Experiences Switching an Archiving Web Crawler to Support HTTP/2

Sebastian Nagel

About HTTP/2

HTTP/2 was introduced in 2015 as the second major version of the Hypertext Transfer Protocol. HTTP/2 addresses several issues with HTTP/1.1 and focuses on performance, specifically low latency when loading a web page in web browsers and efficient use of network and server resources.

The current standard is defined by RFC 9113 [1] which obsoletes earlier versions [2, 3].

HTTP 1.x – Overview and Limitations

In HTTP/0.9 (1991) a single request line `GET /index.html` is sent by the client over TCP and the server sends the file content back.

HTTP/1.0 (1996) adds name-value headers to transfer metadata in requests and responses. Compressed transfer of content is supported.

HTTP/1.1 (1997) introduces several improvements: the reuse of TCP connections (`keep-alive`), virtual hosting (`Host` request header) and chunked `Transfer-Encoding`.

HTTP/1.1 is a simple protocol but has two major limitations:

First, it is a synchronous protocol. The TCP connection is blocked after the request is sent until the response is returned and processed. TCP connections are resource-intensive, esp. if HTTPS is used. Although softened by `keep-alive` connections in HTTP/1.1, still only a single resource can be transferred at time.

Second, it is a text-based protocol and although the bulk of the payload is binary, even text-based formats (HTML, CSS, Javascript) are compressed into a binary format for transfer. "So, the web had basically moved on from text-based transport a long time ago, but HTTP had not." [5]

HTTP/2 – Technical Outline

HTTP/2 continues to use the same building blocks as HTTP/1.1 – methods, status codes, header fields, and URIs. But the transport layer between client and server comes with several improvements:

Messages are wrapped into binary frames which are more efficient to parse because no scan for newline characters is required. Frames of different streams (request-response pairs) can be sent on the same TCP connection without interfering which each other. This enables connection multiplexing, even with stream prioritization, and flow control (transfer rate of a stream).

HTTP headers can reach a size of few kilobytes if HTTP cookies are transferred. The HPACK format [8] defines header compression based on the following principles:

- Client and server need to store and update a table of header fields seen over multiple requests. If the value of a header is the same between request, the header is only referenced.
- Commonly used headers are predefined in a static table.
- Header names and values, not yet seen, are compressed using a static Hufmann coding.

The optional "HTTP/2 Server Push" [9] allows servers to preemptively send resources to the client. In practice, is has appeared to waste network bandwidth because servers cannot know which resources are stored in the client's cache. Resource hints (e.g., `<link rel=preload>`) and "HTTP 103 Early Hints" [10] have found wider adoption as mechanism for resource preemption and major web browsers stopped supporting "Server Push".

TLS Encryption

The HTTP/2 standard defines requests over both encrypted (`https://`) and unencrypted (`http://`) connections. However, almost all HTTP/2 client implementations only support the encrypted variant – abbreviated as "`h2`". The unencrypted variant "`h2c`" is rarely used.

Outlook – HTTP/3

HTTP/3, was published in 2022. HTTP/3 is based on the same principles as HTTP/2, but uses QUIC and UDP as transport layer improving the performance over TCP.

About Common Crawl

Common Crawl is a non-profit organization which regularly crawls a significant sample of the web and makes the data accessible free of charge to everyone interested in running machine-scale analysis on web data.

At present, we crawl every month up to 3.0 billion web pages. The data is hosted in the Amazon cloud as part of the AWS Open Data program.

Contact: `https://commoncrawl.org/` `sebastian@commoncrawl.org`

Make a Web Crawler Speak HTTP/2

Looking at the technical complexity of HTTP/2, there was no doubt that a custom implementation of the protocol is out of scope.

Common Crawl uses a Java crawler, Apache Nutch [11], for crawling and web archiving. Nutch is equipped with three pluggable implementations of the HTTP protocol layer:

- A custom implementation of HTTP/1.1 based on Java sockets
- Using the Apache HTTP Client [12], focused on crawling content behind authentication and forms
- Relying on the Square's OkHttp library [13] supporting Brotli content encoding, providing a configurable connection pool and interceptors to record HTTP headers and IP addresses. OkHttp already speaks HTTP/2, and support for HTTP/2 can be switched on or off per configuration.

The OkHttp protocol plugin was and is used by our crawler because the interceptor architecture makes it the most suitable for web archiving. Adding support for HTTP/2 could be implemented by a configuration change; only the WARC writer required minor modifications.

HTTP/2 Captures in WARC Files

HTTP/2 Requests and Responses in WARC Files

The WARC specification requires that HTTP response records "should contain the full HTTP response received over the network, including headers, i.e. it contains the 'response' message defined by section 6 of HTTP/1.1 ([RFC 2616]), or by any previous or subsequent version of HTTP compatible with section 6 of HTTP/1.1 ([RFC 2616])." ([14], section "6.3.2 'http' and 'https' schemes").

