

Computer Networks and Applications

COMP 3331/COMP 9331

Week 3

Application Layer (Web & HTTP)

Chapter 2, Section 2.2

2. Application Layer: outline

2.1 principles of network applications

- app architectures
- app requirements

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 (CDNs)

2.7 socket programming with UDP and TCP



Self study

Self study

The Web – Precursor



Ted Nelson

- ❖ **1967, Ted Nelson, Xanadu:**
 - A world-wide publishing network that would allow information to be stored not as separate files but as connected literature
 - Owners of documents would be automatically paid via electronic means for the virtual copying of their documents
- ❖ **Coined the term “Hypertext”**

Self study

The Web – History



Tim Berners-Lee

- ❖ World Wide Web (WWW): a distributed database of “pages” linked through **Hypertext Transport Protocol (HTTP)**
 - First HTTP implementation - 1990
 - Tim Berners-Lee at CERN
 - HTTP/0.9 – 1991
 - Simple GET command for the Web
 - HTTP/1.0 – 1992
 - Client/Server information, simple caching
 - HTTP/1.1 – 1996
 - HTTP2.0 - 2015

<http://info.cern.ch/hypertext/WWW/TheProject.html>

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

Web and HTTP

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hyperlink Example</title>
    </head>
    <body>
        <p>Click the following link</p>
        <a href = "http://www.cnn.com" target ="_self">CNN</a>
    </body>
</html>
```

Uniform Resource Locator (URL)

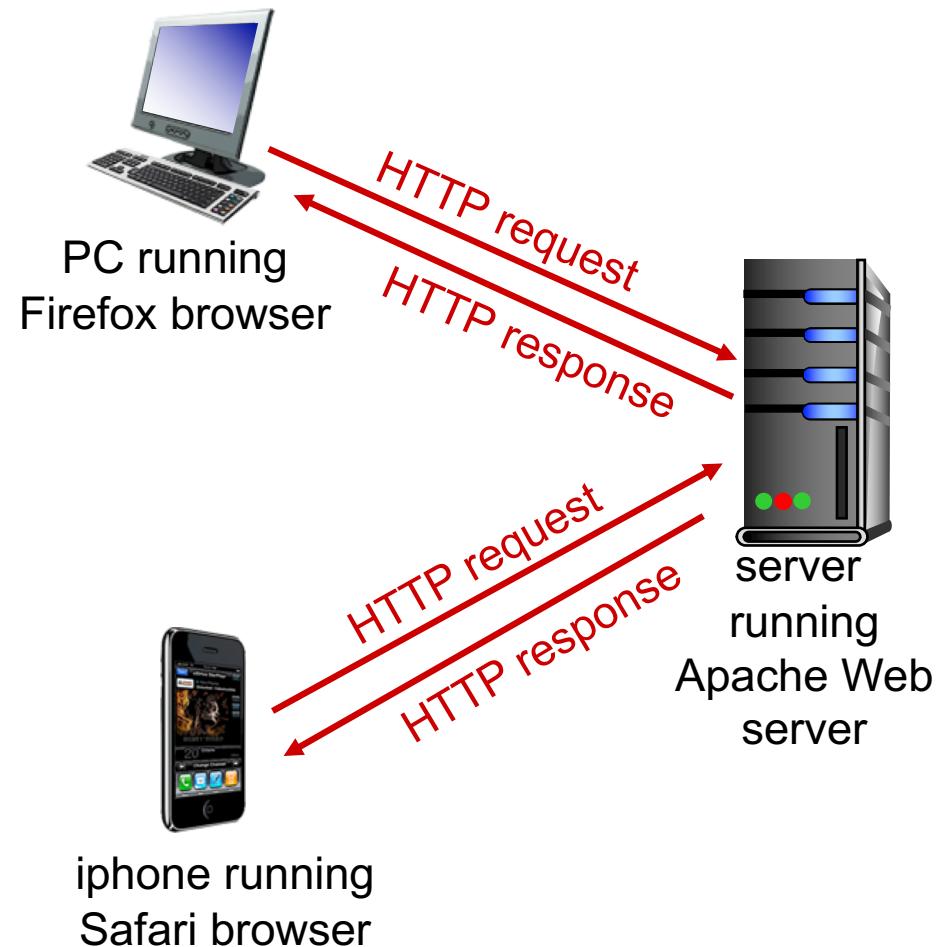
protocol://host-name[:port]/directory-path/resource

- ❖ *protocol*: http, ftp, https, smtp etc.
- ❖ *hostname*: DNS name, IP address
- ❖ *port*: defaults to protocol's standard port; e.g. http: 80 https: 443
- ❖ *directory path*: hierarchical, reflecting file system
- ❖ *resource*: Identifies the desired resource

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 (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

HTTP request message

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

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
```

carriage return character

line-feed character

HTTP response message

status line
(protocol
status code
status phrase)

