Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

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 with UDP and TCP

Web and HTTP

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.,

`www.someschool.edu/someDept/pic.gif`

host name | path name

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
  - Web's application layer protocol [RFC1945, RFC2616] from 1999:
  - HTTP 1.1:
    http://tools.ietf.org/html/rfc2616
    http://en.wikipedia.org/wiki/HTTP/2

- 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 (continued)

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

Getting the Base object

Round-Trip Time (RTT):

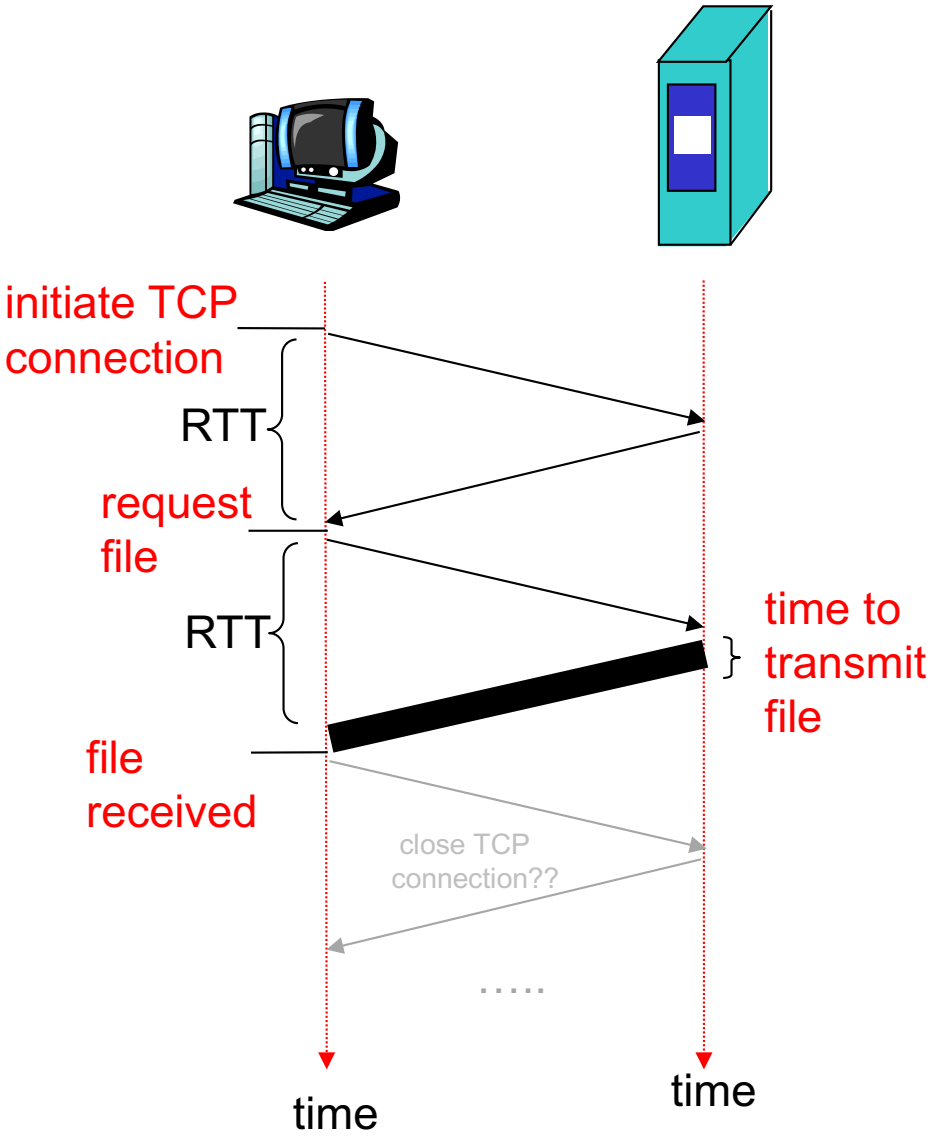
- time for a small packet to travel from client to server and back.

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+transmission time

Close the connection?

- If there are no referenced objects: yes
- If there are referenced objects?



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, 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

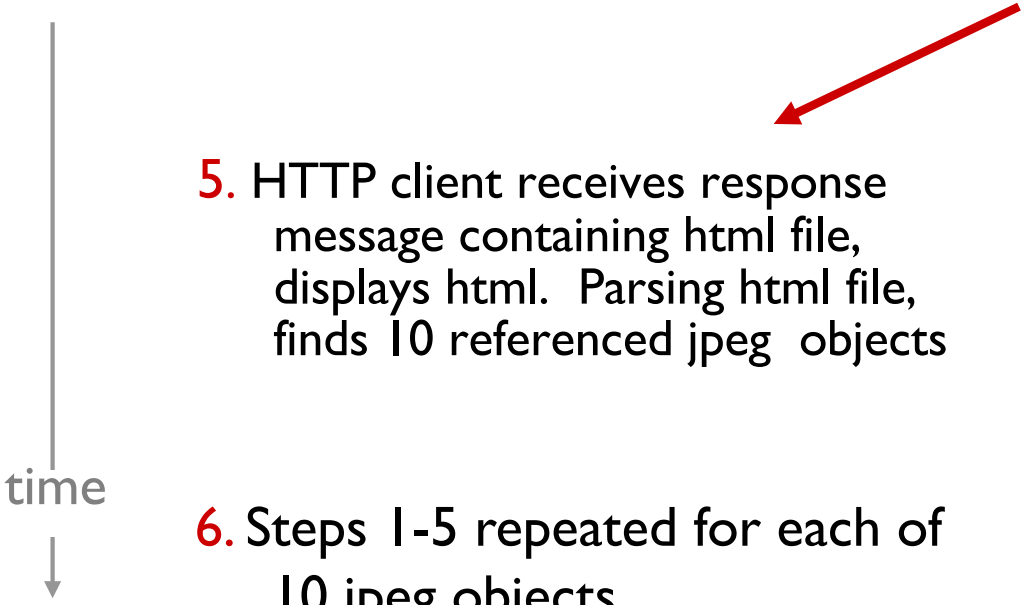
1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

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

time

Non-persistent HTTP (cont.)