By now, there is no specification how to archive HTTP/2 or HTTP/3 traffic [16]. The only way to store HTTP/2 requests and responses is to rely on a compatible HTTP/1.1 representation, while the original HTTP protocol version is stored as WARC header metadata. The `WARC-Protocol` header is proposed in [17] and was implemented in our web crawler by [19, 20].

Example HTTP/2 WARC Capture

```
WARC/1.0
WARC-Type: request
WARC-Date: 2025-02-17T13:30:42Z
...
WARC-Target-URI: https://en.wikipedia.org/wiki/Saturn
WARC-Protocol: h2
WARC-Protocol: tls/1.3
WARC-Cipher-Suite: TLS_AES_128_GCM_SHA256

GET /wiki/Saturn HTTP/1.1
User-Agent: CCBot/2.0 (https://commoncrawl.org/faq/)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: br,gzip
Host: en.wikipedia.org
Connection: Keep-Alive


WARC/1.0
WARC-Type: response
WARC-Date: 2025-02-17T13:30:42Z
...
WARC-Target-URI: https://en.wikipedia.org/wiki/Saturn
WARC-Protocol: h2
WARC-Protocol: tls/1.3
WARC-Cipher-Suite: TLS_AES_128_GCM_SHA256
...

HTTP/1.1 200
date: Sun, 16 Feb 2025 22:21:15 GMT
...

<!DOCTYPE html>
...
<title>Saturn - Wikipedia</title>
...
```

WARC Reader Backward-Compatibility

HTTP/2 does not define a "reason phrase"; e.g., in HTTP/1.1 it would be "`Ok`" for the status code 200. We decided not to include an artificial reason phrase. However, a white space is added after the status code to ensure that WARC readers can split the status line into three parts at space characters:

```
HTTP/1.1␣200␣\r\n
```

In 2020, HTTP/2 was accidentally enabled [21]. Tests run in the aftermath showed that many WARC readers complain or fail on a HTTP status line with an unknown HTTP version number:

```
GET /index.html HTTP/2
```

The `WARC-Procotol` field was the only option in order not to break downstream user code. As shown in the example, we decided to repeat the `WARC-Procotol` header to encode both the HTTP and TLS version. Of course, this requires that downstream tools reading the WARC files are able to deal with the repeated headers. The WARC-to-WAT converter required a fix [22] which also was necessary for repeated HTTP headers, e.g., `Set-Cookie`. Further specification regarding repeated headers is recommended, cf. [23].

Experiences Crawling With HTTP/2

Support for HTTP/2 was enabled during a running crawl. This allowed us to measure the performance of crawling with or without support for HTTP/2 in an identical setup at almost the same time.

The page fetching in a typical CCF main crawl is split over 100 segments, each segment is fetched in a separate Hadoop Map-Reduce job running for 3 hours. Fetcher job performance was measured over five jobs each before and after HTTP/2 was enabled.

Share of HTTP Versions

Not unexpected, HTTP/2 is the dominant HTTP protocol version, and more than 70% of the web pages were fetched over HTTP/2.

HTTP/2	HTTP Version	Captures	%
disabled	http/1.0	49 759	0.04
	http/1.1	128 788 781	99.96
enabled	http/1.0	40 759	0.03
	http/1.1	35 427 285	27.58
	h2	92 977 534	72.39

A similar adoption rate is reported in the WebAlmanac [24]: 22% – HTTP/1.1, 71% – HTTP/2 and 7% – HTTP/3.

CPU Usage

There is a 2.5% reduction in CPU time spent for the entire fetcher job.

