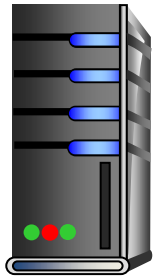
CS 43: Computer Networks
HTTP

Kevin Webb
Swarthmore College

September 8, 2015

What IS A Web Browser?

HTTP Overview



1. User types in a URL.

   http://some.host.name.tld/directory/name/file.ext

HTTP Overview

2. Browser establishes connection with server.
   - Looks up “some.host.name.tld”
   - Calls connect()

HTTP Overview

3. Browser requests the corresponding data.

GET /directory/name/file.ext HTTP/1.0
Host: some.host.name.tld
[other optional fields, for example:]
User-agent: Mozilla/5.0 (Windows NT 6.1; WOW64)
Accept-language: en

HTTP Overview

4. Server responds with the requested data.

HTTP/1.0 200 OK
Content-Type: text/html
Content-Length: 1299
Date: Sun, 01 Sep 2013 21:26:38 GMT
[Blank line]
(Data data data data…)

HTTP Overview

5. Browser renders the response, fetches any additional objects, and closes the connection.

HTTP Overview

1. User types in a URL.
2. Browser establishes connection with server.
3. Browser requests the corresponding data.
4. Server responds with the requested data.
5. Browser renders the response, fetches other objects, and closes the connection.

It's a document retrieval system, where documents point to (link to) each other, forming a "web".

HTTP Overview (Lab 1)

1. User types in a URL.
2. Browser establishes connection with server.
3. Browser requests the corresponding data.
4. Server responds with the requested data.
5. ~~Browser renders the response, fetches other objects,~~ and closes the connection.

It's a document retrieval system, where documents point to (link to) each other, forming a "web".

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet www.cs.swarthmore.edu 80
```

Opens TCP connection to port 80 (default HTTP server port) at CS Dept server. Anything typed is sent to server on port 80 at www.cs.swarthmore.edu

2. Type in a GET HTTP request:

```
GET /~kwebb/ HTTP/1.1
Host: www.cs.swarthmore.edu
```

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to the HTTP server.

3. Look at response message sent by HTTP server!

Example

```
ip-91-47:~ cynthiataylor$ telnet occs.oberlin.edu 80
Trying 132.162.201.24...
Connected to occs.cs.oberlin.edu.
Escape character is '^]'.
GET /~ctaylor/sample.html HTTP/1.1
Host: occs.oberlin.edu

HTTP/1.1 200 OK
Date: Mon, 11 Feb 2013 16:02:24 GMT
Server: Apache/2.2.14 (Unix) mod_ssl/2.2.14 OpenSSL/0.9.8p DAV/2 SVN/1.6.4
Last-Modified: Mon, 11 Feb 2013 15:59:42 GMT
ETag: "5be-3c-4d574ff40876c"
Accept-Ranges: bytes
Content-Length: 60
Content-Type: text/html

<html>
<head></head>
<body>
I am a website!
</body>
</html>
```

Example

```
ip-91-47:~ cynthiataylor$ telnet occs.oberlin.edu 80
Trying 132.162.201.24...
Connected to occs.cs.oberlin.edu.
Escape character is '^]'.
GET /~ctaylor/sample.html HTTP/1.1
Host: occs.oberlin.edu

HTTP/1.1 200 OK
Date: Mon, 11 Feb 2013 16:02:24 GMT
Server: Apache/2.2.14 (Unix) mod_ssl/2.2.14 OpenSSL/0.9.8p DAV/2 SVN/1.6.4
Last-Modified: Mon, 11 Feb 2013 15:59:42 GMT
ETag: "5be-3c-4d574ff40876c"
Accept-Ranges: bytes
Content-Length: 60
Content-Type: text/html

<html>
<head></head>
<body>
I am a website!
</body>
</html>
```

Response headers

Response body

(This is what you should be saving in lab 1.)

HTTP request message

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

carriage return character

line-feed character

request line (GET, POST, HEAD, etc. commands)

header lines

carriage return, line feed

```
GET /index.html HTTP/1.1\r\n
Host: web.cs.swarthmore.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
```

Why do we have these \r\n (CRLF) things all over the place?

