

Revisit: Persistent HTTP (HTTP 1.1)

Non-persistent HTTP issues:

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

Persistent HTTP (HTTP1.1):

- 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 (cutting response time in half)

Revisit: Maintaining user/server state: cookies

Web sites and client browser use *cookies* to maintain some state between transactions

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 uses browser on laptop, visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, *site creates:*
 - *unique ID (aka "cookie")*
 - entry in backend database for ID
- subsequent HTTP requests from Susan to this site *will contain cookie ID value, allowing site to "identify" Susan*

HTTP cookies: comments

What cookies can be used for:

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

Challenge: How to keep state:

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

aside

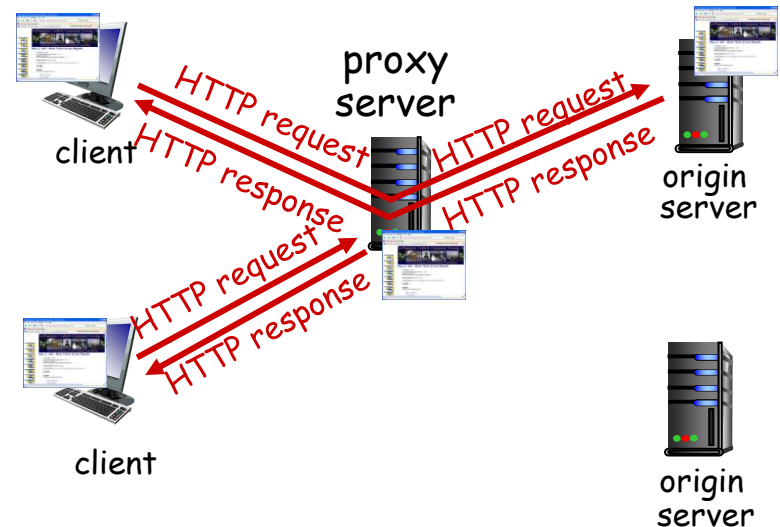
cookies and privacy:

- cookies permit sites to *learn* a lot about you on their site.
- third party persistent cookies (tracking cookies) allow common identity (cookie value) to be tracked across multiple web sites

Web caches (proxy servers)

Goal: satisfy client request without involving origin server

- user configures browser to point to a **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 (proxy servers)

- Web 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
 - 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

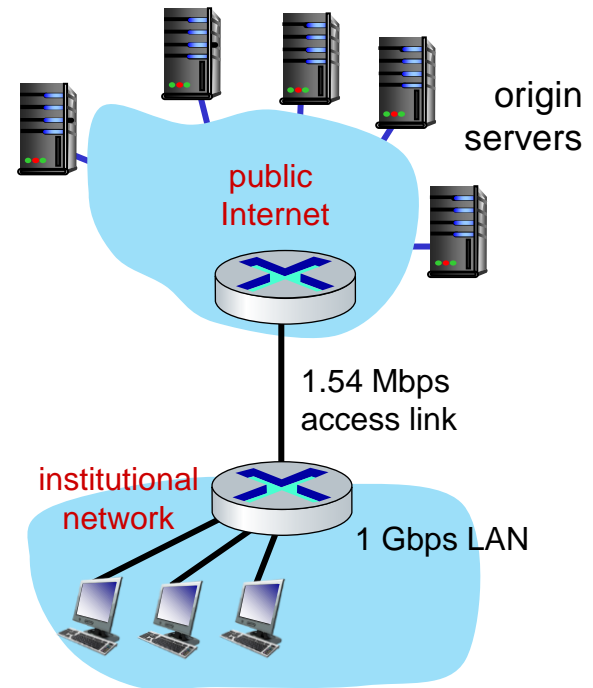
Scenario:

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

Performance:

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

problem:
large delays
at high
utilization!



