

changelog

1 Nov: `\0a \0d` → `\x0a \x0d`

HTTP: An Evolvable Narrow Waist for the Future Internet

Lucian Popa*

Patrick Wendell*

Ali Ghodsi*

Ion Stoica*

Abstract

While the Internet is designed to accommodate multiple transport and application layer protocols, a large and growing fraction of Internet traffic runs directly over HTTP. Observing that HTTP is poised to become the de-facto “narrow waist” of the modern Internet, this paper asks whether an HTTP narrow waist, compared with the an IP-layer waist, facilitates a more *evolvable* Internet. Evolvability is highly desirable for the Internet, since communication patterns change must faster than the underlying infrastructure. Furthermore, the narrow waist plays in important role in enabling or preventing architectural evolvability. We argue that HTTP is highly evolvable, due to (i) naming flexibility, (ii) indirection support and (iii) explicit middleboxes. We point to

then all applications can take advantage of such functionality. If the narrow waist is not evolvable, the applications have to either implement the functionality themselves, or wait for their protocol of choice to implement it. In fact, one could argue that the main motivation behind the flurry of recent proposals for new network architectures [9, 20, 26, 27, 39, 42] is a response to IP’s inability to evolve and support features such as content dissemination, explicit support for middleboxes, and anycast. It should come as no surprise that evolvability has recently been singled out by several clean-slate proposals as the most desirable feature of a future architecture [8, 19].

In this context, we ask the following natural question: *Is HTTP evolvable?* Despite the fact that one could convincingly argue that HTTP is already an “ossified” pro-

URL / URIs

Uniform Resource Locators (URL)

tells how to find “resource” on network

uniform — one syntax for multiple protocols (types of servers, etc.)

Uniform Resource Identifiers

superset of URLs

URI examples

`https://kytos02.cs.virginia.edu:443/cs3130-spring2023/
quizzes/quiz.php?qid=02#q2`

`https://kytos02.cs.virginia.edu/cs3130-spring2023/
quizzes/quiz.php?qid=02`

`https://www.cs.virginia.edu/`

`sftp://cr4bd@portal.cs.virginia.edu/u/cr4bd/file.txt`

`tel:+1-434-982-2200`

`//www.cs.virginia.edu/~cr4bd/3130/S2023/
/~cr4bd/3130/S2023`

scheme and/or host implied from context

URI generally

scheme://authority/path?query#fragment

scheme: — what protocol

//authority/

authorirty = user@host:port OR host:port OR user@host OR host

path

which resource

?query — usually key/value pairs

#fragment — place in resource

most components (sometimes) optional

HTTP typical flow

client

server

```
sequenceDiagram
    participant client
    participant server
    client->>server: GET / cr4bd/4457/F2024/ HTTP/1.1  
Host: www.cs.virginia.edu  
...
    server-->>client: HTTP/1.1 200 OK  
Content-Type: text/html  
Content-Length: 5432  
...  
<!DOCTYPE html...
    client->>server: GET / cr4bd/4457/F2024/main.css HTTP/1.1  
Host: www.cs.virginia.edu  
...
```

GET / cr4bd/4457/F2024/ HTTP/1.1
Host: www.cs.virginia.edu
...

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 5432
...
<!DOCTYPE html...

GET / cr4bd/4457/F2024/main.css HTTP/1.1
Host: www.cs.virginia.edu
...

HTTP message fields

requests:

- method (GET, HEAD, POST, ...) — what to do
- URI ('path' and 'query' part of URL, usually)

responses:

- status code and message (200 OK, 404 Not Found, etc.)

both:

- headers (key-value pairs)
- (sometimes) message body (arbitrary data)

HTTP/1.1 message format (RFC 2616)

ASCII text over TCP or TLS

all newlines use 'CRLF' (`\x0d\x0a` = `\r\n`)

request

```
method URI HTTP/1.1  
header-name: header-value  
header-name: header-value  
  
(depending on method) message-body
```

response

```
HTTP/1.1 status-code status message  
header-name: header-value  
header-name: header-value  
  
(depending on method+status code) message-body  
(depending on headers) header-name: header-value
```

HTTP/2, HTTP/3

‘new’ versions, not ubiquitously deployed

HTTP/2: over TCP *or* over TLS over TCP

HTTP/3: over QUIC over UDP

multiple ‘streams’ within one connection

send series of ‘frames’ with stream ID + type + data

frame types include:

HEADERS — encode message headers (key/value pairs)

DATA — include message bodies

method, status-code, URI encoded as special headers

HTTP/1.1 example (GET)

```
GET /~cr4bd/4457/F2024/ HTTP/1.1
Host: www.cs.virginia.edu
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
sec-ch-ua: "Chromium";v="130", "Google Chrome";v="130", "Not?A_Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
```

```
HTTP/1.1 200 OK
Date: Sun, 27 Oct 2024 02:08:48 GMT
Server: Apache/2.4.52 (Ubuntu)
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 1665
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
```

```
.....Xmo.6..._..Z...1l...+.5kR..T...-q.DM...]. ....#9N.....E.sw.....>...D.b.`_D...9,s...h...i.4%aB....)....;.....~+.z.|p.?
```