```
GET /index.html HTTP/1.1\r\n
Host: web.cs.swarthmore.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
```

A. They're generated when the user hits 'enter'.

B. They signal the end of a field or section.

C. They're important for some other reason.

D. They're an unnecessary protocol artifact.

How else might we delineate messages?

A. There’s not much else we can do.

B. Force all messages to be the same size.

C. Send the message size prior to the message.

D. Some other way (discuss).

HTTP is all text…

- Makes the protocol simple
  - Easy to delineate message (\r\n)
  - (Relatively) human-readable
  - No worries about encoding or formatting data
  - Variable length data
- Not the most efficient
  - Many protocols use binary fields
    - Sending “12345678” as a string is 8 bytes
    - As an integer, 12345678 needs only 4 bytes
  - The headers may come in any order
  - Requires string parsing / processing

Example

```
ip-91-47:~ cynthiataylor$ telnet occs.oberlin.edu 80
Trying 132.162.201.24...
Connected to occs.cs.oberlin.edu.
Escape character is '^]'.
GET /~ctaylor/sample.html HTTP/1.1
Host: occs.oberlin.edu

HTTP/1.1 200 OK
Date: Mon, 11 Feb 2013 16:02:24 GMT
Server: Apache/2.2.14 (Unix) mod_ssl/2.2.14 OpenSSL/0.9.8p DAV/2 SVN/1.6.4
Last-Modified: Mon, 11 Feb 2013 15:59:42 GMT
ETag: "5be-3c-4d574ff40876c"
Accept-Ranges: bytes
Content-Length: 60
Content-Type: text/html

<html>
<head></head>
<body>
I am a website!
</body>
</html>
```

Wireshark

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: http Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Length	Info
14	14.211168	130.58.68.164	130.58.68.137	HTTP	68	GET /~kwebb/ HTTP/1.1
16	14.268895	130.58.68.137	130.58.68.164	HTTP	5447	HTTP/1.1 200 OK (text/html)

```
▶ Frame 16: 5447 bytes on wire (43576 bits), 5447 bytes captured (43576 bits)
▶ Ethernet II, Src: SuperMic_74:d5:b8 (00:25:90:74:d5:b8), Dst: Hewlett-_94:5f:1e (d8:d3:85:94:5f:1e)
▶ Internet Protocol Version 4, Src: 130.58.68.137 (130.58.68.137), Dst: 130.58.68.164 (130.58.68.164)
▶ Transmission Control Protocol, Src Port: http (80), Dst Port: 35736 (35736), Seq: 1, Ack: 55, Len: 5381
▼ Hypertext Transfer Protocol
  ▼ HTTP/1.1 200 OK\r\n
    ▶ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      Request Version: HTTP/1.1
      Status Code: 200
      Response Phrase: OK
    Date: Mon, 02 Sep 2013 20:10:28 GMT\r\n
    Server: Apache/2.2.22 (Ubuntu)\r\n
    Last-Modified: Sat, 31 Aug 2013 20:44:44 GMT\r\n
    ETag: "c3f7c-1401-4e54468c06210"\r\n
    Accept-Ranges: bytes\r\n
  ▼ Content-Length: 5121\r\n
      [Content length: 5121]
    Vary: Accept-Encoding\r\n
    Content-Type: text/html\r\n
    \r\n
▼ Line-based text data: text/html
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" >\r\n
    <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en" >\r\n
    <head>\r\n
```

