# CSCI 446 Introduction to Computer Networks

**Zhaohong Wang**
**Department of EECE**
**California State University, Chico**

# Topic

## HTTP

# 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.,

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

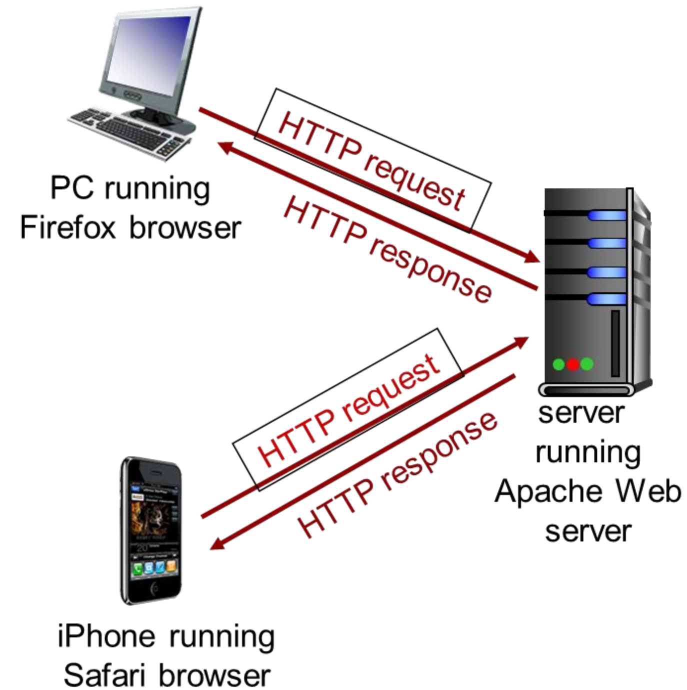
# HTTP Overview (1 of 2)

**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 (2 of 2)

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, server

# Non-Persistent HTTP (1 of 2)

suppose user enters URL:

www.someSchool.edu/someDepartment/home.index

time

**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 (2 of 2)

**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

time

# Non-Persistent HTTP: Response Time

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

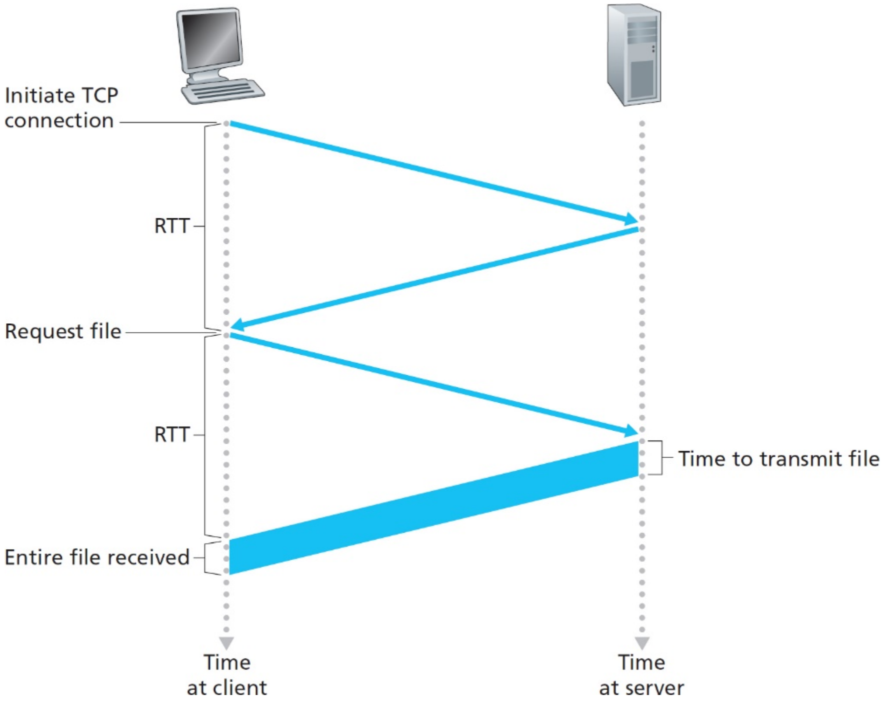
**HTTP response time:**

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 RTT s 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:**
  - ASCII (human-readable format)

carriage return character

line-feed character

request line (GET, POST, HEAD commands)

