National University of Computer & Emerging Sciences

CS 3001 - COMPUTER NETWORKS

Lecture 08

Chapter 2

15th September, 2022

Nauman Moazzam Hayat

nauman.moazzam@lhr.nu.edu.pk

Office Hours: 02:30 pm till 06:00 pm (Every Tuesday & Thursday)

HTTP is *Stateless*

- Each request-response treated independently
  - Servers *not* required to retain state
- **Good**: Improves scalability on the server-side
  - Failure handling is easier
  - Can handle higher rate of requests
  - Order of requests doesn’t matter
- **Bad**: Some applications need persistent state
  - Need to uniquely identify user or store temporary info
  - *e.g.*, Shopping cart, user profiles, usage tracking, …

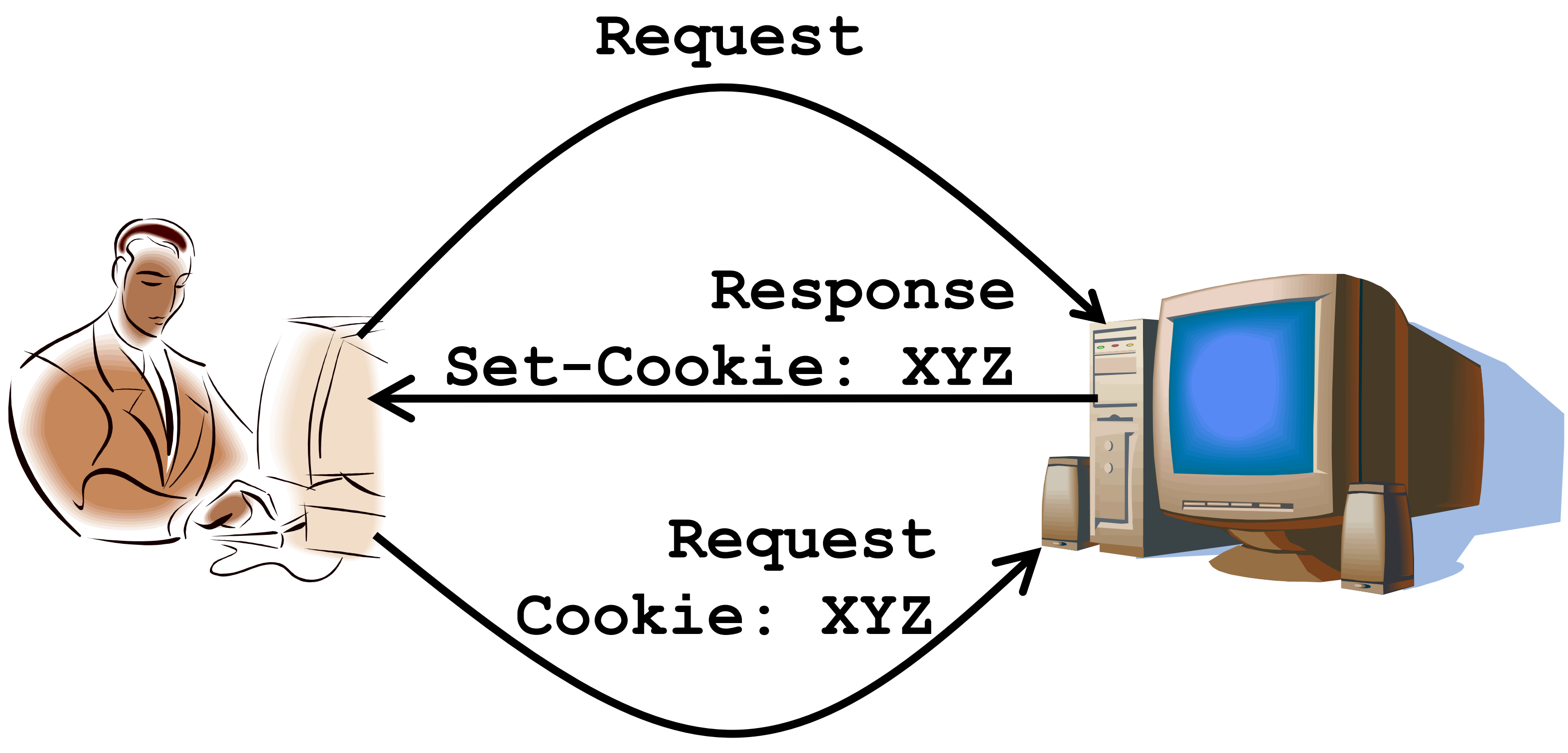
Question

- How does a stateless protocol keep state?

