Networks Lecture 3

Paolo Ciancarini

Innopolis University

January 24, 2022

Source of the material

- This lecture is based on the following two resources
  - Chapter 15 of Computer Science Illuminated (3rd Edition) by Nell Dale and John Lewis
  - Chapter 1, 2 of Computer Networking: A Top-Down Approach
    (8th edition) by Jim Kurose and Keith Ross
  - The material is aligned and add/deleted according to the need of the students.

Topic of the lecture

- Principles of Network Applications
- The transport service
- The Web and HTTP
- Cookies
- Web caching

Addressing Processes

- To receive messages, process must have *identifier*
- Host device has unique 32-bit IP address
- **Q:** does IP address of host on which process runs suffice for identifying the process?
- **A:** no, many processes can be running on same host
- *Identifier* includes both IP address and port numbers associated with process on host.

- Example port numbers:
  - HTTP server: 80
  - Mail server: 25

- To send HTTP message to web server:
  - IP address: 128.119.245.12
  - Port number: 80

App-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

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

Proprietary protocols
e.g., Skype

What transport service does an app need?

- **Data integrity**
  - 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**
  - Encryption, data integrity, …

Requirements of selected apps: Common Apps

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 video:10kbps-5Mbps	yes, 100’s msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100’s msec
text messaging	no loss	elastic	yes and no

Services provided by transport protocol to apps

TCP Service:

- *Reliable transport* between sending and receiving processes
- *Flow control:* the sender will not overwhelm the 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:

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

Internet apps: Application, Transport Protocols

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 5321]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 7230]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), DASH	TCP or UDP
Internet telephony	SIP[RFC 3261], RTP[RFC 3550], proprietary (eg Skype)	TCP or UDP

TCP vs UDP

TCP

- Slower but more reliable transfers
- Typical Applications:
  - File Transfer Protocol (FTP)
  - Web Browsing
  - Email

unicast

UDP

- Faster but not guaranteed transfers ("best effort")
- Typical Applications:
  - Live Streaming
  - Online Games
  - VoIP

unicast multicast broadcast

Securing TCP

- TCP & UDP
  - No encryption
  - Cleartext passwords sent into socket traverse Internet in cleartext
- SSL (Secure Socket Layer)
  - Provides encrypted TCP connection
  - Data integrity
  - End-point authentication
- SSL is at app layer
  - Apps use SSL libraries, which "talk" to TCP
- SSL socket API
  - Clear text passwords sent into socket traverse Internet encrypted

  - Remark: Now SSL is called TLS Transport Layer Security
  - HTTP Over TLS is RFC 2818

The Application Layer

- Principles of network applications
- Web and HTTP
- FTP

Web and HTTP

First, a review…

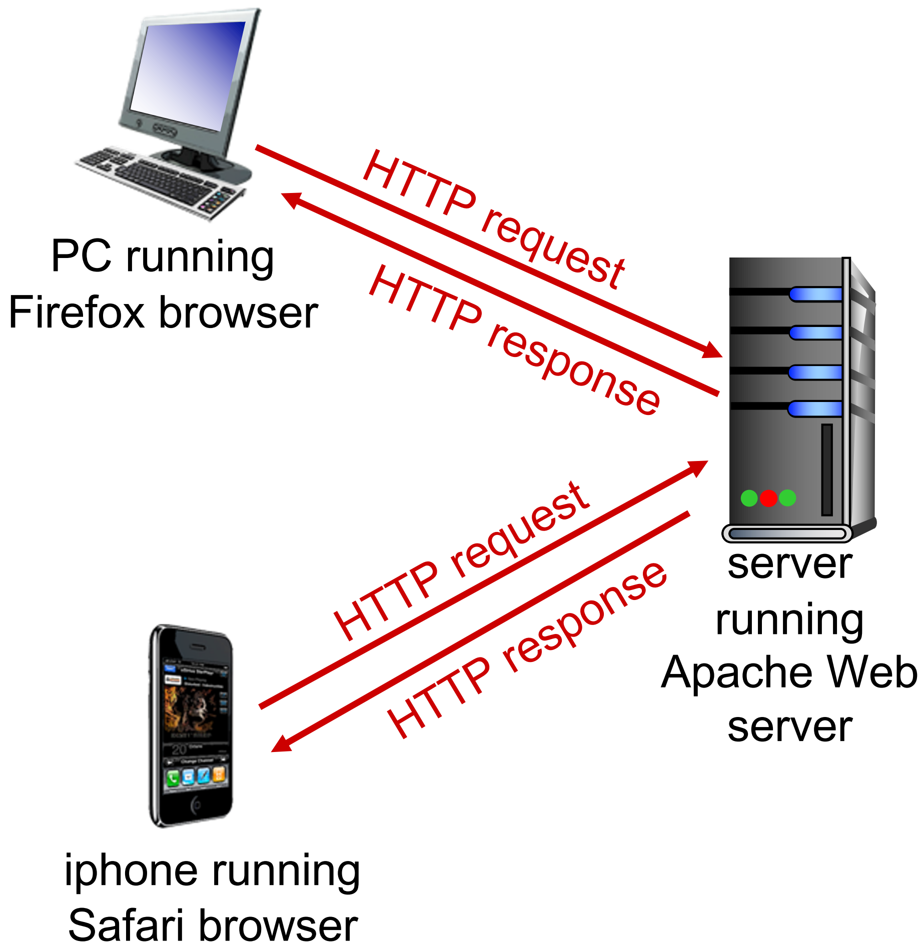
- *A web page* includes some *objects* and some *links*
- An object can be a HTML file, JPEG image, Java applet, audio file,…
- A web page consists of *base HTML-file* which includes *several referenced objects*
- each object is addressable by a *URL,* e.g.,

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

HTTP Overview – 1/2

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 – 2/2

- *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

- 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 a single TCP connection between the client and the 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

time

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

time

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

time

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

tim
e

4. HTTP server closes TCP connection.

Non-persistent HTTP

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

tim
e

Non-persistent HTTP

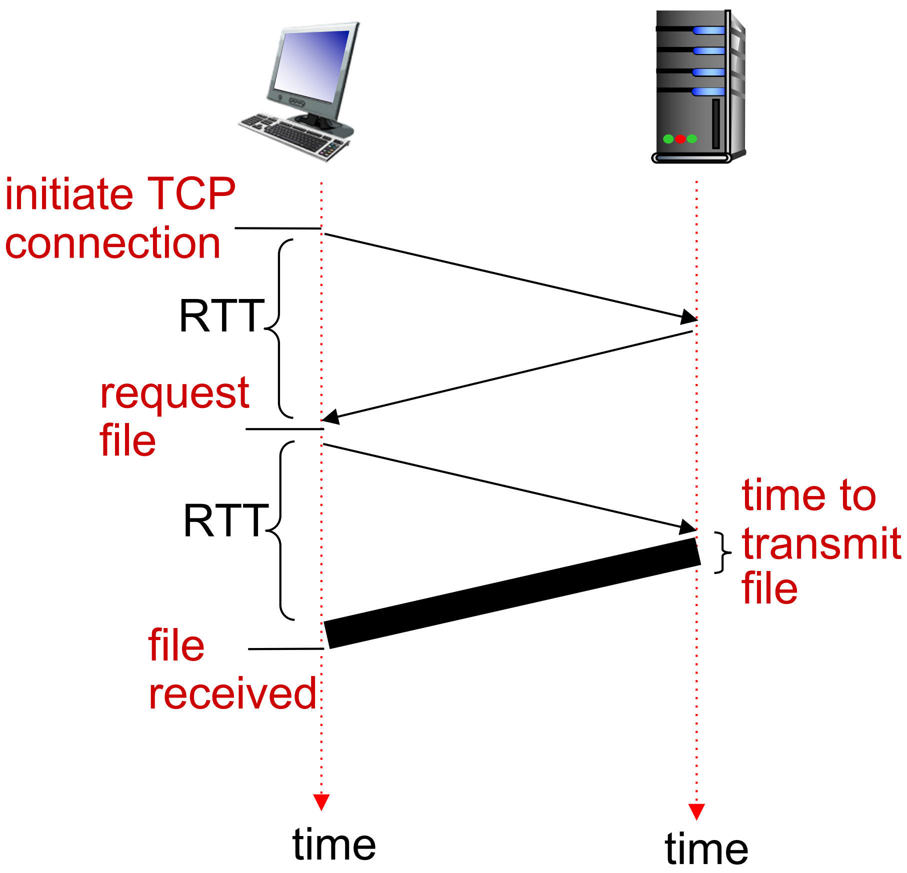
4. HTTP server closes TCP connection.