4. HTTP server closes TCP connection.

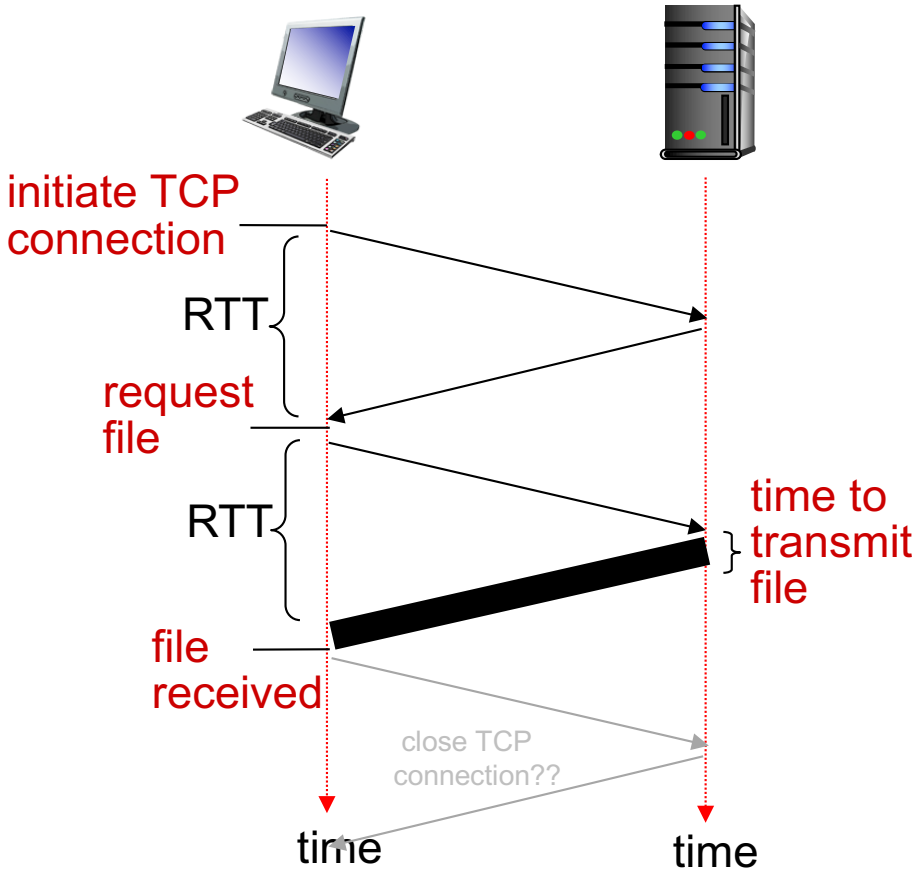
5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

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

Non-persistent HTTP: response time

For each object

- 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 (per object)



Non-Persistent vs. Persistent HTTP

non-persistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- can request referenced objects serially or in parallel
- speedup: browsers often open parallel TCP connections to fetch referenced objects

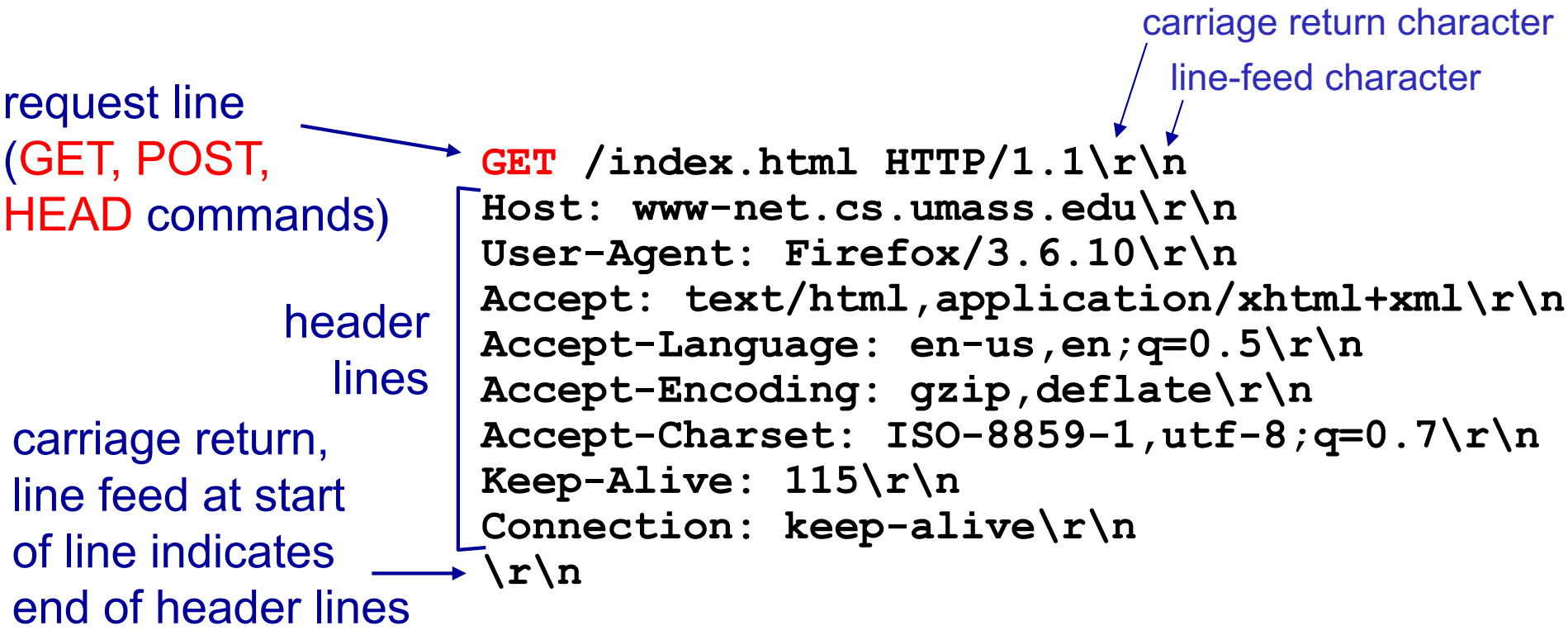
- Default browsers:
  - non-pestisent
  - 5 parallel connections

