CSC/CPE 138

Computer Network Fundamentals

Lecture 2_1: Application Layer

California State University, Sacramento
Fall 2024

Slide Courtesy: Computer Networking: A Top-Down Approach, Kurose Ross, 8th Edition

Update

- Quiz 1
  - Due – September 15
- Homework 1
  - Due – September 15

What is Lecture 2_1?

- Principles of network applications
  - Conceptual *and* implementation aspects of application-layer protocols
    - Network Application
    - client-server paradigm
    - peer-to-peer paradigm
    - Transport-layer service models
- Web and HTTP
  - Examining popular application-layer protocols and infrastructure , HTTP

Name Some Network Apps.

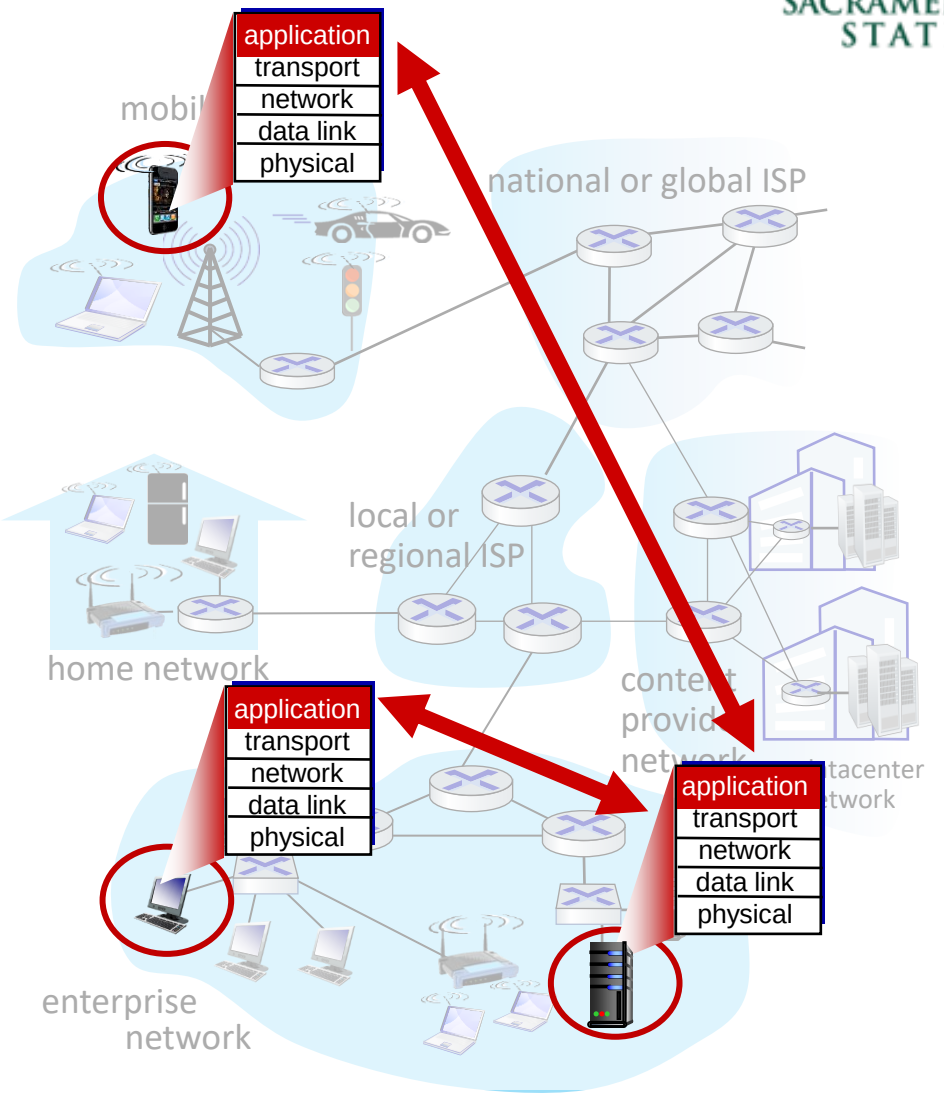
- Social networking
- Web
- e-mail
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)
- P2P file sharing
- Voice over IP (e.g., Skype)

Question:

How do you develop them?

Creating a network app

- Write programs that:
  - run on (different) end systems
  - communicate over network
  - e.g., web server software communicates with browser software
- No need to write software for network-core devices
- What architecture do you need?
  - Client-server or P2P?



Client-server Architectures

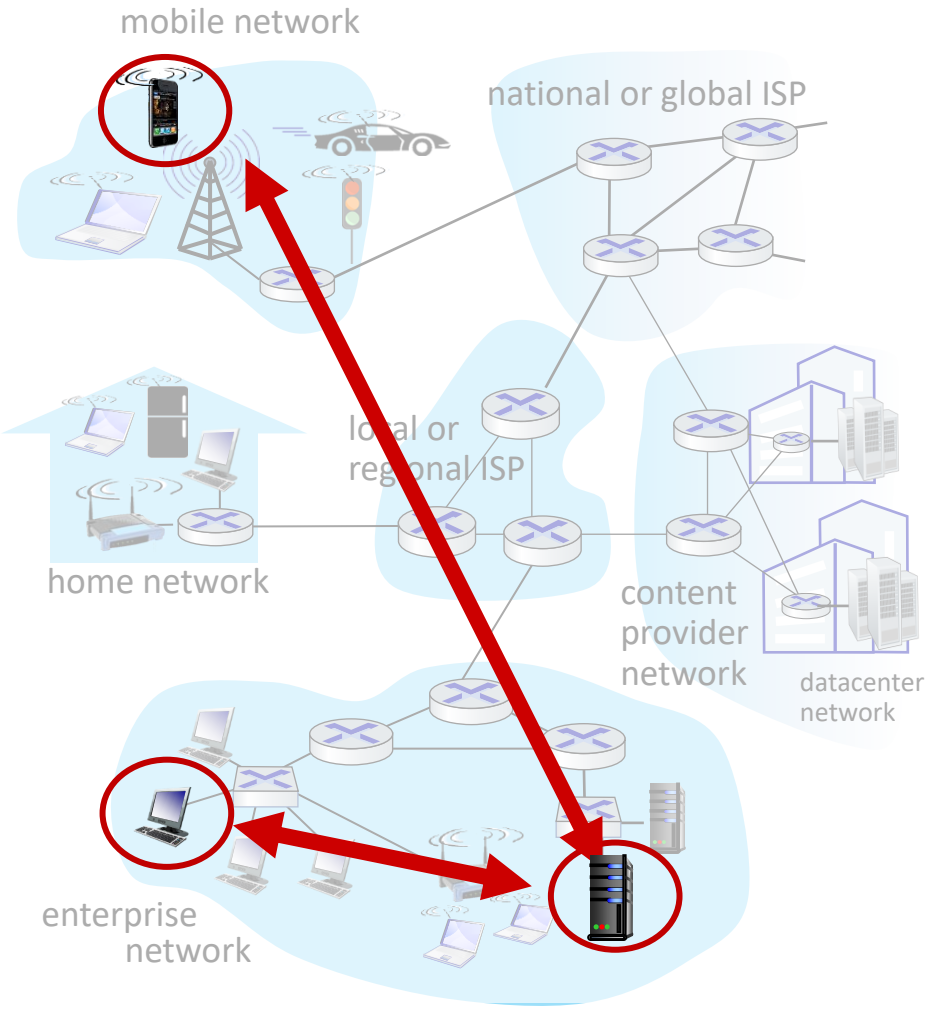
server:

- always-on host
- permanent IP address
- often in data centers, for scaling

clients:

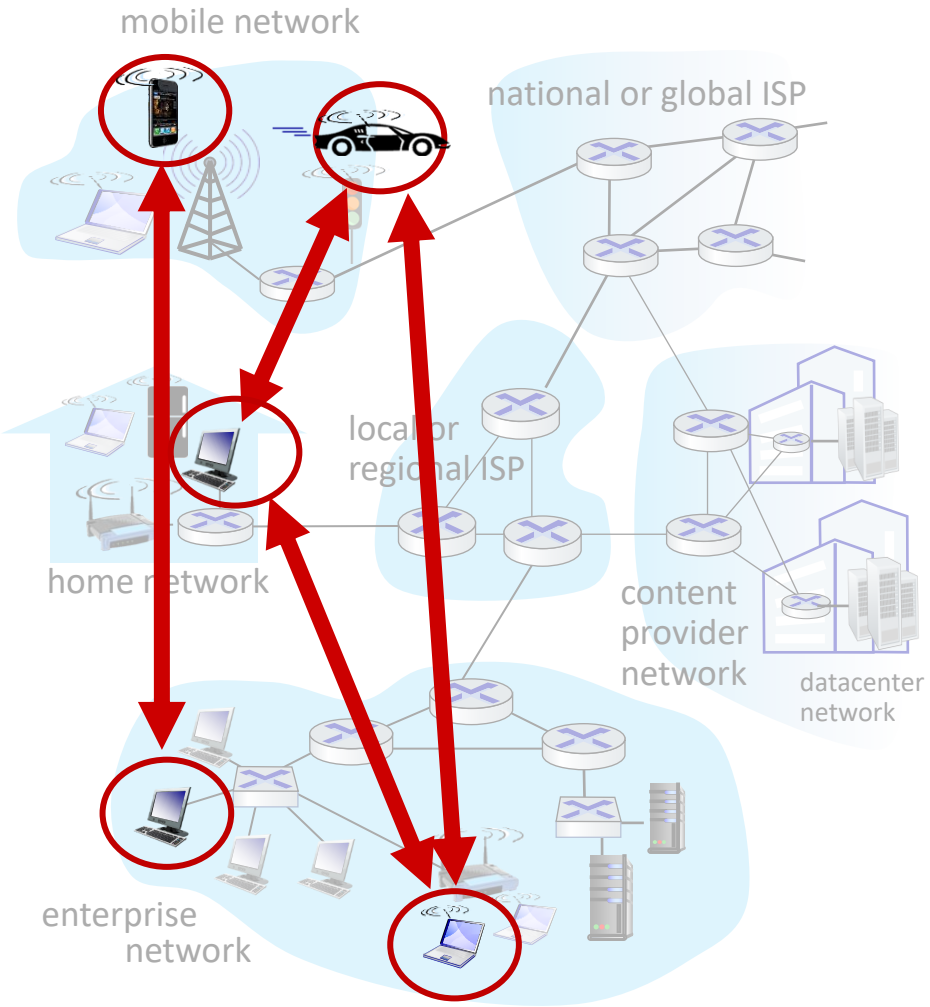
- contact, communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do *not* communicate directly with each other

- examples: HTTP, IMAP, FTP

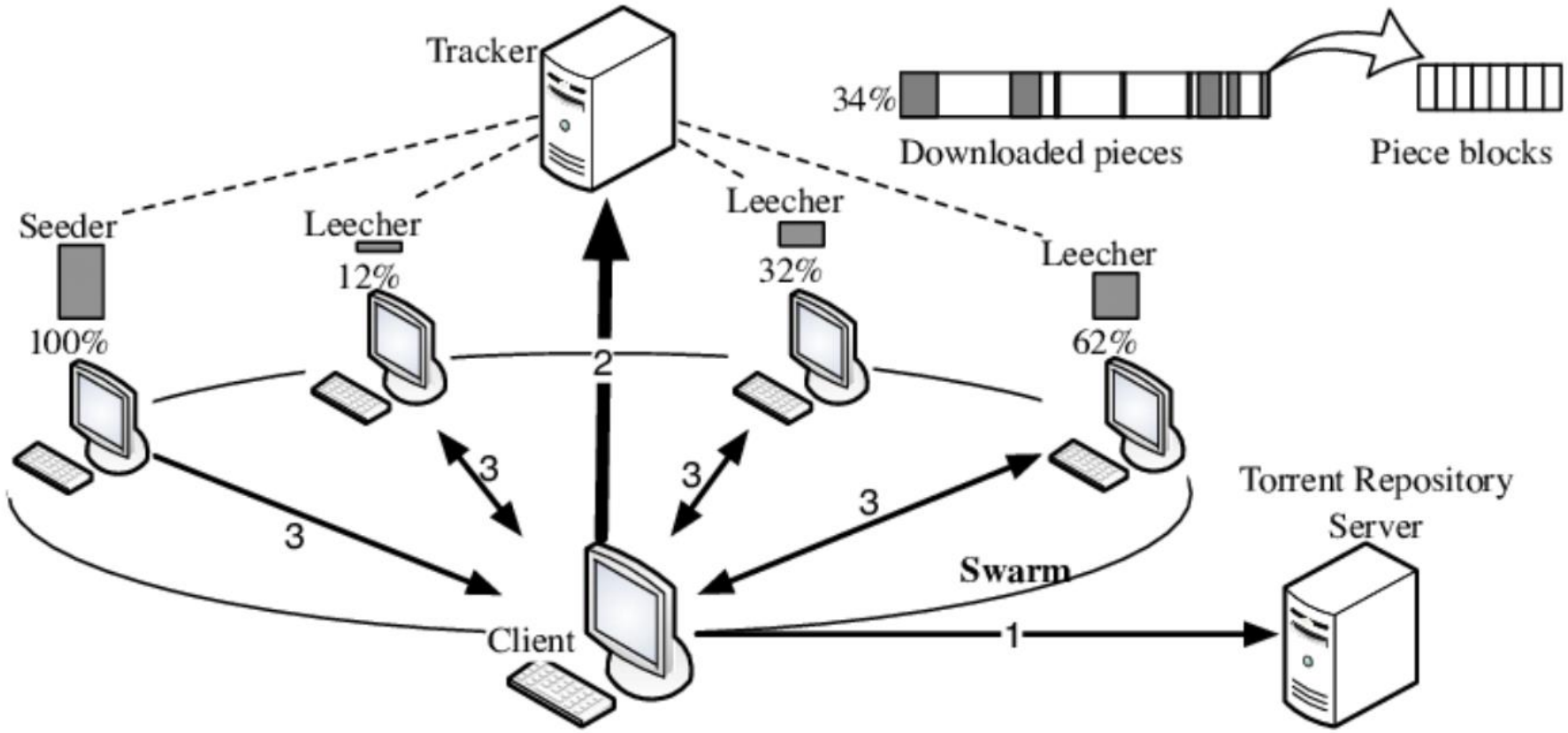


Peer-peer Architecture

- *No* always-on server
- Arbitrary end systems directly communicate
- Peers request service from other peers, provide service in return to other peers
- Peers are intermittently connected and change IP addresses
  - complex management
- Example: P2P file sharing [BitTorrent]



Bit Torrent Architecture (Informational)



Source : https://www.researchgate.net/figure/BitTorrent-Architecture_fig2_221082210

Processes communicating

Process: program running within a host

- Within same host, two processes communicate using inter-process communication (defined by OS)
- **Processes** in different hosts communicate by exchanging messages

clients, servers

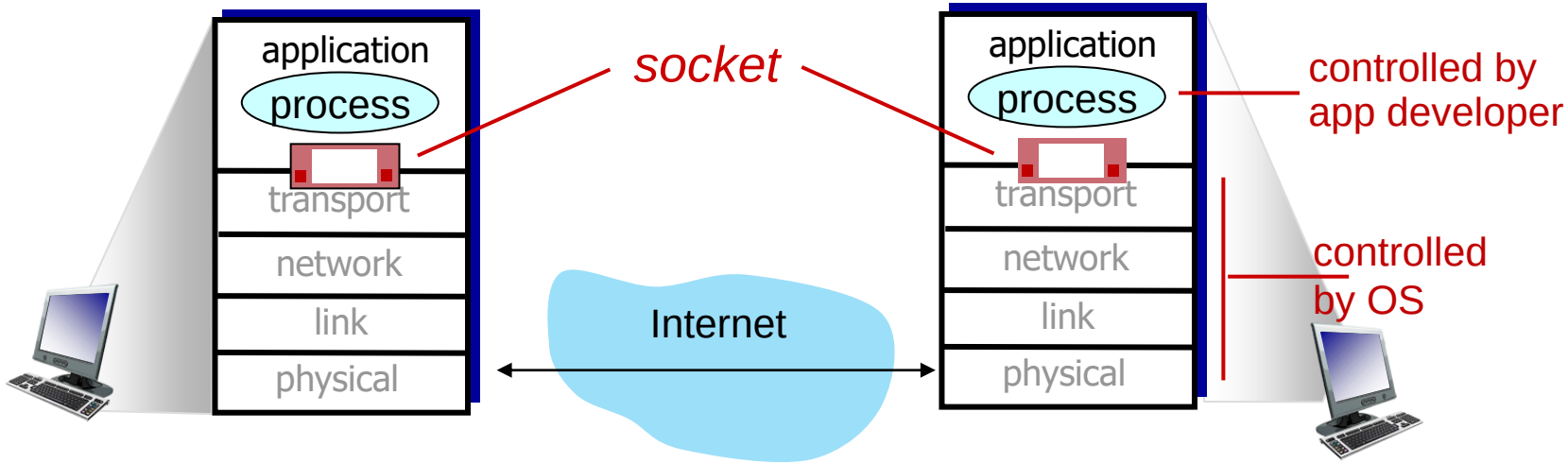
client process: Process that initiates communication

