Take the Red Pill for H3 and See How Deep the Rabbit Hole Goes

Minh Nguyen
Alpen-Adria-Universität Klagenfurt
Klagenfurt, Austria

Christian Timmerer
Alpen-Adria-Universität Klagenfurt
Klagenfurt, Austria

Stefan Pham
Fraunhofer FOKUS
Berlin, Germany

Daniel Silhavy
Fraunhofer FOKUS
Berlin, Germany

Ali C. Begen
Ozyegin University
Istanbul, Turkey

ABSTRACT

With the introduction of HTTP/3 (H3) and QUIC at its core, there is an expectation of significant improvements in Web-based secure object delivery. As HTTP is a central protocol to the current adaptive streaming methods in all major streaming services, an important question is what H3 will bring to the table for such services. To answer this question, we present the new features of H3 and QUIC, and compare them to those of H/1.1/2 and TCP. We also share the latest research findings in this domain.

CCS CONCEPTS

• **Networks** → **Application layer protocols**; • **Information systems** → *Multimedia streaming*.

KEYWORDS

HTTP adaptive streaming, QUIC, CDN, ABR, OTT, DASH, HLS.

ACM Reference Format:
Minh Nguyen, Christian Timmerer, Stefan Pham, Daniel Silhavy, and Ali C. Begen. 2022. Take the Red Pill for H3 and See How Deep the Rabbit Hole Goes. In *Mile-High Video Conference (MHV'22), March 1–3, 2022, Denver, CO, USA*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3510450.3517302

1 INTRODUCTION

1.1 The Birth of the Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) is an application-level protocol that has been in use by the World Wide Web (WWW) for over 30 years. The initial HTTP version (HTTP/0.9) was quite a simple protocol prototyped for raw data transfer from a server to a client. In 1996, an informational RFC (RFC 1945 [1]) came out of the Internet Engineering Task Force (IETF) to document HTTP/1.0 where the protocol was improved by allowing messages to be in the format of MIME-like messages, containing meta information about the data transferred and modifiers on the request/response semantics. This RFC was published with some concerns since HTTP/1.0 was somewhat loose on specifying what was mandatory or optional to implement and it did not sufficiently take into consideration the effects of hierarchical proxies, caching, the need for persistent connections and virtual hosts. Incomplete implementations caused interoperability issues and this has necessitated a protocol version change. Shortly after in early 1997, a standards-track RFC (RFC 2068 [2]) specified HTTP/1.1 with more stringent requirements than HTTP/1.0 to ensure reliable and interoperable implementations. Two years later, RFC 2068 was revised to RFC 2616 [3], which stayed in effect for a long time. In 2014, HTTP/1.1 was revised significantly and upgraded with new features in RFCs 7230 [4], 7231 [5], 7232 [6], 7233 [7], 7234 [8] and 7235 [9]. Today, most Web traffic is still carried over HTTP/1.1.

1.2 HTTP/2 (H2)

The second major version of the HTTP, dubbed as HTTP/2 (H2), was released in 2015 by the IETF [10]. H2 was derived mainly from the ideas inherited from Google's SPDY proposal [11], an experimental protocol first announced in 2009[1] with the objective of reducing page load times. H2 included many new features such as stream multiplexing, server push, stream priority and stream termination. Instead of running multiple TCP connections in parallel for handling multiple simultaneous requests as in the case of HTTP/1.1, H2 used a single TCP connection to carry multiple request-response pairs (via multiple streams). In the context of HTTP Adaptive Streaming (HAS) [12], this new stream multiplexing feature was studied in [13, 14] and found to improve the HAS. However, the reported improvements were not substantial. While all major browsers rushed into adding H2 support, content delivery network (CDN) providers have not implemented H2 on their servers until Apple's low-latency extensions for HLS required them to do so [15]. At the end of the day, H2 added several new features, but it still uses TCP as the underlying reliable transport protocol and suffers from head-of-line (HoL) blocking.

Several studies investigated H2 in the context of HAS. Wei *et al.* evaluated the server push feature in the live streaming scenario with the *k-Push* strategy, where an HTTP GET request was sent to the server to download a fixed number of (k) continuous segments [16]. The proposed method showed that server push was able to reduce the request overhead significantly. However, fixing k could result in rebuffering when the throughput was unfavorable. The work in [17–19] focused on methods that flexibly determined k based on throughput, buffer and segment duration to trade off the request overhead and rebuffering risk. Although server push is not widely

MHV'22, March 1–3, 2022, Denver, CO, USA

ACM ISBN 978-1-4503-9222-8/22/03…$15.00
https://doi.org/10.1145/3510450.3517302

[1] https://www.chromium.org/spdy/spdy-protocol/

used today, the studies mentioned above show its benefits to the performance of HAS.

