CS 305: Computer Networks

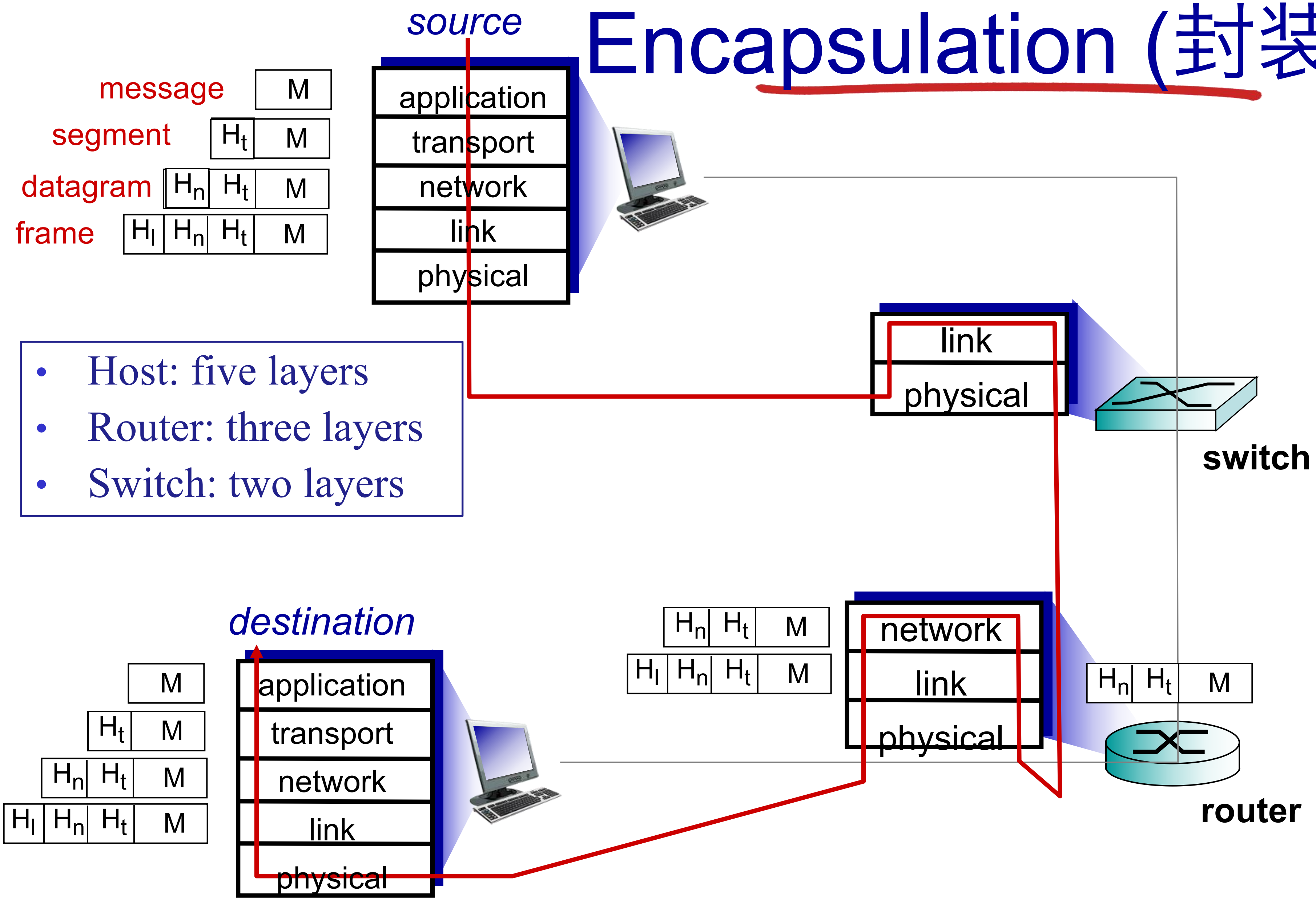
Fall 2022

Lecture 3： Application Layer

Ming Tang

Department of Computer Science and Engineering
Southern University of Science and Technology (SUSTech)

Encapsulation (封装)



- Host: five layers
- Router: three layers
- Switch: two layers

Chapter 1: roadmap

1.1 what is the Internet?

1.2 network edge

- end systems, access networks, links

1.3 network core

- packet switching, circuit switching, network structure

1.4 delay, loss, throughput in networks

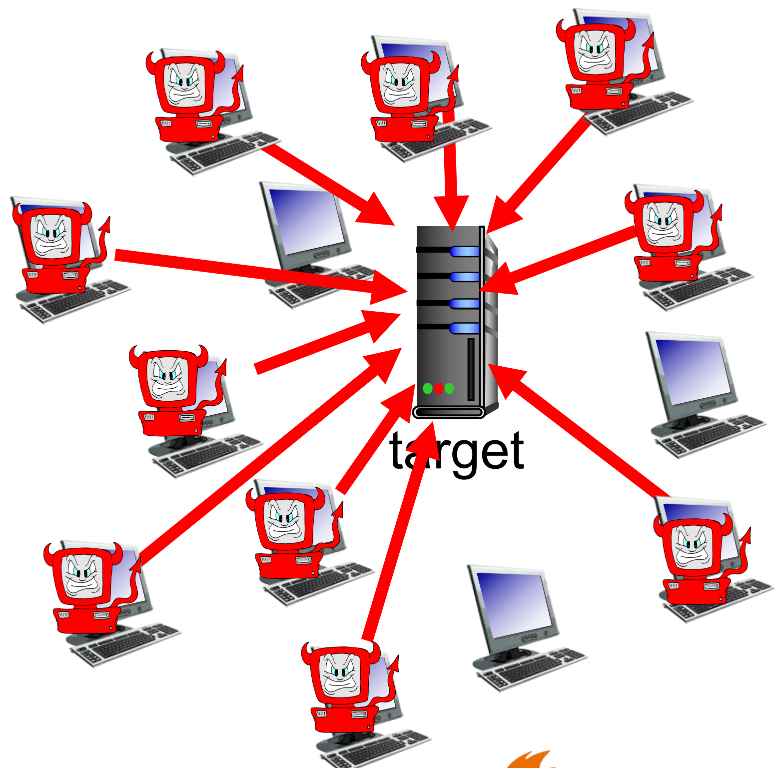
1.5 protocol layers, service models

1.6 networks under attack: security

Bad guys: attack server, network infrastructure

Denial of Service (DoS): attackers make resources (server, bandwidth) unavailable to legitimate （合法的） traffic by overwhelming resource with bogus （伪造的） traffic

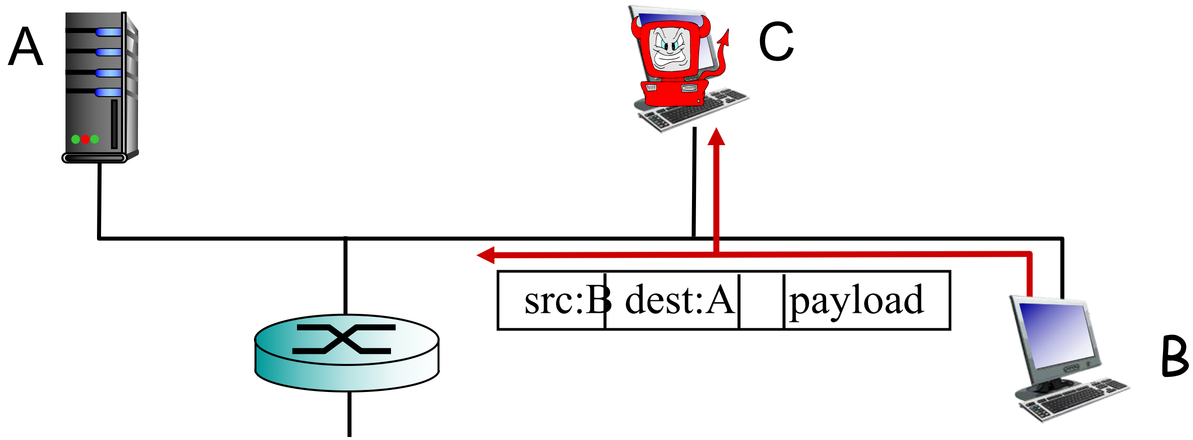
1. select target
2. break into hosts around the network (see botnet)
3. send packets to target from compromised hosts



Bad guys can sniff packets

Packet "sniffing":

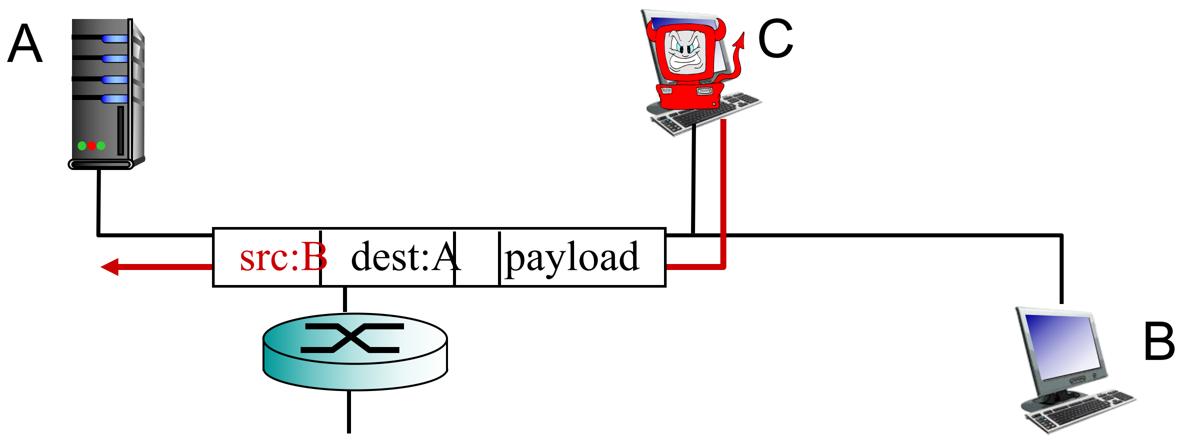
- Broadcast media (shared ethernet, wireless)
- Reads/records all packets (e.g., including passwords!) passing by
- They are difficult to detect



- wireshark software used for end-of-chapter labs is a (free) packet-sniffer

Bad guys can use fake addresses

IP spoofing: send packet with false source address



Lines of defense:

- authentication: proving you are who you say you are
  - cellular networks provides hardware identity via SIM card; no such hardware assist in traditional Internet
- confidentiality: via encryption
- integrity checks: digital signatures prevent/detect tampering
- access restrictions: password-protected VPNs
- firewalls: specialized "middleboxes" in access and core networks:
  - off-by-default: filter incoming packets to restrict senders, receivers, applications
  - detecting/reacting to DOS attacks

... lots more on security (throughout, Chapter 8)

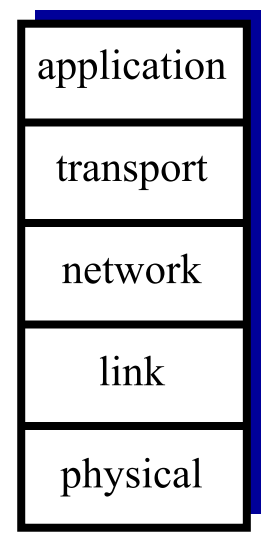
Introduction: summary

Covered a "ton" of material!

- Internet overview
- what's a protocol?
- network edge: access network
- network core
  - packet-switching versus circuit-switching
  - Internet structure
- performance: loss, delay, throughput
- layering, service models
- security

you now have:

- context, overview, "feel" of networking
- more depth, detail *to follow!*



Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

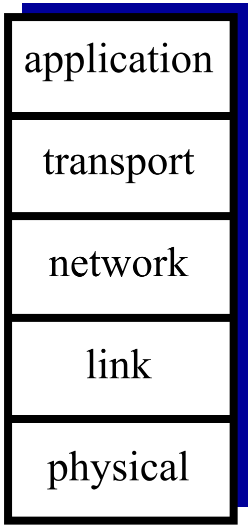
- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks

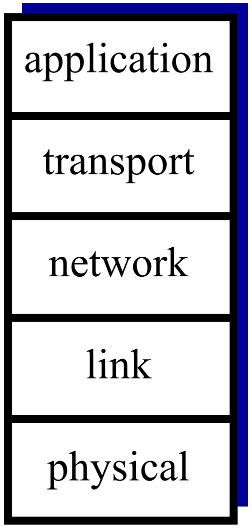
2.7 socket programming with UDP and TCP



Chapter 2: application layer

Our goals:

- Conceptual, implementation aspects of network application protocols
  - client-server architecture
  - peer-to-peer architecture
  - transport-layer service models
- Learn about protocols by examining popular application-level protocols
  - HTTP
  - SMTP / POP3 / IMAP
  - DNS
- Creating network applications
  - socket API



Some network apps

- e-mail
- web
- text messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)
- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
- search
- …
- …

Creating a network app

To build a network application - Application layer:

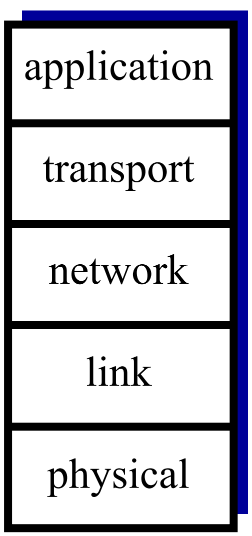
Q1: Which architecture? client-server or peer-to-peer?

Q2: Which transport layer protocol to choose, e.g., TCP? UDP?

Q3: Which protocol to follow? HTTP for web? SMTP for email? Or even your own designed protocol?

Transport layer (TCP and UDP):

Sending the message from process to process



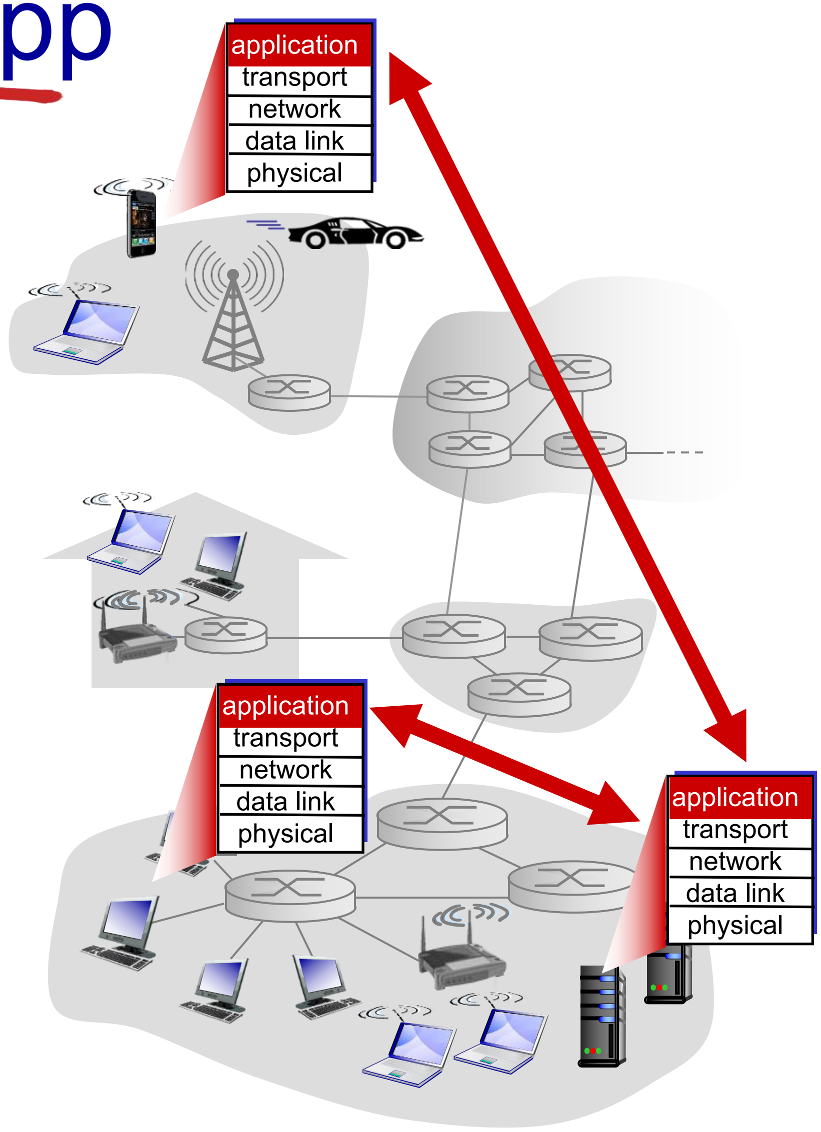
Creating a network app

Write programs that:

- run on (different) *end systems*
- communicate over network
- e.g., web server software (at server's host) communicates with browser software (at user's host)

No need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allows for rapid app development



Some network apps

To build a network application - Application layer:

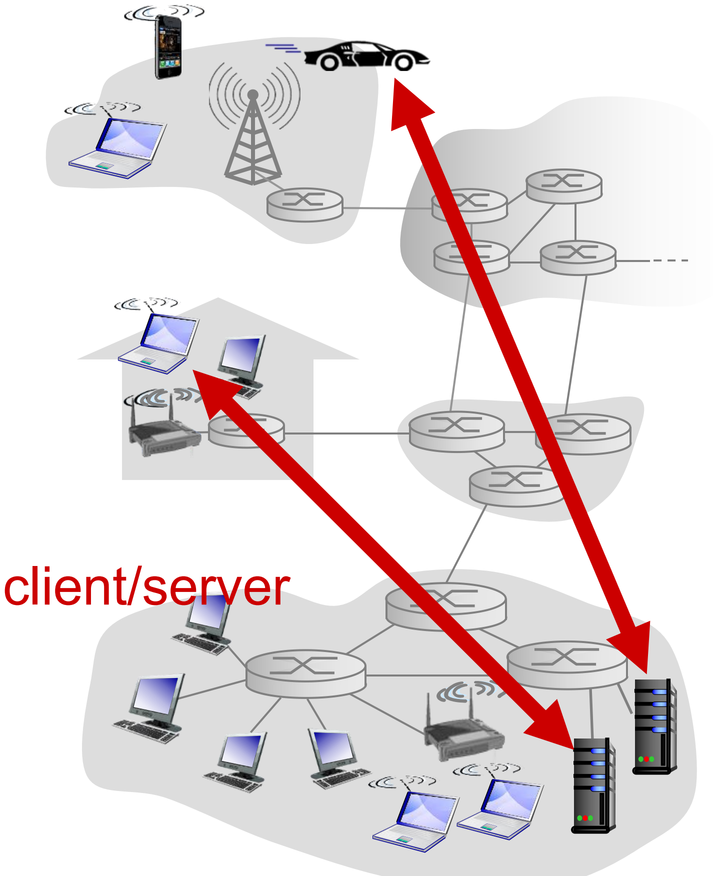
Q1: Which architecture?

- client-server architecture: e.g., web, email
- peer-to-peer architecture: e.g., P2P file sharing

Q2: Which transport layer protocol to choose, e.g.g., TCP? UDP?

Q3: Which protocol to follow? HTTP for web? SMTP for email? Or even your own designed protocol?

Client-server architecture

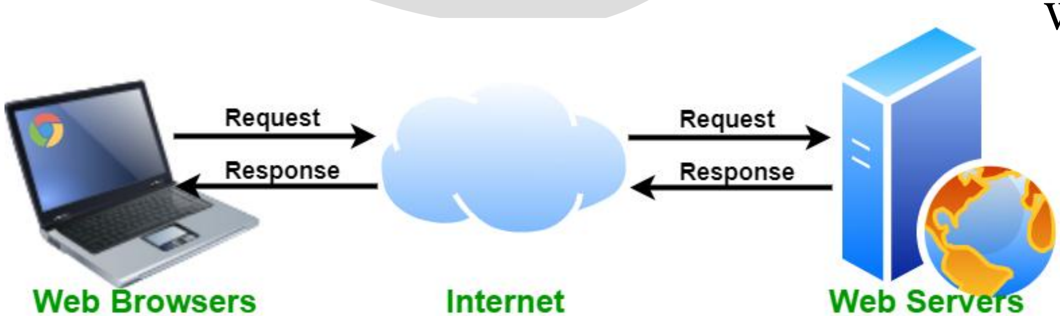


Server:

- always-on host
- Permanent (fixed, well-konwn) IP address
- data centers for scaling

Clients:

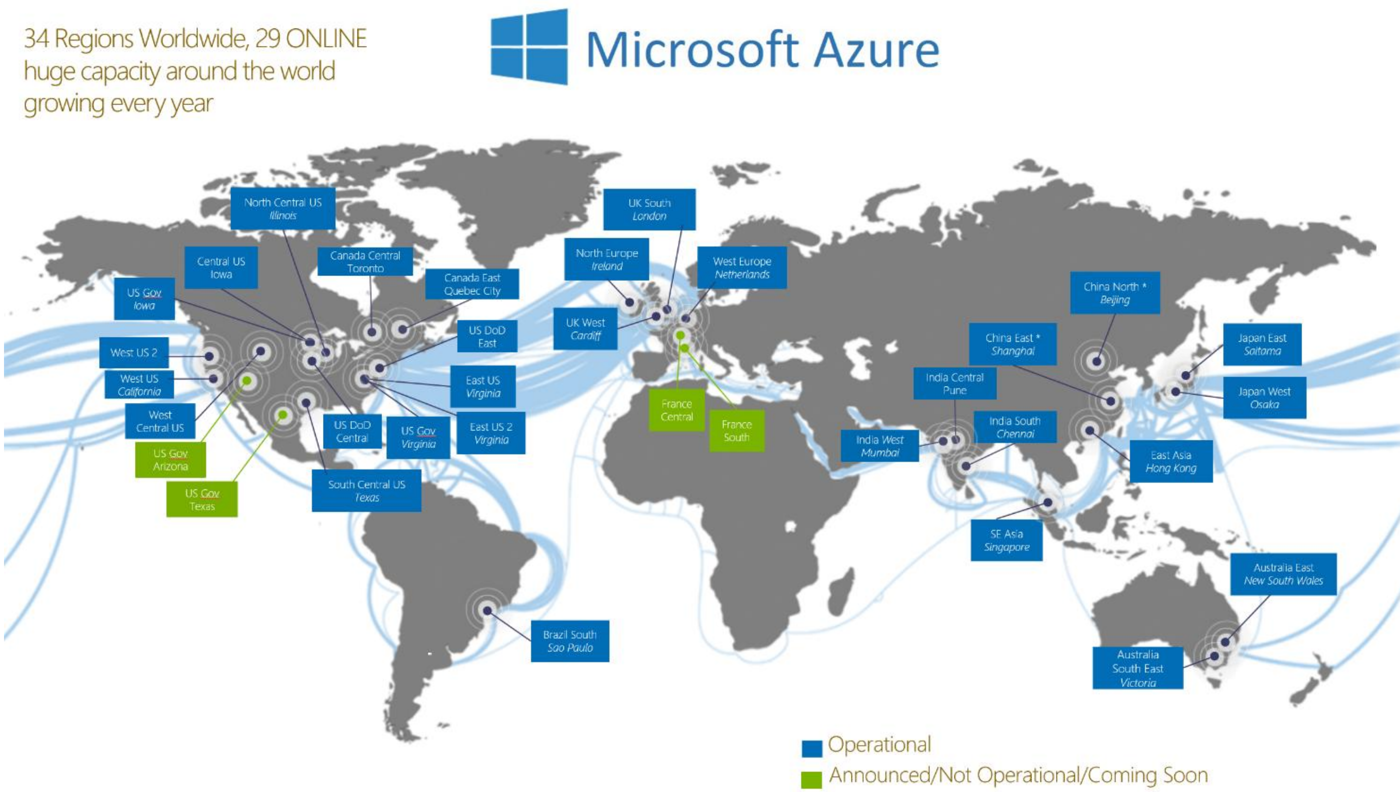
- communicate with server
- may be intermittently (间接性) connected
- may have dynamic IP addresses
- do not communicate directly with each other



Examples: Web and E-mail

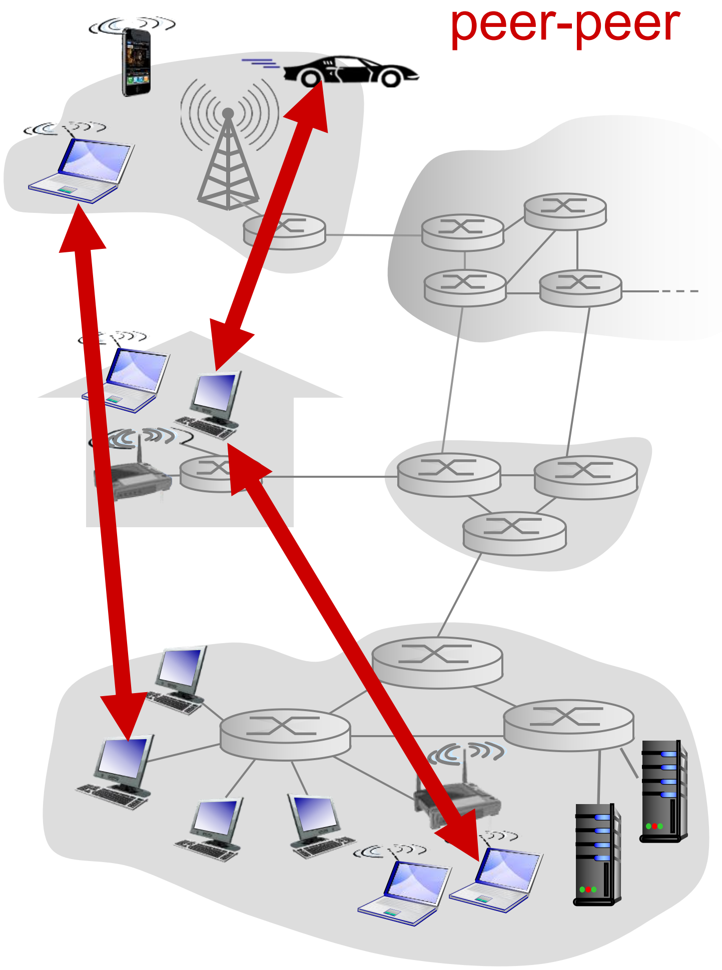
Client-server architecture

Data centers for scaling： a large number of hosts to create a powerful virtual server; distributed around the world



P2P Architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- **Example**: P2P file sharing



Peers are intermittently connected and change IP addresses
- complex management

Hybrid architectures: client-server + P2P

Creating a network app

To build a network application - Application layer:

Q1: Which architecture? client-server or peer-to-peer?

Q2: Which transport layer protocol to choose, e.g., TCP? UDP?

- How do apps (at end systems) exchange messages?
  - E.g., how does a browser exchange message with a server?
- How to choose transport services?

Q3: Which protocol to follow? HTTP for web? SMTP for email? Or even your own designed protocol?

How exchange msg?

How do end systems communicate with each other?

- Who send/recv msg to/from network? Processes (进程)
- Where does process send/recv msg to/from? Socket (套接字)

Processes within same host communicate using inter-process communication (defined by OS)

Processes in different hosts communicate by exchanging messages across the computer network

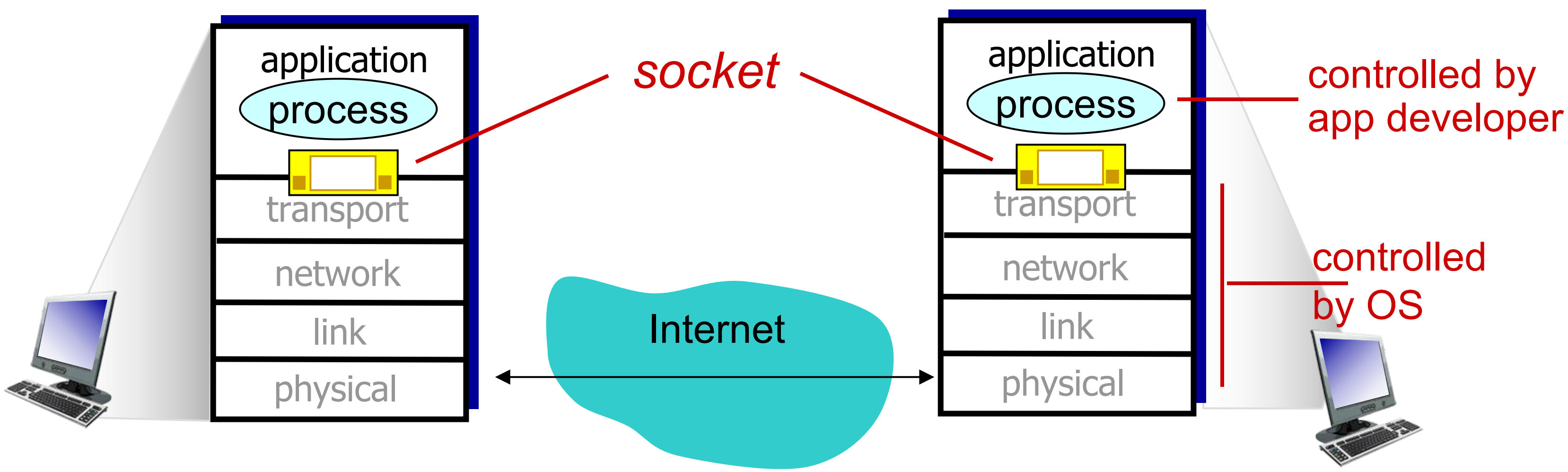
client process: process that initiates communication

server process: process that waits to be contacted

- Client-server architecture
- P2P architectures have client processes & server processes

Interface between Process and Computer Networks: Sockets

- Process sends/receives messages to/from the network through socket
- Socket is analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure (on other side of door) to deliver message to socket at receiving process



The control that the application developer has on the transport-layer side is

- The choice of transport protocol, e.g., UDP, TCP
- Perhaps, the ability to fix a few transport-layer parameters

Addressing Process: IP and Port number

To receive messages, sockets must be identified by

- The address of the host: IP address
- An identifier that specifies the receiving process/socket： port numbers

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

Port numbers:

HTTP server: 80

mail server: 25

Example： send HTTP message to gaia.cs.umass.edu web server:

- IP address: 128.119.245.12
- port number: 80

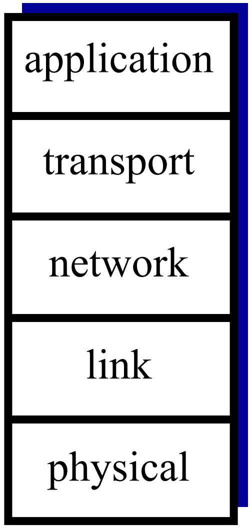
How to choose transport service?

When you develop an application:

- Applications have different requirements
- You must choose one of the available transport-layer protocols (e.g., UDP, TCP):

Reliable data transfer

- delivered correctly, completely, in proper order
- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss



How to choose transport service?

Throughput

- Bandwidth-sensitive applications: require minimum amount of throughput to be "effective", *r* bits/sec
  - E.g., multimedia
- Elastic applications: use whatever throughput they get
  - E.g., E-mail, file transfer, web transfer

Timing

- Real-time applications: some apps require low delay to be "effective"
  - E.g., Internet telephony, interactive games

Security

- encryption, data integrity, …

Transport service requirements: common apps

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
interactive games	loss-tolerant	few kbps up	yes, 100's msec
text messaging	no loss	elastic	yes and no

Internet transport protocols services

When you create a new network application for the Internet, one of the first decisions you have to make is whether to use UDP or TCP

TCP service:

- *connection-oriented:* setup required between client and server processes
  - TCP connection; full-duplex
- *reliable transport* between sending and receiving process
  - Without error; in proper order; no duplicate bytes
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantee, security

UDP service:

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

Q: Why is there a UDP?

UDP is commonly used in **time-sensitive communications** where occasionally dropping packets is better than waiting.

Internet apps: application, transport protocols

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

Creating a network app

To build a network application - Application layer:

Q1: Which architecture? client-server or peer-to-peer?

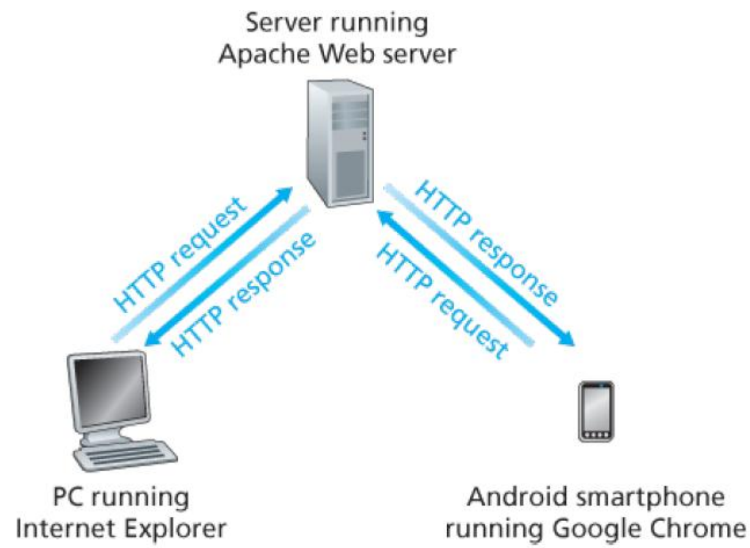
Q2: Which transport layer protocol to choose, e.g., TCP? UDP?

