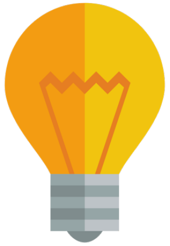
Lecture 03-03: Application Layer – HTTP

CS 326E Elements of Networking

Mikyung Han

mhan@cs.utexas.edu

Internet Reference Model



Example Protocols		Responsible for
FTP, HTTP, SMTP	Application	application specific needs
TCP, UDP	Transport	process to process data transfer
IP	Network	host to host data transfer across different network
Ethernet, WiFi	Link	data transfer between physically adjacent nodes
802.3 PHY	Physical	bit-by-bit or symbol-by-symbol delivery

Outline

1. Web and HTTP recap

Web and HTTP recap

A quick review…

- What does a web page consist of?
  - object can be HTML, JPEG, Java applet, audio,…
  - Should all objects be stored in the same Web server

- Each object is addressable by what?

www.someschool.edu/someDept/pic.gif

host name: www.someschool.edu

path name: /someDept/pic.gif

HTTP is a __________ protocol

- Server or client does not track "state" of each other
- Each request/response pair is independent of each other
- No past requests affect the current request
- No need for client/server to recover from a partially-completed transaction

Enables server to handle massive client requests!

If HTTP is stateless,
how come web servers remembers me?

Name 4 places to find cookie

HTTP uses ______ as underlying transport

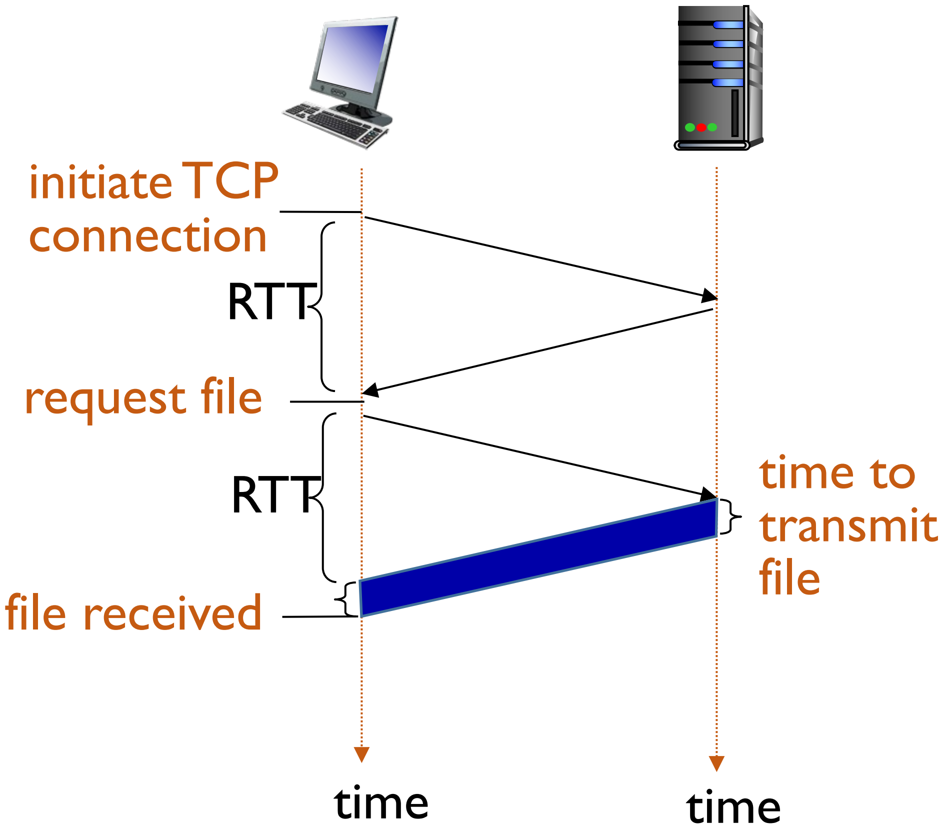
You have an html page that references 3 objects

- How many TCP connections are used with HTTP 1.0?
  - Is HTTP 1.0 persistent or non-persistent?
- How about HTTP 1.1?

Non-persistent HTTP
takes 2 RTT + object transmission time per object!

