CS 640 Introduction to Computer Networks

Lecture20

Today's lecture

- World Wide Web
  - HTML
  - HTTP
  - Caching
  - Content delivery networks

HTML Basics

- Hyper-Text Markup Language
  - A subset of Standardized General Markup Language (SGML)
  - Facilitates a hyper-media environment
    - Embedded links to other documents and applications (ftp, email, etc.)
- Documents use elements to "mark up" or identify sections of text for different purposes or display characteristics
- Mark up elements are not seen by the user when page is displayed
- Documents are rendered by browsers
- NOTE: Not all documents in the Web are HTML!
- Most people use WYSIWYG editors (MS Word) to generate HTML

HTML Example

```
<HTML>
<HEAD>
<TITLE> PB's HomePage </TITLE>
</HEAD>
<BODY>
<CENTER><IMG SRC = "bad_picture.gif" ALT = " "><BR></CENTER>
<P><CENTER><H1>UW Computer Science Department</H1></CENTER>
Welcome to my goofy HomePage!
…
<A HREF = http://www.cs.wisc.edu/~pb/mydogs_page.html> Spot's Page </A>
</BODY>
</HTML>
```

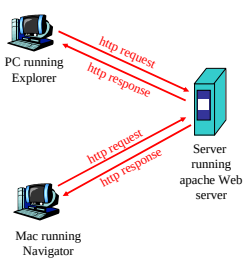

Beyond static documents

- Programs on the client produce output
  - JavaScript
  - Java applets
- Programs on the server generate HTML
  - CGI scripts
  - ASP (Active Server Pages) use Microsoft's VB
  - JSP (JavaServer Pages)
  - PHP is an open source alternative

The Web: the http protocol

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
- http1.0: RFC 1945
- http1.1: RFC 2068



The http protocol: more

http: TCP transport service:

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

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. Port 80 is default for http server.

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

3. http server receives request message, forms *response message* containing requested object (someDepartment/home.index), sends message into socket

time

http example (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

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

time

HTTP/1.0 Network Interaction

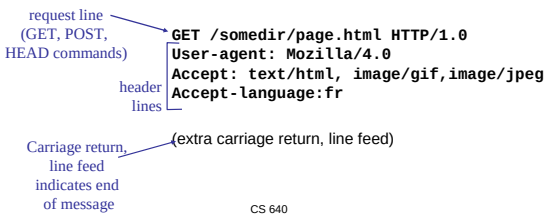
- Clients make requests to port 80 on servers
  - Uses DNS to resolve server name
- Clients make separate TCP connection for each URL
  - Some browsers open multiple TCP connections
    - Netscape default = 4
- Server returns HTML page
  - Many types of servers with a variety of implementations
  - Apache is the most widely used
    - Freely available in source form
- Client parses page
  - Requests embedded objects

HTTP/1.1 Enhancements

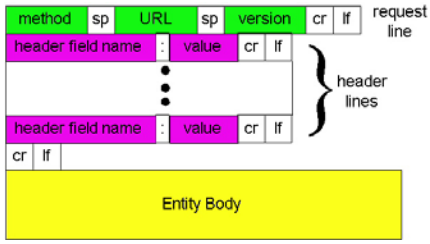
- HTTP/1.0 is a “stop and wait” protocol
  - Separate TCP connection for each file
    - Connect setup and tear down is incurred for each file
    - Inefficient use of packets
    - Server must maintain many connections in TIME_WAIT
- Mogul and Padmanabhan studied these issues in ’95
  - Resulted in HTTP/1.1 specification focused on performance enhancements
    - Persistent connections
    - Pipelining
    - Enhanced caching options
    - Support for compression

http message format: request

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



http request message: general format



http message format: response

status line (protocol status code status phrase)

header lines

data, e.g., requested html file

```
HTTP/1.0 200 OK
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 data data data data ...
```


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

Trying out http (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet www.eurecom.fr 80
```

Opens TCP connection to port 80 (default http server port) at www.eurecom.fr. Anything typed in sent to port 80 at www.eurecom.fr

2. Type in a GET http request:

```
GET /~ross/index.html HTTP/1.0
```

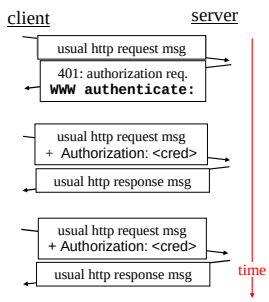
By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to http server

3. Look at response message sent by http server!

User-server interaction: authentication

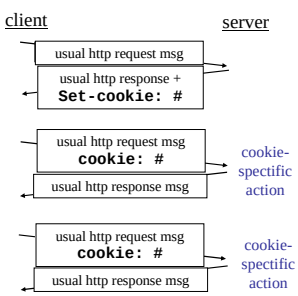
Authentication : control access to server content

- authorization credentials: typically name, password
- stateless: client must present authorization in *each* request
  - authorization: header line in each request
  - if no authorization: header, server refuses access, sends
    `WWW authenticate:`
    header line in response



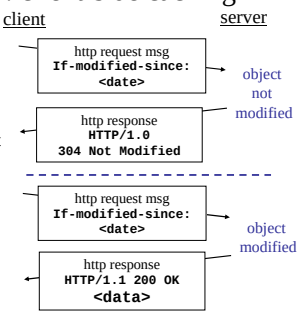
Cookies: keeping "state"

- server-generated # , server-remembered #, later used for:
  - authentication
  - remembering user preferences, previous choices
- server sends "cookie" to client in response msg
  `Set-cookie: 1678453`
- client presents cookie in later requests
  `cookie: 1678453`



Conditional GET: client-side caching

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



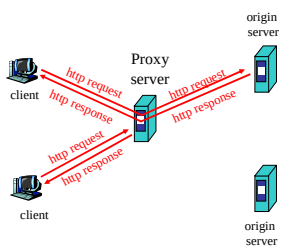
Today's lecture

- World Wide Web
  - HTML
  - HTTP
  - Caching
  - Content delivery networks

Web Caches (proxy server)

Goal: satisfy client request without involving origin server

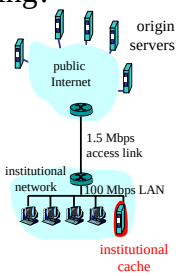
- user sets browser: Web accesses via web cache
- client sends all http requests to web cache
  - object in web cache: web cache returns object
  - else web cache requests object from origin server, then returns object to client



Why Web Caching?

Assume: cache is "close" to client

- Advantages
  - smaller response time: cache "closer" to client
  - decrease traffic to distant servers (uplink often bottleneck)
- Disadvantages
  - introduces new point of failure
  - some overhead on misses
  - does not work with dynamic personalized content



Content Delivery Networks

- e.g. Akamai, DigitalIsland, etc.
- Has its own network of caches that replicates some content of the customer (e.g. cnn.com)
  - e.g. all images
  - In the index.html file all references of:

    www.cnn.com/images/sports.gif is re-mapped to www.akamai.com/www.cnn.com/images/sports.gif
    - Server domain name: www.akamai.com
    - File: www.cnn.com/images/sports.gif

Content Delivery Networks

- Client downloads www.cnn.com/index.html
- Next tries to resolve www.akamai.com
- When local nameserver of client tries to resolve www.akamai.com
  - DNS server of Akamai will identify one of its caches that is closest to the local nameserver of client
  - Expectation is that the client is close to its local nameserver
- Client connects to the nearby cache and gets the image