```
0040  fe fc 48 54 54 50 2f 31  2e 31 20 32 30 30 20 4f   ..HTTP/1 .1 200 O
0050  4b 0d 0a 44 61 74 65 3a  20 4d 6f 6e 2c 20 30 32   K..Date:  Mon, 02
0060  20 53 65 70 20 32 30 31  33 20 32 30 3a 31 30 3a    Sep 201 3 20:10:
0070  32 38 20 47 4d 54 0d 0a  53 65 72 76 65 72 3a 20   28 GMT.. Server:
```

HTTP Response Status Code (... Packets: 20 Displayed: 2 Marked: 0 Load time: 0:00.000 Profile: Default

Request Method Types ("verbs")

HTTP/1.0 (1996):

- GET
  - Requests page.
- POST
  - Uploads user response to a form.
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1 (1997 & 1999):

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field
- TRACE, OPTIONS, CONNECT, PATCH

Request Method Types ("verbs")

HTTP/1.0 (1996):

- GET
  - Requests page.
- POST
  - Uploads user response to a form.
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1 (1997 & 1999):

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field
- TRACE, OPTIONS, CONNECT, PATCH
- (+) Persistent connections

Request Method Types ("verbs")

HTTP/1.0 (1996):

- GET
  - Requests page.
- POST
  - Uploads user response to a form.
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1 (1997 & 1999):

- **GET, POST**, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field
- TRACE, OPTIONS, CONNECT, PATCH
- (+) Persistent connections

Uploading form input

GET (in-URL) method:

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

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

POST method:

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

GET vs. POST

- GET can be used for *idempotent* requests
  - Idempotence: an operation can be applied multiple times without changing the result (the final state is the same)

GET vs. POST

- GET can be used for *idempotent* requests
  - Idempotence: an operation can be applied multiple times without changing the result (the final state is the same)

How many of the following operations are idempotent?

I. Incrementing a variable
II. Assigning a value to a variable
III. Allocating memory
IV. Compiling a program

A. None of them
B. One of them
C. Two of them
D. Three of them
E. All of them

GET vs. POST

- GET can be used for *idempotent* requests.
  - Idempotence: an operation can be applied multiple times without changing the result (the final state is the same)

- POST should be when…
  - A request changes the state of the SESSION or server or DB
  - Sending a request twice would be harmful
    - (Some) browsers warn about sending multiple post requests
  - Users are inputting non-ascii characters
