HTTP – The Gory Details

CS 360 Internet Programming

Daniel Zappala

Brigham Young University
Computer Science Department

HTTP 0.9

- simple request

```
1 GET URI CRLF
```

- simple response

```
1 [entity body]
```

- no version
- no headers
- no way of determining type of body (HTTP, GIF, etc)

HTTP 1.0 Specification

- RFC 1945: http://www.ietf.org/rfc/rfc1945.txt
- servers must be backward compatible with HTTP/0.9

GET, HEAD

- GET
  - see RFC 1945, section 8.1
  - MAY encode arguments to a script
  - `GET /book.cgi?lastname=Stevens`
  - MAY be modified by `If-Modified-Since` header
- HEAD
  - see RFC 1945, section 8.2
  - MUST NOT return entity body
  - SHOULD ignore `If-Modified-Since` header

POST

- see RFC 1945, section 8.3
- MUST have a `Content-Length` to determine length of entity body
- actions taken depend on the URI

PUT, DELETE, LINK, UNLINK

- see RFC 1945, Appendix D
- PUT and DELETE considered dangerous because the client is in charge
- not usually supported by web servers

Header Syntax

```
1  HTTP-header = field-name ":" [ field-value ] CRLF
```

- arbitrary length
- order is not significant, but recommended that you send General, Request, Response, and Entity Headers in that order
- *many headers can be parsed by simply reading until CRLF, particularly if the header is going to be ignored; your server does not necessarily need to understand the syntax of all the headers*
- see RFC 1945, section 4.2

General Headers

- `Date`
  - RFC 1945, Section 10.6
  - date and time at which message originated
  - three different formats given in Section 3.3, but use the first format (RFC 1123)
  - MUST be in GMT
- `Pragma`
  - RFC 1945, Section 10.12
  - `no-cache` is only specified directive, tells cache to always forward to origin server
  - server should parse and ignore

Request Headers

- `If-Modified-Since`
  - RFC 1945, Section 10.9
  - defines a conditional GET - retrieve resource only if it hasn't been changed since the given date
  - uses the same format as the `Date` header
- `From`
  - RFC 1945, Section 10.8
  - used to give user's email address
  - server should parse and log
- `User-Agent`
  - RFC 1945, Section 10.15
  - information about user's browser
  - server should parse and log

Response Headers

- `Server`
  - RFC 1945, Section 10.14
  - identifies server software
  - many servers include this in all responses

Entity Headers

- `Content-Length`
  - RFC 1945, Section 10.4
  - size of the entity body in bytes
  - MUST be included for any message containing an entity body
  - server should send an error if it is missing, see Section 7.2.2
- `Last-Modified`
  - RFC 1945, Section 10.10
  - date and time at which resource was last modified
  - MUST NOT send a time later than message origination
- `Content-Type`
  - RFC 1945, Section 10.5
  - indicates media type of entity body
  - server SHOULD include this header if the message has an entity body

Response Classes

- HTTP groups response codes together into classes
- 100: informational class
- 200: success class
- 300: redirection class
- 400: client error class
- 500: server error class

Notable Error Codes

- 200 OK
- 304 Not Modified
- 400 Bad Request
- 403 Forbidden
- 404 Not Found
- 501 Not Implemented

Some History

- HTTP/1.1 standardization took 4 years
- after two years RFC 2068 captured the early state of the HTTP/1.1 specification *process*
- browsers started implementing the "HTTP/1.1 standard" before it was officially a standard
- HTTP/1.1 needed to be backward-compatible with many browsers or else many sites would not deploy it
- as a result, RFC 2616 contains some idiosyncrasies

Multiple Web Sites

- HTTP/1.0 allows only one web site per server

```
1  GET /index.html HTTP/1.0
```

- HTTP/1.1 allows many web sites per server
- server MUST include `Host` header in HTTP/1.1

```
1  GET /index.html HTTP/1.1
2  Host: ilab.cs.byu.edu
```

Content Negotiation

- significantly more mature than in HTTP/1.0
- server driven
  - client sends hints about user's preference using `Accept-Language`, `Accept-Charset`, `Accept-Encoding`
  - server chooses the best match
- agent-driven
  - server responds with `300 Multiple Choices` that includes list of available representations
  - client makes a new request with the chosen variant
- server driven is widely used

HTTP Inefficiency

- TCP works best for long-lived connections
  - starts with a slow rate
  - increases rate as segments are delivered successfully
  - decreases rate when loss occurs
- HTTP/1.0 uses a separate connection for each request
  - most HTTP requests are short (4 KB in a 1997 study)
  - TCP connections never ramp up to a high rate
- need persistent connections and pipelining