RTT (definition): Round trip time between client and server

HTTP response time (per object):
- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- object/file transmission time



This was HTTP 1.0

Outline

1. Web and HTTP
2. HTTP 2.0
3. Web Cookies
4. Web Cache
5. Making web even faster

Goal: Reduce delay in multi-object HTTP requests

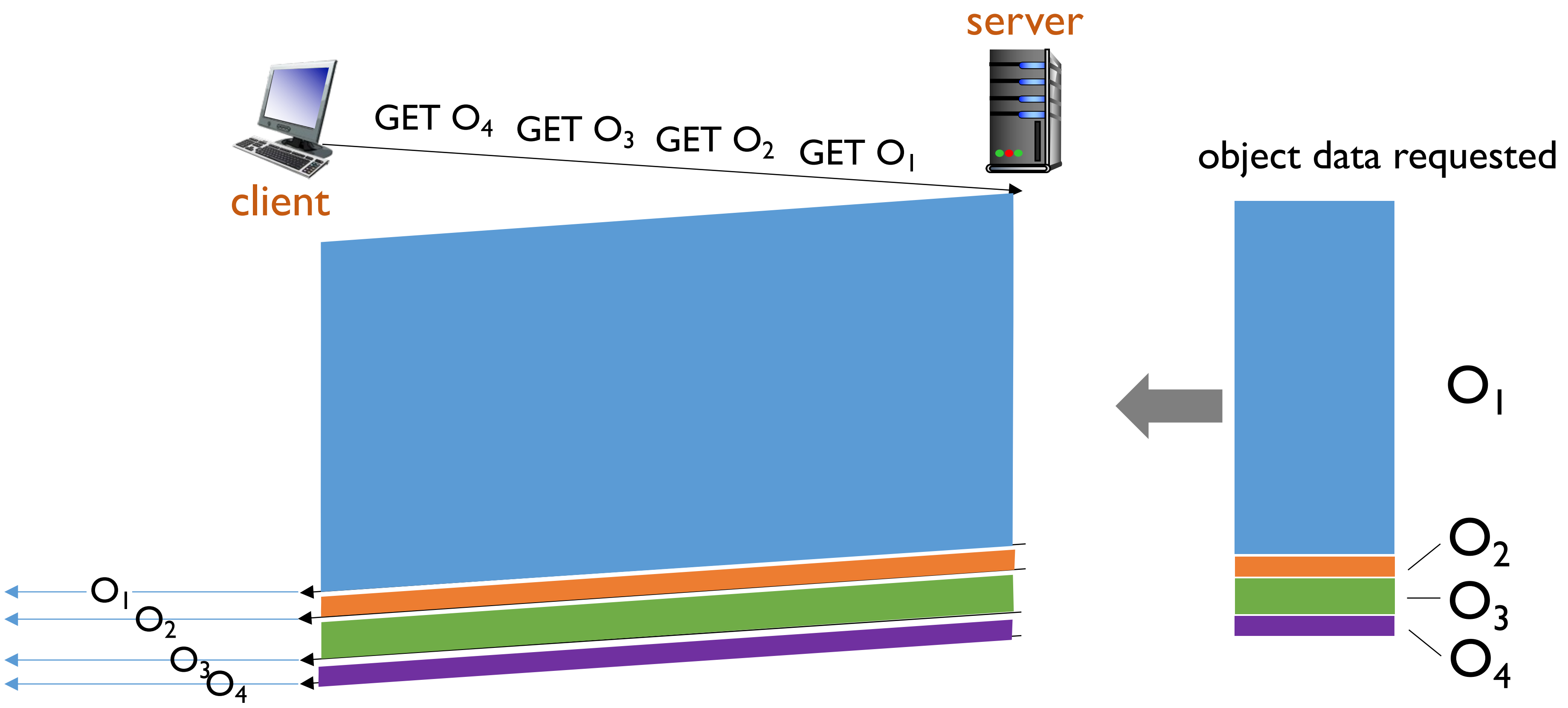
HTTP1.1 is first-come-first-served

- Server responds in-order to GET requests

Why this is BAD?