HTTP/2.0 example (GET request)

```
▶ Frame 182: 620 bytes on wire (4960 bits). 620 bytes captured on interface 0
▶ Ethernet II, Src: [REDACTED]
▶ Internet Protocol Version 6, Src: [REDACTED]
▶ Transmission Control Protocol, Src Port: 44774, Dst Port: 443
▶ Transport Layer Security
▶ HyperText Transfer Protocol 2
  ▶ Stream: HEADERS, Stream ID: 15, Length 490, GET /wiki/Main_Page
    Length: 490
    Type: HEADERS (1)
    ▶ Flags: 0x25, Priority, End Headers, End Stream
      0...0000 0000 0000 0000 0000 0000 1111 = Stream ID
      [Pad Length: 0]
      0...0000 0000 0000 0000 0000 0000 1101 = Stream ID
      Weight: 41
      [Weight real: 42]
      Header Block Fragment: 82058b63c1ba998d0335516b1cc
      [Header Length: 877]
      [Header Count: 22]
    ▶ Header: :method: GET
    ▶ Header: :path: /wiki/Main_Page
    ▶ Header: :authority: en.wikipedia.org
    ▶ Header: :scheme: https
    ▶ Header: user-agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/109.0
    ▶ Header: accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
    ▶ Header: accept-encoding: gzip, deflate, br, zstd
    ▶ Header: cookie: WMF-Last-Access=27-Oct-2024
    ▶ Header: cookie: WMF-Last-Access-Global=27-Oct-2024
    ▶ Header: cookie: NetworkProbeLimit=0.001
    ▶ Header: cookie: GeoIP=US:VA:Charlottesville:38.03:-78.07
    ▶ Header: cookie: enwikimwuser-sessionId=2771b23b954
    ▶ Header: upgrade-insecure-requests: 1
    ▶ Header: sec-fetch-dest: document
    ▶ Header: sec-fetch-mode: navigate
```

```
0000 00 01 ea 01 25 00 00 00 0f 00 00 00 0d 29 82 05 .....%.....
0010 8b 63 c1 ba 99 8d 03 35 51 6b 1c c5 41 8c 2d 4b ..c.....5 Qk...
0020 f8 37 53 56 59 0c 35 cf 64 df 87 7a bb d0 7f 66 ..7SVY-5- d-2
0030 a2 81 b0 da e0 53 fa cf 08 7e d4 e1 1d b5 26 df .....S.....
0040 b5 33 9a ab 7c a9 e5 e7 22 71 af b5 2c ef 70 2c ..3-|... "q...
0050 85 70 7f 6a 62 29 3a 9d 81 00 20 00 40 15 30 9a ..p-jb):.....
0060 c2 ca 7f 2c 05 90 ae 0f 53 d3 49 7c a5 89 d3 4d ..C...A...3...
0070 1f 43 ae ba 0c 41 a4 c7 a9 8f 33 a6 9a 3f df 9a ..h-u-b-& =Ly...
0080 68 fa 1d 75 d0 62 0d 26 3d 4c 79 a6 8f be d0 01 w-H+... 1~t...
0090 77 fe 8d 48 e6 2b 03 ee 69 7e 8d 48 e6 2b 1e 0b ..F-s.....-h...
00a0 1d 7f 46 a4 73 15 8a ea 9b e8 d4 8e 62 b0 8e f3 Kp-Z-@...P...
00b0 7f bf 34 d1 f5 f2 c7 cf df 68 00 bb df 51 8b 2d RB-@-#...Or...
00c0 4b 70 dd f4 5a be fb 40 05 db 50 92 9b d9 ab fa ..4.....-h...
00d0 52 42 cb 40 d2 5f a5 23 b3 e9 4f 68 4c 9f 60 94 Kp-Z-@...P...
00e0 e5 a3 0a d9 c6 84 ad 09 08 54 22 01 3a b6 a2 25 RB-@-#...Or...
00f0 61 00 4d 7f 7e 99 e5 a3 0a d9 c6 84 ad 09 08 54 ..T"...
0100 21 6c 54 1e 31 d1 00 9d 5b 51 12 b0 80 26 bf 7f a-M~.....
0110 00 91 d2 54 f8 3d 9d 75 d8 78 cb 9c d4 99 30 01 !LT-1... [Q...
0120 70 00 3f 7f 01 a1 c4 53 e4 d7 07 0d d7 38 c3 72 ...T-=u-x...
0130 f4 e3 b2 83 a5 25 47 73 51 41 6e 32 f2 e0 66 e2 p-?-S...
0140 ce bc bb 4f dc ee d7 7f 02 9f 2d 5e 0d d4 d4 f8 ...%Gs QAn2...
0150 b5 05 b1 64 15 08 31 ea c9 24 02 75 d0 c6 26 63 ...0...../...
0160 7d b6 99 7d b9 43 20 86 3f 40 92 b6 b9 ac 1c 85 }-}C-?@...
0170 58 d5 20 a4 b6 c2 ad 61 7b 5a 54 25 1f 81 0f 40 X-...a {ZT9...
0180 8a 41 48 b4 a5 49 27 5a 42 a1 3f 86 90 e4 b6 92 ..AH-I'Z B-?...
0190 d4 9f 40 8a 41 48 b4 a5 49 27 5a 93 c8 5f 86 a8 ..@-AH-I'Z...
01a0 7d cd 30 d2 5f 40 8a 41 48 b4 a5 49 27 59 06 49 }-@_@A H-...
01b0 7f 83 a8 f5 17 40 8a 41 48 b4 a5 49 27 5a d4 16 .....@A H-...
01c0 cf 82 ff 03 40 86 ae c3 1e c3 27 d7 85 b6 00 7d .....@...'.
01d0 28 6f 40 85 ae c1 cd 48 ff 86 a8 eb 10 64 9c bf (o@...H...
01e0 58 86 a8 eb 10 64 9c bf 40 82 49 7f 86 4d 83 35 X...d-@-I...
01f0 05 b1 1f 00 00 04 08 00 00 00 00 0f 00 be 00 00 .....
```

