

ECE 435 – Network Engineering

Lecture 4

Vince Weaver

`https://web.eece.maine.edu/~vweaver`

`vincent.weaver@maine.edu`

12:30pm, Barrows 125

25 January 2024

Announcements

- HW#1 was due.
- HW#2 will be posted. Write a mini-webserver.
-



http

- HyperText Transfer Protocol
RFC 2068 (1997), RFC 2616 (1999), RFC 7230 (2016)
- Make ASCII request, get a MIME-like response
- Connect with TCP socket
- Plain text request, followed by text headers
- Expects carriage returns in addition to linefeeds



http Commands

- GET *filename* HTTP/1.1
get file
- HEAD
get header (can check timestamp. why? see if cache up to date)
- PUT
send a file
- POST
append to a file (send form data)



- DELETE
remove file (not used much)
- TRACE
debugging
- CONNECT, OPTIONS



http three digit status codes

- 1xx – informational – not used much
- 2xx – Success – 200 = page is OK
- 3xx – Redirect – 303 = page moved
- 4xx – Client Error – 403 = forbidden, 404 = not found
- 5xx – Server Error – 500 = internal, 503 = try again



Example http request from browser

```
GET / HTTP/1.1
Host: 471-pi3:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/109.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```



Selected http request headers (included after GET)

- Host: server you are requesting
Can configure browser to open up helper util for this (for example, run Office if it's a word file)
- User-Agent (browser info). Can you lie? Can you leak info?
- Accept-*: type of documents can accept, compression, character set
- Authorization: if you need special permissions/login

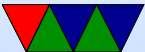


- Referer [sic] URL that referred to here
- Cookie: deals with cookies
Statelessness – how do you remember setting, logins, shopping cart, etc. “cookies”. Expire. Can be misused.
- If-Modified-Since – caching



Example http response

```
HTTP/1.1 200 OK\r\n
Date: Fri, 26 Jan 2024 04:56:25 GMT\r\n
Server: ECE435\r\n
Last-Modified: Sun, 26 Mar 2017 04:31:47 GMT\r\n
Content-Length: 64\r\n
Content-Type: text/html\r\n
\r\n
<html><head><title>Test</title></head>
<body>test</body></html>
```



Selected http response headers

- Content-Encoding, Language, Length, Type
- Last-Modified: helps with caching
- Location: used when redirecting
- Accept-Ranges: partial downloads (downloading a large file, interrupted, can restart where left off)
- Content-Length: length of file being sent
- Content-Type: type of data
- Date: current date
- Server: Name of webserver (should you be reporting



this?)



http 1.0/1.1

- HTTP 1.0, single request was sent and single response. Each file/image requested was separate TCP connection
- HTTP 1.1 (1997) supports persistent connections, allowing multiple requests to happen with one TCP connection (lowering overhead). How do you know when to close? (timeout after 60s?)
- For improved performance, open multiple simultaneous connections instead? Yes, but frowned upon (server/network load)



HTTP/2

- 2015. RFC 7540
- <https://http2.github.io/faq/>
- Google push through, extension of their SPDY (speedy)
Microsoft and Facebook giving feedback
- Why does google care about (relatively) small increases in web performance?
- Leaves a lot of high level things the same. Negotiate what level to use.
- Decrease latency of rendering web pages:



- compress headers
- Server can push data the browser didn't request yet but it knows it will need (like images, etc)
- pipeline requests
Send multiple requests without waiting for response
good on high-latency links (FIFO on 1.1, new makes it asynchronous)
- multiplex multiple requests over one TCP connection
- head-of-line blocking problem?
line of packets held up by processing of first
FIFO first requests waits until done until next, can't



run in parallel

- Page load time 10-50% faster
- While can use w/o encryption, most browsers say will only do with encryption
- Criticism: was rushed through. Is way complex. Does own flow control (has own TCP inside of TCP) Re-implements transport layer at application layer
- Can check if your web-browser implements HTTP by going to <https://http2.golang.org/>



HTTP/3 or H3

- Standardized by RFC 9000 (QUIC) and 9114 (HTTP/3)
- As of 2024 supported by most web-browsers
- Web-servers, supported by IIS and nginx, no Apache support yet
- <https://blog.apnic.net/2023/09/25/why-http-3-is-eating-the-world/>
- Uses QUIC – runs sort of custom network congestion protocol in userspace over top of UDP
- HTTP/3 started as HTTP/2 over QUIC but has developed more



- QUIC is almost more of a TCP replacement
- HTTPS only
- Can handle better roaming around switching IP addresses w/o losing connection
- Interface is no longer a sockets interface



HTTP/3 Firefox issue 2022

- <https://hacks.mozilla.org/2022/02/retrospective-and-technical-details-on-the-recent-firefox-outage/>
- Firefox stopped responding worldwide because of a bug in their HTTP/3 stack made their telemetry break a few weeks ago
- The fact that they let the telemetry break the browser is a whole other concerning tale
- But it turns out recent firefox has HTTP/3 set to automatic, and will use it if found, and google has been rolling out HTTP/3



- Part of the bug is http headers are supposed to be case-insensitive, and HTTP/2, HTTP/3 suggests they should be all lowercase, which can break your parser if you don't expect it
- Postel's Law in action?



Do you need a browser? (old)

