WWW - HTTP/1.1

- 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, 2616
HTTP Connections

- 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
Raw 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
- Cookies maintain state
Non-persistent and Persistent Connections

• HTTP 1.0 uses non-persistent connections
  – Browser makes connection and request for each object
  – Server parses request, responds, and closes connection
  – Each object transfer suffers from slow start

• HTTP/1.1 uses persistent connections
  – On same TCP connection: server, parses request, responds, parses new request ...
  – Client sends requests for all referenced objects as soon as it receives base HTML.

HTTP Message Format: Request

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

```
GET /somedir/page.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: fr
```
(extra carriage return, line feed)
HTTP Request Message: General Format

```
method  sp  URL  sp  version  cr  lf
header field name  :  value  cr  lf

header lines

Entity Body
```
HTTP Message Format: Response

status line (protocol status code status phrase)

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 data data ...

data, e.g., requested html file
HTTP Example

Suppose user enters URL

www.someSchool.edu/someDepartment/home.index

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
HTTP Example (cont.)


6. Steps 1-5 repeated for each of 10 jpeg objects
HTTP 1.1 Persistent Connections

• In previous example, a connection is made for each request. This is bad. Why?
  – TCP 3-way hand-shake
  – TCP slow start

• In http 1.1, persistent connections were added as default behavior.

• Connection stays open unless client or server specify close when request satisfied with the header:
  `Connection: close`
HTTP 1.1 Pipelining

• Persistent connection improves performance some, but connection half session is still under-utilized.
  – Client sends request, then waits for response
  – Server sends response, the waits for new request

• **Pipelining** - send next request before previous response is received.

• Studies show that persistent connections alone perform poorly. Pipelining improves performance considerably.
HTTP Request Methods

• First word in HTTP request is method
• HTTP 1.1 methods are:
  – GET (retrieve a resource)
  – HEAD (retrieve info about resource)
  – POST (submit an html form)
  – PUT (create/modify resources)
  – DELETE (delete a resource)
  – TRACE (trace requests in proxy chains)
  – OPTIONS (request for settings from proxy or server)
HTTP Headers

• Headers appear in requests and response to provide more information about the request or response.

• Formatting follows RFC 822, Section 3.1
  Name : Value

• Header types
  • General headers
  • Request headers
  • Response headers
  • Entity headers (optional)

• Described in RFC 2616, Section 14
HTTP Response Format

- Responses have similar structure to request.
- Response format:
  - HTTP/version status-code reason-text
    - General headers
    - Response headers
    - Entity headers (optional)
    - blank line
    - resource entity (if any)
General Headers

- Can be present in request or response
  - *Cache-Control*: control resource caching
  - *Connection*: specify connection options
  - *Date*: time of request/response
  - *Transfer-Encoding*: indicate message has been transformed (“chunked” is only option)
  - *Via*: show hosts through which request or response has passed
Request Headers

• Meaningful only in requests. Important headers are:
  • **Accept**: client tells server what media types are acceptable (e.g., text/html)
  • **Host**: specifies host and port number for URL
  • **If-Modified-Since**: used when performing conditional GET
  • **User-Agent**: reports client software name and version.

Lets run HTTP and look at some headers.
Response Headers

- **Accept-Ranges**: indicates server is capable of responding to range requests (e.g., byte range requests)
- **Age**: typically, the age of a cached object
- **Location**: used to indicate re-direction destination
- **Server**: identifies the server software that generated the response
- **WWW-Authenticate**: authentication challenge sent to client with “401 Unauthorized” response
Entity Headers

- Carry meta information about the requested resource.
  - **Content-Base**: defines base URL for relative URL in this document.
  - **Content-Encoding**: indicates encoding of entity (e.g., gzip or compress)
  - **Content-Length**: length of entity in bytes
  - **Content-Type**: media type of entity (e.g., text/html, image/gif, etc)
  - **Expires**: date/time at which entity expires (is no longer valid)
  - **Last-Modified**: when entity modified on origin server
HTTP Response Status Codes

- Successful 2xx
  - 200 OK
- Redirection 3xx
  - 301 Moved permanently
  - 304 Not modified
- Client Error 4xx
  - 400 Bad request
  - 401 Unauthorized
  - 404 Not found
- Server Error 5xx
  - 504 Gateway time-out
  - 505 HTTP version not supported
Server design issues

• Functionality
  – Handle multiple clients concurrently
  – Maintaining client-specific state at a server (usually)
    » Reduce message sizes
    » Require fault-tolerance
  – Access control
    » Authenticate clients
    » Evaluate whether client is allowed access
    » Secure data

• Robustness
  – Performance
  – Reliability