server process: Process that waits to be contacted

- Note: applications with P2P architectures have client processes & server processes

Sockets

- Process sends/receives messages to/from its **socket**
- Socket analogous to door
  - Sending process kicks message out the door
  - Sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
  - Two sockets involved: one on each side



Addressing processes

- To receive messages, process must have ***identifier***
- Host device has unique **32-bit IP address**
- *Q:* Does IP address of host on which process runs suffice for identifying the process?
  - *A:* No, *many* processes can be running on same host

- ***Identifier*** includes both **IP address** and **port numbers** associated with process on host.
- Example port numbers:
  - **HTTP server: 80**
  - **mail server: 25**
- To send HTTPS message to

Csus.edu web server:

- **IP address**: 130.86.9.189
- **port number**: 443

An application-layer protocol defines:

SACRAMENTO STATE

- **Types** of messages exchanged,
  - e.g., request, response
- Message **syntax**:
  - What fields in messages & how fields are delineated
- Message **semantics**
  - Meaning of information in fields

- **Open protocols**:
  - Defined in RFCs, everyone has access to protocol definition
  - Allows for interoperability
  - e.g., HTTP, SMTP
- Proprietary protocols:
  - e.g., Skype, Zoom

What transport service does an app need?

SACRAMENTO STATE

data integrity

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”

throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be “effective”
- other apps (“elastic apps”) make use of whatever throughput they get

security

- encryption, data integrity, …

Transport service requirements: common apps

application	throughput	time sensitive?
file transfer/download	elastic	no
e-mail	elastic	no
Web documents	elastic	no
real-time audio/video	audio: 5Kbps-1Mbps video:10Kbps-5Mbps	yes, 10's msec
streaming audio/video	same as above	yes, few secs
interactive games	Kbps+	yes, 10's msec
text messaging	elastic	yes and no

Internet transport protocols services

SACRAMENTO STATE

TCP service:

- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *connection-oriented:* setup required between client and server processes
- *does not provide:* timing, minimum throughput guarantee, security

UDP service:

- *unreliable data transfer* between sending and receiving process
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup.

Q: why bother? *Why* is there a UDP?

Internet applications, and transport protocols

application	application layer protocol	transport protocol
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Web documents	HTTP [RFC 7230, 9110]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
streaming audio/video	HTTP [RFC 7230], DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

Group Activity 2.1 – Graded (10 minutes)

- Form groups of 4
- Choose an application you want to develop
- Now discuss and answer the following
  - What network architecture will you use for your applications? Justify.
  - What transport services do you require? Justify.

- Share your answer with another group and receive feedback

Web and HTTP

SACRAMENTO STATE

First, a quick review…

- Web page consists of *objects,* each of which can be stored on different Web servers
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects, each* addressable by a *URL,* e.g.,

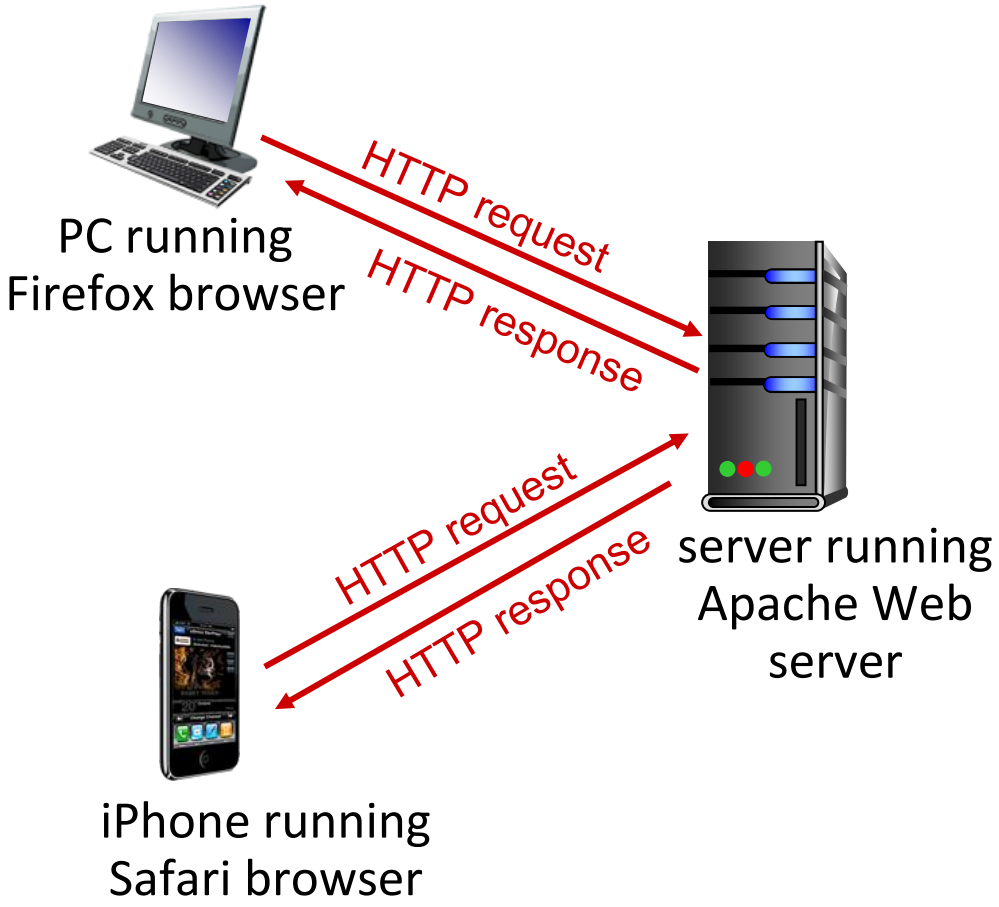
`www.someschool.edu/someDept/pic.gif`

host name | path name

HTTP overview

HTTP: hypertext transfer protocol

- Web's application-layer protocol
- client/server model:
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests



HTTP overview (continued)

HTTP uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- server maintains *no* information about past client requests

aside

protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP connections: two types

Non-persistent HTTP

1. TCP connection opened
2. at most one object sent over TCP connection
3. TCP connection closed

downloading multiple objects required multiple connections

Persistent HTTP

- TCP connection opened to a server
- multiple objects can be sent over *single* TCP connection between client, and that server
- TCP connection closed

Non-persistent HTTP: example

User enters URL: **`www.someSchool.edu/someDepartment/home.index`**
(containing text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80 "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

Non-persistent HTTP: example (cont.)