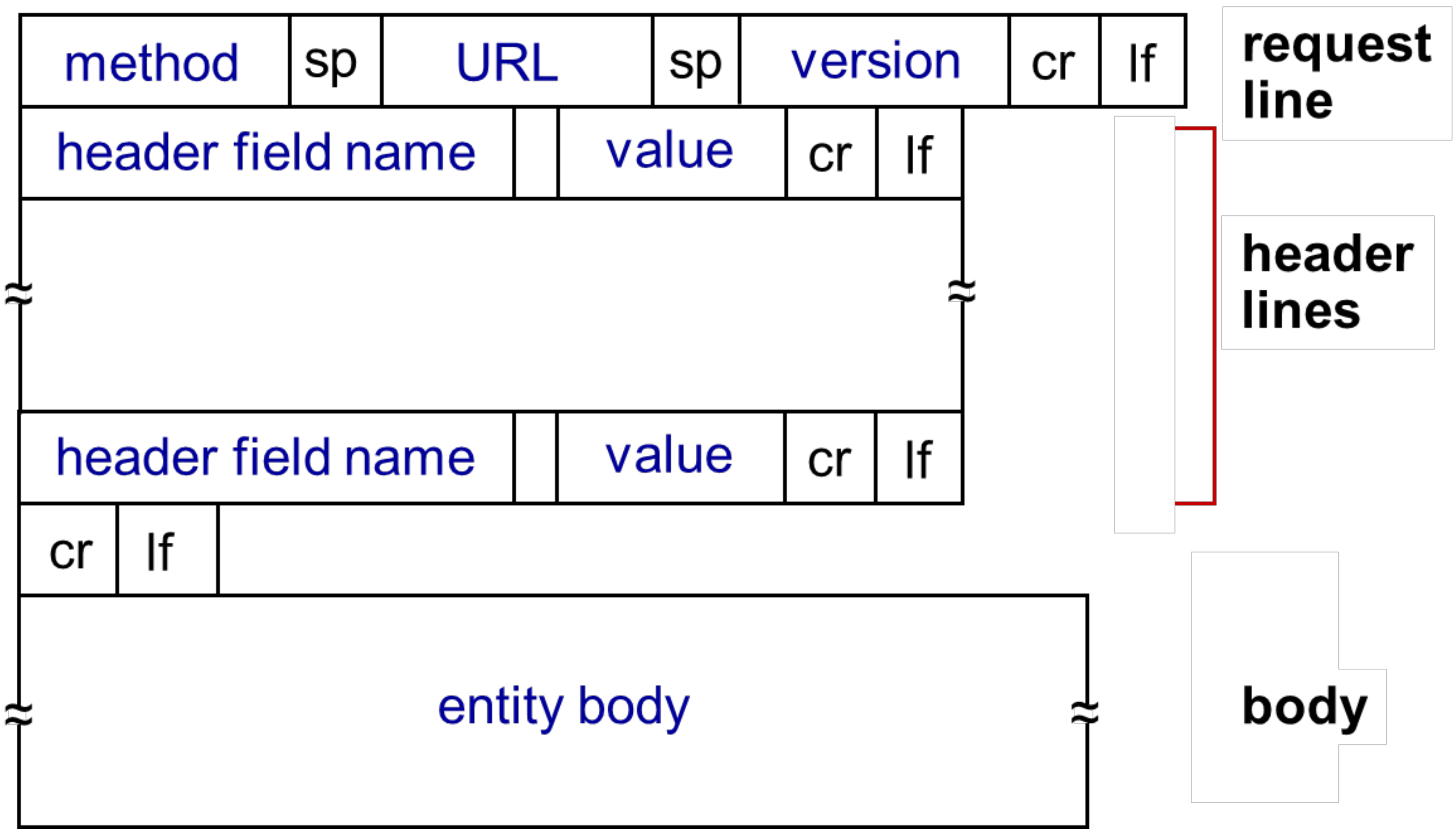
header lines

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

```
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
```