HTTP/1.1 suffers from HOL blocking

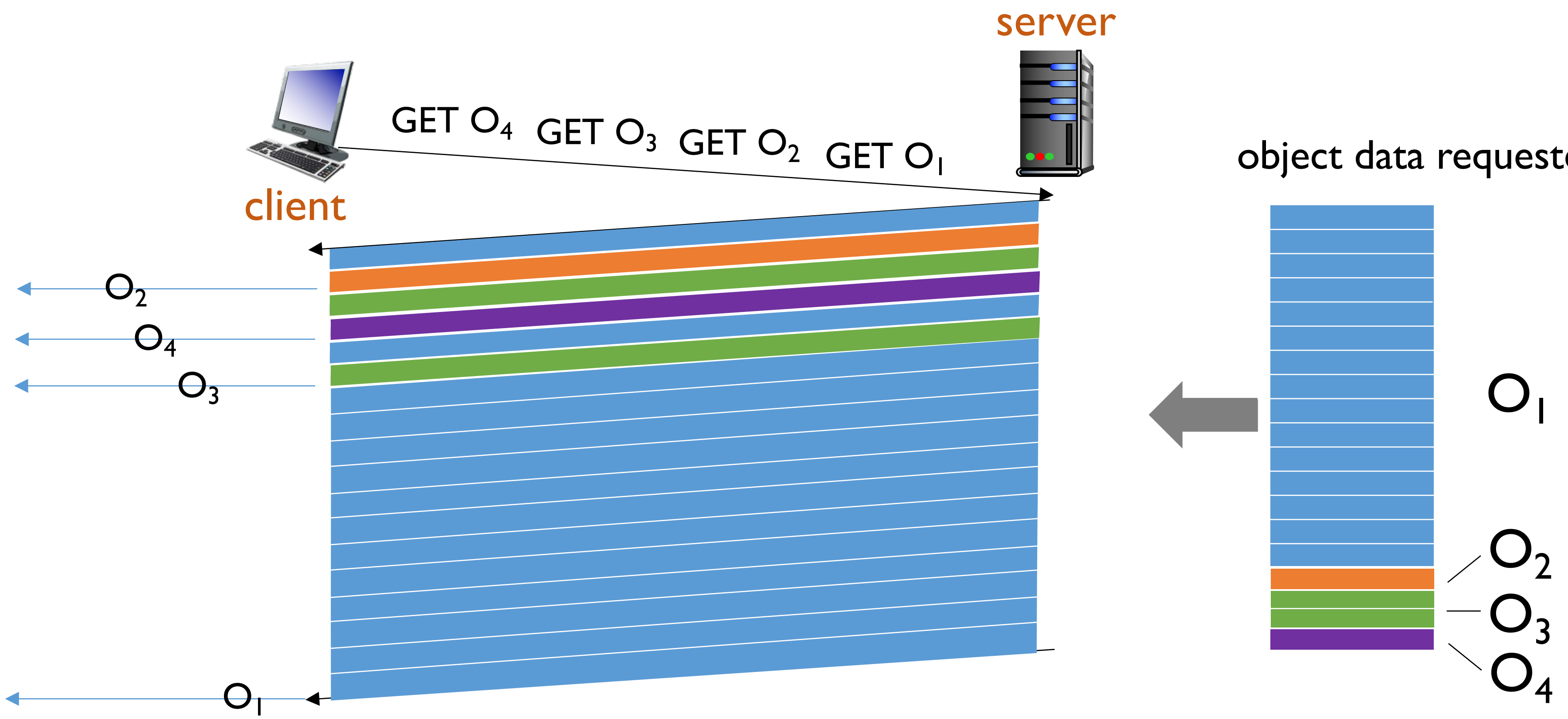
client requests 1 large object (e.g., video file) and 3 smaller objects



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

HTTP/2 mitigates 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 aims to further reduce the delay by increasing flexibility in server when sending objects

[RFC 7540, 2015]

- divide objects into frames, schedule frames to mitigate HOL blocking
- transmission order of requested objects based on client-specified object priority (not necessarily FCFS)
- server push: pre-sends yet-to-be requested objects to client
  - Parses tags such as `link` `script` `img` `source` `audio` `video` `track`, etc.

HTTP/3 adds per-object error, congestion control, and security over UDP

- Short comings of HTTP/2
  - Recovery from a packet loss still stalls all object transmissions
  - No security over vanilla TCP connection

More on HTTP/3 in Ch 3 transport layer!

