CMPE 150: Introduction to Computer Networks

Dr. Chane L. Fullmer
chane@cse.ucsc.edu
Homework Assignments

Homework assignment #5
Chapter two
Due June 5th – This Thursday
(Optional) Class Project

- Network programming project
  - In lieu of taking final examination
  - Or, just a wildcard – for mid-term or final

- Goal:
  - Build an FTP client/server from scratch
  - Using ‘C’ language

- Details on web page.. now...

Due by tomorrow
(June 4th)
Class Final Exam

- Final exam
  - Three hours
  - 40 – 50 questions -- comprehensive
  - Multiple choice as the midterm
  - Scantron – bring your sheet and pencils

- **Wednesday – June 11th**  8:00 – 11:00am
  - Eight short days.. Tick, tick, tick...
Class Review Lecture

- Thursday – June 5\textsuperscript{th}
  - (last class meeting)
- Coverage of high points of the quarter
  - Much of what might appear on the final exam.....
CMPE 150: Introduction to Computer Networks
Set 18:

Application-level Protocols – continued
HTTP
Background

- Hypertext Transport Protocol.
- Created by Tim Berners-Lee at CERN.
  - Physicists, not computer scientists!
  - Share data from physics experiments.
- Standardized and much expanded by the IETF.
Web and HTTP

- 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
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

- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068
HTTP Overview (continued)

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

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
Nonpersistent HTTP

Suppose user enters URL
www.someSchool.edu/someDepartment/home.index

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
Nonpersistent HTTP (cont.)

4. HTTP server closes TCP connection.


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

Nonpersistent HTTP issues:
- requires 2 RTTs per object
- OS must work and allocate host resources for each TCP connection
- but 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 are sent over 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
HTTP request message

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

  ```
  GET /somedir/page.html HTTP/1.1
  Host: www.someschool.edu
  User-agent: Mozilla/4.0
  Connection: close
  Accept-language: sp
  ```

  (extra carriage return, line feed)
HTTP request message: general format

```
method sp URL sp version cr lf

header field name : value cr lf

... 

header field name : value cr lf

Entity Body
```
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
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
HTTP response message

status line (protocol status code status phrase)

HTTP/1.1 200 OK
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 data data data data data data ...

data, e.g., requested HTML file

header lines
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
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: authorization

Authorization: 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

```
client

usual http request msg

401: authorization req.

WWW authenticate:

usual http request msg
+ Authorization: <cred>

usual http response msg

usual http request msg
+ Authorization: <cred>

usual http response msg

server

time
```
Cookies: keeping “state”

Many major Web sites use cookies

**Four components:**

1) cookie header line in the HTTP response message
2) cookie header line in HTTP request message
3) cookie file kept on user’s host and managed by user’s browser
4) back-end database at Web site

**Example:**

- Susan access Internet always from same PC
- She visits a specific e-commerce site for first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID
Cookies: keeping “state” (cont.)

Client

Server

server creates ID 1678 for user

Cookie file
amazon: 1678
ebay: 8734

usual http request msg
usual http response +
Set-cookie: 1678

usual http request msg
cookie: 1678
usual http response msg

usual http request msg
cookie: 1678
usual http response msg

one week later:

Cookie file
amazon: 1678
ebay: 8734

Cookie file
amazon: 1678
ebay: 8734

cookie-specific action

cookie-specific action

entry in backend database

access

access
Cookies (continued)

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
**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`

- **server**: response contains object if cached copy is modified:
  
  `HTTP/1.0 200 OK <data>`
Review Thursday
Web caching
Caching architectures

- **Proxy Caches**
  - Serve a specific client population.
  - Can store recently accessed documents.
    - Lower latency for the end user.
    - Better use of wide area bandwidth by avoiding repeated transfers of recently-used information.

- **Cooperative caching.**
  - Multiple communicating caches.
  - ... to increase hit rate and reduce access latency.
  - Approaches to cooperative caching
    - Hierarchical caches.
    - Non-hierarchical routable cache systems.
Why Web Caching

Problems:

- Content server is overwhelmed by clients.
- Content server may be very far from client.
Web Caching with Client-based Hashing

Limitations:
• Client is modified with hash
• One cache per object
• Search uses URLs
Web Caching with Client-based Hashing

Added Limitation:
- Content server is hot spot
Web Caching with Hierarchies

Limitations:
- Caches are *manually* organized
- Hierarchy defined per system or per URL
- Hierarchy is independent of congestion and load

Advantage:
Content server is buffered from requests
Web Caching with Hierarchies

Limitations:

- Added variable latency retrieving object
- Congestion due to search among siblings
- Client must choose the cache (static choice or require performance data at client)
Web Caching with DNS-based Hashing

Limitations:

- Cache chosen with respect to local DNS, not client
- Latency and congestion limitations incurred by cache hierarchy
Web Caching with DNS-based Hashing

Limitations:

- Cache chosen with respect to local DNS, not client
- Latency and congestion limitations incurred by cache hierarchy
Summary of Limitations

- Special DNS servers or modified browsers needed
- Special DNS servers receive all requests
- Search for cache-client match must be done at the time the request is received
- Either hashing is independent of network congestion or performance information must be sent to special DNS servers or clients
- Caches must be (manually) organized into a hierarchy
- DNS-based hashing optimizes with respect to local DNS, not the client
- Latencies incurred getting responses can be long
- Hierarchy of caches incurs additional congestion
## Transport service requirements of common apps

<table>
<thead>
<tr>
<th>Application</th>
<th>Data loss</th>
<th>Bandwidth</th>
<th>Time Sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>file transfer</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>e-mail</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>Web documents</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>real-time audio/video</td>
<td>loss-tolerant</td>
<td>audio: 5kbps-1Mbps</td>
<td>yes, 100’s msec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>video: 10kbps-5Mbps</td>
<td></td>
</tr>
<tr>
<td>stored audio/video</td>
<td>loss-tolerant</td>
<td>same as above</td>
<td>yes, few secs</td>
</tr>
<tr>
<td>interactive games</td>
<td>loss-tolerant</td>
<td>few kbps up</td>
<td>yes, 100’s msec</td>
</tr>
<tr>
<td>instant messaging</td>
<td>no loss</td>
<td>elastic</td>
<td>yes and no</td>
</tr>
</tbody>
</table>