header
lines

data, e.g.,
requested
HTML file

```
HTTP/1.1 200 OK\r\nDate: Sun, 26 Sep 2010 20:09:20 GMT\r\nServer: Apache/2.0.52 (CentOS) \r\nLast-Modified: Tue, 30 Oct 2007 17:00:02  
GMT\r\nETag: "17dc6-a5c-bf716880"\r\nAccept-Ranges: bytes\r\nContent-Length: 2652\r\nKeep-Alive: timeout=10, max=100\r\nConnection: Keep-Alive\r\nContent-Type: text/html; charset=ISO-8859-1\r\n\r\ndata 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

451 Unavailable for Legal Reasons

429 Too Many Requests

418 I'm a Teapot

HTTP is all text

- ❖ Makes the protocol simple
 - Easy to delineate messages (\r\n)
 - (relatively) human-readable
 - No issues 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
 - Headers may come in any order
 - Requires string parsing/processing

Request Method types (“verbs”)

HTTP/1.0:

- ❖ GET
 - Request page
- ❖ POST
 - Uploads user response to a form
- ❖ 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
- ❖ TRACE, OPTIONS, CONNECT, PATCH
 - For persistent connections

Uploading form input

POST method:

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

Get (in-URL) method:

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

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

User-server state: cookies

many Web sites use cookies

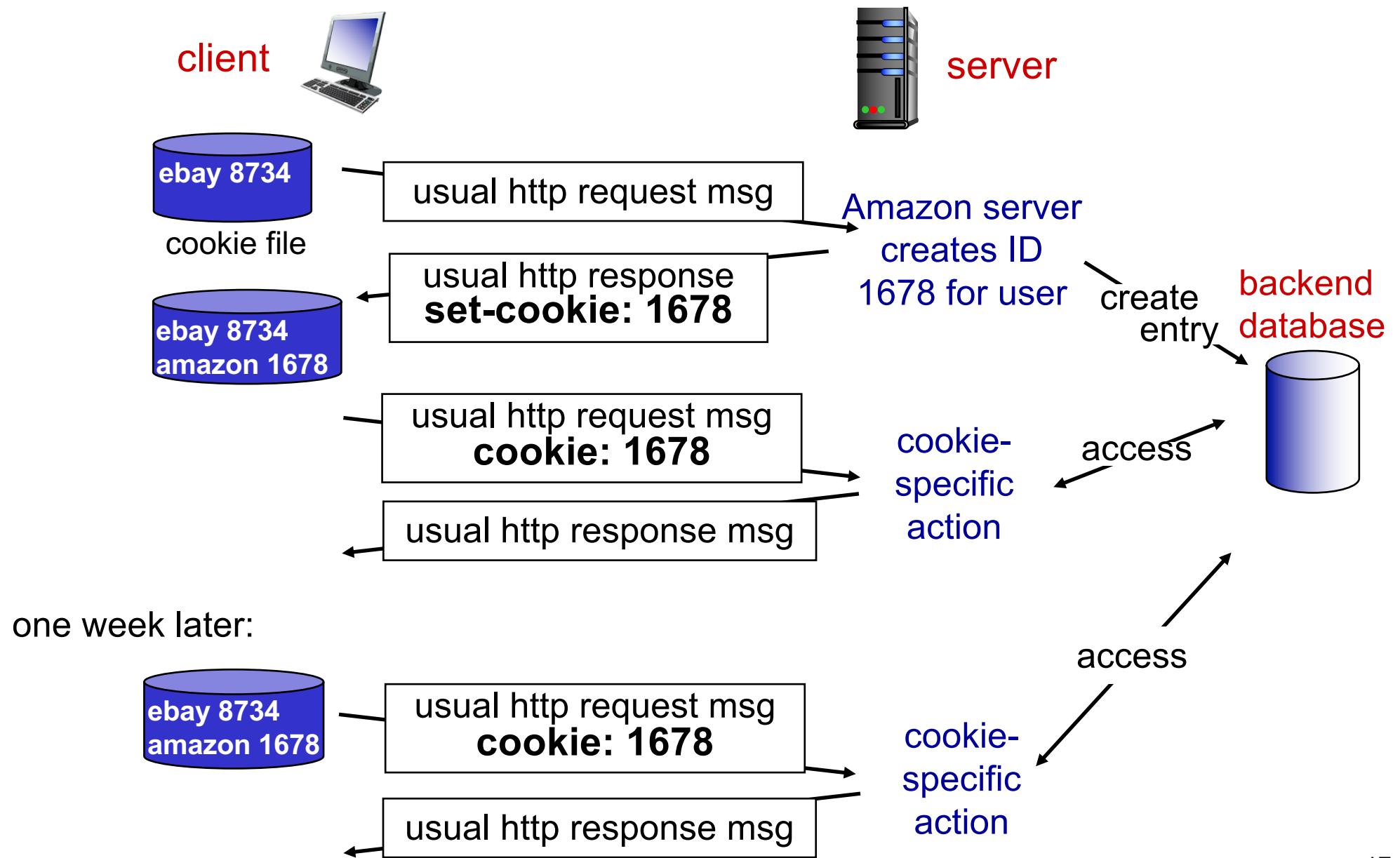
four components:

- 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

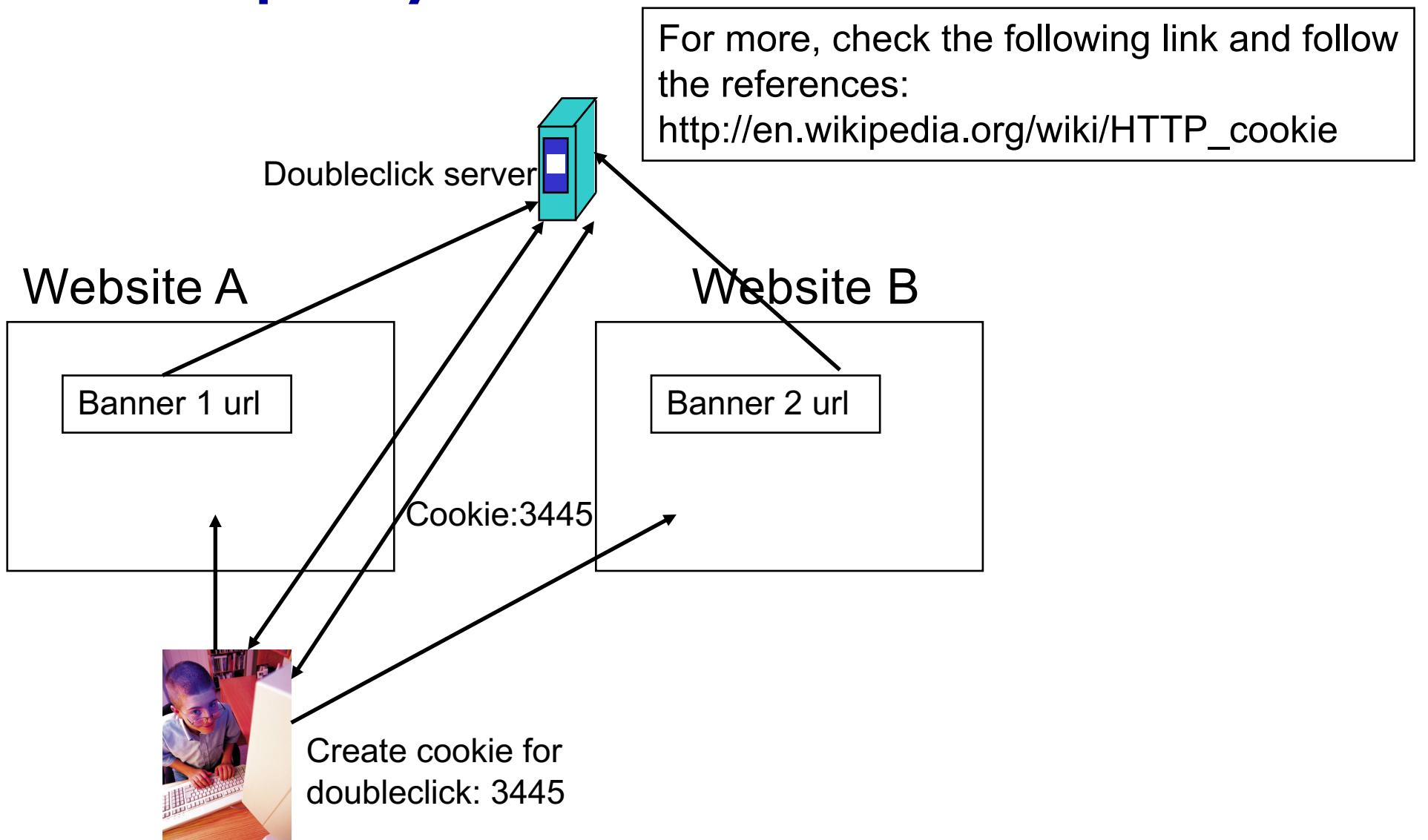
Cookies: keeping “state” (cont.)



The Dark Side of Cookies

- ❖ 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 ?
 - Check your browser's cookie file (cookies.txt, cookies.plist)
 - Do you see a website that you have never visited