persistent HTTP:

- server leaves connection open after sending response, waiting for requests
- 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
- speedup: pipelining of requests and responses

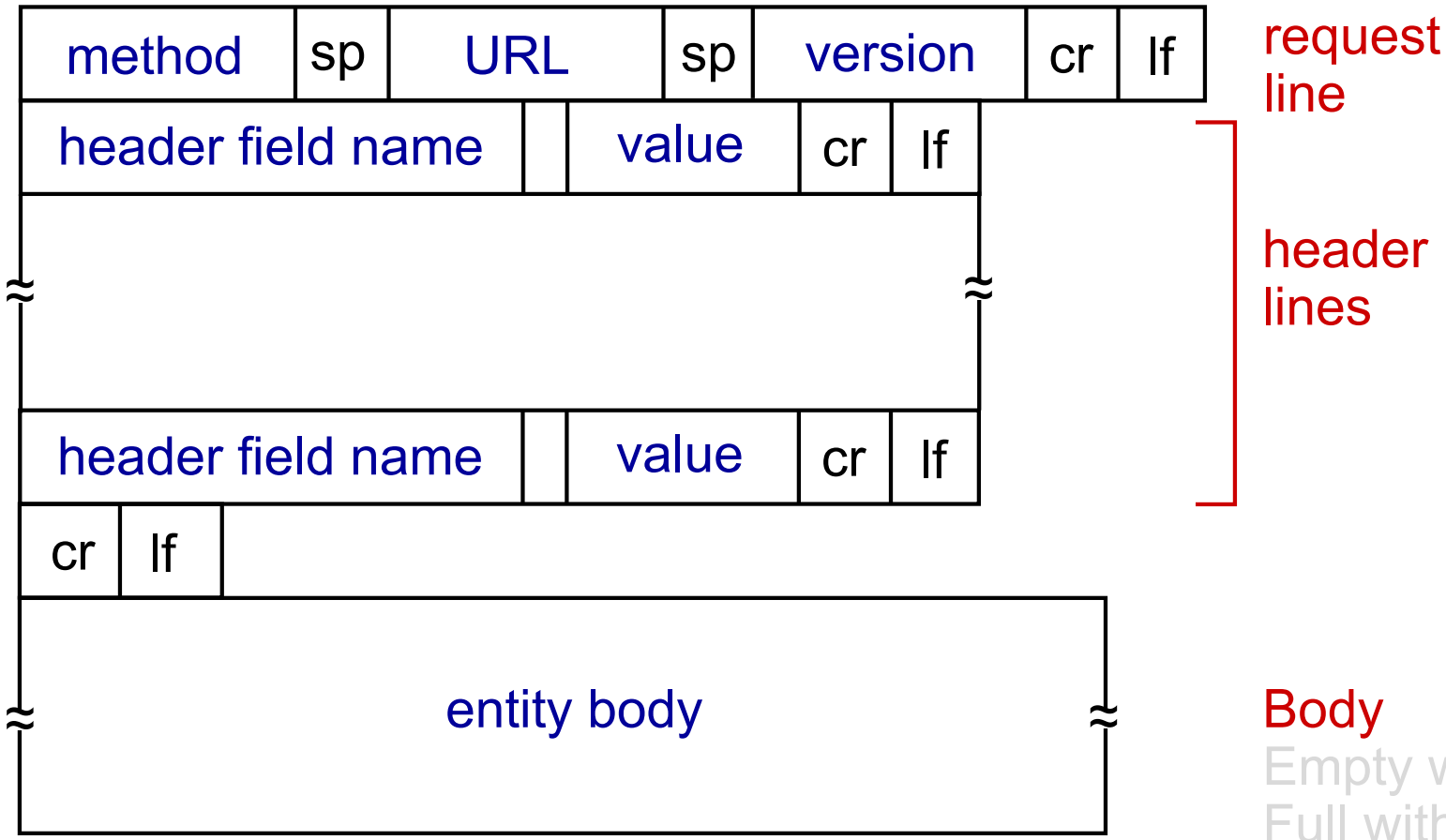
HTTP request message

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



- Check out the online interactive exercises for more examples:
- http://gaia.cs.umass.edu/kurose_ross/interactive/