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

**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 ...
```

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

**`400 Bad Request`**

- request msg not understood by server

**`404 Not Found`**

- requested document not found on this server

**`505 HTTP Version Not Supported`**

# Trying out HTTP (Client Side) for Yourself

1. Telnet to your favorite Web server:

```
telnetgaia.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)

# User-Server State: Cookies

many Web sites use cookies

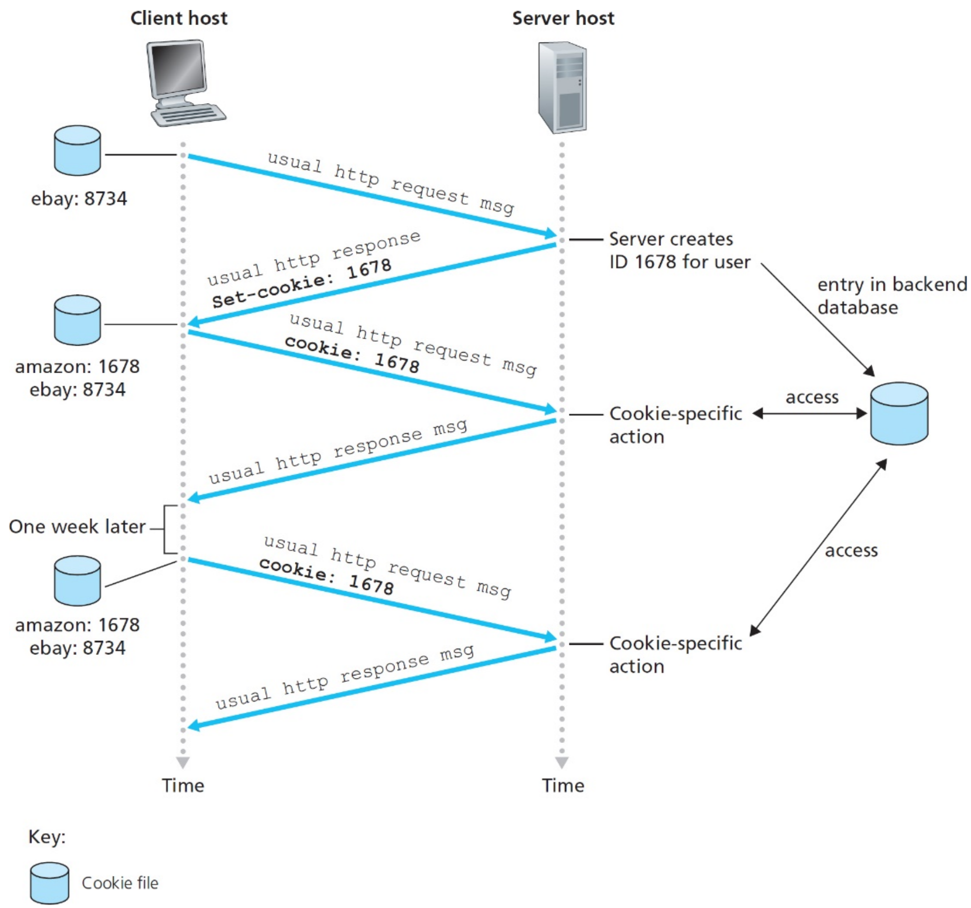
**four components:**

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"



