Hypertext Transport Protocol
CSE 333 Winter 2020

Instructor: Justin Hsia

Teaching Assistants:
Andrew Hu        Austin Chan        Brennan Stein
Cheng Ni         Cosmo Wang         Diya Joy
Guramrit Singh   Mengqi Chen        Pat Kosakanchit
Rehaan Bhimani   Renshu Gu          Travis McGaha
Zachary Keyes
Administrivia

- Exercise 16 due Wednesday (3/4)
  - Server-side programming

- hw4 due next Thursday (3/12)
  - Can still use 2 late days
  - Part of section this week will cover tools for debugging hw4
  - Demo today
hw4 demo

- Multithreaded Web Server (333gle)
  - Don’t worry – multithreading has mostly been written for you
  - `./http333d <port> <static files> <indices+>`
  - Some security bugs to fix, too
HTTP Basics

Application Layer (built on top of Transport Layer)

“I’d like index.html”

“Found it, here it is: (index.html)”

- A client establishes one or more TCP connections to a server
  - The client sends a request for a web object over a connection and the server replies with the object’s contents

- We have to figure out how to let the client and server communicate their intentions to each other clearly
  - We have to define a protocol
Protocols

- A **protocol** is a set of rules governing the format and exchange of messages in a computing system
  - What messages can a client exchange with a server?
    - What is the *syntax* of a message? → "grammar"
    - What do the messages *mean*? → "semantics"
    - What are legal replies to a message?
  - What sequence of messages are legal?
    - How are errors conveyed?

- A protocol is (roughly) the network equivalent of an API
HTTP

- **Hypertext Transport Protocol**
  - A request / response protocol
    - A client (web browser) sends a request to a web server
    - The server processes the request and sends a response
  - Typically, a *request* asks a server to retrieve a *resource*
    - A *resource* is an object or document, named by a Uniform Resource Identifier (**URI**)
  - A *response* indicates whether or not the server succeeded
    - If so, it provides the content of the requested response

HTTP Requests

- **General form:**
  
  ```
  [METHOD] [request-uri] HTTP/[version]\r\n
  [headerfield1]: [fieldvalue1]\r\n
  [headerfield2]: [fieldvalue2]\r\n
  [...]

  [headerfieldN]: [fieldvalueN]\r\n
  \r\n
  [request body, if any]
  ```

  - **METHOD** specifies the type of "action" to take.
  - **request-uri** specifies the resource to take action on.
  - **version** specifies the HTTP version (e.g., 1.1).
  - Headers can have any number of headers for flexibility.
  - Blank line to end headers (two '\r\n' in a row).
HTTP Methods

- There are three commonly-used HTTP methods:
  - **GET**: “Please send me the named resource” (used in hw4)
HTTP Methods

- There are three commonly-used HTTP methods:
  - **GET**: “Please send me the named resource”
  - **POST**: “I’d like to submit data to you” (e.g. file upload)
HTTP Methods

- There are three commonly-used HTTP methods:
  - **GET**: “Please send me the named resource”
  - **POST**: “I’d like to submit data to you” (e.g. file upload)
  - **HEAD**: “Send me the headers for the named resource”
    - Doesn’t send resource; often to check if cached copy is still valid
HTTP Methods

- There are three commonly-used HTTP methods:
  - **GET**: “Please send me the named resource”
  - **POST**: “I’d like to submit data to you” (e.g. file upload)
  - **HEAD**: “Send me the headers for the named resource”
    - Doesn’t send resource; often to check if cached copy is still valid

- Other methods exist, but are much less common:
  - **PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH, ...**
    - For instance: **TRACE** – “show any proxies or caches in between me and the server”
HTTP Versions

- All current browsers and servers “speak” **HTTP/1.1**
  - Version 1.1 of the HTTP protocol
    - [https://www.w3.org/Protocols/rfc2616/rfc2616.html](https://www.w3.org/Protocols/rfc2616/rfc2616.html)
  - Standardized in 1997 and meant to fix shortcomings of HTTP/1.0
    - Better performance, richer caching features, better support for multihomed servers, and much more