Cookies

State in a Stateless Protocol: Cookies

- *Client-side* state maintenance
  - Client stores small state on behalf of server
  - Client sends state in future requests to the server
- Can provide authentication



User-server state: cookies

many Web sites use cookies

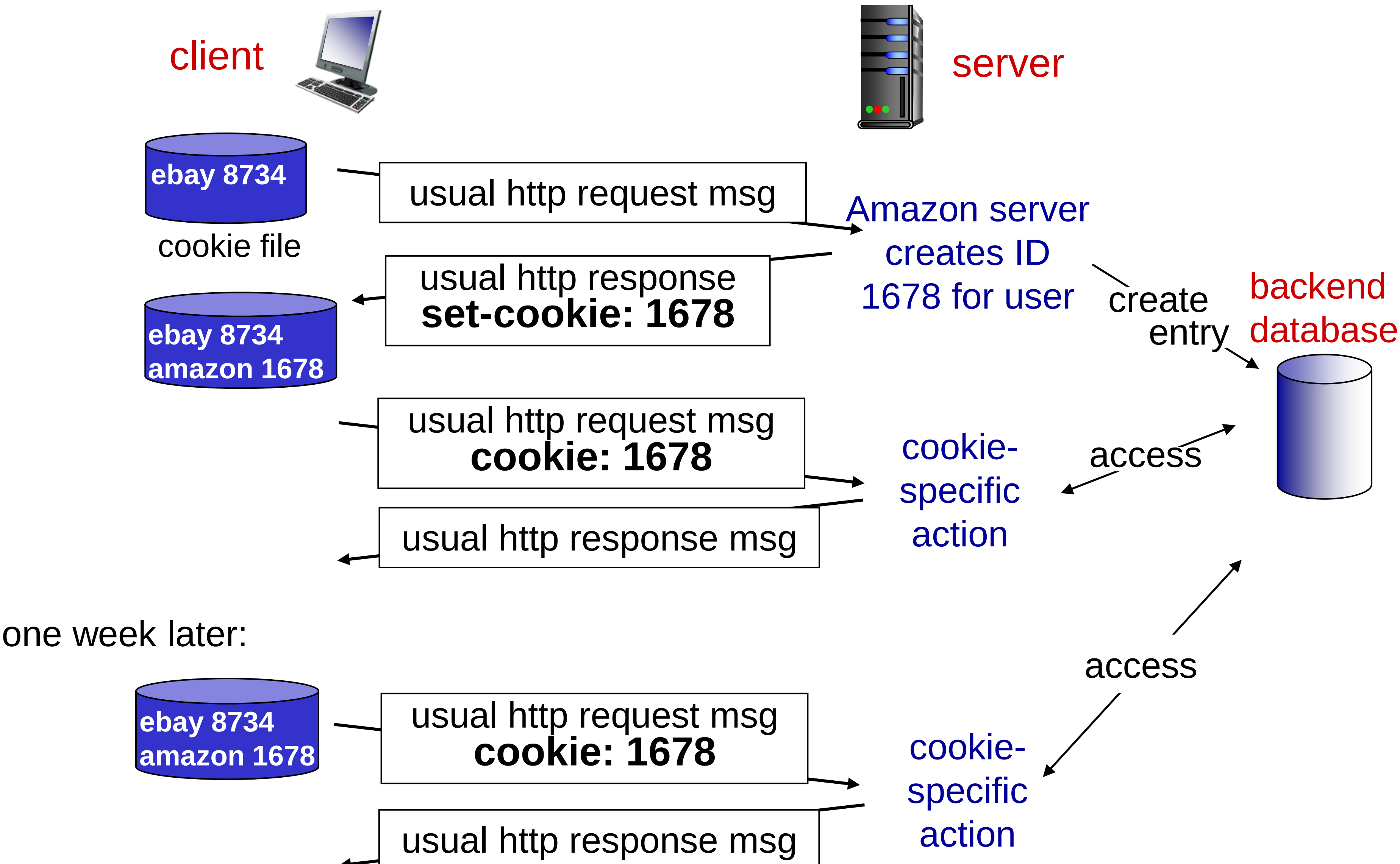
four components:

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 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)

aside

cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

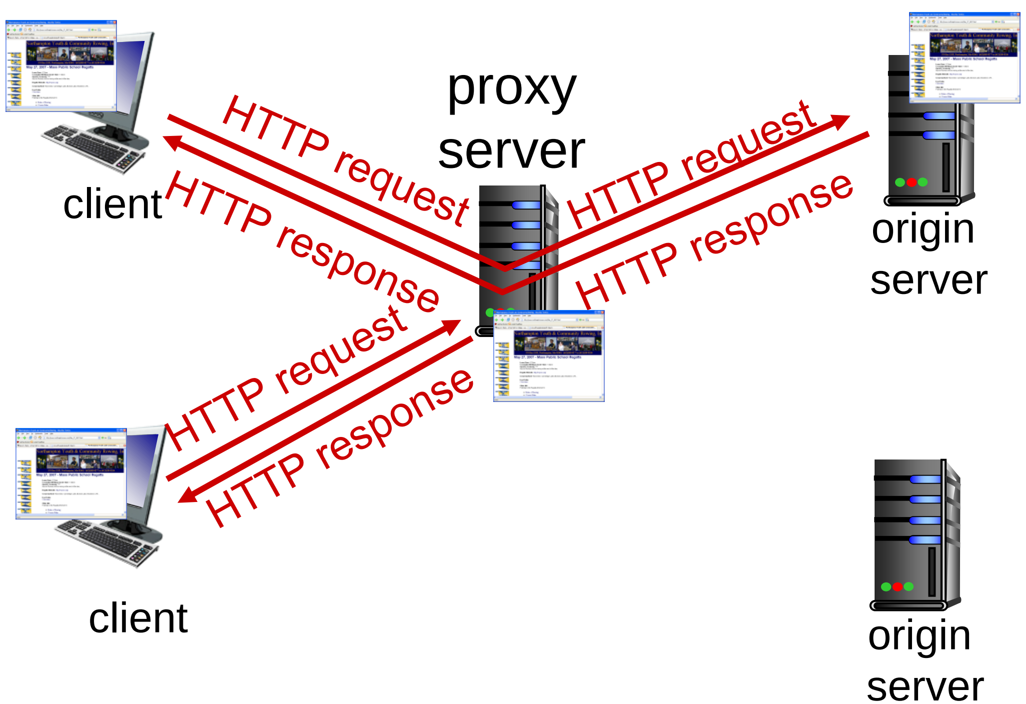
how to keep "state":

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

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:

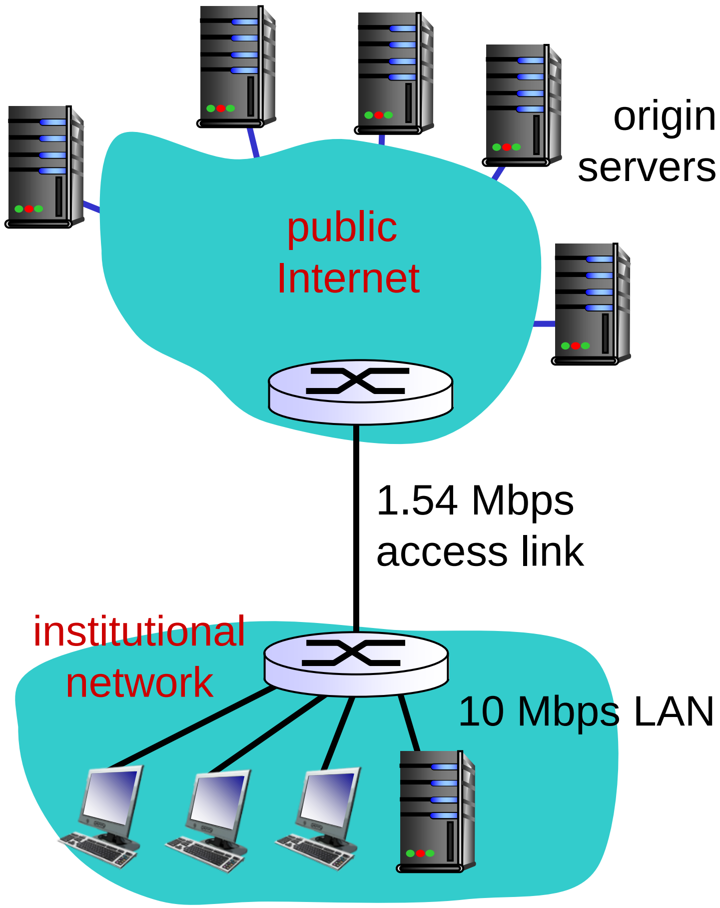
assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers:15 requests/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec (Internet Delay i.e. between Internet Router to local router)
- access link rate: 1.54 Mbps

consequences:

- LAN Utilization = 1.50 Mbps/10 Mbps = 15% Or
- LAN Traffic Intensity (La/R) = 0.15 (100kbit/request * 15 requests / sec **/** 10 Mbps)
- Thus negligible LAN delay (10s of milliseconds)
- Access Link Utilization = 1.50 Mbps / 1.54 Mbps = *97%* Or *problem!*
- Access Link Traffic Intensity = *0.97* (100kbit/request * 15 requests / sec **/** 1.54Mbps)
- Delay on access link (*Huge!*)
- total delay = Internet delay + access delay + LAN delay

  = 2 sec + *minutes* + μsecs



Note: *Utilization* of 97% is same as *Traffic Intensity* of .97

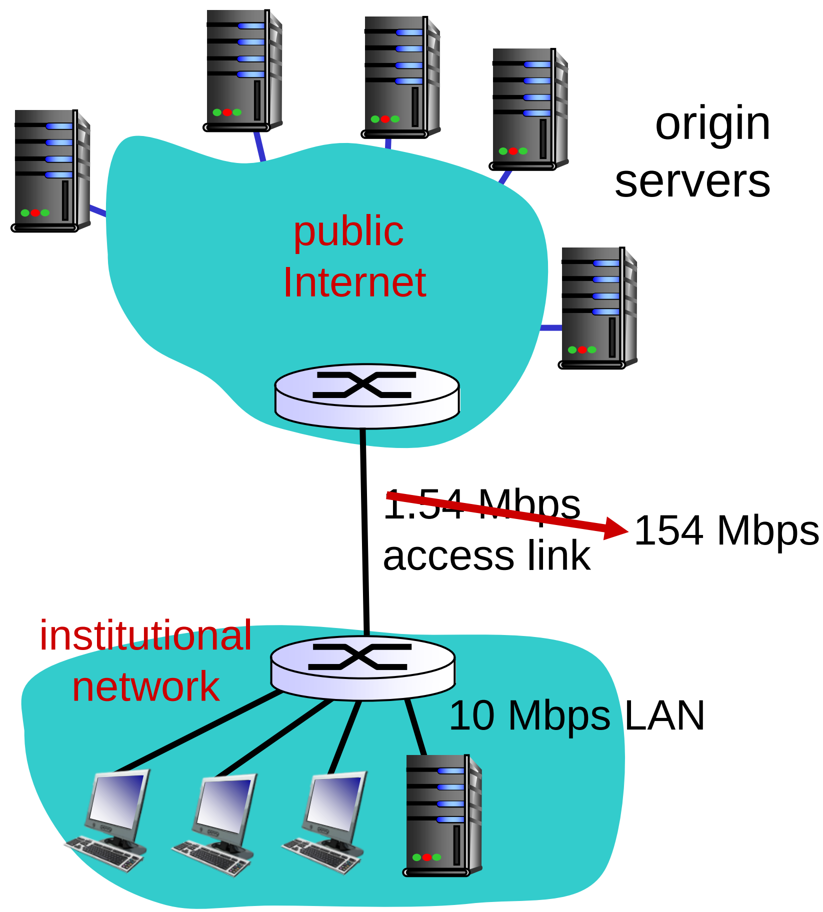
Caching example: fatter access link

assumptions:

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

consequences:

- LAN utilization: 15%
- access link utilization = 97% → 0.97%
- Or Access link Traffic Intensity = .0097
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + µsecs → msecs



Cost: increased access link speed (not cheap!)