selected HTTP methods

| method | purpose | request body? | responses body? | 'safe' |
|---------|---------------------------|---------------|-----------------|--------|
| GET | retrieve resource | never | usually | yes |
| HEAD | retrieve resource headers | never | never | yes |
| POST | provide data | always | usually | no |
| PUT | set contents of resource | always | maybe | no |
| DELETE | delete resource | never | maybe | no |
| OPTIONS | get info about server | maybe | maybe | no |

safety

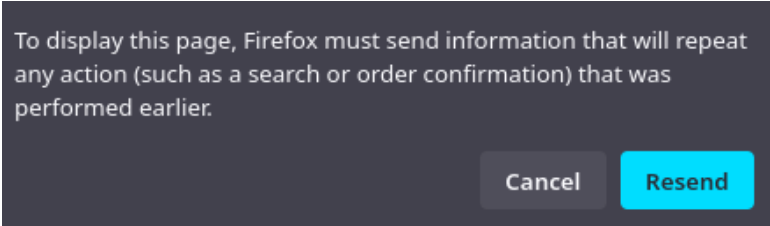
GET, HEAD = 'safe' methods

okay for clients to repeat, send unprompted

- 'prefetch' resources

- redo when user presses back button unprompted

other methods: that's not okay!

A screenshot of a Firefox security warning dialog box. The background is dark gray. The text is white and reads: "To display this page, Firefox must send information that will repeat any action (such as a search or order confirmation) that was performed earlier." At the bottom right, there are two buttons: "Cancel" in a light gray button and "Resend" in a bright cyan button.

To display this page, Firefox must send information that will repeat any action (such as a search or order confirmation) that was performed earlier.

Cancel

Resend

HTTP POST

POST /cs4457-fall2024-quiz-listener.php HTTP/1.1

Host: kytos02-noauth.cs.virginia.edu

Content-Type: application/json

Content-Length: 184

...

{"user":"cr4bd","realuser":"cr4bd","session_id":"abcdefabcdef0"

HTML forms (GET)

Name:

Query:

```
<form action="https://example.com/foo" method="get">  
Name: <input type="text" name="name"><br>  
Query: <input type="text" name="query"><br>  
<input type="submit" value="Submit">  
</form>
```

GET /foo?name=Some+Name&query=the+thing+to+find%21 HTTP/1.1
Host: example.com
...

HTML forms (POST)

Name:

Comment:

A comment
in
several lines.

Submit

```
<form action="https://example.com/foo" method="post">  
Name: <input type="text" name="name"><br>  
Comment:  
<textarea name="comment">  
</textarea>  
<br>  
<input type="submit" value="Submit">  
</form>
```

POST /foo HTTP/1.1

Host: example.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 60

...

name=Some+Name&comment=A+comment%0D%0Ain%0D%0Aseveral+lines.

HTML forms (multipart/form-data)

```
<form action="https://example.com/foo" method="post"
      enctype="multipart/form-data">
  ...
```

```
POST /foo HTTP/1.1
Host: example.com
Content-Type: multipart/form-data; boundary=-----81545828010
Content-Length: 321
...
```

```
-----30871118663472832060210928793
Content-Disposition: form-data; name="name"
```

Some Name

```
-----30871118663472832060210928793
Content-Disposition: form-data; name="comment"
```

A comment
in
several lines.

```
-----30871118663472832060210928793--
```

GET v POST

GET

works with back button, caches
limited by URL size
saving URL accesses page again

only simple text fields

POST

not resent automatically
huge possible size
form info never 'leaked' in browser
history, referer, etc.
supports file uploads (via
multipart/form-data)

exercise: which method

GET or POST or something else for

image that shows a clock with current time

rating a product and displaying the resulting summary of all ratings

search query for a Twitter-like website

getting the 2nd page of search results

multiple names, one IP

```
$ dig +short es.wikipedia.org aaaa  
dyna.wikimedia.org.
```

```
2620:0:860:ed1a::1
```

```
$ dig +short en.wikipedia.org aaaa  
dyna.wikimedia.org.
```

```
2620:0:860:ed1a::1
```

es.wikipedia.org = Spanish Wikipedia

en.wikipedia.org = English Wikipedia

how does this work?

