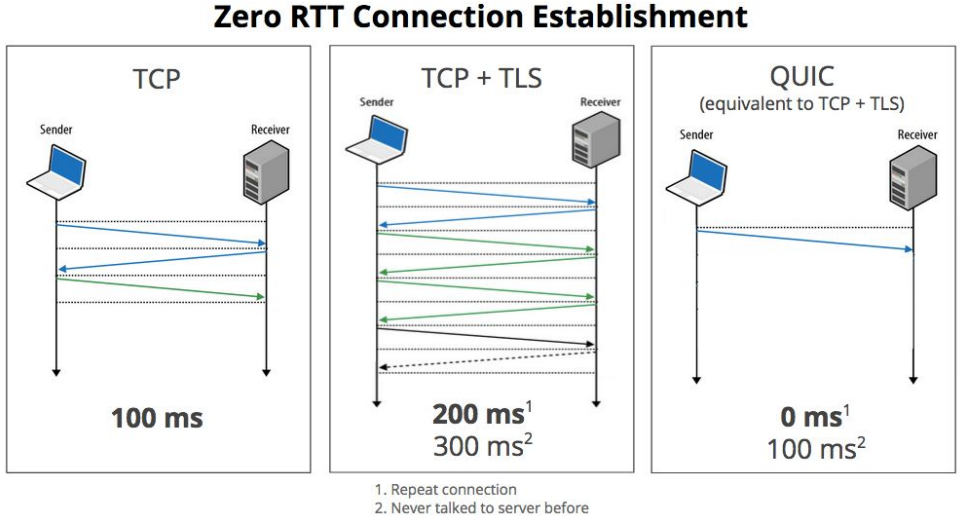
Computer Networks

HTTP and more

Jianping Pan
Fall 2022



Last lecture

- Web, URL, HTML and HTTP basics
  - the application-layer protocol for the Web
  - following the client-server model
  - (stateless) request-reply transaction
    - HTTP request: GET / HTTP/1.0
    - HTTP response: HTTP/1.0 200 OK
  - the service expected from lower layers
    - reliable data transfer
    - normally by TCP

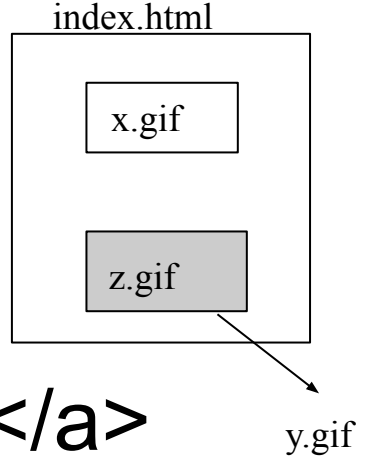
Today's topics

- HTTP: advanced topics
  - fit better on TCP
    - improve HTTP efficiency
  - become stateful
    - server and client can know/remember each other
  - deal with scalability (may be left for Friday)
    - web caching and content delivery

Web browsing examples

- http://www.a.com/index.html
  <html>
  <img src="http://www.a.com/x.gif">

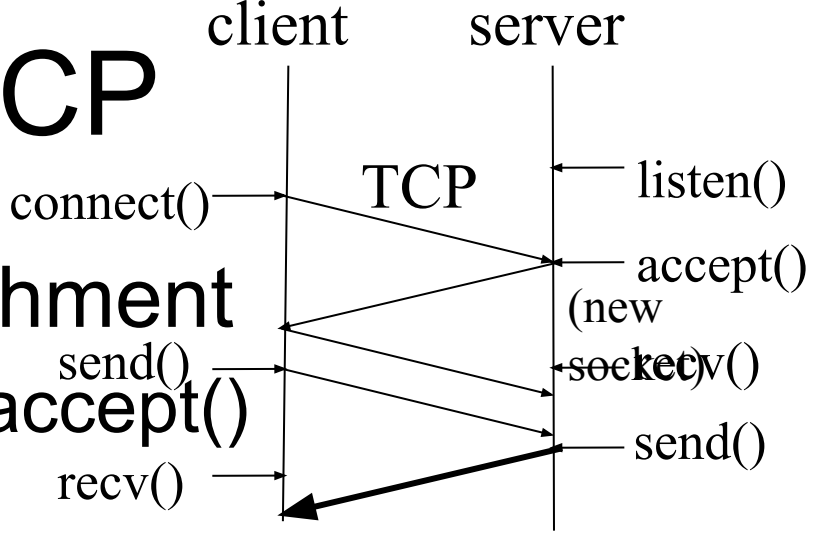
  <img src="http://www.a.com/z.gif">
  </html>



