Remember: mic!

Web & HTTP

SCC. 203 – Computer Networks

Geoff Coulson

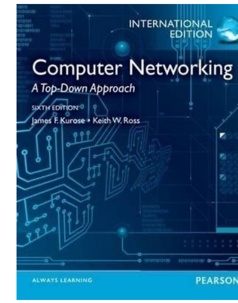
Week 13 Lecture 2

Contents

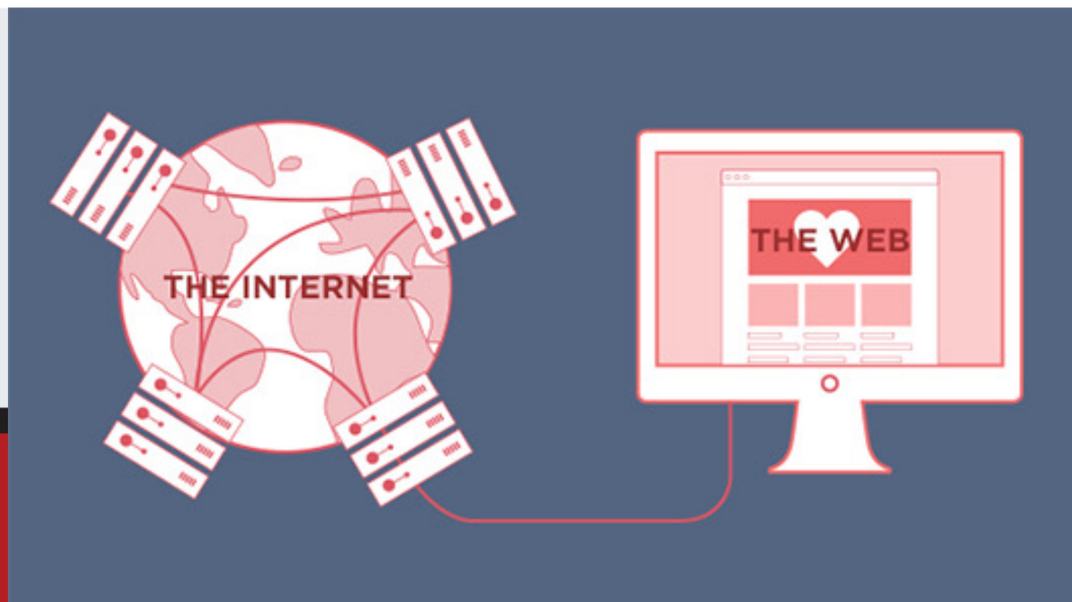
- HTTP Connection Basics
- HTTP Protocol
- Cookies, keeping state + tracking
- Web caching (proxy)

Chapter 2.2

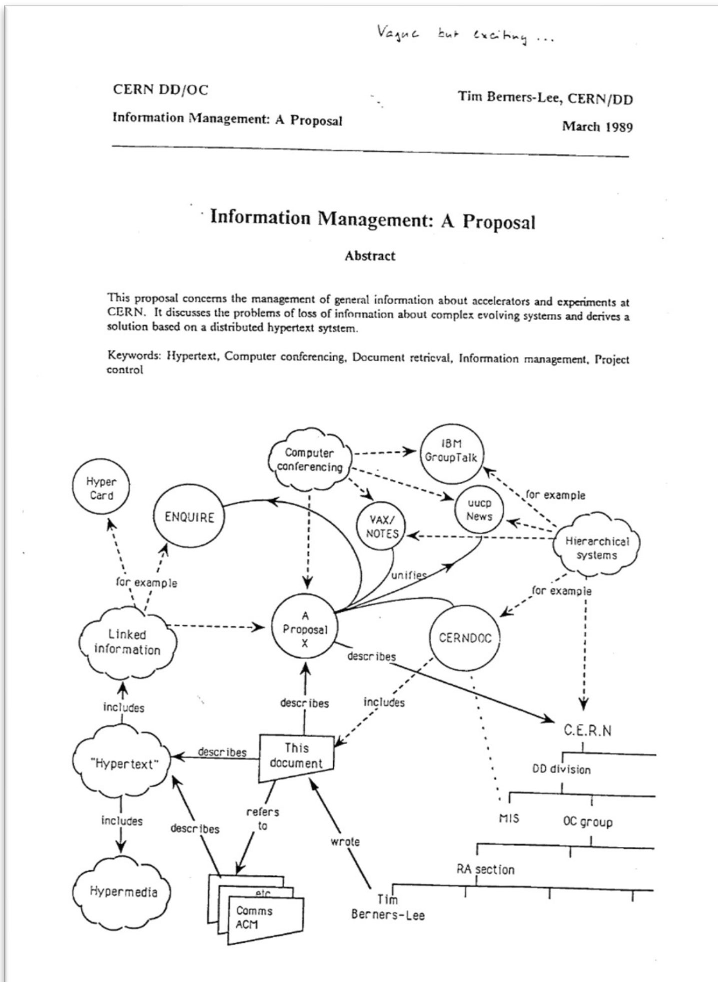
Web & HTTP



The Web is not the Internet!



Tim Berners-Lee to devised the World Wide Web (WWW) in 1989



Vague but exciting ...

CERN DD/OC

Information Management: A Proposal

Tim Berners-Lee, CERN/DD

March 1989

Information Management: A Proposal

Abstract

This proposal concerns the management of general information about accelerators and experiments at CERN. It discusses the problems of loss of information about complex evolving systems and derives a solution based on a distributed hypertext sytstem.

Keywords: Hypertext, Computer conferencing, Document retrieval, Information management, Project control

COMPUTER 'WEB' TO CHANGE BILLIONS OF LIVES (YEAH, RIGHT)

A BRITISH computer geek's brainwave could be one of the greatest inventions ever, it was claimed last night.

Tim Berners-Lee, 35, has enabled computer users to see documents and pictures made available by others in "cyberspace".

He uses the "Internet" system, which so far only

By DOT COMME

links academics but could eventually include anyone.

Berners-Lee, who works at a nuclear research base near Geneva, calls his idea the "World Wide Web".

One scientist said: "This could be huge. The idea of strangers worldwide sharing ideas instantly is mind-boggling." But another sneered: "They said Sinclair's C5 would change the world. Now you'd struggle to give one away."

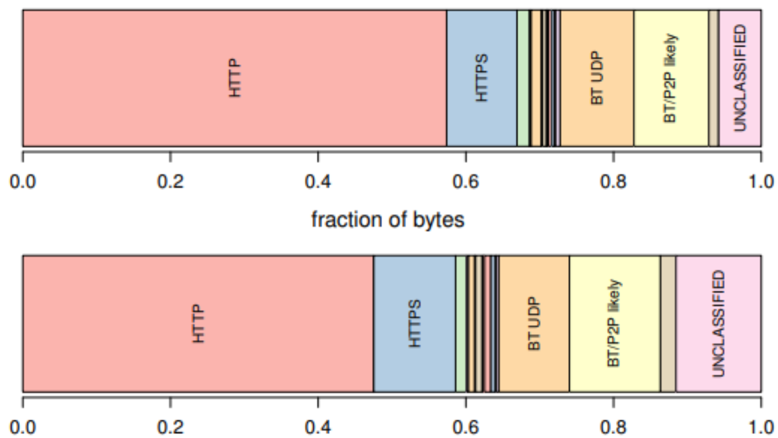
Riddle of 'E' mail — Page 8

Web feat . . . Berners-Lee

Source: The Sun, 1991

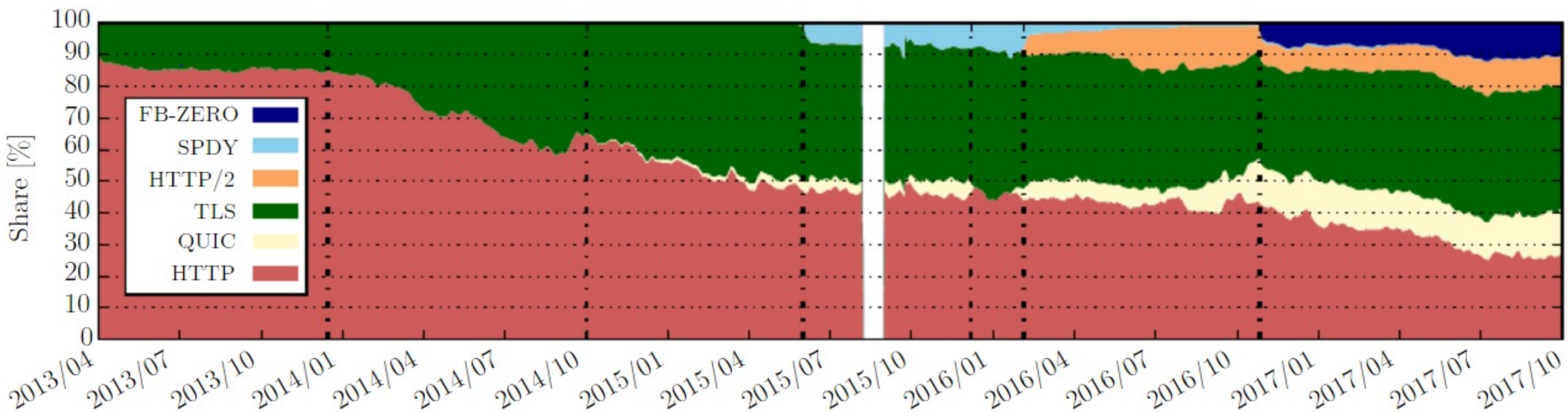
https://www.w3.org/History.html

By 2019, HTTP and HTTPS accounted for 70% of total internet traffic!



