

Hypertext Transport Protocol (and Beyond!)

CSE 333 Autumn 2025

Instructors: Naomi Alterman, Chris Thachuk

Teaching Assistants:

Ann Baturytski	Derek de Leuw	Blake Diaz
Rishabh Jain	Chendur Jel Jayavelu	Lucas Kwan
Irene Xin Jie Lau	Nathan Li	Maya Odenheim
Advay Patil	Selim Saridede	Deeksha Vatwani
Angela Wu	Jiexiao Xu	

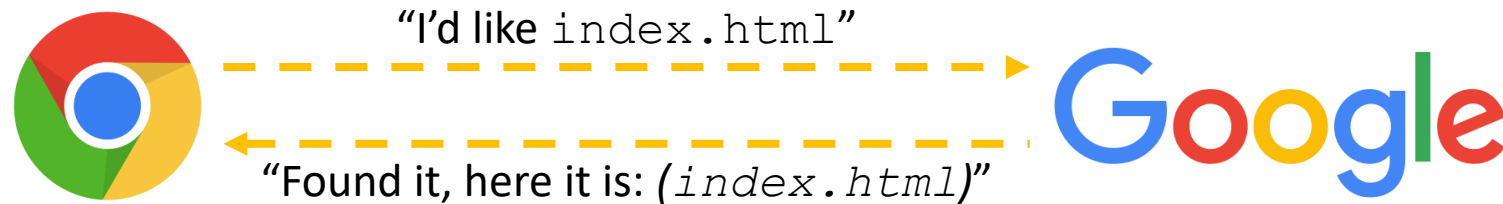
Administrivia

- ❖ EX16 due Wednesday
- ❖ HW4 out out out

Today

- ❖ HTTP
- ❖ Beyond sockets

HTTP Basics



- ❖ 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
- ❖ How do we **formalize** the way our web browser asks Google's web server for data?
 - An application layer protocol 🧐

What exactly are protocols again?

- ❖ 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?
 - What do the messages mean?
 - 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** contains the bytes of that resource
 - Or, if not, an error code expressing why the bytes aren't there
- All you could want to know (and more!):

https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

HTTP Requests

- ❖ ASCII plaintext

- ❖ 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]`

- ❖ Demo: use nc to see a real request

HTTP Methods

- ❖ Common HTTP methods used when you visit websites:
 - `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, mostly for HTTP APIs:
 - `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>
 - 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 mid 2010's (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

Client Headers

- ❖ The client can provide zero 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
 - `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
 - https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers#request_context

A Real Request

```
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=5c18d7ed6d369d56b69a1c0aa441d7
8f; SESSd47cbe79be51e625cab059451de75072=d137dbe7bbe1e90149797dcd89c639b1;
_sdsat_DMC_or_CCODE=null; _sdsat_utm_source=; _sdsat_utm_medium=; _sdsat_utm_term=;
_sdsat_utm_content=; adblock=blocked; s_fid=50771A3AC73B3FFF-3F18A
ABD559FFB5D; s_cc=true; prev_page=science.%3A%2Fcontent%2F347%2F6219%2F262%
2Ftab-pdf; ist_usr_page=1; sat_ppv=79; ajs_anonymous_id=%229225b8cf-6637-49
c8-8568-ecb53cfc760c%22; ajs_user_id=null; ajs_group_id=null; __utma=598078
07.316184303.1491952757.1496310296.1496310296.1; __utmc=59807807; __utmc=80
...
```

