COMP 431
Internet Services & Protocols

HTTP Persistence & Web Caching

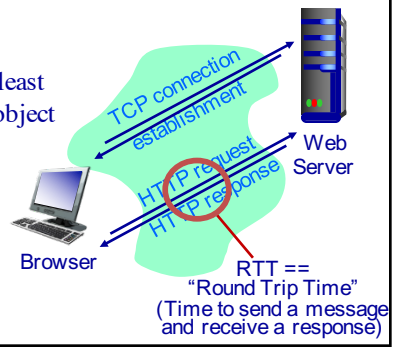
Jasleen Kaur

January 30, 2020

HTTP Protocol Design
Non-persistent connections

- The default browser/server behavior in HTTP/1.0 is for the connection to be closed after the completion of the request
  - » Server parses request, responds, and closes TCP connection
  - » The `Connection: keep-alive` header allows for persistent connections

- With non-persistent connections at least 2 RTTs are required to fetch every object
  - » 1 RTT for TCP handshake
  - » 1 RTT for request/response

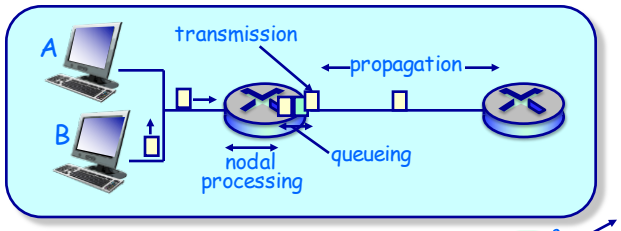


Assume everything comes from the same server

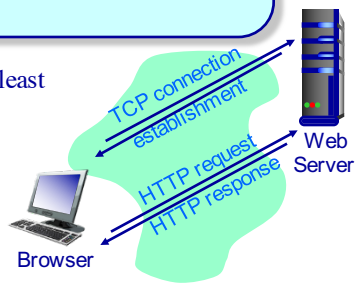
RTT = round trip time = time to go to the server and to get the response back\

webpage with 10 images is going to take 22 RTTs to retrieve
2 for the webpage and 20 for all of the images

Non-Persistent Connections
Performance



- With non-persistent connections at least 2 RTTs are required to fetch every object
  - » 1 RTT for TCP handshake
  - » 1 RTT for request/response

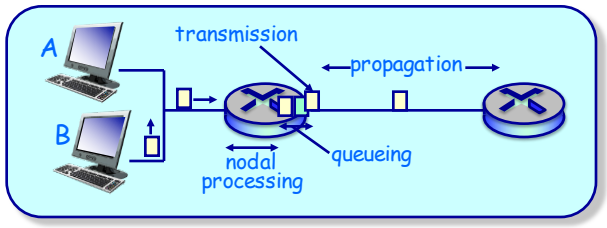


3

Which is part is going to dominate is depends on the path that is taken and the distance

Not much traffic - queuing delay will be small

Non-Persistent Connections
Performance



- Example: A 1 Kbyte base page with five 1 Kbyte embedded images coming from the West coast on an OC-48 link
  - » 1 RTT for TCP handshake = 0.001 *ms* + 50 *ms*
  - » 1 RTT for request/response = 0.006 *ms* + 50 *ms*
- Page download time with non-persistent connections?
- Page download time with a persistent connection?

4

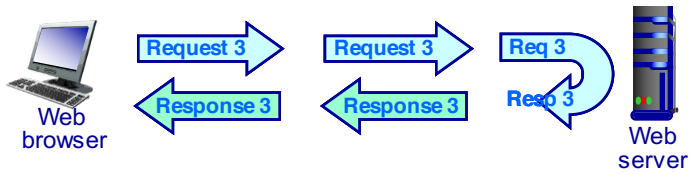
Non-persistent:
(50+.001)*6+(.006+50)*6
1 for first webpage
5 for each image

every object I get I need to open up a connection in non-persistent

Persistent: (50+.001)*2+(.006+50)*5
1 tcp connection setup
1 base webpage
get 5 objects

Persistent Connections

Persistent connections with pipelining



What is the page download time in previous example with a pipelined connection?

- Pipelining in a persistent connection allows a client to make a next request before a response to the previous request has been received
  - » Connections are persistent and pipelined by default in HTTP/1.1
- Page download time with a persistent, pipelined connection?