- ❖ You COULD turn them off
 - But good luck doing anything on the Internet !!

Third party cookies



Performance of HTTP

- Page Load Time (PLT) as the metric
 - From click until user sees page
 - Key measure of web performance
- Depends on many factors such as
 - page content/structure,
 - protocols involved and
 - Network bandwidth and RTT

Performance Goals

- ❖ User
 - fast downloads
 - high availability
- ❖ Content provider
 - happy users (hence, above)
 - cost-effective infrastructure
- ❖ Network (secondary)
 - avoid overload

Solutions?

- ❖ User
 - fast downloads
 - high availability
- ❖ Content provider
 - happy users (hence, above)
 - cost-effective infrastructure
- ❖ Network (secondary)
 - avoid overload

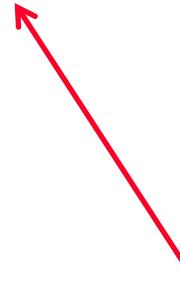
Improve HTTP to
achieve faster
downloads



Solutions?

- ❖ User
 - fast downloads
 - high availability
 - ❖ Content provider
 - happy users (hence, above)
 - cost-effective delivery infrastructure
 - ❖ Network (secondary)
 - avoid overload
-
- Improve HTTP to achieve faster downloads
- Caching and Replication
- Caching and Replication

Solutions?

- ❖ User
 - fast downloads
 - high availability
 - ❖ Content provider
 - happy users (hence, above)
 - cost-effective delivery infrastructure
 - ❖ Network (secondary)
 - avoid overload
- Improve HTTP to
achieve faster
downloads
- Caching and Replication
- Exploit economies of scale
(Webhosting, CDNs, datacenters)
- 

How to improve PLT

- Reduce content size for transfer
 - Smaller images, compression
- Change HTTP to make better use of available bandwidth
 - Persistent connections and pipelining