  - Input may be very large
  - You want to hide how the form works/user input

When might you use GET vs. POST?

	GET	POST
A.	Forum post	Search terms, Pizza order
B.	Search terms, Pizza order	Forum post
C.	Search terms	Forum post, Pizza order
D.	Forum post, Search terms, Pizza Order	
E.		Forum post, Search terms, Pizza Order

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 first line of server-to-client response message.
- Some common response 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

`403 Forbidden`

- You don't have permission to read the object

`404 Not Found`

- Requested document not found on this server

`505 HTTP Version Not Supported`

HTTP response status codes

- Status code appears in first line of server-to-client response message.
- Many others too. Search "list of HTTP status codes"

420 Enhance Your Calm (twitter)

- Slow down, you're being rate limited

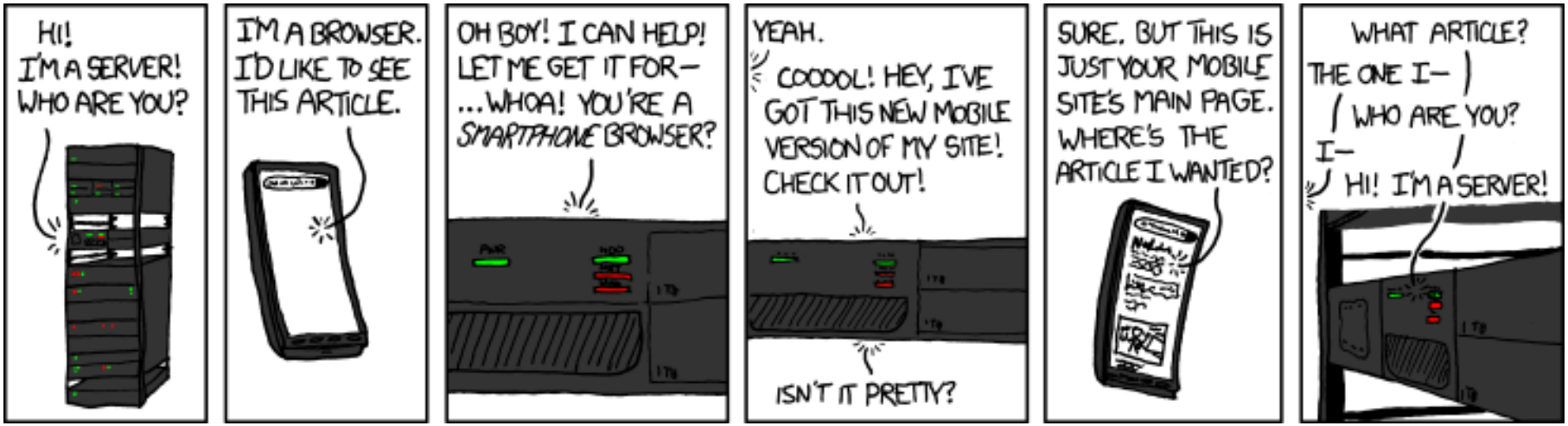
451 Unavailable for Legal Reasons

- Censorship?

418 I'm a Teapot

- Response from a teapot requested to brew a beverage (announced Apr 1)

State(less)



(XKCD #869, “Server Attention Span”)

State(less)

- Original web: simple document retrieval
- Server is not required to keep state between connections (often it might want to though)
- Client is not required to identify itself (server might refuse to talk otherwise though)

User-server state: cookies

Many web sites use cookies

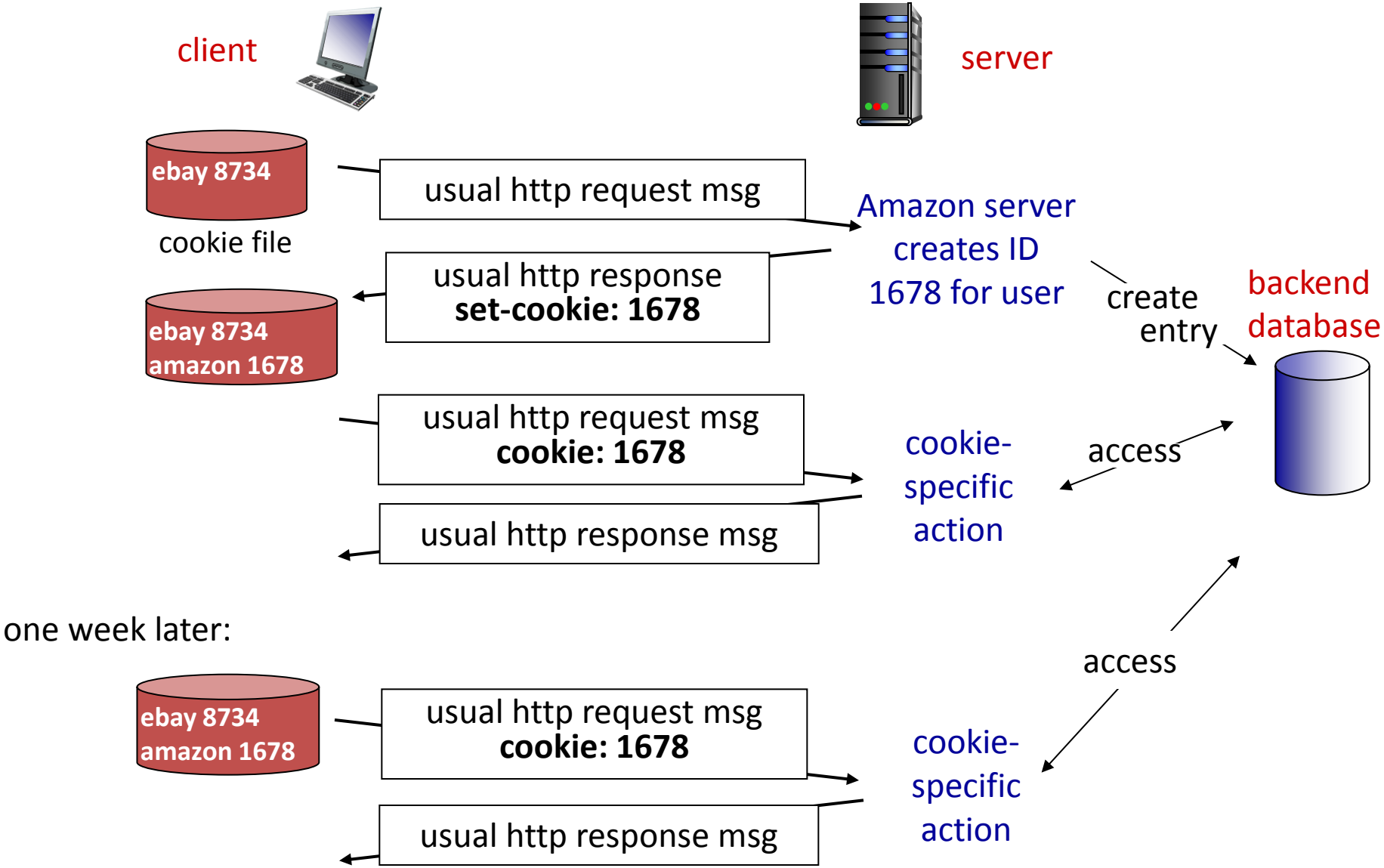
Four components:

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

Example:

- Susan always accesses the Internet from her PC
- She visits specific e-commerce site for the first time
- When initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies: keeping "state" (cont.)



Cookies (continued)

What cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

How to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

Cookies: Teaching the Controversy

- Cookies permit sites to learn a lot about you
- You may supply name and e-mail to sites (and more!)
- 3rd party cookies (from ad networks, etc) can follow you across multiple sites.
  - Ever visit a website, and the next day ALL your ads are from them?
- You COULD turn them off
  - But good luck doing anything on the internet!

HTTP Performance

HTTP connections

non-persistent HTTP

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

persistent HTTP

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

object: image, script, stylesheet, etc.

Pseudocode Example

non-persistent HTTP