Pipeline persistence

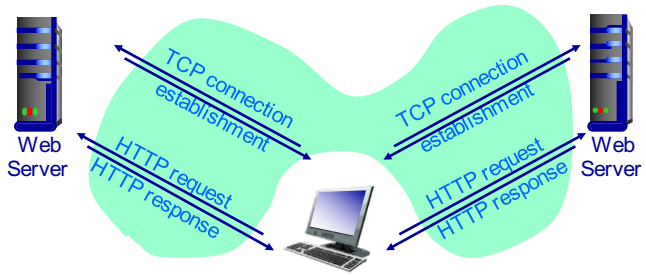
base webpage: 100.007
first image: 50+.006
2nd image: .006
3rd image: .006
4th image: .006
5th image: .006

Figure 1

Non-Persistent Connections

Parallel connections

- To improve performance a browser can issue multiple requests in parallel to a server (or servers)
  - » Server parses request, responds, and closes TCP connection

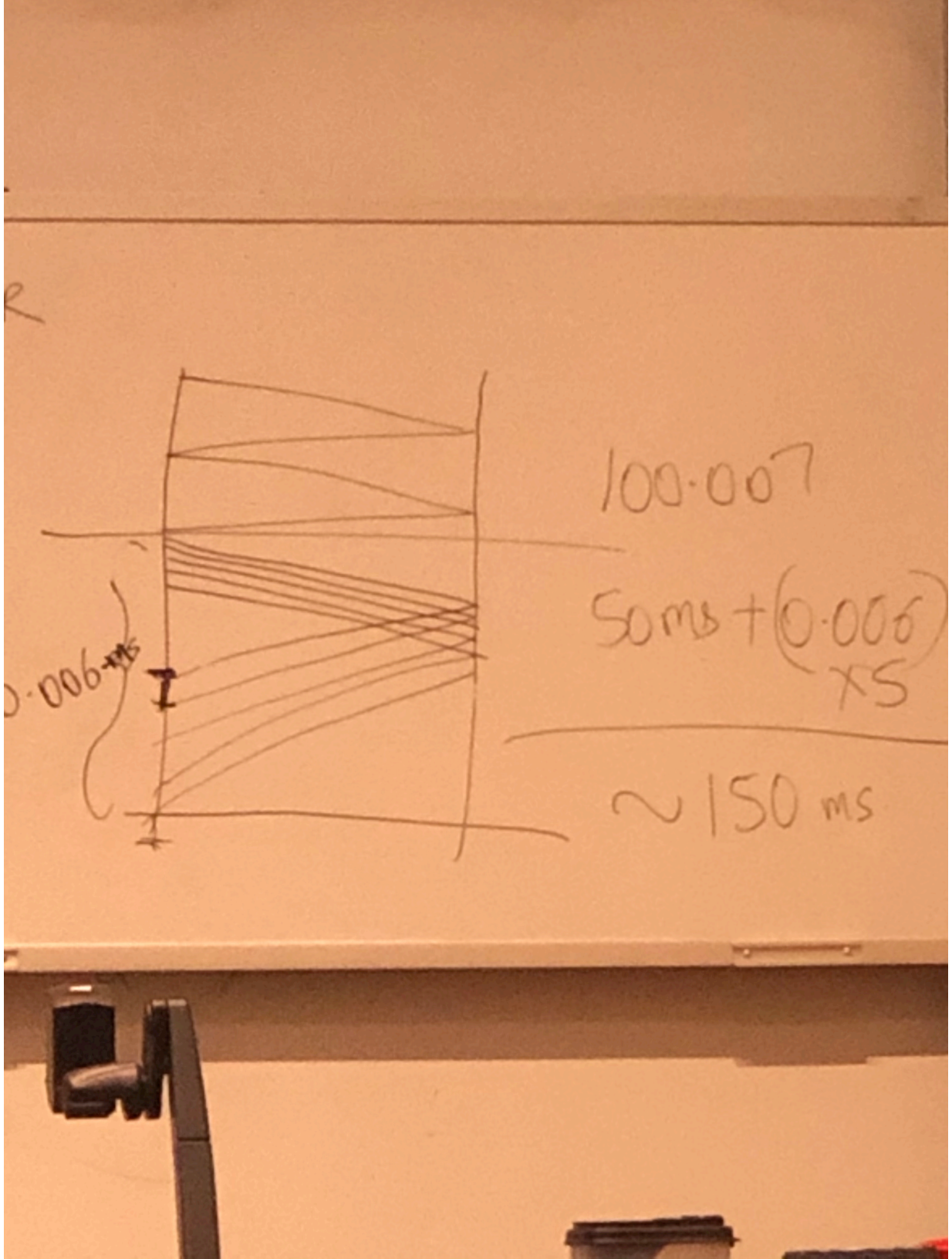


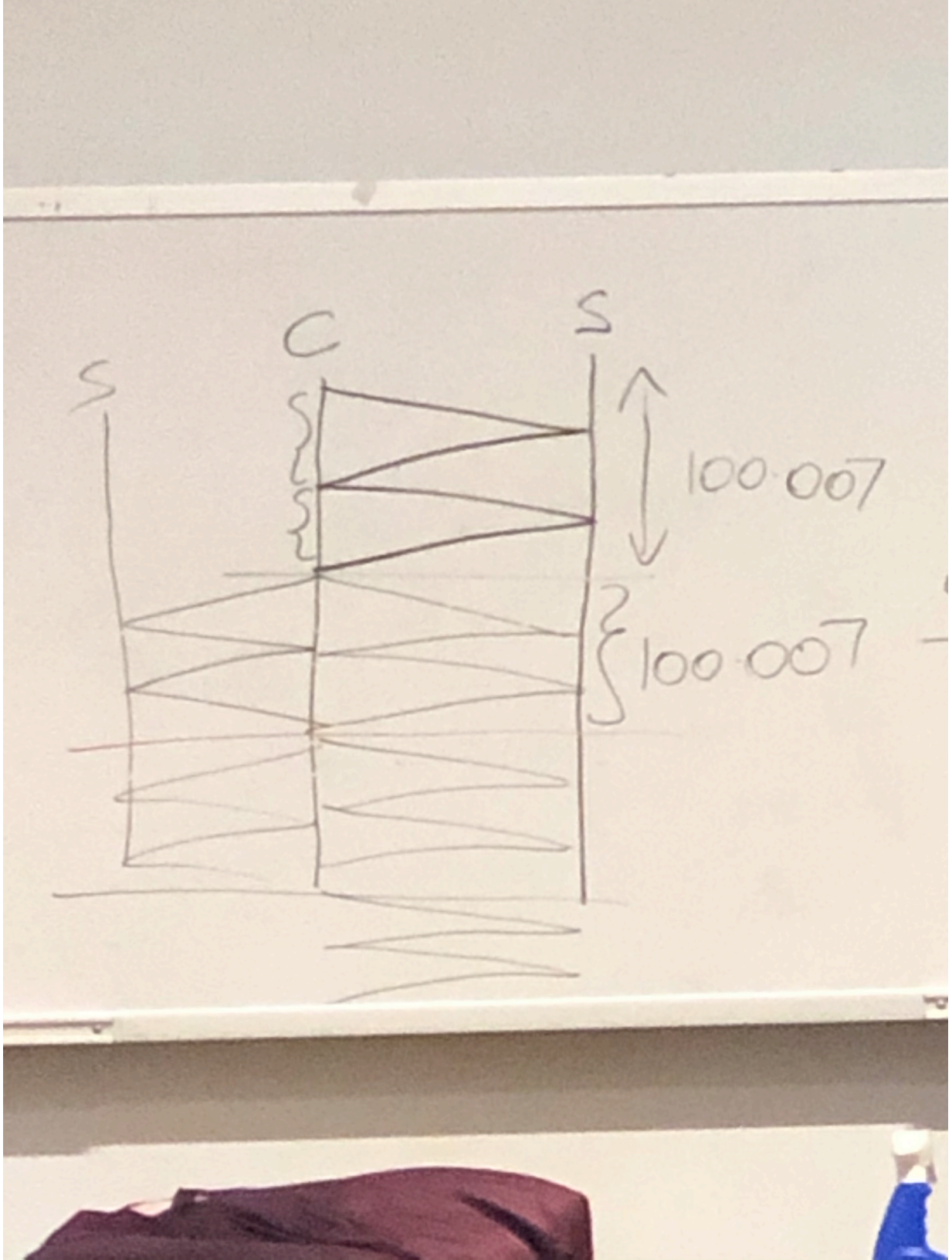
- Page download time with parallel connections?
  - » 2 parallel connections =
  - » 4 parallel connections =