HTTP request message: general format



Uploading form input

POST method:

- web page often includes form input
- input is uploaded to server in entity body
- answer of server depends on input

URL method:

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

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

Method Types - Summary

HTTP/1.0:

- GET
- POST
  - still a request, but used to fill a form (e.g. a search)
- 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 (for web publishing)
- DELETE
  - deletes file specified in the URL field

See HTTP 1.1: http://www.ietf.org/rfc/rfc2616.txt , Section 5

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 ...
```

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

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:) Client can retrieve new URL

400 Bad Request

- request msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

See HTTP 1.1: http://www.ietf.org/rfc/rfc2616.txt , Section 6

[Trying out HTTP (client side) for yourself]

1. Telnet to your favorite Web server:

```
telnet gaia.cs.umass.edu 80
```

opens TCP connection to port 80 (default HTTP server port) at gaia.cs.umass. edu. anything typed in will be sent to port 80 at gaia.cs.umass.edu

2. type in a GET HTTP request:

```
GET /kurose_ross/interactive/index.php HTTP/1.1
Host: gaia.cs.umass.edu
```

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

3. look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)

Try out HTTP (client side)

http://odysseas.calit2.uci.edu/doku.php/public:teaching-eecs148-f16

1. ssh to your favorite Web server:

```
telnet odysseas.calit2.uci.edu 80
```

opens TCP connection to port 80 (default HTTP server port) at odysseas.calit2.uci.edu. Anything typed in, is sent there.

2. type in a GET HTTP request:

```
GET /doku.php/public:teaching-eecs148-f16/ HTTP/1.1
Host: odysseas.calit2.uci.edu
```

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

3. look at response message sent by HTTP server (or use Wireshark)

Try out HTTP -variations

http://odysseas.calit2.uci.edu/doku.php/public:teaching-eecs148-f16

1. ssh to your favorite Web server:

```
telnet odysseas.calit2.uci.edu 80
```

2. type in a GET HTTP request:

```
GET /doku.php/public:teaching-eecs148-f16/ HTTP/1.1
Host: odysseas.calit2.uci.edu
```

4. Try variations …

```
HEAD /doku.php/public:teaching-eecs148-f16 HTTP/1.1
Host: odysseas.calit2.uci.edu

GET /doku.php/public:teaching-eecs148-f16/ HTTP/1.1
Host: odysseas.calit2.uci.edu
Keep-Alive: timeout=100, max=100

HEAD /doku.php/public:teaching-eecs148-f16 HTTP/1.1
Host: odysseas.calit2.uci.edu