Protocol	% bytes	% pkts
HTTP	57.39	47.52
HTTPS	9.53	11.08
RTMP	1.72	1.48
NNTP	1.41	0.87
NNTPS	0.63	0.38
SMTP	0.53	0.91
DNS	0.45	0.87
SSH	0.42	0.61
other known	0.68	0.74
BT UDP	10.00	9.57
BT/P2P likely	10.14	12.31
P2P likely	1.32	2.10
unclassified	5.78	11.56

https://people.csail.mit.edu/richterp/richter_pam15.pdf



https://blog.apnic.net/2019/01/24/five-years-at-the-edge-recording-the-evolution-of-web-usage-from-an-isp/

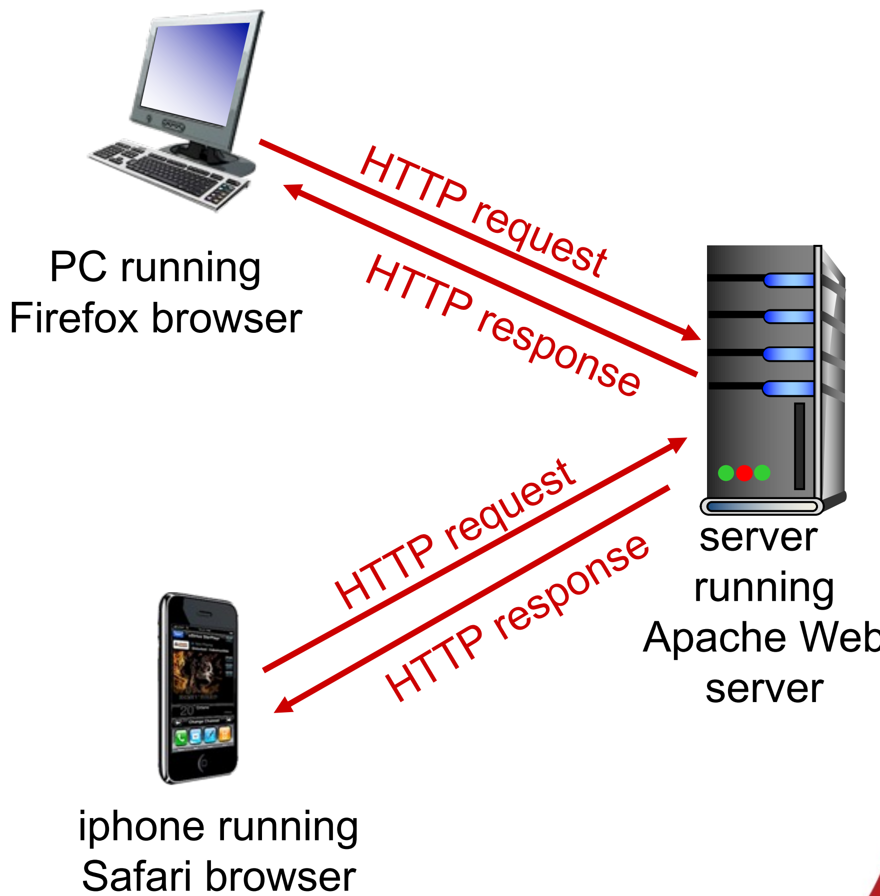
Web and HTTP

- A web page consists of a *base HTML-file* which may include several *referenced objects*
- An object can be HTML file, a JPEG image, a Java applet, an audio file,…
- 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
HyperText Transfer Protocol

- The 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)