Caching example: buy a faster access link

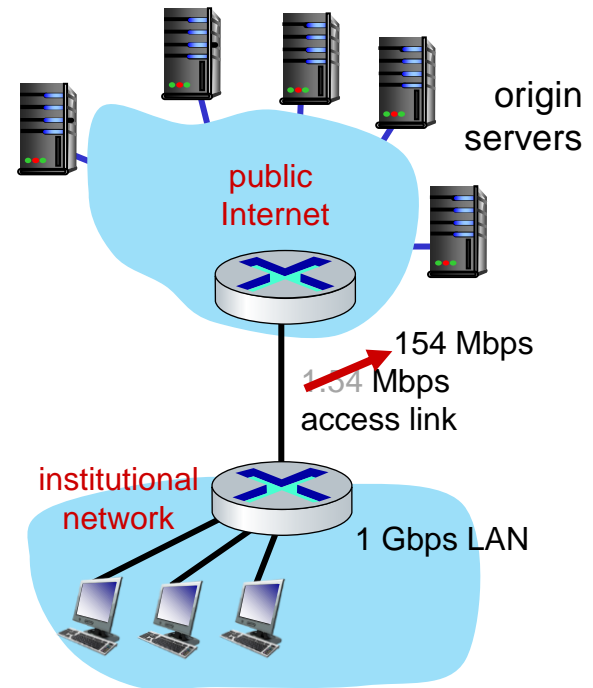
Scenario:

- access link rate: ~~1.54 Mbps~~ ^{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
 - average data rate to browsers: 1.50 Mbps

Performance:

- LAN utilization: .0015
- access link utilization = ~~.97~~ ^{.0097}
- end-end delay = Internet delay +
access link delay + LAN delay
= 2 sec + ~~minutes~~ ^{msecs}

Cost: faster access link (expensive!)



Caching example: install a web cache

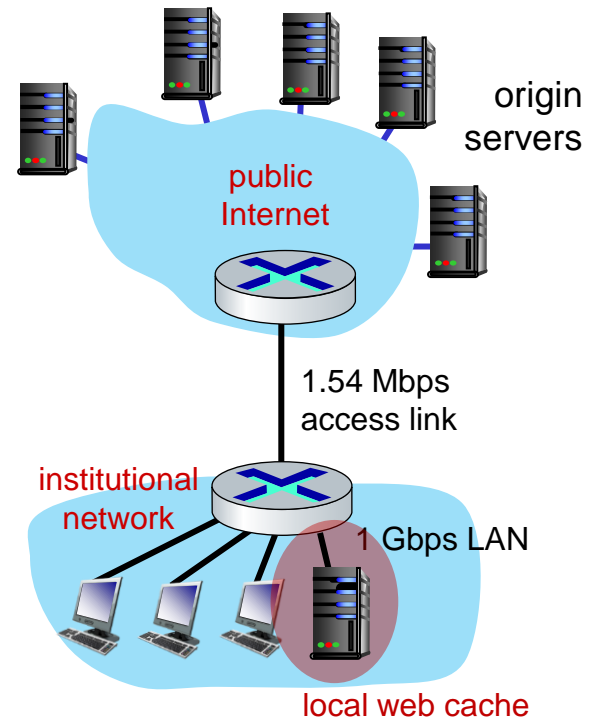
Scenario:

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

Performance:

- LAN utilization: .?
 - access link utilization = ?
 - average end-end delay = ?
- How to compute link utilization, delay?*