HEAD badrequest  H
Host: odysseas.calit2.uci.edu
```

The role of your browser

- Your browser takes your input and constructs HTTP compliant requests
  - Taking into account
    - browser type and version
    - configuration (e.g. language, type of TCP connections)
    - and user input
- Your browser also receives the HTTP response and displays the page

I-clicker Question

Q3: Which one of the following is true?

☐ A: HTTP is a protocol for getting webpages only.

☐ B: HTTP is a protocol for displaying webpages.

☐ C: HTTP is a protocol for requesting and getting any content, including but not limited to webpages

☐ D: HTTP is a program run by web servers.

HTTP Summary

- Last Time: Main HTTP protocol
  - Request, Response format
  - Persistent, Non-Persistent
- Today: Add-ons
  - Cookies
  - Proxies, Caching
- http://tools.ietf.org/html/rfc2616

- Logistics
  - HW1 due this Friday on Canvas.

Add-on 1: User-server state: cookies

many Web sites use cookies

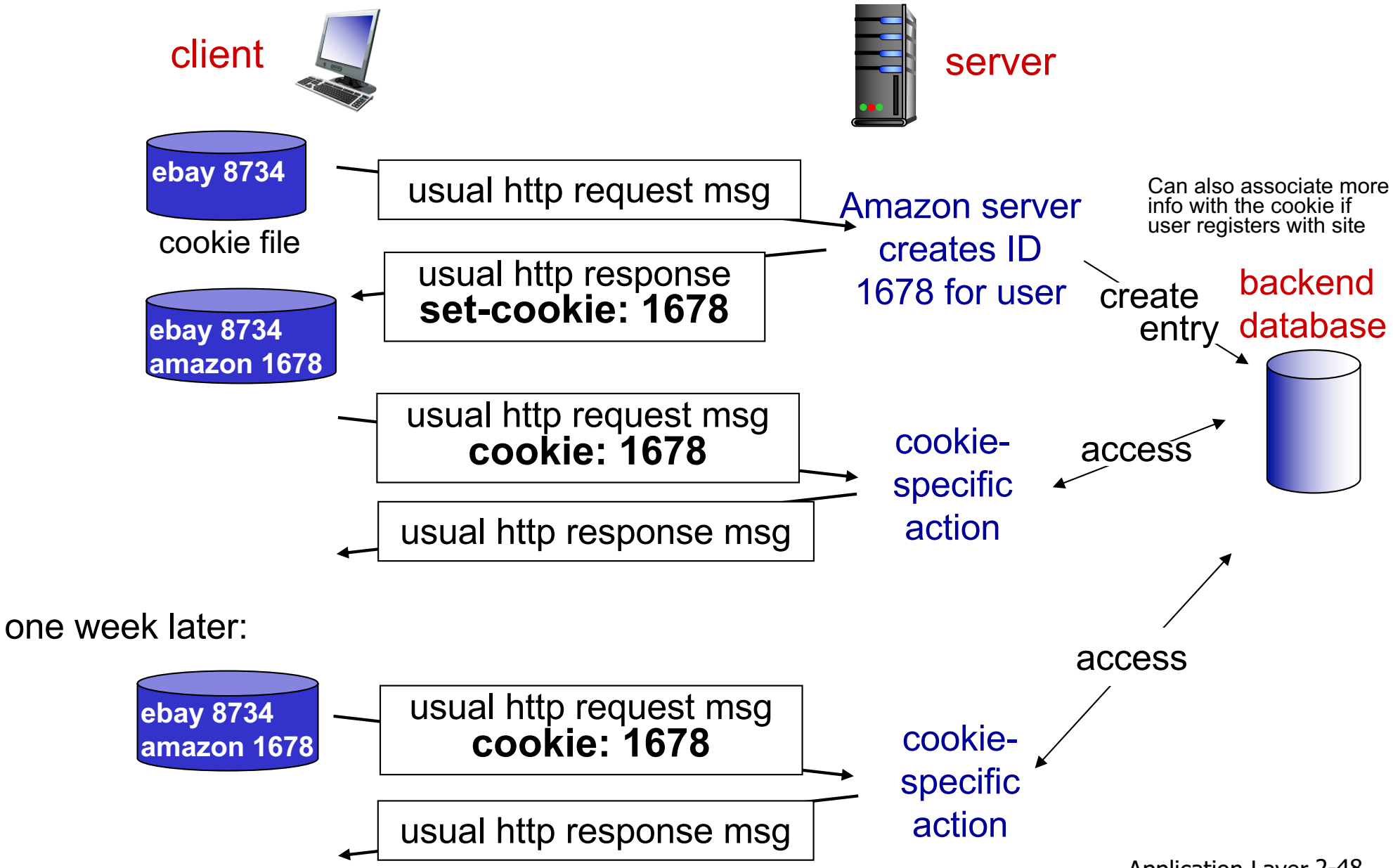
four components:

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 access Internet from PC