Underlying protocol: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

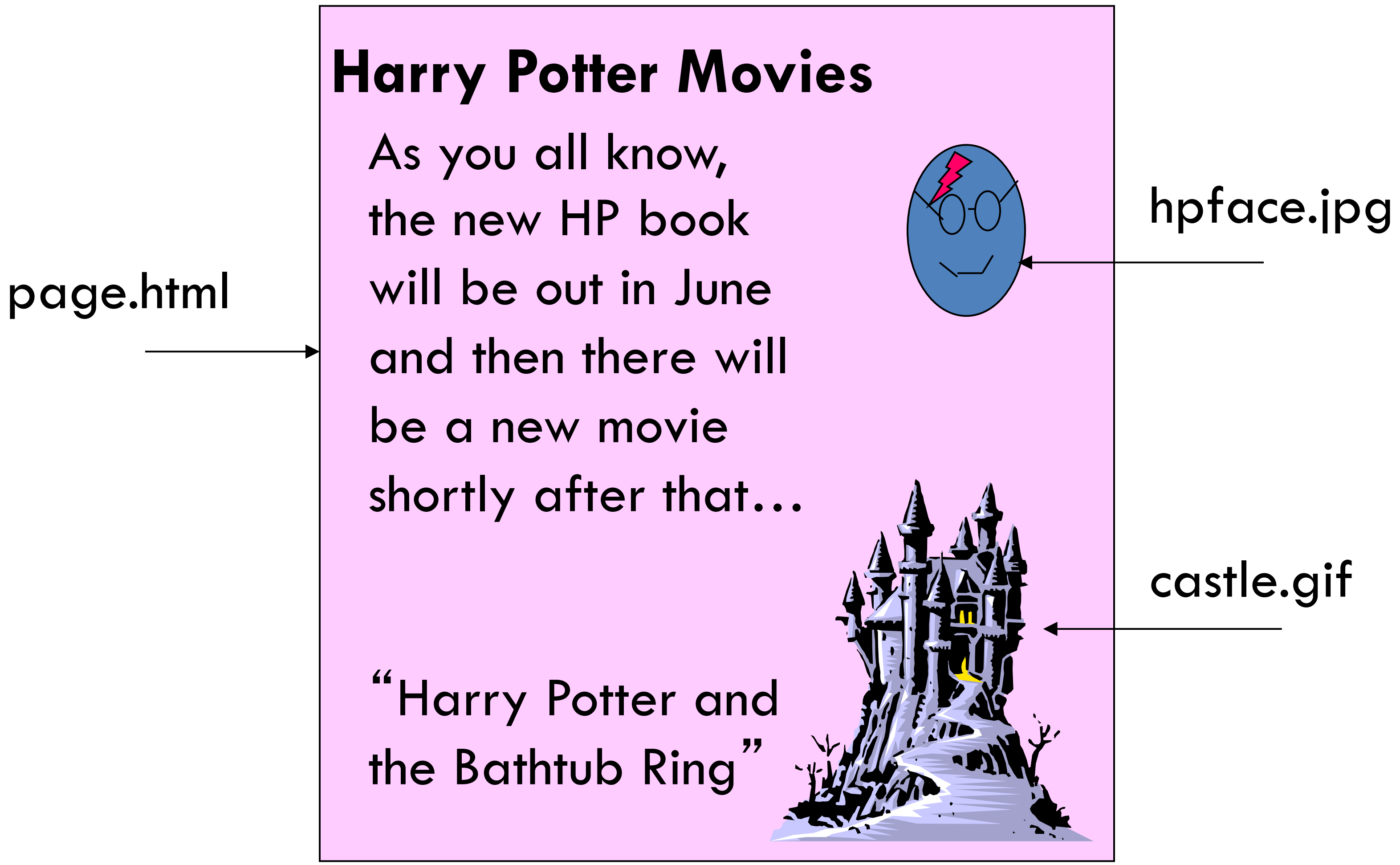
non-persistent HTTP

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

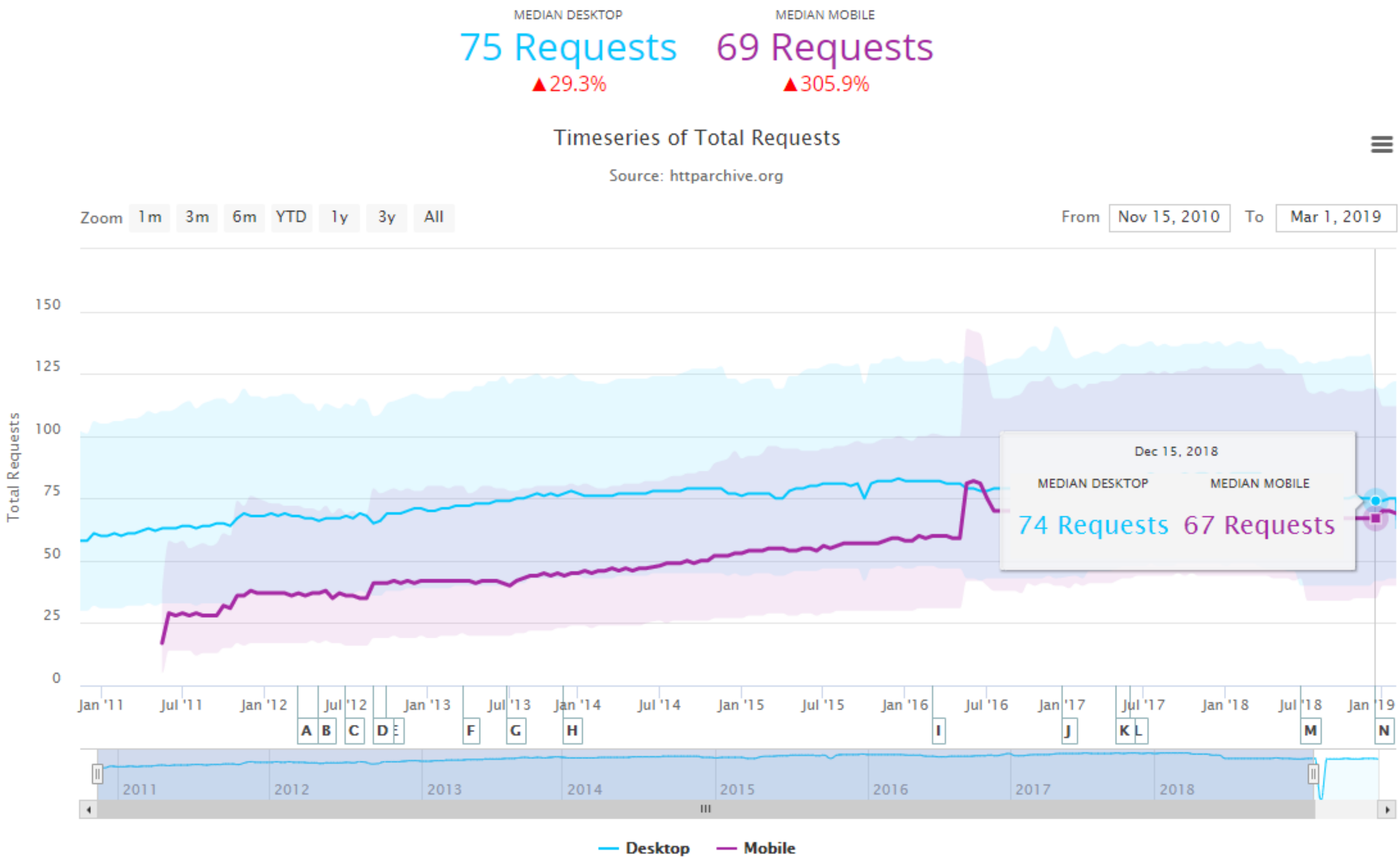
persistent HTTP

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

A web page can consist of many different files

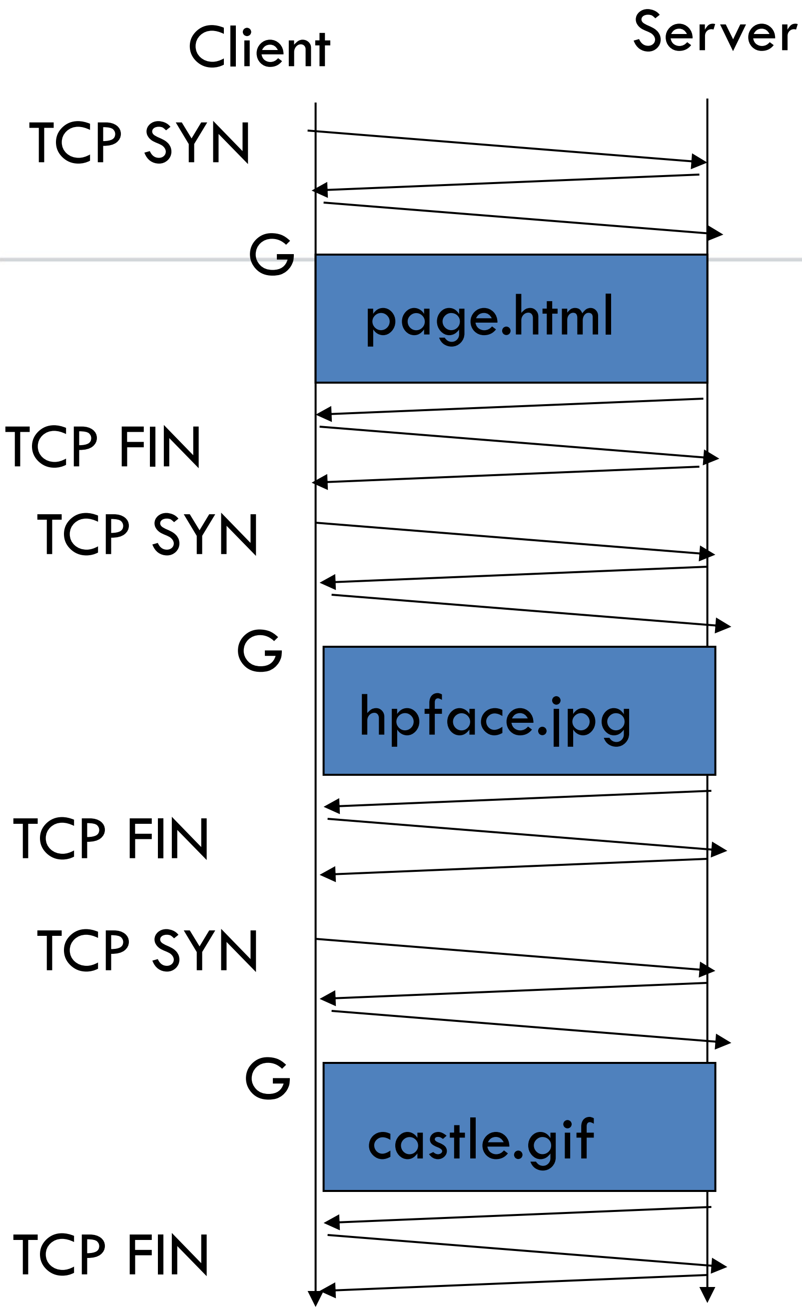


Median number of objects embedded in a webpage



https://httparchive.org/reports/page-weight#reqTotal

Non-Persistent HTTP



The "classic" approach in HTTP/1.0 is to use one HTTP request per TCP connection, serially.

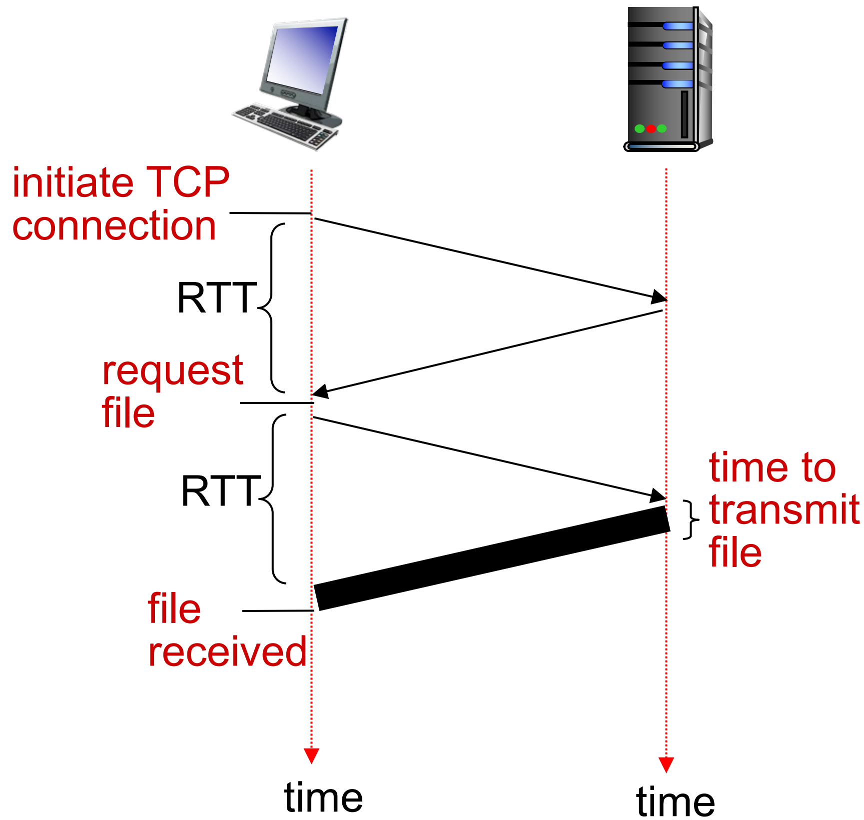
Non-persistent HTTP: response time

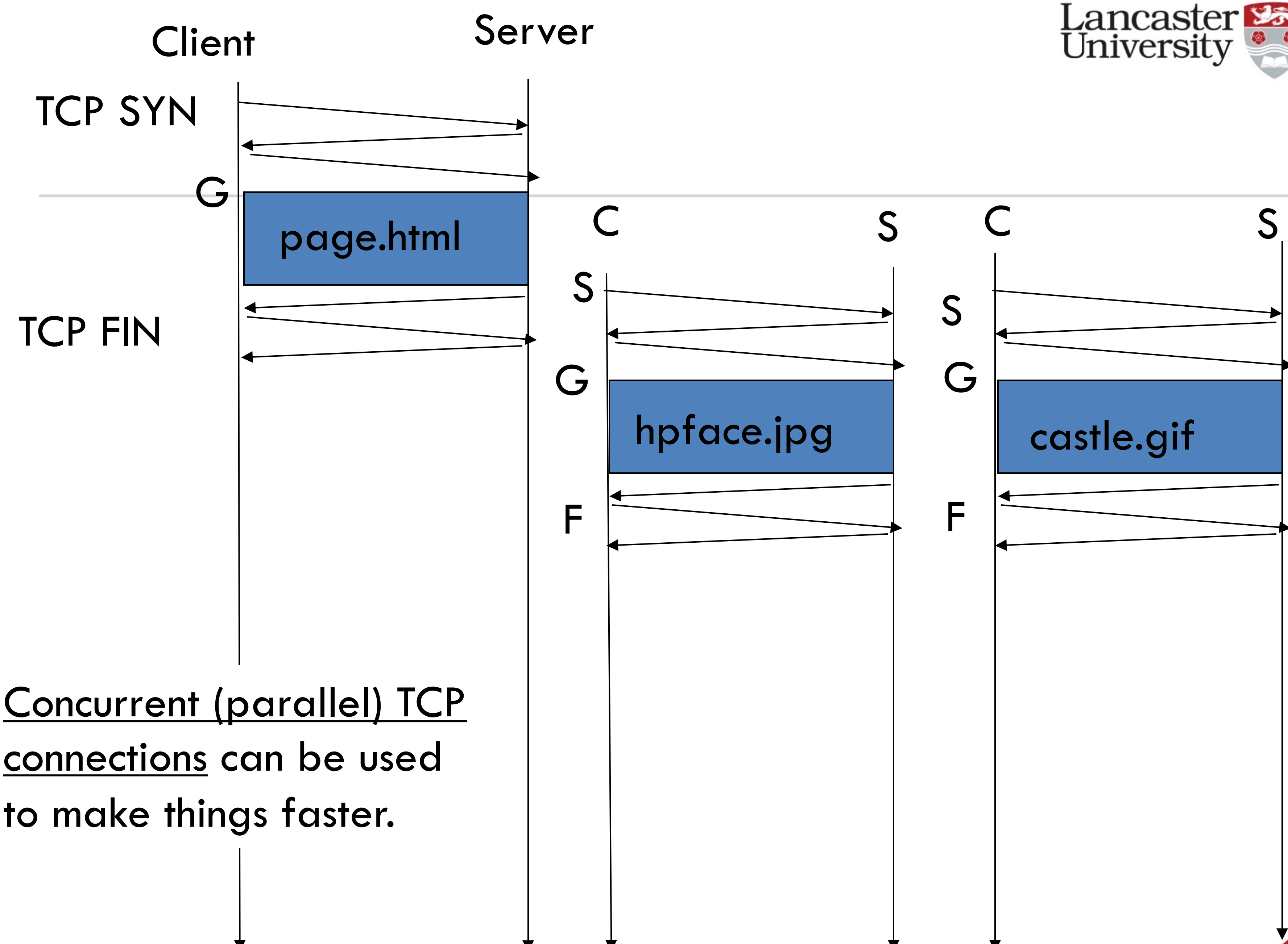
RTT: time for a packet to travel from client to server and back

