CpSc 360: Distributed and Network Programming

HTTP Protocol

James Wang

http://www.faqs.org/rfcs/rfc1945.html
http://www.faqs.org/rfcs/rfc2616.html

What is HTTP?

- **The Hypertext Transfer Protocol (HTTP) is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems.**
- **It is a generic, stateless, object-oriented protocol which can be used for many tasks, such as name servers and distributed object management systems, through extension of its request methods (commands).**
- **HTTP is the protocol that supports communication between web browsers and web servers.**
- **HTTP is also used as a generic protocol for communication between user agents and proxies/gateways to other Internet systems, including those supported by the SMTP, NNTP, FTP, Gopher, and WAIS protocols.**

http://www.faqs.org/rfcs/rfc1945.html
http://www.faqs.org/rfcs/rfc2616.html

2

History

- **HTTP has been in use by the World-Wide Web global information initiative since 1990.**
- **The first version of HTTP, referred to as HTTP/0.9, was a simple protocol for raw data transfer across the Internet.**
- **HTTP/1.0, as defined by RFC 1945, improved the protocol by allowing messages to be in the format of MIME-like messages, containing metainformation about the data transferred and modifiers on the request/response semantics.**
- **HTTP/1.1 takes into consideration the effects of hierarchical proxies, caching, the need for persistent connections, or virtual hosts. It includes more stringent requirements than HTTP/1.0 in order to ensure reliable implementation of its features.**

3

Web Server Basics

- **What is a web server?**
  - Program that responds to requests for documents
    - "http daemon"
  - Uses the Hypertext Transfer Protocol (HTTP) to communicate
  - Physical machine which runs the program
- **Duties**
  - Listen to a port
  - When a client is connected, read the HTTP request
  - Perform some lookup function
  - Send HTTP response and the requested data

4

HTTP Protocol Definitions

- **Connection**
  - Virtual circuit established between two programs for communication
- **Message**
  - Unit of communication adhering to a set of syntax rules and transmitted via a connection
- **Request**
  - A message sent from a client requesting information
- **Response**
  - Message sent from the server acknowledging a request
- **User agent**
  - The client which initiates a request
  - Most often a browser
- **Server**
  - The application program that accepts connections in order to serve requests
- **MIME**
  - Multipurpose Internet Mail Extensions
  - Standards for encoding different media types in a message
  - Originally developed for emailing files and messages in different languages

5

HTTP Session

- **An HTTP session consists of a client request followed by a server response**
- **Requests and responses:**
  - are sent in plain text
  - conform to the HTTP syntax
  - consist of start line, headers, blank line, and message body

6

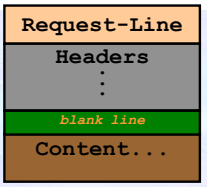
Request - Response

- The HTTP protocol is a request/response protocol. A client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a connection with a server.
- The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity metainformation, and possible entity-body content.
- HTTP can support multiple request-reply exchanges over a single TCP connection.
- The "well known" TCP port for HTTP servers is port 80.
- Other ports can be used as well...

7

HTTP Request

- Lines of text (ASCII).
- Lines end with CRLF `"\r\n"`.
- First line is called "Request-Line".
- Followed by Request Headers.
- Followed by Content if it is a POST.



8

Request Line

Method URI HTTP-Version\r\n

- The request line contains 3 *tokens* (words).
- space characters " " separate the tokens.
- The Request Method can be:

OPTIONS GET HEAD PUT POST DELETE TRACE CONNECT

**** *future expansion is supported* ****

9

Methods

- GET: retrieve information identified by the URI.
- HEAD: retrieve meta-information about the URI.
- POST: send information to a URI and retrieve result.
- PUT: Store information in location named by URI.
- DELETE: remove *entity* identified by URI.
- 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.
- CONNECT: reserved for use with a proxy that can dynamically switch to being a tunnel (e.g. SSL tunneling).

