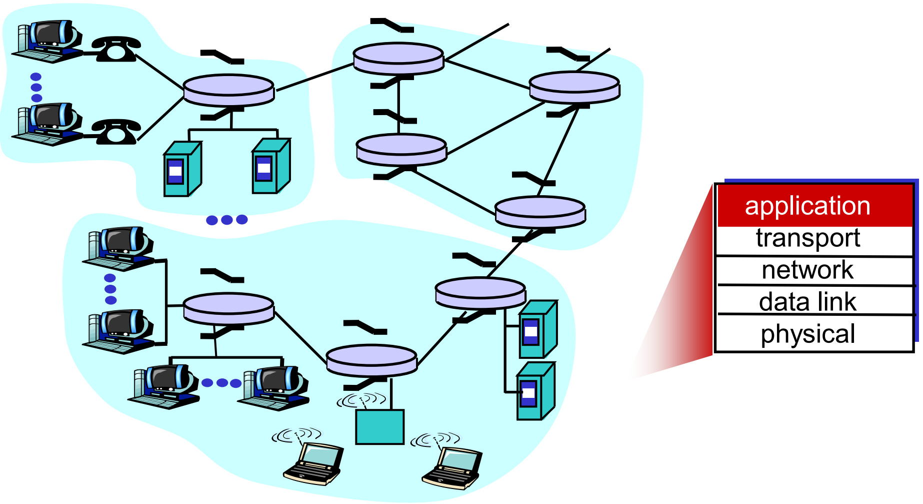
CS 4390
Computer Networks



application
transport
network
data link
physical

Application Layer Overview & HTTP

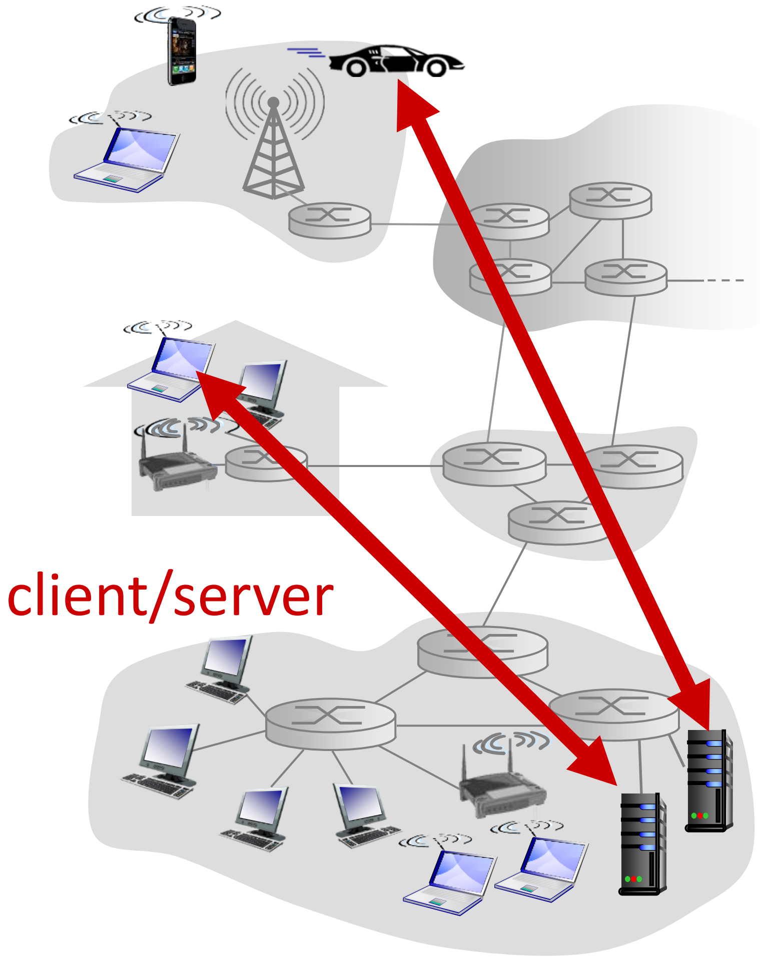
Adapted from "Computer Networking – a Top-Down Approach"