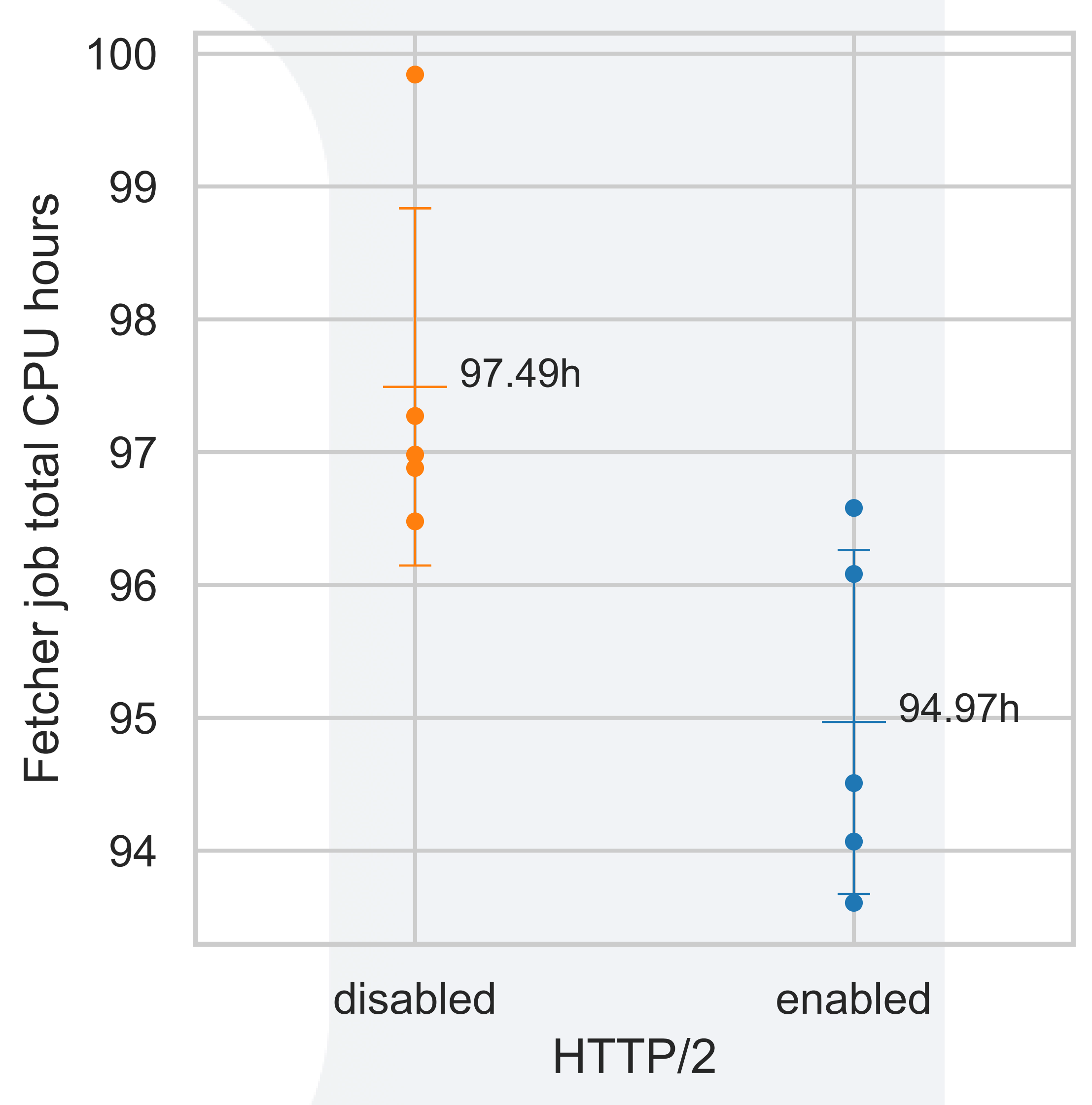

Figure 1. CPU time spent in the five fetcher jobs before and after HTTP/2 was enabled. Error bar: mean and standard deviation.

Profiler snapshots show that by far more time is spent for routines not affected by the switch to HTTP/2, that is TLS handshakes, robots.txt parsing, MIME type identification, WARC digests and compression. Most CPU time is saved when reading or writing HTTP headers.

Page Fetch Time

The average time spent to fetch a web page is slightly reduced with HTTP/2 enabled – from 1038 to 1020 milliseconds. A speed up is also visible for the most common combination of HTTP/2 and TLS 1.3 in comparison to HTTP/1.1 and TLS 1.3.

HTTP/2	HTTP	TLS	Captures	Capt %	Avg Fetchtime ms
disabled	*	*	128 841 301	100.00	**1038.29**
	http/1.0	-	15 166	0.01	1011.45
		tls/1.2	13 122	0.01	1168.32
		tls/1.3	21 471	0.02	1708.62
	http/1.1	-	14 681 800	11.40	900.04
		tls/1.2	20 781 482	16.13	1155.79
		tls/1.3	93 328 260	72.44	1033.71
enabled	*	*	128 445 578	100.00	**1020.31**
	http/1.0	-	15 892	0.01	1464.92
		tls/1.2	12 070	0.01	847.56
		tls/1.3	12 797	0.01	1846.70
	http/1.1	-	14 590 516	11.36	913.69
		tls/1.2	9 425 891	7.34	1186.34
		tls/1.3	11 410 878	8.88	1169.11
	h2	tls/1.2	11 327 309	8.82	1113.72
		tls/1.3	81 650 225	63.57	986.26

