

HTTP

Hypertext Transfer Protocol

Refs:

RFC 1945 (HTTP 1.0)

RFC 2616 (HTTP 1.1)

HTTP Usage

- HTTP is the protocol that supports communication between web browsers and web servers.
- A “Web Server” is a HTTP server
- Most clients/servers today speak version 1.1, but 1.0 is also in use.

From the RFC

“HTTP is an application-level protocol with the lightness and speed necessary for distributed, hypermedia information systems.”

Transport Independence

- The RFC states that the HTTP protocol generally takes place over a TCP connection, but the protocol itself is not dependent on a specific transport layer.

Request - Response

- HTTP has a simple structure:
 - client sends a request
 - server returns a reply.
- HTTP can support multiple request-reply exchanges over a single TCP connection.

Well Known Address

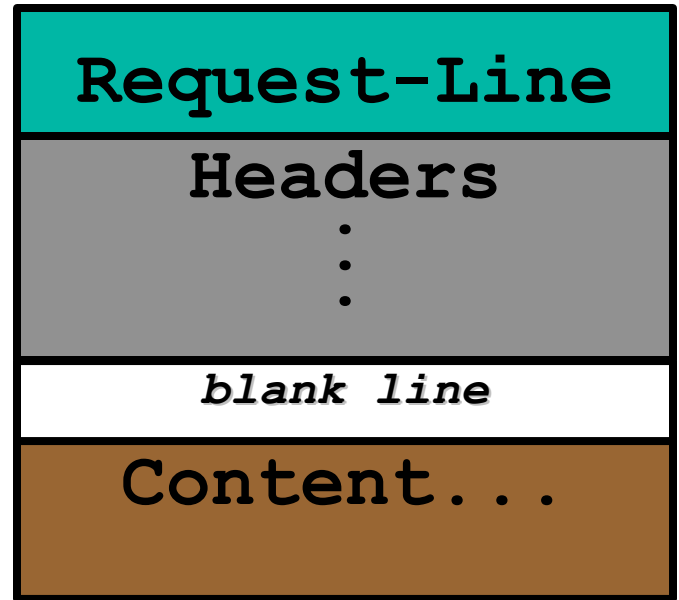
- The “well known” TCP port for HTTP servers is port 80.
- Other ports can be used as well...

HTTP Versions

- The original version now goes by the name “HTTP Version 0.9”
 - HTTP 0.9 was used for many years.
- Starting with HTTP 1.0 the version number is part of every request.
 - tells the server what version the client can talk (what options are supported, etc).

HTTP 1.0+ Request

- Lines of text (ASCII).
- Lines end with CRLF “\r\n”
- First line is called “Request-Line”



Request Line

Method URI HTTP-Version\r\n

- The request line contains 3 *tokens* (words).
- space characters “ ” separate the tokens.
- Newline (\n) seems to work by itself (but the protocol requires CRLF)

Request Method

- The Request Method can be:

GET

HEAD

PUT

POST

DELETE

TRACE

OPTIONS

future expansion is supported

Methods

- GET: retrieve information identified by the URI.
- HEAD: retrieve meta-information about the URI.
- POST: send information to a URI and retrieve result.

Methods (cont.)

- PUT: Store information in location named by URI.
- DELETE: remove *entity* identified by URI.


More Methods

- **TRACE:** used to trace HTTP forwarding through proxies, tunnels, etc.
- **OPTIONS:** used to determine the capabilities of the server, or characteristics of a named resource.

Common Usage

- GET, HEAD and POST are supported everywhere.
- HTTP 1.1 servers often support PUT, DELETE, OPTIONS & TRACE.

URI: Universal Resource Identifier

- URIs defined in RFC 2396.
- Absolute URI:
`scheme://hostname[:port]/path`
`http://www.cs.rpi.edu:80/blah/foo`
- Relative URI: /path
 /blah/foo
No server mentioned 

URI Usage

- When dealing with a HTTP 1.1 server, only a *path* is used (no scheme or hostname).
 - HTTP 1.1 servers are required to be capable of handling an absolute URI, but there are still some out there that won't...

HTTP Version Number

`"HTTP/1.0"` or `"HTTP/1.1"`

HTTP 0.9 did not include a version number in a request line.

If a server gets a request line with no HTTP version number, it assumes 0.9

The Header Lines

- After the *Request-Line* come a number (possibly zero) of HTTP *header lines*.
- Each header line contains an attribute name followed by a “:” followed by a space and the attribute value.

Headers

- Request Headers provide information to the server about the client
 - what kind of client
 - what kind of content will be accepted
 - who is making the request
- There can be 0 headers (HTTP 1.0)
- HTTP 1.1 requires a **Host**: header

Example HTTP Headers

Accept: text/html

Host: www.rpi.edu

From: neytmann@cybersurg.com

User-Agent: Mozilla/4.0

Referer: http://foo.com/blah

End of the Headers

- Each header ends with a CRLF (`\r\n`)
- The end of the header section is marked with a blank line.
 - `just CRLF`
- For GET and HEAD requests, the end of the headers is the end of the request!