- In your favorite web browser
  - URL: http://www.a.com
  - Q: how many HTTP requests?

discussion

HTTP/TCP



- TCP connection establishment
  - client: connect(); server: accept()
- HTTP transaction
  - request: client: send(); server: recv()
  - response: server: send(); client: recv()
- TCP connection release
  - server: close(); client: close()
- Client is to retrieve the embedded objects

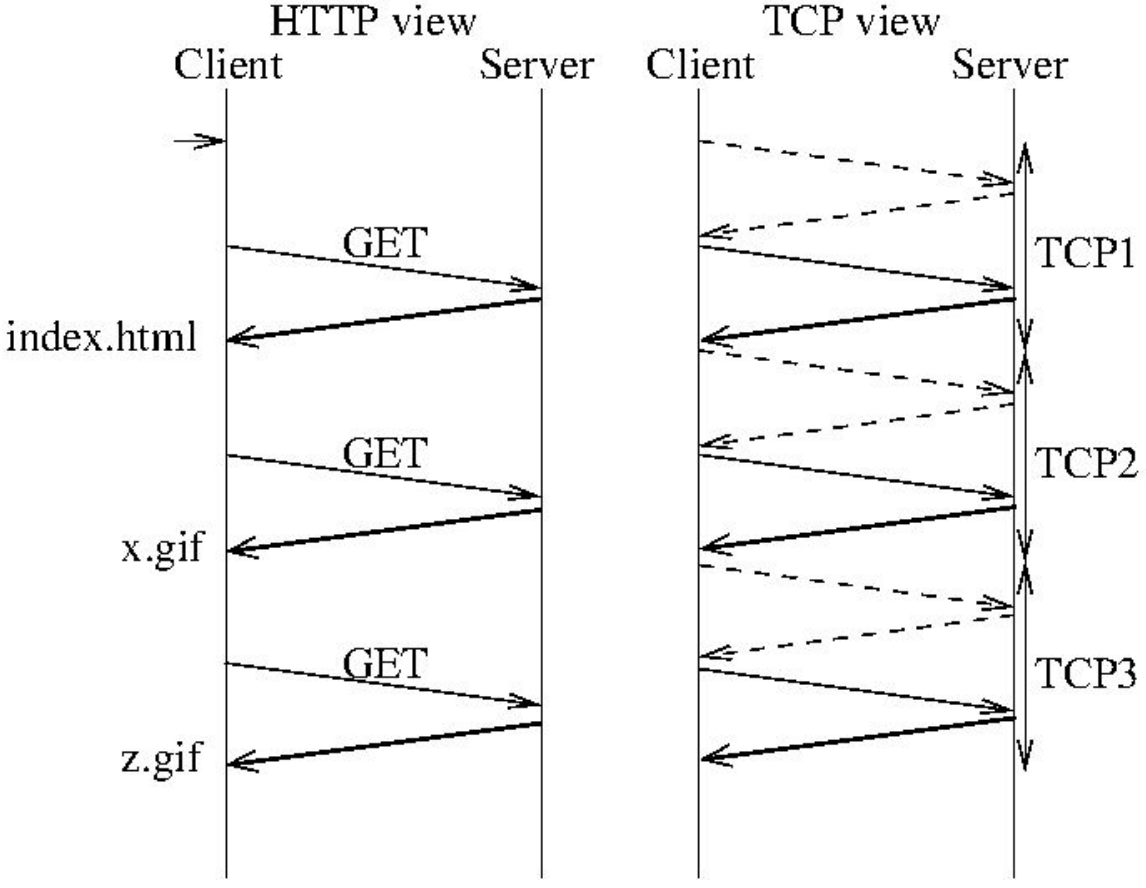
Q: round-trips?

