Hypertext Transport Protocol HTTP/1.1

Jim Gettys

Digital Equipment Corporation, ISBU

Visiting Scientist, World Wide Web Consortium

10/17/96

Acknowledgments

- HTTP/1.1 Authors
  - Roy Fielding (UCI)
  - Jim Gettys - Editor (Digital ISBU / W3C)
  - Jeff Mogul (Digital / WRL)
  - Henrik Frysyk Nielsen (W3C)
  - Tim Berners Lee (W3C)
- IETF HTTP Working Group
  - Larry Masinter - Working Group Chair

Introduction

- HTTP/1.1 is an upward compatible evolution of HTTP
  - Focus is on making HTTP a good Internet citizen, while improving performance for both clients and servers, while minimizing time required to deploy clients and servers

Presentation Organization

- Major Problems of HTTP/1.0
- What's Not in HTTP/1.1, that needs to be done soon
- Fixes to HTTP/1.0
- New Features Introduced in HTTP/1.1
  Cache Features
  Persistent Connections / Chunked encoding
  New Methods
  Range Requests

HTTP Structure

- Very good idea: adoption of MIME type registry
- Less good idea: HTTP Based on Internet Mail Protocols (SMTP, MIME)
- Different enough from and similar enough to MIME to confuse MIME wizards
- Consequences
  - Slow to parse
  - Verbose, therefore high latency
  - Don't know length of a request/response until after parsing the protocol

HTTP/1.0 Design Problems

- HTTP/1.0 Protocol - Informational RFC
  - Feature: Simple; open, operation, close
  - Bug: Fetches single URL per TCP connection
  - Mean size of gets only a few thousand bytes
  - Bimodal size of URL's, usually short
- Out and out mistakes in the protocol
  - Host bug
  - Caching primitive at best
- 'Flash Crowd' Problem due to success

Consequences of HTTP/1.0

- Closing connection causes loss of congestion information
- Connection opens may be congesting low bandwidth links, due to lack of flow control on TCP opens and closes
- Poor user perceived performance (most connections in slow-start)
  - Workaround has been opening multiple simultaneous connections, with resulting congestion problems
- Servers have thousands of connections in close_wait state
  - e.g. AltaVista server is at > 20 million connections/day, or >230/second averaged over 24 hours
  - Cost is primarily memory, on systems running reasonable TCP implementations
- Vanity servers with HTTP result in big servers using 100's of I.P. addresses and consequential routing headaches
- Caching model is primitive, and broken enough that content providers often defeat caching

Multiple Connection Hack

- Problem:
  - Rendering requires meta-data of objects embedded in that page
- Solution:
  - Open N connections simultaneously (by default 4 with Netscape)
- Result:
  - Faster time to render
  - but… Self congestion of nearly simultaneous opens
  - "Unfair" use of bandwidth (problem of the "commons")
- Has made HTTP/1.0's use of TCP, a bad problem, much worse
- Even with persistent connections, we will have fairness problems for clients that use multiple connections, versus "well behaved" clients using one. RED algorithm in routers may help.

Goals of HTTP/1.1

1) Reduce HTTP's impact on the Internet, and make HTTP a 'well behaved' Internet protocol
2) Finish HTTP/1.1 quickly (see 1.)
3) Be as compatible as possible with HTTP/1.0, particularly for origin servers and clients (see 2.)

Now an IETF Proposed Standard

Deferred to Future

- Hit count reporting to avoid cache busting
  - Important to increase caching
  - Draft out for review (Leach & Mogul)
- Compressed Protocol
  - Sticky headers proposal from Paul Leach
- Multiplexing of HTTP stream
  - Reduces further need for multiple connections
  - Transition strategy to future protocols
- Transparent Content Negotiation
- User agent negotiation
- PEP - HTTP protocol extensibility/negotiation

