has been received
- One RTT for each referenced object + its transmission time

## Persistent with pipelining:

- Default in HTTP/1.1
- Client sends requests as soon as it encounters a referenced object
- One RTT for all referenced objects + their transmission times

# HTTP Request Message

- Two types of HTTP messages: *request, response*
- HTTP request message:
    - 1.0 and 1.1 use ASCII (human-readable format)



request line (GET, POST, HEAD commands)

header lines

Carriage return, line feed indicates end of header

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr

 Message body (optional)
```

HTTP/2 is a binary protocol, requires TLS for negotiation

# Uploading Form Input

POST method:

- Web page often includes form input
- Input is uploaded to server in entity body
- Used for large amounts of data
  - Data is coded using tuples “field=value”, where + stands for space and & for the field separator

```
POST /map.cgi HTTP/1.0
User-Agent: HTTPTool/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 30

city=College+Station&zip=77843
```

# Uploading Form Input (Cont'd)

URL method:

- Uses the GET command
- Input is encoded in the URL field of request line
  - Append ? to the script path, followed by the URL-coded data
  - GET /path/script.cgi?field1=value1&field2=value2 HTTP/1.0
- For the previous example
  - GET /map.cgi?city=College+Station&zip=77843 HTTP/1.0
- Google example
  - Javascript forces the URL method:
  - www.google.com/search?hl=en&source=hp&q=computer+science&aq=f&aqi=g10&oq=

# Method Types

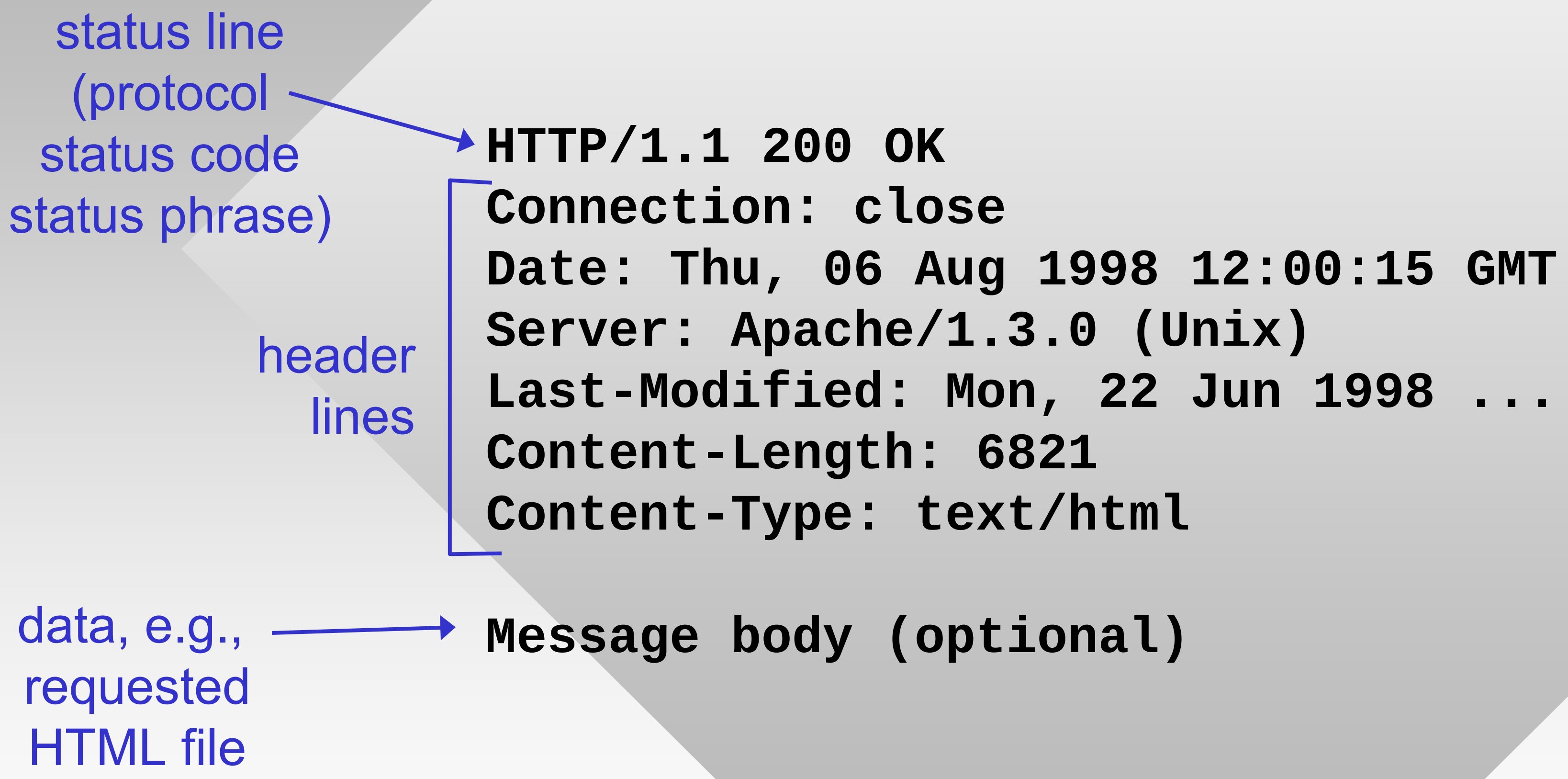
## HTTP/1.0

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

## HTTP/1.1

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

# HTTP Response Message



status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 ...
Content-Length: 6821
Content-Type: text/html

Message body (optional)
```

# HTTP Response Status Codes

- Status code is always in the first line of response
  - Followed by a nice textual explanation
- 200 OK
  - Request succeeded, requested object later in this message
- 301 Moved Permanently
  - Requested object moved, new location specified later in this message (see field Location:)
- 400 Bad Request
  - Request message not understood by server
- 404 Not Found
  - Requested document not found on this server
- 505 HTTP Version Not Supported