A Brief Overview on HTTP

Justus Wendroth, Benedikt Jaeger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ge35fup@mytum.de, jaeger@net.in.tum.de

***Abstract*—HTTP has evolved over the last 32 years as new requirements had to be tackled due to the changing use of the web. In this paper, we compare HTTP/1.1, HTTP/2, and HTTP/3 regarding their features and show the limitations they have. Additionally, we compare the versions in terms of latency, packet loss, and usage. We see that higher latency affects HTTP/1.1 worse than HTTP/2. When packet loss increases, the advantages of HTTP/2 decrease and HTTP/1.1 can also be faster. HTTP/3 performs better under high packet loss than HTTP/2. The picture is more mixed for the impacts of latency. HTTP/2 currently appears to be the most widely used version, while the use and adoption of HTTP/3 is increasing.**

***Index Terms*—HTTP/1.1, HTTP/2, HTTP/3, QUIC, latency, packet loss**

1. Introduction

The modern Internet is powered by many technologies: IP addresses for connecting across multiple hops, TCP for reliable data transfer, and TLS for secure data transfer. In this paper, we look at the application protocol HTTP. HTTP is a stateless protocol which was first conceived for sending hypertext (HTML documents). Over the years HTTP evolved and gained new features, including the ability to send data other than HTML. Nowadays, HTTP is not only used for retrieving websites but also to exchange data using REST APIs.

To give a better understanding of HTTP, we briefly look at its history and explore key features introduced with new versions. In Section 2, we also look at the limitations of the three major HTTP versions (HTTP/1.1, HTTP/2, and HTTP/3). For readability, we use h1, h2, and h3 to mean HTTP/1.1, HTTP/2, and HTTP/3 respectively. We then compare the effects of latency and packet loss, and usage for the three major versions in Section 3. In Section 4 we examine related work on comparing HTTP versions. Section 5 concludes this paper.

2. Background

HTTP/0.9, formerly known simply as HTTP, was the first version of the protocol, developed by Tim Berners-Lee at CERN and released in 1990 [1]. HTTP/0.9 only supported GET requests to retrieve resources specified by their path [1]. Due to no HTTP headers being included in this version, only HTML documents could be transmitted [1].

HTTP/1.0 was defined in RFC 1945 in 1996 [2]. This RFC describes the common practices and usages for HTTP at the time as the protocol was extended by different parties and interoperability problems often occurred [1]. One issue with HTTP/1.0 was that a connection could not be reused for multiple requests [1].

HTTP/1.1 (h1) was the first standardized version of HTTP and released in 1997 in RFC 2068 [3]. h1 added new headers, the ability to reuse a connection, pipelining to send multiple request before receiving a response, and other additional functionality.

HTTP/0.9, HTTP/1.0, and HTTP/1.1 could only transfer textual data [2], [3]. HTTP/2 (h2) standardized in 2015 in RFC 7540 is a binary protocol [4]. h2 stems from the Google SPDY[1] project [5]. Additional features include multiplexing of requests (using streams) over a single connection, stream prioritization, and more [4].

HTTP/3 (h3) was standardized in June 2022 [6]. h3 operates on top of QUIC [7], which is a UDP-based transport protocol, instead of TCP. QUIC was first developed at Google [8]. Additional features of h3 and QUIC are the ability for 0-RTT[2] handshakes [7], header compression using QPACK [9], and more.

In the following subsections, we take a closer look at h1 (Section 2.1), h2 (Section 2.2), and h3 (Section 2.3).

2.1. HTTP/1.1

In this section, we look at RFC 2616 from 1999 [10] which was an update of the first h1 version in 1997 [3]. If not otherwise mentioned, we are referring to RFC 2616 [10] in this entire section. One additional feature of HTTP/1.1 is caching, which allows minimizing the number of requests a client has to make to a server as the client can cache previous responses for a certain time. Caching can also be performed by proxy servers, which is another feature of HTTP not discussed further here.