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

tim
e

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

Non-persistent HTTP: Response Time



RTT (definition): **round trip 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 the 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

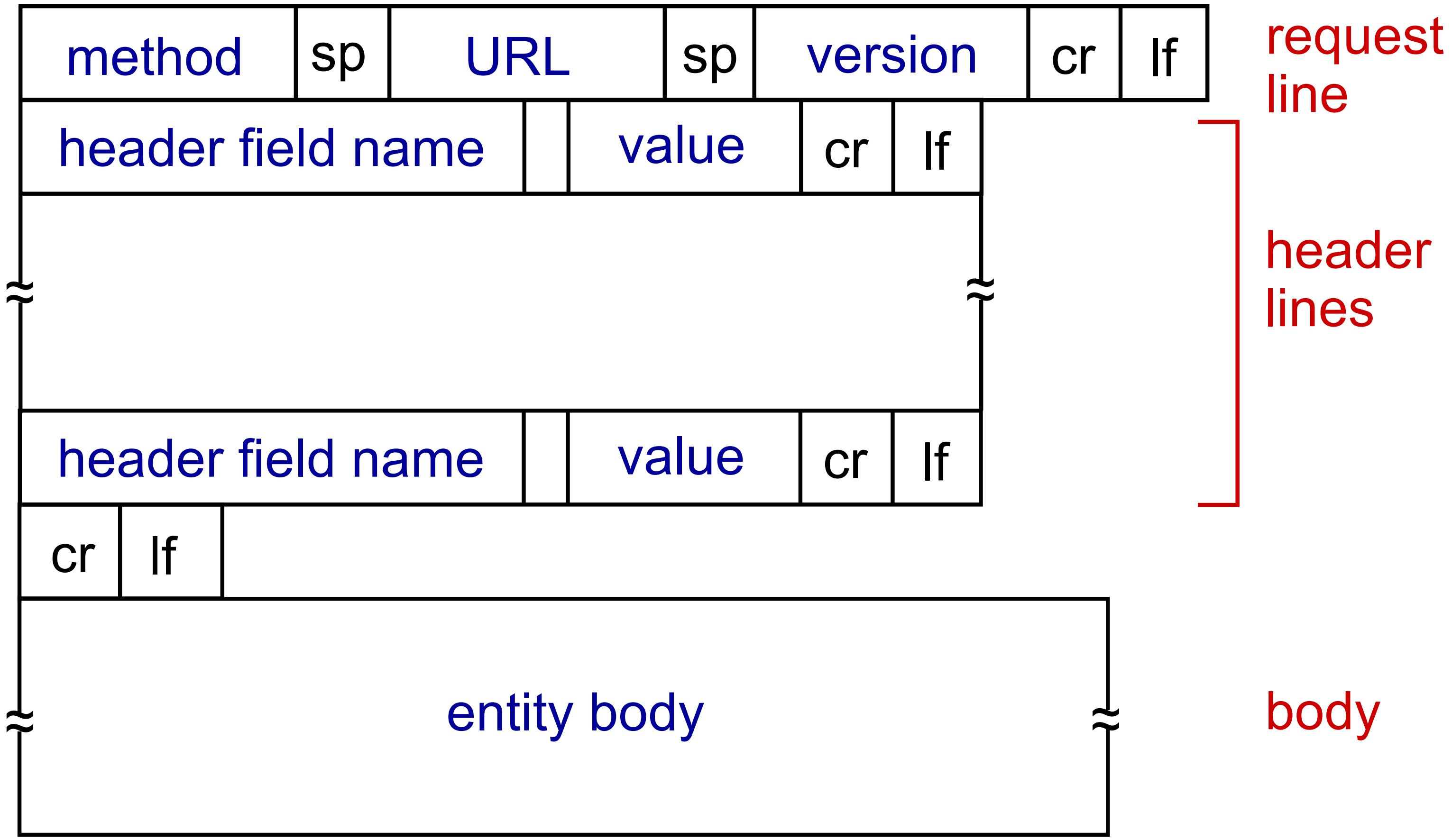
request line (GET, POST, HEAD commands)

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
```

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 ...
```

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

User Server State: cookies

Many Websites 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

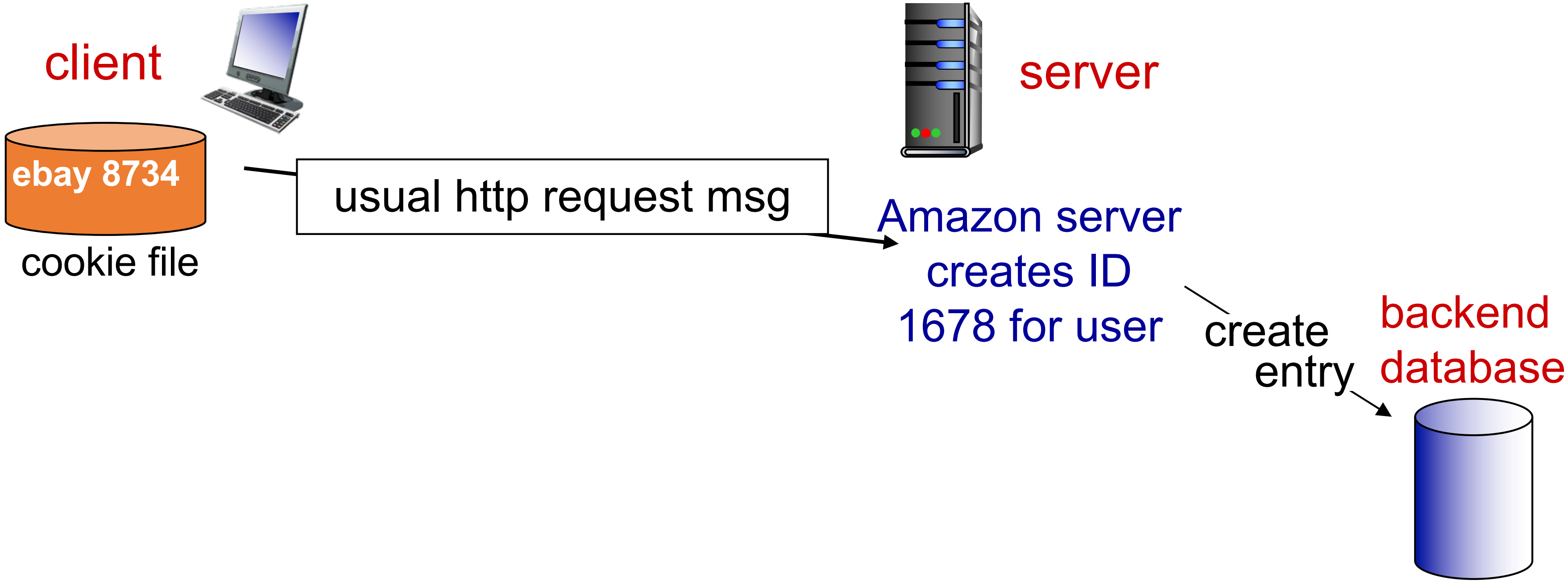
Example:

- Susan always accesses Internet from her PC
- Visits specific e-commerce site for first time
- When initial HTTP request arrives at site, site creates:
  - Unique ID
  - Entry in backend database for ID