2 parallel connections
base webpage: 100ms
first 2 objects: 100ms
next 2 objects: 100 ms
last object: 100ms

Figure 2

4 Parallel connections
base webpage: 100ms
first 4 objects: 100ms
last object: 100ms


100.007
50ms + (0.006 x5
~150 ms


S
C
S
100·007
100·007

HTTP Protocol Design

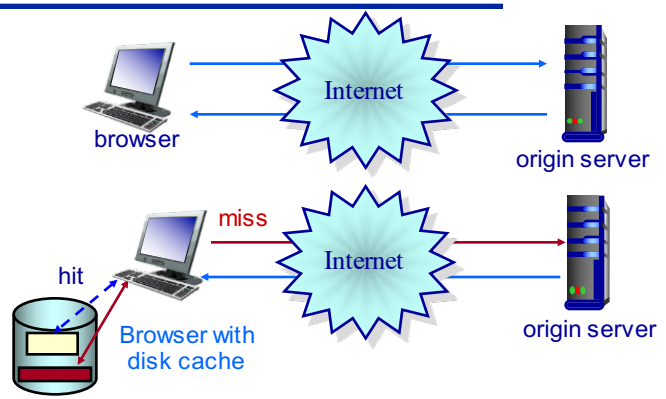
Persistent *v.* non-persistent connections

- Non-persistent
  - » HTTP/1.0
  - » Server parses request, responds, and closes TCP connection
  - » At least 2 RTTs to fetch every object
- Persistent
  - » Default for HTTP/1.1 (negotiable in 1.0)
  - » Client sends requests for multiple objects on one TCP connection
  - » Server parses request, responds, parses next request, responds...
  - » Fewer RTTs
- Parallel vs. persistent connections?
- What is my browser doing?
  - » Chrome –> Inspect –> Network –> Waterfall
  - » Wireshark

Parallel is better is your content is coming from different servers

HTTP User-Server Interaction

Browser caches



- Browsers cache content from servers to avoid future server interactions to retrieve the same content
- Caching-related issues?

Cache replacement policy: if storage is limited what do I do?

HTTP User-Server Interaction

The conditional GET

- If object in browser cache is "fresh," the server won't re-send it
  - » Browsers save current date along with object in cache

Client Server

Conditional get: when you send you're get request you also ask the server to send a copy of the thing if it is newer than what you have

Cache control: max-age = 120

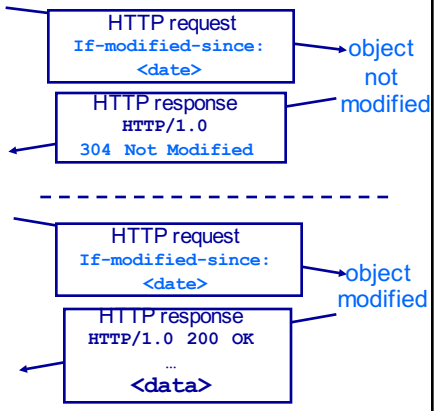
HTTP User-Server Interaction

The conditional GET

- If object in browser cache is "fresh," the server won't re-send it
  - » Browsers save current date along with object in cache
- Client specifies the date of cached copy in HTTP request

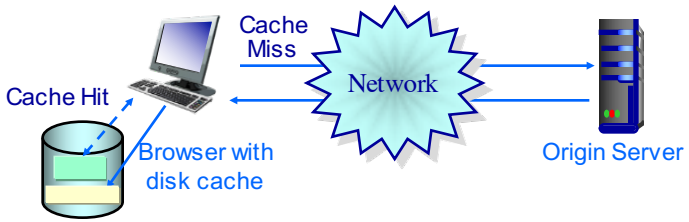
  `If-modified-since:<date>`
- Server's response contains the object only if it has been changed since the cached date
- Otherwise server returns:

  `HTTP/1.0 304 Not Modified`



HTTP User-Server Interaction

Cache Performance for HTTP Requests



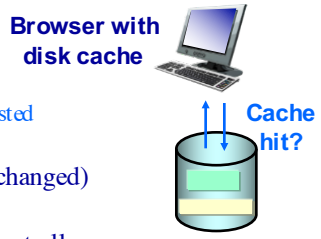
- What is the average time to retrieve a web object?
  - » $T_{mean} = hit\ ratio \times T_{cache} + (1 - hit\ ratio) \times T_{server}$ where *hit ratio* is the fraction of objects found in the cache
  - » Mean access time from a disk cache =
  - » Mean access time from the origin server =
- For a 60% hit ratio, the mean client access time is:
  - » $(0.6 \times 10\ ms) + (0.4 \times 1{,}000\ ms) = 406\ ms$

hit ratio: just have to go to cache to ge tthe object
1- hit ratio: have to go all the way to the origin server to get the info

Cache Performance for HTTP Requests

What determines the hit ratio?

- Cache size
- *Locality* of references
  - » How often the same web object is requested
- How long objects remain "fresh" (unchanged)
- Object references that can't be cached at all
  - » Dynamically generated content
  - » Protected content
  - » Content purchased for each use
  - » Advertisements ("pay-per-click" issues)
  - » Content that must always be up-to-date

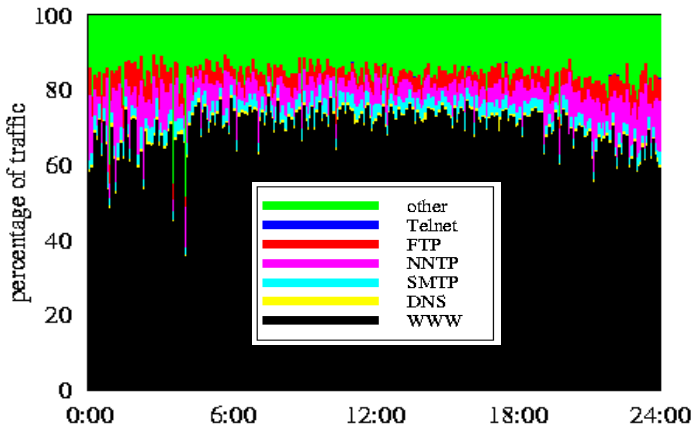


If your cache is small then your hit ratio wont be good because you will constantly be throwing things out of your cache

Content providers don't like when you cache because they lose control and cant host ads every time you access a resource

The Impact of Web Traffic on the Internet

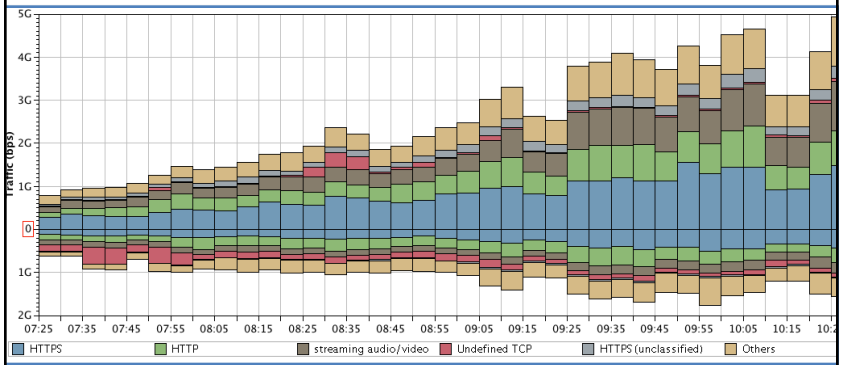
MCI backbone traffic in bytes by protocol (1998)



16

Traffic Makeup on UNC Link

Inbound traffic (2016)



Incoming

Outgoing

- Note the dominance of HTTPS over HTTP
- Also note that "streaming" excludes streaming done over HTTP

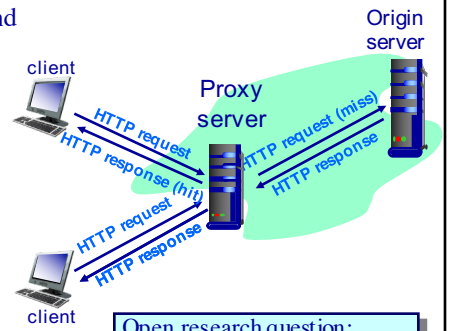
17

Text

Caching on the Web

Web caches (Proxy servers)