- **HTTP/2** standardized recently (published in 2015)
  - Allows for higher performance but doesn’t change the basic web request/response model
  - Will coexist with HTTP/1.1 for a long time
    - Hard to change a networking protocol “in the wild”
Client Headers

- The client can provide one or more request “headers”
  - These provide information to the server or modify how the server should process the request

- You’ll encounter many in practice
  - [https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html](https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html)
  - **Host**: the DNS name of the server
  - **User-Agent**: an identifying string naming the browser
  - **Accept**: the content types the client prefers or can accept
  - **Cookie**: an HTTP cookie previously set by the server
A Real Request

```plaintext
GET / HTTP/1.1
Host: attu.cs.washington.edu:3333
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
DNT: 1
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: SESS0c8e598bbe17200b27e1d0a18f9a42bb=5c18d7ed6d369d56b69a1c0aa441d78f; SESSd47cbe79be51e625cab059451de75072=d137dbe7bbe1e90149797dcd89c639b1; _sdsat_DMC_or_CC0DE=null; _sdsat_utm_source=; _sdsat_utm_medium=; _sdsat_utm_term=; _sdsat_utm_content=; adblock=blocked; s_fid=50771A3AC73B3FFF-3F18AABD559FF5B5D; s_cc=true; prev_page=science.%3A%2Fcontent%2F347%2F6219%2F262%2Ftab-pdf; ist_usr_page=1; sat_ppv=79; ajs_anonymous_id=%229225b8cf-6637-49c8-8568-ecb53cfc760c%22; ajs_user_id=null; ajs_group_id=null; __utma=59807807.316184303.1491952757.1496310296.1496310296.1; __utmc=59807807; __utmcsr=...`
```

 Demo: use `nc` to see a real request
HTTP Responses

- General form:
  - `HTTP/[version] [status code] [reason]\r\n[headerfield1]: [fieldvalue1]\r\n[headerfield2]: [fieldvalue2]\r\n[...]
[headerfieldN]: [fieldvalueN]\r\n\r\n[response body, if any] ← typically the requested resource

- a number (readable by machines)
- a string (readable by humans)
Status Codes and Reason

- **Code**: numeric outcome of the request – easy for computers to interpret
  - A 3-digit integer with the 1st digit indicating a response category
    - 1xx: Informational message
    - 2xx: Success
    - 3xx: Redirect to a different URL
    - 4xx: Error in the client’s request
    - 5xx: Error experienced by the server

- **Reason**: human-readable explanation
  - e.g. “OK” or “Moved Temporarily”
Common Statuses

- **HTTP/1.1 200 OK 😊**
  - The request succeeded and the requested object is sent

- **HTTP/1.1 404 Not Found 😞**
  - The requested object was not found

- **HTTP/1.1 301 Moved Permanently**
  - The object exists, but its name has changed
    - The new URL is given as the “Location:” header value

- **HTTP/1.1 500 Server Error 🙁**
  - The server had some kind of unexpected error
Server Headers

- The server can provide zero or more response “headers”
  - These provide information to the client or modify how the client should process the response

- You’ll encounter many in practice
  - [https://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html](https://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html)
  - **Server**: a string identifying the server software (e.g., Apache server)
  - **Content-Type**: the type of the requested object
  - **Content-Length**: size of requested object
  - **Last-Modified**: a date indicating the last time the request object was modified
A Real Response