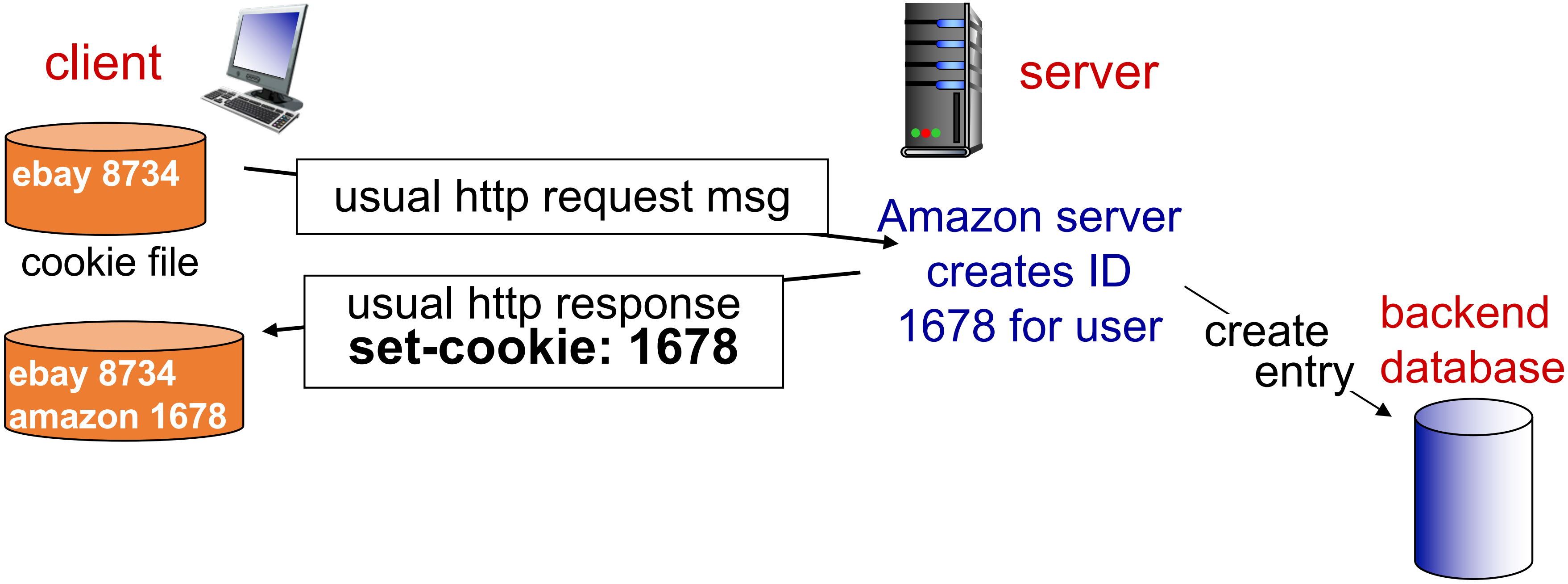
More detail in tutorial session

Cookies: keeping "state" (cont.)

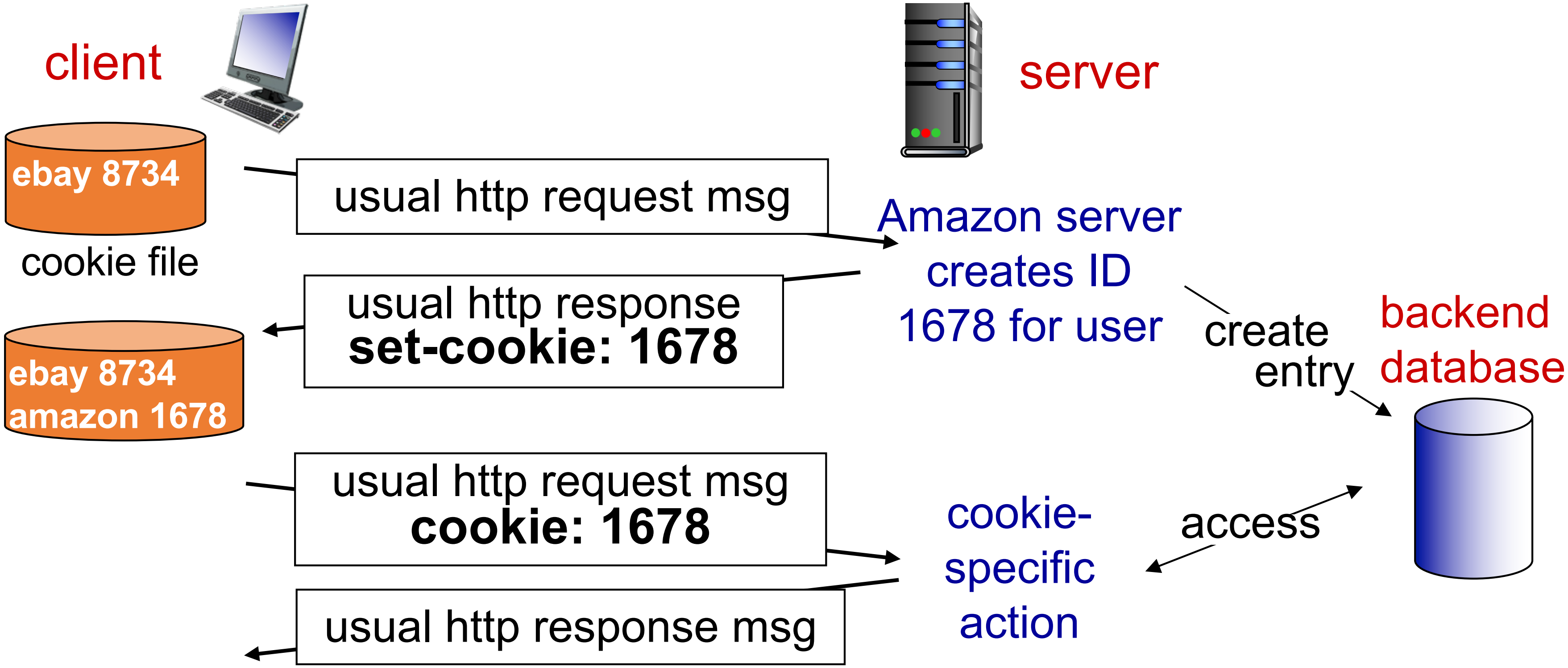
Cookies: keeping "state" (cont.)



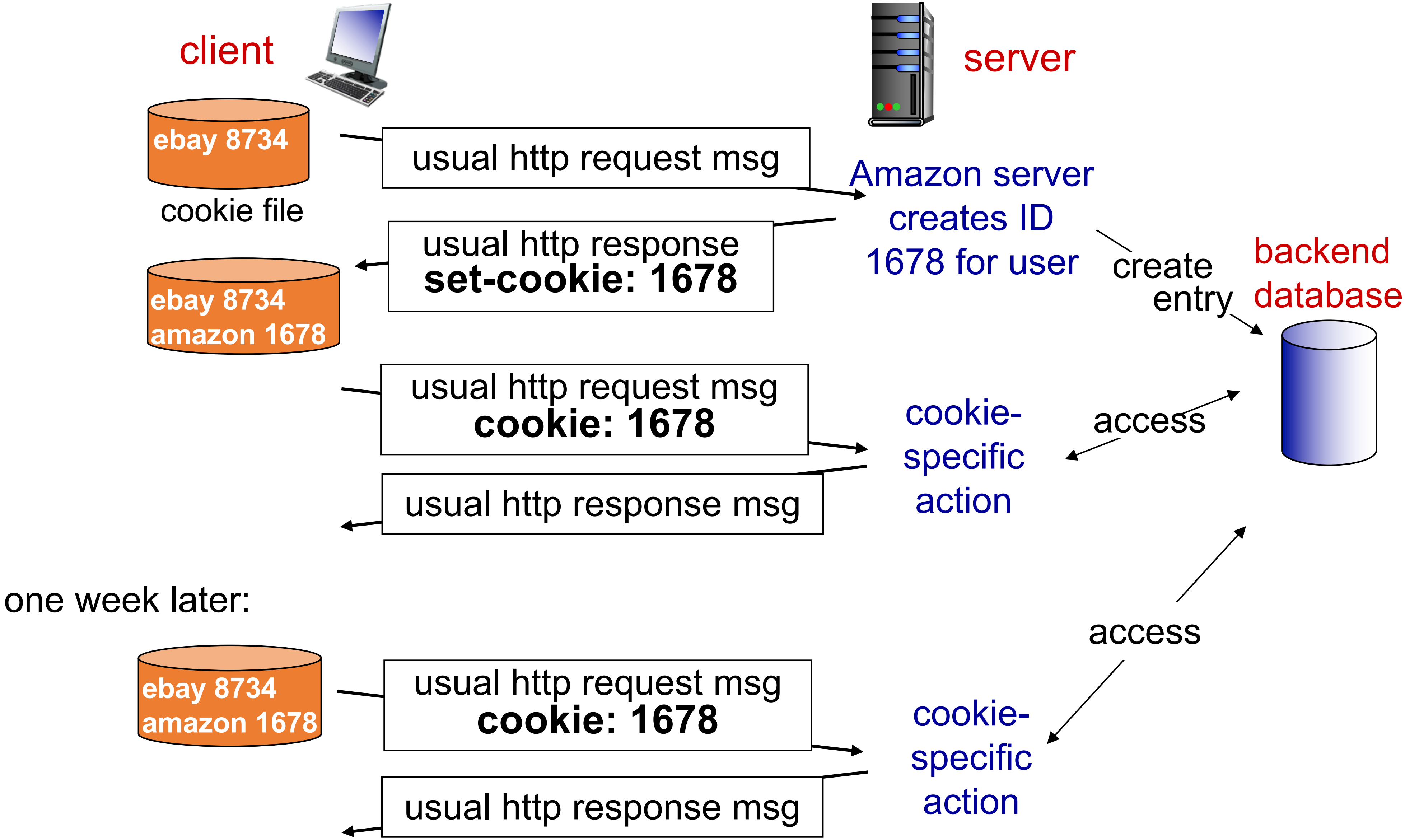
Cookies: keeping "state" (cont.)