- Change HTTP to avoid repeated transfers of the same content
 - Caching and web-proxies
- Move content closer to the client
 - CDNs

HTTP Performance

- ❖ Most Web pages have multiple objects
 - e.g., HTML file and a bunch of embedded images
- ❖ How do you retrieve those objects (naively)?
 - *One item at a time*
- ❖ New TCP connection per (small) object!

non-persistent HTTP

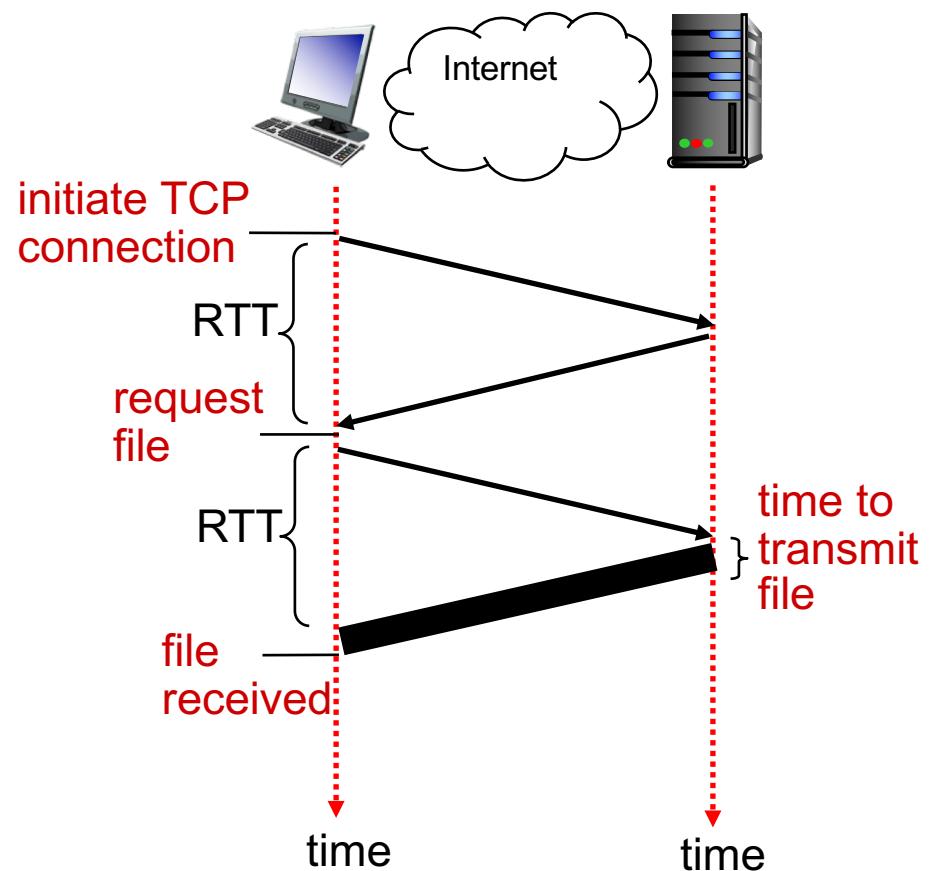
- ❖ at most one object sent over TCP connection
 - connection then closed
- ❖ downloading multiple objects required multiple connections

Non-persistent HTTP: response time

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

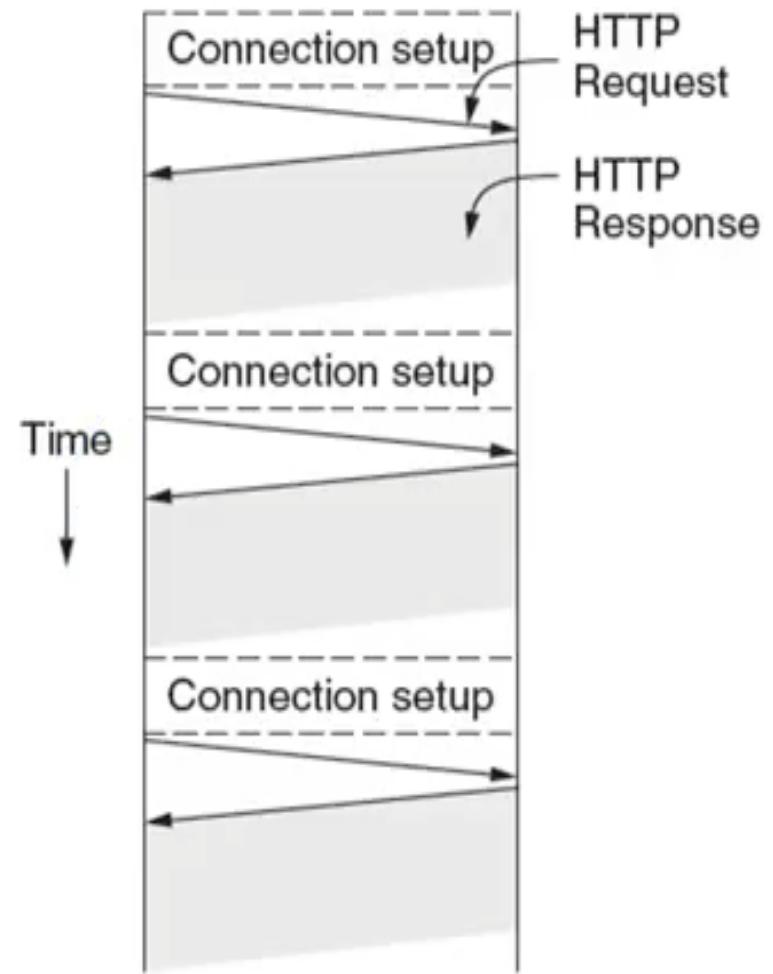
HTTP response time:

- ❖ 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 = $2\text{RTT} + \text{file transmission time}$



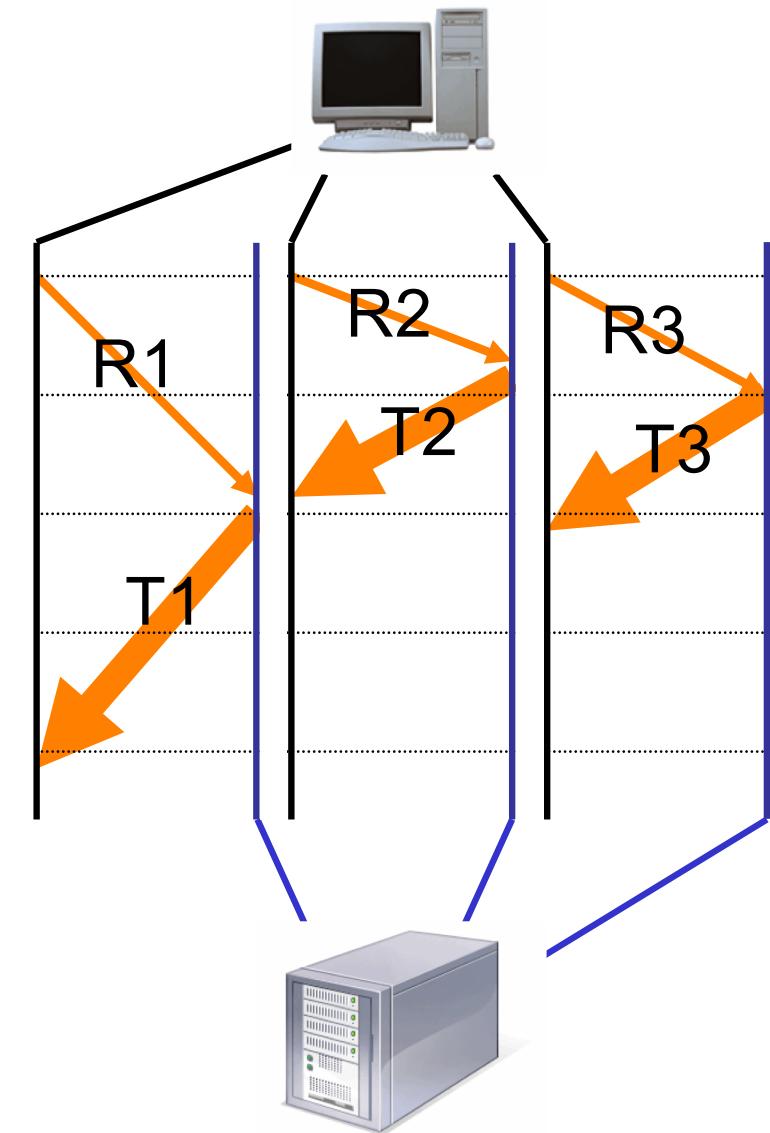
HTTP/1.0

- Non-Persistent: One TCP connection to fetch one web resource
- Fairly poor PLT
- 2 Scenarios
 - Multiple TCP connections setups to the **same server**
 - Sequential request/responses even when resources are located on **different servers**
- Multiple TCP slow-start phases (more in lecture on TCP)



Improving HTTP Performance: Concurrent Requests & Responses

- ❖ Use multiple connections *in parallel*
- ❖ Does not necessarily maintain order of responses



Quiz: Parallel HTTP Connections



- ❖ What are potential downsides of parallel HTTP connections, i.e. can opening too many parallel connections be harmful and if so in what way?

Persistent HTTP

Persistent HTTP

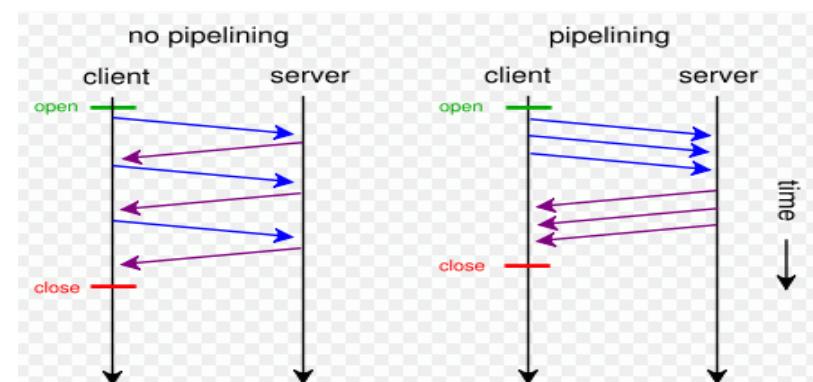
- ❖ server leaves TCP connection open after sending response