POST

- A POST request includes some *content* (*some data*) after the headers (after the blank line).
- There is no format for the data (just raw bytes).
- A POST request must include a Content-Length line in the headers:

Content-length: 267

Example GET Request

```
GET /~hollingd/testanswers.html HTTP/1.1
```

```
Accept: */*
```

```
Host: www.cs.rpi.edu
```

```
User-Agent: Internet Explorer
```

```
From: cheater@cheaters.org
```

```
Referer: http://foo.com/
```



There is a blank line here!

Example POST Request

POST /~hollingd/changegrade.cgi HTTP/1.1

Accept: */*

Host: www.cs.rpi.edu

User-Agent: SecretAgent V2.3

Content-Length: 35

Referer: http://monte.cs.rpi.edu/blah

stuid=6660182722&item=test1&grade=99

Typical Method Usage

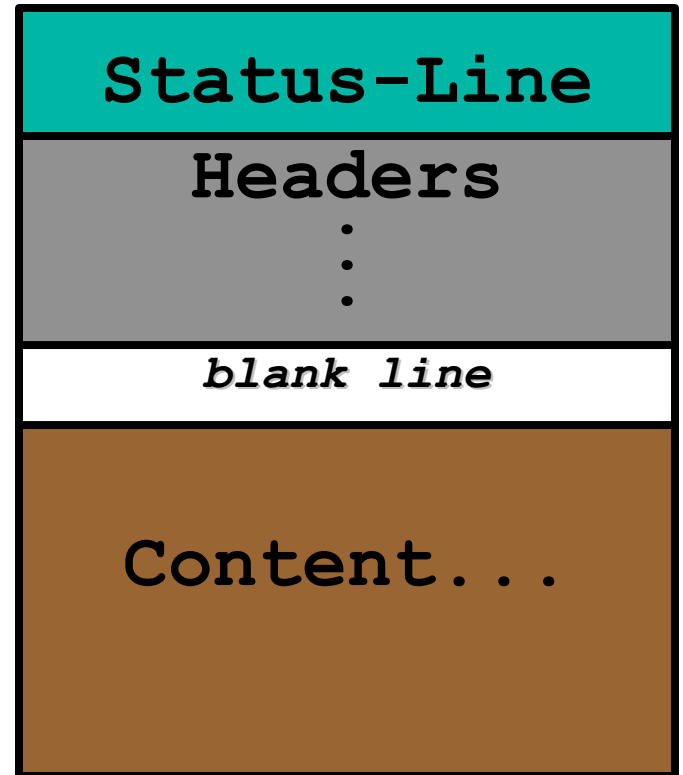
GET used to retrieve an HTML document.

HEAD used to find out if a document has changed.

POST used to submit a form.

HTTP Response

- ASCII Status Line
- Headers Section
- Content can be anything (not just text)
 - typically an HTML document or some kind of image.



Response Status Line

HTTP-Version Status-Code Message

- *Status Code* is 3 digit number (for computers)
- Message is text (for humans)

Status Codes

1xx Informational

2xx Success

3xx Redirection

4xx Client Error

5xx Server Error

Example Status Lines

HTTP/1.0 200 OK

HTTP/1.0 301 Moved Permanently

HTTP/1.0 400 Bad Request

HTTP/1.0 500 Internal Server Error

Response Headers

- Provide the client with information about the server and the returned *entity* (document).
 - what kind of document
 - how big the document is
 - how the document is encoded
 - when the document was last modified
- Response headers end with blank line

Response Header Examples

Date: Wed, 30 Jan 2002 12:48:17 EST

Server: Apache/1.17

Content-Type: text/html

Content-Length: 1756

Content-Encoding: gzip

Content

- Content can be anything (sequence of raw bytes).
- Content-Length header is required for any response that includes content.
- Content-Type header also required.

Single Request/Reply

- The client sends a complete request.
- The server sends back the entire reply.
- The server closes it's socket.

- If the client needs another document it must open a new connection.

Persistent Connections

- HTTP 1.1 supports persistent connections (this is the default).
- Multiple requests can be handled over a single TCP connection.
- The **Connection:** header is used to exchange information about persistence (HTTP/1.1)
- 1.0 Clients used a **Keep-alive:** header

Try it with `telnet`

```
> telnet www.cs.rpi.edu 80
```

```
GET / HTTP/1.0
```

Blank Line
(end of headers)



```
HTTP/1.0 200 OK
```

```
Server: Apache
```

Response



```
...
```

Try it with telnet 1.1

```
> telnet www.cs.rpi.edu 80
```

```
GET / HTTP/1.1
```

```
Host: www.cs.rpi.edu
```

```
HTTP/1.0 200 OK
```

```
Server: Apache
```

```
...
```