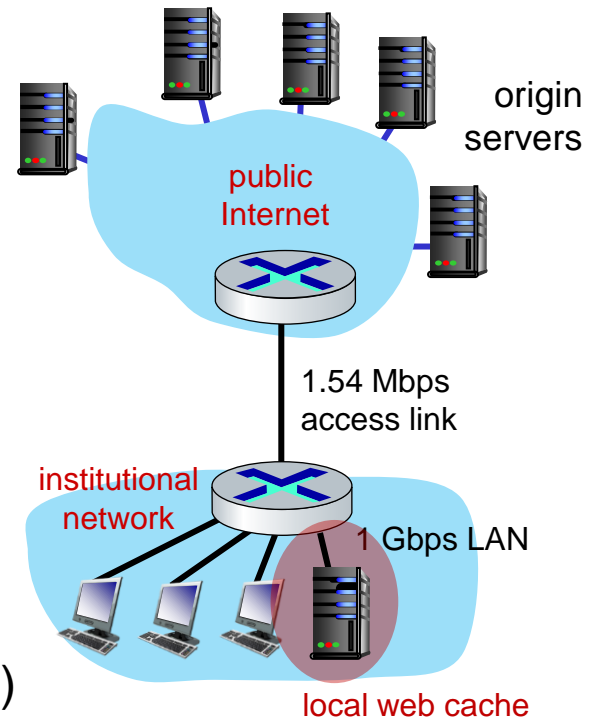
Cost: web cache (cheap!)



Caching example: install a web cache

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

- suppose cache hit rate is 0.4: 40% requests satisfied at cache, 60% requests satisfied at origin
- access link: 60% of requests use access link
- data rate to browsers over access link
 $= 0.6 * 1.50 \text{ Mbps} = .9 \text{ Mbps}$
- utilization $= 0.9 / 1.54 = .58$
- average end-end delay
 $= 0.6 * (\text{delay from origin servers})$
 $+ 0.4 * (\text{delay when satisfied at cache})$
 $= 0.6 (2.01) + 0.4 (\sim \text{msecs}) = \sim 1.2 \text{ secs}$



lower average end-end delay than with 154 Mbps link (and cheaper too!)

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

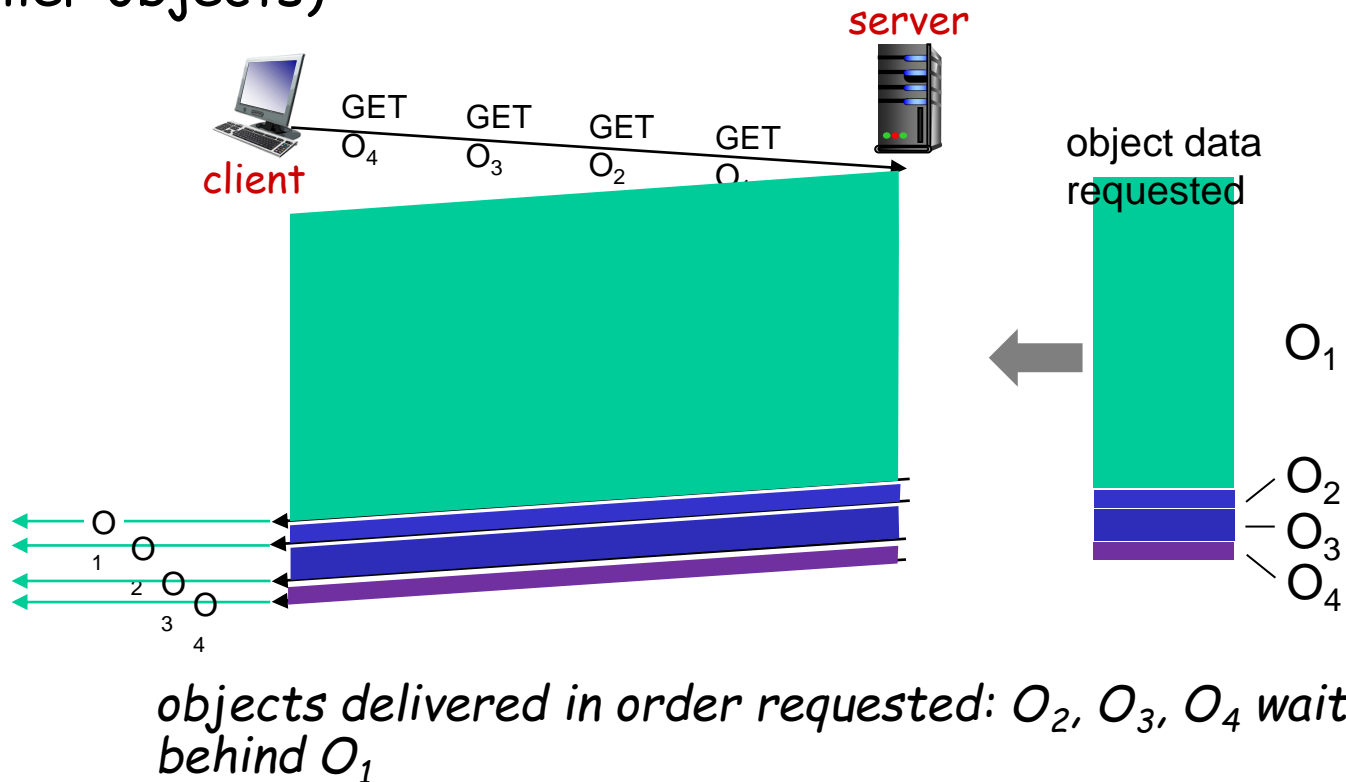
Key goal: decreased delay in multi-object HTTP requests