- ❖ subsequent HTTP messages between same client/server are sent over the same TCP connection
- ❖ Allow TCP to learn more accurate RTT estimate (APPARENT LATER IN THE COURSE)
- ❖ Allow TCP congestion window to increase (APPARENT LATER)
- ❖ i.e., leverage previously discovered bandwidth (APPARENT LATER)

Persistent without pipelining:

- ❖ client issues new request only when previous response has been received
- ❖ one RTT for each referenced object

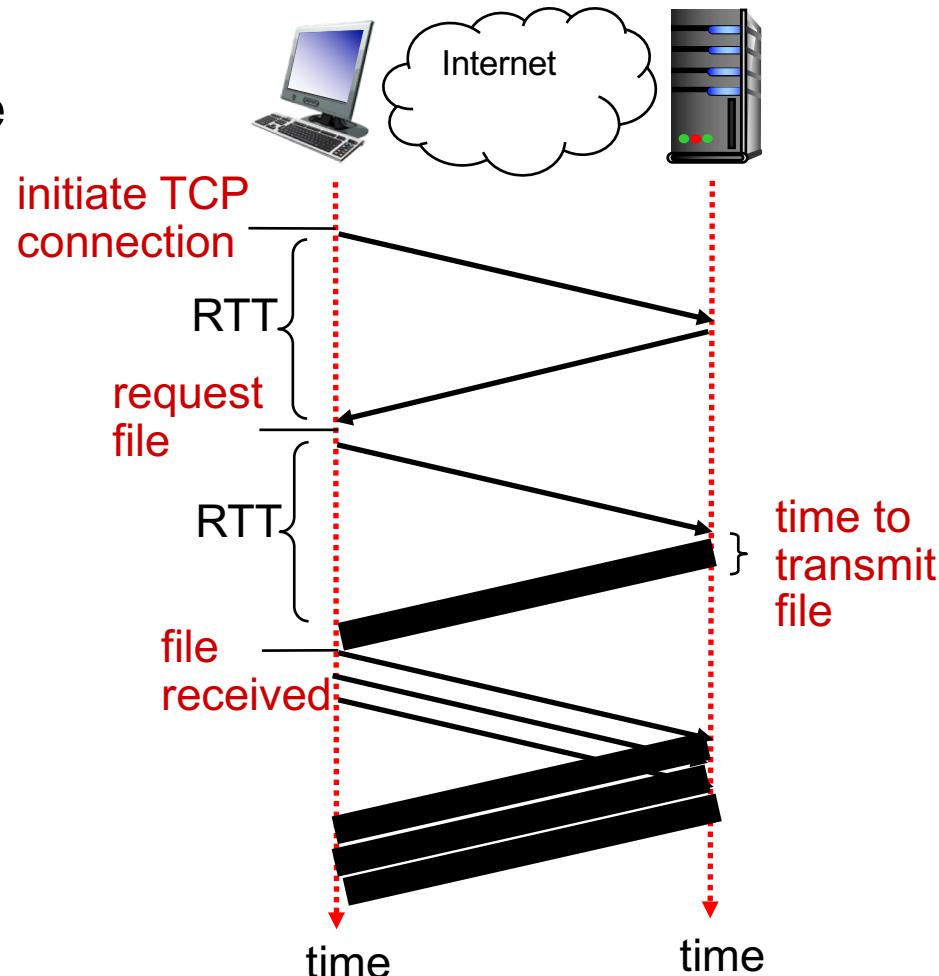
Persistent with pipelining:

- ❖ introduced in HTTP/1.1
- ❖ client sends requests as soon as it encounters a referenced object
- ❖ as little as one RTT for all the referenced objects



HTTP 1.1: response time with pipelining

Website with one index page and three embedded objects



How to improve PLT

- Reduce content size for transfer
 - Smaller images, compression
- Change HTTP to make better use of available bandwidth
 - Persistent connections and pipelining
- Change HTTP to avoid repeated transfers of the same content
 - Caching and web-proxies
- Move content closer to the client
 - CDNs

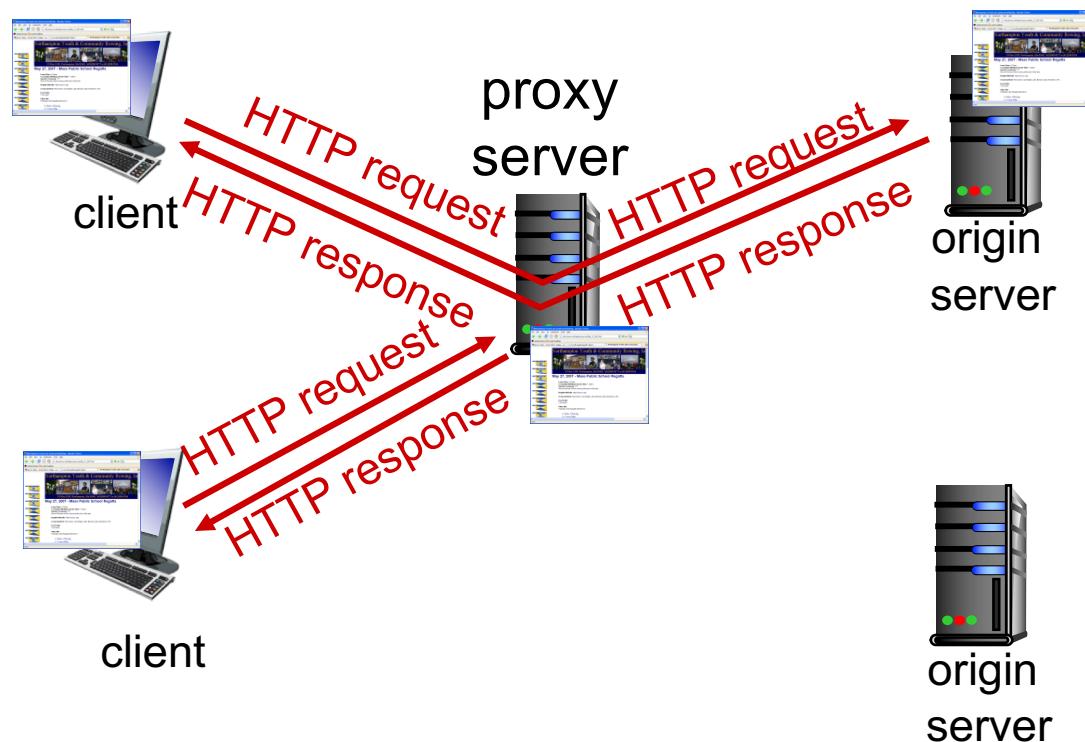
Improving HTTP Performance: Caching

- ❖ Why does caching work?
 - Exploits *locality of reference*
- ❖ How well does caching work?
 - Very well, up to a limit
 - Large overlap in content
 - But many unique requests

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

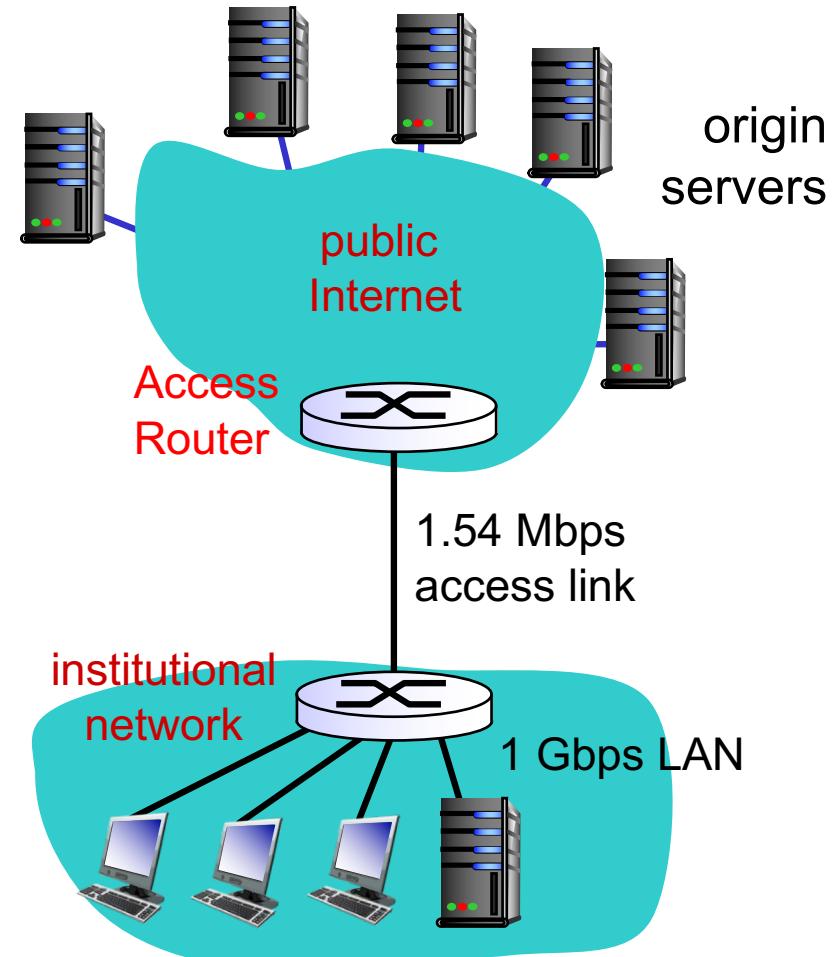
Caching example:

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 access router to any origin server: 2 sec
- ❖ access link rate: 1.54 Mbps

consequences:

- ❖ LAN utilization: 0.15%
- ❖ access link utilization = **99%**
- ❖ total delay = Internet delay +