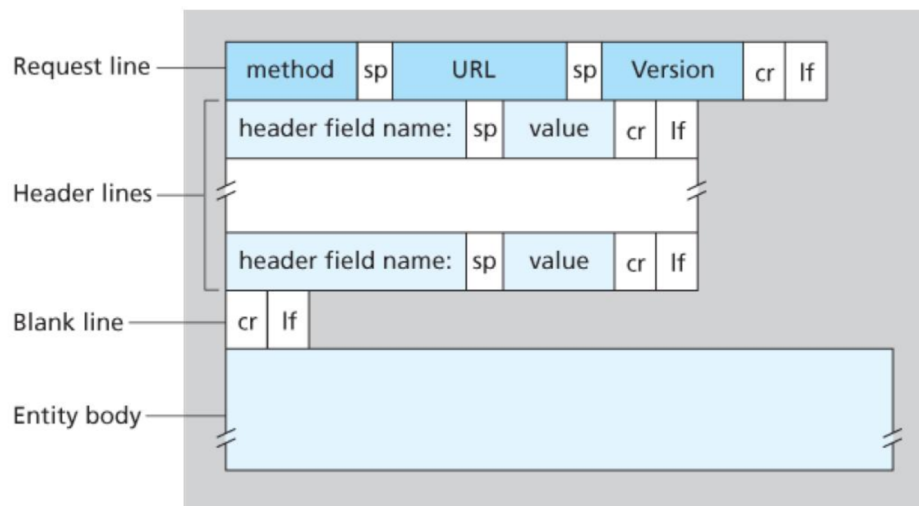
Q3: Which protocol to follow? HTTP for web? SMTP for email? Or even your own designed protocol?

- What are defined in application-level protocols?

App-layer protocol defines

- types of messages exchanged,
  - e.g., request, response
- message syntax (语法):
  - what fields in messages & how fields are delineated
- message semantics （语义）
  - meaning of information in fields
- rules for when and how processes send & respond to messages





Application-Layer Protocols

An application-layer protocol is one piece of a network application.

For example, the Web is a client-server application that allows users to obtain documents from Web servers on demand.

- a standard for document formats (that is, HTML),
- Web browsers (for example, Firefox and Microsoft Internet Explorer)
- Web servers (for example, Apache andMicrosoft servers)
- an application-layer protocol

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

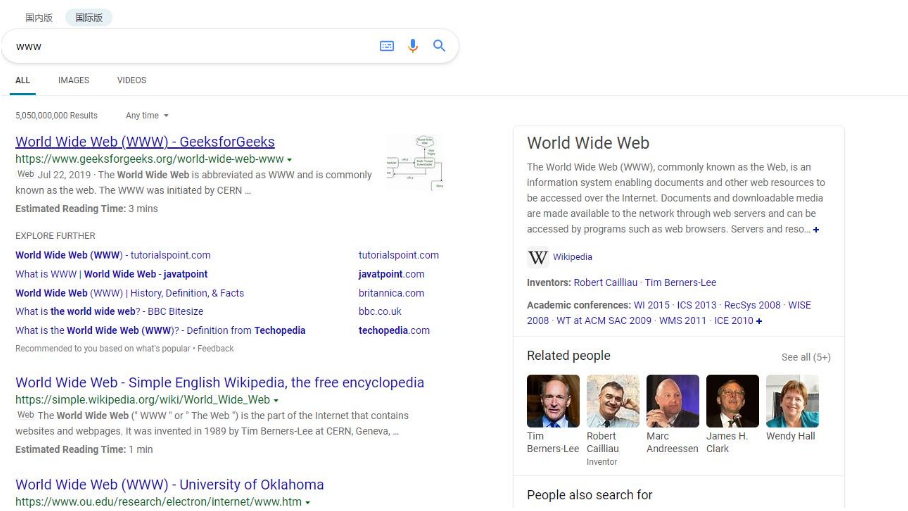
2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks

2.7 socket programming with UDP and TCP

Web



Searching engine

Web-based email

Video streaming platforms

Social networks

Web

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>Example</title>
  </head>
  <body>
    <p>See info about <a href="">dogs</a> or
    <a href="">cats</a></p>
  </body>
</html>
```

- *web page* consists of *objects*
  - object can be HTML file, JPEG image, Java applet （小程序）, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects*
  - E.g., a HTML text and five JPEG images
- each object is addressable by a *URL,* e.g.,

$$\underbrace{\texttt{www.sustc.edu.cn}}_{\text{host name}}\underbrace{\texttt{/resources/cn/image/p27.png}}_{\text{path name}}$$

HTML: hypertext markup language

HTTP: hypertext transfer protocol

HTTP and Web

Web

- client-server architecture
- use HTTP as its application layer protocol

HTTP (hypertext transfer protocol) defines

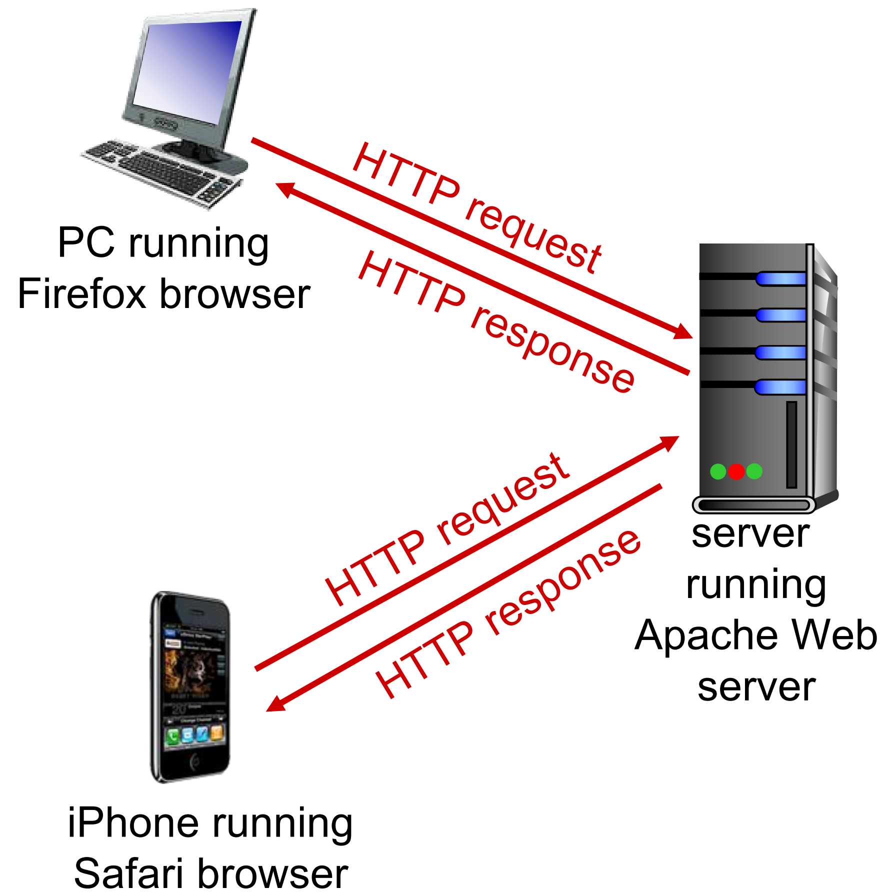
- **HTTP request:** how Web clients request Web pages from Web servers and
- **HTTP response:** how servers transfer Web pages to clients

HTTP and Web

Client-server architecture:

client: browser that requests, receives, (using HTTP protocol) and "displays" Web objects

server: Web server sends (using HTTP protocol) objects in response to requests



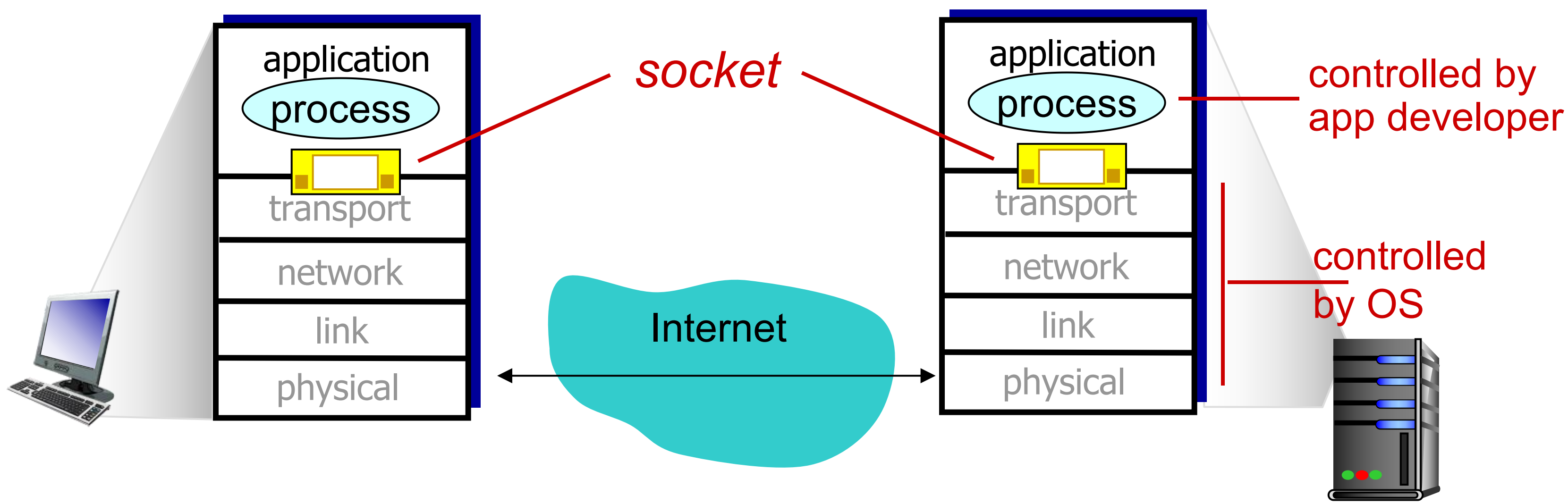
HTTP Outline

- HTTP Overview
  - HTTP runs over TCP
  - HTTP is stateless
  - Persistent and non-persistent connection
- Request and response messages
- Cookies
- Web caching

HTTP overview: TCP

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 need not worry about lost data or the details of how TCP recovers from loss or reordering of data.

HTTP overview (continued)

HTTP is "stateless"

- Server maintains no information about past client requests
- If a client asks for the same object twice, the server resends the object.

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

Should each request/response pair be sent over a *separate* TCP connection, or should all of the requests and their corresponding responses be sent over the *same* TCP connection?

non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

persistent HTTP

- multiple objects can be sent over single TCP connection between client and server
- default mode

Non-persistent HTTP

suppose user enters URL:
`www.someSchool.edu/someDepartment/home.index`

(contains 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 (cont.)

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

time

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

(As the browser receives the Web page, it displays the page to the user. HTTP has nothing to do with how a Web page is interpreted by a client.)

Non-persistent HTTP: each TCP connection transports exactly one request message and one response message.

Users can configure modern browsers to control the degree of parallelism, i.e., multiple TCP in parallel.

Non-persistent HTTP: response time

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

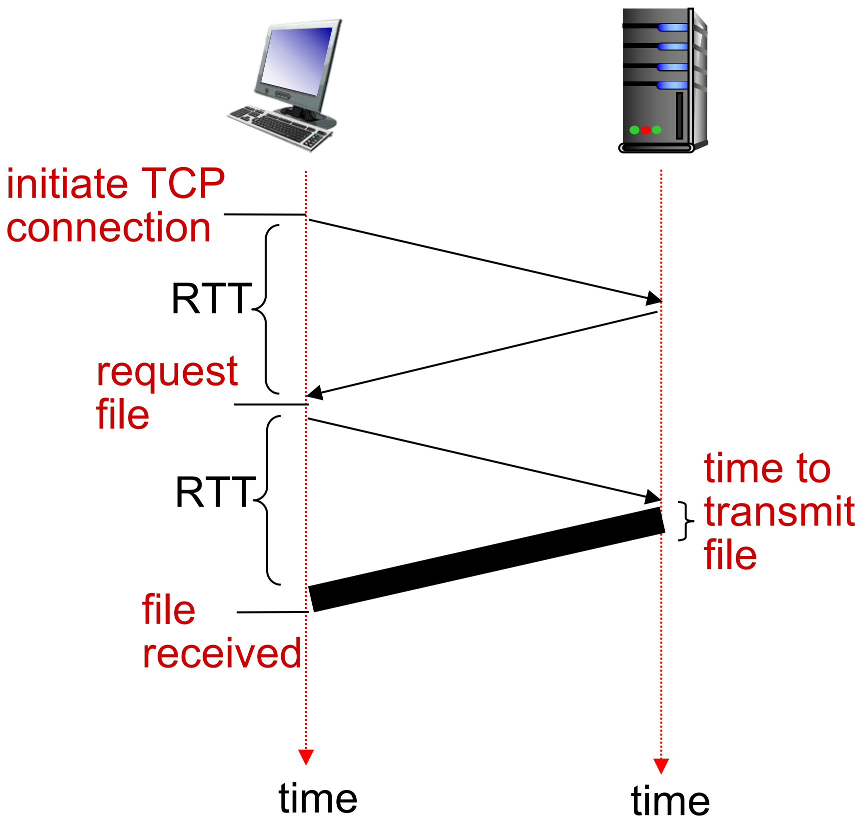
- Propagation, queuing, processing

When a user clicks on a hyperlink:

HTTP response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time
- non-persistent HTTP response time =

  2RTT+ file transmission time



Persistent HTTP

non-persistent HTTP issues:

- requires 2 RTTs per object
- OS overhead (TCP buffer, variables) for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

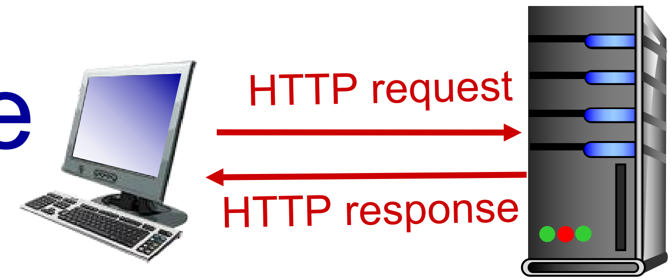
persistent HTTP:

- 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
- server closes a connection when it isn't used for a certain time
- as little as one RTT for all the referenced objects

HTTP Outline

- HTTP Overview
  - HTTP runs over TCP
  - HTTP is stateless
  - Persistent and non-persistent connection
- Request and response messages
- Cookies
- Web caching

HTTP request message



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

carriage return (回车) character

line-feed （换行） character

request line
(GET, POST,
HEAD commands)

header lines

carriage return,
line feed at start
of line indicates
end of header lines

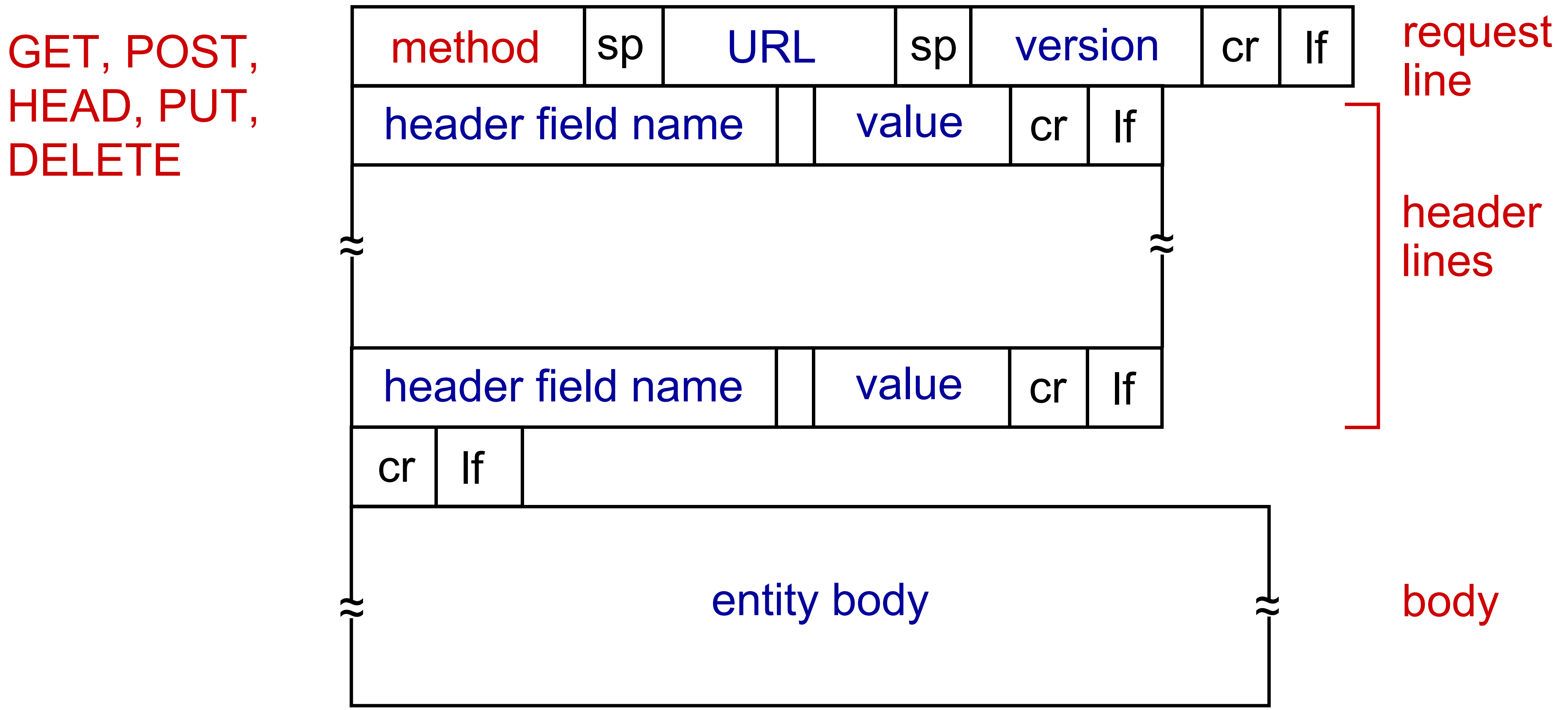
```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Connection: keep-alive\r\n
\r\n
```

Browser type: server can send different version of the same object

Connection: close

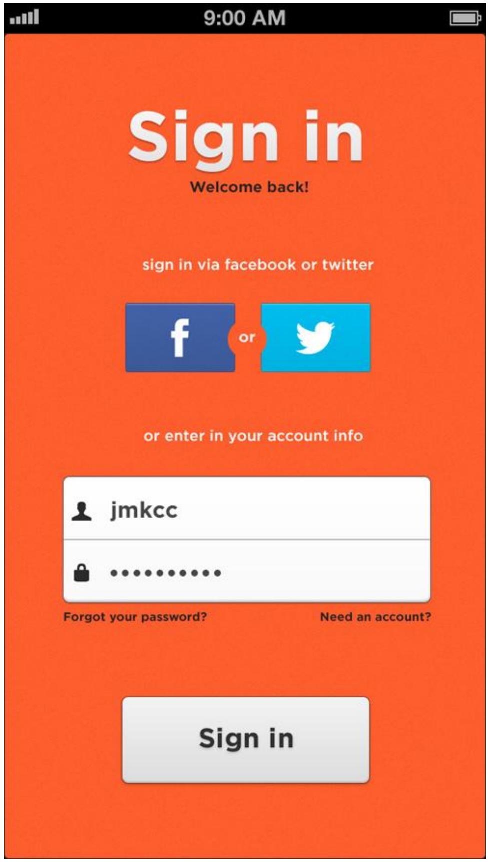
Method filed, URL field, HTTP version field

HTTP request message: general format



For example, the entity body is used with the POST method (e.g., search words to a search engine).

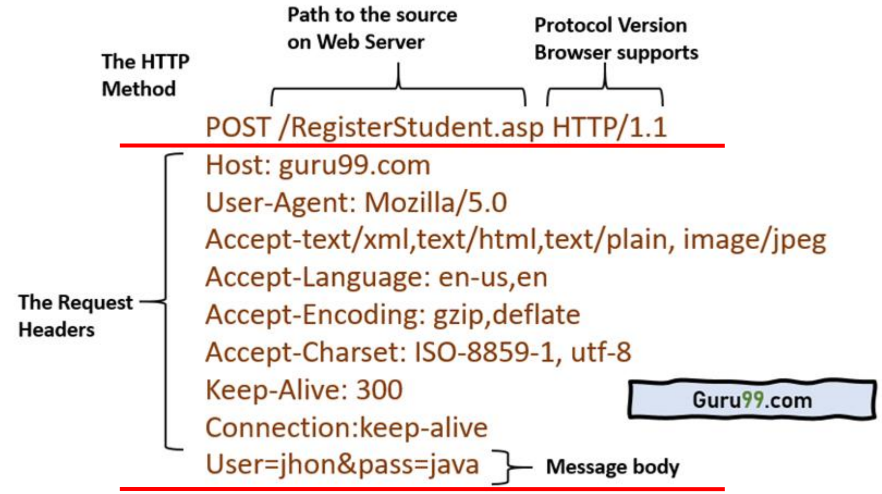
HTTP request message: general format



Uploading form input

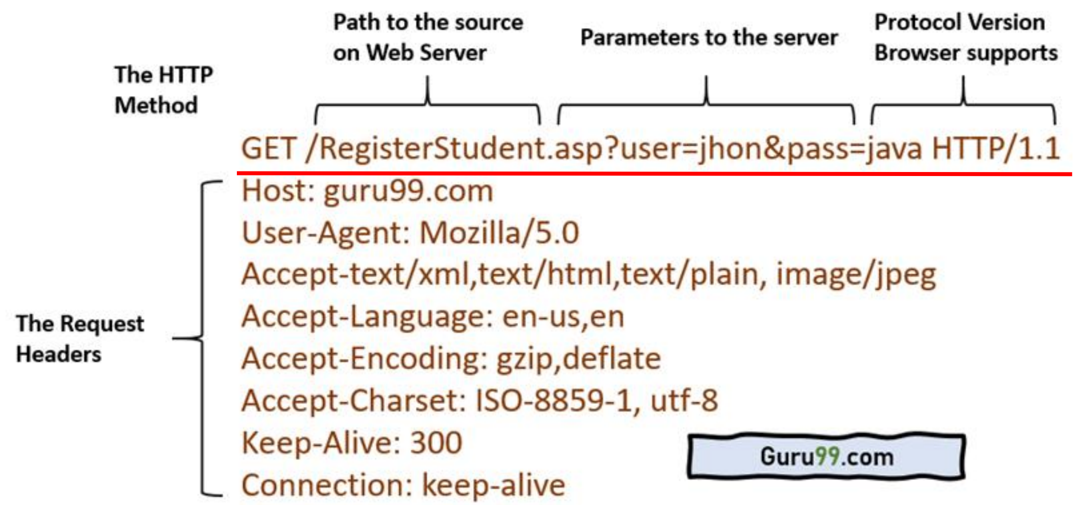
POST method:

- web page often includes form input
- input is uploaded to server in entity bodys



URL method:

- uses GET method
- input data is included in URL field of request line



Method types

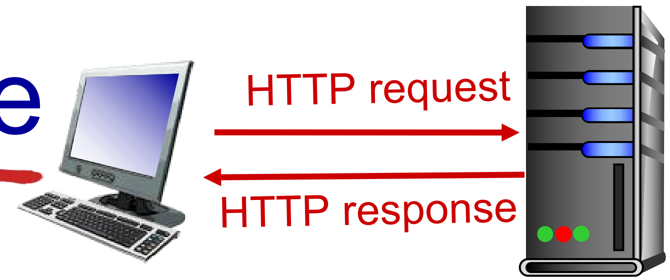
HTTP/1.0:

- GET, POST
- HEAD
  - Similar to the *GET* method
  - Server responds with an HTTP message but it leaves out the requested object
  - Used for debugging

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

HTTP response message



status line
(protocol version
status code
status message)

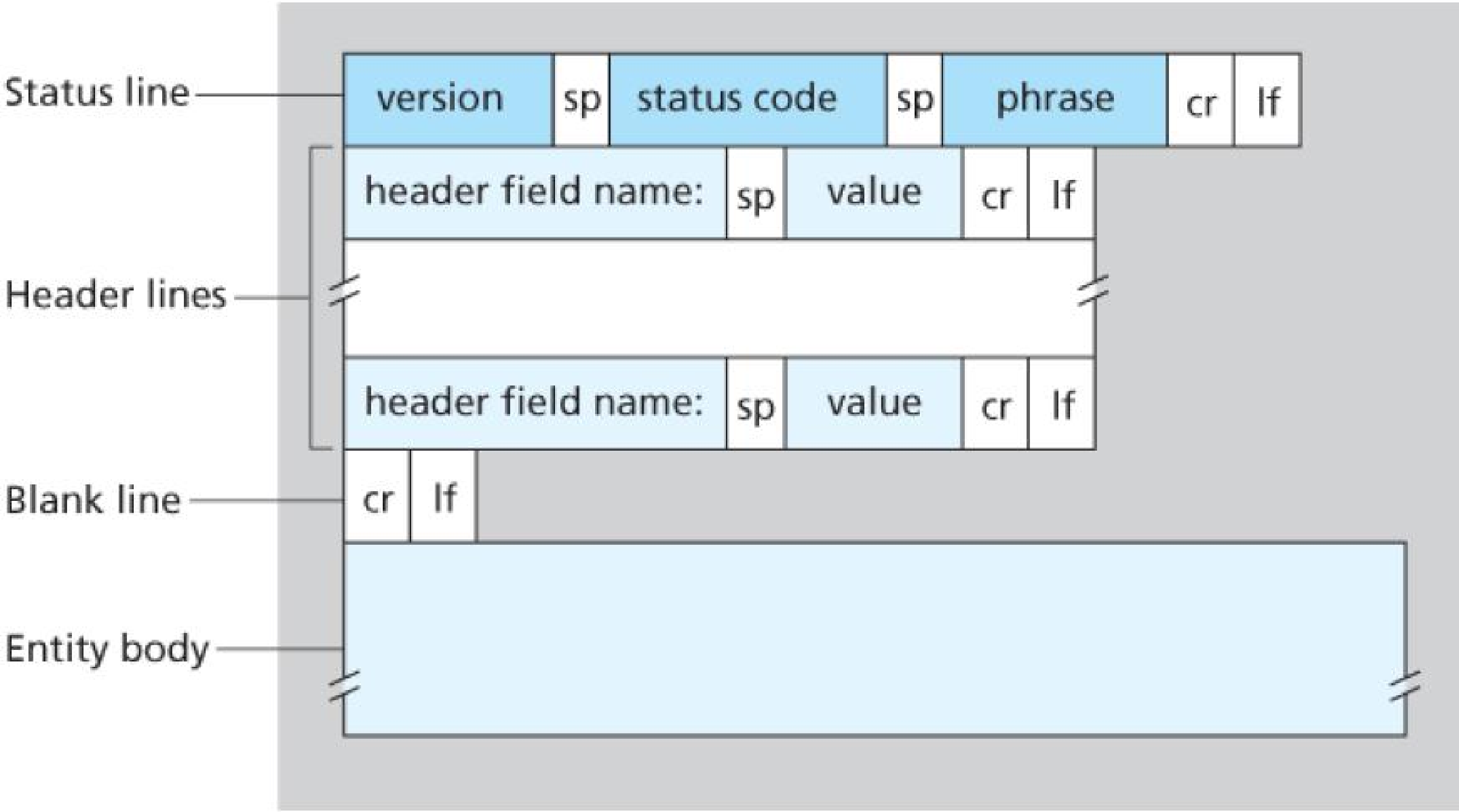
that is, the server has found, and
is sending the requested object

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

header
lines

entity body,
e.g.,
requested
HTML file

HTTP response message



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 msg

301 Moved Permanently

- requested object moved, new location specified later in this msg (Location:)

400 Bad Request

- request msg not understood by server

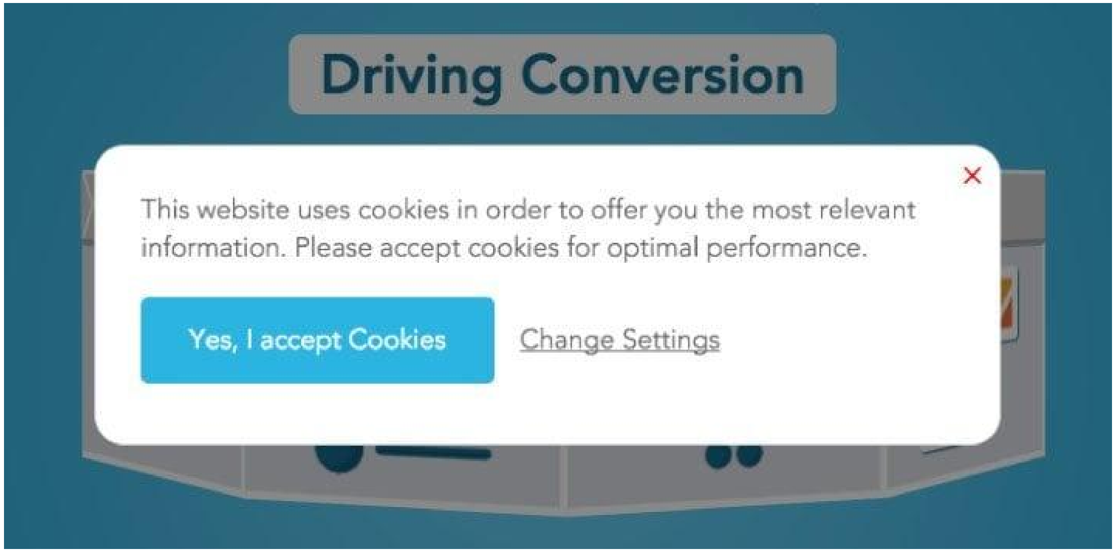
404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

HTTP Outline

- HTTP Overview
  - HTTP runs over TCP
  - HTTP is stateless
  - Persistent and non-persistent connection
- Request and response messages
- Cookies
- Web caching

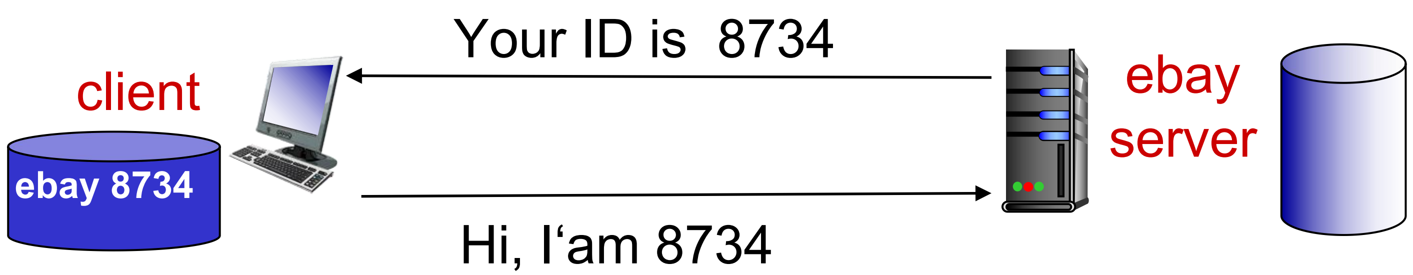


User-server state: cookies

HTTP is Stateless, and servers handle thousands of simultaneous TCP connections.

However, it is often desirable for a Web server to identify users.

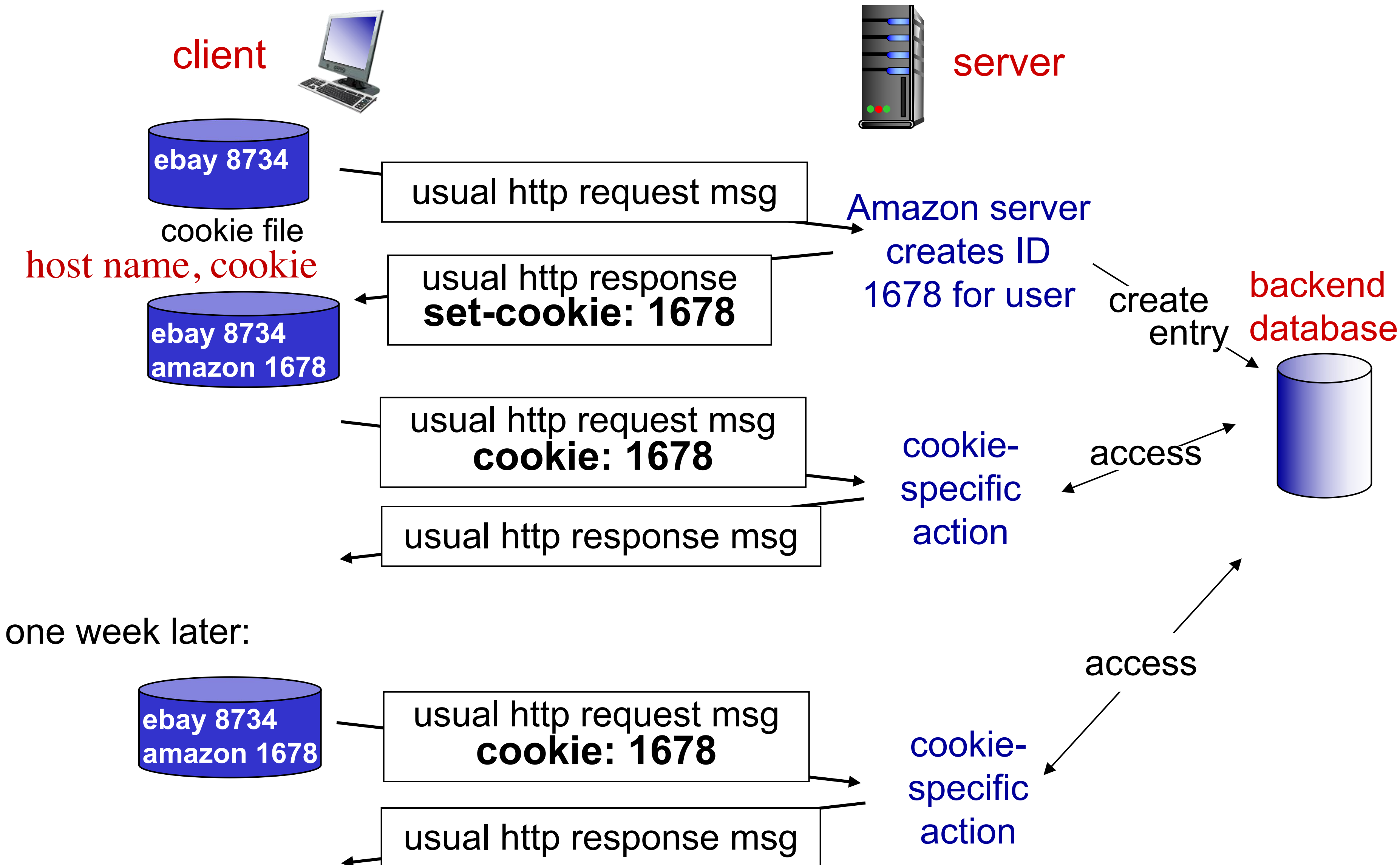
- **Web servers use cookies**