User enters URL: **`www.someSchool.edu/someDepartment/home.index`**
(containing text, references to 10 jpeg images)

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

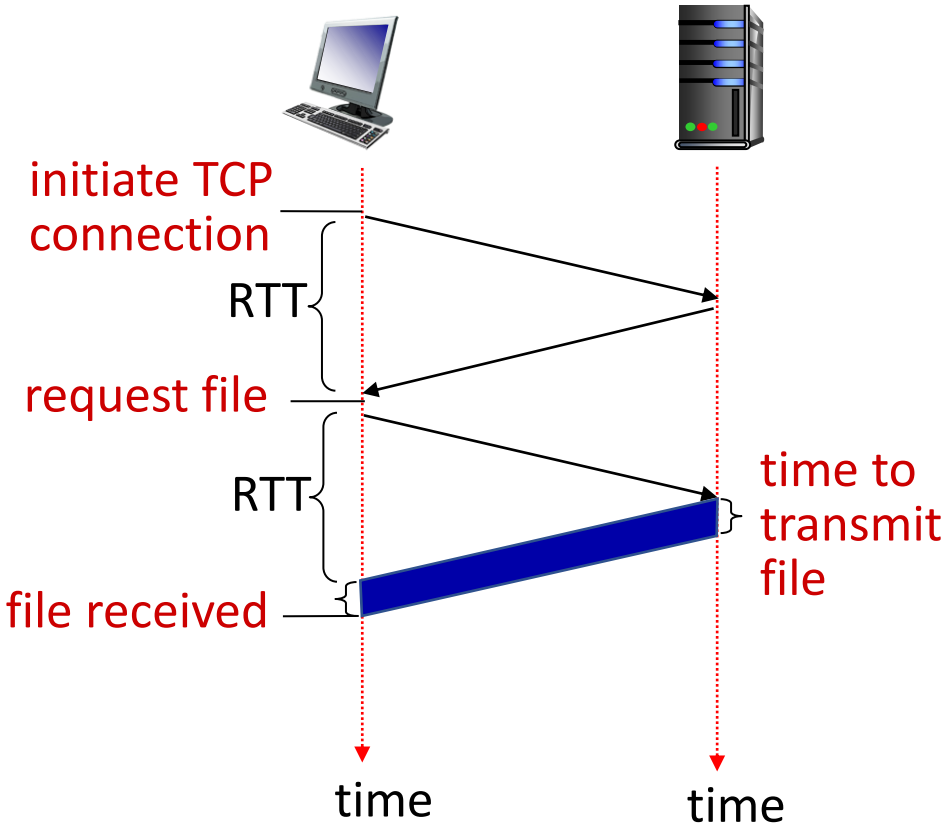
time

Non-persistent HTTP: response time

RTT (definition): time for a small packet to travel from client to server and back

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



Non-persistent HTTP response time = 2RTT+ file transmission time

Persistent HTTP (HTTP 1.1)

SACRAMENTO STATE

Non-persistent HTTP issues:

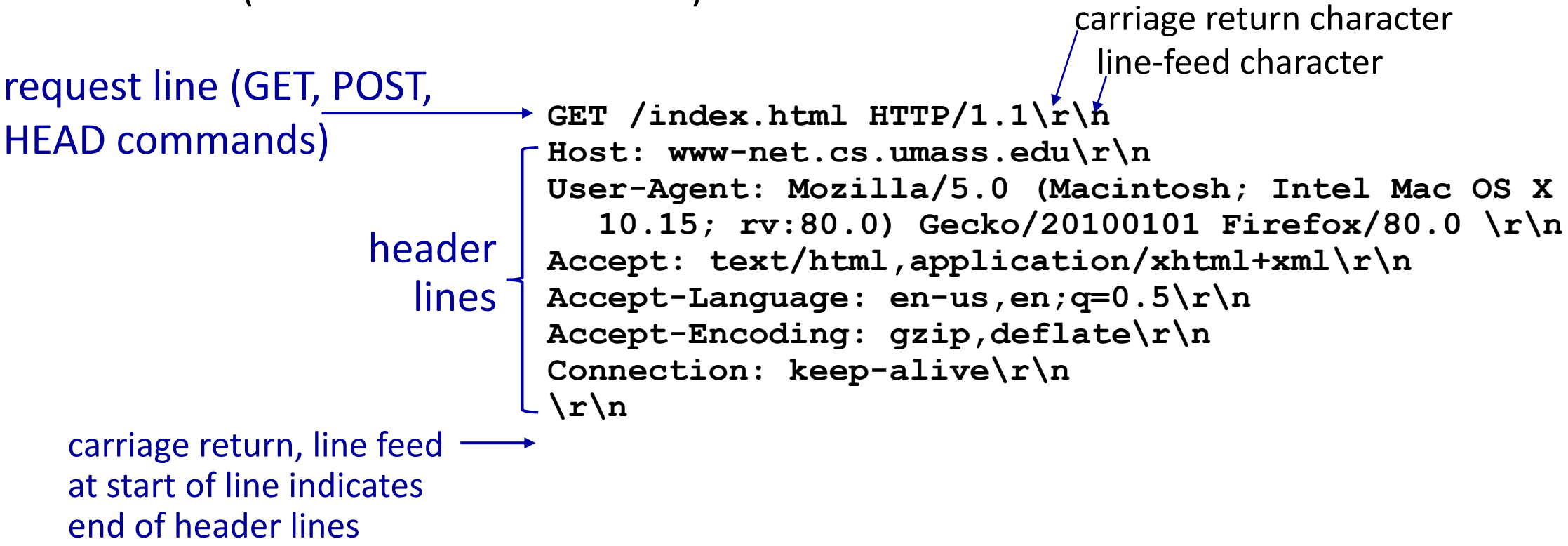
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open multiple parallel TCP connections to fetch referenced objects in parallel

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)

HTTP request message

- two types of HTTP messages: *request*, *response*
- HTTP request message:
  - ASCII (human-readable format)



Other HTTP request messages

SACRAMENTO STATE

POST method:

- user input sent from client to server in entity body of HTTP POST request message

GET method (for sending data to server):

- include user data in URL field of HTTP GET request message (following a '?'):

```
www.somesite.com/animalsearch?monkeys&banana
```

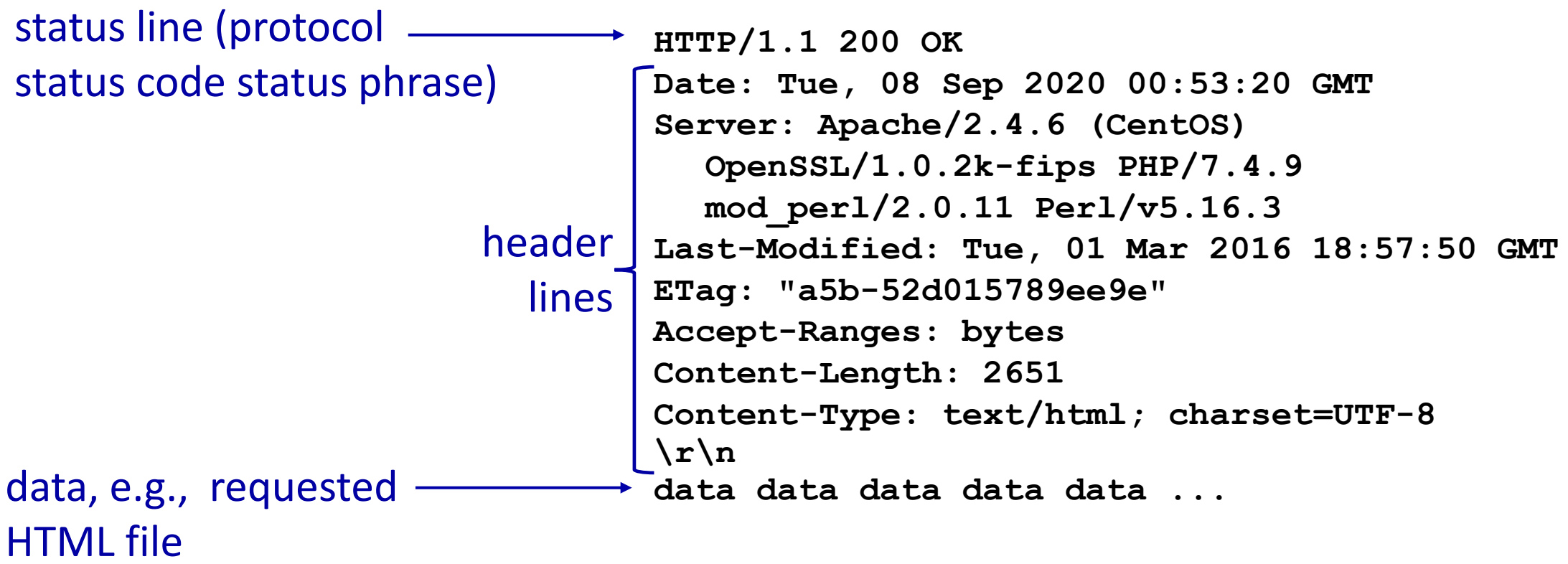
HEAD method:

- requests headers (only) that would be returned *if* specified URL were requested with an HTTP GET method.