HTTP/1.1 200 OK
Date: Mon, 21 May 2018 07:58:46 GMT
Server: Apache/2.2.32 (Unix) mod_ssl/2.2.32 OpenSSL/1.0.1e-fips
mod_pubcookie/3.3.4a mod_uwa/3.2.1 Phusion_Passenger/3.0.11
Last-Modified: Mon, 21 May 2018 07:58:05 GMT
ETag: "2299e1ef-52-56cb2a9615625"
Accept-Ranges: bytes
Content-Length: 82
Vary: Accept-Encoding,User-Agent
Connection: close
Content-Type: text/html
Set-Cookie: bbbbbbbbbbbbbbb=DBMLFDMJCGAOILMBPIIAAILGBAKOJNNMCJKKBKCDMDEJHMPONHCILPIBL
ADEXAKCIABMEEPAOPMMKAOLHOKJMIKGDIIHNCANAPHMFMBSBAPFента=PDANJAPIBOIOO0OD;
HttpOnly

<html><body>
<font color="chartreuse" size="18pt">Awesome!!</font>
</body></html>

- Demo: use `telnet` to see real responses
Cool HTTP/1.1 Features

- “Chunked Transfer-Encoding”
  - A server might not know how big a response object is
    - *e.g.* dynamically-generated content in response to a query or other user input
  - How do you send Content-Length?
    - Could wait until you’ve finished generating the response, but that’s not great in terms of latency – we want to start sending the response right away
  - Chunked message body: response is a series of chunks
Cool HTTP/1.1 Features

- Persistent connections
  - Establishing a TCP connection is costly
    - Multiple network round trips to set up the TCP connection
    - TCP has a feature called “slow start”; slowly grows the rate at which a TCP connection transmits to avoid overwhelming networks
  - A web page consists of multiple objects and a client probably visits several pages on the same server
    - **Bad idea:** separate TCP connection for each object
    - **Better idea:** single TCP connection, multiple requests
20 years later...

- World has changed since HTTP/1.1 was adopted
  - Web pages were a few hundred KB with a few dozen objects on each page, now several MB each with hundreds of objects (JS, graphics, ...) & multiple domains per page
  - Much larger ecosystem of devices (phones especially)
  - Many hacks used to make HTTP/1.1 performance tolerable
    - Multiple TCP sockets from browser to server
    - Caching tricks; JS/CSS ordering and loading tricks; cookie hacks
    - Compression/image optimizations; splitting/sharding requests
    - etc., etc. ...
HTTP/2

- Based on Google SPDY; standardized in 2015
  - Binary protocol - easier parsing by machines (harder for humans); sizes in headers, not discovered as requests are processed; ...
    - But same core request/response model (GET, POST, OK, ...)
  - Multiple data streams multiplexed on single TCP connections
  - Header compression, server push, object priorities, more...
- All existing implementations incorporate TLS encryption (https)
- Supported by all major browsers and servers since ~2015
- Used now by most major web sites
  - Coexists with HTTP/1.1
  - HTTP/2 used automatically when browser and server both support it

This is extra (non-testable) material
Polling Question

- Are the following statements True or False?

<table>
<thead>
<tr>
<th></th>
<th>Q1</th>
<th>Q2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>B</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>C</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>D</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>E</td>
<td>We’re lost…</td>
<td></td>
</tr>
</tbody>
</table>

**Q1:** A protocol only defines the “syntax” that clients and servers can communicate with. Also the semantics/meaning.

**Q2:** Clients and servers use the same header fields.
Polling Question

- Which HTTP status code family do you think the following Reasons belong to?

Q1       Q2
A. 4xx    2xx
B. 4xx    3xx
C. **5xx** 2xx
D. **5xx** 3xx
E. We’re lost...

Q1: Gateway Time-out
- 4xx: Client Fail
- 5xx: Server Fail

Q2: No Content
- 2xx: Success
- 3xx: Redirect

Q1: Gateway acting as gateway timed out (not an issue with the client request)

Q2: OK! Resource successfully retrieved, but happened to be empty
Extra Exercise #1

Write a program that:

- Creates a listening socket that accepts connections from clients
- Reads a line of text from the client
- Parses the line of text as a DNS name
- Connects to that DNS name on port 80
- Writes a valid HTTP request for “/”

```
GET / HTTP/1.1\r\nHost: <DNS name>\r\nConnection: close\r
\r
```

- Reads the reply and returns it to the client