- Web caches are used to satisfy client requests without contacting the origin server
- Users configure browsers to send all requests through a shared *proxy* server
  - » Proxy server is a large cache of web objects
- Browsers send all HTTP requests to proxy
  - » If object in cache, proxy returns object in HTTP response
  - » Else proxy requests object from origin server, then returns it in HTTP response to browser



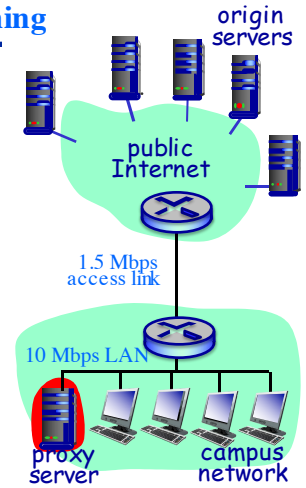
Open research question:
How does the proxy hit ratio change with the population of users sharing it?

18

Why do Proxy Caching?

The performance implications of caching

- Consider a cache that is "close" to client
  - » *e.g.*, on the same LAN
- Nearby caches help with:
  - » Smaller response times
  - » Decreased traffic on egress link to institutional ISP (often the primary bottleneck)



To improve Web response times should one buy:
- a 10 Mbps access link?
- or a proxy server?

19

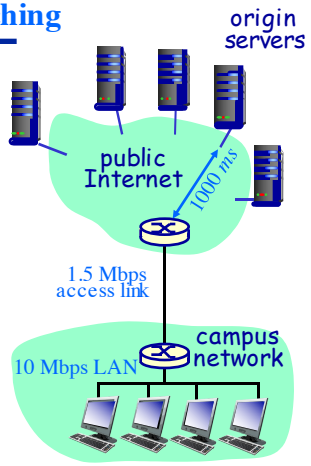
Proxy cache: basically a cache but just not on your machine

Beneficial because if this is within an organization, people will have similar interests and

Why do Proxy Caching?

The performance implications of caching

- Web performance without caching:
  - » Mean object size = 50 Kbits
  - » Mean request rate = 29/*sec*
  - » Mean origin server access time = 1 *sec*
  - » Average response time = ??



20

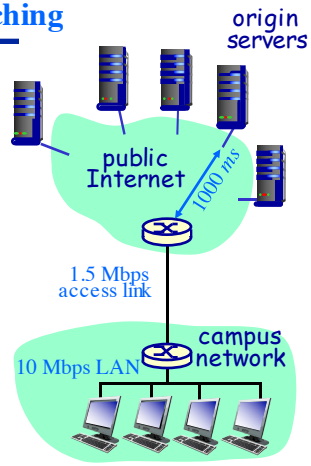
Why do Proxy Caching?

The performance implications of caching

- Web performance without caching:
  - » Mean object size = 50 Kbits
  - » Mean request rate = 29/*sec*
  - » Mean origin server access time = 1 *sec*
  - » Average response time = ??

- Traffic intensity on the access link:

$$29 \frac{reqs}{sec} \times \frac{50 \text{ Kbits}/req}{1.5 \text{ Mbps}} = 0.97$$



23

Traffic intensity (amount of traffic coming onto the link divided by the capacity of the link) = 29*(50/1500)=
.9666667

I = traffic load / transmission capacity = ((50*10^3) * 29) / (1.5*10^6) = .97

Queuing delay = I / (1 - I) = .97 / (1 - .97) = 32

Average queuing delay = (32*(50*10^3))/(1.5*10^6) =

if you upgrade the link to 10mbps then the queuing delay is .145

Why do Proxy Caching?

The performance implications of caching

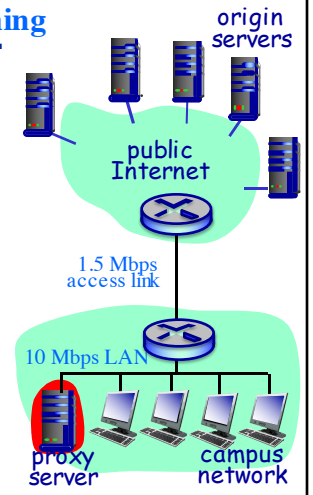
- Upgrade the access link to 10 Mb/s
  - » Response time = ??
  - » Queuing is negligible hence response time = 1 *sec (+ 10 ms)*
- Add a proxy cache with 40% hit ratio and 10 *ms* access time
  - » Response time = ??
  - » Traffic intensity on access link =

$$0.6 \times 0.97 = 0.58$$

  - » Response time = (1000+1.4*33+33+5+10)

$$0.4 \times 10\ ms + 0.6 \times 1{,}089\ ms = 653\ ms$$

- A proxy cache lowers response time, lowers access link utilization, and saves money!