PUT method:

- uploads new file (object) to server
- completely replaces file that exists at specified URL with content in entity body of POST HTTP request message

HTTP response message



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

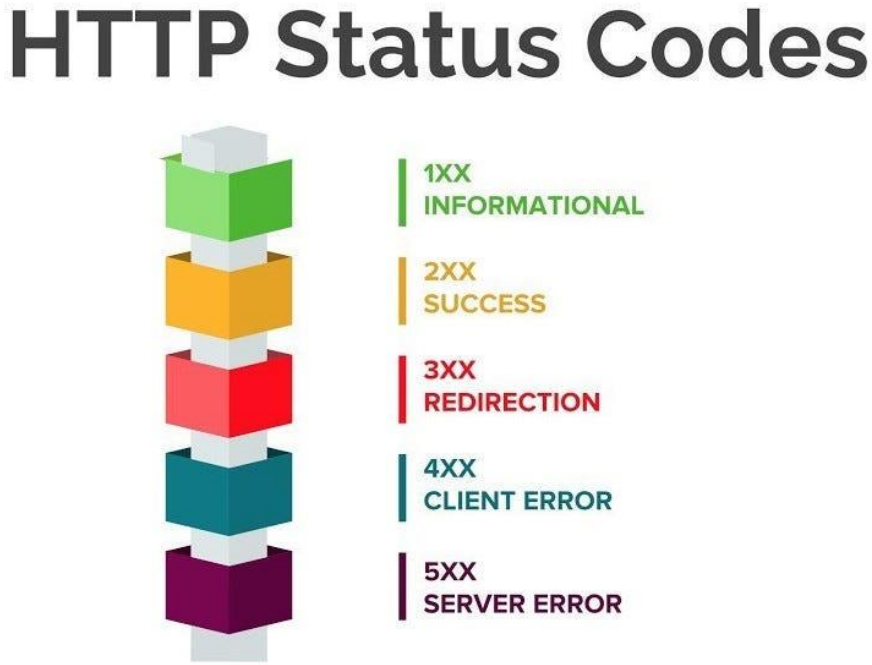
200 OK
- request succeeded, requested object later in this message

301 Moved Permanently
- requested object moved, new location specified later in this message (in Location: field)

400 Bad Request
- request msg not understood by server

404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported



Group Activity 2.2 - Graded

You are a group of network application developers. You are given a task by your supervisor to develop a network application to host a e-commerce application. You are advised to maintain a low overhead on TCP. For this you jot down following points that you need to discuss with your group members.

1. What protocol will you use to develop the website application?
2. Does this protocol have persistent or non-persistent connection choice?
3. How does this affect your application?
4. What protocol methods do you need to consider for user inputs to this website?
5. What response codes you need to configure for it?
6. What transport service is required for the website? Justify?

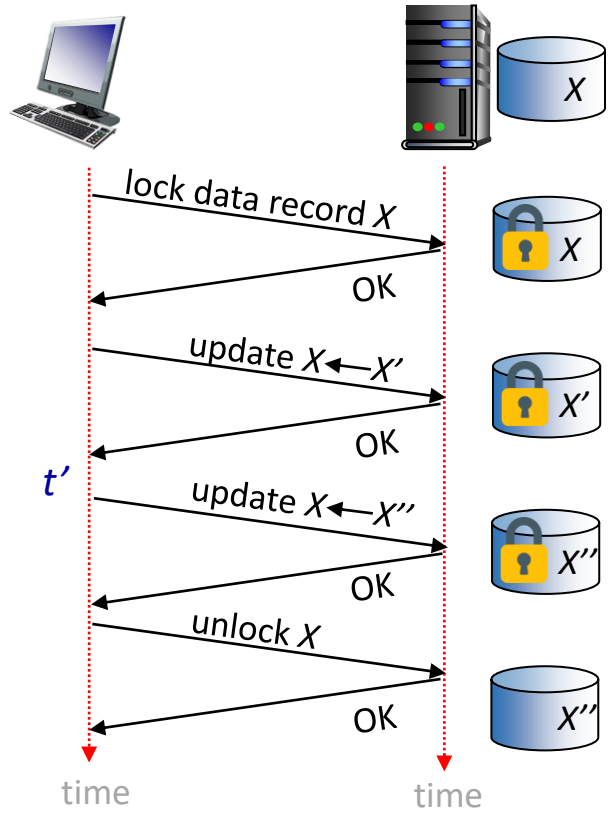
After your discussion is over, share the summary of your discussion in class

Maintaining user/server state: cookies

Recall: HTTP GET/response interaction is *stateless*

- no notion of multi-step exchanges of HTTP messages to complete a Web "transaction"
  - no need for client/server to track "state" of multi-step exchange
  - all HTTP requests are independent of each other
  - no need for client/server to "recover" from a partially-completed-but-never-completely-completed transaction

a stateful protocol: client makes two changes to X, or none at all



Q: what happens if network connection or client crashes at *t'* ?

Maintaining user/server state: cookies

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

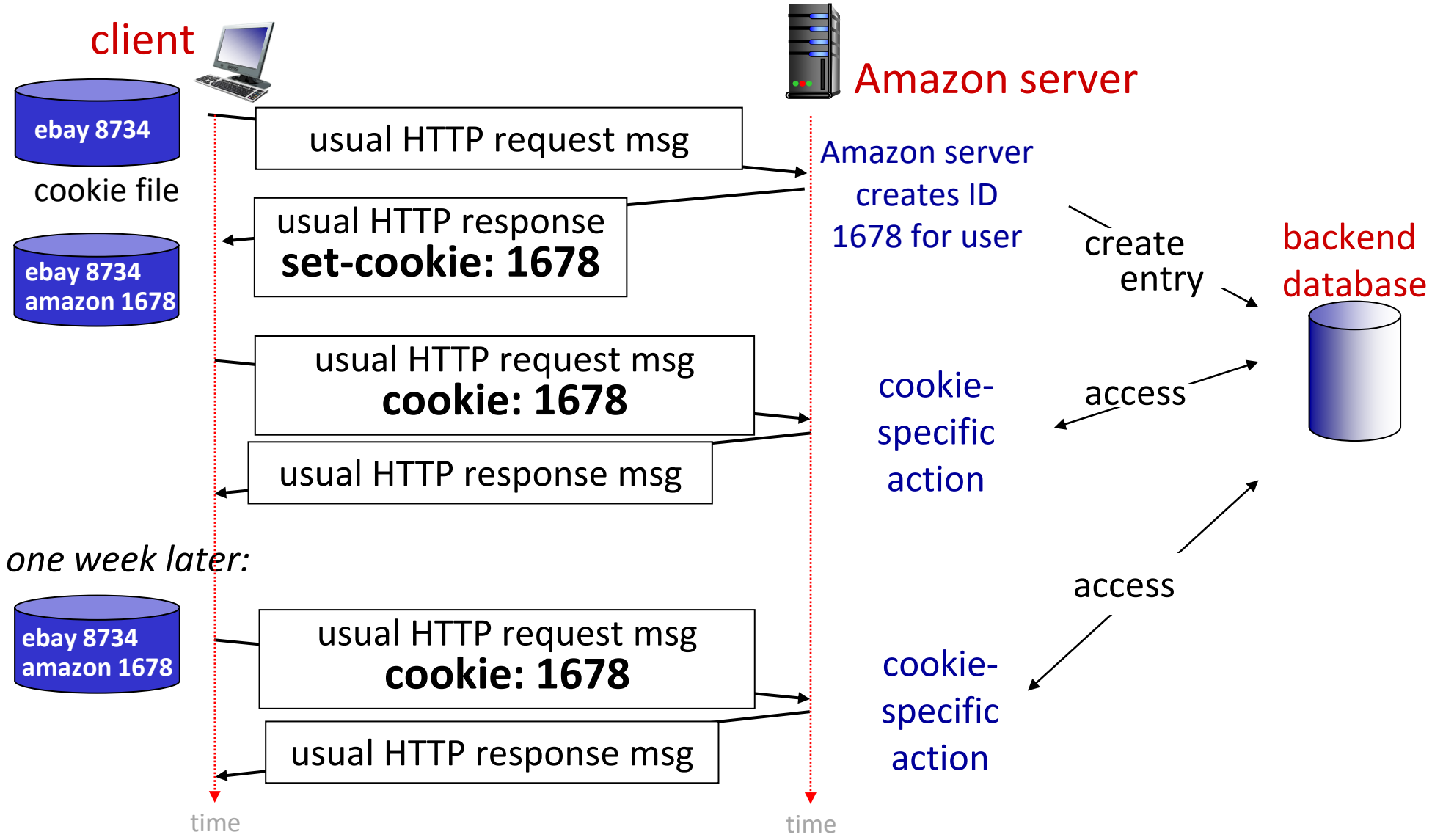
four components:

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

Maintaining user/server state: cookies



HTTP cookies: comments

What cookies can be used for:

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

Challenge: How to keep state?

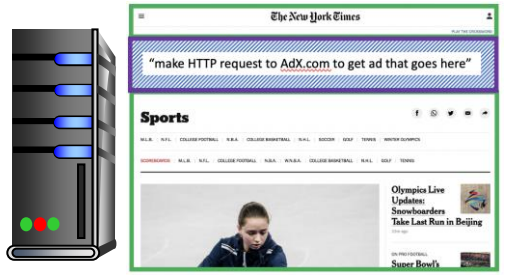
- *At protocol endpoints:* maintain state at sender/receiver over multiple transactions
- *In messages:* cookies in HTTP messages carry state

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

Cookies: tracking a user's browsing behavior



nytimes.com (sports)

1634: sports, 2/15/22

"first party" cookie – from website you chose to visit (provides base html file)

HTTP GET

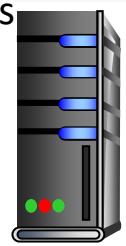
HTTP reply
Set cookie: 1634

HTTP GET
Referrer: NY Times Sports

4

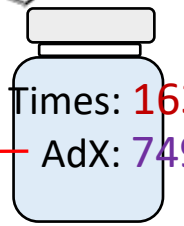
5

HTTP reply
Set cookie: 7493



7493: NY Times sports, 2/15/22

AdX.com

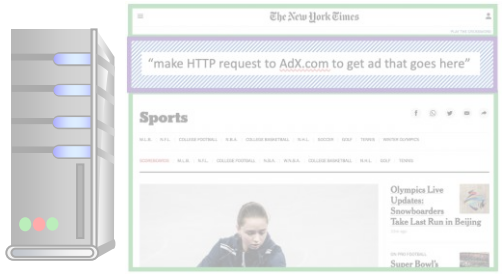


NY Times: 1634
AdX: 7493

"third party" cookie – from website you did not choose to visit

Cookies: tracking a user's browsing behavior

socks.com



nytimes.com

HTTP reply 2

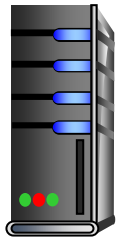
HTTP GET 1

HTTP GET
Referrer: socks.com, cookie: 7493

4

5

HTTP reply
Set cookie: 7493



AdX.com

AdX:

- *tracks my web browsing* over sites with AdX ads
- can return targeted ads based on browsing history

Cookies: tracking a user's browsing behavior

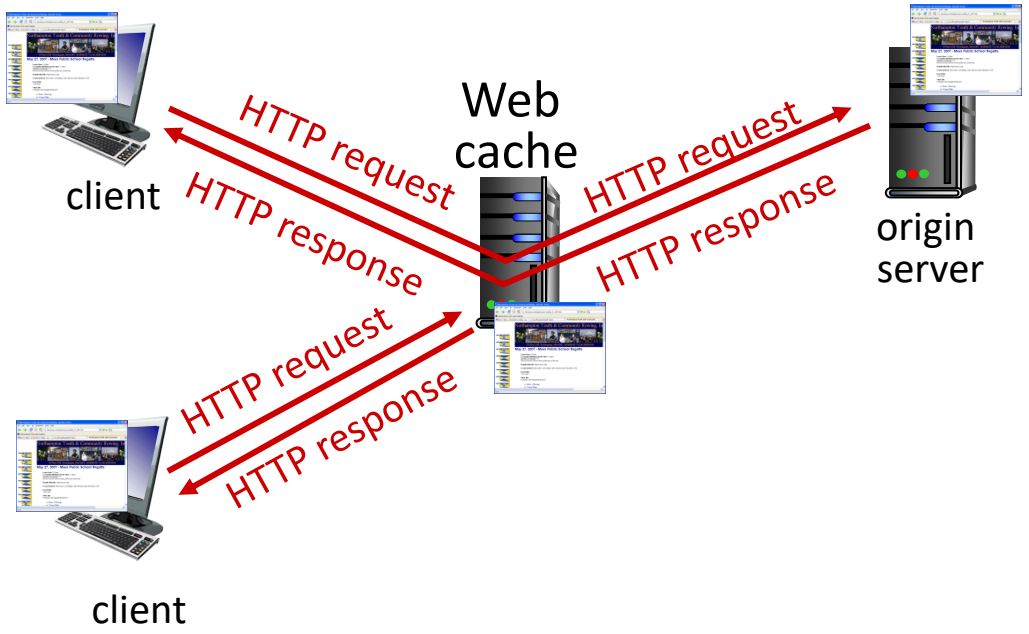
Cookies can be used to:

- Track user behavior on a given website (first party cookies)
- Track user behavior across multiple websites (third party cookies) without user ever choosing to visit tracker site (!)
- Tracking may be *invisible* to user:
  - rather than displayed ad triggering HTTP GET to tracker, could be an invisible link

Web caches

Goal: satisfy client requests 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

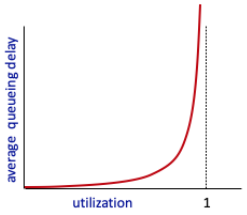
Caching example

Scenario:

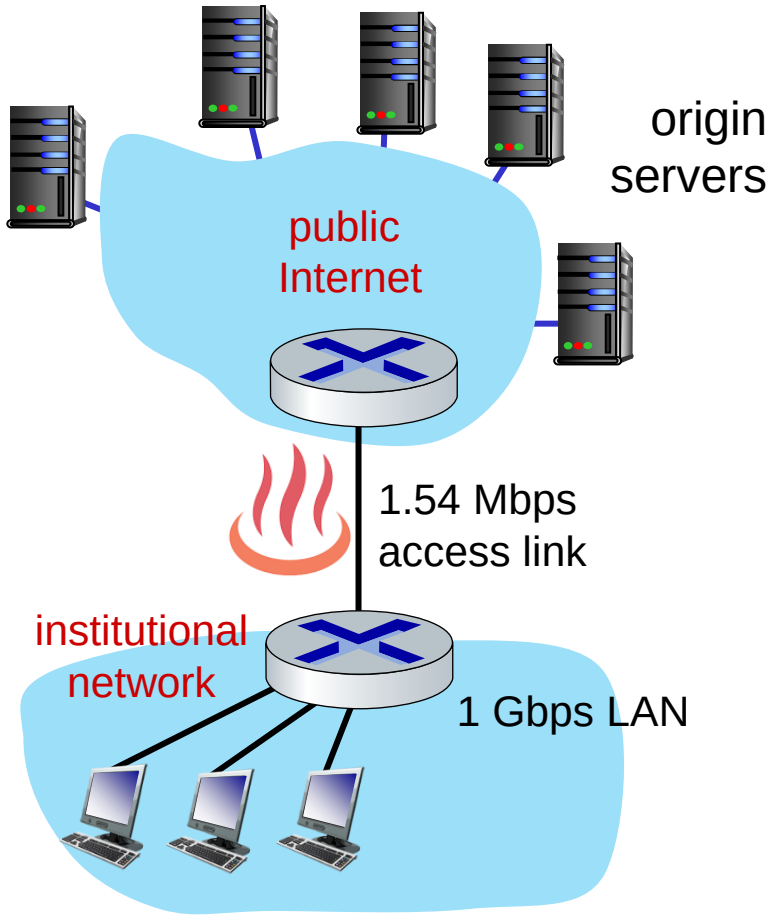
- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