Messages. To transfer data between endpoints (e.g. server and client) HTTP uses HTTP messages. There are two different types of messages. Requests can be used to retrieve data or send data. Responses are the answer to a request. HTTP messages consist of a Request-Line or Status-Line, a list of headers to convey additional information, and the message body with the actual data. Requests have a Request-Line consisting of an HTTP

1. SPDY, QUIC, HPACK, and QPACK are not acronyms but names of projects or standards.
2. RTT = Round-Trip Time

doi: 10.2313/NET-2022-11-1_11

```
GET /en-US/docs/Glossary/Simple_header HTTP/1.1
Host: developer.mozilla.org
Accept: text/html, application/xhtml+xl, application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US, en; q=0.5
Accept-Encoding: gzip, deflate, br
...


200 OK
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Keep-Alive: timeout=5, max=1000
Transfer-Encoding: chunked
...

(content)
```

Figure 1: h1 message exchange. Adapted from [1].

method (e.g. GET, POST, PUT, etc.), a Request-URI to indicate the server and resource to be retrieved or sent, and the HTTP version. A Status-Line is composed of the HTTP version, a Status-Code (e.g. 1xx informational, 2xx success, 3xx redirection, 4xx client error, and 5xx server error) to indicate the success of a request, and a Reason-Phrase, which is a textual description of the Status-Code. Status-Lines are used within responses. Fig. 1 shows an example message exchange with h1.

Headers. Headers in HTTP are used to send additional information with a message. There are headers which are required to be sent in requests or responses. Examples of headers are Content-Length to specify the length of the message body, Content-Type to specify the type of resource transported via the message body, Cache-Control to instruct caching mechanisms how to cache resources, and the Cookie headers to add state management to HTTP [11].

Persistent connections. In contrast to HTTP/1.0, h1 allows persistent connections. This means that multiple requests and responses can be sent over a single TCP connection. By not having to open multiple TCP connections for multiple requests overhead is reduced. h1 also introduces pipelining. Pipelining enables sending multiple requests without having to wait for the responses of earlier requests. Responses to pipelined requests must be sent back in the same order in which they were received. Non-idempotent requests (e.g. using the POST method) cannot be pipelined.

Limitations of HTTP/1.1. h1 has several limitations which led to the development of h2 [4]. Pipelining has the problem of head-of-line blocking (HoLB) [12]. HoLB can occur when the first request is for a large file. Sending this large file will cause subsequent responses to be blocked. Pipelining is not widely used by clients and servers as it is difficult to implement [13]. As a solution, multiple TCP connections are opened to be able to send multiple requests at the same time [12]. This causes additional overhead, especially if HTTPS [14] is used. As a consequence, browsers limited the amount of TCP connections per host [12]. A workaround for this is domain sharding, which places resources on different hosts to be able to evade this limitation [12]. Browsers then limited the total number of TCP connections [12]. Other workarounds for h1 include CSS spriting [12].

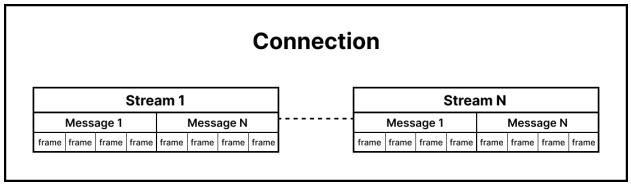

Figure 2: h2 connection. Adapted from [15].

2.2. HTTP/2

h2 builds on top of the core semantics of h1 but introduces multiplexing of requests and responses, which completely changes how data is transferred [4]. In contrast to h1, h2 is a binary protocol, which means that message bodies can be sent in binary format. For most of the new features presented in this section we are referring to RFC 7540 [4]. If this is not the case, we mention this explicitly.