(50*(10^3)*29) / (10*(10^6))

Response time = hit ratio * Tproxy + (1-hit ratio) * Tserver =
h = 40%, Tproxy = 10ms

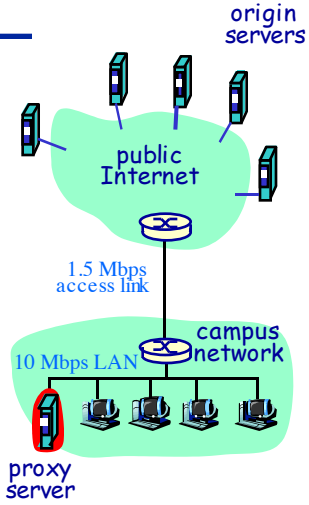
Traffic Intensity = traffic load / (1.5*(10^6)) = (29*(50*10^3)*.6) / (1.5*10^6)=.58

queuing delay = .58/.42

Why do Proxy Caching?

The case for proxy caching

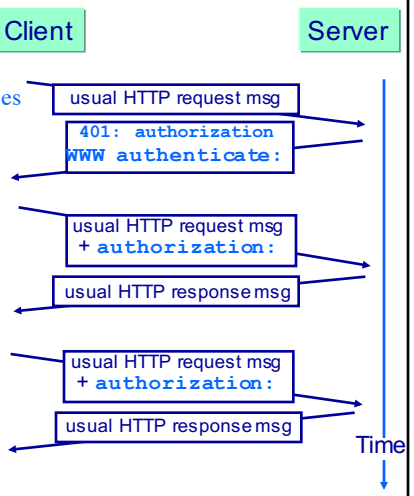
- Lower latency for user's web requests
- Reduced traffic at all network levels
- Reduced load on servers
- Some level of fault tolerance (network, servers)
- Reduced costs to ISPs, content providers, *etc.*, as web usage continues to grow exponentially
- More rapid distribution of content



3 levels of caching
browser cache
proxy cache
content cache or content distribution network

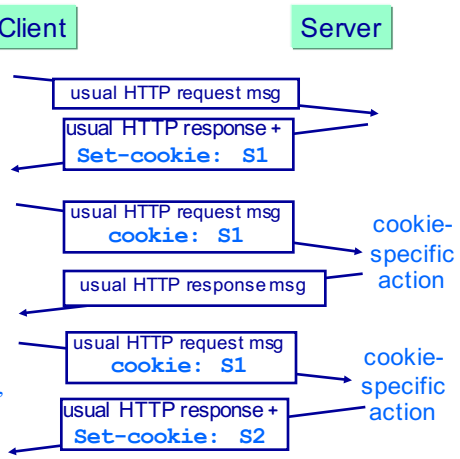
HTTP User-Server Interaction
Authentication

- Problem: How to limit access to server documents?
  - » Servers provide a means to require users to authenticate themselves
- HTTP includes a header tag for user to specify name and password (on a GET request)
  - » If no authorization presented, server refuses access, sends **`WWW authenticate:`** header line in response
- Stateless: client must send authorization for each request
  - » A stateless design
  - » (But browser may cache credentials)



HTTP User-Server Interaction
Cookies

- Server sends "cookie" to browser in response message
  `Set-cookie: <value>`
- Browser presents cookie in later requests to same server
  `cookie: <value>`
- Server matches cookie with server-stored information
  - » Provides authentication
  - » Client-side state maintenance (remembering user preferences, previous choices, …)
- Chrome: Inspect –> Application –> Cookies



Browser is stateless

At the basic level protocols are only transferring files

HTTP User-Server Interaction

Anatomy of a cookie

- Invented (and patented!) by Netscape

```
Set-cookie:
  <name-value-pair>;
  <name-value-pair>;…
```

- Cookie fields…
  - » Name: an opaque data type
  - » Expiration date: day/year/time
  - » Domain: Set of servers to which cookie can be sent
  - » Path: Subset of directories in the DOMAIN for which cookie is valid
  - » Secure: send over HTTPS