changelog

1 Nov: \0a \0d → \x0a \x0d

2012 opinion piece

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

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.)

Unifrom Resources 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/

authoirty = user@host:port OR host:port OR user@host OR host

path

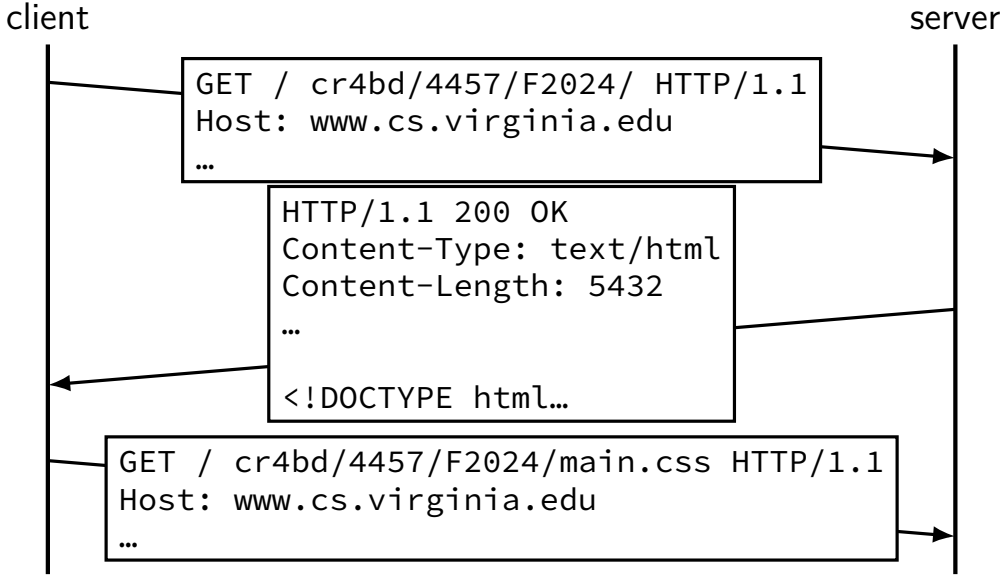
which resource

?query — usually key/value pairs

#fragment — place in resource

most components (sometimes) optional

HTTP typical flow



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

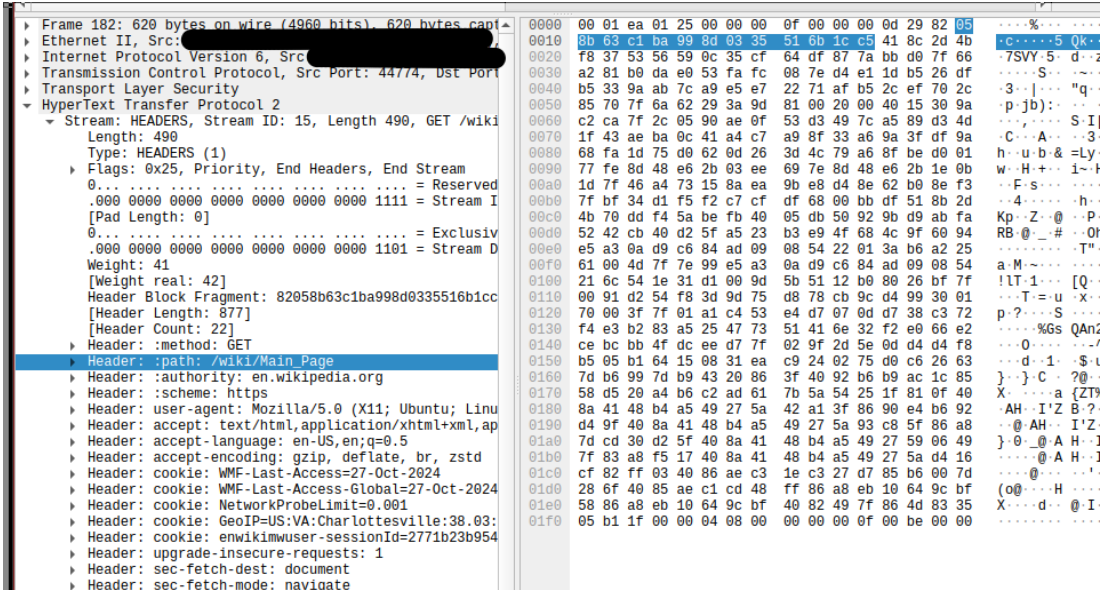
HTTP/1.1 example (GET)

```
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)



HTTP/1.1 example (POST)

selected HTTP methods

method	purpose	request body?	respones 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

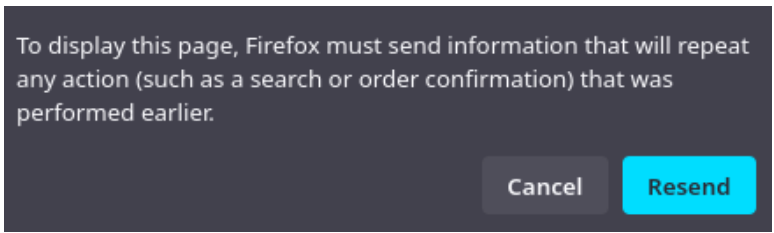
safety

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!

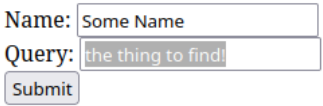


HTTP POST