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

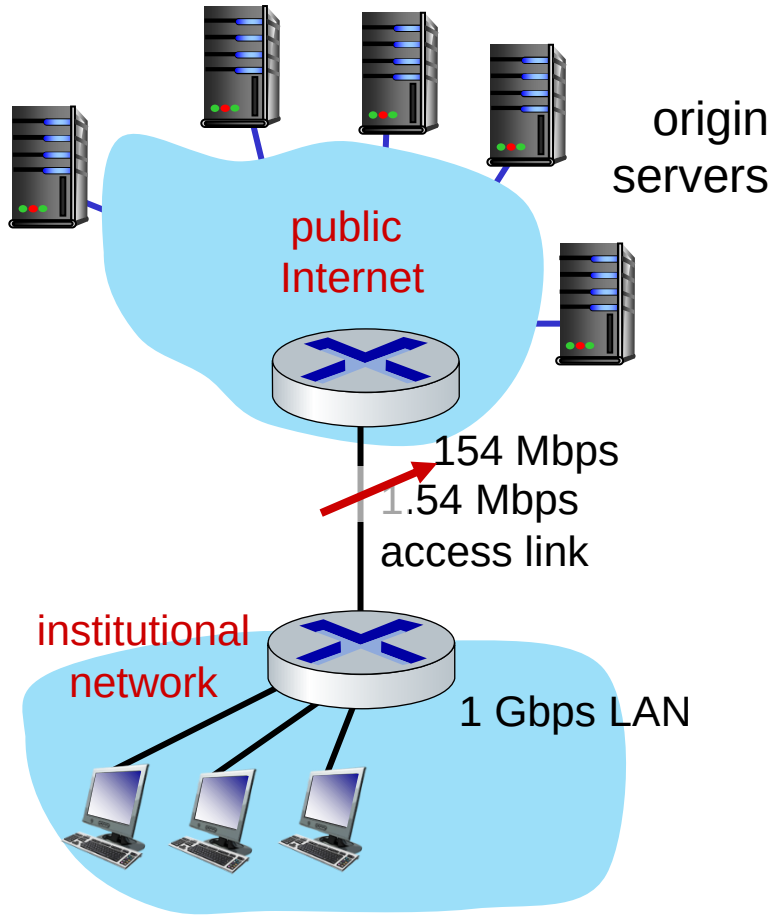
Scenario:

- 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~~ + usecs msecs

Cost: faster access link (expensive!)



Option 2: install a web cache

Scenario:

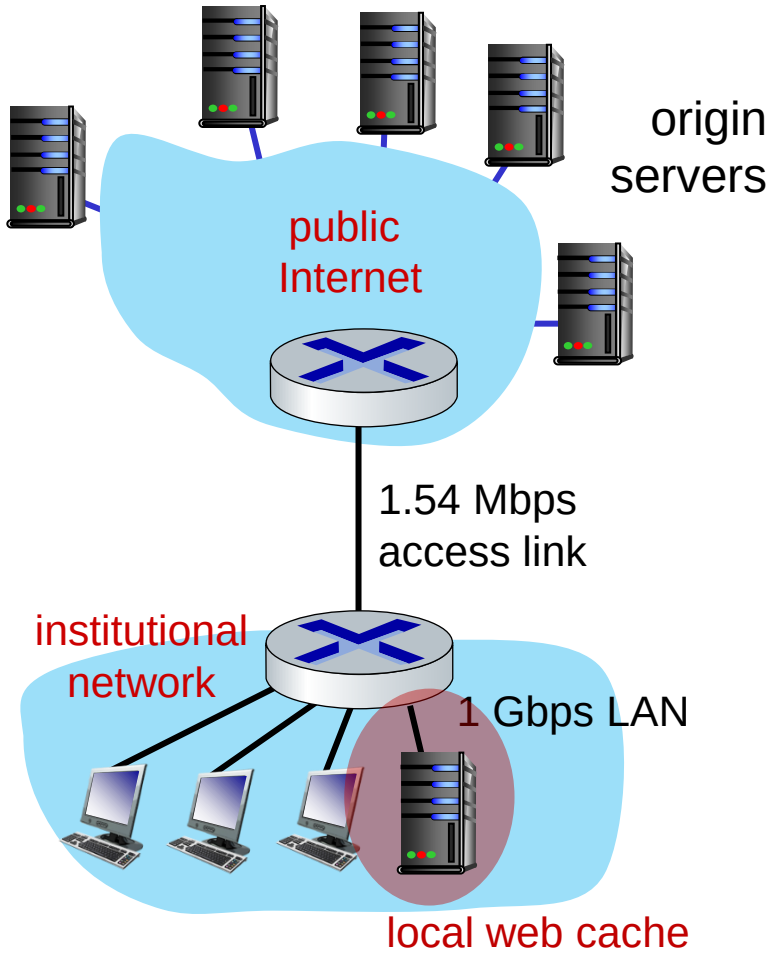
- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Cost: web cache (cheap!)

Performance:

- LAN utilization: .?
- access link utilization = ?
- average end-end delay = ?

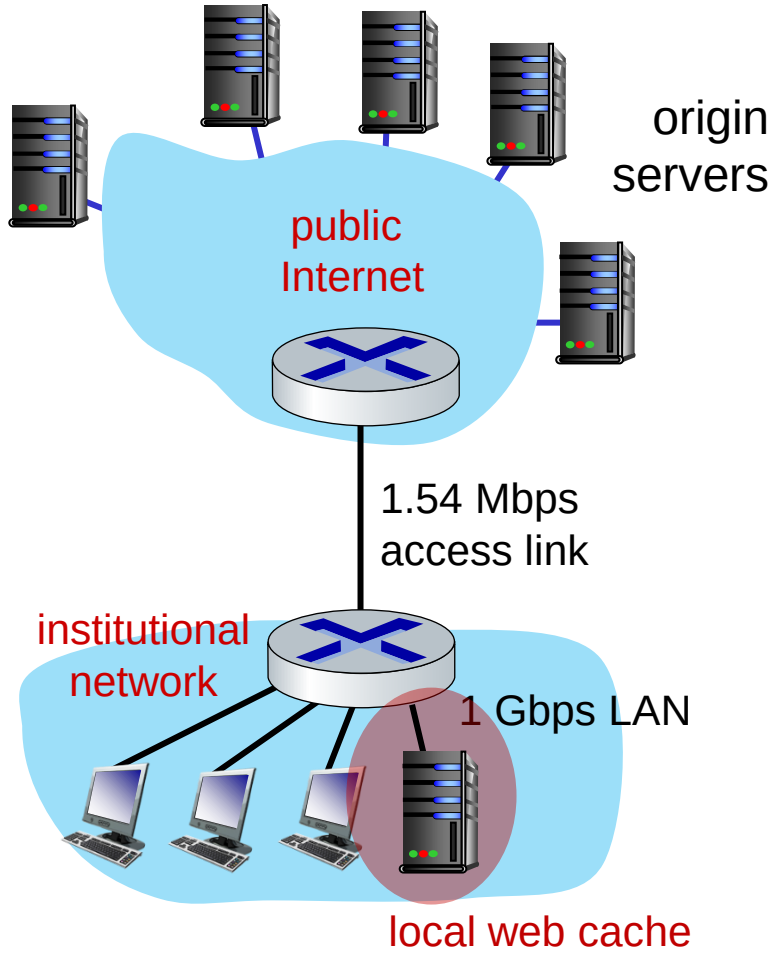
How to compute link utilization, delay?