# Cookies

**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

**aside**

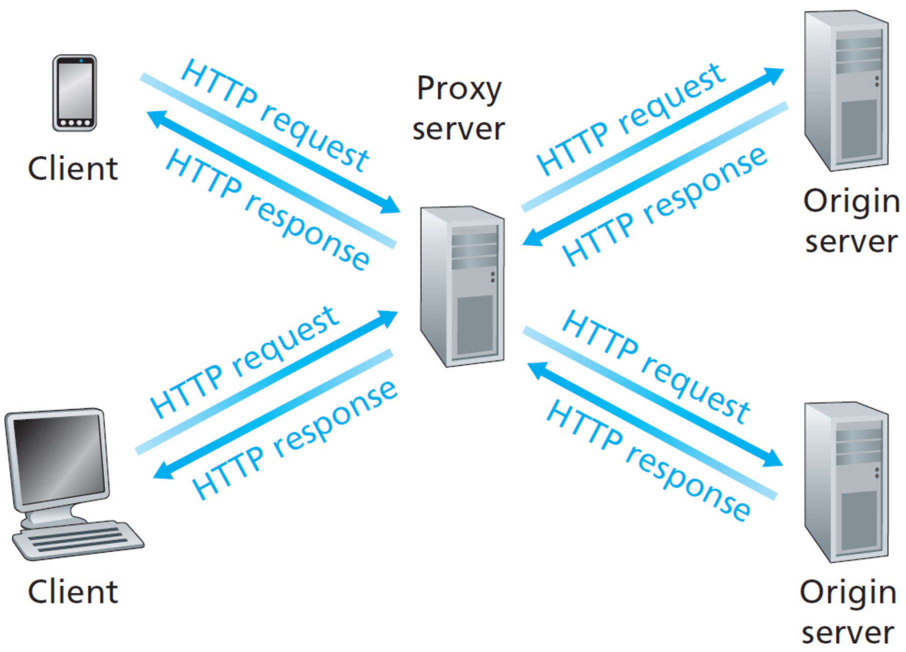
**cookies and privacy:**

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

# 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)

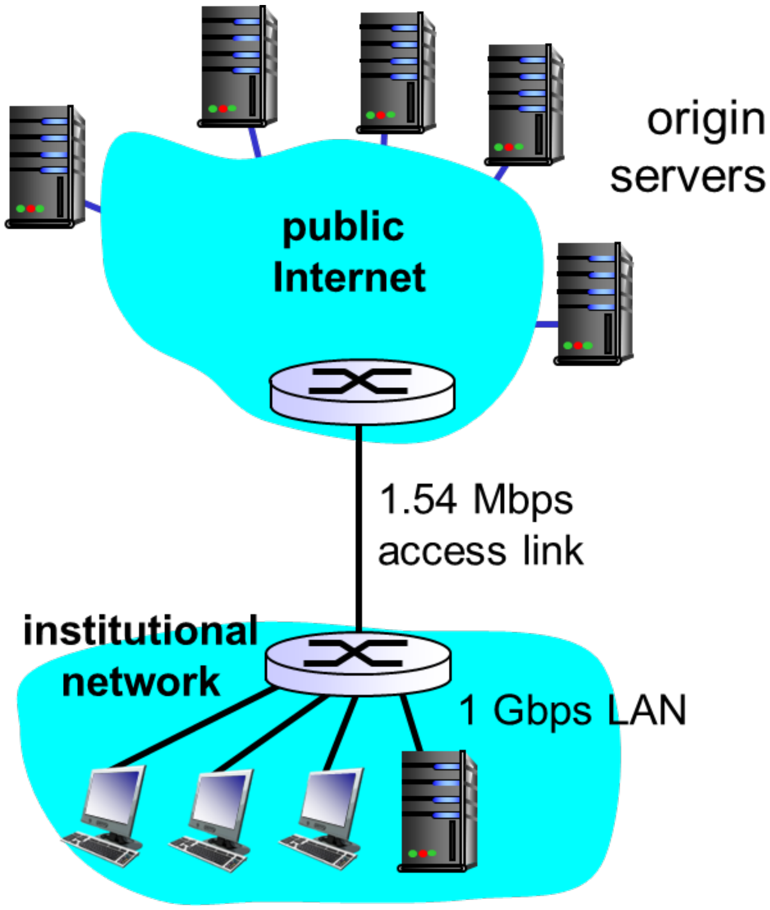
**why Web caching?**

- 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 (so too does P2P file sharing)