```
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":"abcdefabcdef
```

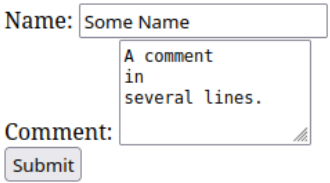
HTML forms (GET)




```
<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)




```
<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=---------------------------8154582801
Content-Length: 321
...

-----------------------------308711186634728320602109287 93
Content-Disposition: form-data; name="name"

Some Name
-----------------------------308711186634728320602109287 93
Content-Disposition: form-data; name="comment"

A comment
in
several lines.
-----------------------------308711186634728320602109287 93--
```

GET v POST

GET	POST
works with back button, caches	not resent automatically
limited by URL size	huge possible size
saving URL accesses page again	form info never ‘leaked’ in browser history, referer, etc.
only simple text fields	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 proce
automatically, or if there's a problem following it
to the serer what to put here, but typical might be

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 usualy 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

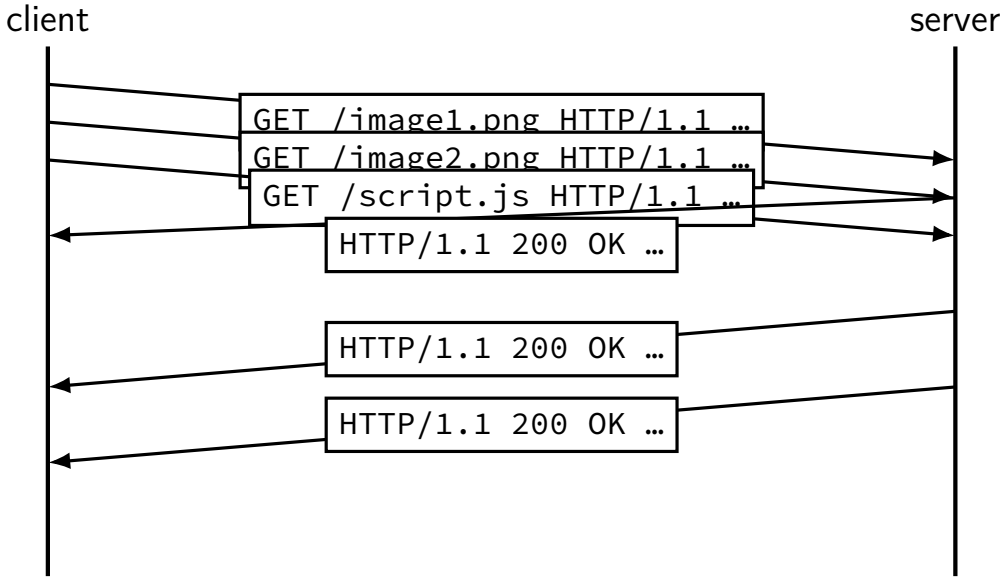
disadvantage: no persistent connections!