Multiplexing. h2 communication between a client and a server takes place over a single TCP connection (see Fig. 2). A connection is split into multiple streams, where every exchange of request and response is assigned its own stream (see Fig. 2). A request/response exchange fully consumes a stream. Both clients and servers can open new streams. Each stream has an ID, which is assigned by the endpoint initiating the stream and cannot be reused. As can be seen in Fig. 2, Streams are divided into multiple frames. There are different types of frames. For example, the DATA frame carries data, and the HEADER frame is used to open a stream. In every connection there is the stream with ID zero, used for exchanging control messages. In contrast to h1, where most of the information regarding the connection is transferred using headers, h2 conveys a lot of this information using the control stream.

Prioritization. In h2, streams can be dependent on other streams. A dependent always has a lower priority than its parent. When a stream is reprioritized, their dependents move with them. A stream that does not depend on another stream has the stream with ID zero as its parent. If two streams depend on the same stream (can be the zero stream), they can be assigned weights from 1 to 256 and resources should be allocated proportionally to their weight. Assigning priorities is only a suggestion and does not have to be followed by other endpoints.

Server Push. h2 allows a server to push data to a client without a specific request from a client. This can be useful when a client requests a website from the server. Instead of just sending the HTML and waiting for requests of other resources of the website, a server could also push these resources (e.g. CSS and images) directly. Before pushing data to the client, a server needs to send a `PUSH_PROMISE` frame to inform the client about the push.

Flow Control. h2 provides flow control for data frames on the level of individual streams and on the level of the whole connection. Flow control is always specific to connection and cannot be disabled. A receiver can regulate how much data they are able to receive by sending a `WINDOW_UPDATE` frame to the other endpoint.

doi: 10.2313/NET-2022-11-1_11

Header Compression. Header fields (i.e. the name-value pairs in a header) are repetitive and verbose, which causes overhead. Therefore, h2 introduces HPACK [16] for header compression. HPACK uses static and dynamic tables, which are dictionaries that are indexed by indices and contain header fields, to reduce the amount of data to send. The static table is ordered and read-only and provides a list of 61 commonly used headers (e.g. 'accept-encoding: gzip, deflate' has index 16 in the static table) [16, Appendix A]. Dynamic tables are specific to a connection. They are constrained in size and store the header fields encountered during a connection. Additionally, any string can be Huffman encoded using a static Huffman code created for HTTP [16, Appendix B].

Limitations of HTTP/2. h2 has the problem of TCP HoLB, as only one connection is used in h2 [12]. TCP HoLB occurs when one stream has a packet loss, since then all streams have to wait. This is caused by TCP's in-order delivery of packets. h1 does not have this problem, since most of the time multiple TCP connections are used.

Another problem with h2 is the cost of TCP and TLS, since handshakes for TCP and TLS need to be completed to establish a connection [8]. This was also a problem with h1 over TLS, as we have seen previously in Section 2.1. TLS is an optional feature of h2, but most browsers only allow h2 over TLS [12].

2.3. HTTP/3

h3 is the newest version of HTTP [6]. Instead of operating on top of TCP and TLS, h3 operates on top of QUIC [7], which is UDP-based and includes TLS. Using QUIC alleviates the problem of TCP HoLB. h3 includes many of the features introduced with h2. Some of them are implemented directly in h3, while others were moved to QUIC. The semantics of h3 are similar to h1, and h2 [17]. Similar to h2, h3 supports protocol extension. h3 also supports server push [6].

Multiplexing. For this entire subsection, we are referring to [6]. h3 uses QUIC streams for communication, as it has no separate multiplexing mechanism. Similar to h2, a request-response pair consume a stream. To communicate over streams, h3 uses frames similar to h2. There are multiple frame types, including DATA, SETTINGS and HEADERS frames. To exchange control information, h3 uses two separate unidirectional QUIC streams. Two unidirectional streams allow the endpoints to send data as soon as their able to do so after a 0-RTT or 1-RTT connection.