# Caching Example: (1 of 2)

**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



# Caching Example: (2 of 2)

**consequences:**

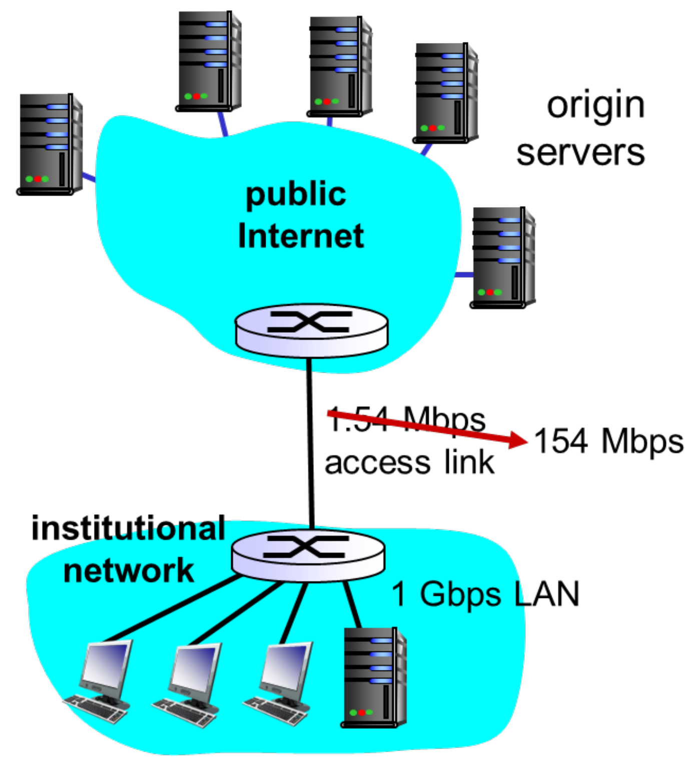
- LAN utilization: 15%
- access link utilization = 99% problem!
- total delay = Internet delay + access delay + LAN delay

  = 2 sec + minutes + usecs

# Caching Example: Fatter Access Link (1 of 2)

**assumptions:**

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



# Caching Example: Fatter Access Link (2 of 2)

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 (1 of 3)

**assumptions:**

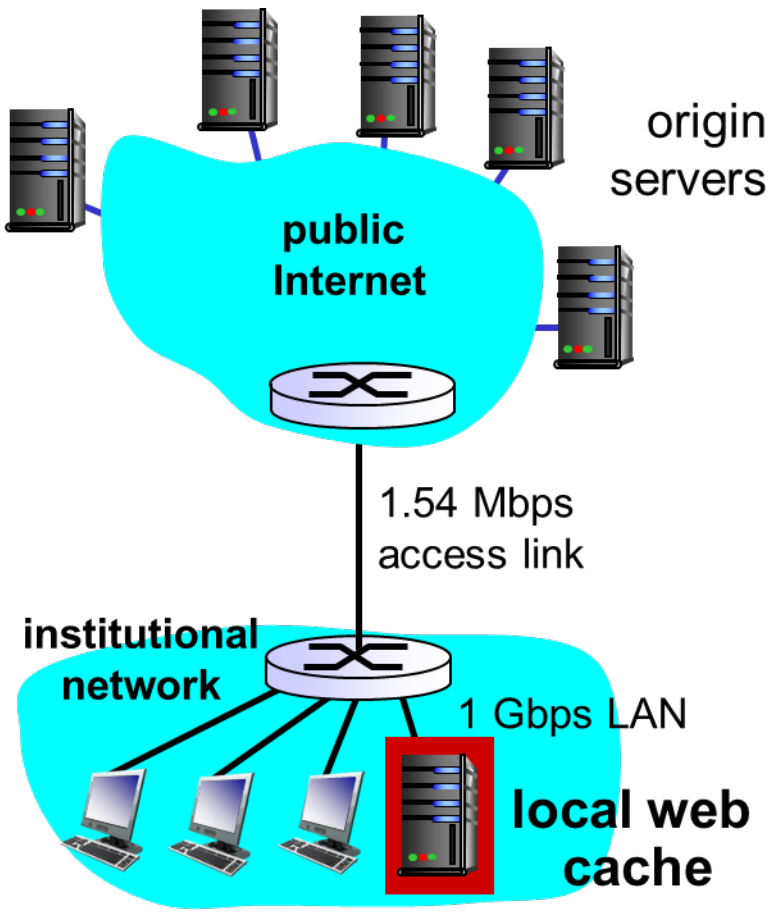
avg object size: 100K bits

avg request rate from browsers to origin servers:15/sec

avg data rate to browsers: 1.50 M bps

RTT from institutional router to any origin server: 2 sec

access link rate: 1.54 Mbps



# Caching Example: Install Local Cache (2 of 3)

**consequences:**

- LAN utilization: 15%
- access link utilization = 100?
- total delay = Internet delay + access delay + LAN delay 2 sec + minutes + usecs