HTTP response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
  - This assumes HTTP GET piggy backed on the ACK
- file transmission time
- non-persistent HTTP response time =

  2RTT+ file transmission time





Concurrent (parallel) TCP connections can be used to make things faster.

Persistent HTTP

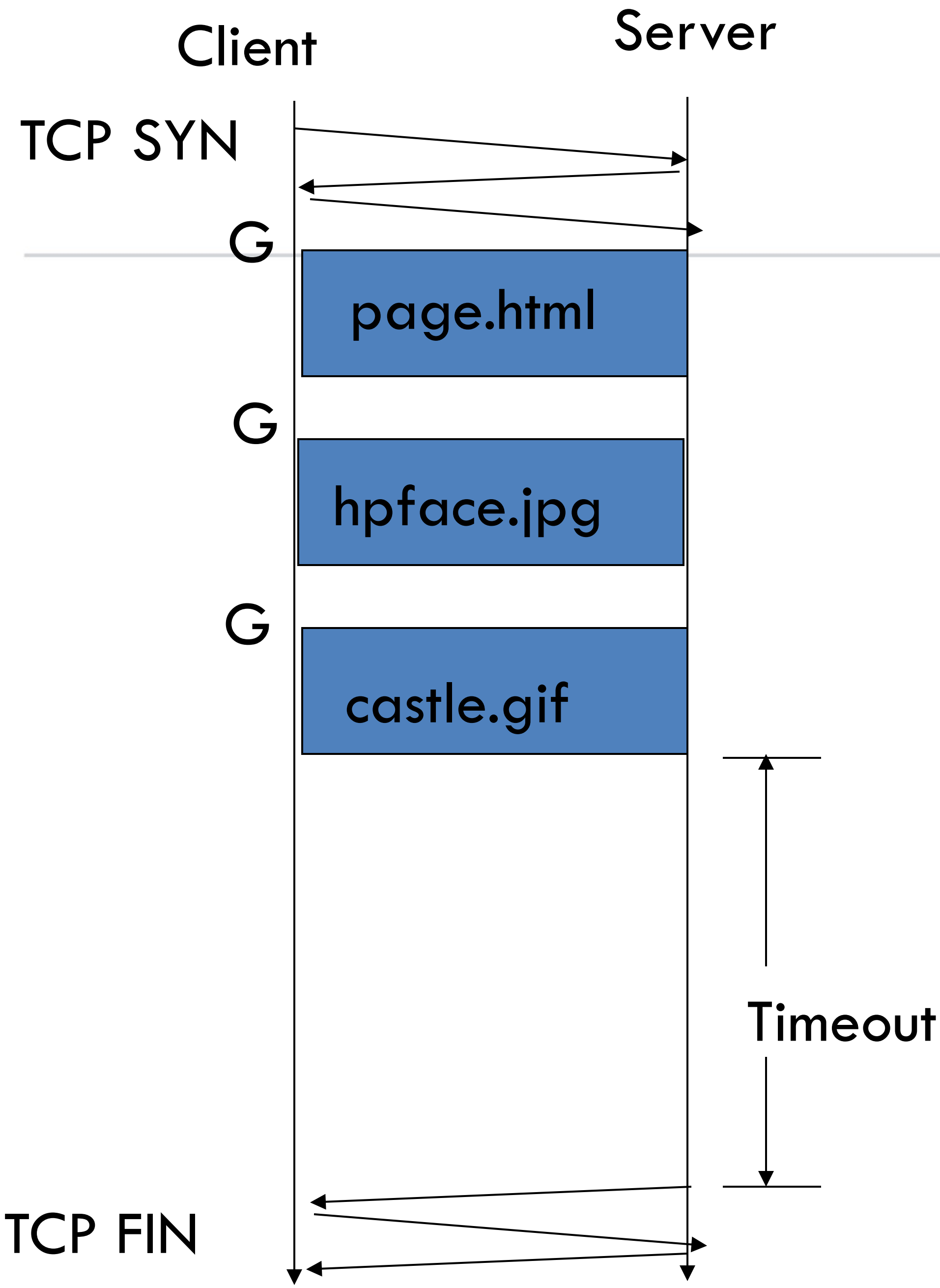
non-persistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

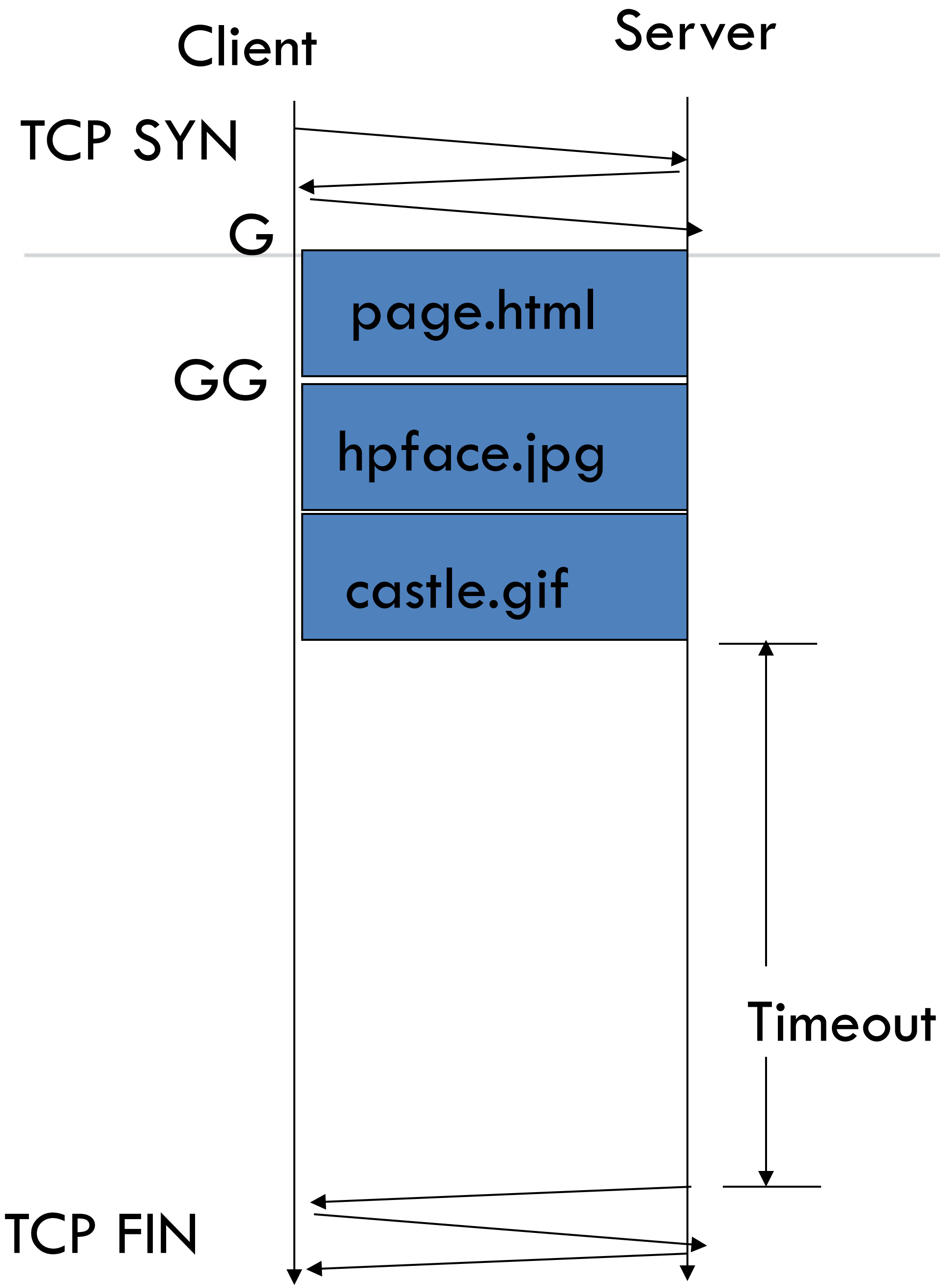
persistent HTTP:

- **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**

Persistent HTTP



The "persistent HTTP" approach can re-use the same TCP connection for Multiple HTTP transfers, one after another, serially. Amortizes TCP overhead, but maintains TCP state longer at server.



The "<u>pipelining</u>" feature in HTTP/1.1 allows requests to be issued asynchronously on a persistent connection. Requests must be processed in proper order. Can do clever packaging.

