

# Computer Networks

## HTTP

Based on Computer Networking, 4<sup>th</sup> Edition by Kurose and Ross

Stan Kurkovsky

### Web and HTTP

#### First some jargon

- **Web page** consists of **objects**
- Object can be HTML file, JPEG image, Java applet, audio file,...
- Web page consists of **base HTML-file** which includes several referenced objects
- Each object is addressable by a **URL**
- Example URL:

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

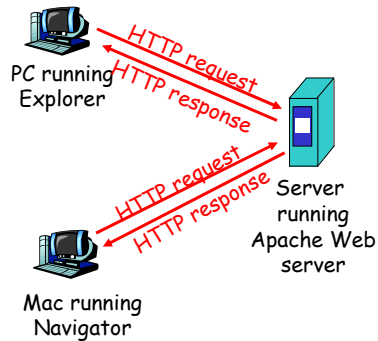
host name

path name

Stan Kurkovsky

## HTTP overview

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



Stan Kurkovsky

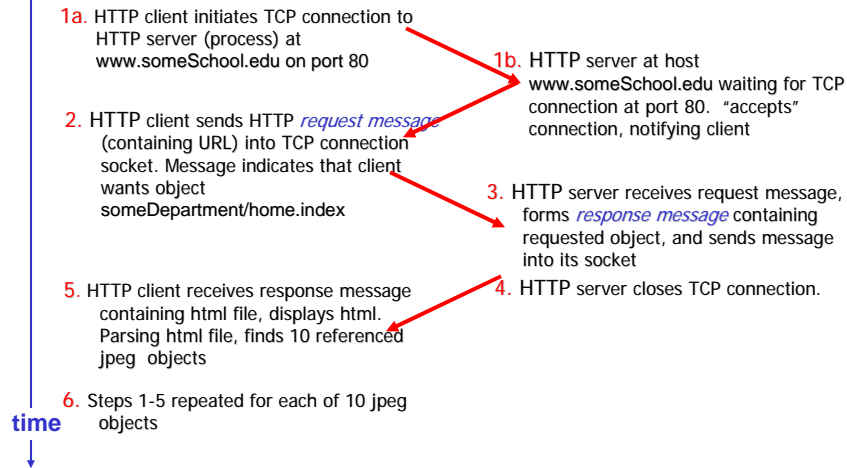
## HTTP connections

- Nonpersistent HTTP
  - At most one object is sent over a TCP connection.
  - HTTP/1.0 uses nonpersistent HTTP
- Persistent HTTP
  - Multiple objects can be sent over single TCP connection between client and server.
  - HTTP/1.1 uses persistent connections in default mode

Stan Kurkovsky

## Nonpersistent HTTP

- Suppose user enters URL `www.someSchool.edu/someDepartment/home.index` contains text, references to 10 jpeg images



Stan Kurkovsky

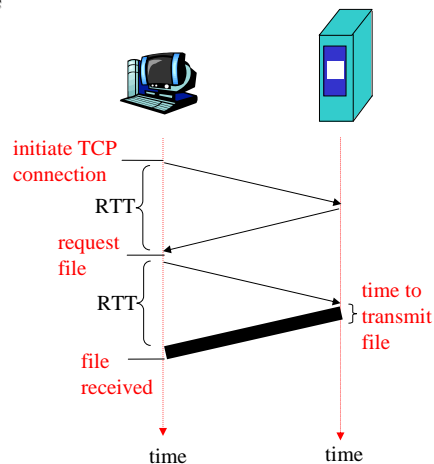
## Response time modeling

**Definition of RTT (roundtrip time):** time to send a small packet to travel from client to server and back.

### 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

$$\text{total} = 2\text{RTT} + \text{transmit time}$$



Stan Kurkovsky

## Persistent HTTP

### Nonpersistent HTTP issues:

- requires 2 RTTs per object
- OS overhead 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

### Persistent *without* pipelining:

- client issues new request only when previous response has been received
- one RTT for each referenced object

### Persistent *with* pipelining:

- default in HTTP/1.1
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

Stan Kurkovsky

## HTTP request message

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

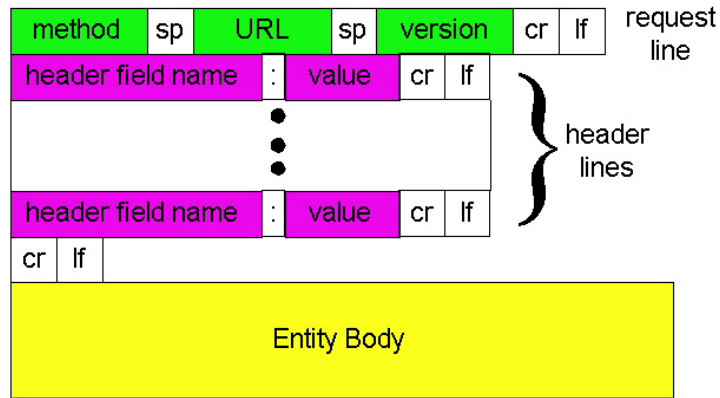
request line  
(GET, POST,  
HEAD commands) → GET /somedir/page.html HTTP/1.1

header  
lines → Host: www.someschool.edu  
User-agent: Mozilla/4.0  
Connection: close  
Accept-language: fr

Carriage return,  
line feed  
indicates end  
of message → (extra carriage return, line feed)

Stan Kurkovsky

## HTTP request message: general format



Stan Kurkovsky

## Uploading form input

### Post method:

- Web page often includes form input
- Input is uploaded to server in entity body

### URL method:

- Uses GET method
- Input is uploaded in URL field of request line:

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

Stan Kurkovsky

## Method types

### HTTP/1.0

- GET
- POST
- HEAD
  - asks server to leave requested object out of response

### 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

Stan Kurkovsky

## HTTP response message

The diagram shows an HTTP response message with three main parts: a status line, header lines, and data. Blue arrows point from text labels to the corresponding parts of the message.

status line  
(protocol  
status code  
status phrase) → HTTP/1.1 200 OK

header lines → Connection close  
Date: Thu, 06 Aug 1998 12:00:15 GMT  
Server: Apache/1.3.0 (Unix)  
Last-Modified: Mon, 22 Jun 1998 .....  
Content-Length: 6821  
Content-Type: text/html

data, e.g.,  
requested  
HTML file → data data data data data ...