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

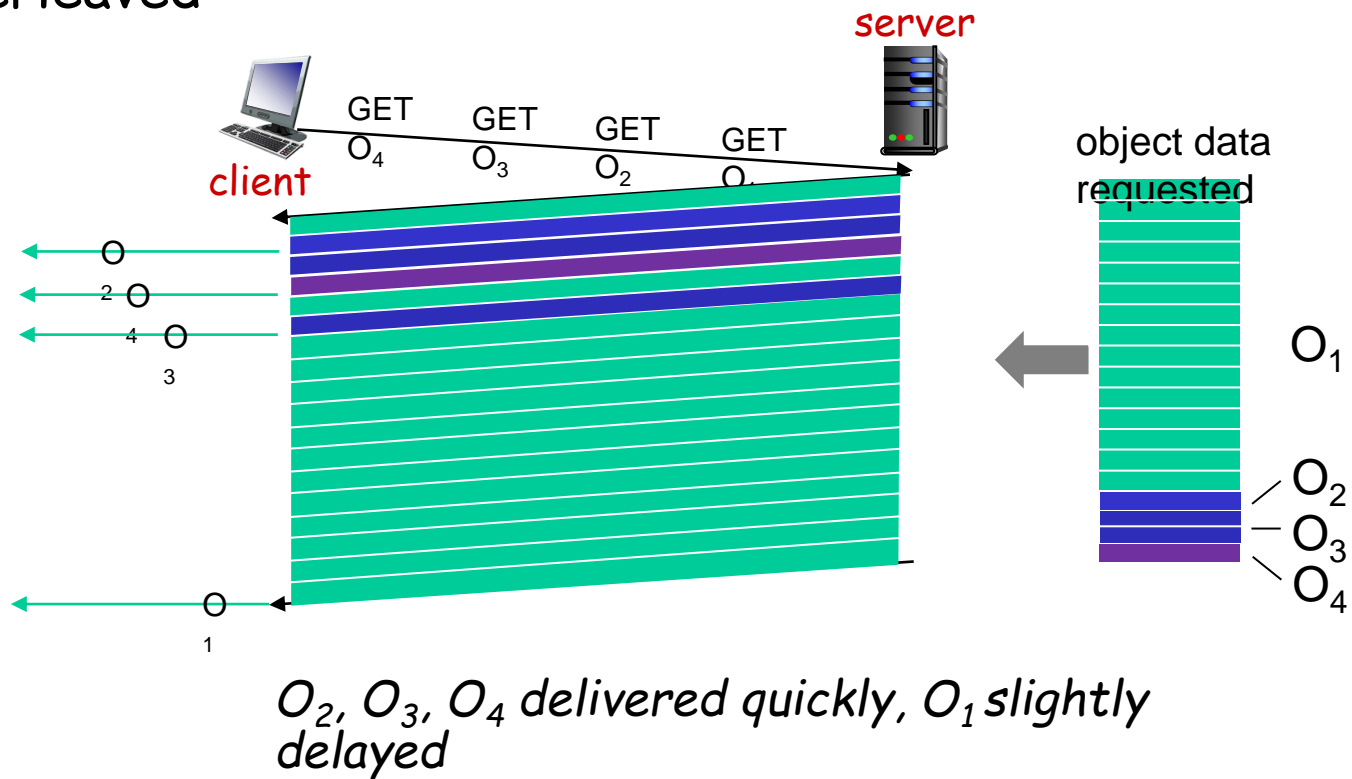
HTTP/2: mitigating HOL blocking

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



HTTP/2: mitigating HOL blocking

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