HTTP request message

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

carriage return character

line-feed character

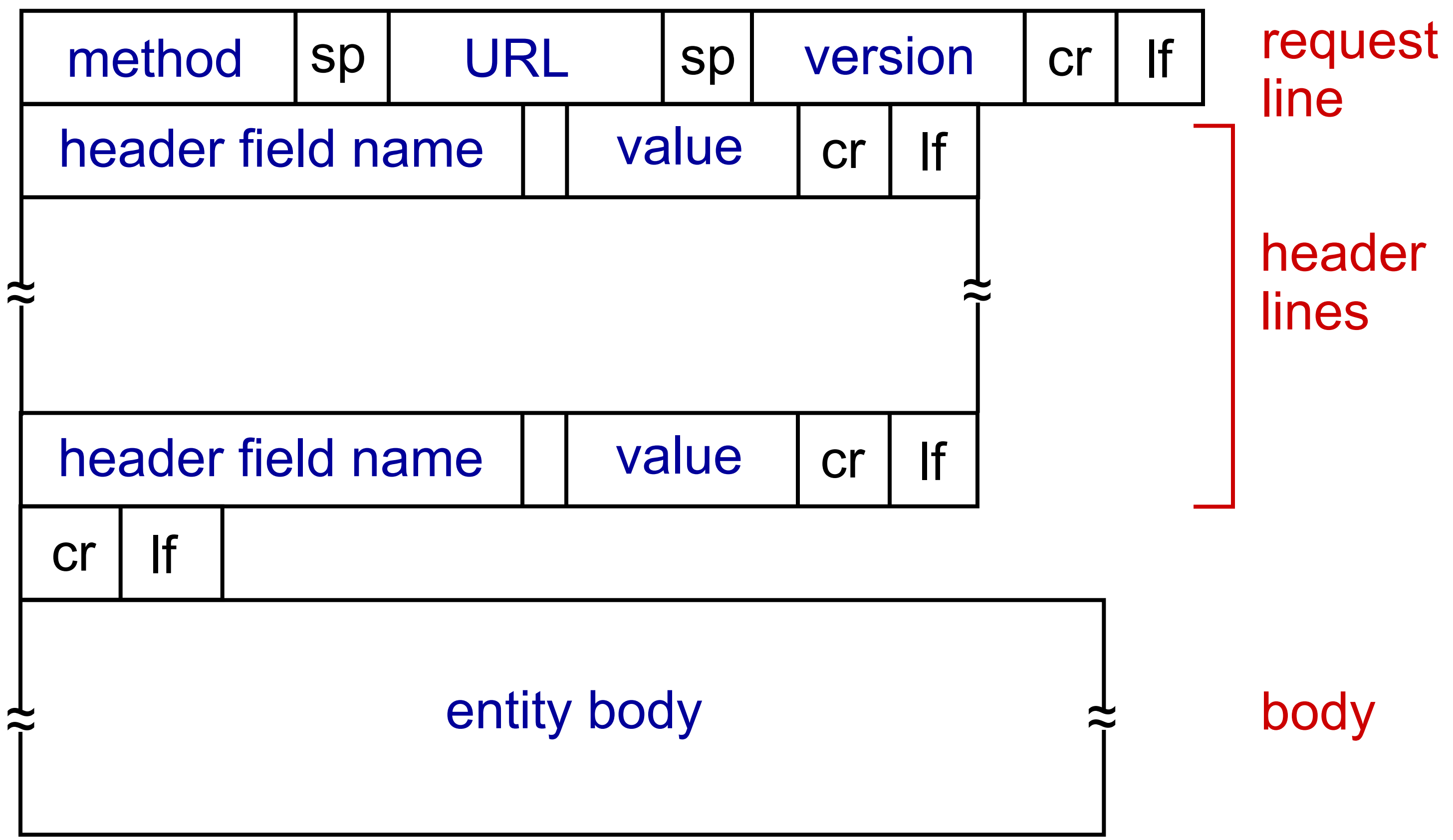
request line (GET, POST, HEAD commands)

header lines

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

```
GET somedir/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

Lancaster University

POST method:

- web pages often include form input
- input is uploaded to server in entity body

URL method:

- uses PUT 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
    - Often used for debugging

- 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

Lancaster University

status line

header lines