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
GET /image1.png HTTP/1.1 …
GET /image2.png HTTP/1.1 …
GET /script.js HTTP/1.1 …
HTTP/1.1 200 OK …
HTTP/1.1 200 OK …
HTTP/1.1 200 OK …

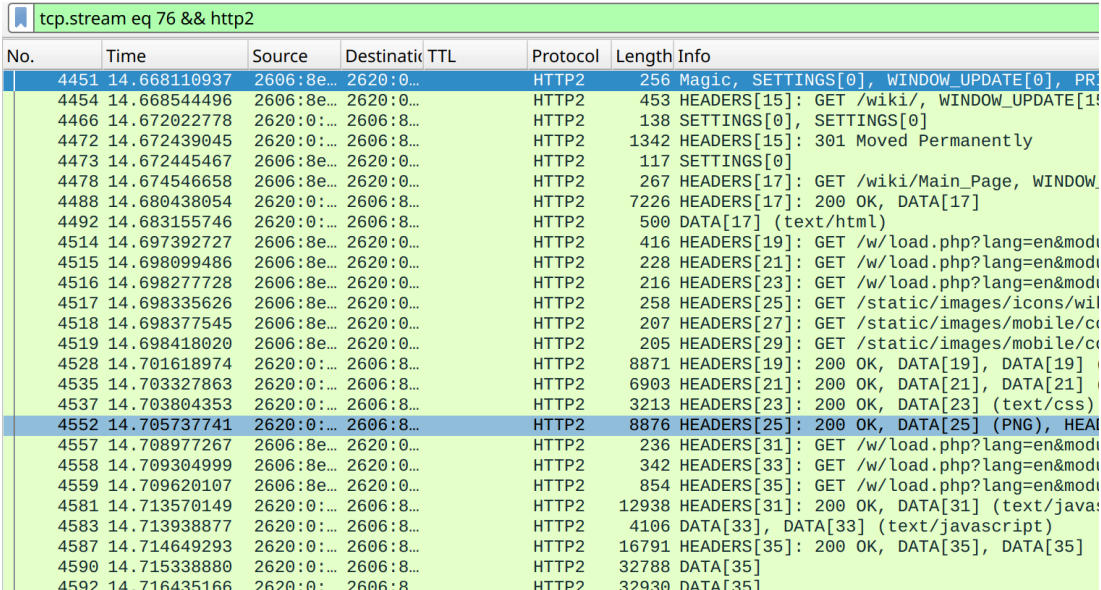
HTTP/1.1 ‘pipelining’

send series of requests before receiving any response

potentailly server can potentially process requests in parallel

need to handle resending requests if connection dropped early

HTTP/2.0 multiple 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/a
```

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 negotation

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 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=31d4d96e407aad42; Path=/; Domain=example
```

client → example.com on later requests:

```
GET /some-path HTTP/1.1
Host: example.com
Cookie: SessionID=31d4d96e407aad42
```

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-Lnaguage, 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 cokie rules

part of set of ideas called 'supercookies'

Firefox supercookie mitigations

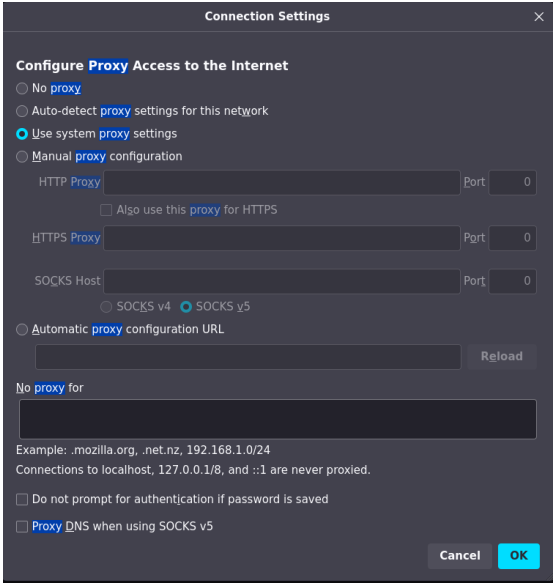
https://blog.mozilla.org/security/2021/01/26/supercookie-protections/