Host/:authority header

when getting `http://somehostname/path`, send header

`Host: somehostname (HTTP/1.1)`

`:authority: somehostname (HTTP/2, HTTP/3)`

allows for 'virtual hosts'

selected HTTP status codes

1xx — informational

2xx — successful

200 OK, 204 No Content

3xx — redirection

301 Moved Permanently, 302 Found, 303 See Other

'Location' header gives next URL to use

304 Not Modified (conditional GET — later)

4xx — client error

403 Forbidden, 404 Not Found

5xx — server error

HTTP redirects

```
HTTP/1.1 301 Moved Permanently  
Location: https://foo.com/quux/bar  
Content-Type: text/plain
```

(This text may be shown by clients that don't process the redirect automatically, or if there's a problem following it. It is up to the server what to put here, but typical might be something like:)

```
Redirecting to https://foo.com/quux/bar
```

HTTP redirect codes

a bunch of different status codes:

- 301 Moved Permanently

- 302 Found

- 303 See Other

- 307 Temporary Redirect

- 308 Permanent Redirect

mostly behave all the same, but...

- POST request receiving 301/302 redirects into GET request

HTTP error pages

HTTP/1.1 403 Forbidden

Content-Type: text/html

Content-Length: ..

[This can be a full web page that is displayed....]

error status codes can still have full responses

web browsers will usually render response normally

delay for errors

PUT /some/file/location HTTP/1.1

Content-Length: 5368709120

(lots of data)

HTTP/1.1 403 Forbidden

...

...

100 continue (error case)

```
PUT /some/file/location HTTP/1.1  
Content-Length: 5368709120  
Expect: 100-continue
```

```
HTTP/1.1 403 Forbidden
```

```
...
```

```
...
```

100 continue (no error case)

```
PUT /some/file/location HTTP/1.1  
Content-Length: 5368709120  
Expect: 100-continue
```

```
HTTP/1.1 100 Continue
```

(now send lots of data)

if server does not support (good case)

```
PUT /some/file/location HTTP/1.1  
Content-Length: 5368709120  
Expect: 100-continue
```

```
HTTP/1.1 417 Expectation Failed  
...
```

```
PUT /some/file/location HTTP/1.1  
Content-Length: 5368709120
```

(lots of data)

if server does not support (bad case)

PUT /some/file/location HTTP/1.1

Content-Length: 5368709120

Expect: 100-continue

client
waits a while, but gets not response

(lots data)

one connection, multiple requests

HTTP/0.9, HTTP/1.0 — one request+response per connection
big efficiency problem

solution 1: persistent connections

solution 2: pipelining

solution 3 (HTTP/2+): multiple 'streams' in one connection

end-of-request/response

body of request/response can be variable length

so when does request/response end if it has a body?

HTTP/1.0 original solution (RFC 1945)

“the length of that body may be determined in two ways. If a Content-Length field is present, the value in bytes represents the length of the Entity-Body. Otherwise, the body length is determined by the closing of the connection by the server.”

advantage of latter idea: don't need to generate whole document before sending headers

disadvantage: no persistent connections!

chunked transfer coding

HTTP/1.1 200 OK

Content-Type: text/plain

Transfer-Coding: chunked

...

1B

This is 0x1B bytes of text.

21

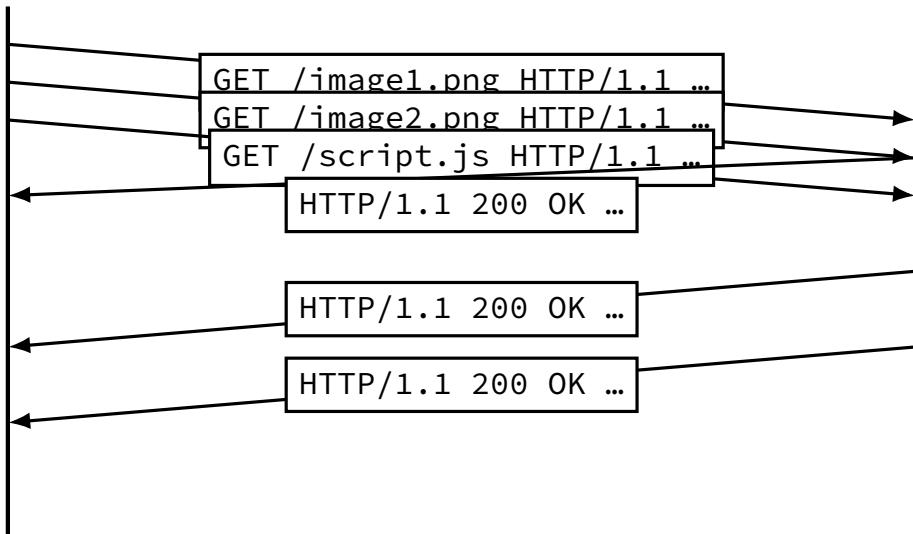
And 0x21 bytes
with more lines.