- visits specific e-commerce site for 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)

aside

cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

how to keep "state":

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

FAQ on cookies

Q1: If I delete my cookie, next time I visit the same webserver, will I be assigned the same? Or will there be a mapping between the new and old cookie?

A1: Not through HTTP (stateless). Although the server could do and save the mapping.

- Q1a: How can you test which sites track you through cookies?

FAQ on cookies

Q2: If somebody steals my cookies file (or overhears my cookies sent in the clear over wireless), can he login (i.e., authenticate) as me?

A2: Yes, as long as the cookie is still valid.

Q2 cont'd: Are there any mechanisms in place to prevent others from using my cookie?

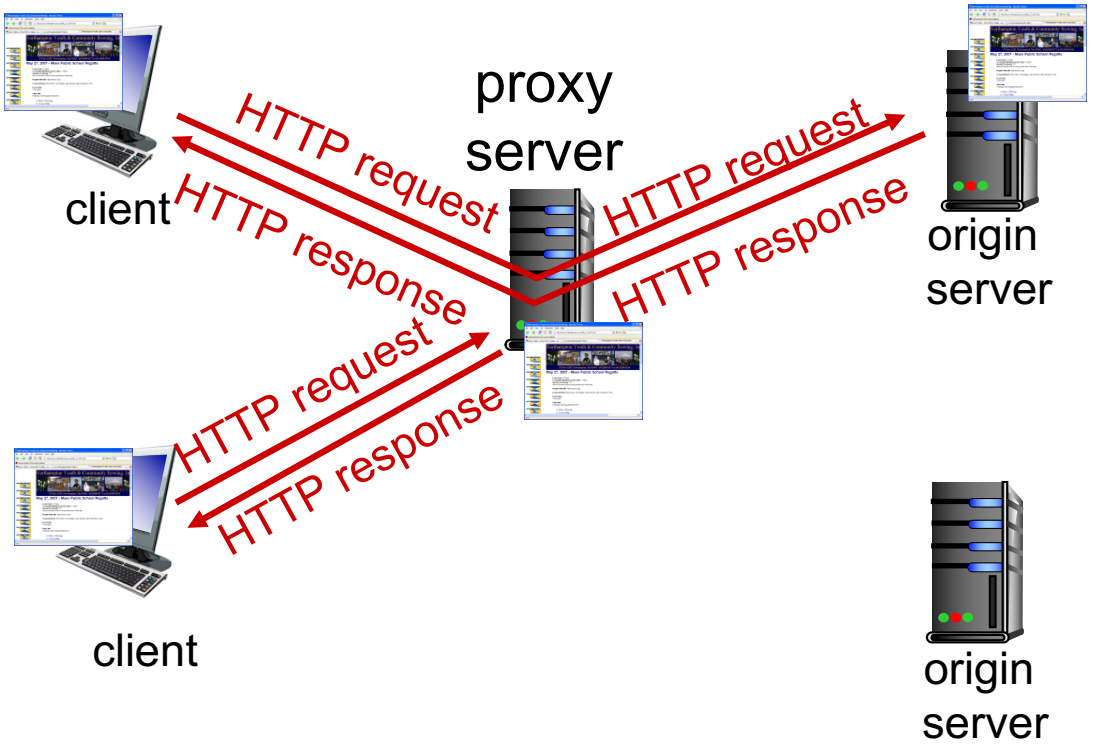
A2 cont'd: YES, through authentication mechanisms.

- E.g. see http://sconce.ics.uci.edu/cs203-w12/lec4-web-auth.pdf or http://web.eecs.umich.edu/~kevinfu/talks/Fu-cookie-slides.pdf
- Ideas:
  - HTTPS encrypts the cookie so it can't be overheard
  - Session ID is part of the cookie: while logged & active in the website the cookie is valid; then the server makes it invalid.