Outline

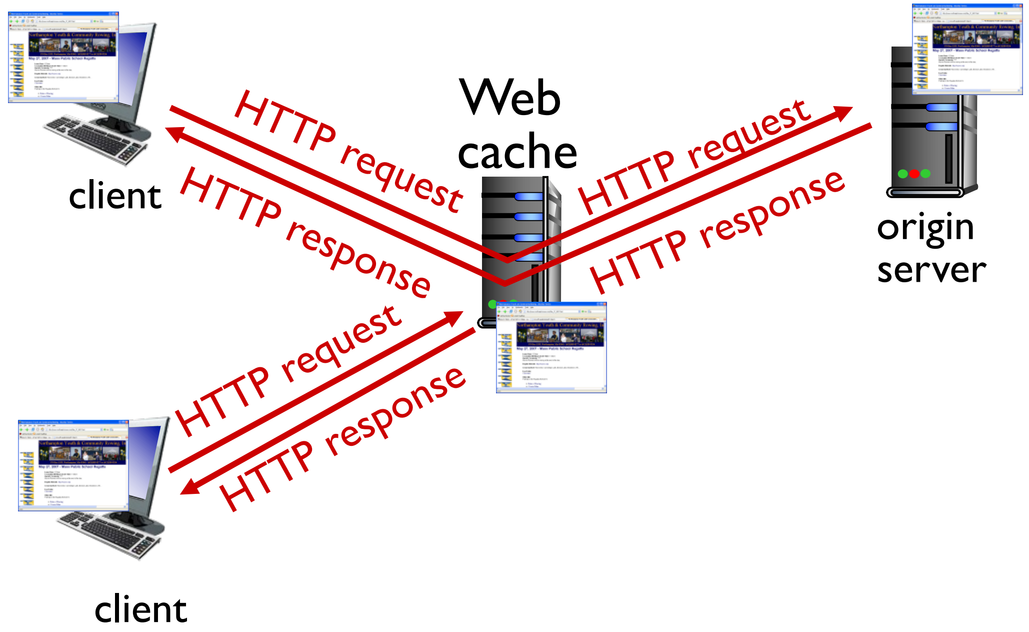
1. Web and HTTP recap
2. HTTP 2.0
3. Making web even faster: Web caching

Motivation: How to make HTTP request even faster?

Original server – slow or even not available

Web caches serves client requests quickly without involving origin server

- user configures browser to point to a (local) Web cache
- browser sends all HTTP requests to cache
  - if object in cache: cache returns object to client
  - else cache requests object from origin server, caches received object, then returns object to client



Web caches (aka proxy servers)

- Web cache acts as both client and server
  - server for original requesting client
  - client to origin server
- server tells cache about object's allowable caching in response header:

```
Cache-Control: max-age=<seconds>
```

```
Cache-Control: no-cache
```

Why Web caching?

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

Caching example

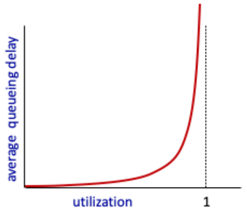
Scenario:

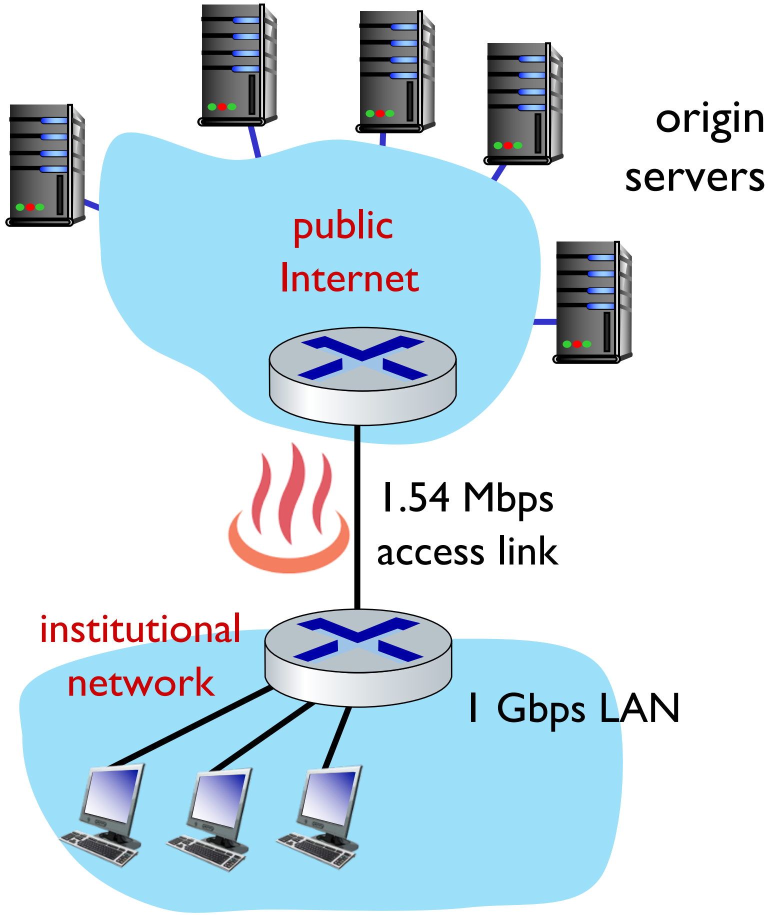
- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- avg request rate from browsers to origin servers: 15 obj/sec
- avg data rate to browsers: 1.50 Mbps