The stream termination and stream priority features were evaluated in 360-degree video streaming [20, 21] to cope with the high throughput requirement of such applications. The 360-degree video is spatially divided into multiple tiles. The tiles located inside the viewport of the user are downloaded in higher quality with more allocated bandwidth using the stream priority feature, whereas the other tiles are downloaded in lower quality with less bandwidth. This strategy reduces the total downloaded data, resulting in a smaller throughput requirement without significantly impairing the user's quality of experience (QoE) [22].

1.3 HTTP/3 (H3)

The third major version of the HTTP is HTTP/3 (H3) [23] and it is built on the QUIC protocol that was recently published in RFC 9000 [24] after a substantial undertaking by the IETF QUIC working group participants. QUIC was developed by Google [25] to provide a secure and reliable transport protocol over UDP. Like TCP, QUIC is a connection-oriented and stateful protocol that provides interaction between a client and server. QUIC allows an application to establish a connection quickly and use flow-controlled streams multiplexed over that single connection. QUIC also enables network path migration without losing the connection. Google reported that QUIC decreased search latency by 8%, and (YouTube) video rebuffering by 18% on desktop devices [25]. Although QUIC and TCP show similar performance in non-lossy networks, QUIC provides significant improvements when the network experiences packet losses [26]. Over the years, many and somewhat incompatible QUIC implementations[2] with different sets of features came out and were tested for interoperability and benchmarking [27, 28]. Nonetheless, as the RFC is now published, we expect all major codebases to converge on the standard QUIC.

Although H3 inherits most of the features of H2, it provides additional features, too, due to the use of QUIC as opposed to TCP. One such feature is the zero round-trip time (0-RTT) connection setup. While H2 wastes some RTTs for the three-way handshake due to TCP connections, H3 takes advantage of Transport Layer Security (TLS) 1.3 directly integrated into QUIC to save at least one RTT for connection setup. QUIC-based H3 is also free from the HoL blocking issues inherent to TCP. Different requests can be processed simultaneously with stream multiplexing by putting each pair of HTTP requests and responses in an individual stream. A stream is an independent sequence of data delivered between the server and client. The data belonging to streams are interleaved as illustrated in Figure 1.

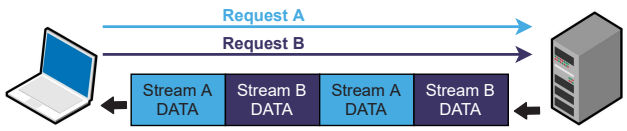
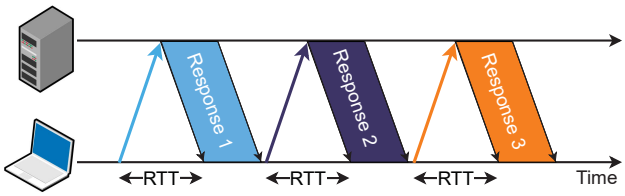
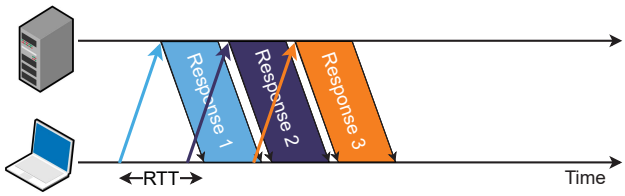


Figure 1: Stream multiplexing feature in H2 and H3. The client sends two requests at the same time.

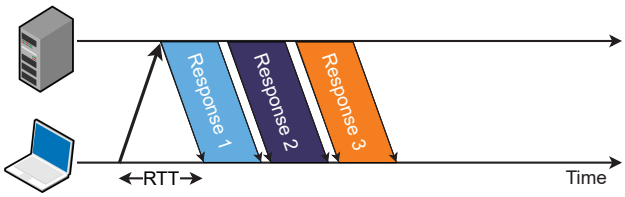
[2]https://github.com/quicwg/base-drafts/wiki/Implementations



(a) Non-pipelined HTTP/1.1.



(b) Pipelined HTTP/1.1.



(c) Server push in H2/H3.

Figure 2: Server push feature in H2 and H3 compared with HTTP/1.1. (a) In non-pipelined HTTP/1.1, the pairs of requests and responses come one by one, which uses one RTT for each pair. (b) When pipelining is enabled for HTTP/1.1, some RTTs can be saved but the number of requests remains the same. (c) H2 and H3 provide server push feature that eliminates multiple RTTs with only one request.

The server push feature enables the client to receive multiple responses from the server by sending a single HTTP GET request. It might be helpful when the server knows that these responses are necessary for the client to completely process the corresponding request. This feature helps the client save multiple RTTs compared to non-pipelined HTTP/1.1, where a new request is sent only if the response of the previous request has been fully received (Figure 2a). HTTP/1.1 offers the pipelining feature that allows many requests to be delivered over a single TCP connection without waiting for each response (Figure 2b). However, it needs more requests than H2 and H3 to obtain the same responses. A comparison of the server push feature in H2 and H3 to HTTP/1.1 (non-)pipelining is shown in Figure 2.

Among the multiple streams over a single connection, the client can indicate its preference for a stream over the others by the stream priority feature. If the stream priority is not indicated, these streams share the connection's throughput equally. This feature exists in both H2 and H3, though H2 in RFC 7450 has a more complex design for the stream priority feature than the mechanism in H3 [29]. The IETF HTTP Working Group is currently working on an extensible prioritization scheme [29], which is being implemented in H3 and might also be adopted by H2 through a revision [30].

The last feature considered in this paper is stream termination. When the client considers a stream's data is no longer necessary, it can send an RST_STREAM frame (in H2) or H3_REQUEST_CANCELLED error code (in H3) to cancel that stream while keeping the connection with the server. This feature is a performant termination technique of H2 and H3 to avoid wasting the throughput for downloading unused data. Meanwhile, the only way for HTTP/1.1 client to cancel a request is to close the TCP connection, which in turn causes other delays due to the three-way handshake and slow start when a new connection is needed.

In this paper, our goals are four-fold:

- **Different flavors of HTTP**: We provide the development path of HTTP and explain the recent new features.
- **Survey of results reported in academia/industry**: We give a glimpse of the results showing the importance of the H2/H3 features in HAS.
- **H2BR**: We describe an H2/H3-aware retransmission technique to fill the quality gaps in the buffer with the objective of enhancing QoE in HAS. H2BR uses H2/H3's features including server push, stream priority, stream multiplexing and stream termination.
- **H3 testbed**: We design an H3-based testbed for testing streaming clients while using predefined network traces. This testbed is a functional tool for automated testing and providing visualized results.

2 SURVEY OF RESULTS REPORTED IN ACADEMIA AND INDUSTRY

H2 and H3 have been studied intensely in both academia and industry. Though the work in [12] covers some studies about H2 and QUIC-based streaming, the efforts from the industry were not mentioned and the results for QUIC's performance in the standardized IETF version were not revealed. This section provides a glimpse of the recent studies in this space.

Mueller *et al.* evaluated H2 in the context of Dynamic Adaptive Streaming over HTTP (DASH) [31]. The authors showed that H2 performed better than its predecessor, HTTP/1.0, and more or less as the same as HTTP/1.1 in terms of bandwidth utilization when the RTT was increased. This was attributed to using one TCP connection for each request in HTTP/1.0 and a single TCP connection for multiple requests in HTTP/1.1 and H2. Timmerer *et al.* continued the work to compare H2 over TCP with QUIC [32]. It was found that QUIC did not improve the DASH performance at the client for any values of RTT.

Similarly, Bhat *et al.* compared the performance of common adaptive bitrate (ABR) algorithms over QUIC and TCP in time-varying network conditions [33]. The authors stated that the ABR algorithms originally designed for running over TCP did not achieve QoE improvements when used over QUIC. On the other hand, Arisu *et al.* found that HAS performance over QUIC was better than TCP, especially in terms of the initial startup delay and the wait time after frame seeking [34]. However, the impact of packet loss rate was not taken into account in these works.

The authors in [35, 36] investigated the performance of H2 over TCP and H3 over QUIC at different packet loss rates. Retransmission techniques to enhance the QoE in HAS were tested in [14, 37]. These techniques took advantage of server push, stream multiplexing, stream priority and stream termination of H2 and H3. The experimental results confirmed the conclusions of [32] when the packet loss rate was close to 0%. Additionally, they showed that H3 provided significantly better performance than H2 when the packet loss rate increased. The reason was that H3 did not suffer from HoL blocking as H2 did, so the throughput was utilized more efficiently. It should be noted that these papers focused on using H2 and H3 independent of the ABR algorithms at the client.

The work in [38] introduced an ABR scheme that made the bitrate decisions for the next segment and the low-quality segments in the buffer by using prominent features of H3 including streaming multiplexing and stream termination. The proposed method, namely Days of Future Past, utilizes a Mixed Integer Linear Programming (MILP) model to decide on which buffered segments should be upgraded at which bitrate to enhance the QoE. Stream multiplexing feature is used to send multiple requests (for the next and to-be-upgraded segments) to the server. If the throughput becomes insufficient, stream termination is called to cancel the download of the buffered segments to avoid rebuffering.

On the industry side, Google and Akamai have been globally deploying QUIC on their servers [39]. According to W3Tech, there are many Websites that enabled H2: around 46% of all Websites are using H2 [3] at the writing of this paper. Though this figure for H3 is more modest, it has been growing drastically. H3 was used by less than 5% of all Websites by the end of 2020, and currently, around 25% of the Websites are using this version of HTTP [4].

In real-world environments, H2 most likely makes the Website faster than HTTP/1.1 though the improvement varies due to *(i)* latency, *(ii)* packet loss, *(iii)* content type and *(iv)* the amount of third-party content [40]. H2 outperforms HTTP/1.1 in high-latency networks (*e.g.*, mobile networks) due to binary framing and header compression (fewer bytes to deliver). On the other hand, HTTP/1.1's performance surpasses H2's in high-packet-loss connections according to Akamai[5]. This is attributed to using multiple concurrent TCP connections by the browsers using HTTP/1.1. In terms of content type, there is a significant improvement of the page load time for the pages containing many small objects when H2 is enabled thanks to the stream multiplexing feature and header compression. This comparison can be conducted by a demo from Akamai[6] as shown in Figure 3. As the third-party content from external domains may not be using H2 features like stream priority, the performance of H2 can be impaired when the number of external content increases. Cloudflare investigated the 0-RTT connection setup of H3 and saw a 12.4% decrease in the delay from the first request to the first byte arrived, compared to H2 [41].

In summary, the later versions of HTTP outperform their predecessors in most cases. H2 significantly reduces the latency and request overhead by the server push feature, and improves the page load time for the Websites containing multiple small objects thanks to the stream multiplexing feature. These features combined with stream priority and stream termination can boost the performance

[3] https://w3techs.com/technologies/details/ce-http2
[4] https://w3techs.com/technologies/details/ce-http3
[5] https://developer.akamai.com/blog/2020/04/14/quick-introduction-http3
[6] https://http2.akamai.com/demo

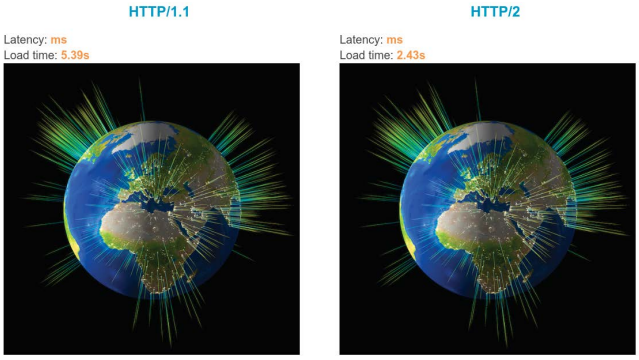


Figure 3: A result obtained in Akamai's demo to compare HTTP/1.1 and H2. The load time of small pieces assembling the Earth graph over H2 (2.43 s) is significantly less than the one over HTTP/1.1 (5.39 s).

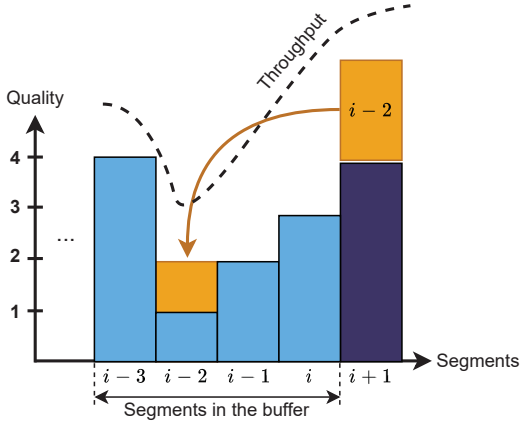


Figure 4: Motivation for the H2BR technique.

of HAS in many scenarios. However, for the networks with a high packet loss rate, HTTP/1.1 might perform better as most Web browsers can open up to six TCP connections over this HTTP version, compared to one TCP connection used by H2, to recover the lost data faster. Moreover, HoL blocking and three-way handshake still occur in H2 because of the TCP connection.