Header compression. In this section, we are citing [9] unless mentioned otherwise. h3 uses QPACK [9] instead of HPACK [16] for header compression. This is because HPACK relies on all frames across all streams being delivered in order. Using HPACK with h3 would therefore cause HoLB, since QUIC does not guarantee in-order delivery across streams but only within a stream. QPACK has similar design goals and concepts as HPACK (e.g. a dynamic table, a static table, and a static Huffman encoding) but uses different mechanisms to achieve this.

QUIC. In this section, we are citing [7] unless mentioned otherwise. To multiplex data QUIC uses streams. Streams are a visible abstraction for application protocols (e.g. h3) operating on to of QUIC. To prioritize streams, QUIC relies on information from the application protocol, as there is no built-in mechanism for exchanging priorities. Streams are split into multiple frames (e.g. STREAM frames to send data or ACK frames to acknowledge packets). For transferring data QUIC uses QUIC packets which consist of multiple frames (from different streams). Packets are sent using UDP datagrams (multiple packets can be in one datagram).

QUIC enables 0-RTT and 1-RTT handshakes by combining the cryptographic (uses TLS 1.3 [18], [19]) and transport handshake. 0-RTT handshakes are possible if there was a prior connection between endpoints. In contrast, h2 over TCP and TLS has a 3-RTT handshake [20].

QUIC connections have ConnectIDs which allow the connection to persist across changes to the underlying IP or port.

Additional features of QUIC include flow control for individual streams and the entire connection, loss detection and recovery mechanisms, and others.

Limitations of HTTP/3. Googles' QUIC implementation consumes twice as much CPU as TCP/TLS [8]. The reasons for the higher CPU usage are the cryptography, the exchange of UDP packets, and maintaining QUIC state. TLS 1.3 enables 0-RTT and 1-RTT cryptographic handshakes [19]. Using TLS 1.3 with h2 reduces the advantages of QUIC over TCP/TLS.

3. Evaluation

In this section, we compare the different HTTP versions regarding their usage and how they react to latency and packet loss.

Effects of latency. Two aspects that have an impact on the latency of the different versions are the handshake they perform (i.e. cryptographic handshake and transport handshake) and their general structure.

h3 performs at most 1-RTT handshakes as the cryptographic and transport handshakes are combined. h1 and h2 over TLS need to perform the TLS handshake and the TCP handshake separately. As mentioned previously, for TLS 1.2 and earlier versions this takes 3-RTT. Therefore, an increase in latency should affect the h3 handshake less than h1 and h2. Latency should affect h2 less than h1 because when h1 is used, often multiple connections are opened and for each connection the handshake has to be repeated. Langley et al. [8] show that the handshake latency increases linearly for h2 over TCP/TLS and remains almost constant for Google QUIC.

Since h2 and h3 support multiplexing of requests and responses, they should handle latency better than h1. References [12], [21] show that h2 reacts better to latency than h1. Trevisan et al. [22] show the order h3, h2, and h1 from best to worst. In contrast, Saif et al. [23] show h3 performing worse than h2 regarding QoE (Quality of Experience) measurements using Lighthouse. This is because in their tests LCP (i.e. Largest Contentful Paint: time for the largest payload to be rendered completely)

performed consistently much worse for h3 than h2. In addition, no 0-RTT handshakes were used for h3.

Effects of packet loss. h2 has the problem of TCP head of line blocking, where a packet loss affects all streams which are multiplexed over a single TCP connection. Therefore, packet loss might affect h2 worse than h1. h3 solves the TCP HoLB by using QUIC over UDP instead of TCP. It should therefore react better to packet loss than h2. De Saxcé et al. [12] show decreasing performance improvements for h2 compared to h1 with higher rates of packet loss. h1 can also perform better for higher packet loss. In contrast to this, Corbel et al. [21] show that h2 performs better than h1 under high packet loss rates. This might be because they used h1 pipelining over a single TCP connection which would also be affected by TCP HoLB. h3 shows better performance than h2 at higher packet loss rates [22], [23].