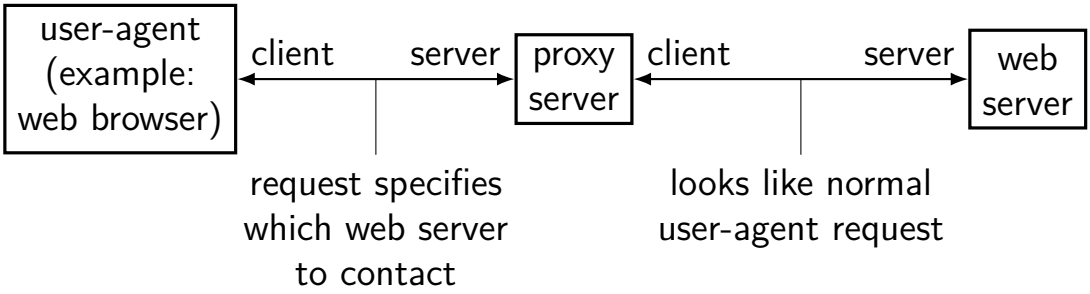
for each top-level website, separate:

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

connections (even for same hostname)

HTTP proxies (1)




user-agent
(example:
web browser)
client
server
proxy
server
client
server
web
server
request specifies
which web server
to contact
looks like normal
user-agent request

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

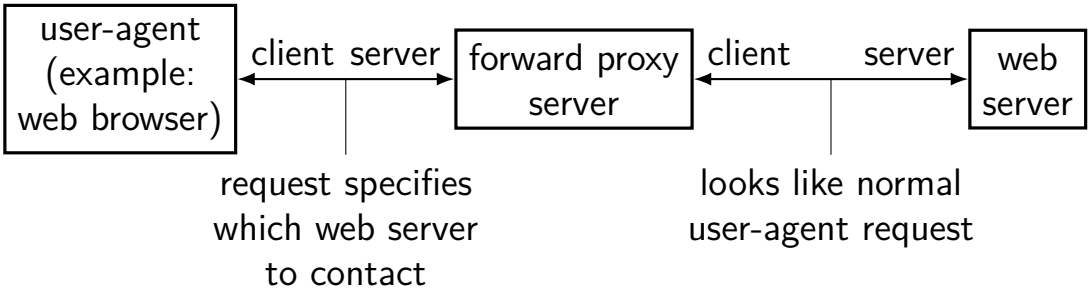
- reason for `Cache-Control: private`

filtering content

- antimalware, adblocking, etc.

logging content (example: debugging webapp)

...


user-agent
(example:
web browser)
client
server
forward proxy
server
client
server
web
server
request specifies
which web server
to contact
looks like normal
user-agent request
user-agent
(example:
web browser)
client
server
reverse proxy
server
client
server
backend
server
looks like normal
user-agent request
selected from
user-agent request path

reverse proxy

why not just go directly to actual web server?

make multiple web severs 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 webservers

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:53
```

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)

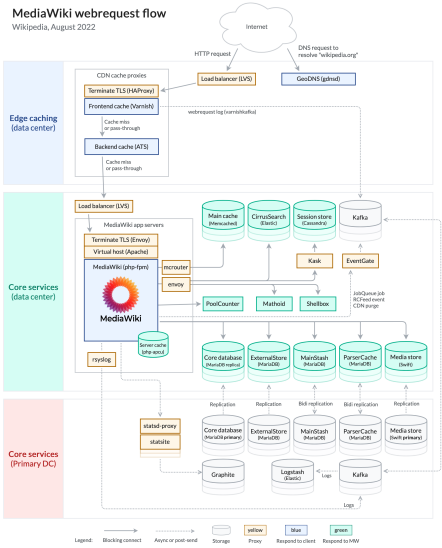
...

CONNECT

allows “tunnelling” arbitrary TCP connections through HTTP

often not implemented by HTTP proxies and/or very restricted

Wikimedia architecture



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
- possibly with script to submit automatically
- data in form tells foo.com about username, etc.
- cryptographically signed or similar

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

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: prevoiusly,.."
    ...
}
```

```
HTTP/1.1 200 OK
...

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

backup slides