Built on QUIC and UDP, H3 features the 0-RTT connection establishment and eliminates HoL blocking as concurrent streams are delivered independently. Many studies have confirmed that H3 performs better than its predecessors in high-packet-loss connections. However, some state that this new version does not benefit the conventional ABR algorithms in low-packet-loss connections. Therefore, some attempts have been made to use H3's features efficiently in the context of HAS and answer the question: *What will H3 bring to the table for over-the-top (OTT) services?*

From our survey, we observe that among H3's features, stream multiplexing and the ability to deal with HoL blocking can significantly enhance the performance of such services. However, new ABR methods should be developed to take advantage of these features, as most methods were designed with HTTP/1.1 in mind. The next section describes our recently proposed retransmission technique that leverages H2/H3's features to improve user QoE. In addition, the new scheme of stream priority feature [29] in H3 has not been actively investigated in the context of HAS. Thus, utilizing H3's stream priority in streaming applications is an exciting topic for future work.

3 H2BR: H2/H3-AWARE RETRANSMISSION TECHNIQUE

3.1 Motivation

One of the main issues in HAS is the quality variation due to rate adaptation, which is due to the time-varying nature of the available bandwidth. An ABR algorithm at the client is in the charge of selecting a suitable bitrate (or representation) for the media segments. Most ABR schemes consider the available bitrate choices for the upcoming segment(s) but do not replace the segments that have been already downloaded in low quality. If such segments can be replaced by higher quality versions, the user experience can be improved.

Figure 4 shows the need to re-download some low-quality segments located in the buffer. Assume that the video is encoded at four different quality levels. There are four segments in the buffer with the indexes from $i-3$ to i and the quality levels 4, 1, 2, 3, respectively. At the time of downloading the next segment $i+1$, the throughput is good enough to download the next segment with the highest quality level (4) and improve the quality of the segment $i-2$ from 1 to 2. This way, the visual quality can be improved.

3.2 H2BR

We developed an **H**TTP/**2**-**B**ased **R**etransmission technique (H2BR) [14] that takes advantage of features of the newer HTTP versions, including server push, streaming multiplexing, stream priority and stream termination. H2BR decides *(i)* whether a retransmission process should be triggered, *(ii)* which segments should be retransmitted at *(iii)* which bitrates.

After an ABR algorithm decides for the next segment's bitrate, H2BR considers the selected bitrate and the current buffer occupancy to determine whether or not a retransmission should be attempted. To minimize the impact of re-downloading segments on the future bitrate selection, a retransmission is triggered only when the selected bitrate is less than the estimated throughput and the current buffer level is safe (*e.g.*, more than half of the buffer size). To decide the segments in the buffer to be re-downloaded, H2BR reads the quality levels of the buffered segments and detects the quality gaps. We define a quality gap at a segment whose neighboring segments have a higher bitrate. A quality gap should be filled when H2BR can find a suitable bitrate for the segments in that gap, which is as high as possible and between the range of the adjacent segments' bitrates. Also, we need to ensure that the retransmission completes before the re-downloaded segments need to be passed to the decoder.

In addition to the stream for the next segment, H2BR concurrently sends another request to open a new stream for the retransmitted segments using stream multiplexing. In this work, we use server push to re-download the buffered segments by a

single request to reduce the request overhead. Also, the stream priority feature is used to allocate the available throughput among the two streams (*i.e.*, one for the next segment and one for the retransmitted segments). The retransmitted segments are delivered with a higher proportion of the throughput if their deadline is soon. While re-downloading the segments, the instant buffer level and the remaining re-download time are monitored. If these get lower than predefined thresholds, the client uses the stream termination feature to cancel the retransmission stream. These thresholds should be small to prevent frequent terminations but the one related to the instant buffer level should be high enough to avoid the risk of stalling. The results in [14] show that H2BR can significantly reduce the watching time spent at the lowest-quality video (up to more than 70%). Also, the QoE is improved by up to 13% when H2BR is enabled.

H2BR was originally proposed for non-scalable video streaming where a low-quality segment was replaced by a higher-quality one for better QoE. However, it can be applied for scalable video streaming as well. In scalable video streaming, a segment can be upgraded by adding enhancement layers on top of the base layer. In this scenario, H2BR additionally downloads enhancement layers to fill the quality gaps. Our work in [36] validated that H2BR improves the user QoE in scalable video streaming over H2 and H3. Especially when the network has a high packet loss rate, H3 increases H2BR's performance better than H2 due to the ability to eliminate HoL blocking. The results in [36] showed that H3 enabled H2BR to re-download up to 53.8% more enhancement layers.