access delay + LAN delay
= 2 sec + minutes + usecs



problem!

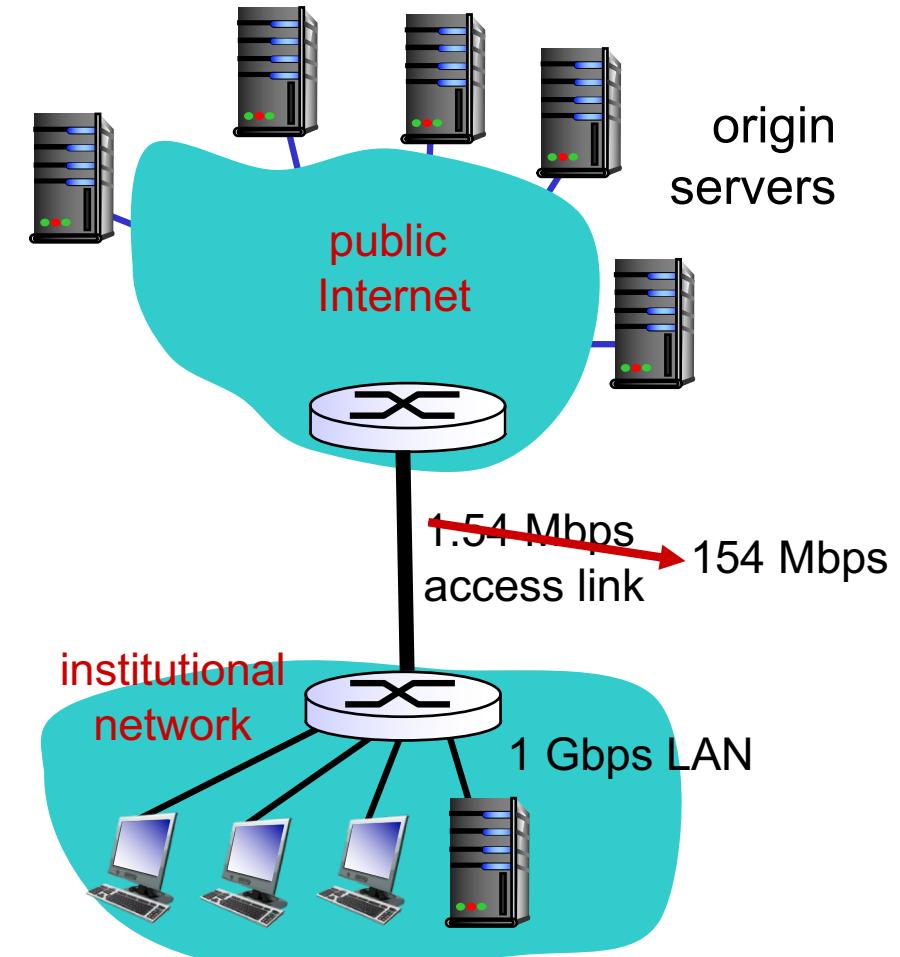
Caching example: fatter access link

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 access router to any origin server: 2 sec
- ❖ access link rate: 1.54 Mbps

consequences:

- ❖ LAN utilization: 0.15%
- ❖ access link utilization = ~~99%~~ 0.99%
- ❖ total delay = Internet delay + access delay + LAN delay
= 2 sec + ~~minutes + usecs~~ msec



Cost: increased access link speed (not cheap!)

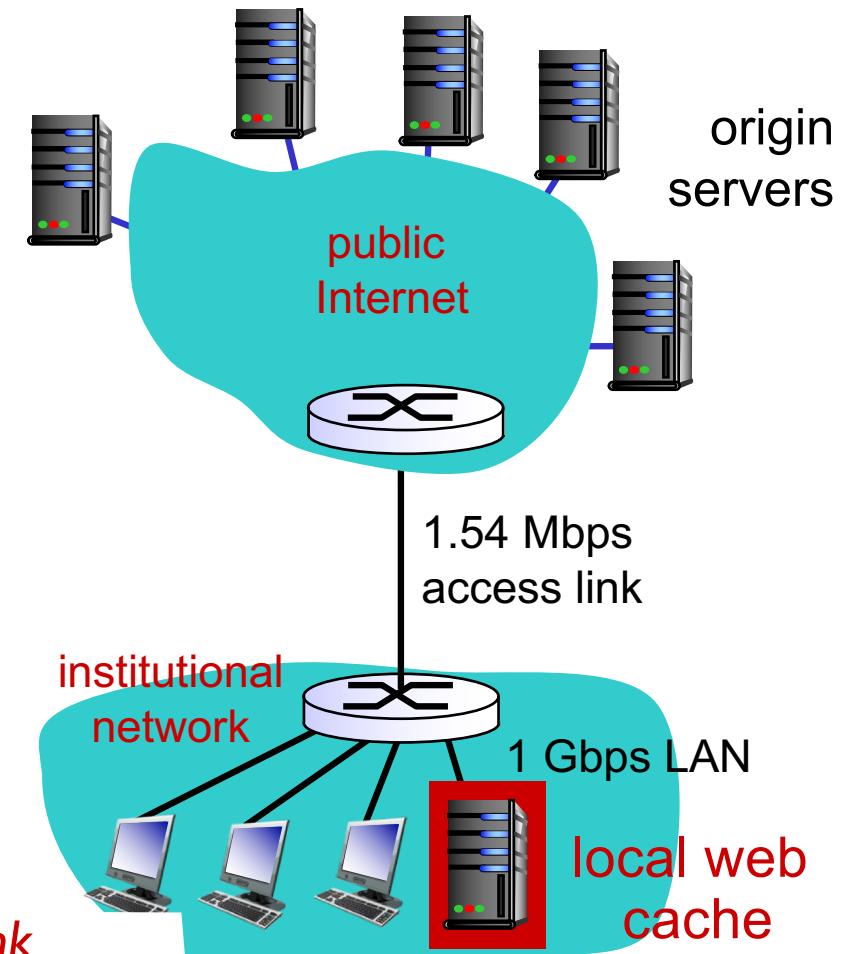
Caching example: install local cache

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 access router to any origin server: 2 sec
- ❖ access link rate: 1.54 Mbps

consequences:

- ❖ LAN utilization: ?
- ❖ access link utilization = ?
- ❖ total delay = ? *How to compute link utilization, delay?*

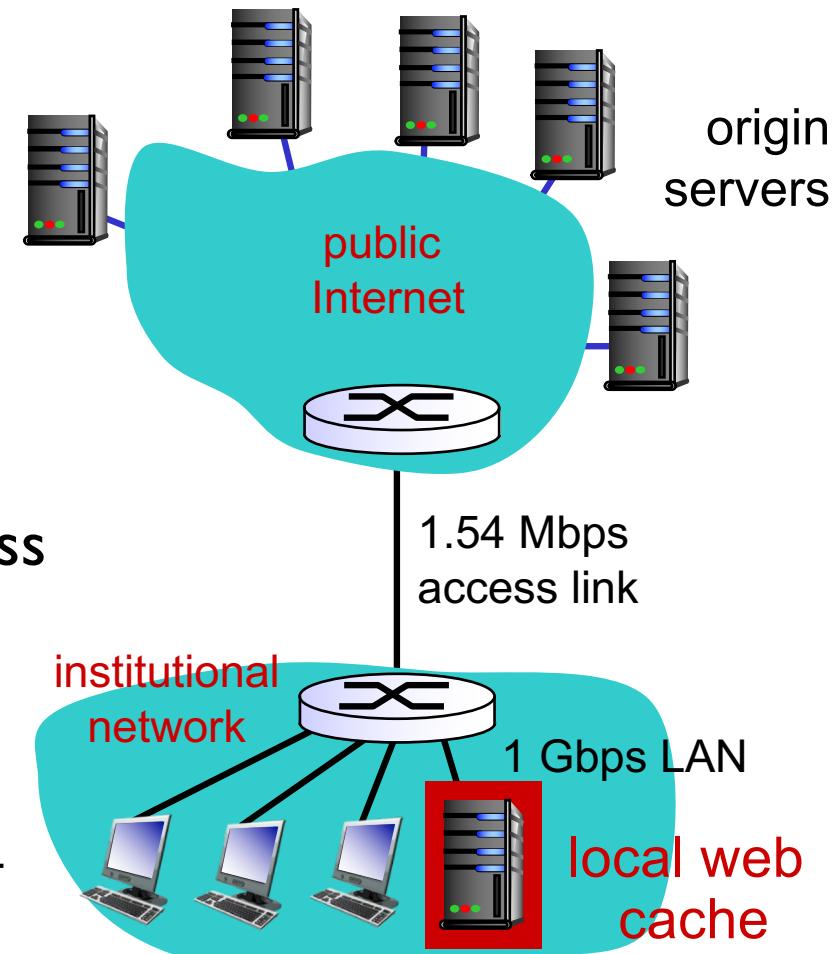


Cost: web cache (cheap!)

Caching example: install local cache

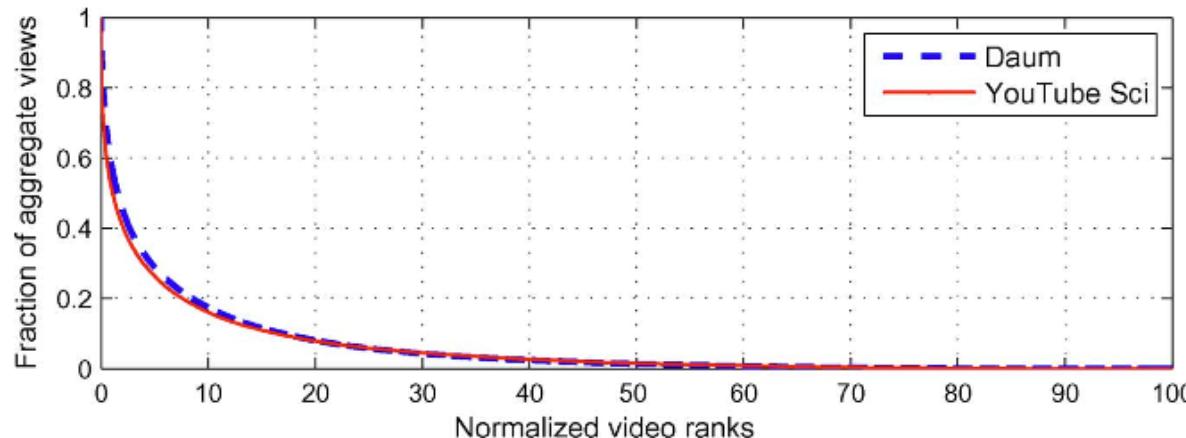
Calculating access link utilization, delay with cache:

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



But what is the likelihood of cache hits?

- ❖ Distribution of web object requests generally follows a Zipf-like distribution
- ❖ *The probability that a document will be referenced k requests after it was last referenced is roughly proportional to 1/k*. That is, web traces exhibit excellent **temporal locality**.



Video content exhibits similar properties: 10% of the top popular videos account for nearly 80% of views, while the remaining 90% of videos account for total 20% of requests.

Paper – <http://yongyeol.com/papers/cha-video-2009.pdf>