```
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 ...
```

the HTML file that was requested

* 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 responses
- Some example 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:)

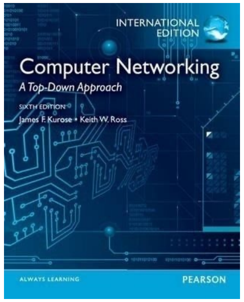
  `400 Bad Request`
  - request msg not understood by server

  `404 Not Found`
  - requested document not found on this server

  `505 HTTP Version Not Supported`

Chapter 2.2.4

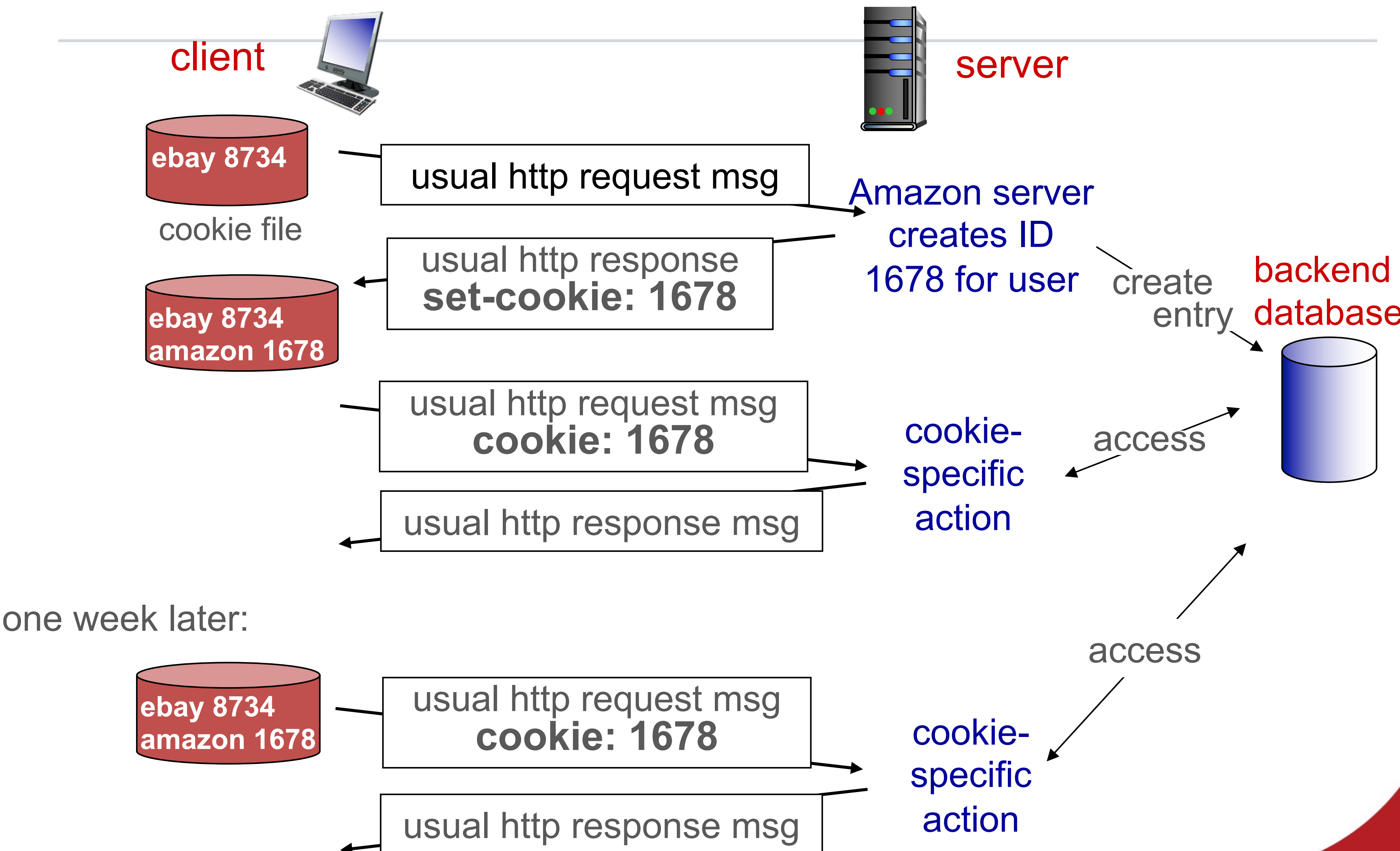
Cookies: User-Server Interactions



User-server state: cookies

- **Cookie** == a small file with data (up to 4KB)
  - Sent to the Web browser by the Web server
  - Saved locally inside the browser
  - Sent back by the browser in all subsequent requests
- 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)
- **How to keep “state”**
  - Protocol endpoints: maintain state at sender/receiver over multiple transactions
  - Cookies: http messages carry state

- **Cookies and privacy:**
  - **Cookies permit sites to learn a lot about you**
  - **You may supply name and e-mail to sites**

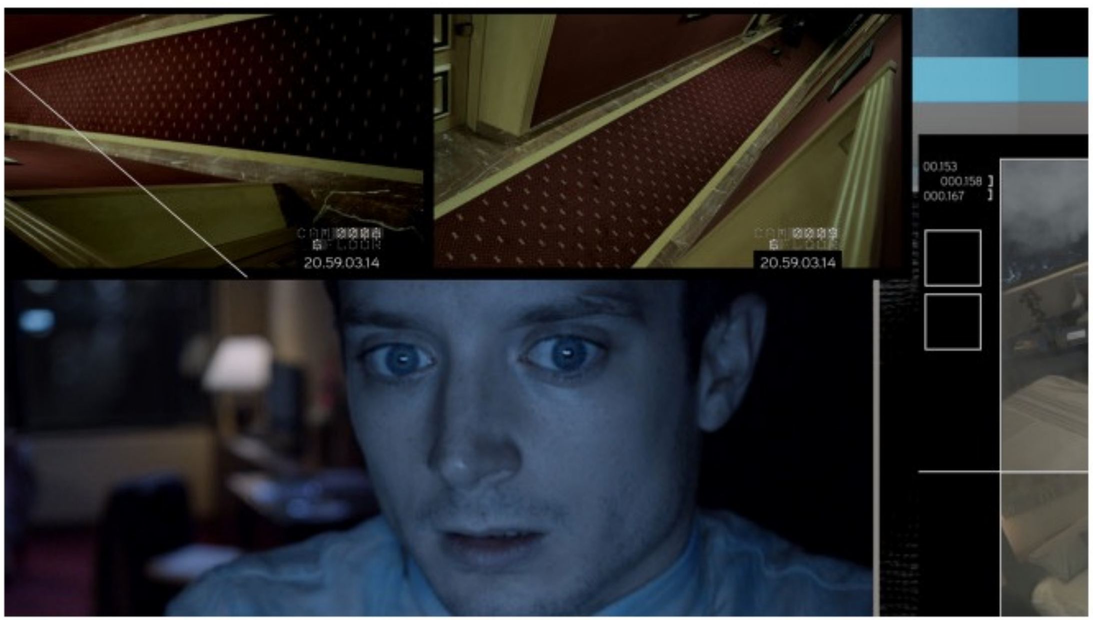
Cookies + Third Parties

Lancaster University

Elijah Wood's New Movie Is a Prophetic Thriller About Celebrity Hacking

BY ANGELA WATERCUTTER 10.02.14 | 6:30 AM | PERMALINK

Share 115 Tweet 327 g+1 7 in Share 7 Pin it



Elijah Wood in *Open Windows.* courtesy Cinedigm

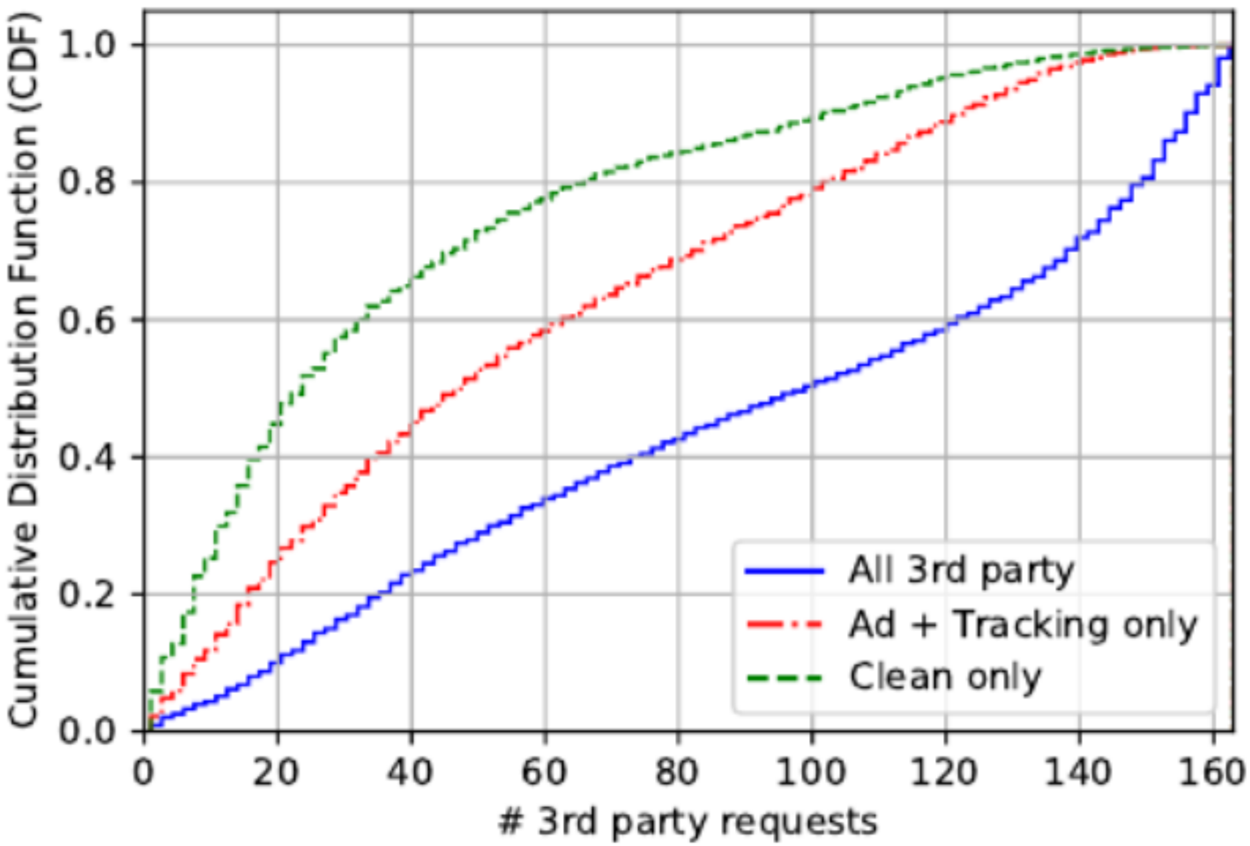
https://www.youtube.com/watch?v=QWw7Wd2gUJk

How it works

And it's not just Facebook!

Facebook now knows you visited this Wired article.
Works for all pages where 'like'/'share' button is embedded!

50% of the websites are using more than 100 3rd party cookies!



https://dl.acm.org/doi/pdf/10.1145/3278532.3278561

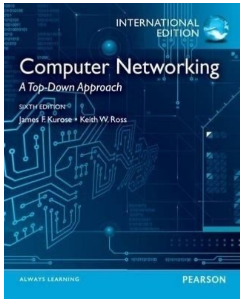
What can we do about it?

- Different ad block products (block cookies/connections to third party sites)
  - Ghostery, Ad Block etc.

- Doesn't completely solve the problem…
  - Trackers getting smarter. Use browser features to fingerprint
  - E.g., combination of installed extensions/fonts etc.
    - Surprisingly unique!
  - Optional fun reading – "Cookie-less monster": http://www.securitee.org/files/cookieless_sp2013.pdf

Chapter 2.2.5

Web Caching



Keep the action close to the user

Evolution of Serving Web Content

- In the beginning…
  - …there was a single server
  - Probably located in a closet
  - And it probably served blinking text

- Issues with this model
  - Site reliability
    - Unplugging cable, hardware failure, natural disaster