References

[1] Martin Thomson and Cory Benfield. HTTP/2. RFC 9113. June 2022. 10.17487/RFC9113. https://www.rfc-editor.org/info/rfc9113.
[2] Mike Belshe, Roberto Peon, and Martin Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540. May 2015. 10.17487/RFC7540. https://www.rfc-editor.org/info/rfc7540.
[3] David Benjamin. Using TLS 1.3 with HTTP/2. RFC 8740. Feb. 2020. 10.17487/RFC8740. https://www.rfc-editor.org/info/rfc8740.
[4] Roy T. Fielding et al. Hypertext Transfer Protocol – HTTP/1.1. RFC 2068. Jan. 1997. 10.17487/RFC2068. https://www.rfc-editor.org/info/rfc2068.
[5] Barry Pollard et al. "HTTP/2". In: The 2019 Web Almanac. HTTP Archive, 2019. Chap. 20. https://almanac.httparchive.org/en/2019/http.
[6] Wikipedia contributors. HTTP — Wikipedia, The Free Encyclopedia. 2024. https://en.wikipedia.org/w/index.php?title=HTTP&oldid=1262579277.
[7] Wikipedia contributors. HTTP/2 — Wikipedia, The Free Encyclopedia. 2025. https://en.wikipedia.org/w/index.php?title=HTTP/2&oldid=1278233300.
[8] Roberto Peon and Herve Ruellan. HPACK: Header Compression for HTTP/2. RFC 7541. May 2015. 10.17487/RFC7541. https://www.rfc-editor.org/info/rfc7541.
[9] Wikipedia contributors. HTTP/2 Server Push — Wikipedia, The Free Encyclopedia. 2024. https://en.wikipedia.org/w/index.php?title=HTTP/2_Server_Push&oldid=1254202868.
[10] Kazuho Oku. An HTTP Status Code for Indicating Hints. RFC 8297. Dec. 2017. 10.17487/RFC8297. https://www.rfc-editor.org/info/rfc8297.
[11] Apache Nutch. https://nutch.apache.org/.
[12] Apache HttpClient. https://hc.apache.org/httpcomponents-client-4.5.x/index.html.
[13] OkHttp. https://square.github.io/okhttp/.
[14] The WARC Format 1.1. https://iipc.github.io/warc-specifications/specifications/warc-format/warc-1.1/.
[15] Henrik Nielsen et al. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616. June 1999. 10.17487/RFC2616. https://www.rfc-editor.org/info/rfc2616.
[16] WARC revision 1.1 (modification): support of HTTP 2.X protocol in WARC format. - Issue #15 - iipc/warc-specifications. https://github.com/iipc/warc-specifications/issues/15.
[17] WARC-Protocol field proposal - Issue #42 - iipc/warc-specifications. https://github.com/iipc/warc-specifications/issues/42.
[18] WARC-Cipher-Suite field proposal - Issue #86 - iipc/warc-specifications. https://github.com/iipc/warc-specifications/issues/86.
[19] NUTCH-3062 protocol-okhttp: optionally record HTTP and SSL/TLS versions. https://issues.apache.org/jira/browse/NUTCH-3062.
[20] WARC writer support HTTP/2 · Issue #29 · commoncrawl/nutch. https://github.com/commoncrawl/nutch/issues/29.
[21] Do not use "http/2" protocol version in HTTP headers in WARC files. https://github.com/commoncrawl/news-crawl/issues/42.
[22] Make MetaData multi-valued to preserve values of repeating WARC and HTTP headers - Pull Request #98 - iipc/webarchive-commons. https://github.com/iipc/webarchive-commons/pull/98.
[23] Multiple extension-fields of the same type on the same record? - Issue #95 - iipc/warc-specifications. https://github.com/iipc/warc-specifications/issues/95.
[24] Robin Marx, Barry Pollard, and Chris Böttger. "HTTP". In: The 2024 Web Almanac. HTTP Archive, 2024. Chap. 18. 10.5281/zenodo.14065825. https://almanac.httparchive.org/en/2024/http.
[25] Joe Viggiano et al. "CDN". In: The 2024 Web Almanac. HTTP Archive, 2024. Chap. 17. 10.5281/zenodo.14065633. https://almanac.httparchive.org/en/2024/cdn.
[26] Vaspol Ruamviboonsuk et al. "HTTP". In: The 2022 Web Almanac. HTTP Archive, 2022. Chap. 23. https://almanac.httparchive.org/en/2022/http.
[27] Dominic Lovell et al. "HTTP". In: The 2021 Web Almanac. HTTP Archive, 2021. Chap. 24. https://almanac.httparchive.org/en/2021/http.
[28] Andrew Galloni et al. "HTTP/2". In: The 2020 Web Almanac. HTTP Archive, 2020. Chap. 22. https://almanac.httparchive.org/en/2020/http.