Caching example: install local cache

assumptions:

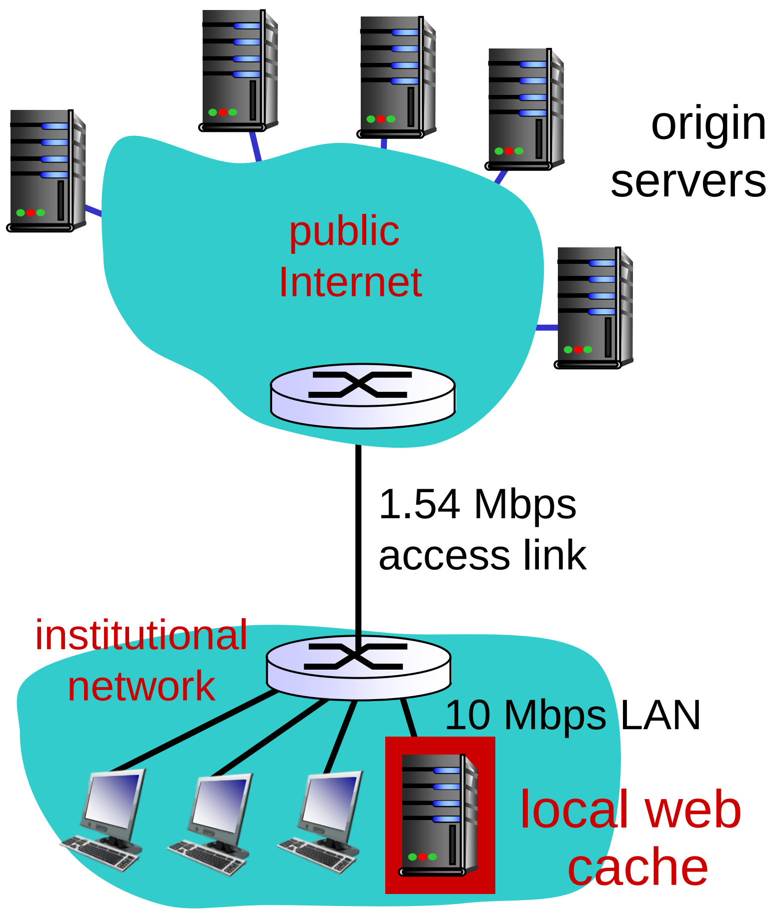
- avg object size: 100K bits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = ?
- total delay = ?

How to compute link utilization, delay?

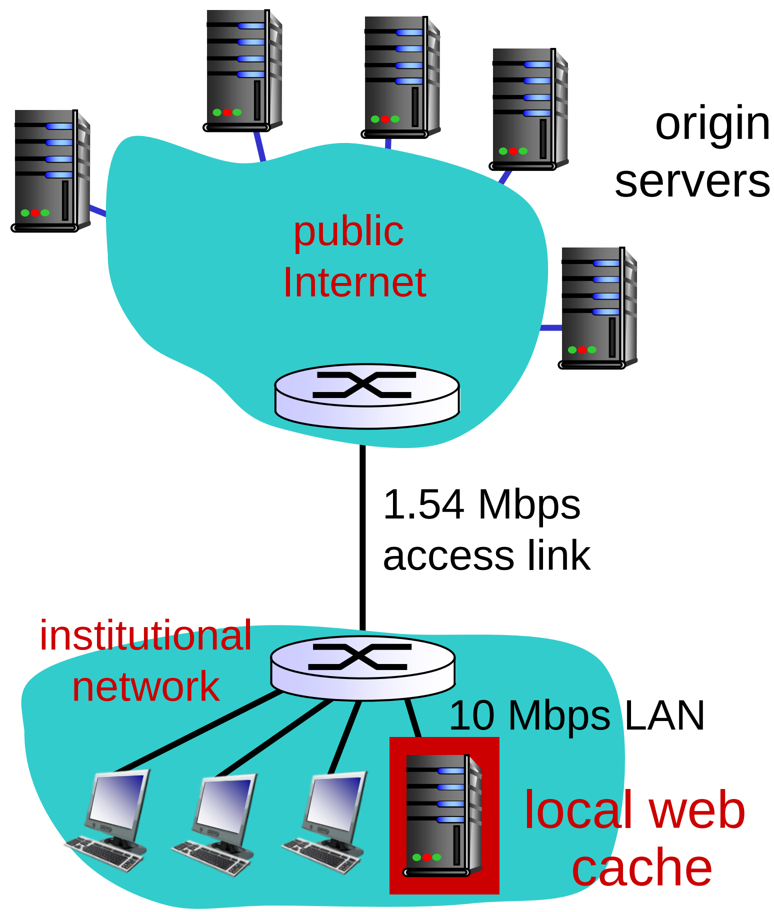
Cost: web cache (cheap!)