Non-persistent HTTP

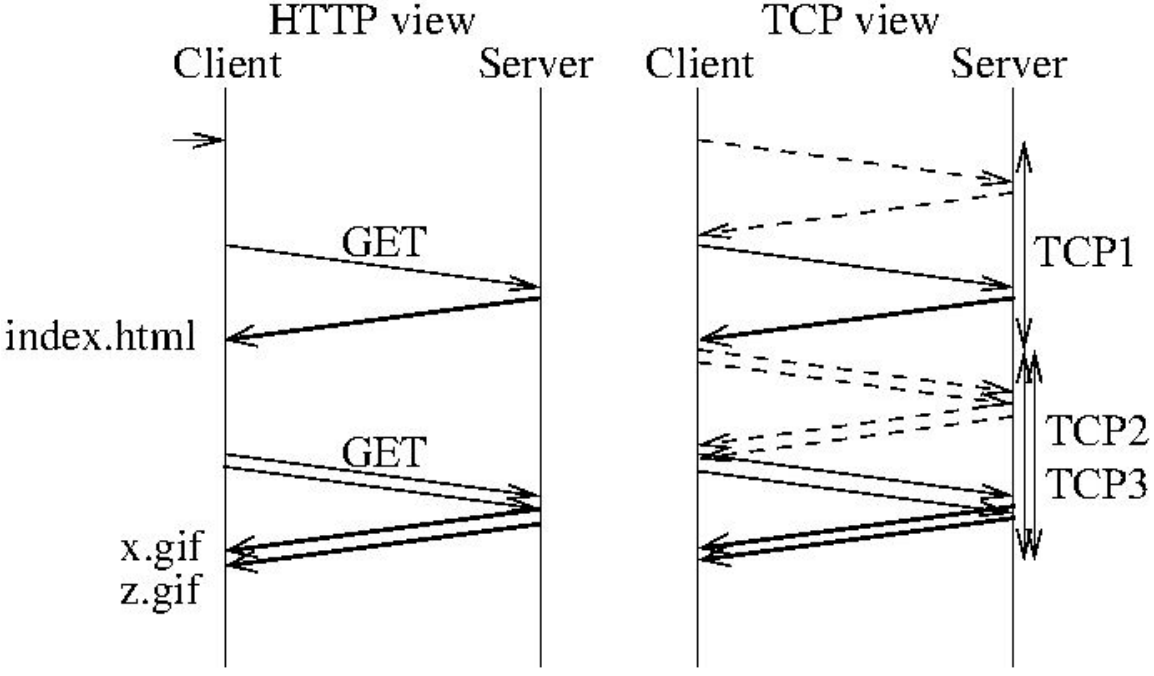
- One object per TCP connection
  - default behavior in HTTP/1.0
  - network cost: ~2*RTT per object
  - end-host cost: 1 socket() for each object
- Performance improvement
  - parallel/concurrent connections
    - e.g., an HTML page with 2 embedded objects

Q: cost reduced?

Figure on next slide


HTTP view
Client
Server
GET
index.html
GET
x.gif
GET
z.gif
TCP view
Client
Server
TCP1
TCP2
TCP3