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]
```

❖ Demo: use **nc -C** to see a real response

- The “-C” option uses \r\n as line endings

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
 - `Server`: a string identifying the server software
 - `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
 - https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers#response_context

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:
bbbbbbbbbbbbbbbb=DBMLFDMJCGAOILMBPIIAAIFLGBAKOJNNMCJIKKBKCDMDEJHMPONHCILPIBL
ADEAKCIABMEEPAOPMMKAOLHOKJMIGMIDKIHNCANAPHMFMBLBABPFENPDANJAPIBOIOOOD;
HttpOnly

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


Evolving abstractions

❖ 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. ...

Beyond HTTP/1.x

- ❖ HTTP/2 standardized in 2015
 - Based on Google's "SPDY" protocol
 - Binary, not plaintext!
 - Multiple data streams "multiplexed" on single TCP connections
- ❖ HTTP/3 standardized in 2022
 - Switched from TCP to QUIC (a new transport layer protocol from Google)
 - Does multiplexing better (TCP was holding us baaaack 🙄)
- ❖ All use same core request/response model (URLs and methods like GET, POST, ...)
- ❖ All existing implementations incorporate TLS encryption (https)
- ❖ Widely adopted
 - All major browsers
 - 30% of websites speak HTTP/3
 - 60% speak HTTP/2
 - All speak HTTP/1.1

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

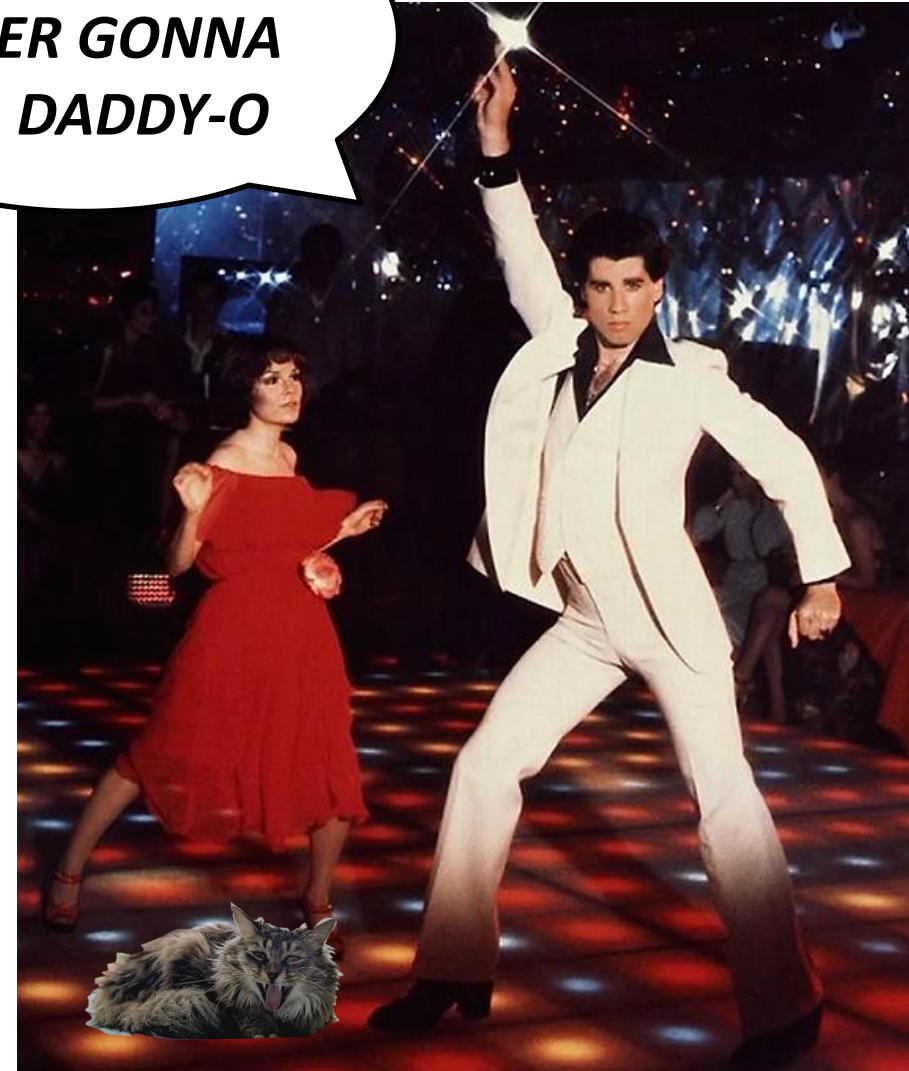
Today

- ❖ HTTP
- ❖ **Beyond sockets**

Beyond sockets?

- ❖ Sockets are a 40 year old API
 - Is there anything we'd have done differently in retrospect?
 - Are operating systems the same as they were in the 1970s?
 - Is the *internet* the same as it was in the 1970s?
- ❖ In practice, a shocking amount of the internet landscape has stayed the same
 - *It is a profoundly well engineered set of systems!!*

**AF_INET IS
NEVER GONNA
DIE, DADDY-O**



The modern internet


- ❖ In practice, a few important differences (beyond programming language and scale):
 - We **maintain sessions across connections** *all the time*, and expect continuous service
 - **Privacy and security** are more important than ever
 - **Metered** connections (connections on which you pay by-the-byte) are much more common

Apple's Network.framework

- ❖ An OS API for TCP/UDP-level network access that stands next to sockets
 - Includes affordances for modern network usage
 - <https://developer.apple.com/videos/play/wwdc2018/715/>

- ❖ Why now?
 - Most non-systems code does networking at the “application” layer of the OSI model and doesn't actually touch the sockets API
 - This makes it *much easier* to change the sockets abstraction
(*Fewer codebases to update, fewer angry developers to placate*)

Higher level transports

- ❖ A lot of internet traffic is **facilitated** by HTTP connections, if not directly transported over them
 - Eg: you open a zoom call using a web URL, but the call itself is peer-to-peer over UDP ()
- ❖ **Websockets** (circa 2011):
 - https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
 - Application-level communication
 - Uses HTTP for **connection setup** and TCP for **transport**
 - An “easy byte pipe”
 - Case study: <http://slither.io/>

Careful about reinventing the wheel

- ❖ I've seen a lot of networking research over the past decade and a half
 - And *a lot* of the same mistakes get made now that were getting made back then
- ❖ To re-invent an architecture, you need to understand it **as deeply as the people who built it**
 - Otherwise you're just pursuing an exercise in vanity
- ❖ (*"okay, thanks grandma :P"*)

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\n
Host: <DNS name>\r\n
Connection: close\r\n
\r\n
```

- Reads the reply and returns it to the client