Cookies: keeping "state" (cont.)



Cookies: keeping "state" (cont.)



Cookies (continued)

What cookies can be used for:

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

aside

Cookies and privacy:

- cookies permit sites to learn a lot about you ("user profiling")
- you may supply name and e-mail to sites

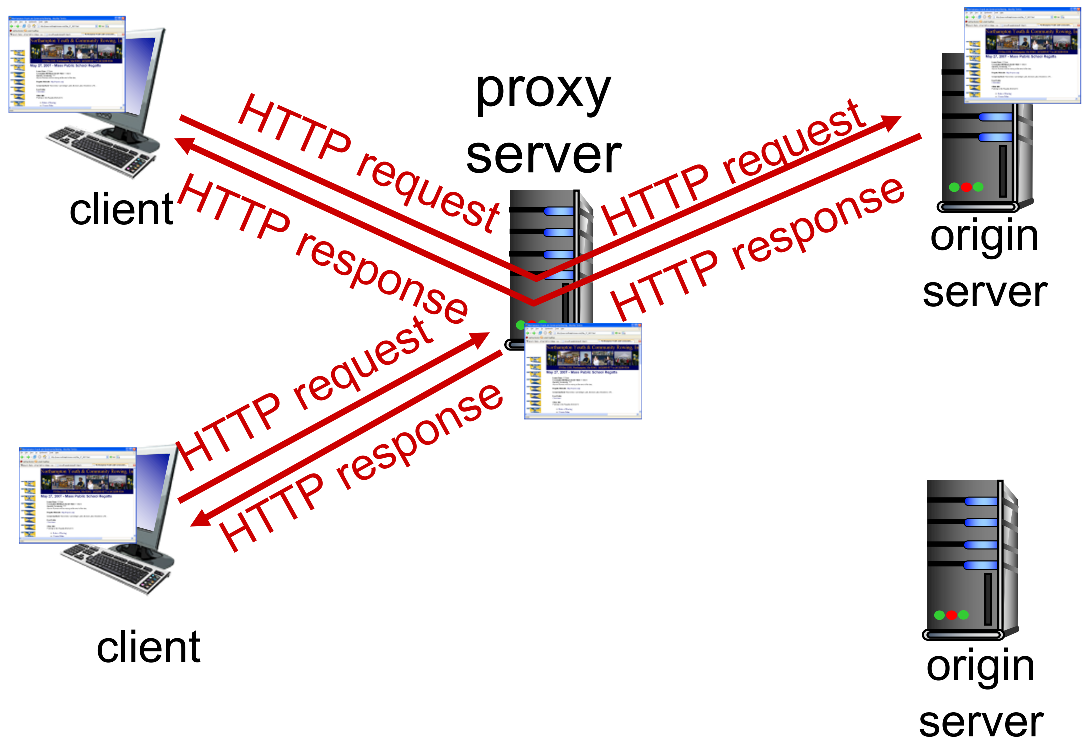
How to keep "state":

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

Web cache (or proxy Server)

Goal: satisfy client request without involving origin server

- User sets her browser: Web accesses via cache
- Browser sends all HTTP requests to cache
  - If object in cache: then 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 the cache is installed by your ISP (eg. university, company, or residential ISP)

- *Why Web caching?*
  - Reduce response time for client requests
  - Reduce traffic on an institution’s access link
  - Internet is dense with caches: enables “weak” content providers to effectively deliver content

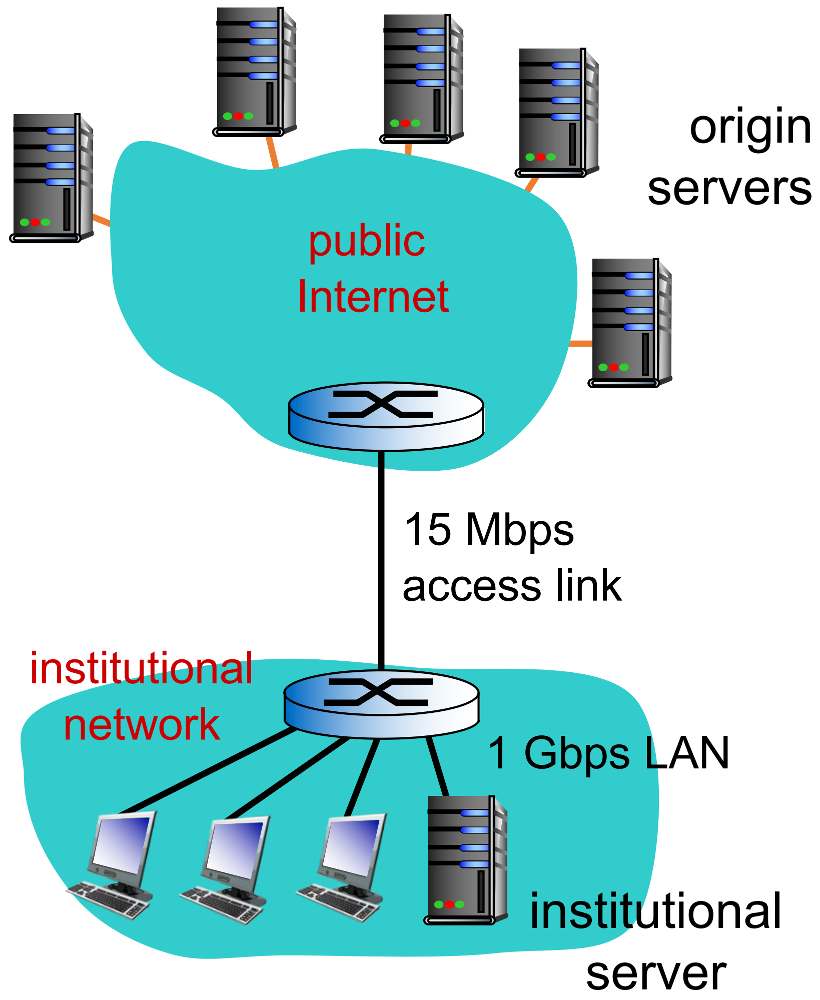
Caching Example

Assumptions:

- avg object size: 1 Mbits
- avg request rate from browsers to origin servers:15reqs/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 15 Mbps

Consequences:

- LAN utilization: 15%
- access link intensity = 99%
- total delay =
  Internet delay + access delay + LAN delay
  = 2 sec + minutes + ~0



Traffic intensity: La/R
where
La is the average lenght and R the transmission rate