Application Architectures

Common structure of network applications

- client-server
- peer-to-peer (P2P)
- hybrid of client-server and P2P

Client-Server Architecture



server:

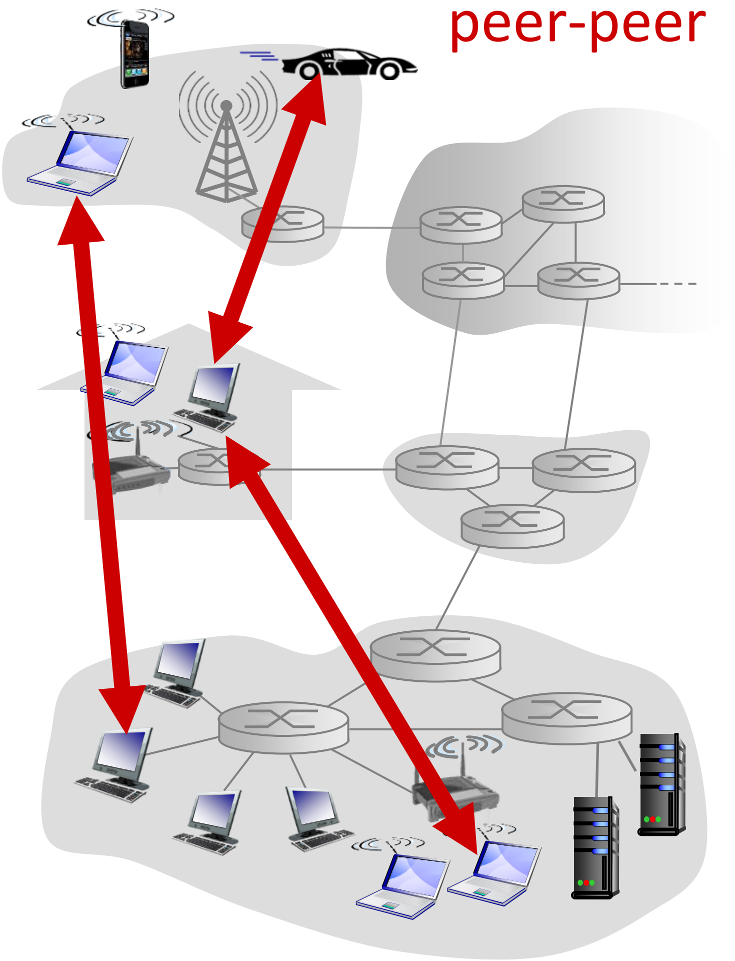
- always-on host
- permanent IP address
- data centers for scaling

clients:

- send requests to server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

P2P Architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - *complex management*



Hybrid of Client-server and P2P

Skype

- voice-over-IP P2P application
- centralized server: finding address of remote party:
- client-client connection: direct (not through server)

