HTTP and the Web

Mike Freedman

https://www.cs.princeton.edu/courses/archive/spr20/cos461/

Two Forms of Header Formats

- **Fixed:** Every field (type, length) defined
  - Fast parsing (good for hardware implementations)
  - Not human readable
  - Fairly static (IPv6 ~20 years to deploy)
  - E.g., Ethernet, IP, TCP headers

- **Variable length headers**
  - Slower parsing (hard to implement in hardware)
  - Human readable
  - Extensible
  - E.g., HTTP (Web), SMTP (Email), XML

HTTP Basics (Overview)

- **HTTP over bidirectional byte stream (e.g. TCP)**

- **Interaction**
  - Client looks up host (DNS)
  - Client sends request to server
  - Server responds with data or error
  - Requests/responses are encoded in text

- **Stateless**
  - HTTP maintains no info about past client requests
  - HTTP “Cookies” allow server to identify client and associate requests into a client session

HTTP Response

<table>
<thead>
<tr>
<th>version</th>
<th>sp</th>
<th>status code</th>
<th>sp</th>
<th>phrase</th>
<th>cr</th>
<th>if</th>
</tr>
</thead>
<tbody>
<tr>
<td>header field name</td>
<td>value</td>
<td>cr</td>
<td>if</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>header field name</td>
<td>value</td>
<td>cr</td>
<td>if</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
"cr" is \r
"lf" is \n
Entity Body
```
HTTP Request

- Request line
  - Method
    - GET – return URI
    - HEAD – return headers only of GET response
    - POST – send data to the server (forms, etc.)
  - URL (relative)
    - E.g., /index.html
  - HTTP version

HTTP Request Example

GET /index.html HTTP/1.1
Host: www.example.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Connection: Keep-Alive

HTTP Request (cont.)

- Request headers
  - Variable length, human-readable
  - Uses:
    - Authorization – authentication info
    - Acceptable document types/encodings
    - From – user email
    - If-Modified-Since
    - Referrer – what caused this page to be requested
    - User-Agent – client software
  - Blank-line
  - Body

HTTP Response

- Status-line
  - HTTP version (now “1.1”)
  - 3 digit response code
    - 1XX – informational
    - 2XX – success
      - 200 OK
    - 3XX – redirection
      - 301 Moved Permanently
      - 303 Moved Temporarily
      - 304 Not Modified
    - 4XX – client error
      - 404 Not Found
    - 5XX – server error
      - 505 HTTP Version Not Supported
  - Reason phrase
HTTP Response (cont.)

• Headers
  – Variable length, human-readable
  – Uses:
    • Location – for redirection
    • Server – server software
    • WWW-Authenticate – request for authentication
    • Allow – list of methods supported (get, head, etc)
    • Content-Encoding – E.g. gzip
    • Content-Length
    • Content-Type
    • Expires (caching)
    • Last-Modified (caching)
• Blank-line
• Body

HTTP Response Example

HTTP/1.1 200 OK
Date: Tue, 27 Mar 2001 03:49:38 GMT
Server: Apache/1.3.14 (Unix) (Red-Hat/Linux) mod_ssl/2.7.1
  OpenSSL/0.9.5a DAV/1.0.2 PHP/4.0.1pl2 mod_perl/1.24
Last-Modified: Mon, 29 Jan 2001 17:54:18 GMT
Accept-Ranges: bytes
Content-Length: 4333
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
…..

How to Mark End of Message?

• Close connection
  – Only server can do this
  – One request per TCP connection. Hurts performance.
• Content-Length
  – Must know size of transfer in advance
• No body content. Double CRLF marks end
  – E.g., 304 never have body content
• Transfer-Encoding: chunked (HTTP/1.1)
  – After headers, each chunk is content length in hex, CRLF, then body. Final chunk is length 0.

Example: Chunked Encoding

HTTP/1.1 200 OK <CRLF>
Transfer-Encoding: chunked <CRLF>
  <CRLF>
  25 <CRLF>
  This is the data in the first chunk <CRLF>
  1A <CRLF>
  and this is the second one <CRLF>
  0 <CRLF>

• Especially useful for dynamically-generated content, as length is not a priori known
  – Server would otherwise need to cache data until done generating, and then go back and fill-in length header before transmitting
Proxies

- End host that acts a broker between client and server
  - Speaks to server on client’s behalf

- Why?
  - Privacy
  - Content filtering
  - Caching!!!

Proxies (Cont.)

- Accept requests from multiple clients
- Takes request and reissues it to server
- Takes response and forwards to client

HTTP Caching

- Why cache?
  - Lot of objects don’t change (images, js, css)
  - Reduce # of client connections
  - Reduce server load
  - Reduce overall network traffic; save $$$
Caching is Hard

- Significant fraction (>50%) of distinct HTTP objects may be uncacheable
  - Dynamic data: Stock prices, scores, webcams
  - CGI scripts: results based on passed parameters
  - Cookies: results may be based on passed data
  - SSL: encrypted data is not cacheable
  - Advertising / analytics: owner wants to measure # hits
    - Random strings in content to ensure unique counting

- Yet significant fraction of HTTP bytes are cacheable
  - Images, video, CSS pages, etc.

- Want to limit staleness of cached objects

How long should the client cache for?

- Clients (and proxies) cache documents
  - When should origin be checked for changes?
    - Every time? Every session? Date?

- HTTP includes caching information in headers
  - HTTP 0.9/1.0 used: "Expires: <date>"; "Pragma: no-cache"
  - HTTP/1.1 has "Cache-Control"
    - "No-Cache", "Max-age: <seconds>"
    - "ETag: <opaque value>

Why the changes between 1.0 and 1.1?

- Timestamps
  - Server hints when an object “Expires” (Expires: xxx)
  - Server provides last modified date, client can check if that’s still valid

- Problems
  - Client and server might not have synchronized clocks
  - Server replicas might not have synchronized clocks
  - Max-age solves this: relative seconds, not abs time

What if cache expires?

- Store past expiry time (if room in cache)
- Upon request, first revalidate with server

GET / HTTP/1.1
Accept-Language: en-us
If-Modified-Since: Mon, 29 Jan 2001 17:54:18 GMT
Host: www.example.com
Connection: Keep-Alive

HTTP/1.1 304 Not Modified
Date: Tue, 27 Mar 2001 03:50:51 GMT
Connection: Keep-Alive
Another problem!

- What if server replicas don’t have aligned modification times?

HTTP/1.1 200
Date: Tue, 27 Mar 2001 03:50:51 GMT
ETag: 686897696a7c876b7e

GET / HTTP/1.1
Accept-Language: en-us
If-None-Match: "686897696a7c876b7e"
Host: www.example.com
Connection: Keep-Alive

HTTP xfer = single object
Web pages = many objects

How to handle many requests?

- Maximize goodput by reusing connections
  - Avoid connection (TCP) setup
  - Avoid TCP slow-start
- Client-server will maintain existing TCP connection for up to K idle seconds

GET / HTTP/1.1
Host: www.example.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 27 Mar 2001 03:50:51 GMT
Connection: Keep-Alive
Three approaches to multiple requests

<table>
<thead>
<tr>
<th>Persistent Connections</th>
<th>Pipelined Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conn 1:</td>
<td>Conn 1:</td>
</tr>
<tr>
<td>• Request 1</td>
<td>• Request 1</td>
</tr>
<tr>
<td>• Response 1</td>
<td>• Response 1</td>
</tr>
<tr>
<td>• Request 2</td>
<td>• Request 2</td>
</tr>
<tr>
<td>• Response 2</td>
<td>• Response 2</td>
</tr>
<tr>
<td>Conn 2:</td>
<td>Conn 1:</td>
</tr>
<tr>
<td>• Request 2</td>
<td>• Request 1</td>
</tr>
<tr>
<td>• Request 3</td>
<td>• Request 2</td>
</tr>
<tr>
<td>• Response 3</td>
<td>• Response 1</td>
</tr>
<tr>
<td>• Response 3</td>
<td>• Response 2</td>
</tr>
</tbody>
</table>

What are challenges with pipelining?

- Head-of-line blocking
  - Small xfers can “block” behind large xfer
- No reordering
  - HTTP response does not “identify” which request it’s in response to; obvious in simple request/response
- Can behave worse than parallel + persistent
  - Can send expensive query 1 on conn 1, while sending many cheap queries on conn 2

Google’s SPDY -> HTTP/2

- Server “push” for content
  - One client request, multiple responses
  - After all, server knows that after parsing HTML, client will immediately request embedded URLs
- Better pipelining and xfer
  - Multiplexing multiple xfers w/o HOL blocking
  - Request prioritization
  - Header compression

https://developers.google.com/web/fundamentals/performance/http2