Caching example: install local cache

Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin
- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link = 0.6*1.50 Mbps = 0.9 Mbps
  - utilization = 0.9/1.54 = 0.58 = 58%
- total delay
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2 + ~msec for access link & LAN) + 0.4 (~ μsec for LAN)
  - = 0.6 (2.01) + 0.4 (~msec)
  - = ~ 1.2 sec
  - ***less than with 154 Mbps link (and cheaper too!)***

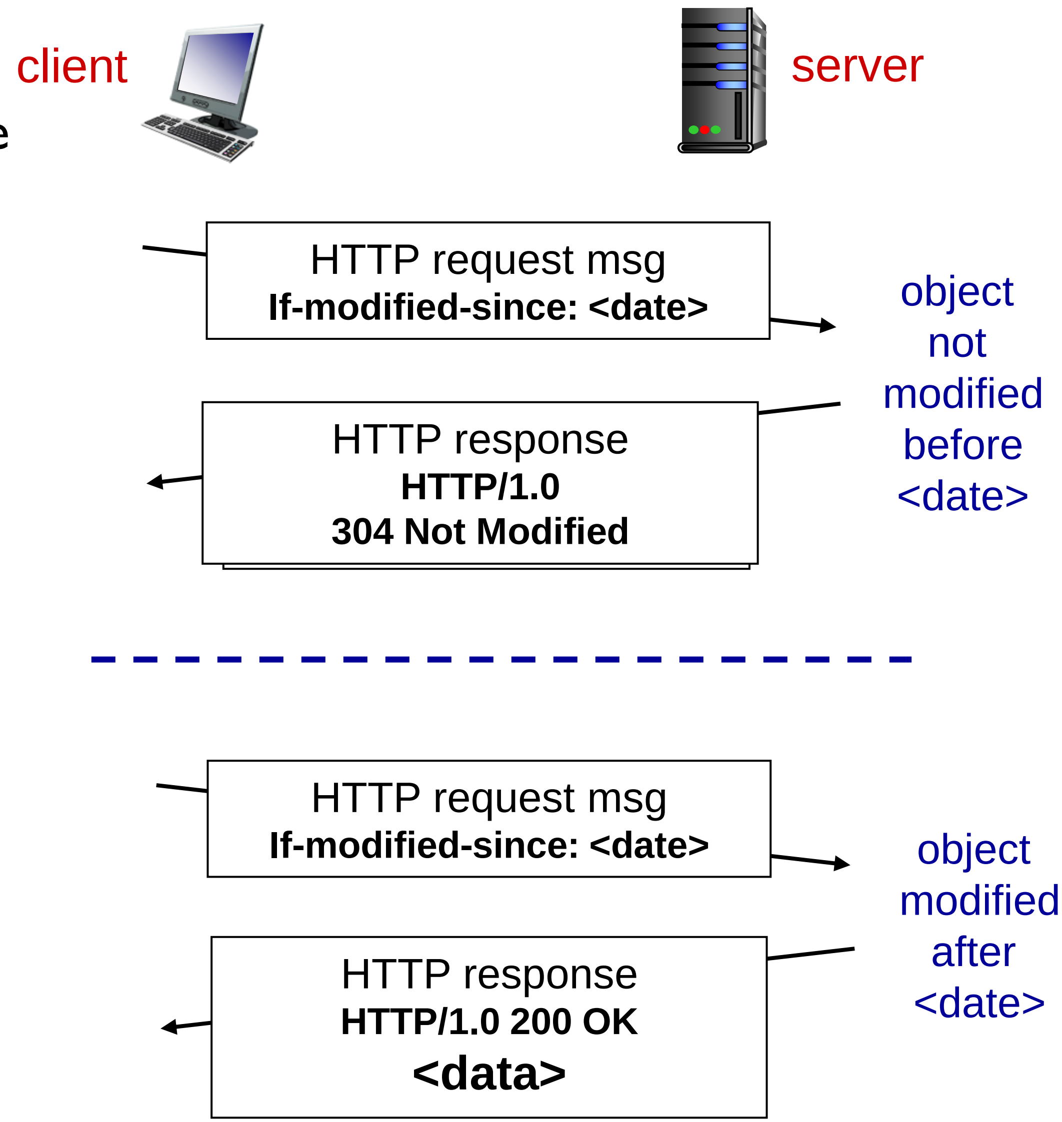


Problem

- The copy of the object in the web cache may be stale!!!