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 server

Cookies: keeping “state” (cont.)



Cookies (continued)

- Cookies are associated with web browser
- If Susan also registers herself with Amazon, the database can associate Susan's name with her identification number (cookies).

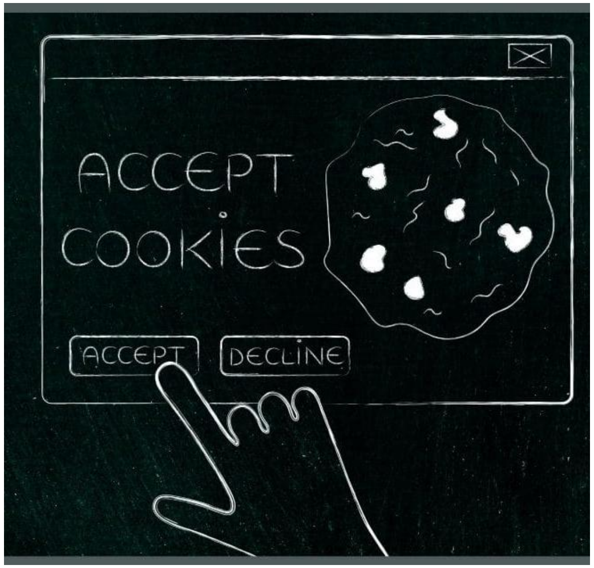
What cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session on top of stateless HTTP

aside

cookies and privacy:

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



HTTP Outline

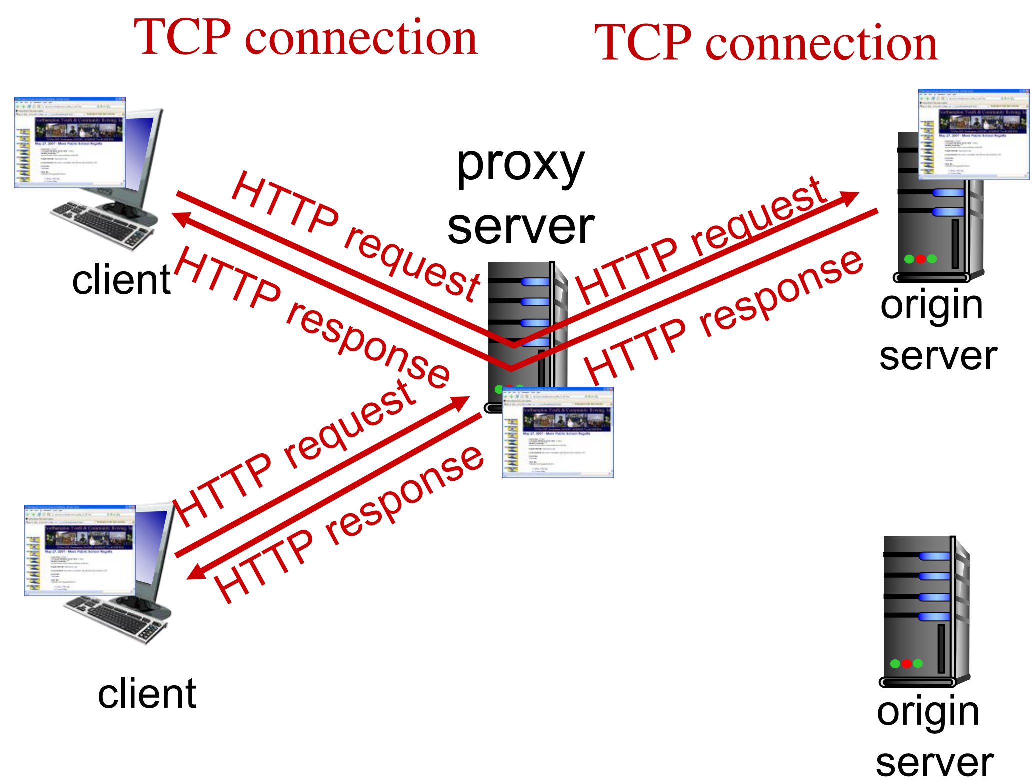
- HTTP Overview
  - HTTP runs over TCP
  - HTTP is stateless
  - Persistent and non-persistent connection
- Request and response messages
- Cookies
- Web caching

Web caches: proxy （代理） server

goal: satisfy client request without involving origin server

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 (Proxy server) 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 (bottleneck bandwidth)
- 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: 1M bits
- avg request rate from browsers to origin servers:15 requests/sec
- avg data rate to all browsers: 15 Mbps
- RTT from router A to any origin server: 2 sec → "Internet delay"
- access link rate: 15.4 Mbps

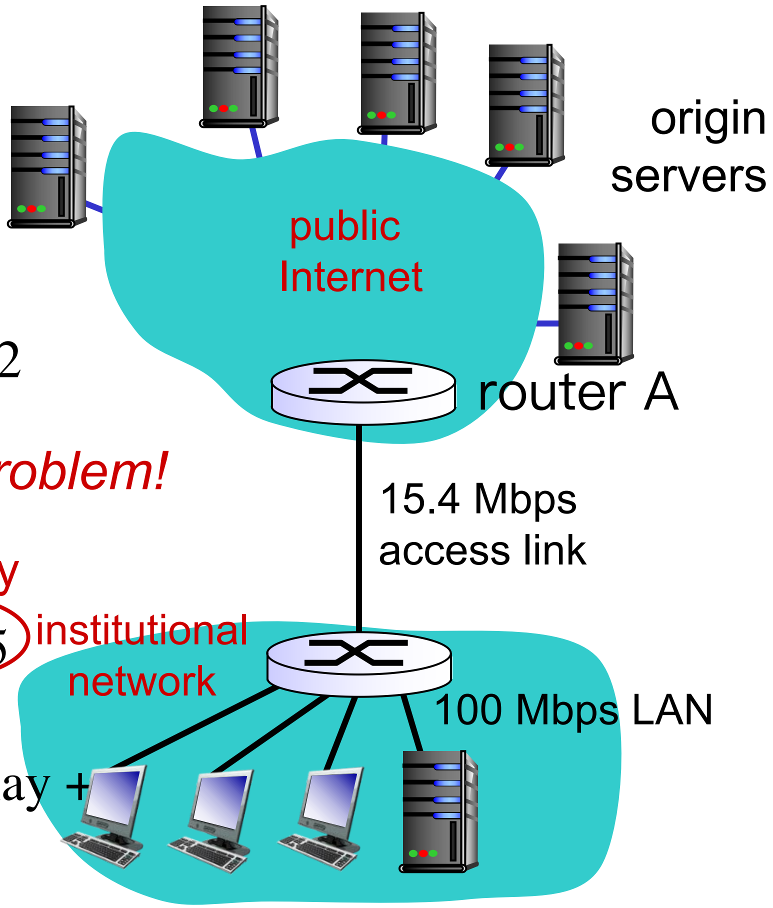
Consequences:

Traffic intensity

- LAN utilization: 15Mbps/100Mbps=0.15
- access link utilization =15/15.4= 0.974
- total delay = Internet delay + access delay + LAN delay

  = 2 sec + minutes + milliseconds

problem!



Caching example: fatter access link

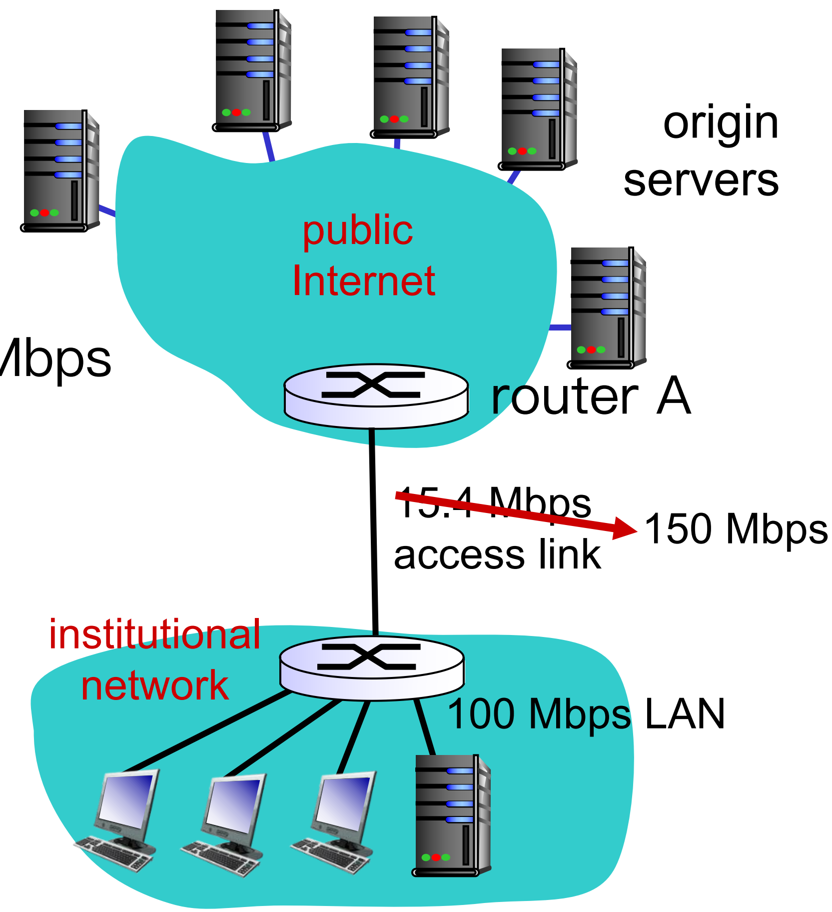
assumptions:

- avg object size: 1M bits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: ~~15 Mbps~~ 150 Mbps
- RTT from router A to any origin server: 2 sec
- access link rate: 15.4 Mbps

consequences:

- LAN utilization: 0.15
- access link utilization = ~~0.974~~ 0.1
- total delay = Internet delay + access delay + LAN delay

  = 2 sec + ~~minutes~~ + milliseconds
  milliseconds



Cost: increased access link speed (not cheap!)

Caching example: install local cache

assumptions:

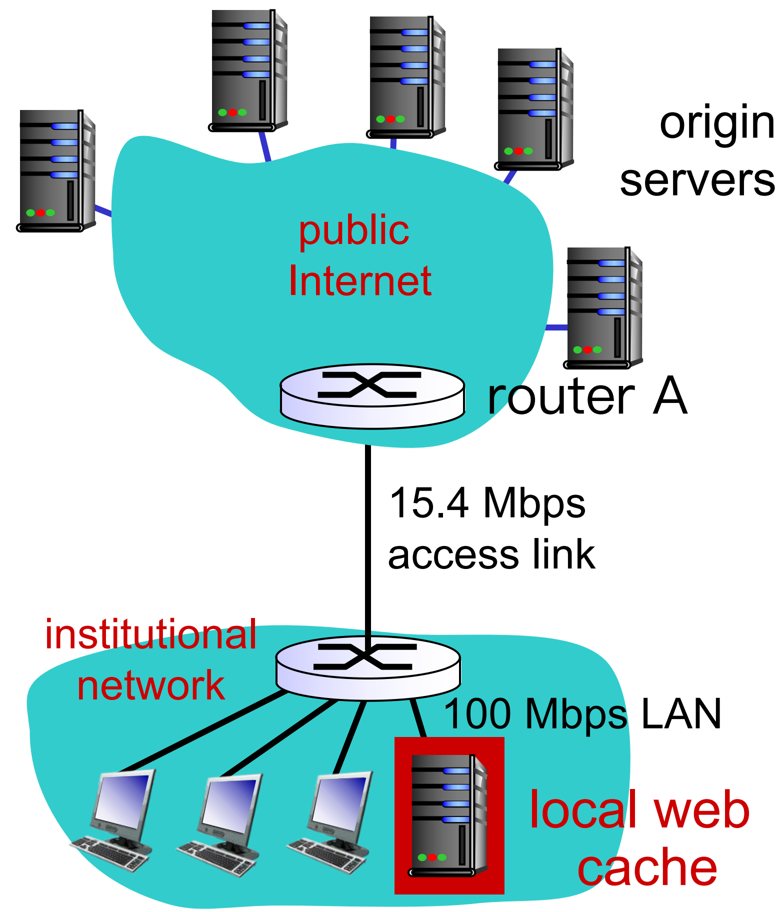
- avg object size: 1M bits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 15 Mbps
- RTT from router A to any origin server: 2 sec
- access link rate: 15.4 Mbps

consequences:

- LAN utilization: 0.15
- access link utilization = ?
- total delay = ?

How to compute link utilization, delay?

Cost: web cache (cheap!)



Hit rates: the fraction of requests that are satisfied by a cache. Typically, 0.2—0.7.

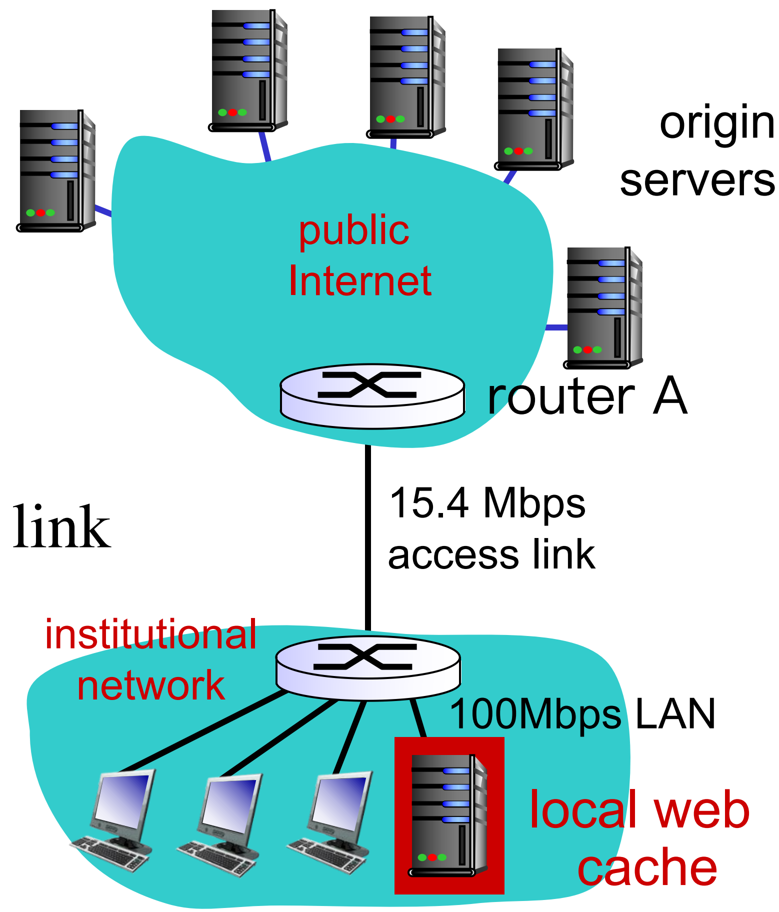
Caching example: install local cache

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*15 Mbps = 9 Mbps
  - utilization = 9/15.4 = 0.58

- Average delay
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs
  - less than with 150 Mbps link (and cheaper too!)



Typically, a traffic intensity less than 0.8 corresponds to a small delay, say, tens of milliseconds

Conditional GET

The copy of an object residing in the cache may be out-of-date:

Conditional GET

- GET method
- If-Modified-Since

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 9 Sep 2015 09:23:24
```