Caching Example

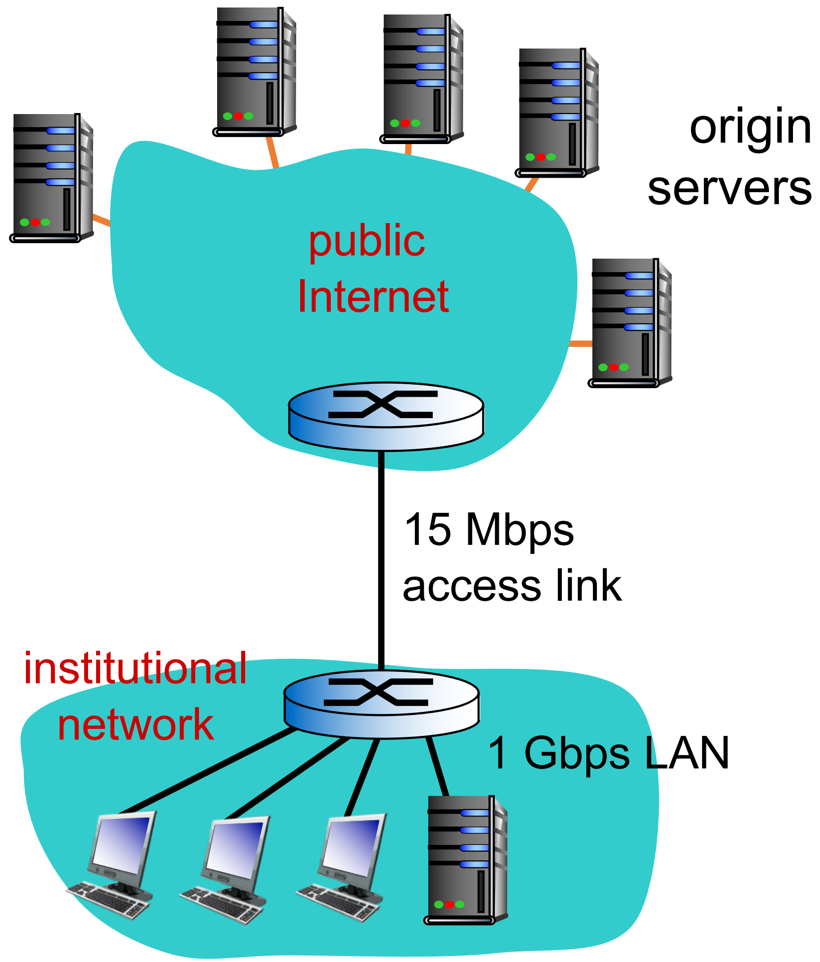
Assumptions:

- avg object size: 1M 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: 15 Mbps

Consequences:

problem!

- LAN utilization: 15%
- access link intensity = 99%
- total delay =
  Internet delay + access delay + LAN delay
  = 2 sec + minutes + ~0



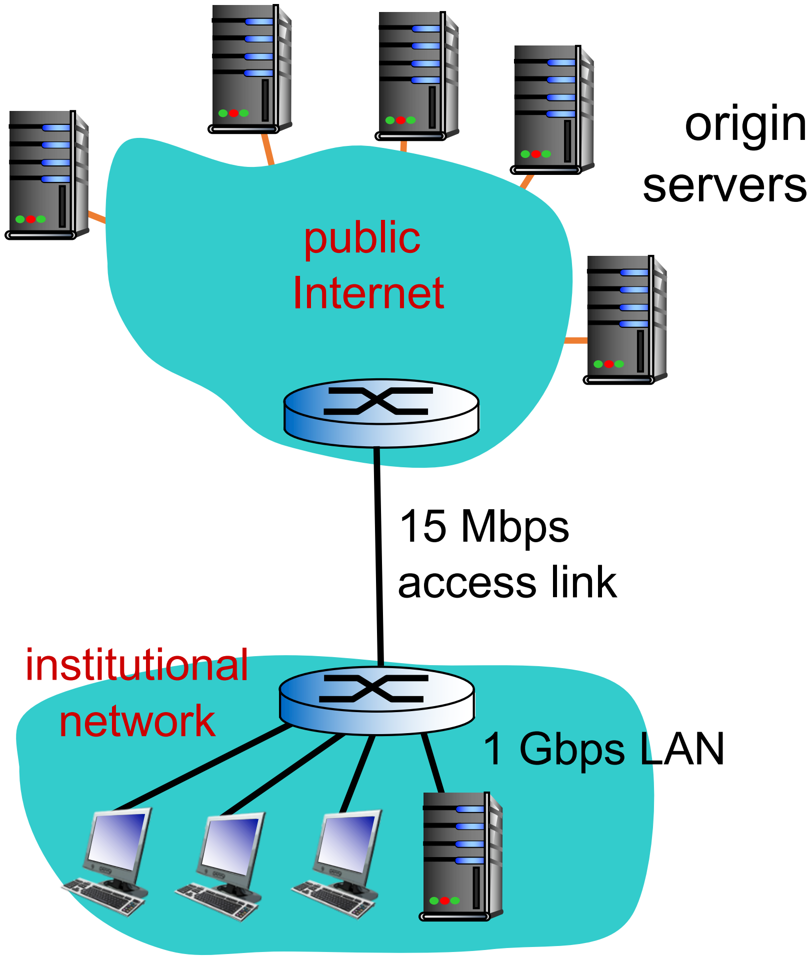
Caching Example: fatter access link

Assumptions:

- avg object size: 1M 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: 15 Mbps

Consequences:

- LAN utilization: 15%
- access link intensity = 99%
- total delay = Internet delay + access delay + LAN delay = 2 sec + minutes + ~0



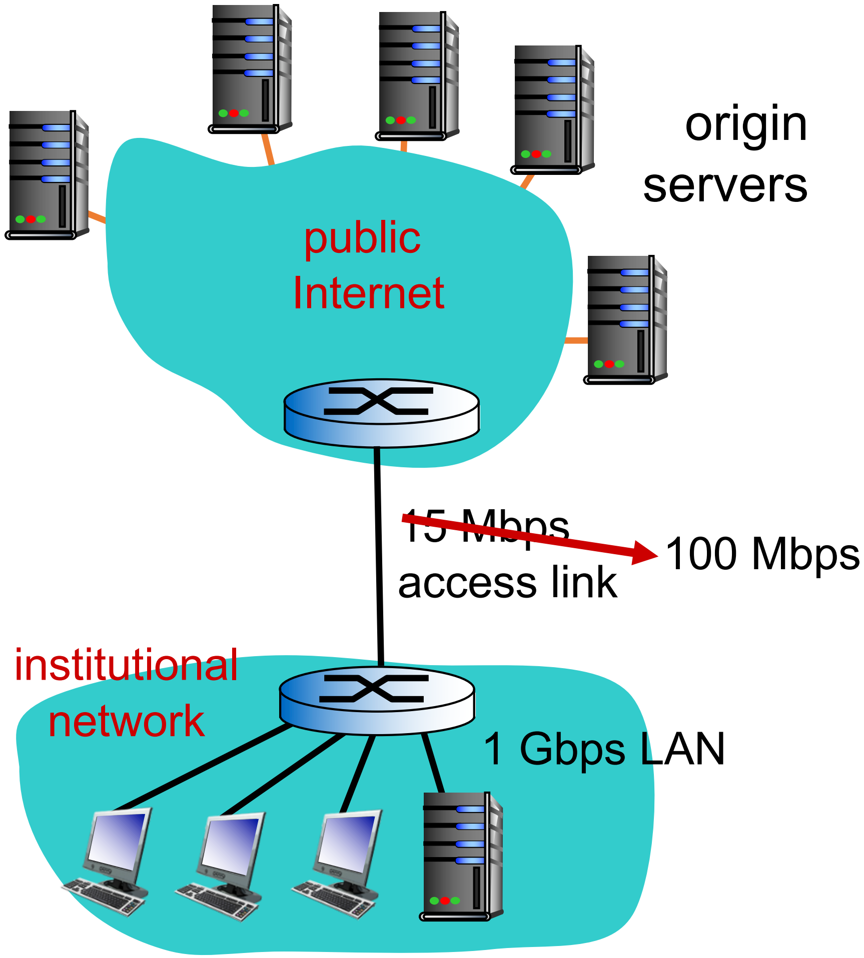
Caching Example: fatter access link

Assumptions:

- avg object size: 1M 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: ~~15 Mbps~~ 100 Mbps

Consequences:

- LAN utilization: 15%
- access link intensity = ~~99%~~ 9.9%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + ~~minutes~~ + ~0
  msecs



Cost: increased access link speed (not cheap!)

