HTTP

CS 640, Lecture 9

Lecture outline

- Overview of http
- Http message formats
- Web caching

HyperText Transfer Protocol

- The web's application level protocol
  - HTTP 1.0 (RFC 1945), HTTP 1.1 (2068)
- Runs on top of TCP, uses well-known port 80
- Request-response interaction
  - Stateless: there is no per-session state at the server (requests treated independently)
  - Protocols that maintain state are complex
    - Managing state adds complexity
    - If server/client crashes, their views of "state" may be inconsistent, must be reconciled

http example

User enters URL `www.someSchool.edu/someDept/index.html`

1a. browser initiates TCP connection to http server (process) at port 80 of www.someSchool.edu.

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

2. browser sends http *request message* (containing URL) into TCP connection socket

3. http server receives request, forms *response message* with requested object (someDepart/index.html), 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 use 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 – open source
    - IIS (Internet Information Services) – Microsoft
- Client parses page
  - Requests embedded objects

HTTP/1.1 Enhancements

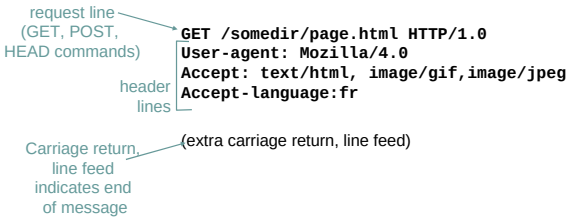
- HTTP/1.0 is a “stop and wait” protocol
  - Separate TCP connection for each file
    - Connect setup and tear down is incurred for each file
    - Inefficient use of packets
    - Server must maintain many connections in TIME_WAIT
- Mogul and Padmanabhan studied these issues in ’95
  - Resulted in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

Lecture outline

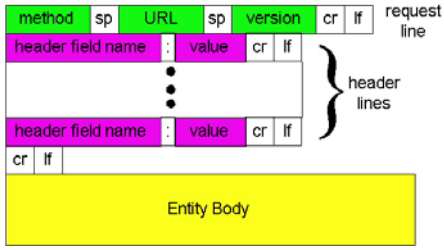
- Overview of http
- Http message formats
- Web caching

http message format: request

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



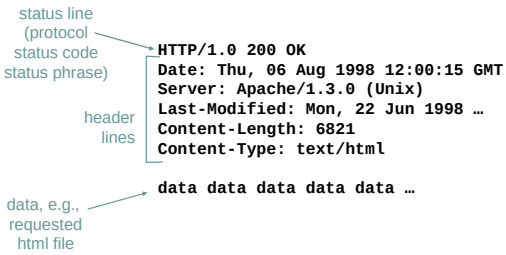
http message format: request



Sending data to server (forms)

- As part of URL, using the GET command
  - File name followed by ?
  - Fields separated by &
  - = between each field and its value
  - URL-encoding: %3d for =, %26 for &, etc.
  - Should be used only when idempotent (causes no persistent changes in application state at server)
- Inside request body using POST
  - Can also be used for non-form data (e.g. XML)

http msg. format: response



http response status codes

Start of first line in response from server to client.

`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

Try out http (client side)

1. Telnet to your favorite Web server:

```
telnet www.cs.wisc.edu 80
```

Opens TCP connection to port 80 (default http server port) at www.cs.wisc.edu. Anything typed in sent to port 80 at www.cs.wisc.edu/

2. Type in a GET http request:

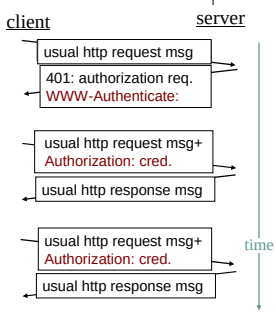
```
GET /~estan/examples/ HTTP/1.0
```

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!

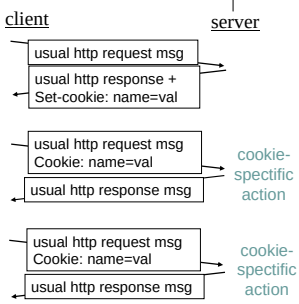
User-server interaction: authentication

- Authentication: control of access to server content
- Authorization credentials: typically name, password
- Http stateless – client must present authorization credentials 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 string later used for:
  - Authentication
  - Remembering user preferences, previous choices
- Server sends “cookie” to client in response
- Client presents cookie in later requests
- Privacy concerns

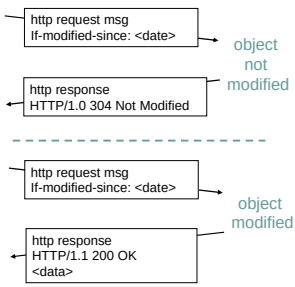


Lecture outline

- Overview of http
- Http message formats
- Web caching

Conditional GET: client-side caching

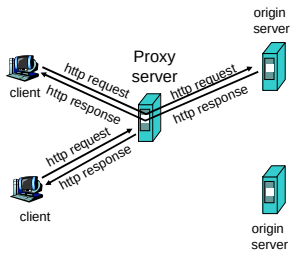
- Goal is not to send object if client has up-to-date cached version
- Client specifies date of cached copy in request
  `If-modified-since: <date>`
- Server response contains no object if cached copy is up-to-date:
  `HTTP/1.0 304 Not Modified`



Web Caches (proxy server)

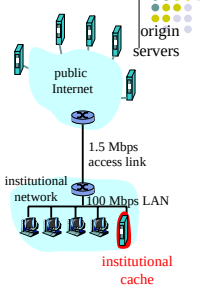
Goal: satisfy client request without involving origin server

- User sets browser: Web accesses via web cache
- Client sends all http requests to web cache
  - If object in web cache, it is returned to client
  - Otherwise web cache requests object from origin server, then returns object to client



Why Web Caching?

- Assuming cache close to client
- Advantages
  - Smaller response time
  - 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
- Decreasing popularity



Content Delivery Networks

- e.g. Akamai, Digital Island, etc.
- Has its own network of caches that replicates some content of the customer (e.g. cnn.com)
  - Typically images, sometimes HTML also
  - In the index.html file all references of: www.cnn.com/images/sports.gif 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 close to the local nameserver of client
  - Expectation is that the client is close to its local nameserver
- Client gets image from the nearby cache