```
telnet www.maine.edu 80
GET / HTTP/1.1
Host: www.maine.edu
(enter)(enter)
control-]
close
```



Do you need a browser? (https)

```
openssl s_client -connect www.maine.edu:443  
GET / HTTP/1.1  
Host: www.maine.edu  
(enter)(enter)
```



Do you need a browser? (HTTP2)

```
openssl s_client -connect http2.akamai.com:443  
GET / HTTP/1.1  
Host: http2.akamai.com
```

Does not work.

See <http://www.chmod777self.com/2013/07/http2-status-update.html>

But need to first send a binary SETTINGS frame.

```
50 52 49 20 2a 20 48 54 54 50  
2f 32 2e 30 0d 0a 0d 0a 53 4d
```



0d 0a 0d 0a 00 00 04 00 00 00
00 00

Then HEADERS frame, then compressed HEADERS.

Response is compressed HEADERS and DATA frames.



How simple can a server be?

- My Apple II webserver project

http://www.deater.net/weave/vmwprod/apple2_eth/



High-Level WWW Concerns



Compression

- Even with 1.1 could use deflate compression
- CRIME attack, could figure out encryption things by seeing how well values compressed (?)
- Because of this http compression is usually disabled
- http2 HPACK special compression to be resistant



What if Server Overloaded?

- Slashdot effect (modern: HackerNews?)
- caching/proxy – squid
- Content Delivery Network – akami
- Server farms



Web Security

- SSL – Secure Socket Layer
- Replaced by TLS (Transport Layer Security)
- Port 443 for https (we'll talk about soon)
- Public key encryption.



Web Privacy

- Cookies
- Cross-device tracing
- Browser Fingerprinting



Setting Up a Web-server

- Apache
- Easy to do, more difficult to secure



Web Search

- Web-bots index the web. robots.txt file
- Altavista, Hotbot, Excite, Inktomi, etc.
- Curated search like Yahoo (people organize links rather than automatically search)
- Google (1996 some machine in Stanford, 1997-1998)
- MSN search 1999, rebranded Microsoft Bing 2009



HW#2

- Can use existing server code, will connect to it with any web-browser
- Listen on port 8080 (why not 80?)
- Once browser connects, read entire request into buffer (more proper way to dynamically allocate memory?)
- Ignore most of the headers, mostly want to parse the GET request
- Generate headers for response
- Send header and file back to browser over socket



- Handle a few corner cases, like 404 errors



HW#2 Hints

- Get the header printing first, then worry about correctness of headers (dates, length))



HW#2 – Parsing for filename

- Know how to search for a string and point to location after it?
 - Find a string and point to beginning of it.

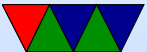
```
char *pointer;  
pointer=strstr(haystack,needle);
```
 - Look for "GET "

Actually points to beginning of GET. How to skip ahead?
 - `pointer+=4` is one way. (pointer math, ugh)
 - How to get to first space?



- `strtok(pointer, " ");`
Will split the string into chunks, put 0 at end.
- Also can do this manually;

```
pointer2=pointer;
while(*pointer) {
    if (pointer==' ') {
        *pointer=0;
        break;
    }
    pointer++;
}
printf("%s\n",pointer2);
```



HW#2 – Constructing the Headers

- Know how to construct a string on the fly? `strcat()`,
`sprintf()`
`strcpy()` first bit in.
`strcat()` additional strings.

If you want formatting you can do things like

```
sprintf(temp_string, "File size=%d\r\n", filesize);  
strcat(out_string, temp_string);
```

Create big enough buffer.



HW#2 – Calculating Content-length

- How to find size of a file?
- Can read it in, and count. Note: don't use `strlen()` for this as a binary file might have zeros in it
- Might be better to use `stat()` (`man stat.2`) need `.2` (or `man -a`) as there's a command line tool called `stat` that comes up first.



HW#2 – Reading/Writing File

- How to read/write file. There are a large number of ways to do this. `open()/read()/write()/close`
`fopen()/fread/fwrite/fclose` (careful! Buffered!
And maybe need `fdopen()` to print to file descriptor).

```
fd=open(filename ,O_RDONLY);
if (fd<0) fprintf(stderr,"Error opening %s\n",filename);
while(1) {
    result=read(fd ,buffer ,256);
    if (result<=0) break;
    write(network_fd ,buffer ,result);
}
```

Be sure to close afterward.



HW#2 – Getting Filetype

- Easiest way is calculating based on extension
- Take filename, look for . and compare after it
- Can use strstr() again, but think of corner cases
What if multiple dots? What if no dots?