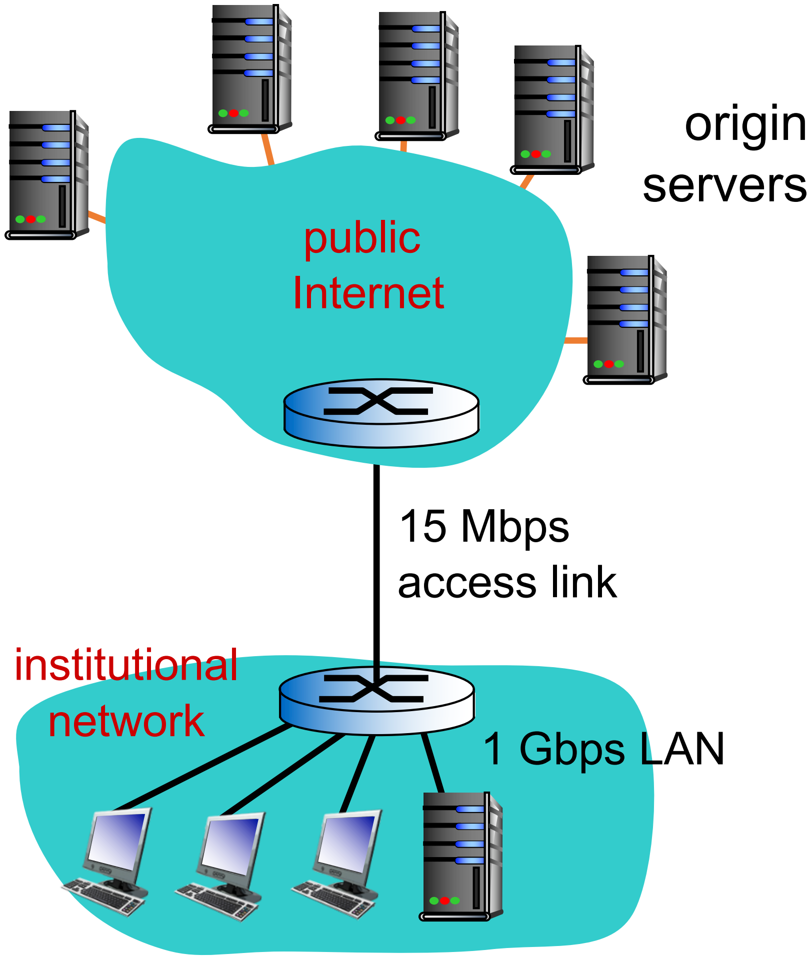
Caching example: install local cache

Assumptions

- Avg object size: 1M 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: 15 Mbps

Consequences

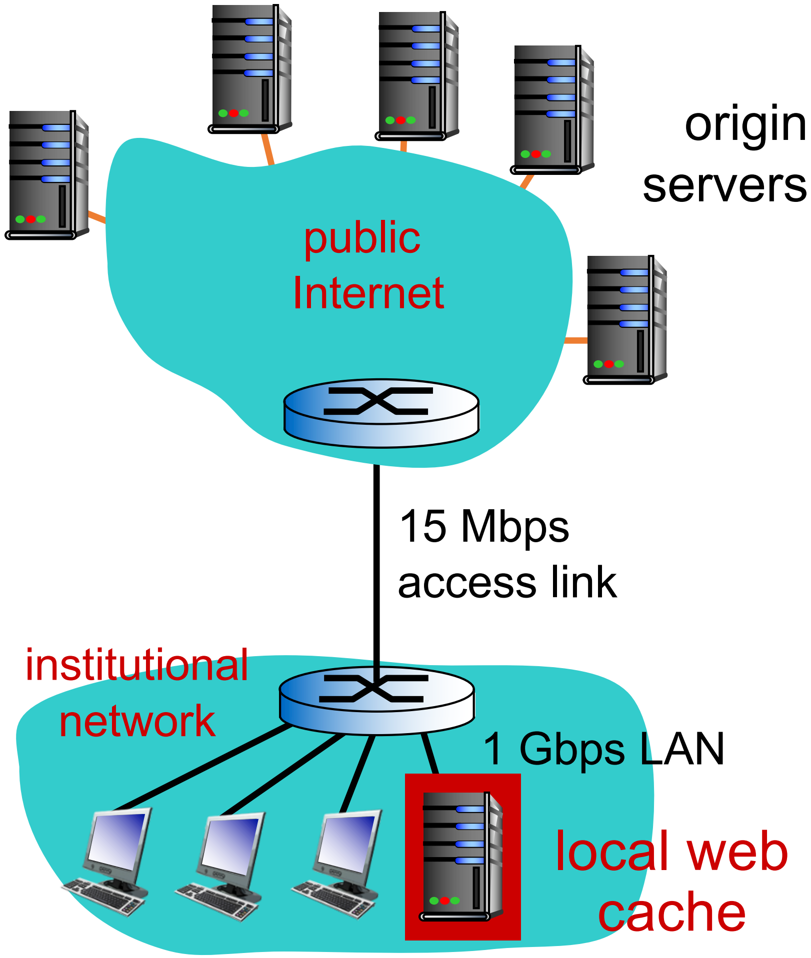
- LAN utilization: 15%
- Access link intensity =
- Total delay =



Caching example: install local cache

Assumptions
- Avg object size: 1M 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: 15 Mbps

Consequences
- LAN utilization: 15%
- Access link intensity =
- Total delay =



Cost: web cache (cheap!)

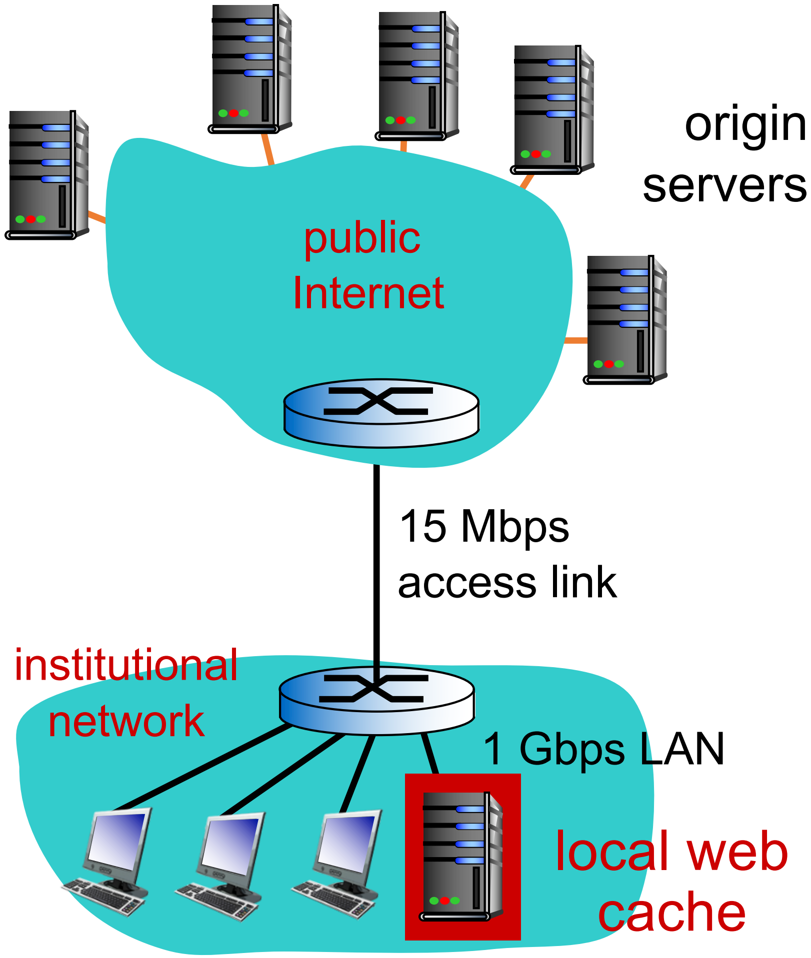
Caching example: install local cache

Assumptions

- Avg object size: 1M 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: 15 Mbps