4 H3 TESTBED

For evaluation, we design a testbed consisting of an H3 server (*e.g.*, NGINX), Web apps (*e.g.*, Chrome) and native apps (*e.g.*, Android). At the time of writing this paper, most projects are in the process of implementing H3 - where possible, we plan to evaluate H3, but fall back to QUIC or H2 if the implementations are not stable enough. Another challenge for the Web platform is to map H3 transport to playback via Media Source Extensions (MSE), which most DASH/HLS clients are using today. WebTransport [42], currently standardized in W3C, could potentially bridge that gap.

Using predefined bandwidth limitation trajectories, different network conditions are emulated to evaluate use cases such as strategies to recover from buffer underruns and optimize low-latency streaming. The testbed enables automated test runs with different streams, streaming clients and bandwidth configurations. The results are tracked and visualized using a streaming analytics solution for evaluation later on. Based on this evaluation, we will present the current state of H3 server and client implementations across Web and native platforms and address compatibility challenges with the existing DASH/HLS-based media delivery.

The concrete setup of the testbed is depicted in Figure 5. The Test Coordinator on the client side queues new jobs in a database. The corresponding worker instances process the jobs based on job-specific configuration parameters such as the target streaming client (*e.g.*, dash.js, ExoPlayer or hls.js). As part of the worker process, the streaming client receives the mandatory manifests/playlists and the corresponding media segments from an H3 server. Internally, the H3 server may use one of the predefined bandwidth limitation trajectories to test the playback under different conditions and scenarios. During the playback, the worker instances report playback metrics such as the average bitrate, startup delay and number of rebuffering events to the metrics server using, for example, Common Media Client Data (CMCD) [43–46] and Server and Network-assisted DASH (SAND) [47]. The reported metrics are analyzed and evaluated in the Evaluation UI. The entire system scales flexibly by adjusting the number of worker processes and docker images.

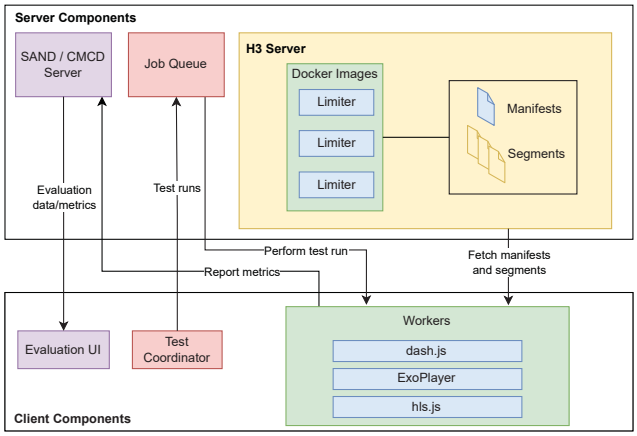


Figure 5: Testbed for automated playback supporting different streaming clients and network conditions.

5 CONCLUSIONS

In this paper, we recap the development path of HTTP and describe new features of the newer HTTP versions: H2 and H3. Moreover, we describe a retransmission technique, H2BR, that takes advantage of features of H2 and H3 to efficiently upgrade the existing low-quality segments in the buffer. H2BR can support ABR algorithms to improve the QoE in both scalable and non-scalable video streaming scenarios. We also lay out a testbed for streaming clients that provides automated testing and result visualization for HAS evaluation over the H3 protocol. Based on the evaluation, the current state of the H3 implementations across Web and native platforms can be presented.

We also reviewed the work published in academia and industry about the performance of H2 and H3 in the context of HAS. It can be concluded that both H2 and H3 bring benefits to HAS when their features are taken into account in the design of ABR algorithms and download strategies. However, the question of *what one can do with H2 and H3 to improve HAS* is still not fully answered.

ACKNOWLEDGMENTS

The financial support of the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development, and the Christian Doppler Research Association, is gratefully acknowledged. Christian Doppler Laboratory ATHENA: https://athena.itec.aau.at/.

REFERENCES

[1] Tim Berners-Lee, Roy Fielding, and Henrik Frystyk. Hypertext Transfer Protocol – HTTP/1.0. [Online] Available: https://datatracker.ietf.org/doc/html/rfc1945, 1996. Accessed on Dec. 27, 2021.

[2] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. [Online] Available: https://datatracker.ietf.org/doc/html/rfc2068, 1997. Accessed on Dec. 27, 2021.

[3] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. [Online] Available: https://datatracker.ietf.org/doc/html/rfc2616, 1999. Accessed on Dec. 27, 2021.

[4] Roy Fielding and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. [Online] Available: https://datatracker.ietf.org/doc/html/rfc7230, 2014. Accessed on Dec. 27, 2021.

[5] Roy Fielding and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. [Online] Available: https://datatracker.ietf.org/doc/html/rfc7231, 2014. Accessed on Dec. 27, 2021.

