CS290i - Lecture 4
HTTP protocol and its quirks

Scalable Internet Services and Systems, Winter 2002

Thorsten von Eicken
Department of Computer Science
University of California at Santa Barbara

HTTP 0.9

- **The Original HTTP as defined in 1991**
  - After-the-fact definition “as originally implemented by the World Wide Web initiative software in the prototype released”
- **Request**
  - Request: "GET", a space, the document address, CR-LF.
  - The document address will consist of a single word (ie no spaces)
  - The search functionality of the protocol lies in the ability of the addressing syntax to describe a search on a named index .
- **Response**
  - The response is a message in HTML. This is a byte stream of ASCII characters.
  - The format of the message is HTML. It also allows for plain ASCII text to be returned following the PLAINTEXT tag .
  - The message is terminated by the closing of the connection by the server.
  - Error responses are supplied in human readable text in HTML syntax. There is no way to distinguish an error response from a satisfactory response except for the content of the text.

HTTP versions

- HTTP/0.9
  - **Deprecated. Only supports one command, GET — which does not specify the HTTP version.**
  - **Does not support headers.**
  - **Does not support POST, the client can't pass much information to the server.**
- HTTP/1.0
  - **First protocol revision to specify its version in communications and still in wide use, especially by proxy servers.**
  - **Allows persistent connections when explicitly negotiated, only works well when not using proxy servers.**
- HTTP/1.1
  - **Current version; persistent connections enabled by default and works well with proxies.**
  - **Supports request pipelining, allowing multiple requests to be sent at the same time**
- HTTP/1.2
  - **The initial 1995 working drafts of *PEP — an Extension Mechanism for HTTP*. PEP later became subsumed by the experimental RFC 2774 — *HTTP Extension Framework*.**

HTTP basics

- **URL = Uniform Resource Locator**
  - Resource ≠ file
- **Requests and responses**
  - Initial request/response line
    - `GET /path/to/file/index.html HTTP/1.0`
    - `HTTP 404 Not found`
  - Zero or more header lines
  - Blank line (CRLF)
  - Optional message body
- **TCP transport**
  - Client opens TCP connection to server

HTTP's 8 methods

- **GET**
  - Requests a representation of the specified resource.
- **HEAD**
  - Asks for the response identical to the one that would correspond to a GET request, but without the response body.
  - Useful for retrieving meta-information written in response headers, without having to transport the entire content.
- **POST**
  - Submits data to be processed (e.g. from a HTML form) to the identified resource.
  - The data is included in the body of the request.
- **PUT**
  - Uploads a representation of the specified resource.
- **DELETE**
  - Deletes the specified resource.
- **TRACE**
  - Echoes back the received request, to see what intermediate servers are adding or changing in the request.
- **OPTIONS**
  - Returns the HTTP methods that the server supports.
- **CONNECT**
  - For use with a proxy that can change to being an SSL tunnel.

HTTP methods (cont.)

- **Safe methods**
  - GET and HEAD are defined as safe
    - they are intended only for information retrieval and should not change the state of the server
  - POST, PUT and DELETE are may modify information
  - OPTIONS and TRACE are also defined to be safe
  - Google web accelerator prefetches links and …
    - Aside: how does this affect stats?
- **Idempotent methods**
  - GET, HEAD, PUT and DELETE are defined to be idempotent
    - multiple identical requests should have the same effect as a single request

HTTP responses

- **Initial response line:**
  - `HTTP/1.0 <code> <text>`
  - Code is error code:
    - 1xx: informational
    - 2xx: success, e.g. 200 OK
    - 3xx: redirect, e.g. 301 Moved Permanently, 302 Moved Temporarily
    - 4xx: error by client, e.g. 404 Not Found
    - 5xx: error by server, e.g. 500 Server error
- **Headers:**
  - <header-name>: <value>, <value>, …
  - Example:

```
Header1: some-value-1a, some-value-1b
HEADER1: some-value-1a,
         some-value-1b
```


Host header

- **Problem: 1 server but 1000 domain names (www.thorsten.com)**
- **Solution: required host header**
  - GET /index.html HTTP/1.1
Host: www.thorsten.com
  - As opposed to assigning 1000 IP addresses to the server

HTTP 1.1 Example

- **Try a GET request**
  - ```
    # telnet www.google.com 80
    Trying 216.239.51.99...
    Connected to www.l.google.com.
    Escape character is '^]'.
    get / http/1.1
    host: www.google.com
    <ret>
    ```
- **Try a HEAD request**
  - ```
    # telnet www.google.com 80
    Trying 216.239.51.99...
    Connected to www.l.google.com.
    Escape character is '^]'.
    head / http/1.1
    host: www.google.com
    <ret>
    ```

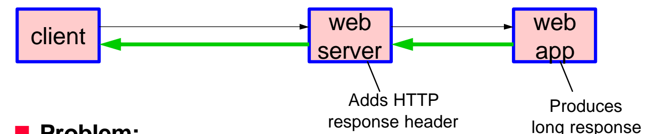

Persistent Connections

- Connections are persistent by default:
  - Send request 1, receive response 1
  - Send request 2, receive response 2
  - Note: requires well-delimited requests & responses (content-length & chunked encoding)
- Termination:
  - Server: "`Connection: close`", or close connection (e.g. idle)
  - Client: close connection (or "`Connection: close`")
  - Note: client must handle aborted connections by retrying
- Pipelining:
  - Send request 2 before response 1 recv'd
  - Strictly in-order responses
  - Careful with non-idempotent requests