Calculating 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 (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



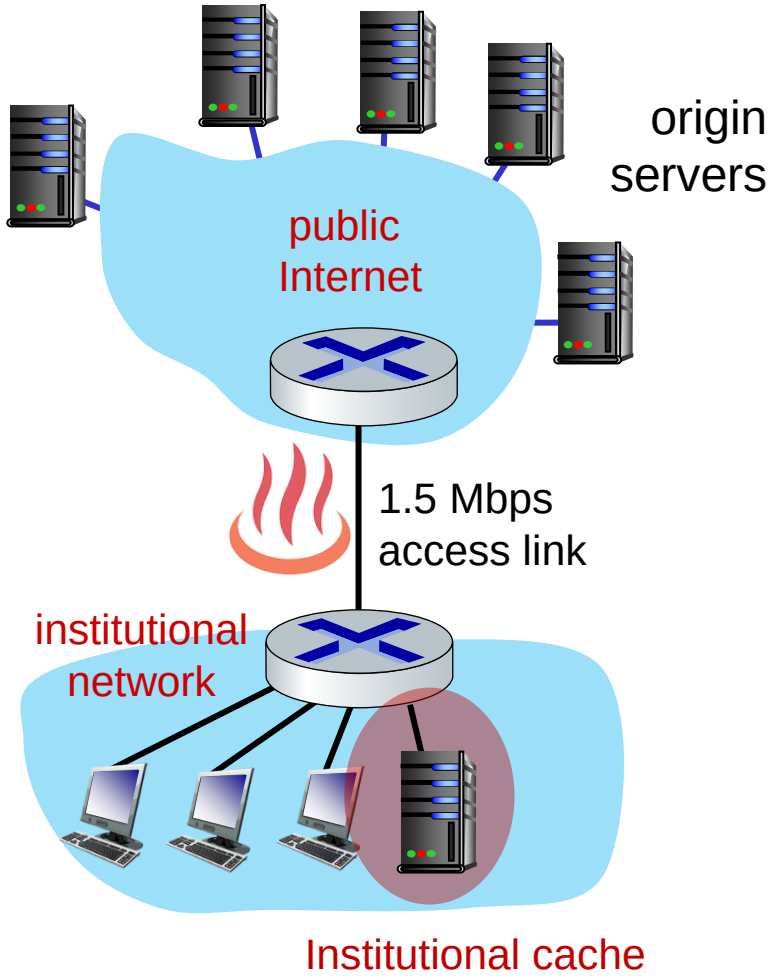
lower average end-end delay than with 154 Mbps link (and cheaper too!)

Activity 3.3

For a file transfer request, the file size is 40,000 bits, average request rate from the institution's browsers to the origin server is 30 requests per second. Also, suppose that the amount of time it takes from when the router on the Internet side of the access link forwards an HTTP request until it receives the response is 2 seconds on average. Model the total average response time as the sum of the average access delay and the average Internet delay. For the average access delay, use $\delta/(1-\delta\alpha)$, where δ is the average time required to send an object over the access link and α is the arrival rate of objects to the access link. You can assume that the HTTP request messages are negligibly small and thus create no realized traffic on the network or the access link. Show your calculations.

a. Find the total average response time.

a. Now, suppose a cache is installed in the institutional LAN (see figure). Suppose the miss rate is 0.3. Find the total response time.



Pair Activity : Solution

a. Find the total average response time.

δ = transmission delay = 40000 / 1.5 *10^6 = 0.0266

α = arrival rate = 30 requests/ sec

Average access delay is given by $\delta/(1-\delta\alpha)$

$\delta/(1-\delta\alpha)$ = 0.0266 / (1 – (0.0266 * 30))

= 0.0266 / (1 – 0.798)

= 0.0266 / 0.202 = 0.131 sec

Total delay = Internet delay + average access delay

= 2 + 0.131

= 2.131 sec

Pair Activity : Solution

b. Now, suppose a cache is installed in the institutional LAN (see figure). Suppose the miss rate is 0.3. Find the total response time.

Since the miss rate is 0.3, assume web-cache delay is 0.

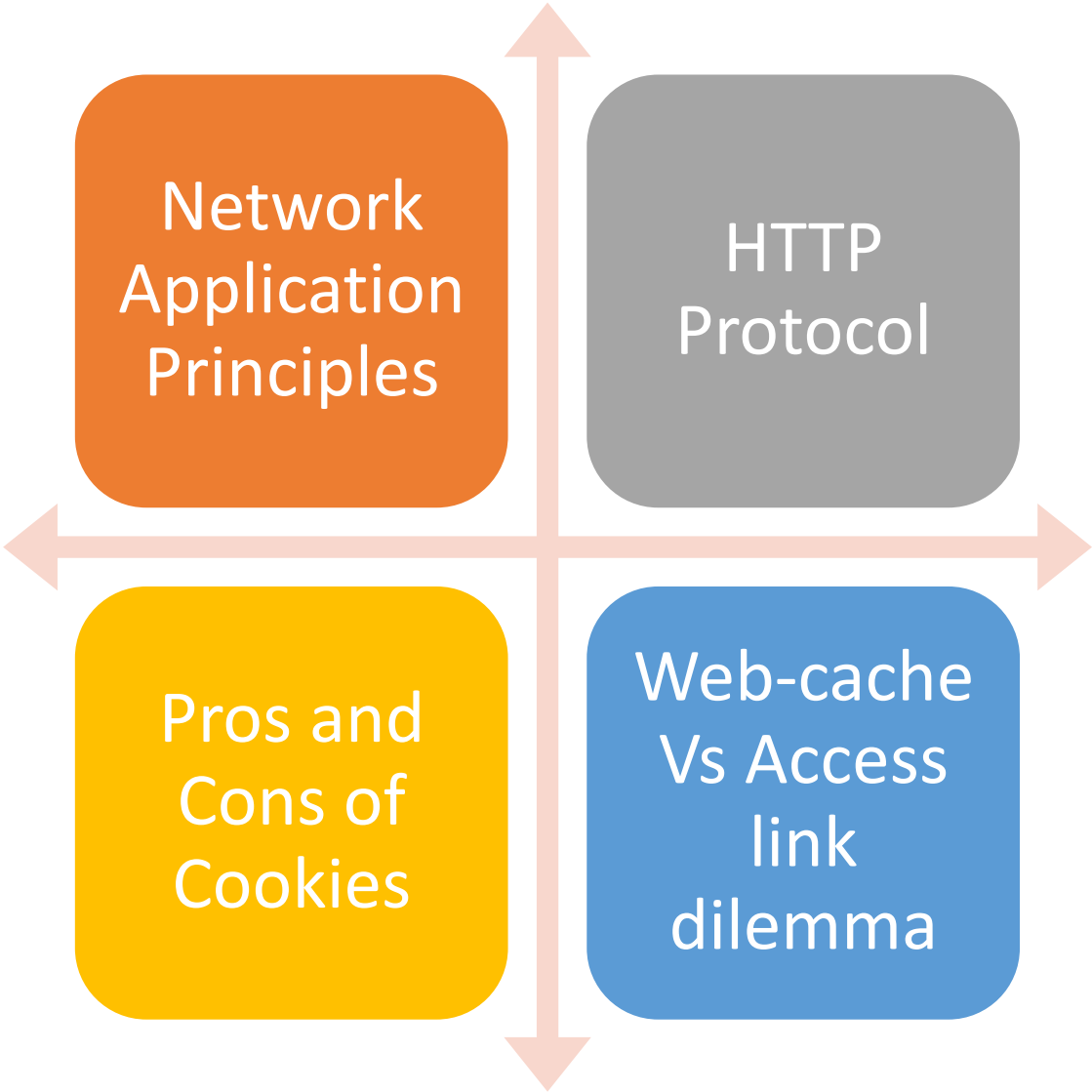
Web-cache delay = 0.3 (time of response) = 0.3 * 0 = 0

Rest of the response takes 70% , for which we use 0.7 $(\frac{\delta}{1-(\delta\alpha)})$.

Average access delay =0.7* $(\delta/(1-(\delta\alpha)))$
=0.7* 0.131
=0.091

Total delay = Internet delay + average access delay
= 2 + 0.091
= 2.091 sec

Summarize Topics for Lecture 2_1



End of Lecture 2_1