Performance:

- access link utilization = .97
- LAN utilization: .0015
- end-end delay = Internet delay + access link delay + LAN delay = 2 sec + minutes + usecs

problem: large queueing delays at high utilization!



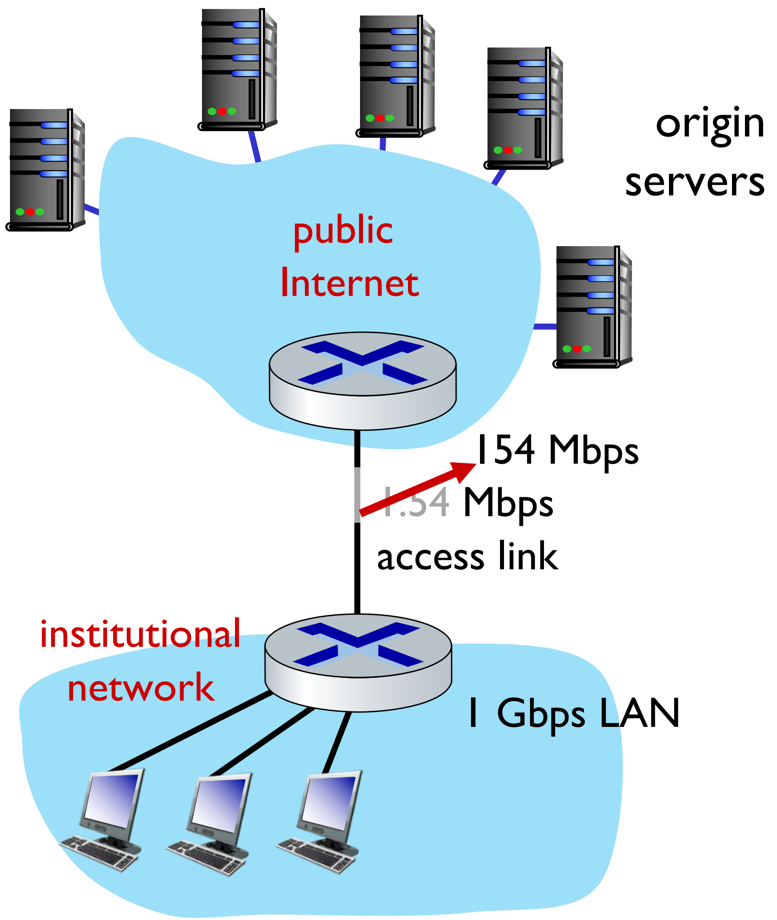


Option 1: buy a faster access link

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

Performance:
- access link utilization = ~~.97~~ → .0097
- LAN utilization: .0015
- end-end delay = Internet delay + access link delay + LAN delay
  = 2 sec + ~~minutes~~ → msecs + usecs

Cost: faster access link (expensive!)