  - Server provides cookie:authenticator
- The mechanisms are not bullet-proof but put the bar higher.

Add-on 2: Web caches (proxy server)

goal: satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



More about Web caching

- cache acts as both client and server
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)

why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link
- Internet dense with caches: enables "poor" content providers to effectively deliver content

Caching example:

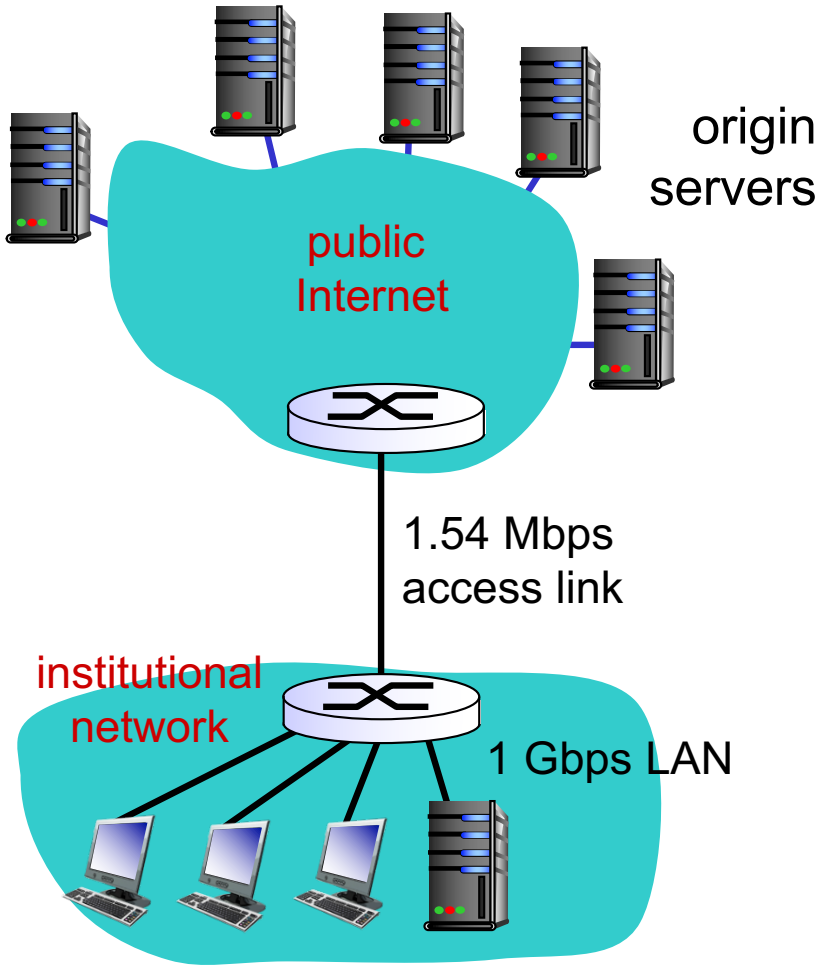
assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:

- LAN utilization: 15% *problem!*
- access link utilization = 99%
- total delay = Internet delay + access delay + LAN delay

  = 2 sec + minutes + usecs



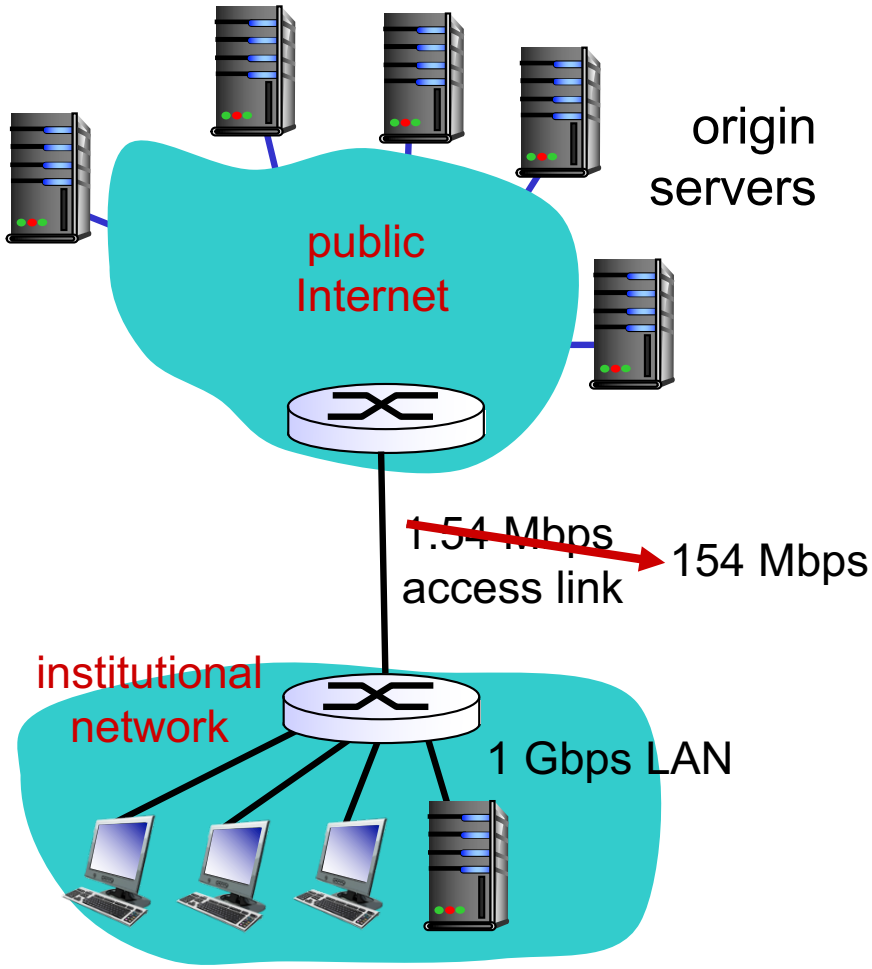
Caching example: fatter access link

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: ~~1.54 Mbps~~ 154 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = ~~99%~~ 9.9%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + ~~minutes~~ + usecs
  msecs



Cost: increased access link speed (not cheap!)