Non-persistent concurrent HTTP



Persistent HTTP

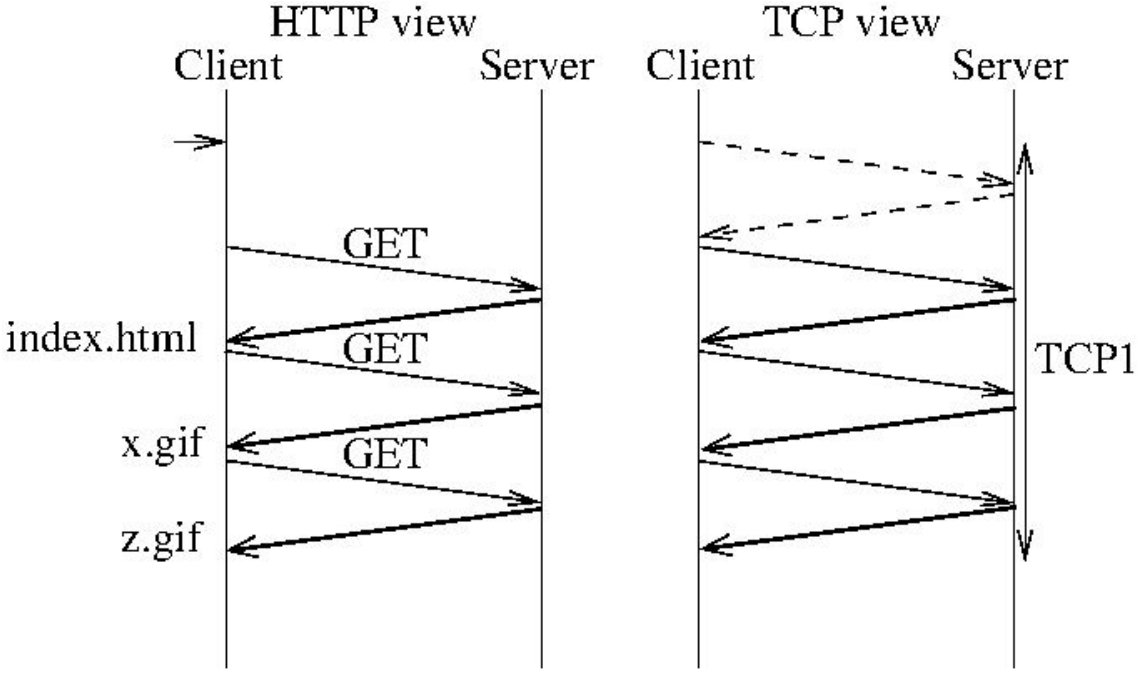
- Multiple objects through a TCP connection
  - between the **same** client/server
  - default behavior in HTTP/1.1
    - for HTTP/1.0: Connection: Keep-Alive
  - network cost: ~RTT per object for many objects
  - to disable/close: Connection: Close
    - client/server Keep-Alive timeout

Q: pros and cons?

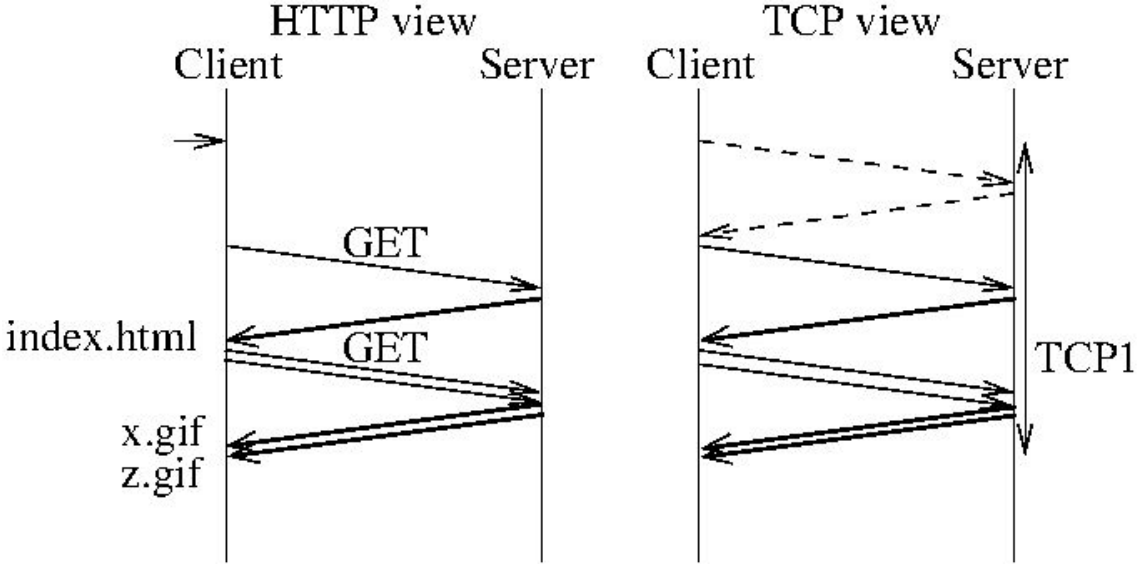
- Performance improvement: pipelining
  - ~2*RTT for all objects from the same server