Paper – “Web Caching and Zipf-like Distributions: Evidence and Implications”
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.34.8742&rep=rep1&type=pdf>

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

client



server



HTTP request msg
If-modified-since: <date>

HTTP response
HTTP/1.0
304 Not Modified

object
not
modified
before
<date>

HTTP request msg
If-modified-since: <date>

HTTP response
HTTP/1.0 200 OK
<data>

object
modified
after
<date>

Example Cache Check Request

GET / HTTP/1.1

Accept: */*

Accept-Language: en-us

Accept-Encoding: gzip, deflate

If-Modified-Since: Mon, 29 Jan 2001 17:54:18 GMT

If-None-Match: "7a11f-10ed-3a75ae4a"

User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT
5.0)

Host: www.intel-iris.net

Connection: Keep-Alive

Example Cache Check Response

HTTP/1.1 304 Not Modified

Date: Tue, 27 Mar 2001 03:50:51 GMT

Server: Apache/1.3.14 (Unix) (Red-Hat/Linux) mod_ssl/2.7.1
OpenSSL/0.9.5a DAV/1.0.2 PHP/4.0.1pl2 mod_perl/1.24

Connection: Keep-Alive

Keep-Alive: timeout=15, max=100

ETag: "7a11f-10ed-3a75ae4a"

Improving HTTP Performance: Replication

- ❖ Replicate popular Web site across many machines
 - Spreads load on servers
 - Places content closer to clients
 - Helps when content isn't cacheable
- ❖ Problem:
 - Want to direct client to particular replica
 - Balance load across server replicas
 - Pair clients with nearby servers
 - Expensive
- ❖ Common solution:
 - DNS returns different addresses based on client's geo location, server load, etc.

Improving HTTP Performance: CDN

- ❖ Caching and replication as a service
- ❖ Integrate forward and reverse caching functionality