Caching example: install local cache

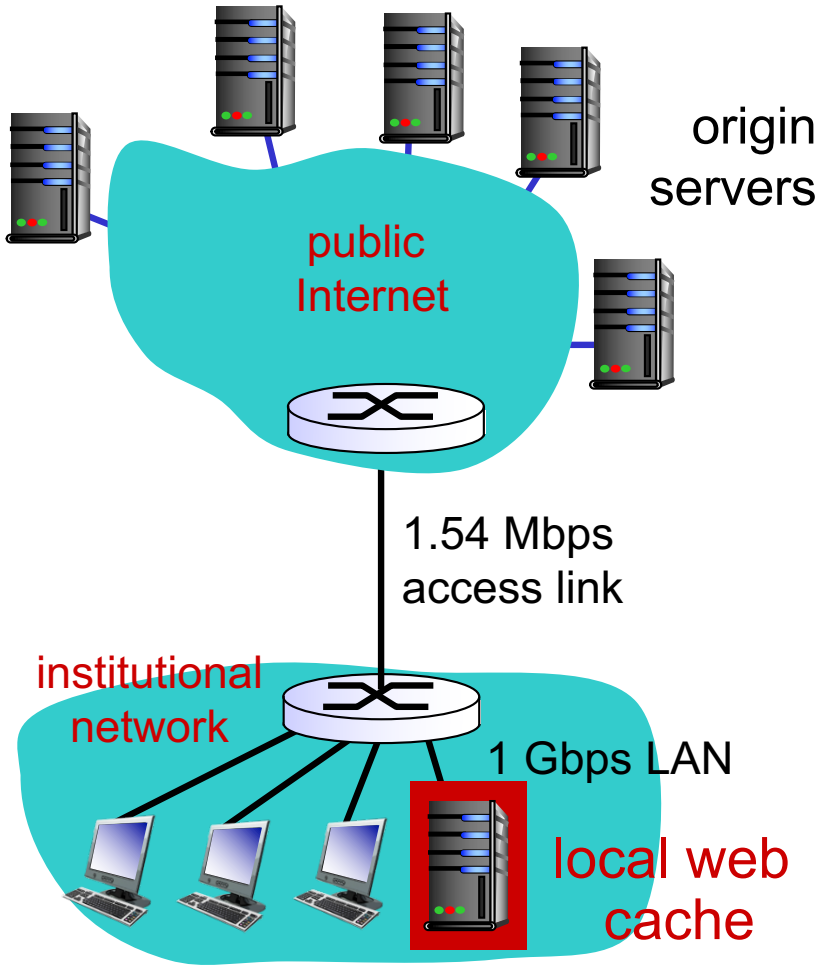
assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = ?
- total delay = ?

How to compute link utilization, delay?



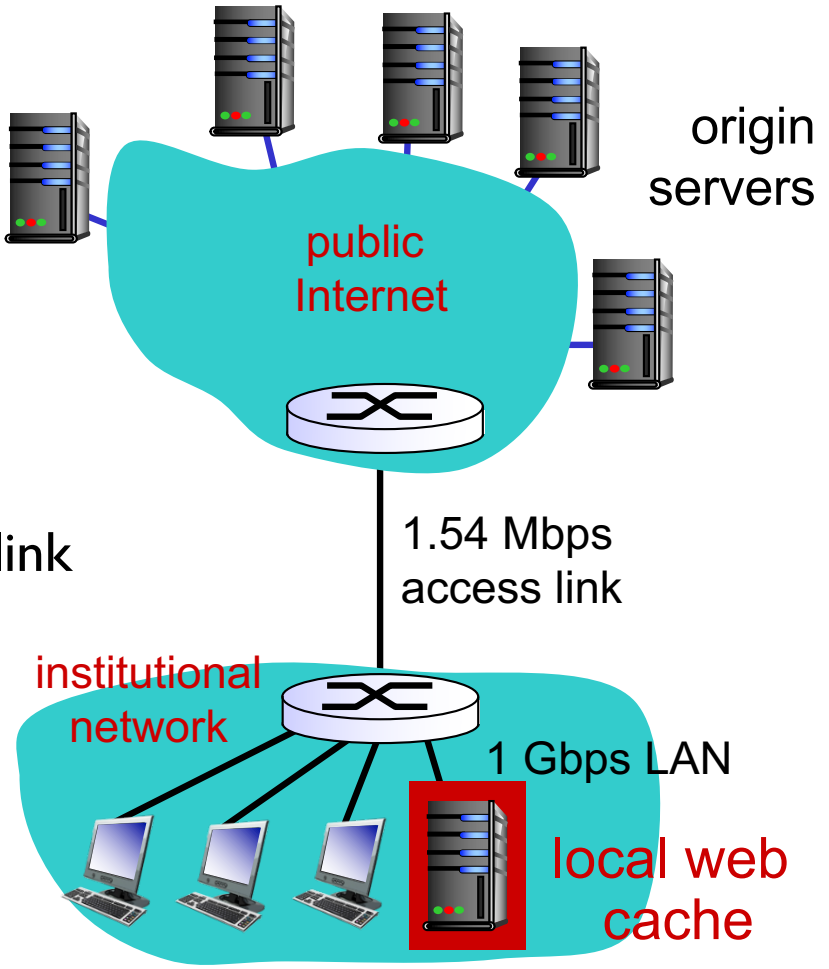
Cost: web cache (cheap!)

Caching example: install local cache

Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin
- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link

  = 0.6*1.50 Mbps = .9 Mbps
  - utilization = 0.9/1.54 = .58
- total delay
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs
  - less than with 154 Mbps link (and cheaper too!)

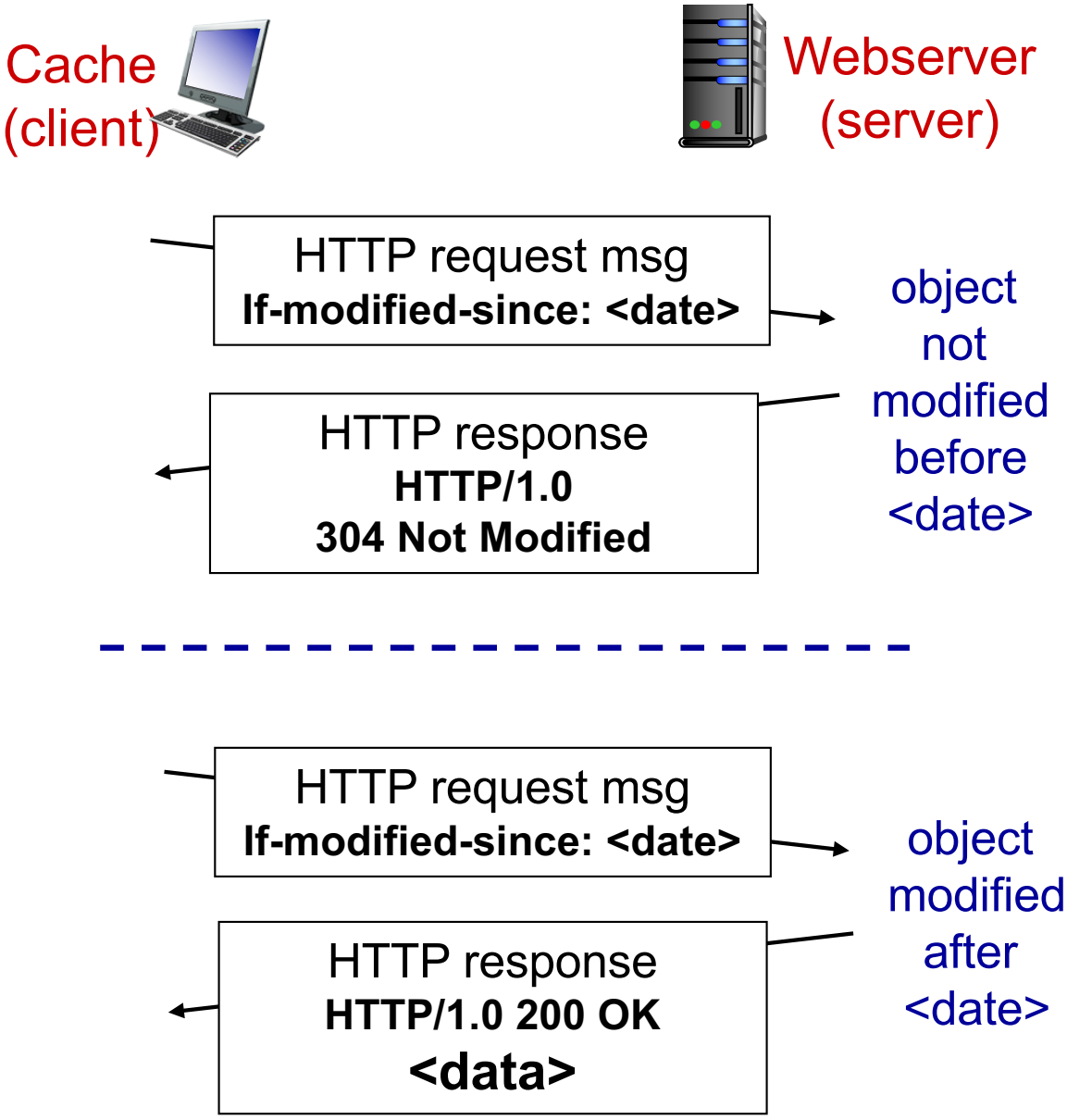


Conditional GET

- *Goal:* don't send object if cache has up-to-date cached version
  - no object transmission delay
  - lower link utilization
- *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`



HTTP Summary

- Book Section 2.2
  - Main HTTP protocol
    - Request, Response
    - Persistent, Non-Persistent
  - Add-ons: Cookies, Caching

- Companion Website
  - Interactive exercises: HTTP GET, RESPONSE
  - Interactive animations: HTTP Delay Estimation

- RFC: http://tools.ietf.org/html/rfc2616 ++