Fixes to HTTP/1.0

- *Host:* header
  - Solve major headache of Web service providers
- Reliable caching
  - Problems with getting stale data in a cache
  - Insufficient cache control to build reliable applications - result: applications defeat caching, and get poor performance
  - Insufficient control to users, or warnings of problems

New Features - *Host:*

- *Host* Header
  - All requests now accompanied by *Host:* header
  - Not dependent on HTTP/1.1 for deployment
  - Already implemented by majority of clients
  - HTTP/1.1 requires *Host:* header to be present, or an error will be returned
    (to detect buggy implementations)
  - Vital that this be implemented as soon as possible, even if your implementation is not otherwise HTTP/1.1 compliant

Range Requests

- Already commonly implemented (Netscape, Microsoft)
- Not dependent on HTTP version; optional feature of protocol
- *Range:* allows client to request a range
- *Content-Range:* specifies where a partial entity should be inserted
- *If-Range:* 'if the entity is unchanged, send me the part(s) that I am missing; otherwise, send me the entire new entity.'

Caching - Semantic Transparency

- Caching that application developers and users can rely on
- Requirements: disconnected operation and island caches
- A cache behaves in a "semantically transparent" manner, with respect to a particular response, when its use affects neither the requesting client nor the origin server, except to improve performance. When a cache is semantically transparent, the client receives exactly the same response that it would have received had its request been handled directly by the origin server.

Caching - *Age:*

- HTTP/1.1 introduces *Age:* header, to allow conservative age of document computation
- the *Age:* value is the sum of the time that the response has been resident in each of the caches along the path from the origin server, plus the amount of time it has been in transit along network paths.
- Enables reliable cache expiration
- See section 13.2.3

Caching - Last-Modified Dates

- HTTP/1.0 has only *Last-Modified:* Dates
  - Unreliable under some circumstances: resolution left to 1 second
  - Notorious problems serving NFS files
  - Hard for some servers to generate (e.g. databases)
- Used with
  - *If-Modified-Since:*
  - *If-Unmodified-Since:*

Caching - Opaque Validators

- HTTP/1.1 Introduces Opaque Validators (Opaque to Client; left to server to choose)
- Strong Validators
  - Indicates equality of entry
- Weak Validators (optional)
- Transmitted using the *Etag:* header
- Typically used with:
  - *If-Match:*
  - *If-None-Match:*

Caching - Strong Validators

- Server chooses validator
- Indicates exact equality of cached entity
- Can be used with range requests

Caching - Weak Validators

- Weak Validators
  - Indicates equivalence, rather than bit equality of an entity
  - Useful for hit-counters, and other applications where equivalence will do
  - e.g. rapidly changing entity in which any version is equivalent
- Cannot be used with Range Requests
- Optional part of HTTP/1.1

Caching - *Cache-Control:*

- Categories of *Cache-Control:* directives
  - Restrictions on what is cachable; these may only be imposed by the origin server.
  - Restrictions on what may be stored by a cache; these may be imposed by either the origin server or UA
  - Modifications of the basic expiration mechanism; these may be imposed by either the origin server or the UA.
  - Controls over cache revalidation and reload; these may only be imposed by a user agent.
  - Control over transformation of entities.
  - Extensions to the caching system.

Caching - *Cache-Control:*

- What is cachable
  - these may only be imposed by the origin server.
  - *public*
    (mark as cachable responses that would otherwise be non-cachable or private)
  - *private*
    (must not be cached in a public cache)
  - *no-cache*
    (do not cache anywhere, even in caches deliberately returning stale responses)

Caching - *Cache-Control:*

- What may be stored in a cache
  - these may only be imposed by the origin server.
  - *no-store*
    prevent the inadvertent release or retention of sensitive information (for example, on backup tapes).

Caching - *Cache-Control:*

- Modifications of the expiration mechanism
  - imposed by either the origin server or the UA.
  - *max-age = delta-seconds*
    Indicates that the client is willing to accept a response whose age is no greater than the specified time in seconds.
  - *min-fresh = delta-seconds*
    The client wants a response that will still be fresh for at least the specified number of seconds.
  - *max-stale [= delta-seconds]*
    Indicates that the client is willing to accept a response that has exceeded its expiration time.