- ❖ Large-scale distributed storage infrastructure (usually) administered by one entity
 - e.g., Akamai has servers in 20,000+ locations
- ❖ Combination of (pull) caching and (push) replication
 - **Pull:** Direct result of clients' requests
 - **Push:** Expectation of high access rate
- ❖ Also do some processing
 - Handle *dynamic* web pages
 - *Transcoding*
 - *Maybe do some security function – watermark IP*

What about HTTPS?

- ❖ HTTP is insecure
- ❖ HTTP basic authentication: password sent using base64 encoding (can be readily converted to plaintext)
- ❖ HTTPS: HTTP over a connection encrypted by Transport Layer Security (TLS)
- ❖ Provides:
 - Authentication
 - Bidirectional encryption
- ❖ Widely used in place of plain vanilla HTTP



What's on the horizon: HTTP/2

- ❖ Google SPDY (speedy) -> HTTP/2: (RFC 7540 May 2015)
- ❖ Better content structure
- ❖ Improvements
 - Servers can **push** content and thus reduce overhead of an additional request cycle
 - Fully multiplexed
 - Requests and responses are sliced in smaller chunks called frames, frames are tagged with an ID that connects data to the request/response
 - overcomes Head-of-line blocking in HTTP 1.1
 - Prioritisation of the order in which objects should be sent (e.g. CSS files may be given higher priority)
 - Data compression of HTTP headers
 - Some headers such as cookies can be very long
 - Repetitive information

More details: <https://http2.github.io/faq/>
Demo: <https://http2.akamai.com/demo>

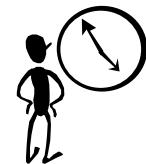
Summary

- ❖ Application Layer (Chapter 2)

- Principles of Network Applications
- HTTP

- ❖ Next:

- E-mail
- DNS



Reading Exercise
Chapter 2: 2.3 – 2.4