  - Scalability
    - Flash crowds (aka Slashdotting)

Replicated Web service

- Device that multiplexes requests across a collection of servers
  - All servers share one public IP
  - Balancer transparently directs requests to different servers

- How should the balancer assign clients to servers?
  - Random / round-robin
    - When is this a good idea?
  - Load-based
    - When might this fail?
- Challenges
  - Scalability (must support traffic for n hosts)
  - State (must keep track of previous decisions)

Load balancing: Are we done?

- Advantages
  - Allows scaling of hardware independent of IPs
  - Relatively easy to maintain
- Disadvantages
  - Expensive
  - Still a single point of failure
  - Location!

Where do we place the load balancer for Wikipedia?

Popping up: HTTP performance

- For Web pages
  - RTT matters most
  - Where should the server go?
- For video
  - Available bandwidth matters most
  - Where should the server go?

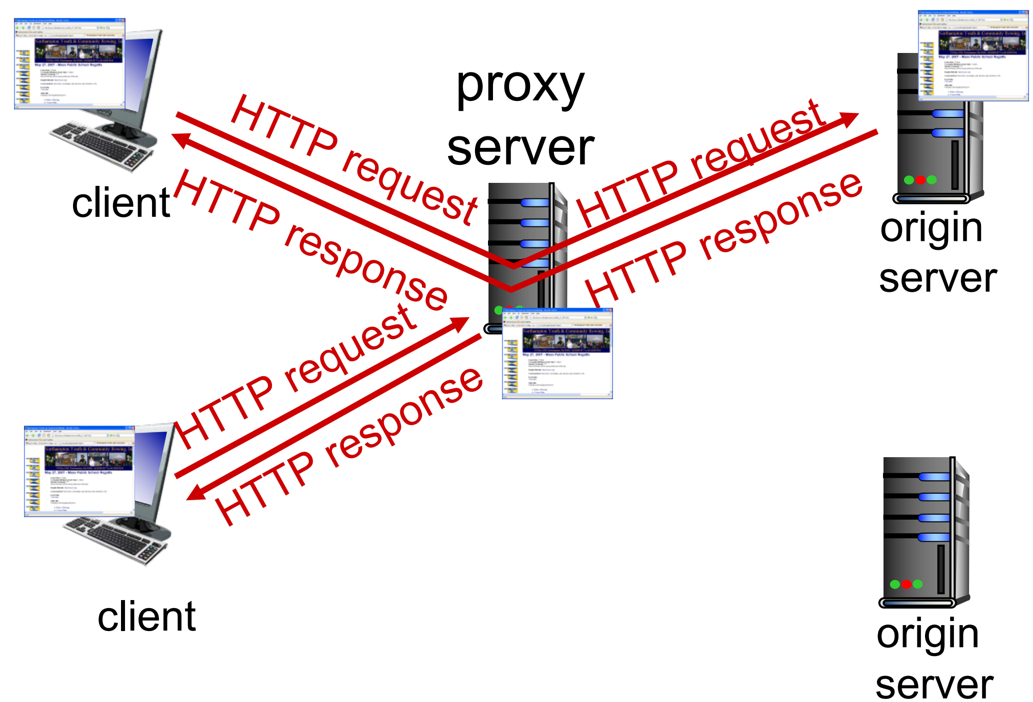
- Is there one location that is best for everyone?

Popping up: HTTP performance

- Impact on user experience
  - Users navigating away from pages
  - Video startup delay
- Impact on revenue
  - Amazon: increased revenue 1% for every 100ms reduction in page load time (PLT)
  - Shopzilla:12% increase in revenue by reducing PLT from 6 seconds to 1.2 seconds
- Ping from LON to NYC: ~100ms

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

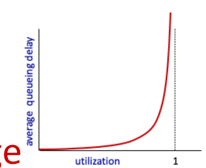
Scenario:

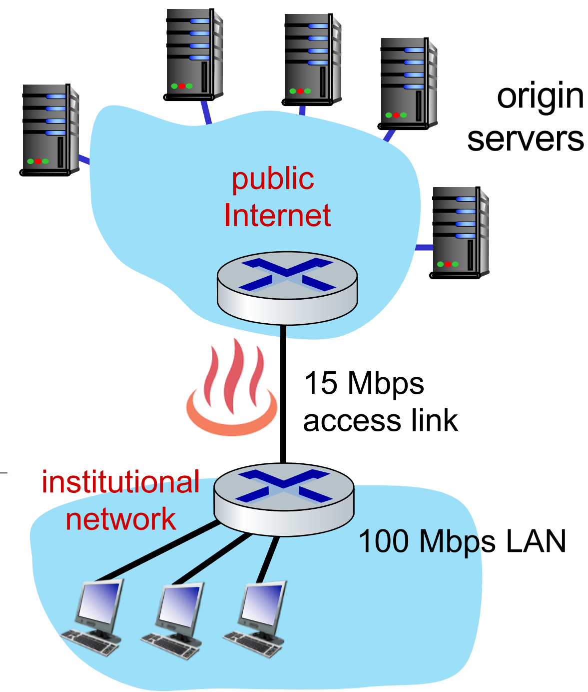
- access link rate: 15 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 1M bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 15 Mbps

Performance:

- access link utilization = 1.0
- LAN utilization: 0.15
- end-end delay = Internet delay + access link delay + LAN delay = 2 sec + minutes + usecs

problem: large queueing delays at high utilization!

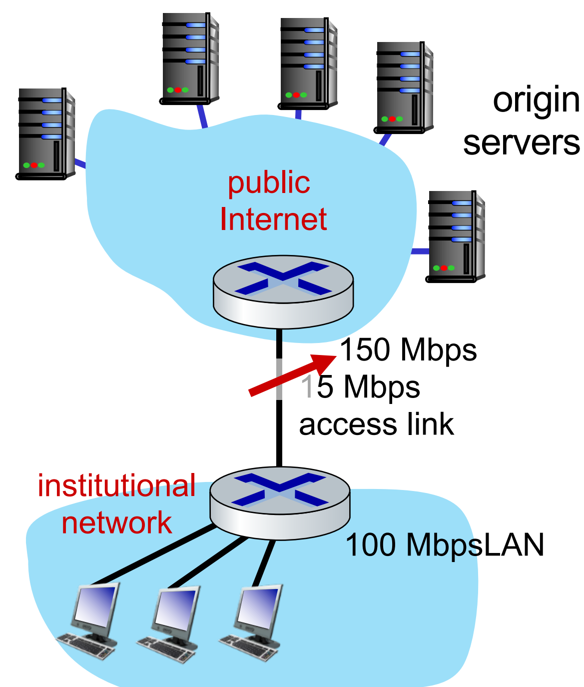




Option 1: buy a faster access link

Scenario:

- access link rate: ~~15 Mbps~~ 150 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 1M bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 15 Mbps



Performance:

- access link utilization = ~~1.0~~ 0.1
- LAN utilization: 0.15
- end-end delay = Internet delay + access link delay + LAN delay
  = 2 sec + ~~minutes~~ msecs + usecs

Cost: faster access link (expensive!)

Option 2: install a web cache

Scenario:

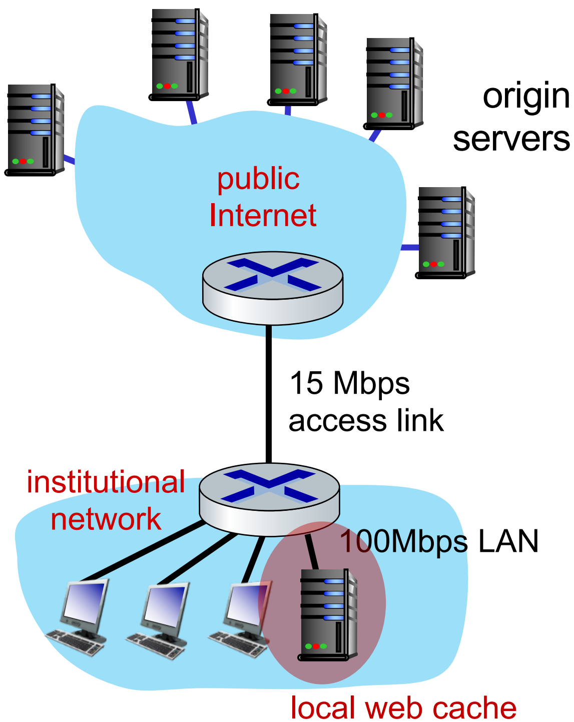
- access link rate: 15 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 1M bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 15 Mbps