10

URI: Universal Resource Identifier

- URIs defined in RFC 2396.
  http://www.faqs.org/rfcs/rfc2396.html
- Absolute URI:
  `scheme://hostname[:port]/path`
  `http://www.cs.clemson.edu:80/blah/foo`
- Relative URI: `/path`
  `/blah/foo`
- The HTTP protocol does not place any a priori limit on the length of a URI.
- URI may not start with "http" since there are other schemes such as "ftp", "telnet", etc.
  ftp://ftp.is.co.za/rfc/rfc1808.txt -- ftp scheme for File Transfer Protocol services

11

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…
- When dealing with a *proxy* HTTP server, an absolute URI is used.
  - client has to tell the proxy where to get the document!
- http URL
  - The "http" scheme is used to locate network resources via the HTTP protocol.

  http_URL = "http:" "//" host [":" port] [abs_path ["?" query]]

  If the port is empty or not given, port 80 is assumed.

12

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

13

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.
  - Name and value are just text.
- 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

14

Headers (Conti.)

- Example:

  Accept: text/html
  Host: www.clemson.edu
  From: who@where.com
  User-Agent: Mozilla/4.0
  Referer: http://foo.com/blah

- 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!

15

Example GET Request

```
GET /~jzwang/teach.htm HTTP/1.1
Accept: */*
Host: www.cs.clemson.edu
User-Agent: Internet Explorer
From: hacker@hacker.org
Referer: http://foo.com/
```

← There is a blank line here!

16

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: 288`

- Example:

```
POST /~jzwang/updategrade.cgi HTTP/1.1
Accept: */*
Host: www.cs.clemson.edu
User-Agent: Grader V2.3
Content-Length: 35
Referer: http://mycle.clemson.edu/me
```

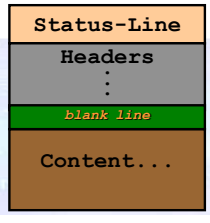
← There is a blank line here!

```
stuid=189182722&item=midterm&grade=99
```

17

HTTP Response

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



18

Response Status Line

HTTP-Version Status-Code Message

- ***Status Code* is 3 digit number (for computers)**
- **Message is text (for humans)**
- **Repose coded by first digit**
  - 1xx: informational, request received
  - 2xx: success, request accepted
  - 3xx: redirection
    - For multiple resources in the same page
  - 4xx: client error
  - 5xx: server error

19

Example Status Lines

```
HTTP/1.1 200 OK
HTTP/1.1 301 Moved Permanently
HTTP/1.1 400 Bad Request
HTTP/1.1 403 Forbidden
HTTP/1.1 404 Not Found
HTTP/1.1 500 Internal Server Error
HTTP/1.1 505 HTTP Version Not Supported
```

20

Response Headers

- **Provide the client with information about 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**
- **Example:**

```
Date: Wed, 30 Jan 2002 12:48:17 EST
Server: Apache/1.17
Content-Type: text/html
Content-Length: 1756
Content-Encoding: gzip
```

← **There is a blank line here!**

21

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.**
- **Header fields can affect content interpretation**
  - required header field: Content-type
  - others: Content-Encoding, Content-Length, Expires, Last-Modified
  - added by web server - we will configure some of these later

22

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.**

This was the default for HTTP 1.0

23

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**

24

Advantages of Persistent Connections

- By opening and closing fewer TCP connections, CPU time is saved in routers and hosts (clients, servers, proxies, gateways, tunnels, or caches), and memory used for TCP protocol control blocks can be saved in hosts.
- HTTP requests and responses can be pipelined on a connection. Pipelining allows a client to make multiple requests without waiting for each response, allowing a single TCP connection to be used much more efficiently, with much lower elapsed time.
- Network congestion is reduced by reducing the number of packets caused by TCP opens, and by allowing TCP sufficient time to determine the congestion state of the network.
- Latency on subsequent requests is reduced since there is no time spent in TCP's connection opening handshake.
- HTTP can evolve more gracefully, since errors can be reported without the penalty of closing the TCP connection. Clients using future versions of HTTP might optimistically try a new feature, but if communicating with an older server, retry with old semantics after an error is reported.