Stan Kurkovsky

## HTTP response status codes

- In first line in server → client response message.
- A few 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 (Location:)
  - **400 Bad Request**
    - request message not understood by server
  - **404 Not Found**
    - requested document not found on this server
  - **505 HTTP Version Not Supported**

Stan Kurkovsky

## User-server state: cookies

- Many major Web sites use cookies
- Four components:
- 1) cookie header line of HTTP *response* message
  - 2) cookie header line in HTTP *request* message
  - 3) cookie file kept on user's host, managed by user's browser
  - 4) back-end database at Web site

What cookies can bring:

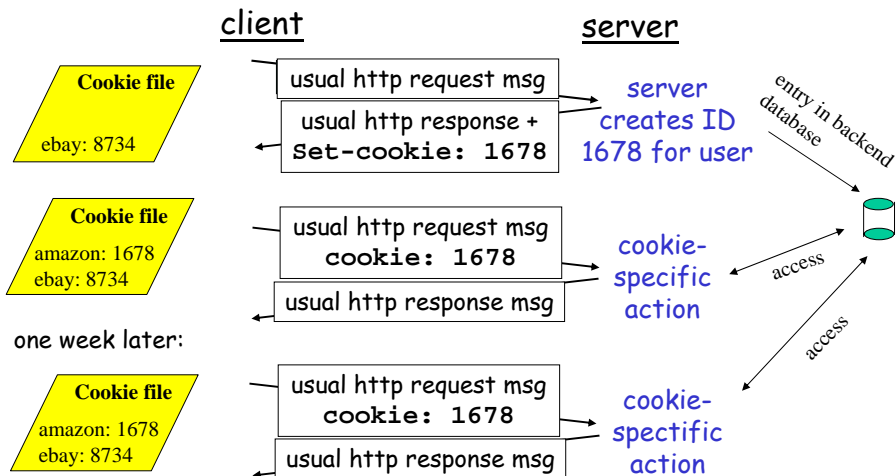
- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

Cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites
- search engines use redirection & cookies to learn yet more
- advertising companies obtain info across sites

Stan Kurkovsky

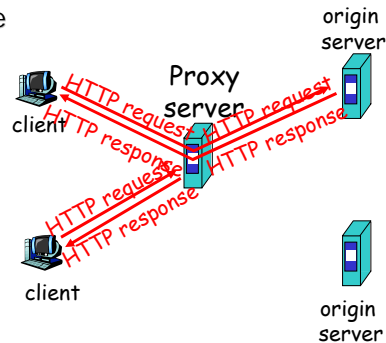
## Cookies: keeping "state"



Stan Kurkovsky

## 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
- Cache acts as both client and 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 (but so does P2P file sharing)

Stan Kurkovsky

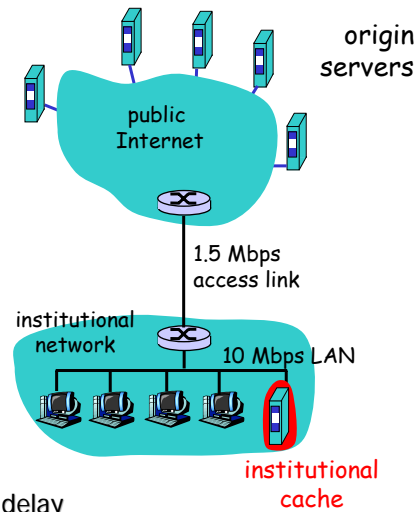
## Caching example

### Assumptions

- average object size = 100,000 bits
- avg. request rate from institution's browsers to origin servers = 15/sec
- delay from institutional router to any origin server and back to router = 2 sec

### Consequences

- utilization on LAN = 15%
- utilization on access link = 100%
- total delay  
= Internet delay + access delay + LAN delay  
= 2 sec + minutes + milliseconds



Stan Kurkovsky

## Caching example (cont)

### Possible solution

- increase bandwidth of access link to, say, 10 Mbps

### Consequences

- utilization on LAN = 15%
- utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay  
= 2 sec + msec + msec
- often a costly upgrade

### Install cache

- suppose hit rate is .4

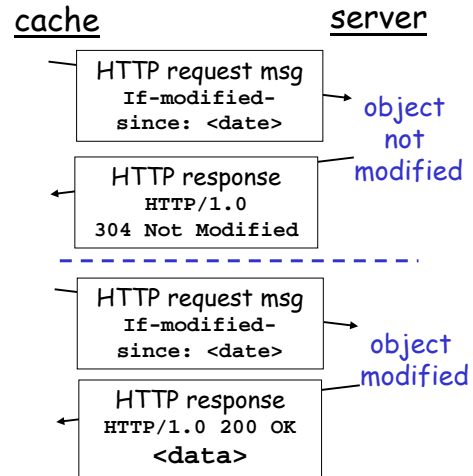
### Consequence

- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total avg delay = Internet delay + access delay + LAN delay =  
.6\*(2.01) secs + .4\*milliseconds < 1.4 secs

Stan Kurkovsky

## 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`



Stan Kurkovsky