Usage. In this section, we mostly focus on data provided by Cloudflare, w3tech and the HTTP archive, as data mentioned in papers quickly becomes outdated. For example, Trevisan et al. [22] measured the adoption of h3 at 4.8% in October 2020, but as we show in the following subsection, the adoption of h3 has significantly increased over the last 1.5 years.

For the 30 days prior to June 11, 2022, Cloudflare radar showed that the traffic passing through their infrastructure was 8% h1, 68% h2, and 24% h3 [24]. Additionally, over the 12 months prior to June 2022, h2 made up the majority of requests for Cloudflare customer content and the number was increasing [25]. h1 requests were stable, while h3 requests increased and surpassed h1. W3tech measures the adoption of h3 and h2 by websites by examining the top 10 million websites from Alexa[3]and 1 million from Tranco[3] [26]. When a technology is found on a website, that website adopts the technology. h2 is adopted by 45.8% of websites, which has stayed relatively stable over the 12 months prior to June 2022 [27]. h3 is adopted by 25.2% of websites, which has increased over the 12 months prior to June 2022 [28]. The HTTP archive crawls millions of URLs from the Chrome User Experience Report on a monthly basis [29]. For June 1, 2022, h3 support was at 15% for desktop and 15.3% for mobile and has been steadily increasing since May/June 2020. The number of h2 requests on June 1, 2022, was at 67.7% for desktop and 67.8% for mobile and also steadily increasing.

It is difficult to compare data from different sources. However, a general trend in increasing usage of h2 and increasing adoption and usage of h3 can be seen. h1 is also still widely used. For example, the Facebook bot, Google bot, and LinkedIn bot, which are crawlers for the respective social media sites and search engines, perform a lot of their requests using h1 [25].

4. Related work

De Saxcé et al. compare h1 and h2 using page load time as the performance indicator [12]. They clone 20 popular websites to a server and use a LAN connection to that server to test the impact of latency and packet loss on h1 and h2. They show that an increase in latency impacts page load times for h1 more than h2. In contrast, h2 shows less performance benefits the higher the packet loss is. h2 can also take longer than h1 for higher packet loss. Some limitations include not using TLS and no domain sharding, as all data is on a single server.

Saif et al. compare h2 (over TCP and TLS 1.3) and h3 using the Lighthouse open-source tool, which measures performance and QoE (Quality of Experience) aggregating different metrics to give a performance score [23]. Their test setup is a server in a virtual machine and Google Chrome as the client on the same machine. They use web content to resemble a realistic web page. To compare effects of latency and packet loss, they increase these parameters individually (latency to 1000ms and even 2000ms; packet loss to 1.4%). The baseline score without any adjustments to latency or packet loss was 65 for h3 and 87 for h2. h3 showed consistently worse performance for increases in latency. For packet loss, h3 performed better than h2 for a packet loss starting at 1% and the score stayed relatively flat in contrast to h2. h3 had consistently worse scores for LCP (i.e. Largest Contentful Paint: time for the largest payload to be rendered completely) which makes up a large percentage of the Lighthouse score. One limitation is that no 0-RTT connection establishments were used for h3.

Trevisan et al. look at the adoption and performance of h3 [22]. Using a data set of 5 million URLs, they found that 4.8% of these websites supported h3 in October 2020. Of the websites that support h3, 51% still retrieve one or more objects using h1. For testing performance they make requests to a subset of websites supporting h3. They increase latency to 200ms, increase packet loss to 5% and decrease bandwidth to 1 Mbit/s separately. To compare the effects of varying these parameters, they used onLoad (when all elements of a webpage have been loaded and parsed) and speedIndex (when visible portions of the page are displayed). They compared h1, h2, and h3 regarding latency and found that the mean onLoad and speedIndex times are best for h3, second for h2, and worst for h1. The improvements increase with higher latencies. They also compare h2 and h3 regarding bandwidth and packet loss and found that h3 had better performance at low bandwidth, but performance was similar for high packet loss. They always use a fresh browser profile with empty cache and no pre-existing connections. This means no 0-RTT for h3.