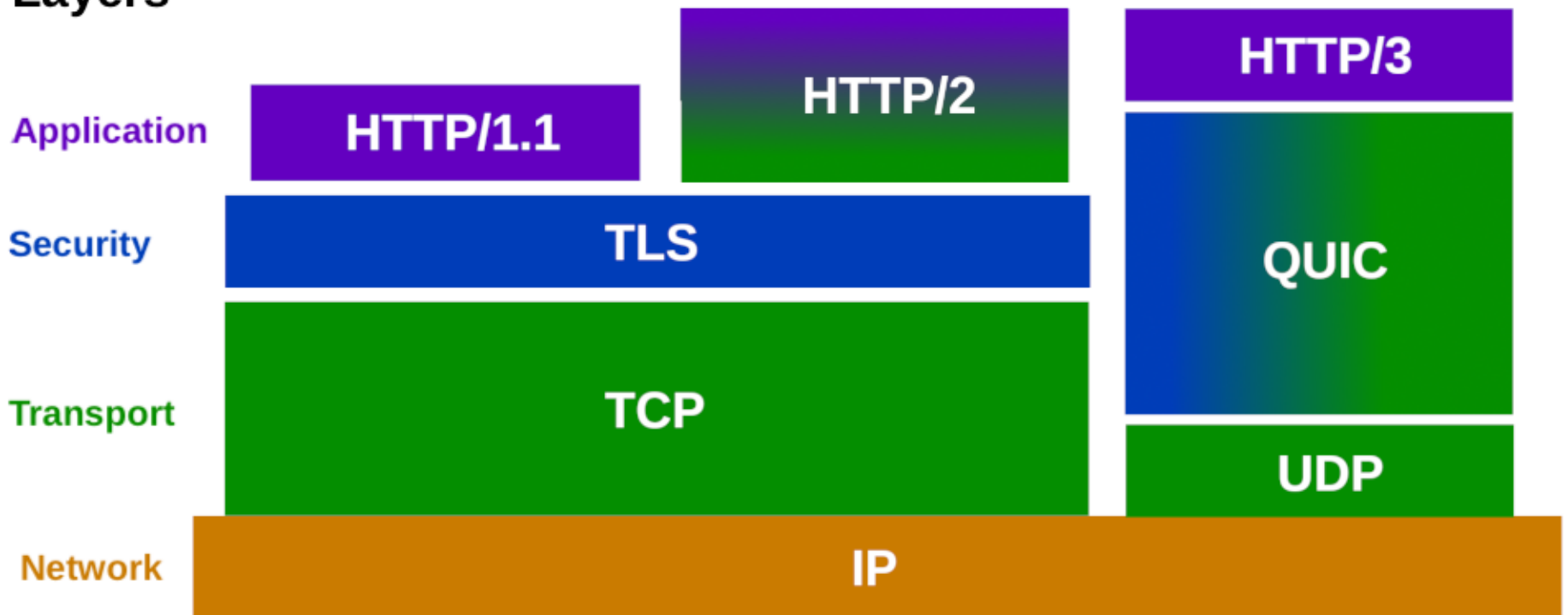
HTTP/2 to HTTP/3

Key goal: decreased delay in multi-object HTTP requests

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

Layers



www.humanlevel.com

Application Layer: Overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP