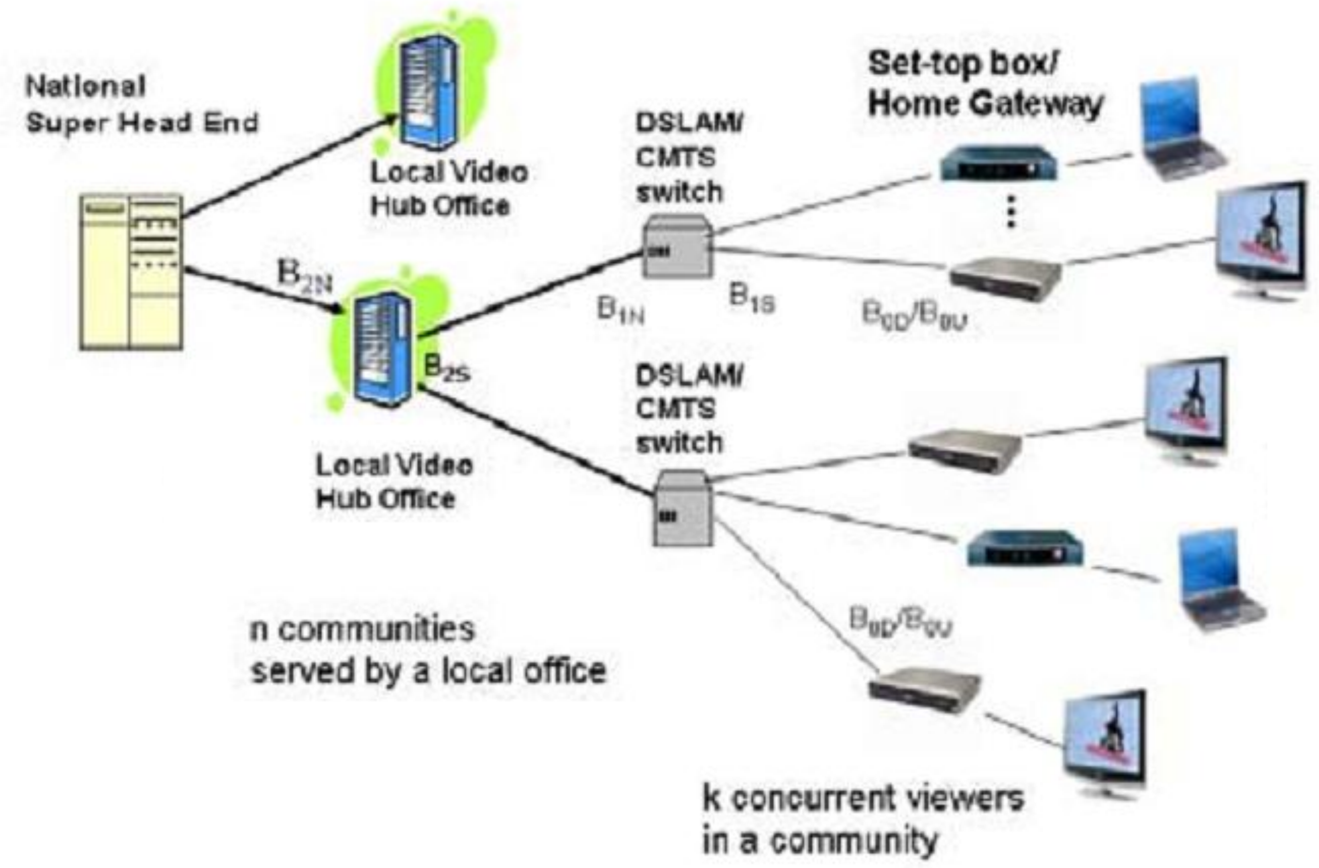
Instant messaging

- chatting between two users is P2P
- centralized service: client presence detection/location
  - user registers its IP address with central server when it comes online
  - user contacts central server to find IP addresses of buddies

P2P IPTV

- Support both live streaming and Video-on-Demand (VOD)

IPTV Distribution



Chen et. al., “When is P2P Technology Beneficial for IPTV Services?”,
NOSSDAV’07 Urbana, Illinois USA

Application Layer Protocol

defines

- types of messages exchanged,
  - e.g., request, response
- message **syntax**:
  - what fields in messages & how fields are delineated
- message **semantics**
  - meaning of information in fields
- rules for when and how processes send & respond to messages

at the ***application*** layer

open protocols:

- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype

What Transport Characteristics does a Network Application Need?

loss tolerance

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- other apps ("elastic apps") make use of whatever throughput they get

security

- confidentiality, data integrity, etc… may be needed

Application Layer
Transport Service Requirements

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps- 1Mbps	yes, 100’s msec
stored audio/video	loss-tolerant	video:10kbps-	
interactive games	loss-tolerant	5Mbps	yes, few secs
text messaging	no loss	same as above	yes, 100’s
		few kbps up	msec
		elastic	yes and no

Internet Transport Protocols Services

TCP service:

- *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 throughput guarantee, security
- *connection-oriented:* setup required between client and server processes

UDP service:

- *unreliable data transfer* between sending and receiving process
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup

Q: why bother? Why is there a UDP service?

Internet Applications and Transport Protocols

application	application layer protocol	underlying transport protocol
e-mail	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 (signaling) UDP (media)

Protocol for Web Surfing: HTTP

(Hyper-Text Transfer Protocol)

First, a review…

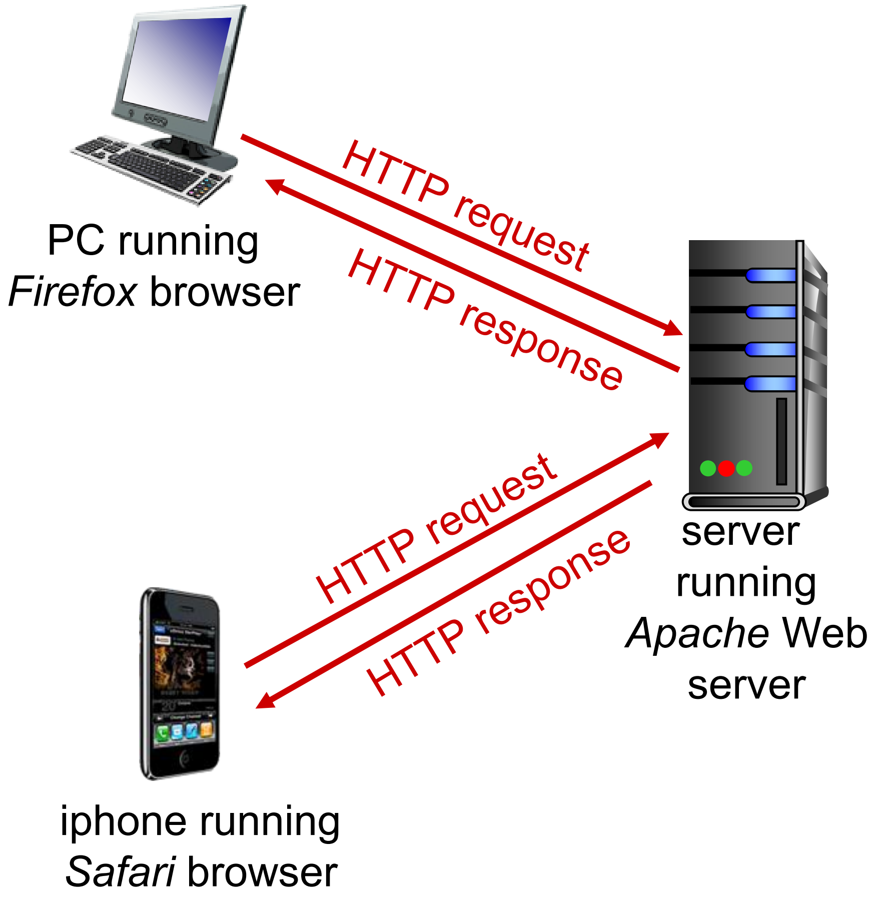
- *web page* consists of *objects*
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects*
- each object is addressable by a *URL,* e.g.,

$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

HTTP Overview

HTTP: *hypertext transfer protocol*

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests



HTTP Overview – cont’d

HTTP uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is “stateless”

- server maintains no information about past client requests

aside

protocols that maintain “state” are complex!

- ❖ past history (state) must be maintained
- ❖ if server/client crashes, their views of “state” may be inconsistent, must be reconciled

HTTP Connections

non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

persistent HTTP

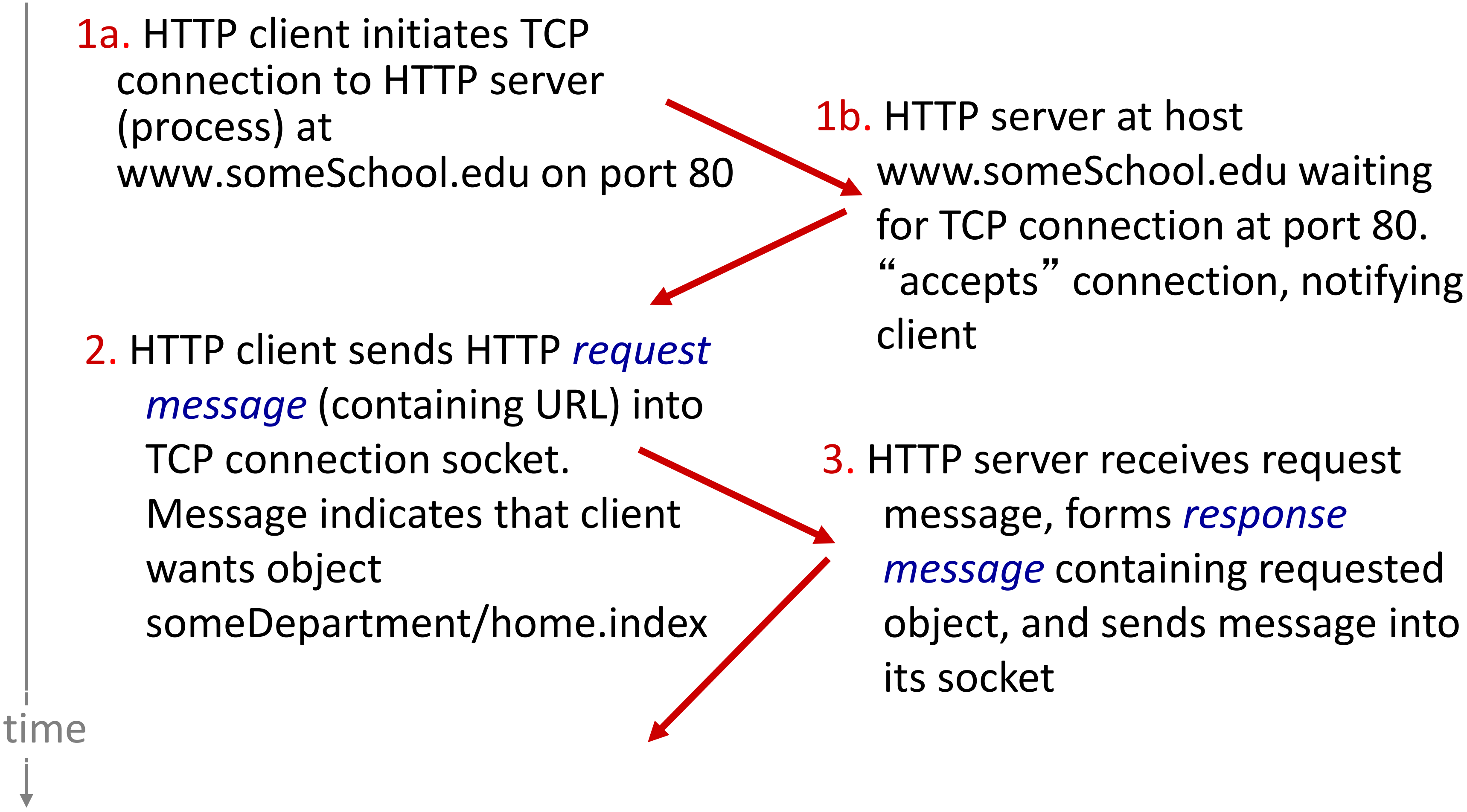
- multiple objects can be sent over single TCP connection between client and server

Non-persistent HTTP

suppose user enters URL:

`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)



1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

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

Non-persistent HTTP – cont’d

4. HTTP server closes TCP connection.

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

time

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

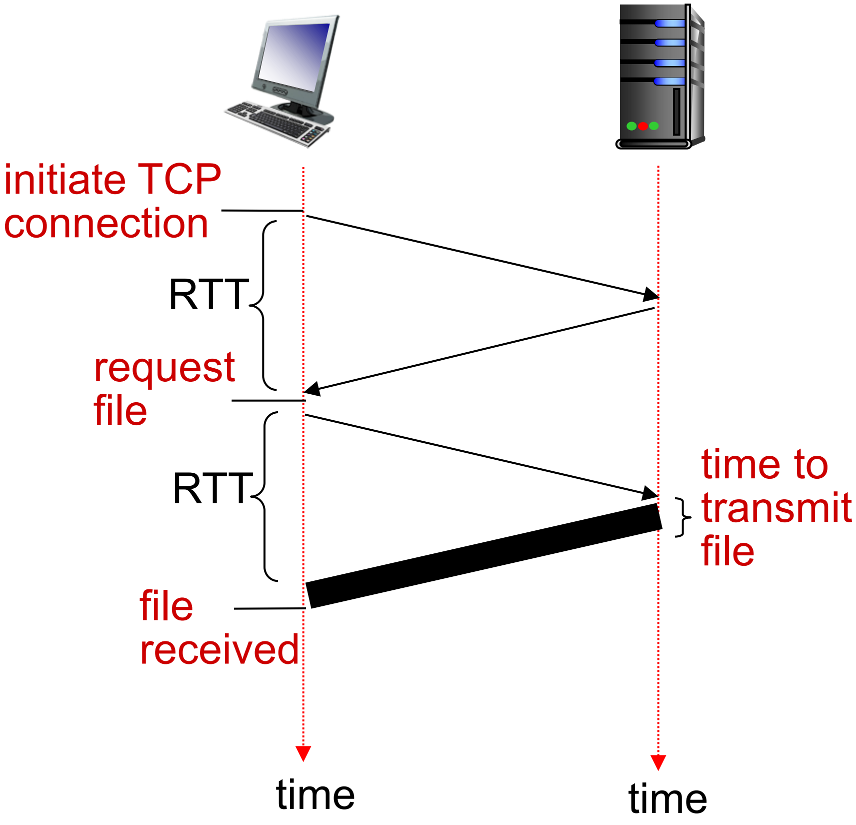
Non-persistent HTTP: Response Time

RTT (definition): time for a small packet to travel from client to server and back

HTTP 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
- non-persistent HTTP response time =

  2RTT+ file transmission time



Persistent HTTP

non-persistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

persistent HTTP:

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

HTTP Request Message

- two types of HTTP messages: *request, response*
- HTTP request message:
  - Text format, e.g. ASCII, UTF
  - human-readable!

carriage return character

line-feed character

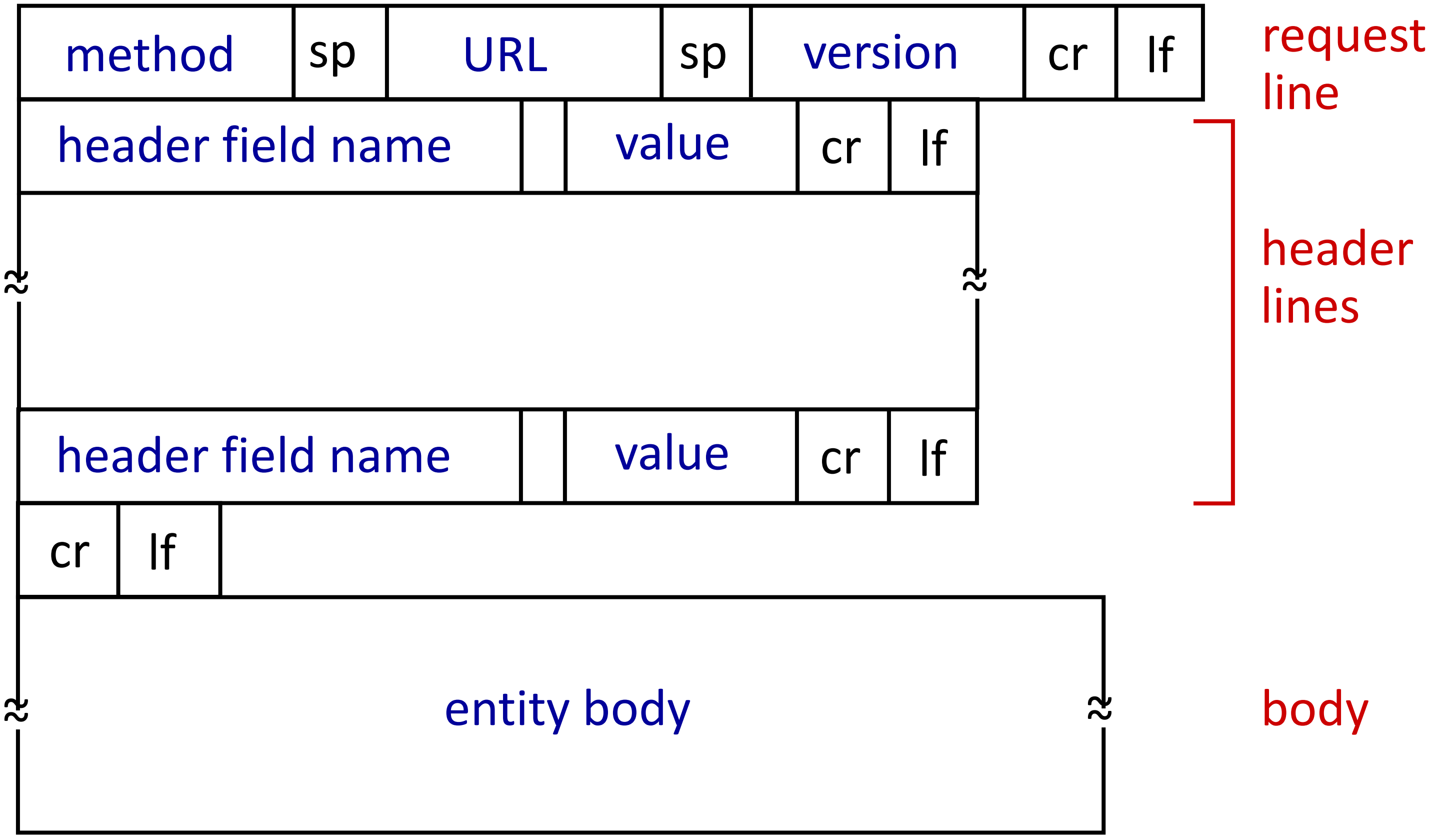
request line
(GET, POST,
HEAD commands)

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

header
lines

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

HTTP Request Message: General Format



method sp URL sp version cr lf — request line

header field name value cr lf

header field name value cr lf — header lines

cr lf

entity body — body

Uploading Form Input

POST method:

- web page often includes form input
- input is uploaded to server in entity body

URL method:

- uses **GET** method
- input is uploaded in URL field of request line:

```
www.somesite.com/animalsearch?monkeys&banana
```

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 in entity body 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\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

HTTP Response Status Codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

`200 OK`

  - request succeeded, requested object later in this msg

`301 Moved Permanently`

  - requested object moved, new location specified later in this msg (Location:)

`400 Bad Request`

  - request msg not understood by server

`404 Not Found`

  - requested document not found on this server

`505 HTTP Version Not Supported`