Figure on next slide

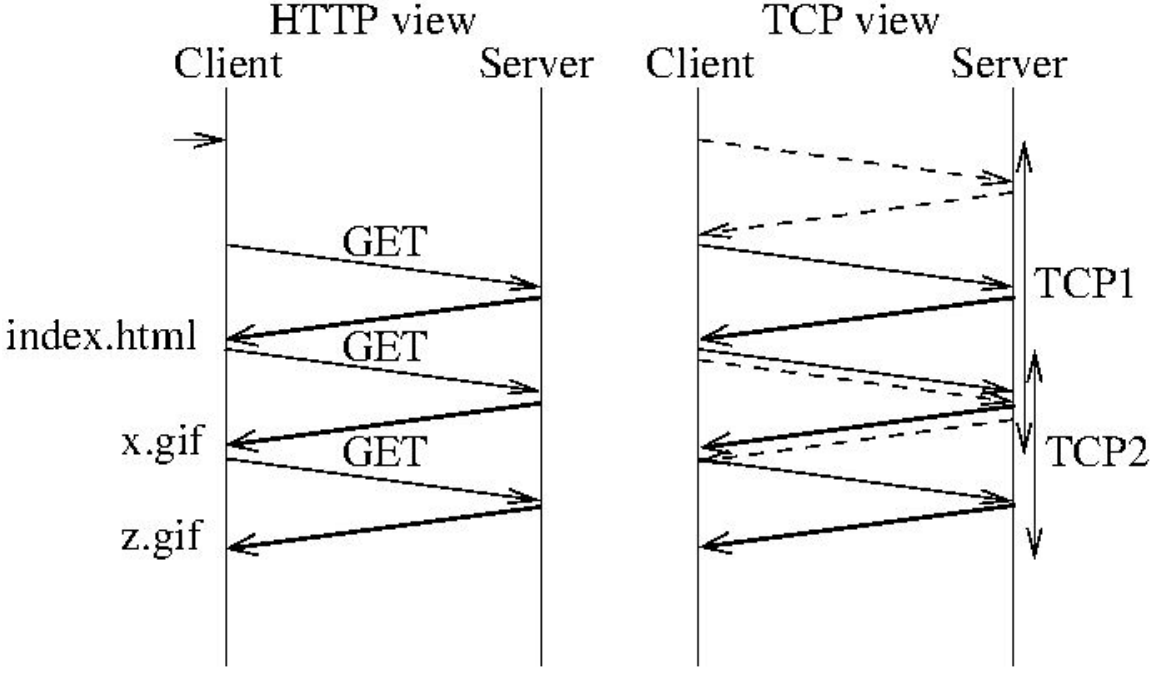
Persistent HTTP



Persistent HTTP + Pipelining



Persistent + concurrent HTTP

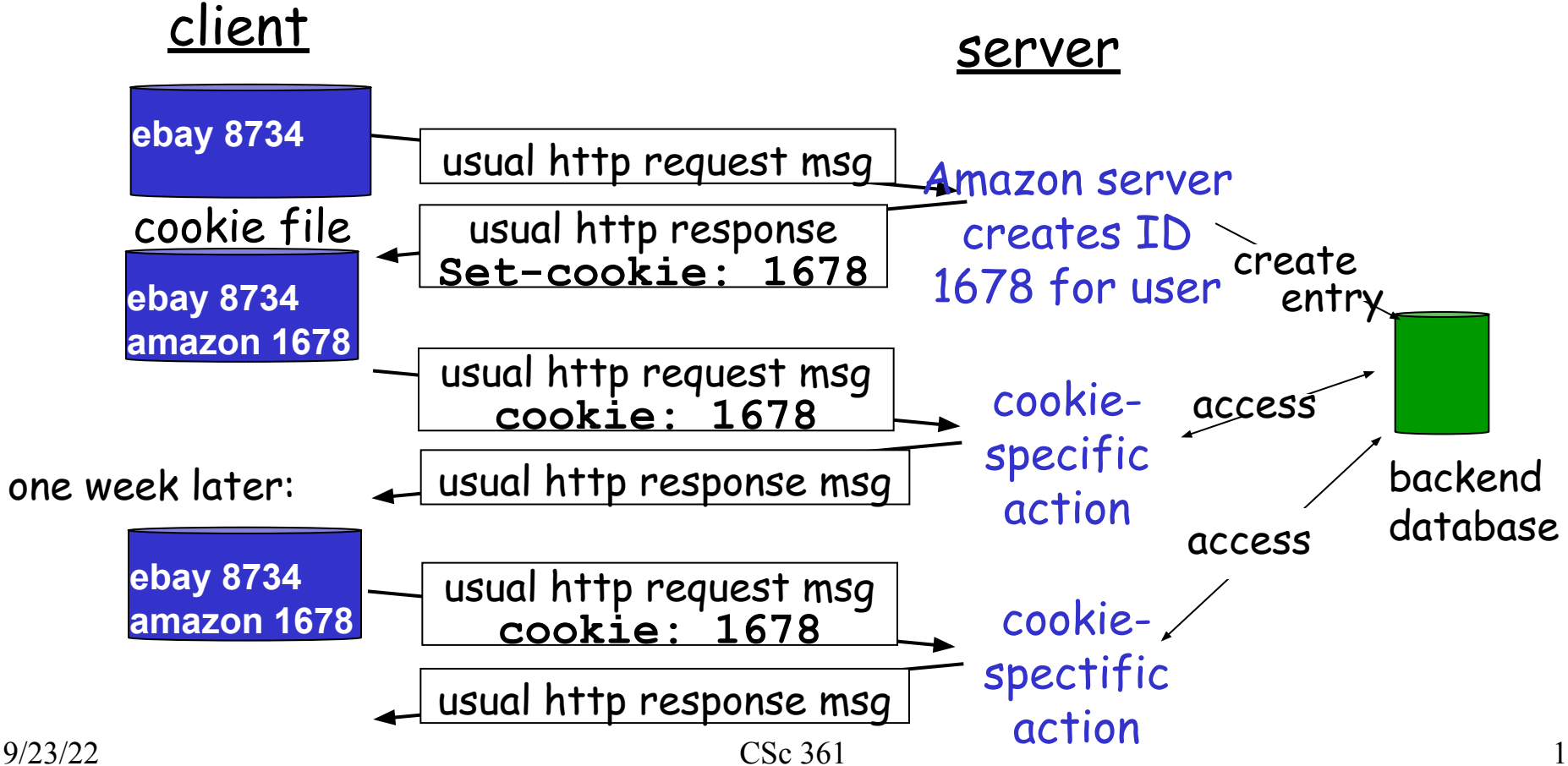


Client-server states

- HTTP itself is stateless
  - request-reply **transactions**
- Many applications require states
  - cookie issued by server's *backend* servers
    - HTTP response header: Set-Cookie
  - client can *choose* to keep cookie
  - client presents cookie in subsequent requests
    - HTTP request header: Cookie

* GDPR-prompted "cookie consent" campaigns?

HTTP cookies: an example

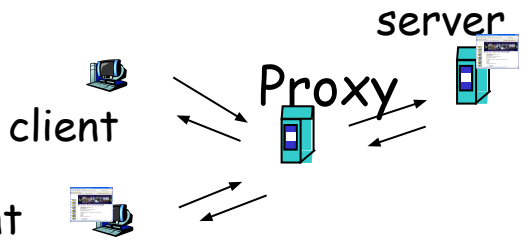


Tracking client?!

- Between web servers
  - HTTP request header: Referer *
  - Referrer!
  - referrer spamming and referrer spoofing
  - de-referring
- User security and privacy
  - e.g., cookie theft, cookie poisoning, web bug
  - browsing anonymizing

* typo for referrer; recall creat() in CSc360?

Web caching



- Scalability issues with the client-server model
  - one server, many clients, concurrent requests
  - server load
  - network traffic
- Web caching: aggregate user requests
  - by caching responses to previous requests
  - explore locality: same requests may occur soon!
  - reduce response time and traffic load

Example on the slide

Consistency control

- Objects retrieved from the cache
  - may be stale due to updates at origin servers
- Strong consistency
  - HTTP request header: If-Modified-Since
  - HTTP response: HTTP/1.0 304 Not Modified
    - reduce traffic load if hit
- Weak consistency

Q: pros and cons?