Consequences

- LAN utilization: 15%
- Access link intensity = **?**
- Total delay = **?**



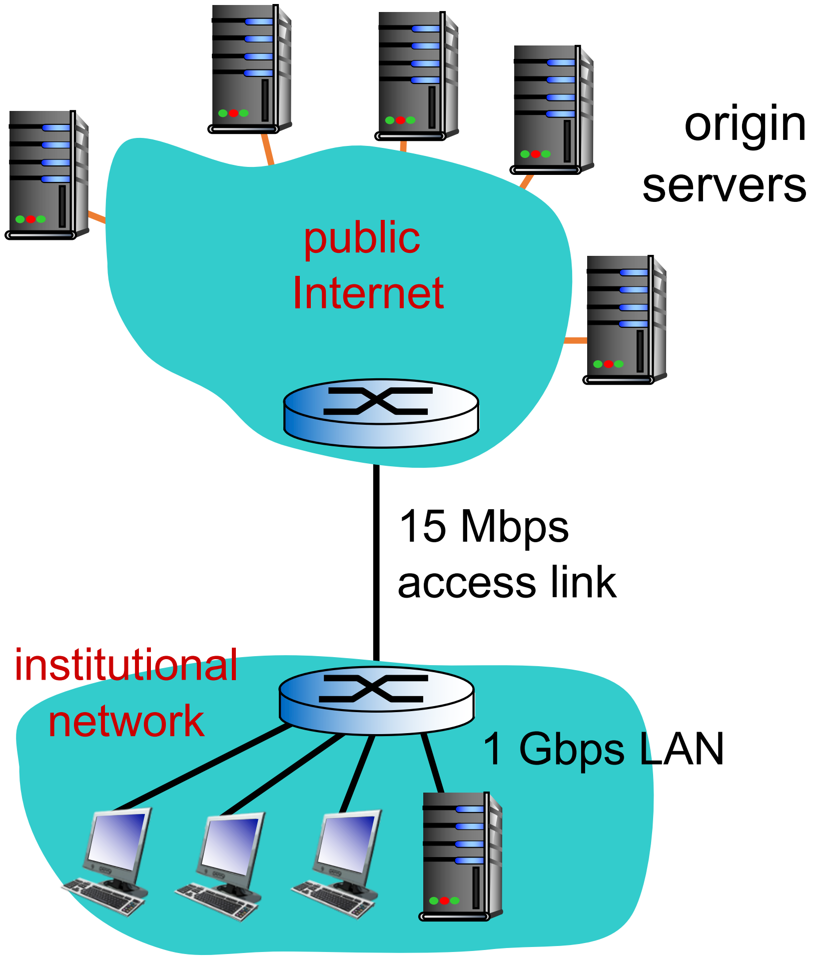
How to compute link intensity, delay?

Cost: web cache (cheap!)

Caching example: install local cache

Calculating access link intensity, delay with cache:

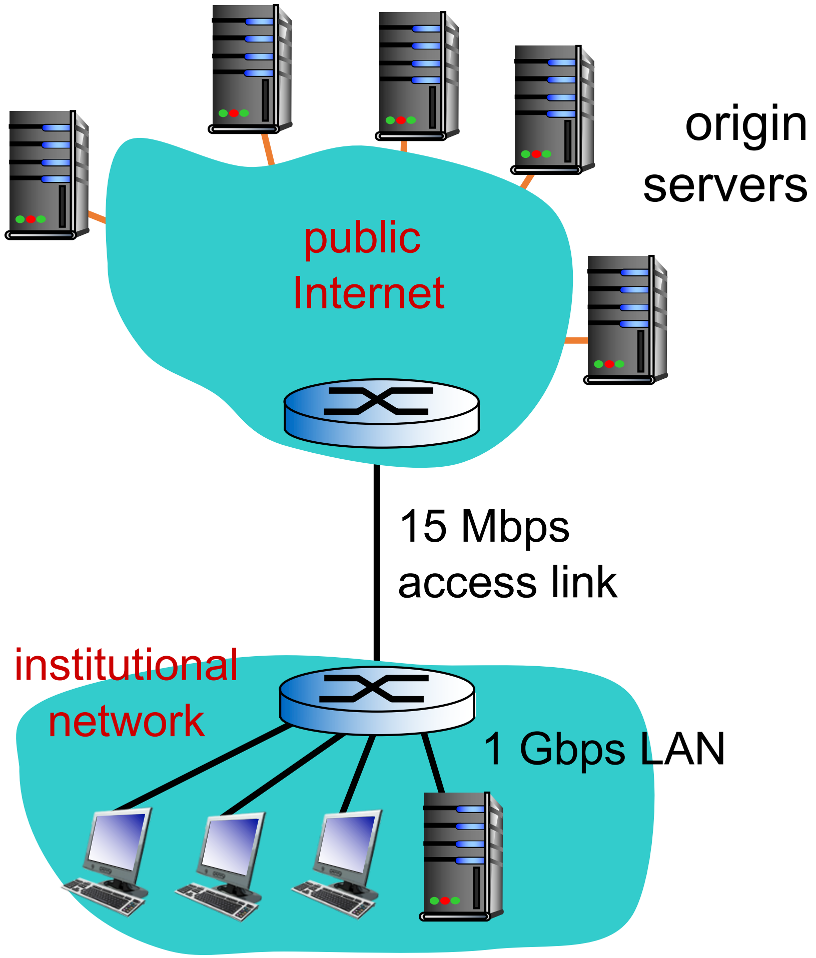
- Suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin



Caching example: install local cache

Calculating access link intensity, delay with cache:

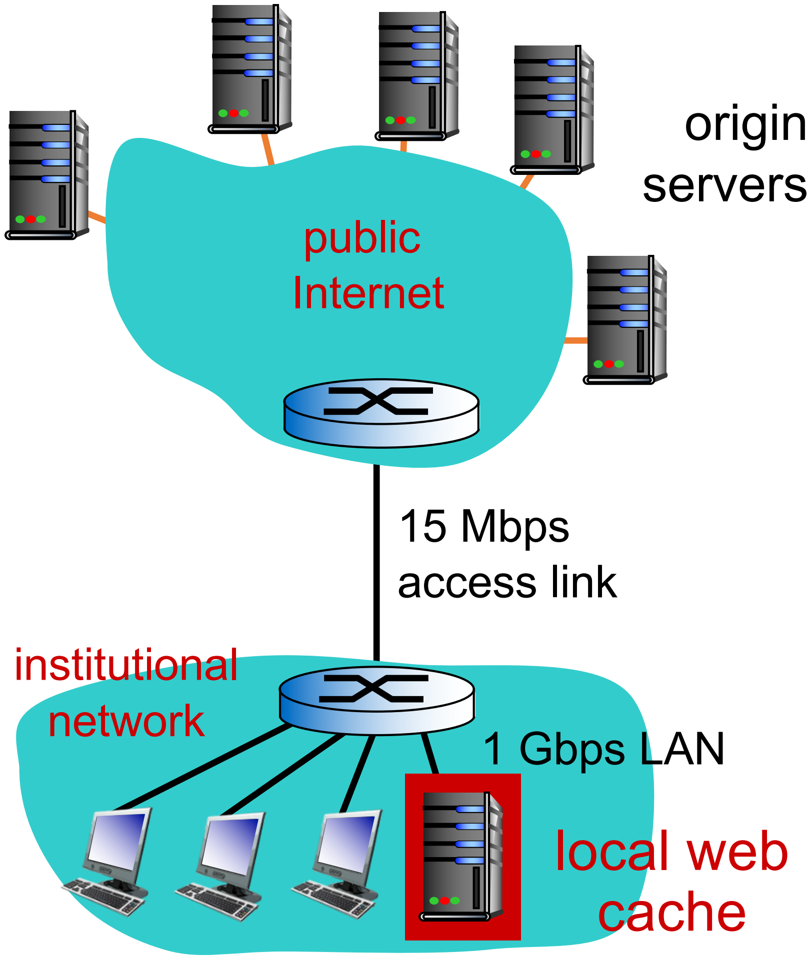
- Suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin
- Access link intensity:
  - 60% of requests use access link
- Data rate to browsers over access link = 0.6*1.50 Mbps = .9 Mbps
  - intensity = 0.9/1.54 = .58



Caching example: install local cache

Calculating access link intensity, delay with cache:

- Suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin
- Access link intensity:
  - 60% of requests use access link
- Data rate to browsers over access link = 0.6*1.50 Mbps = .9 Mbps
  - intensity = 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 100 Mbps link (and cheaper too!)