A limitation of all papers presented here is that they only compare two different HTTP versions (h1 and h2 or h2 and h3). Trevisan et al. have measurements across all versions, but they mostly focus on comparing h2 and h3 [22].

5. Conclusion

This paper gave an overview of the different HTTP versions HTTP/1.1 (h1), HTTP/2 (h2), and HTTP/3 (h3) and their most important features. In h1, messages are exchanged via requests and responses and headers can be used to convey additional information. h2 builds on these features from h1 by including mechanisms to multiplex

3. Alexa [30] and Tranco [31] provide top sites rankings

doi: 10.2313/NET-2022-11-1_11

requests and compress headers. h3 builds on the multiplexing from h2 by using QUIC and UDP instead of TCP and therefore alleviating the TCP HoLB problem of h2.

We then compared h1, h2, and h3 in terms of how they perform under different amounts of latency and packet loss. We also compared the usage of the different versions. h2 performs better than h1 at high latency. For higher packet loss, the benefits of h2 are reduced and h1 can also perform better. By comparing h3 and h2, we see that h3 performs better under higher packet loss. For higher latency, h2 performs better for some tests and h3 for others. h2 currently seems to be the most used version (June 2022). The use and adoption for h3 is increasing. h1 usage is relatively stable.

In the future, there is room for a comparison of all three HTTP versions regarding latency, packet loss, bandwidth, and maybe other parameters, since most papers only compare two different HTTP versions (h1 and h2 or h2 and h3).

References

[1] "Evolution of HTTP - HTTP | MDN." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP

[2] H. Nielsen, R. T. Fielding, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.0," Internet Engineering Task Force, Request for Comments RFC 1945, May 1996, num Pages: 60. [Online]. Available: https://datatracker.ietf.org/doc/rfc1945

[3] R. T. Fielding, H. Nielsen, J. Mogul, J. Gettys, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," Internet Engineering Task Force, Request for Comments RFC 2068, Jan. 1997, num Pages: 162. [Online]. Available: https://datatracker.ietf.org/doc/rfc2068

[4] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," Internet Engineering Task Force, Request for Comments RFC 7540, May 2015, num Pages: 96. [Online]. Available: https://datatracker.ietf.org/doc/rfc7540

[5] "SPDY." [Online]. Available: https://www.chromium.org/spdy/

[6] M. Bishop, "Hypertext Transfer Protocol Version 3 (HTTP/3)," Internet Engineering Task Force, Internet Draft draft-ietf-quic-http-34, Feb. 2021. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-quic-http

[7] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," Internet Engineering Task Force, Request for Comments RFC 9000, May 2021, num Pages: 151. [Online]. Available: https://datatracker.ietf.org/doc/rfc9000

[8] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, Aug. 2017, pp. 183–196. [Online]. Available: https://doi.org/10.1145/3098822.3098842

[9] C. B. Krasic, M. Bishop, and A. Frindell, "QPACK: Header Compression for HTTP/3," Internet Engineering Task Force, Internet Draft draft-ietf-quic-qpack-21, Feb. 2021, num Pages: 51. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-quic-qpack

[10] H. Nielsen, J. Mogul, L. M. Masinter, R. T. Fielding, J. Gettys, P. J. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," Internet Engineering Task Force, Request for Comments RFC 2616, Jun. 1999, num Pages: 176. [Online]. Available: https://datatracker.ietf.org/doc/rfc2616

[11] A. Barth, "HTTP State Management Mechanism," Internet Engineering Task Force, Request for Comments RFC 6265, Apr. 2011, num Pages: 37. [Online]. Available: https://datatracker.ietf.org/doc/rfc6265

[12] H. de Saxcé, I. Oprescu, and Y. Chen, "Is HTTP/2 really faster than HTTP/1.1?" in *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Apr. 2015, pp. 293–299.