Goal: allows a cache to verify that its objects are up to date

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

Conditional GET

When a browser requests an object via proxy cache:

- *Proxy cache:* specify date of cached copy in HTTP request

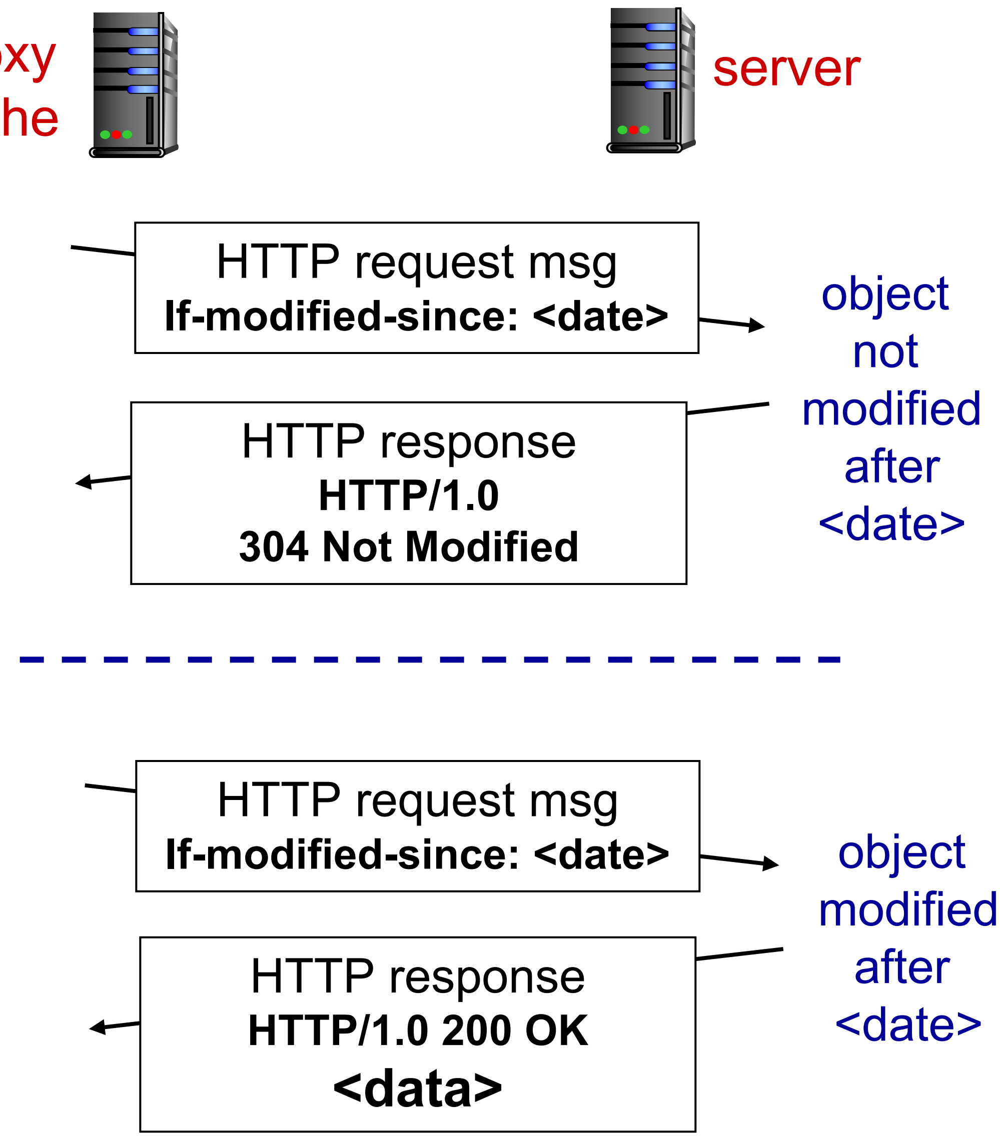
  If-modified-since: <date>

- *Server:* response contains no object if cached copy is up-to-date:

  HTTP/1.0 304 Not Modified

```
HTTP/1.1 304 Not Modified
Date: Sat, 10 Oct 2015 15:39:29
Server: Apache/1.3.0 (Unix)

(empty entity body)
```



Chapter 2: outline

2.1 Principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks

2.7 socket programming with UDP and TCP