[6] Roy Fielding and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests. [Online] Available: https://datatracker.ietf.org/doc/html/rfc7232, 2014. Accessed on Dec. 27, 2021.

[7] Roy Fielding, Yves Lafon, and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Range Requests. [Online] Available: https://datatracker.ietf.org/doc/html/rfc7233, 2014. Accessed on Dec. 27, 2021.

[8] Roy Fielding, Mark Nottingham, and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Caching. [Online] Available: https://datatracker.ietf.org/doc/html/rfc7234, 2014. Accessed on Dec. 27, 2021.

[9] Roy Fielding and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Authentication. [Online] Available: https://datatracker.ietf.org/doc/html/rfc7235, 2014. Accessed on Dec. 27, 2021.

[10] Mike Belshe, Roberto Peon, and Martin Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). [Online] Available: https://datatracker.ietf.org/doc/html/rfc7540, 2015. Accessed on Dec. 27, 2021.

[11] SPDY: An experimental protocol for a faster web. [Online] Available: https://dev.chromium.org/spdy/spdy-whitepaper. Accessed on Dec. 27, 2021.

[12] Abdelhak Bentaleb, Bayan Taani, Ali C Begen, Christian Timmerer, and Roger Zimmermann. A survey on bitrate adaptation schemes for streaming media over HTTP. *IEEE Communications Surveys & Tutorials*, 21(1):562–585, 2019 (DOI: 10.1109/COMST.2018.2862938).

[13] Mariem Ben Yahia, Yannick Le Louedec, Gwendal Simon, Loutfi Nuaymi, and Xavier Corbillon. HTTP/2-based frame discarding for low-latency adaptive video streaming. *ACM Trans. Multimedia Computing, Communications, and Applications (TOMM)*, 15(1):1–23, 2019.

[14] Minh Nguyen, Christian Timmerer, and Hermann Hellwagner. H2BR: an HTTP/2-based retransmission technique to improve the QoE of adaptive video streaming. In *ACM Packet Video Wksp.*, 2020.

[15] Kerem Durak, Mehmet N. Akcay, Yigit K. Erinc, Boran Pekel, and Ali C. Begen. Evaluating the performance of Apple's low-latency HLS. In *IEEE MMSP*, 2020 (DOI: 10.1109/MMSP48831.2020.9287117).

[16] Sheng Wei and Viswanathan Swaminathan. Low latency live video streaming over HTTP 2.0. In *ACM NOSSDAV*, 2014.

[17] Hung T Le, Thang Vu, Nam Pham Ngoc, Anh T Pham, and Truong Cong Thang. Seamless mobile video streaming over HTTP/2 with gradual quality transitions. *IEICE Transactions on Communications*, 100(5):901–909, 2017.

[18] Duc V Nguyen, Hung T Le, Pham Ngoc Nam, Anh T Pham, and Truong Cong Thang. Adaptation method for video streaming over HTTP/2. *IEICE communications Express*, 5(3):69–73, 2016.

[19] Jeroen Van Der Hooft, Stefano Petrangeli, Tim Wauters, Rafael Huysegems, Tom Bostoen, and Filip De Turck. An HTTP/2 push-based approach for low-latency live streaming with super-short segments. *Journal of Network and Systems Management*, 26(1):51–78, 2018.

[20] Mengbai Xiao, Chao Zhou, Viswanathan Swaminathan, Yao Liu, and Songqing Chen. BAS-360: Exploring spatial and temporal adaptability in 360-degree videos over HTTP/2. In *IEEE INFOCOM*, 2018.

[21] Dang H Nguyen, Minh Nguyen, Nam Pham Ngoc, and Truong Cong Thang. An adaptive method for low-delay 360 VR video streaming over HTTP/2. In *IEEE ICCE*, 2018.

[22] Mehmet N. Akcay, Burak Kara, Saba Ahsan, Ali C. Begen, Igor Curcio, and Emre Aksu. Head-motion-aware viewport margins for improving user experience in immersive video. In *ACM Multimedia Asia*, 2021 (DOI: 10.1145/3469877.3490573).

[23] Mike Bishop. Hypertext transfer protocol version 3 (HTTP/3). *Internet Engineering Task Force, Internet-Draft draft-ietf-quic-http-34*, 2021.

[24] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. [Online] Available: https://datatracker.ietf.org/doc/html/rfc9000, 2021. Accessed on Dec. 27, 2021.

[25] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. The QUIC transport protocol: Design and internet-scale deployment. In *ACM SIGCOMM*, 2017.