Caching - *Cache-Control:*

- Controls over cache revalidation and reload
  - these may only be imposed by a user agent.
  - *no-cache* Force end-to-end reload
  - *max-age = delta-seconds* The client is willing to accept a response whose age is no greater than the specified time in seconds.
  - *must-revalidate* Forces revalidation, or generates Gateway Timeout response
  - *proxy-revalidate* Same as *must-revalidate*, except for private user-agent caches

Caching - *Cache-Control:*

- Transformations in caches are desirable for many situations (e.g. gif -> PNG or JPEG, or conversion of resolution for PDA use), *however* some applications fail if their data has been transformed
- Applications that care can prohibit transformations
  - *no-transform*
    Do not transform this entity to a different content-type.

Caching - Warnings

- Provided any time ‘semantic transparency’ is violated
- Intended for both disconnected and ‘island’ cache use
- HTTP/1.1 warnings
  - *Response is stale*
  - *Revalidation failed*
  - *Disconnected operation*
  - *Heuristic expiration*
  - *Transformation applied*

New Methods

- **Trace**
  Used in concert with *Via:* and *Max-Forwards:* for debugging
- **Put**
- **Delete**
- **Options**
  Extensibility hook
- **Upgrade**

Persistent Connections

- Default behavior for HTTP/1.1
- Server can indicate connection will be closed by:
  - *Connection: close*
- Request/responses can be pipelined
- Major performance gain for users, and major goodness to the network
- Different than *Keep-Alive:* which did not work correctly with chained proxies

Chunked Encoding

- Some entities do not have known length (e.g. those generated by scripts)
- Chunked Encoding allows transmission of entities where the length is not known in advance

Other New Headers

- *Upgrade:*
  Supports protocol switching on open connection
- *Proxy-Authenticate:*, *Proxy-Authorization:*
  Authentication of proxy caches
- *Content-MD5:*
  MD5 Digest Message Integrity Check
- *Content-Transfer-Encoding:*
  Hop-by-hop compression of entities

Content Negotiation

- *Accept-**
  - Existing practice codified
  - Transparent negotiation deferred from HTTP/1.1, but caching design takes it into account
- *Vary:* indicates which dimensions a response varies on
- *Alternates:* Hook for future content negotiation; to state what alternate representations are available

Implementation Requirements

- *Host:* mandatory, and HTTP/1.1 servers must check for its presence
- Persistent Connections
- Chunked Encoding
- Honoring DNS Time To Live is now an explicit requirement of HTTP/1.1
- Proxy caches have significant work to implement HTTP/1.1

Associated Documents

- Digest Authentication
  - Now an IETF Proposed Standard
  - Replacement for Basic Authentication
  - Should eliminate cleartext passwords
  - Uses cryptographic hashes for authentication
  - No export control problems
  - Appears everyone will implement
  - Big benefit for Corporate Intranets with distributed proxy caches

Associated Documents (Cont.)

- State Management
  - In IETF last call, Proposed Standard status expected soon
  - Compatible standardization of Netscape "cookies"
  - Resulted in worst pun of HTTP development…

What This Means

- HTTP/1.1
  - Improves user perceived performance
  - Reduces load and increase performance of HTTP servers
  - Enables reliable applications in the face of caching
  - Should help congestion and other operational problems of parts of the Internet
  - Decreases load of HTTP on the Internet

More Information

- More information at http:/www.w3.org/pub/WWW/Protocols
- Specifications, pointers to more information, and this talk can be found at this location

Next Steps

- Deferred items
- Better support for authoring tools
- Multiplexing transport to reduce need for multiple connections
  (but game theory implies additional network level solutions needed, e.g. RED algorithm)
- Reduce latency over low bandwidth/high latency links via protocol compression and pipelining implementations