Range header

- **Problem:**
  - aborted connections
  - fetching prefixes
- **Solution: specify requested byte range**
  - GET /long_song.mpg HTTP/1.1
    Range: bytes=44580–
  - GET /bigdocument.xml HTTP/1.1
    Range: bytes=10000–20000

Chunked encoding



- **Problem:**
  - Streaming dynamically generated responses
- **Solution: chunked encoding**
  - Send response in chunks
  - Each chunk prefixed by length
  - Zero-length chunk signifies end of document
  - Chunk: length_{16} CRLF data CRLF
  - (Additional headers can be sent after last chunk)

MIME types

- **Problem: what type of content can the requestor process?**
  - Accept header
    - Accept: text/plain; q=0.5, text/html, text/x-dvi; q=0.8, text/x-c Verbally, this would be interpreted as "text/html and text/x-c are the preferred media types, but if they do not exist, then send the text/x-dvi entity, and if that does not exist, send the text/plain entity."
  - Accept-charset
  - Accept-encoding
    - Accept-Encoding: compress, gzip
  - Accept-Language: da, en-gb;q=0.8, en;q=0.7
- **Content-Type**

MIME types (cont.)

- **Problem: what type of content is in the response?**
  - File name extensions are not universal…
  - Content-Type: text/plain, image/jpeg, …
  - Content-Language
  - Content-Charset
  - Content-Encoding: gzip
    - Example: content-type: text/plain content-encoding: gzip

Data Model & MIME (cont.)

- Example: What is the server supposed to respond to a request with:
  - Accept: text/plain, text/html
  - Accept-Encoding: compress
  - Transfer-Encoding: gzip, chunked
  - Range: bytes=1024-2047

HTTP types/codings

- **§14.35.1:**
  - "Byte range specifications in HTTP apply to the sequence of bytes in the entity-body (not necessarily the same as the message-body)."
- **§7.2.1:**
  - "When an entity-body is included with a message, the data type of that body is determined via the header fields Content-Type and Content-Encoding.
  - These define a two-layer, ordered encoding model:

```
entity-body := Content-Encoding( Content-Type( data ) )
```

  - Content-Type specifies the media type of the underlying data.
  - Content-Encoding may be used to indicate any additional content codings applied to the data, usually for the purpose of data compression, that are a property of the requested resource."

ETags, conditional gets

- **Problem: how to validate a cached version**
- **Solutions: etag and last-modified**
- **Entity tags (ETag header): unique tag**
  - Two entities may have the same tag only if they are equal byte-by-byte
  - Often MD5 sum is used (e.g. by Amazon S3)
  - Server provides `ETag` with response
  - Client/cache queries with `If-Not-Match:` *`entity-tag`*
  - Server responds with
    - `304 Not Modified` – no response body
    - `200 OK` – new response body
- **Also:**
  - If-Match: *entity-tag* – useful for PUTs

Conditional GETs (cont.)

- **Date-based validation:**
  - Server provides `last-modified` date/time in response
  - Client/cache queries using `If-Modified-since:` *`date`*
    - should use last-modified header value, not arbitrary date/time
  - Server responds with 200 or 304
- **Also:**
  - If-Unmodified-Since

Cache-control header insanity

- **Cache request directives**
  - no-cache – go to origin server
  - no-store – do not store
  - max-age
  - max-stale=*delta-seconds* – beyond age…
  - min-fresh=*delta-seconds* – fresh for at least…
  - no-transform – e.g. for slow links
  - only-if-cached
- **Cache response directives**
  - public – may cache publicly
  - private – may cache privately only
  - no-cache – do not re-use without re-validation
  - no-store – do not store
  - no-transform – e.g. for slow links
  - must-revalidate – independent of max-stale
  - proxy-revalidate
  - max-age=*delta-seconds*
  - s-maxage=*delta-seconds*

Cookies

- **Cookies not part of HTTP/1.1! See RFC2965…**
  - Headers: Set-Cookie2, Cookie
  - Fields:
    - Domain=.expertcity.com
    - Path=/myaccount/
    - Port="80,8080"
    - Discard (when u-a terminates)
    - Max-age=3600
    - Secure
- **Uses?**
  - How to handle users who disable them?
- **Security?**

HTTP tricks

- **Instant notifications**
  - How to get notified when server state changes
  - E.g. instant message / chat message arrives?
- **Bidirectional communication**
  - How can “server” issue “requests” to client?

Summary

- **Why is it so hard?**
  - Each detail is simple
  - The collection is unmanageable
  - Need more layers? More orthogonality?
- **How can you implement it?**
  - What do you expect clients to implement?
  - What do you expect servers to implement?
  - What do you expect proxies to implement?
  - What do you expect all these to get wrong?
  - How do you test your own implementation?
- **How come it works at all?**
  - Nobody uses the full spec?
  - The real spec is IE & NS?