```
for object on web page:
    connect to server
    request object
    receive object
    close connection
```

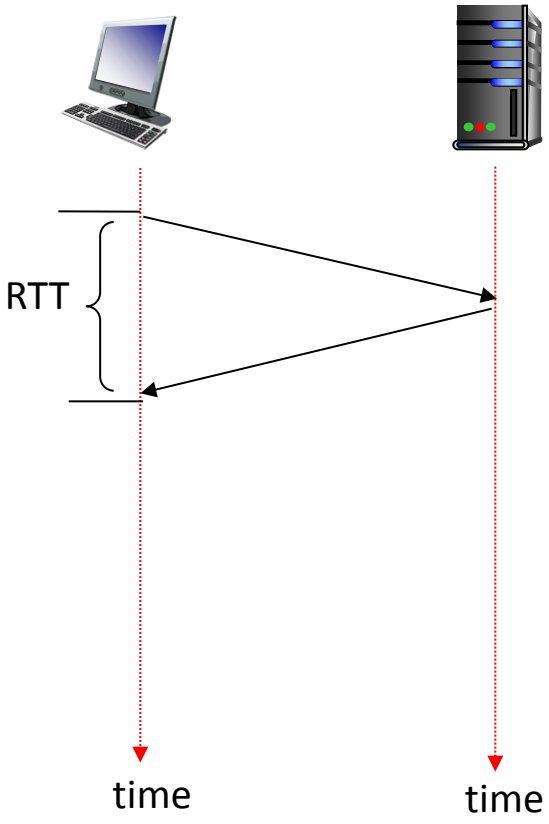
persistent HTTP