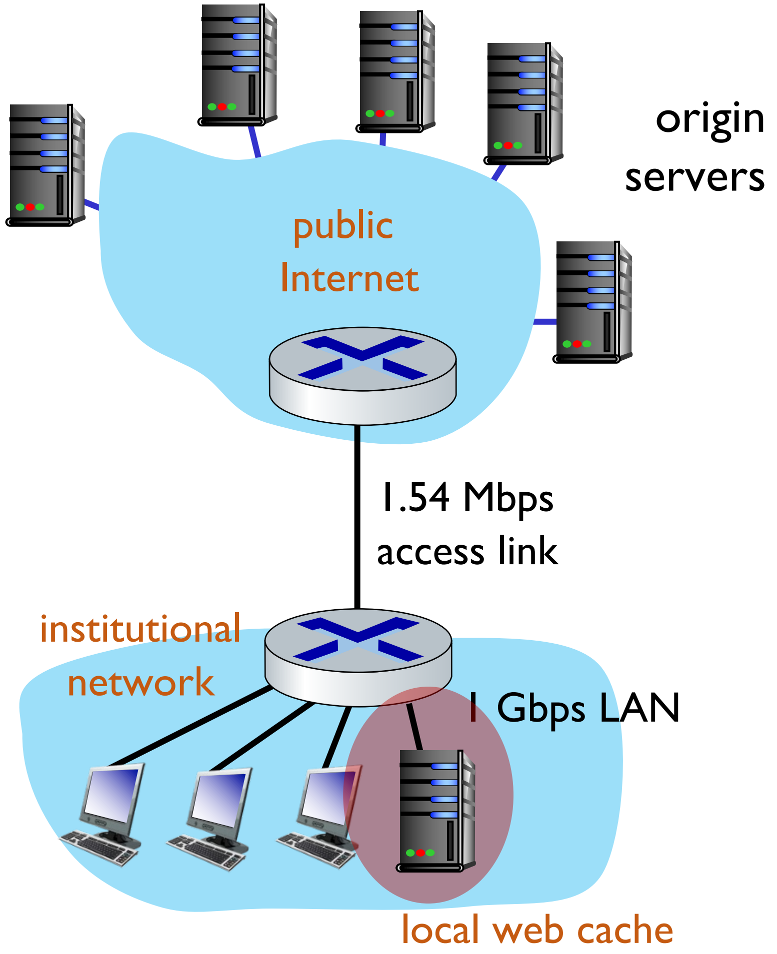
Option 2: install a web cache

Scenario:

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

Cost: web cache (cheap!)

Performance: ?

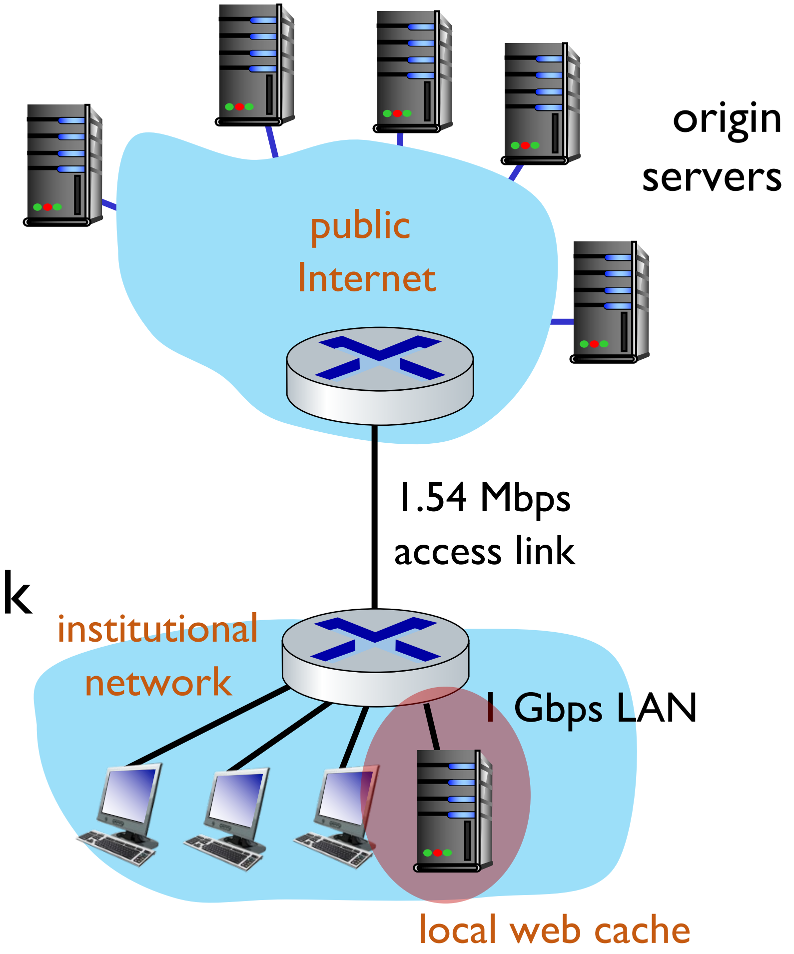


How to compute link utilization and delay?