[26] Tanya Shreedhar, Rohit Panda, Sergey Podanev, and Vaibhav Bajpai. Evaluating QUIC performance over web, cloud storage and video workloads. *IEEE Transactions on Network and Service Management*, 2021.

[27] Robin Marx, Joris Herbots, Wim Lamotte, and Peter Quax. Same standards, different decisions: A study of QUIC and HTTP/3 implementation diversity. In *ACM EPIQ Wksp.*, 2020.

[28] Marten Seemann and Jana Iyengar. Automating QUIC Interoperability Testing. In *ACM EPIQ Wksp.*, 2020.

[29] Kazuho Oku and Lucas Pardue. Extensible Prioritization Scheme for HTTP. [Online] Available: https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-priority, 2022. Accessed on Dec. 27, 2021.

[30] Martin Thomson and Cory Benfield. Hypertext Transfer Protocol Version 2 (HTTP/2). [Online] Available: https://datatracker.ietf.org/doc/draft-ietf-httpbis-http2bis/, 2022. Accessed on Dec. 27, 2021.

[31] Christopher Mueller, Stefan Lederer, Christian Timmerer, and Hermann Hellwagner. Dynamic Adaptive Streaming over HTTP/2.0. In *IEEE ICME*, 2013.

[32] Christian Timmerer and Alan Bertoni. Advanced transport options for the dynamic adaptive streaming over HTTP. *arXiv preprint arXiv:1606.00264*, 2016.

[33] Divyashri Bhat, Amr Rizk, and Michael Zink. Not so QUIC: A performance study of DASH over QUIC. In *ACM NOSSDAV*, 2017.

[34] Sevket Arisu and Ali C. Begen. Quickly starting media streams using QUIC. In *ACM Packet Video Wksp.*, 2018.

[35] Divyashri Bhat, Rajvardhan Deshmukh, and Michael Zink. Improving QoE of ABR streaming sessions through QUIC retransmissions. In *ACM Multimedia*, 2018.

[36] Minh Nguyen, Hadi Amirpour, Christian Timmerer, and Hermann Hellwagner. Scalable high efficiency video coding based HTTP adaptive streaming over QUIC. In *ACM EPIQ Wksp.*, 2020.

[37] Cong Wang, Divyashri Bhat, Amr Rizk, and Michael Zink. Design and analysis of QoE-aware quality adaptation for DASH: A spectrum-based approach. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 13(3s):1–24, 2017.

[38] Daniele Lorenzi, Minh Nguyen, Farzad Tashtarian, Simone Milani, Hermann Hellwagner, and Christian Timmerer. Days of future past: an optimization-based adaptive bitrate algorithm over HTTP/3. In *ACM EPIQ Wksp.*, 2021.

[39] Jan Rüth, Ingmar Poese, Christoph Dietzel, and Oliver Hohlfeld. A First Look at QUIC in the Wild. In *Springer International Conference on Passive and Active Network Measurement*, 2018.

[40] Will HTTP/2 make my site faster? [Online] Available: https://www.oreilly.com/content/will-http2-make-my-site-faster/, 2017. Accessed on Dec. 27, 2021.

[41] Comparing HTTP/3 vs. HTTP/2 Performance. [Online] Available: https://blog.cloudflare.com/http-3-vs-http-2/, 2020. Accessed on Dec. 27, 2021.

[42] Aboba, Bernard and Vasiliev, Victor and Yutaka, Hirano. WebTransport. [Online] Available: https://www.w3.org/TR/webtransport/, 2021. Accessed on Dec. 27, 2021.

[43] Consumer Technology Association. CTA-5004: Web Application Video Ecosystem–Common Media Client Data, Sept. 2020.

[44] Ali C. Begen. Manus manum lavat: media clients and servers cooperating with common media client/server data. In *ACM Applied Networking Research Wksp. (ANRW)*, 2021 (DOI: 10.1145/3472305.3472886).

[45] Stefan Pham, Mariana Avelino, Daniel Silhavy, Troung-Sinh An, and Stefan Arbanowski. Standards-based streaming analytics and its visualization. In *ACM MMSys*, 2021.

[46] Ali C. Begen, Abdelhak Bentaleb, Daniel Silhavy, Stefan Pham, Roger Zimmermann, and Will Law. Road to salvation: streaming clients and content delivery networks working together. *IEEE Communications Magazine*, 59(11):123–128, 2021 (DOI: 10.1109/MCOM.121.2100137).

[47] ISO/IEC 23009-5:2017 Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 5: Server and network assisted DASH (SAND). [Online] Available: https://www.iso.org/standard/69079.html. Accessed on Dec. 27, 2021.