```
connect to server
for object on web page:
    request object
    receive object
close connection
```

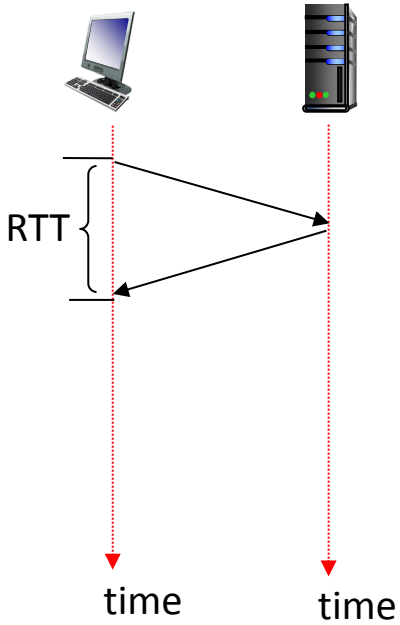
Round Trip Time

Round Trip Time (RTT): time for a small packet to travel from client to server and response to come back

Connection establishment (via TCP) requires one RTT.



Non-Persistent HTTP Connections can download a website with several objects in…



A. One RTT + (File transfer time per object)

B. (One RTT + File transfer time) per object

C. Two RTTs

D. Two RTTs + (File transfer time per object)

E. (Two RTTS + File transfer time) per object

Non-persistent HTTP: response time

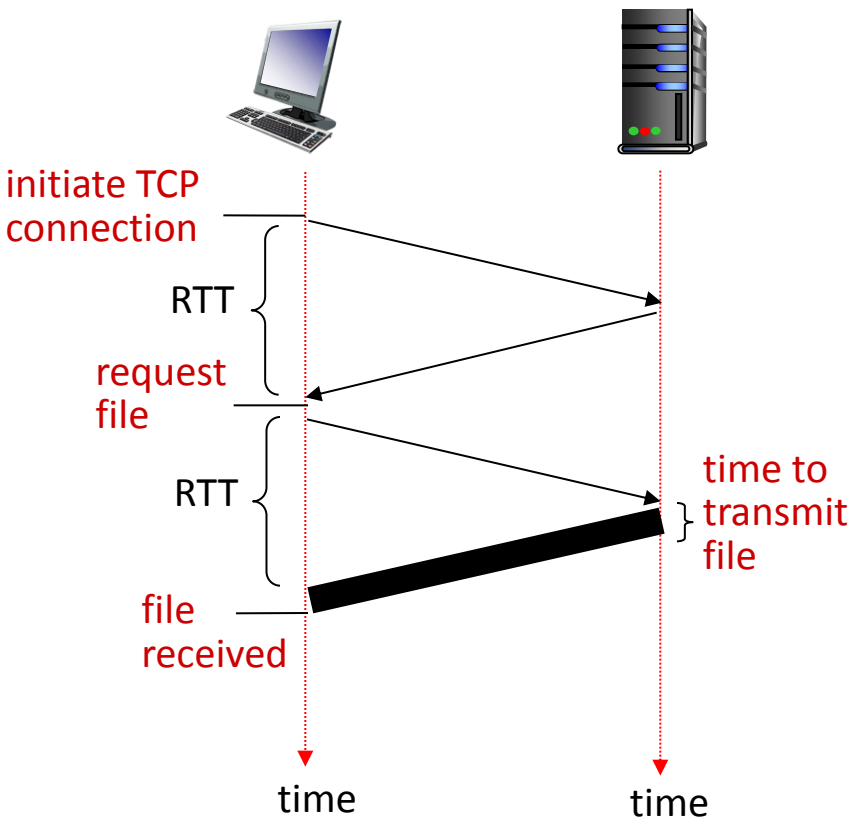
Round Trip Time (RTT): time for a small packet to travel from client to server and back

HTTP response time:

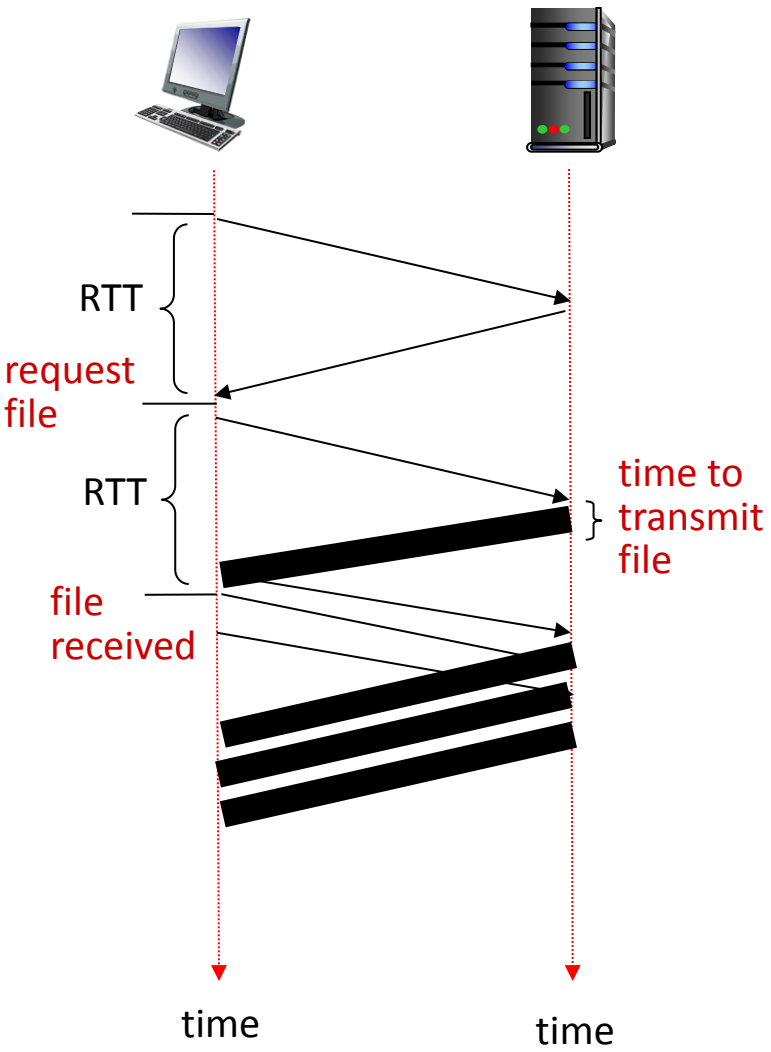
- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time
- non-persistent HTTP response time =

  2RTT+ file transmission time

  <u>For each object</u>



Persistent Connection



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

Reading

- Network applications and transport services
  - Section 2.1