Persistent Connections and Pipelining

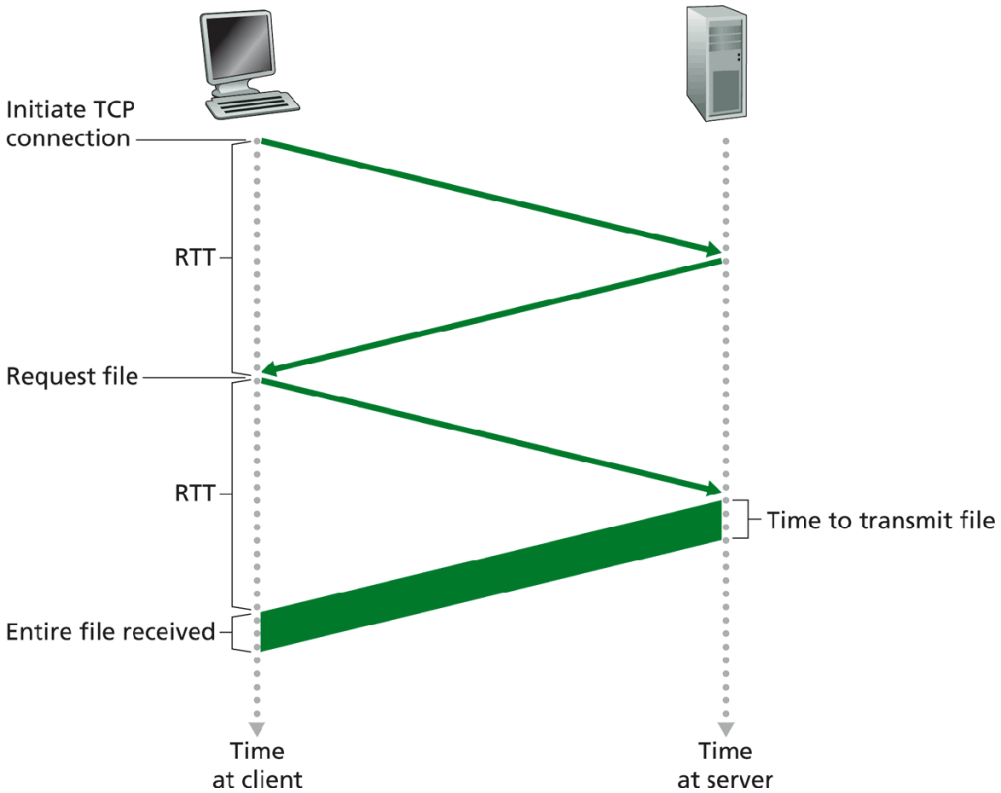
- some HTTP/1.0 browsers started implementing persistent connections using `Connection: Keep-Alive` request header – need cooperation from server
- browsers started using parallel connections to speed up delivery time for client - each connection has to pay TCP startup penalties, consumes server resources
- HTTP/1.1 uses a Connection header
  - by default, connections stay open
  - server sends `Connection: close` to notify client it will close the connection
  - connections may be closed at any time

Persistent Connections and Pipelining

- some HTTP/1.0 browsers started implementing persistent connections using `Connection: Keep-Alive` request header – need cooperation from server
- browsers started using parallel connections to speed up delivery time for client - each connection has to pay TCP startup penalties, consumes server resources
- HTTP/1.1 uses a Connection header
  - by default, connections stay open
  - server sends `Connection: close` to notify client it will close the connection
  - connections may be closed at any time
- pipelining: multiple requests can be sent without waiting for a response, greatly improves TCP performance

Modeling Non-persistent Connection Latency

- RTT: round-trip time: time from client to server and back
- response time
  - one RTT to initiate connection
  - one RTT for HTTP request and response
  - file transmission time
- $2 * RTT + Transmit$



Modeling Parallel Connection Latency

- base page plus n embedded images or files
- use parallel connections to request the images
- can overlap RTTs but not the transmission delay
- latency depends on the bandwidth between you and the server(s) you use
- best case if the images are on different servers and your computer is not the bottleneck
  - $2 * RTT + BaseTransmit + 2 * RTT + EmbeddedTransmit$
- worst case
  - $2 * RTT + BaseTransmit + 2 * RTT + n * EmbeddedTransmit$

Pipelining

- for persistent connections (HTTP 1.1) only
- once client has fetched base web page, can send multiple requests back-to-back
- if all of the objects are on the same server
  - results in one RTT for all the referenced objects
  - $2 * RTT + BaseTransmit + RTT + n * EmbeddedTransmit$
- saves on connection state for the server

Downloading a Part of a File

- user may want only part of a resource
  - continuing after an aborted transfer
  - fetching in parallel from multiple servers
- `Range` header
  - specify one or more ranges of contiguous bytes
  - if server's response contains a range, it uses `206 Partial Content`
- `If-Range` header: conditional get using ETag

Compression

- 1997 study showed that compression could save 40% of the bytes sent via HTTP
- `Content-Encoding`: used in HTTP/1.0, indicates end-to-end content coding
- `Transfer-Encoding`: used in HTTP/1.1, indicates hop-by-hop transfer-codings (applied by proxies)
- `Accept-Encoding`: used in HTTP/1.0, indicates the type of encodings a client can accept
- `TE`: used in HTTP/1.1, indicates the type of transfer-codings a client prefers
- IANA registers transfer-coding token values, including chunked, identity, gzip, compress, and deflate

Chunked Transfers

- chunked transfers are used by a web server for dynamic content, when it doesn't know ahead of time the length of the entire body of the object
- sends the body as a series of chunks, each having the following format:
  - length of the chunk, in hexadecimal, followed by a CRLF (`\r\n`)
  - a sequence of bytes, whose length is given by the previous line
  - a CRLF
- to end the body, the server sends a valid last chunk with a length of 0 bytes

References

- *Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement, 1st Edition*, by Balachander Krishnamurthy and Jennifer Rexford, Addison Wesley Professional, 2001, ISBN 0201710889.