Conditional GET

- *Goal:* don't send object if cache has up-to-date cached version
  - no object transmission delay
  - lower link utilization
- *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`



Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

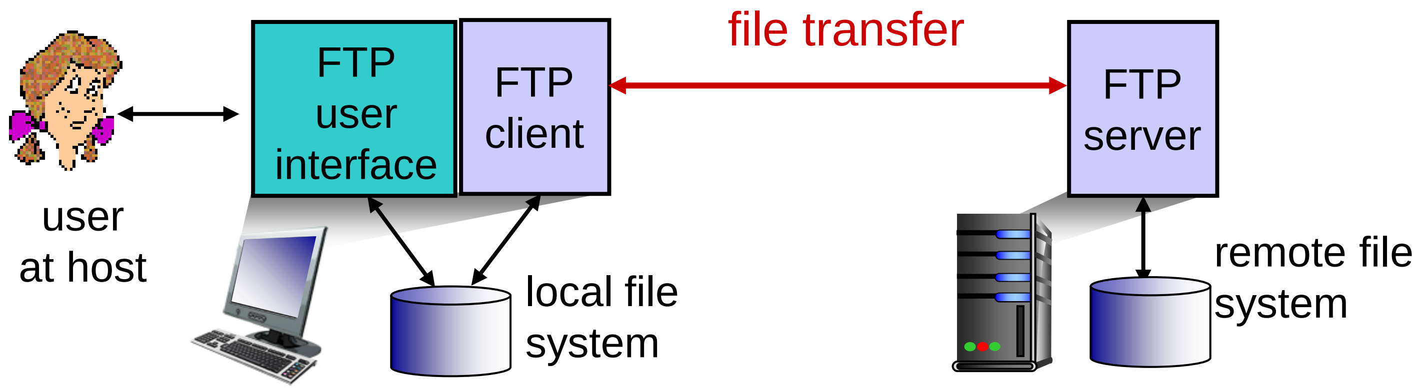
2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

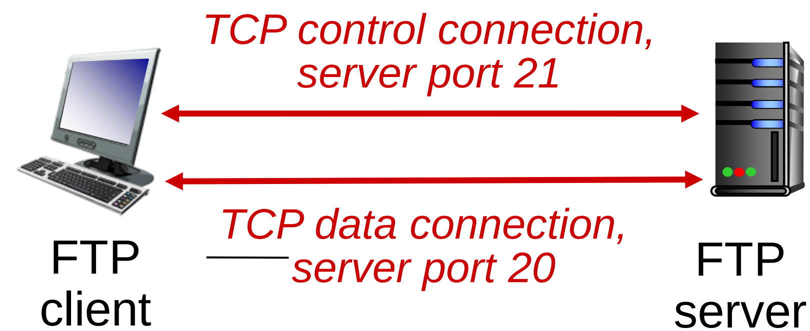
FTP: the file transfer protocol



- transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ftp: RFC 959

FTP: separate control, data connections

- FTP client contacts FTP server at port 21, using TCP
- client authorized over control connection
- client browses remote directory, sends commands over control connection
- when server receives file transfer command, *server* opens *2nd* TCP data connection (for file) *to* client
- after transferring one file, server closes data connection



- server opens another TCP data connection to transfer another file
- control connection: *"out of band"*
- FTP server maintains "state": current directory, earlier authentication

FTP commands, responses

sample commands:

- sent as ASCII text over control channel
- **`USER`** ***`username`***
- **`PASS`** ***`password`***
- **`LIST`** return list of file in current directory
- **`RETR filename`** retrieves (gets) file
- **`STOR filename`** stores (puts) file onto remote host

sample return codes

- status code and phrase (as in HTTP)
- **`331 Username OK, password required`**
- **`125 data connection already open; transfer starting`**
- **`425 Can't open data connection`**
- **`452 Error writing file`**

Quiz # 2 (Chapter - 2)

- *Quiz # 2 for Chapter 2 to be taken in the class on Thursday, 22nd September, 2022 during the lecture time*

- *Quiz to be taken for own section only*

No Retake

Be on time