[13] M. Nottingham, "Making HTTP Pipelining Usable on the Open Web," Internet Engineering Task Force, Internet Draft draft-nottingham-http-pipeline-01, Mar. 2011, num Pages: 10. [Online]. Available: https://datatracker.ietf.org/doc/draft-nottingham-http-pipeline-01

[14] E. Rescorla, "HTTP Over TLS," Internet Engineering Task Force, Request for Comments RFC 2818, May 2000, num Pages: 7. [Online]. Available: https://datatracker.ietf.org/doc/rfc2818

[15] "HTTP/1.1 vs HTTP/2: What's the Difference? | DigitalOcean." [Online]. Available: https://www.digitalocean.com/community/tutorials/http-1-1-vs-http-2-what-s-the-difference

[16] R. Peon and H. Ruellan, "HPACK: Header Compression for HTTP/2," Internet Engineering Task Force, Request for Comments RFC 7541, May 2015. [Online]. Available: https://datatracker.ietf.org/doc/rfc7541

[17] R. T. Fielding, M. Nottingham, and J. Reschke, "HTTP Semantics," Internet Engineering Task Force, Internet Draft draft-ietf-httpbis-semantics-19, Sep. 2021, num Pages: 252. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-httpbis-semantics

[18] M. Thomson and S. Turner, "Using TLS to Secure QUIC," Internet Engineering Task Force, Request for Comments RFC 9001, May 2021, num Pages: 52. [Online]. Available: https://datatracker.ietf.org/doc/rfc9001

[19] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Internet Engineering Task Force, Request for Comments RFC 8446, Aug. 2018, num Pages: 160. [Online]. Available: https://datatracker.ietf.org/doc/rfc8446

[20] "Introducing Zero Round Trip Time Resumption (0-RTT)," Mar. 2017. [Online]. Available: http://blog.cloudflare.com/introducing-0-rtt/

[21] R. Corbel, E. Stephan, and N. Omnes, "HTTP/1.1 pipelining vs HTTP2 in-the-clear: Performance comparison," in *2016 13th International Conference on New Technologies for Distributed Systems (NOTERE)*, Jul. 2016, pp. 1–6, iSSN: 2162-190X.

[22] M. Trevisan, D. Giordano, I. Drago, and A. S. Khatouni, "Measuring HTTP/3: Adoption and Performance," in *2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet)*, Jun. 2021, pp. 1–8.

[23] D. Saif, C.-H. Lung, and A. Matrawy, "An Early Benchmark of Quality of Experience Between HTTP/2 and HTTP/3 using Lighthouse," in *ICC 2021 - IEEE International Conference on Communications*, Jun. 2021, pp. 1–6, iSSN: 1938-1883.

[24] "Cloudflare Radar." [Online]. Available: https://radar.cloudflare.com

[25] "HTTP RFCs have evolved: A Cloudflare view of HTTP usage trends," Jun. 2022. [Online]. Available: http://blog.cloudflare.com/cloudflare-view-http3-usage/

[26] "Web Technologies Statistics and Trends." [Online]. Available: https://w3techs.com/technologies

[27] "Usage Statistics of HTTP/2 for Websites, June 2022." [Online]. Available: https://w3techs.com/technologies/details/ce-http2

[28] "Usage Statistics of HTTP/3 for Websites, June 2022." [Online]. Available: https://w3techs.com/technologies/details/ce-http3

[29] "HTTP Archive: State of the Web." [Online]. Available: https://httparchive.org/reports/state-of-the-web

[30] "End of Service Notice." [Online]. Available: https://www.alexa.com/

[31] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," in *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, ser. NDSS 2019, Feb. 2019.