0

pipelining

client

server



HTTP/1.1 'pipelining'

send series of requests before receiving any response

potentially server can potentially process requests in parallel

need to handle resending requests if connection dropped early

HTTP/2.0 multiple streams

tcp.stream eq 76 && http2							
No.	Time	Source	Destination	TTL	Protocol	Length	Info
4451	14.668110937	2606:8e...	2620:0...		HTTP2	256	Magic, SETTINGS[0], WINDOW_UPDATE[0], PR
4454	14.668544496	2606:8e...	2620:0...		HTTP2	453	HEADERS[15]: GET /wiki/, WINDOW_UPDATE[15]
4466	14.672022778	2620:0:...	2606:8...		HTTP2	138	SETTINGS[0], SETTINGS[0]
4472	14.672439045	2620:0:...	2606:8...		HTTP2	1342	HEADERS[15]: 301 Moved Permanently
4473	14.672445467	2606:8e...	2620:0...		HTTP2	117	SETTINGS[0]
4478	14.674546658	2606:8e...	2620:0...		HTTP2	267	HEADERS[17]: GET /wiki/Main_Page, WINDOW
4488	14.680438054	2620:0:...	2606:8...		HTTP2	7226	HEADERS[17]: 200 OK, DATA[17]
4492	14.683155746	2620:0:...	2606:8...		HTTP2	500	DATA[17] (text/html)
4514	14.697392727	2606:8e...	2620:0...		HTTP2	416	HEADERS[19]: GET /w/load.php?lang=en&mod
4515	14.698099486	2606:8e...	2620:0...		HTTP2	228	HEADERS[21]: GET /w/load.php?lang=en&mod
4516	14.698277728	2606:8e...	2620:0...		HTTP2	216	HEADERS[23]: GET /w/load.php?lang=en&mod
4517	14.698335626	2606:8e...	2620:0...		HTTP2	258	HEADERS[25]: GET /static/images/icons/wil
4518	14.698377545	2606:8e...	2620:0...		HTTP2	207	HEADERS[27]: GET /static/images/mobile/c
4519	14.698418020	2606:8e...	2620:0...		HTTP2	205	HEADERS[29]: GET /static/images/mobile/c
4528	14.701618974	2620:0:...	2606:8...		HTTP2	8871	HEADERS[19]: 200 OK, DATA[19], DATA[19]
4535	14.703327863	2620:0:...	2606:8...		HTTP2	6903	HEADERS[21]: 200 OK, DATA[21], DATA[21]
4537	14.703804353	2620:0:...	2606:8...		HTTP2	3213	HEADERS[23]: 200 OK, DATA[23] (text/css)
4552	14.705737741	2620:0:...	2606:8...		HTTP2	8876	HEADERS[25]: 200 OK, DATA[25] (PNG), HEA
4557	14.708977267	2606:8e...	2620:0...		HTTP2	236	HEADERS[31]: GET /w/load.php?lang=en&mod
4558	14.709304999	2606:8e...	2620:0...		HTTP2	342	HEADERS[33]: GET /w/load.php?lang=en&mod
4559	14.709620107	2606:8e...	2620:0...		HTTP2	854	HEADERS[35]: GET /w/load.php?lang=en&mod
4581	14.713570149	2620:0:...	2606:8...		HTTP2	12938	HEADERS[31]: 200 OK, DATA[31] (text/java:
4583	14.713938877	2620:0:...	2606:8...		HTTP2	4106	DATA[33], DATA[33] (text/javascript)
4587	14.714649293	2620:0:...	2606:8...		HTTP2	16791	HEADERS[35]: 200 OK, DATA[35], DATA[35]
4590	14.715338880	2620:0:...	2606:8...		HTTP2	32788	DATA[35]
4592	14.716435166	2620:0:...	2606:8...		HTTP2	32930	DATA[35]

exercise

assuming TCP with 1500 byte MTU:

how many packets GETing 1k HTML page + two 1k images on that page

...with separate connections?