Cost: web cache (cheap!)

Performance:

- LAN utilization: .?
- access link utilization = ?
- average end-end delay = ?

How to compute link utilization, delay?

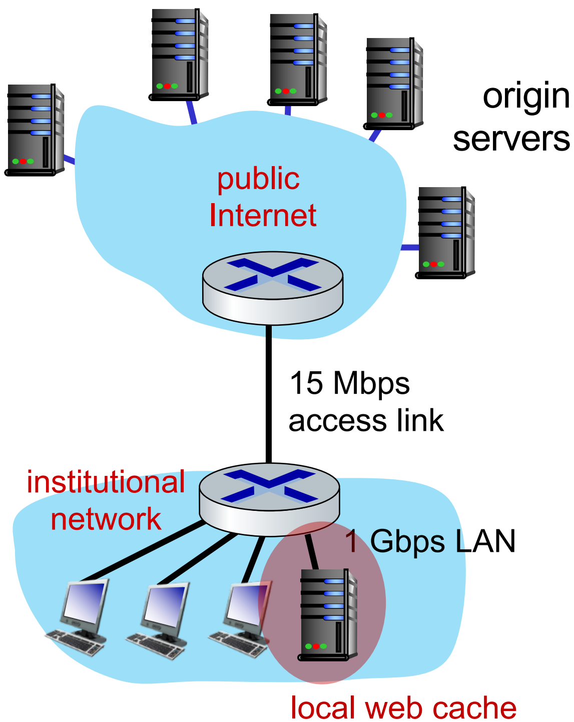


Calculating access link utilization, end-end delay with cache:

suppose cache hit rate is 0.4:

- 40% requests served by cache, with low (msec) delay
- 60% requests satisfied at origin
  - rate to browsers over access link
    = 0.6 * 15 Mbps = 9 Mbps
  - access link utilization = 9/15 = 0.6 is ok (msec) queueing delay at access link
- average end-end delay:

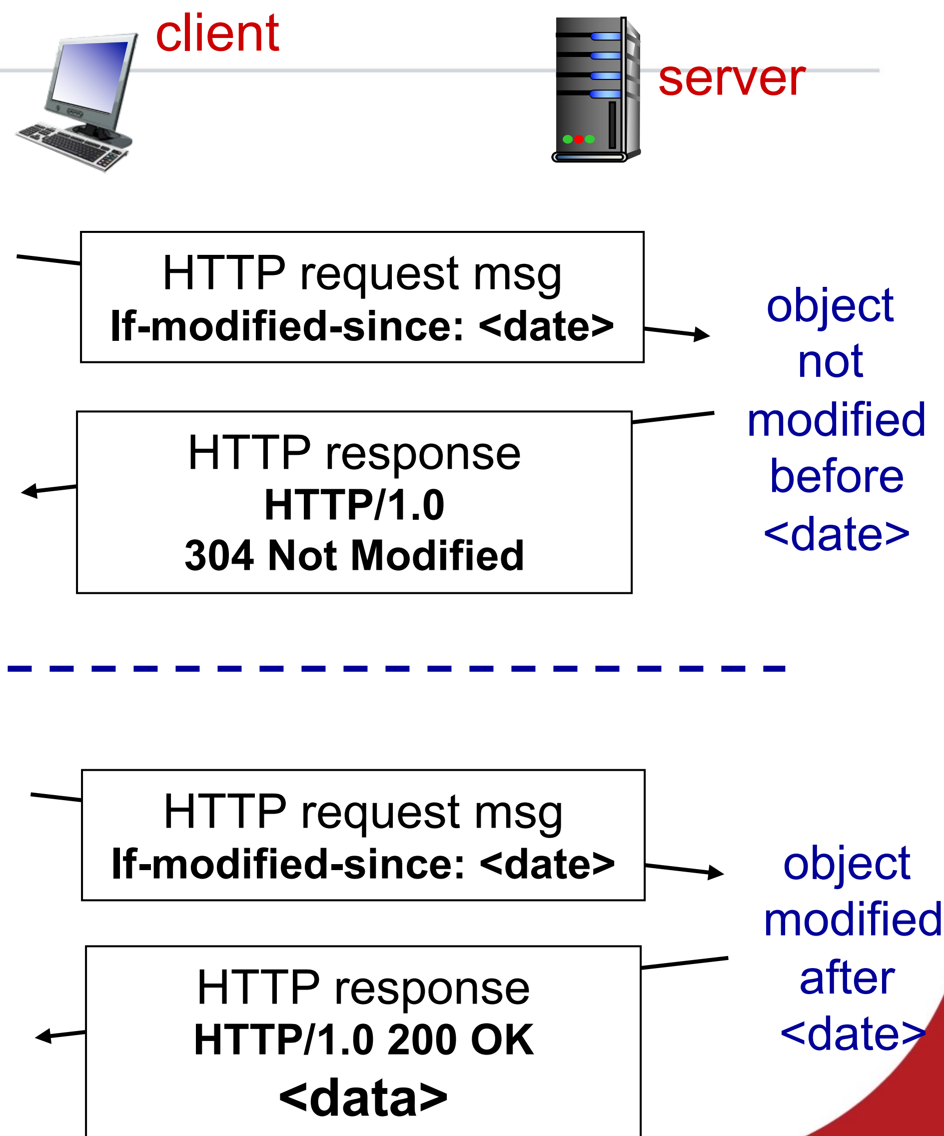
  = 0.6 * (delay from origin servers)
  + 0.4 * (delay when satisfied at cache)
  = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs



lower average end-end delay than with 150 Mbps link (and cheaper too!)

Conditional GET

- Goal: don't send object if client cache has up-to-date cached version
  - No object transmission delay
  - Lower link utilization
- Client 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/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

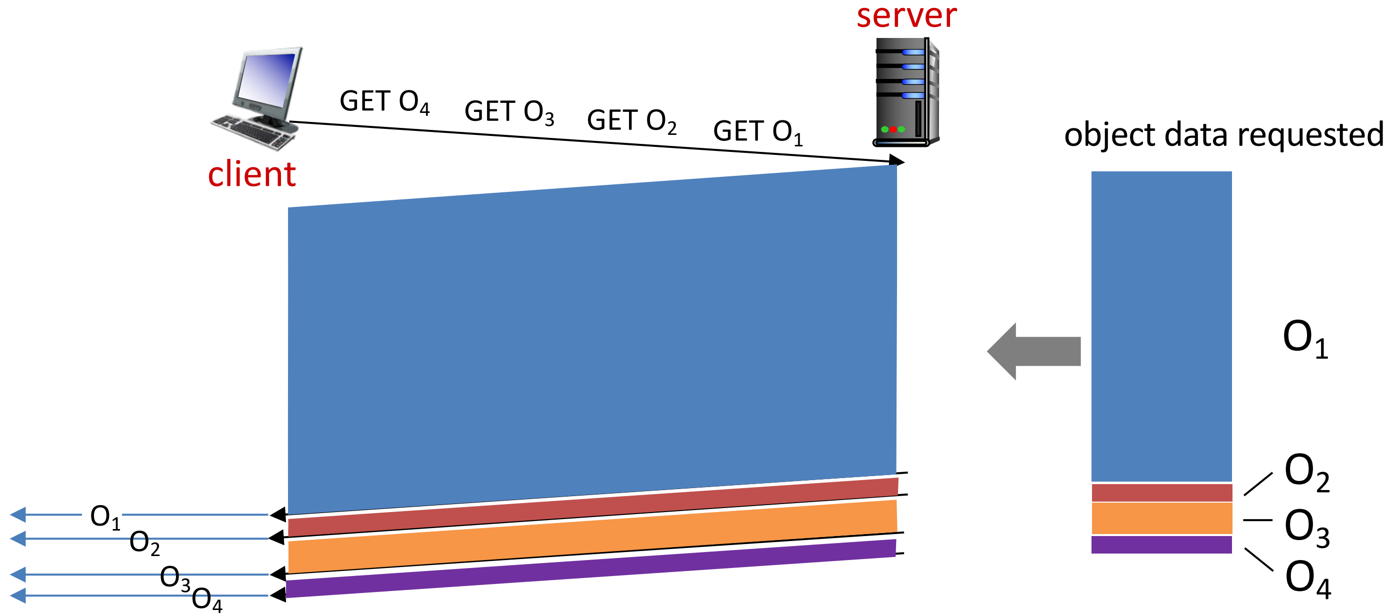
Key goal: decreased delay in multi-object HTTP requests

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

HTTP/2: mitigating HOL blocking

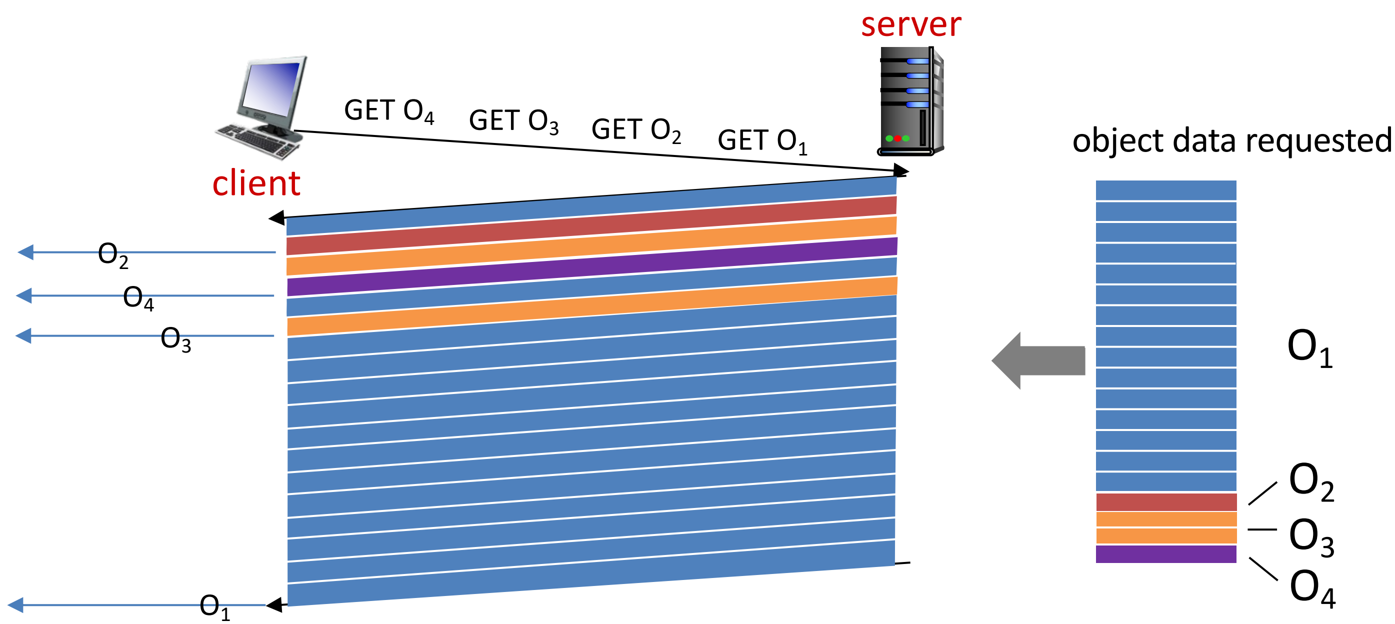
HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects



objects delivered in order requested: O_2, O_3, O_4 wait behind O_1

HTTP/2: mitigating HOL blocking

HTTP/2: objects divided into frames, frame transmission interleaved



O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

HTTP/2 to HTTP/3

HTTP/2 over single TCP connection means:

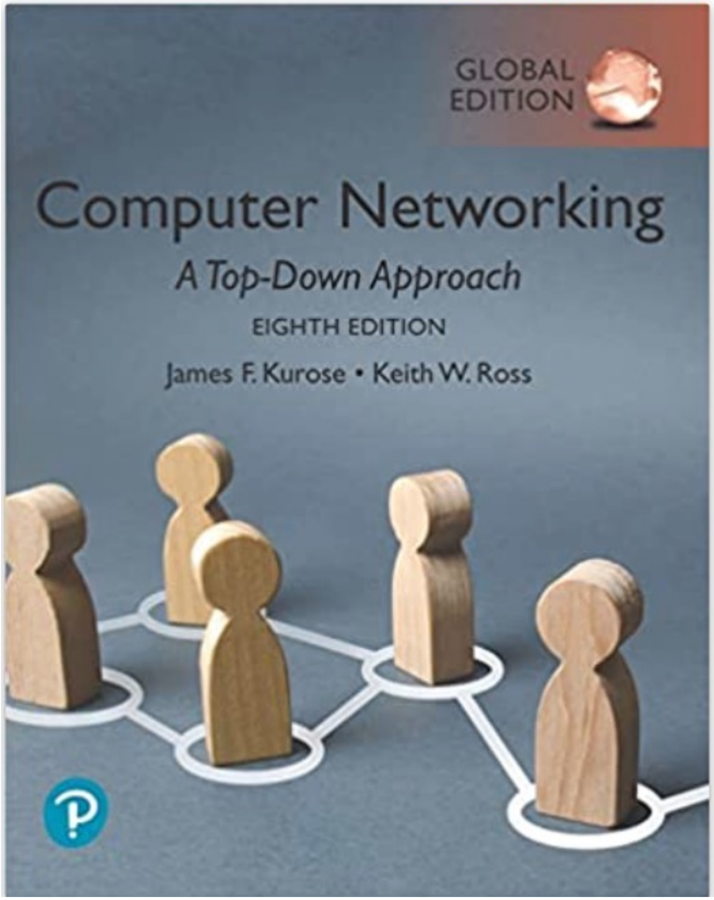
- recovery from packet loss still stalls all object transmissions
  - as in HTTP 1.1, browsers have incentive to open multiple parallel TCP connections to reduce stalling, increase overall throughput
- no security over vanilla TCP connection
- HTTP/3: adds security, per object error- and congestion-control (more pipelining) over UDP
  - more on HTTP/3 in transport layer

Summary

- HTTP provides stateless communication
  - Client/ server based communication
  - Persistent and non-persistent HTTP
  - Pipelining
- Cookies keep information about Client server interaction
  - Client based state information
- Web Caches for improved communication efficiency
  - Caches are located close to users
    - Improved link utilisation
    - Improved access delay

Reading Material

- Section 2.2



Thanks for listening!
Any questions?