See 2.2.5 in Kurose 8ed

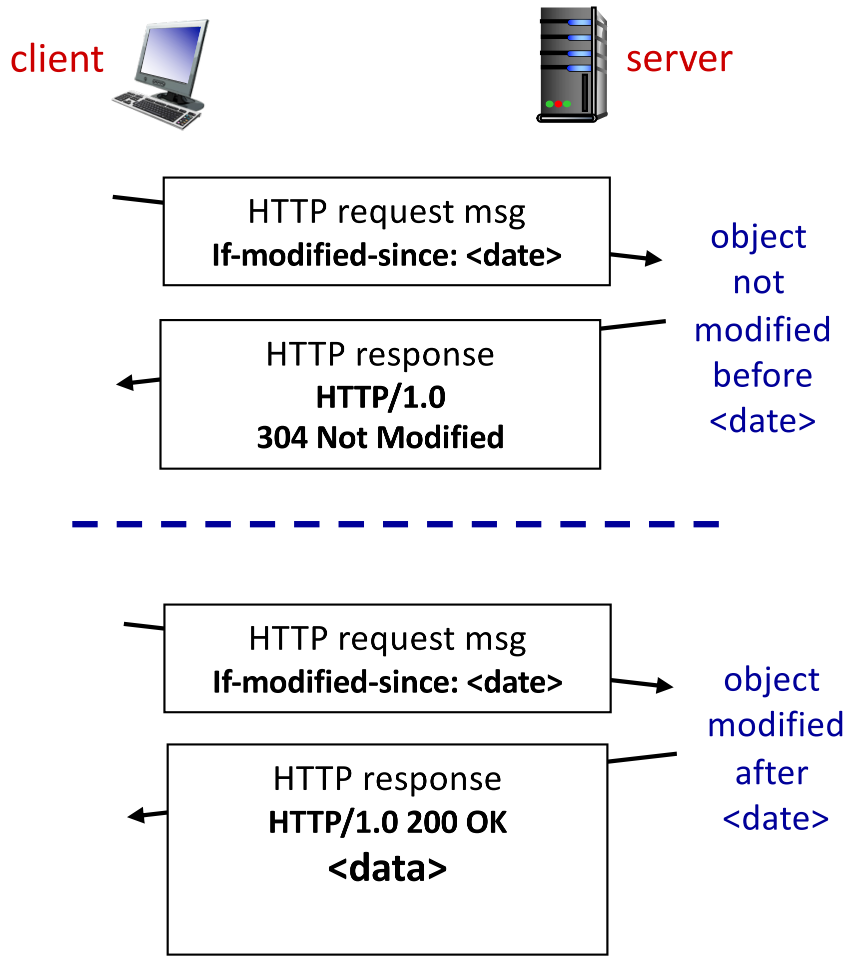
Conditional GET

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

- no object transmission delay (or use of network resources)

- *client:* 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/2

Key goal: decreased delay in multi-object HTTP requests

HTTP1.1: introduced multiple, pipelined GETs over single TCP connection

- server responds *in-order* (FCFS: first-come-first-served scheduling) to GET requests
- with FCFS, small object may have to wait for transmission (head-of-line (HOL) blocking) behind large object(s)
- loss recovery (retransmitting lost TCP segments) stalls object transmission

HTTP/2

Key goal: decreased delay in multi-object HTTP requests

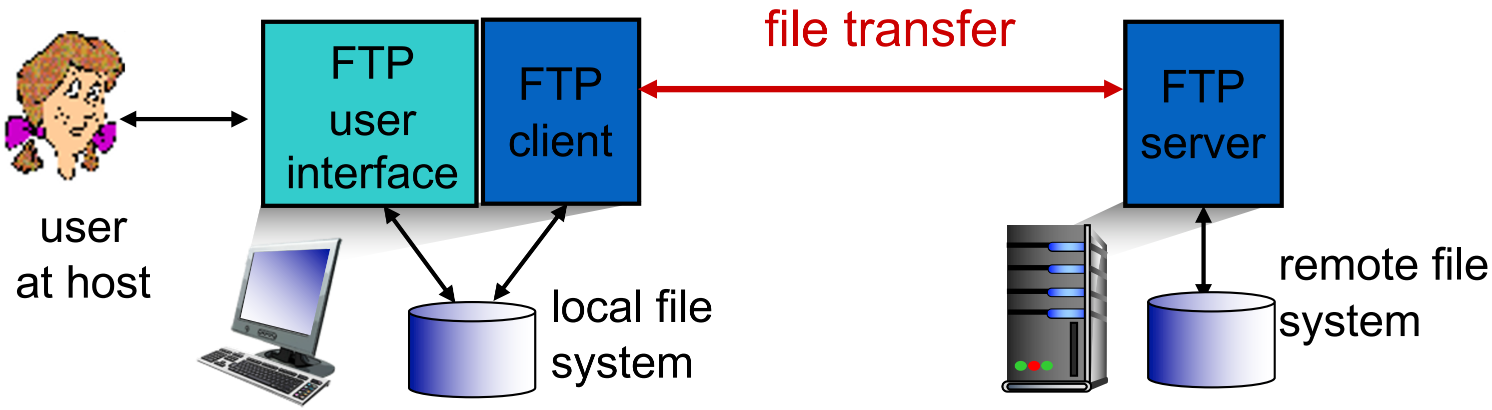
HTTP/2: [RFC 7540, 2015] increased flexibility at *server* in sending objects to client:

- methods, status codes, most header fields unchanged from HTTP 1.1
- transmission order of requested objects based on client-specified object priority (not necessarily FCFS)
- *push* unrequested objects to client
- divide objects into frames, schedule frames to mitigate HOL blocking

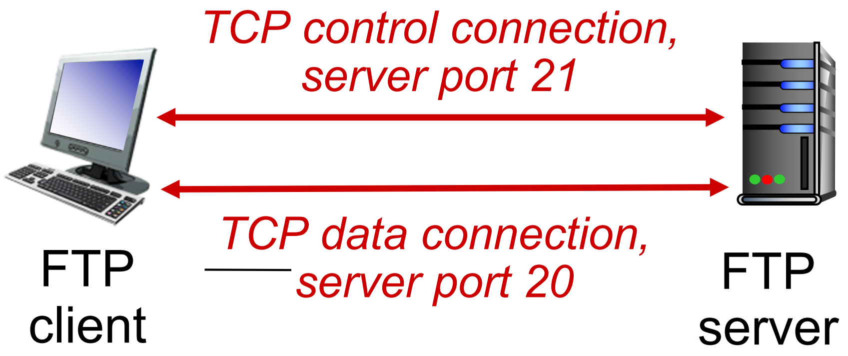
The Application Layer

- Principles of network applications
- Web and HTTP
- FTP

File Transfer Protocol (FTP)



FTP: separate control, data connections



- FTP client contacts FTP server at port 21, using TCP
- Client authorized over control connection
- Client browses remote directory, sends commands over control connection
- When server receives file transfer command, server opens 2nd TCP data connection (for file) to client
- After transferring one file, server closes data connection
- FTP server maintains “state”: current directory, earlier authentication

Summary

- Computer Networks
- Types of Networks
- Open Systems Interconnection (OSI) Model
- The Application Layer
- Application Architecture
- Principles of Network Applications
- Web and HTTP
- FTP