...with one connection + pipelining? HTTP/2 streams?

trailers

```
GET /foo?bar HTTP/1.1
TE: trailers
...
```

```
HTTP/1.1 200 OK
Transfer-Coding: chunked
Trailer: Expires
Date: Wed, 30 Oct 2024 23:57:04 GMT
```

```
12343
...
...
42342
...
...
0
```

```
Expires: Mon, 4 Nov 2024 23:57:04 GMT
```

content negotiation

Firefox on my desktop → wikipedia:

accept:

text/html,application/xhtml+xml,application/xml;q=0.9,image/avif;q=0.8,application/javascript;q=0.7,*/*;q=0.5

list of formats and preference indicator for each (q)
described using “media types” (RFC 6838)

accept-language: en-US,en;q=0.5

accept-encoding: gzip, deflate, br, zstd
allowed compression formats

advice against content negotiation

current HTTP standard (RFC 9110) says this approach “has several disadvantages”:

- advises considering approaches where client chooses version

- ‘impossible for the server to accurately determine what might be “best” ’

- ‘having the [client] describe its capabilities in every request can be very inefficient ...and a potential risk to the user’s privacy’

- ‘complicates the implementation’

- ‘limits ...shared caching’

HTTP authentication (RFC 7235)

relatively uncommon today compared to cookies (next topic)

server→ client: 401 Unauthorized with *WWW-Authenticate* header

typically: web browser shows login prompt

client→ server: Authorization header

example: Authorization: Basic dGVzdDoxMjPCow==
test:123 in Base64 (username test, password 123)

example: Authorization: Bearer mF_9.B5f-4.1JqM
bearer 'token' obtained out of band (common in application 'APIs')

HTTP authentication (RFC 7235)

New Access Token

Access tokens are what allow third-party applications to access Canvas resources on your behalf. These tokens are normally created automatically for applications as needed, but if [you're developing a new or limited project](#) you can just generate the token from here.

Purpose *

Expiration date

Expiration time

Leave the expiration fields blank for no expiration.

Cancel Generate Token

relatively unco

server→ client

typically: w

client→ server

example: A

test:123

example: Authorization: Bearer mF_9.B5f-4.1JqM

bearer 'token' obtained *out of band* (common in application 'APIs')

topic)

icate header

OW==

HTTP non-state

HTTP is a 'stateless'

each request stands on its own

- processed independently of all other requests
- even if multiple in a connection

this is disappointing for websites:
supporting 'login' functionality
supporting user preferences

HTTP cookies (RFC 6265)

example.com → client

HTTP/1.1 200 OK

Set-Cookie: SessionID=31d4d96; Path=/; Domain=example.com

client → example.com on later requests:

GET /some-path HTTP/1.1

Host: example.com

Cookie: SessionID=31d4d96

session ID concept

assign random ID number to each 'session' if no cookie set

in some database:

if they add to shopping cart, associate ID number with shopping cart items

if user logs in, associate ID number with user

...

selected cookie attributes

domain — limit to subset of domains

domain=example.com matches example.com, foo.example.com, but not other.com

secure — only send back on encrypted connections

httponly — do not expose to in-webpage scripts

expires, max-age — limit how long cookie kept around

default = until browser closed

cookies and tracking

cookies often used for tracking users *across websites*

and not by setting cookies valid for tons of domains

how: websites load data from other websites

separate HTTP requests with separate cookies

cookie tracking example

foo.com, bar.com, quux.com all include an image

`https://tracker.com/track-XXX.png` where XXX is foo, bar or quux

tracker.com can read cookie every time image is accessed

and set a cookie to unique number if not set

now tracker.com knows:

when/if every visitor of foo.com visited bar.com and/or quux.com

more detailed tracking?

“just” learned about how many visitors visited combinations of websites

with some cooperation can get more info:

- which subpages on those websites

- username or email entered into those websites

- ...

one way to pass info: add extra data to image filename

third party cookie rules

some browsers might restrict 'third-party cookies'
cookies sent to Y because of visit to X

various options, with variable deployment:

- only make third-party cookies work if marked SameSite=None
- separate cookie storage for each 'root' website
- ignore cookies from unvisited sites
- disable only cookies that heuristically look like tracking

cookie exercise

exercise

time	IP	path	cookie header	set-cookie header
1pm	1.2.3.4	/foo	SID=1234	—
1pm	1.2.3.4	/bar	—	SID=9999
2pm	1.2.3.4	/foo	—	SID=2345
3pm	2.3.4.5	/quux	SID=2345	—
4pm	1.2.3.4	/	SID=1234	—
4pm	3.4.5.6	/foo	SID=2345	—
5pm	1.2.3.4	/quux	SID=1234	—
6pm	1.2.3.4	/quux	—	SID=3456

exercise: how many unique users?

exercise: how many IPs per user?

HTTP caching (RFC 9111)

making webpages fast — let clients cache values for later

some problems:

how to tell if something's out of date

how to tell if changes to cookies/accept-language/etc. change item

is it out of date? options

expire date; max-age in seconds

check with server if it has changed

is it out of date? options

expire date; max-age in seconds

check with server if it has changed

expires

HTTP/1.1 200 OK

Date: Mon, 28 Oct 2024 00:29:02 GMT

Expires: Mon, 28 Oct 2024 04:29:02 GMT

...

HTTP/1.1 200 OK

Cache-Control: max-age=14400

...

aside: why date + expire

server time and client time might differ

makes Expires idea not great...

is it out of date? options

expire date; max-age in seconds

check with server if it has changed

conditional GETs

```
GET /3/library/struct.html HTTP/1.1  
...
```

```
HTTP/1.1 200 OK  
Date: Sun, 27 Oct 2024 20:01:15 GMT  
Last-Modified: Sun, 27 Oct 2024 18:50:46 GMT  
ETag: 671e8b86-13e32
```

```
GET /3/library/struct.html HTTP/1.1  
If-Modified-Since: Sun, 27 Oct 2024 18:50:46 GMT  
If-None-Match: 671e8b86-13e32  
...  
HTTP/1.1 304 Not Modified  
...
```

variable responses

HTTP/1.1 200 OK

...

Vary: Accept, Accept-Language, Cookie

...

page contents may vary even though URL doesn't change

Vary header says what things need to be the same

typically used to discard cached responses

other cache-control settings

seen max-age=X, also...

no-store

do not store a copy of this response

no-cache

do not use without checking for new version first (conditional GET or similar)

private, public

indicate if acceptable for cache shared between users

caches as cookies

let's say we load an image

with unique ETag each time

browser stores ETag, makes If-None-Match request

...kinda acts like cookie

but not susceptible to third-party cookie rules

part of set of ideas called 'supercookies'

Firefox supercookie mitigations

`https://blog.mozilla.org/security/2021/01/26/supercookie-protections/`

for each top-level website, separate:

caches (for everything — images, resolved domain names, fonts, etc.)

connections (even for same hostname)

HTTP proxies (1)

Connection Settings

×

Configure Proxy Access to the Internet

☐ No proxy

☐ Auto-detect proxy settings for this network

☒ Use system proxy settings

☐ Manual proxy configuration

HTTP Proxy

Port

0

☐ Also use this proxy for HTTPS

HTTPS Proxy

Port

0

SOCKS Host

Port

0

☐ SOCKS v4

☒ SOCKS v5

☐ Automatic proxy configuration URL

Reload

No proxy for

Example: .mozilla.org, .net.nz, 192.168.1.0/24

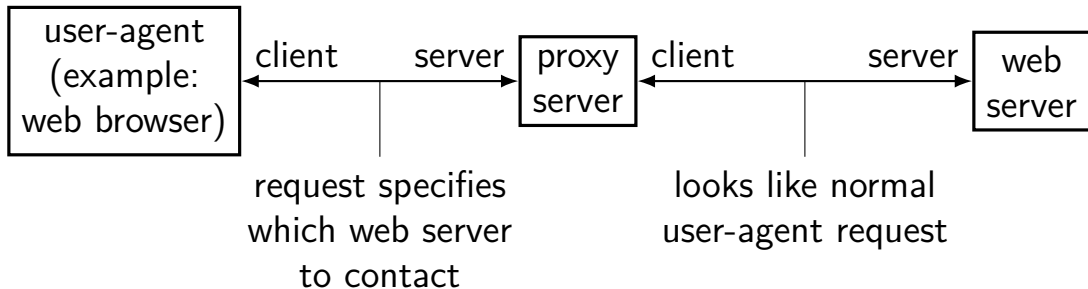
Connections to localhost, 127.0.0.1/8, and ::1 are never proxied.

☐ Do not prompt for authentication if password is saved

☐ Proxy DNS when using SOCKS v5

Cancel

OK



HTTP proxies (2)

browser→HTTP(S) proxy sever:

```
GET http://example.com/somesite HTTP/1.1
```

```
Host: example.com
```

```
...
```

instead of path, can put full URL

doesn't have to be http URL

proxy functionality

- caching for multiple users

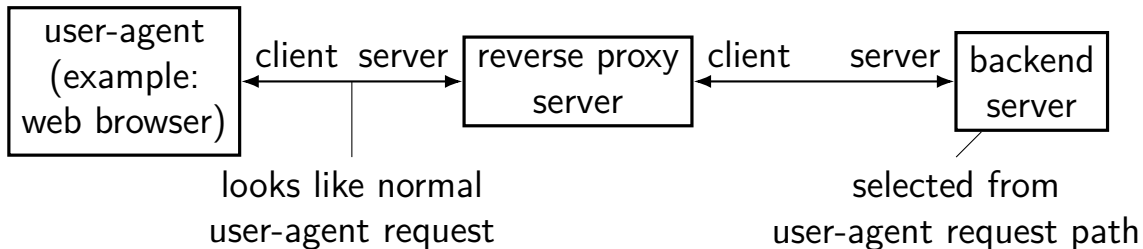
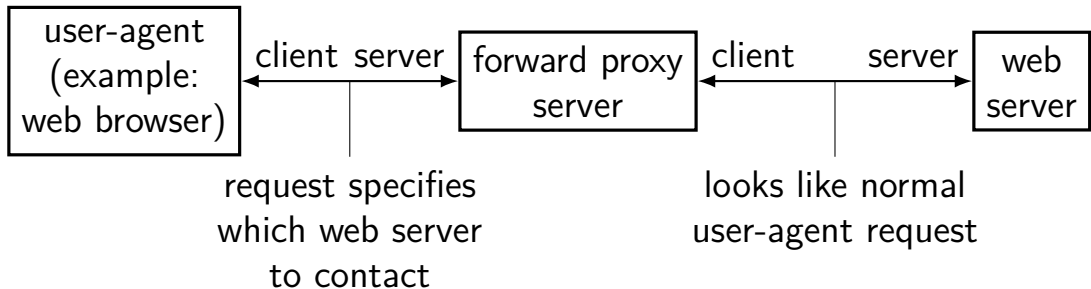
 - reason for Cache-Control: private