**How to compute link utilization, delay?**

**Cost:** web cache (cheap!)

# Caching Example: Install Local Cache (3 of 3)

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 $= 0.6 * 1.50 \text{ Mbps} = .9 \text{ Mbps}$
  - utilization $= \frac{0.9}{1.54} = .58$
- total delay
  - $= 0.6 * (\text{delay from origin servers}) + 0.4 * (\text{delay when satisfied at cache})$
  - $= 0.6\,(2.01) + 0.4\,(\sim \text{millisecs}) = \sim 1.2 \text{ secs}$
  - less than with 154 M b p s link (and cheaper too!)

# Conditional GET

**Goal:** don’t send object if cache has up-to-date cached version

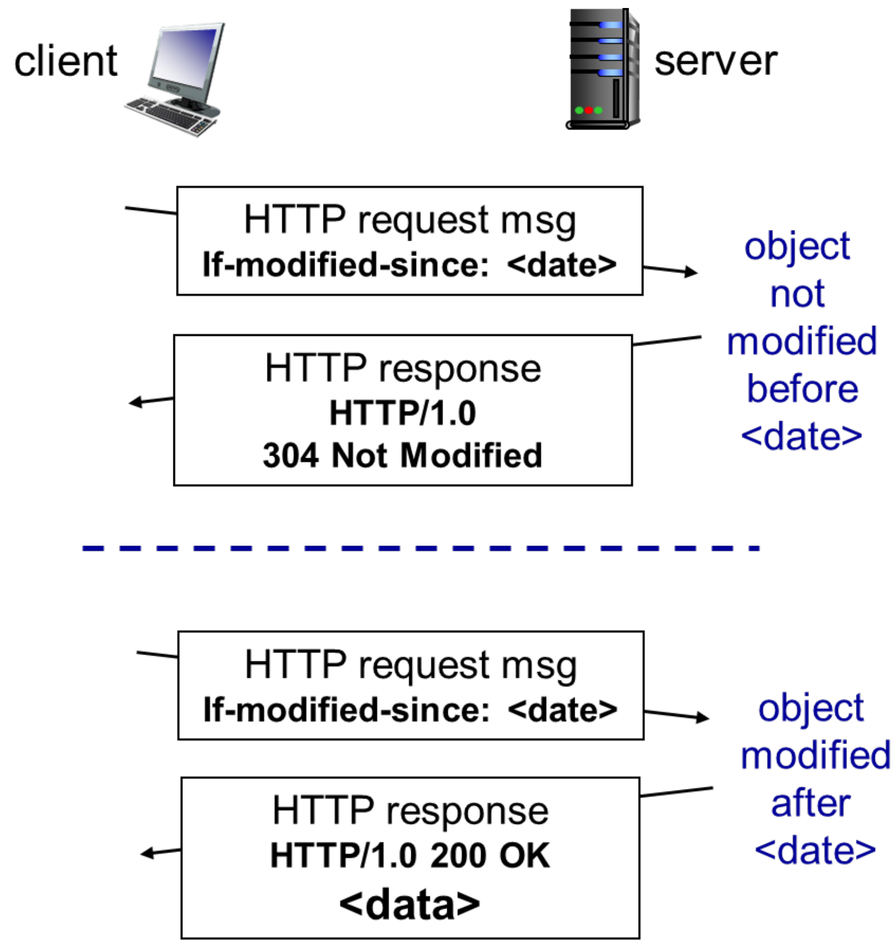
- no object transmission delay
- lower link utilization

**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**



# Thank you!