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 * 1.50 Mbps = .9 Mbps
  - access link utilization = 0.9/1.54 = .58
    means low (10 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 154 Mbps link (and cheaper too!)

What if objects in cache gets stale?

HTTP cache has a way to check if objects are up-to-date

HTTP/1.1 Conditional GET [RFC 7232]

Server does NOT send object if cache has up-to-date cached version

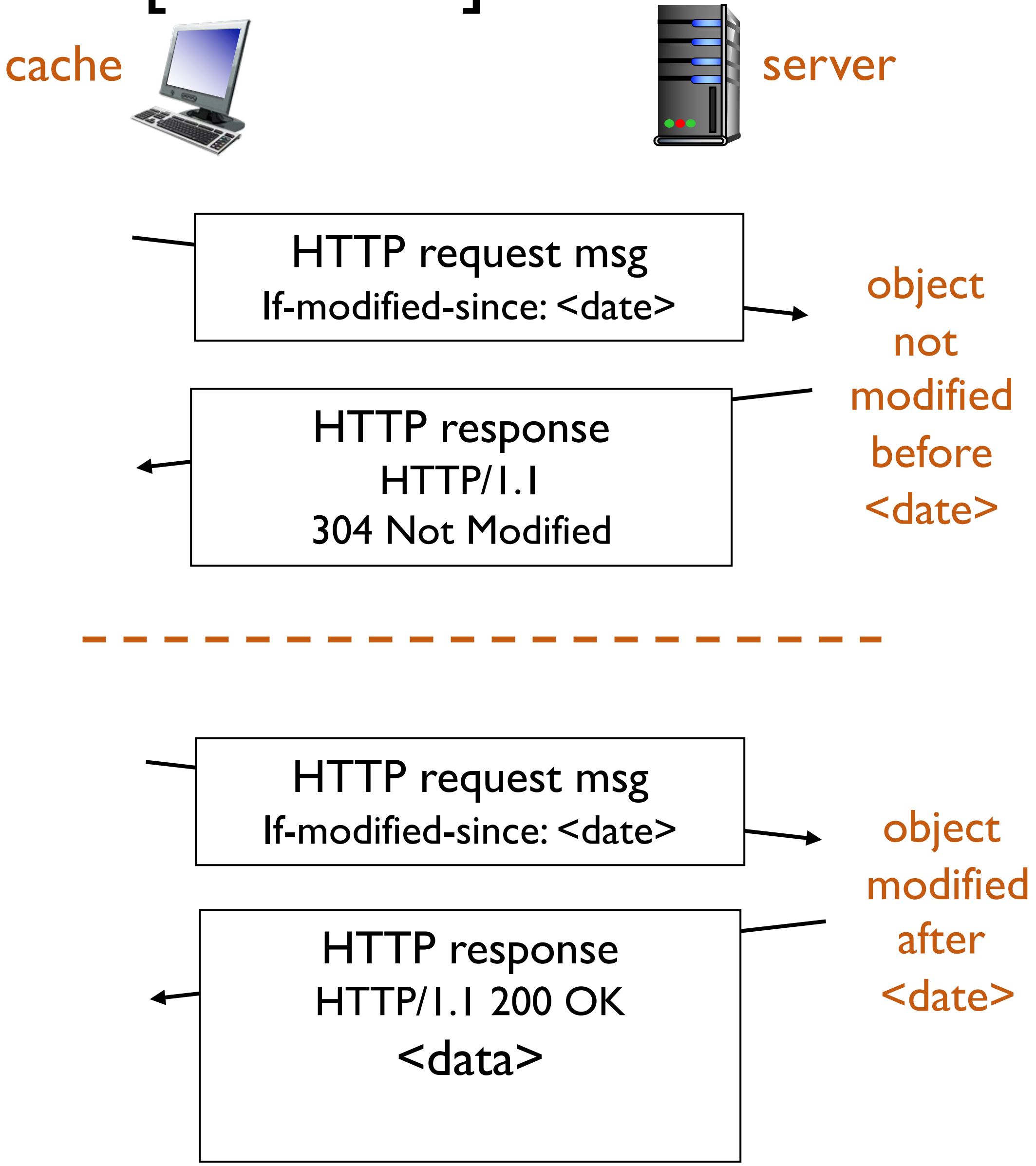
- no object transmission delay (or use of network resources)

- 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



Acknowledgements

Slides are adopted from Kurose' Computer Networking Slides