- filtering content

 - antimalware, adblocking, etc.

- logging content (example: debugging webapp)

- ...



reverse proxy

why not just go directly to actual web server?

make multiple web servers appear as one? example:

`https://example.com/foo/XXX` goes to

`https://foo-backend.example.com/XXX`

`https://example.com/bar/XXX` goes to

`https://bar-backend.example.com/XXX`

`https://example.com/` goes to `https://frontpage.example.com/`

do caching, filtering, or similar on behalf of web servers

split requests between multiple identical servers for performance

non-HTTP in HTTP proxy

client → server:

CONNECT ns.foo.com:53 HTTP/1.1

Host: ns.foo.com:50

server → client:

HTTP/1.1 200 OK

Some-header: some-value

client → server: (dns request)

server → client: (dns response from ns.foo.com)

client → server: (dns request)

server → client: (dns response from ns.foo.com)

...

CONNECT

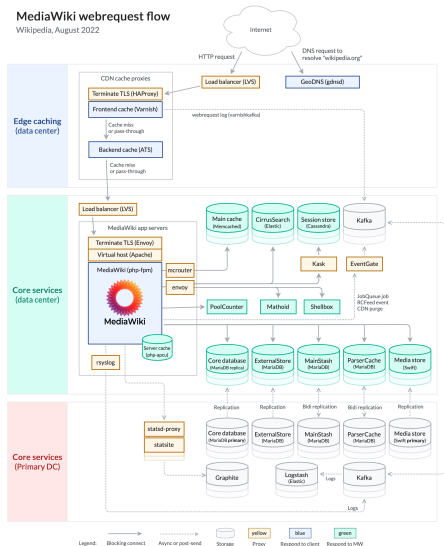
allows “tunnelling” arbitrary TCP connections through HTTP

often not implemented by HTTP proxies and/or very restricted

Wikimedia architecture

MediaWiki webrequest flow

Wikipedia, August 2022



single-sign on

client → foo.com: GET /foo

foo.com → client: redirect to
<https://sso.com/login?from=foo.com&...>

client → sso.com: GET /login?from=foo.com&...
sent with cookies set by sso.com

sso.com → client: web page with form action=<http://foo.com/...>
and method=post
data in form tells foo.com about username, etc.
cryptographically signed or similar
remember login: possibly script to submit automatically

client → foo.com: POST /... with data from sso.com

single-sign on and third-party cookie awkwardness

single-sign on flow also gives a way to do tracking...

not 'third-party' cookies since the 'root' web page becomes the single-sign on site

REST

REpresentational State Transfer

idea for application interface on top of HTTP

entities in system represented with URLs

GET requests to get state of that entity

PUT and/or POST requests to update entity state

DELETE requests to remove entity

example: Canvas API for announcements (1)

client → canvas HTTP server:

```
GET /api/v1/courses/123456/discussion_topics?only_announcements=true  
Authorization: Bearer [secret code]
```

```
HTTP/1.1 200 OK
```

```
...
```

```
[{  
  "id":1,  
  "title":"Welcome to the Course!",  
  "message":"...",  
  ...  
},  
{  
  "id":2,  
  ...  
}]
```

example: Canvas API for announcements (2)

client → canvas HTTP server:

POST /api/v1/courses/123456/discussion_topics

Authorization: Bearer [secret code]

Content-Type: application/json

...

```
{  
  "is_announcement":true,  
  "title":"Class Cancelled",  
  "message":"....."  
}
```

HTTP/1.1 200 OK

...

```
{  
  "id": 41,  
  "title":"Class Cancelled",  
  ....  
}
```

example: Canvas API for announcements (3)

client → canvas HTTP server:

PUT /api/v1/courses/123456/discussion_topics/41

Authorization: Bearer [secret code]

Content-Type: application/json

...

```
{  
  "is_announcement":true,  
  "title":"Class Cancelled [updated!]",  
  "message":"UPDATE: previously,.."  
  ...  
}
```

HTTP/1.1 200 OK

...

```
{  
  "id": 41,  
  "title":"Class Cancelled [updated!]",  
  ....  
}
```

backup slides