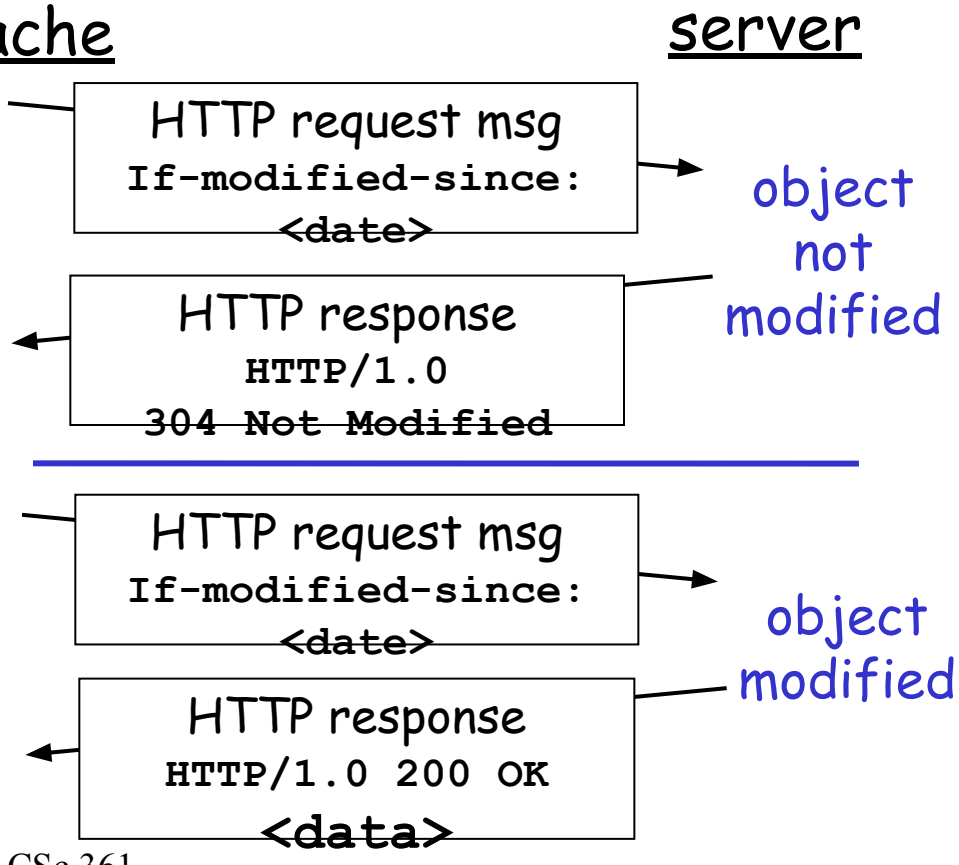
  - time-to-live (TTL)
    - reduce traffic load and response time if “hit”

Conditional GET

- Goal: don't send object if cache has up-to-date cached version
- 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`



Content delivery

- Move content closer to end users
  - content distribution
- Redirect users to closer servers
  - information retrieval
  - how do they do that?
    - e.g., Akamai
    - DNS-based server selection
      - more after DNS lectures

- HTTP evolution
  - Original HTTP: only the GET method for HTML
  - HTTP/0.9 (1991): headers and additional methods
  - HTTP/1.0 (1996): performance improvement
    - **Persistent connection (Keep-Alive) becomes optional**
  - HTTP/1.1 (1997/1999): most widely used now
    - Persistent connection becomes the default
  - HTTP/2 (RFC7540, May 2015)*
    - Based on Google's SPDY
  - HTTP/3 (RFC9000, May 2021)
- More features and how to fit better with TCP

* https://http2.github.io/

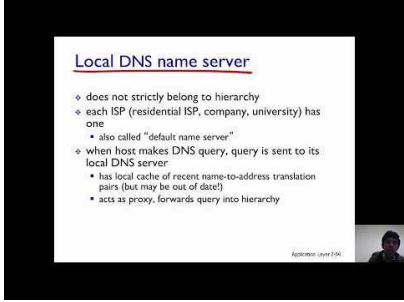
- Google's SPDY
  - A basis for HTTP/2
  - Further reduce the page load time (PLT)
  - Prioritizing and multiplexing embedded objects
    - One connection per client to the same server
    - More flexible than HTTP pipelining (HOL blocking)
  - Encryption (TLS) and compression (Gzip)
  - Server can hint or push content/update
  - Used at Google, Facebook, Twitter, etc
- Supported in Chrome, Mozilla, Opera, etc

* What's latest? QUIC vs HTTP/3? Talk after TCP too Similarly, Microsoft's SM (speed+mobility)

This lecture

- HTTP
  - (non-)persistence, pipelining, cookies, referrers
  - web caching and content delivery

- Explore further
  - How do your favorite web browsers and servers support advanced HTTP features?
  - How's your your W1 and P1?



Next lecture

- DNS (**KR7S2.4***)
  - DNS server hierarchy
  - DNS resolution process
  - DNS-based server selection

* Kurose&Ross, 7th Edition, Chapter 2, Section 2.4