Comparison of the Initial Delay for Video Playout Start for Different HTTP-based Transport Protocols

Thomas Zinner, Stefan Geissler, Fabian Helmschrott, Valentin Burger
Institute of Computer Science, University of Würzburg, Am Hubland, 97074 Würzburg, Germany
Email: {zinner|stefan.geissler|fabian.helmschrott}@informatik.uni-wuerzburg.de

***Abstract*—This paper details a measurement study on the impact of different HTTP-based application layer protocols, namely HTTP/1, HTTP/2 and QUIC, on video streaming performance. In this context we evaluate the influence on the initial delay until video playout is started using the live version of the YouTube platform. Furthermore, we evaluate how different network parameters, i.e. bandwidth, RTTs and packet loss influence the different protocols. This work presents an overview over the characteristics of the compared protocols and presents a detailed measurement methodology on how the data has been obtained. Finally, the observed data is evaluated in the context of YouTube video streaming.**

***Keywords*—YouTube Video Streaming; Initial Delay; HTTP/1.1; HTTP/2; QUIC; Measurements**

I. Introduction

The fraction of video content that is being distributed over the Internet is increasing due to the availability of heterogeneous devices and fixed and mobile broadband Internet access. In order to make their delivery platform scalable video providers like YouTube leverage the HTTP infrastructure made of servers, proxies, caches (at the content provider side), and Internet browsers (at the user side). Video streaming is typically realized using HTTP/1.1, a protocol developed and standardized in the 1990s. Due to the steady evolution from mostly text-based simply designed web pages to complex structures composed of many different files spread among several servers, this protocol is seen as main performance bottleneck negatively affecting the Quality-of-Experience (QoE).

To overcome drawbacks like missing encryption and the missing possibility to request a couple of files HTTP/2 and QUIC have been designed. Both protocols aim at reducing the latency of web page data exchange and thus providing a faster page load time. A couple of studies investigate the impact of these protocols on the page loading time and related metrics. However, little is known on the impact of these protocols on video streaming.

This paper provides a first measurement-based comparison of different network protocols with respect to the initial delay until video playout is started. To measure this time we implemented a browser plugin, which performs the data collection based on browser triggered events. This way, we are able to gather all the relevant data required to compare the performance of QUIC and both HTTP web protocols with respect to this metric. For our measurements we rely on the YouTube platform which allows to access video clips using the outlined HTTP protocols. We investigate the impact of different network characteristics with respect to bandwidth, RTTs and packet loss on the initial delay. As opposed to a dedicated lab environment, we evaluate the differences of the mentioned protocols in a realistic scenario using a live production service. This allows for a quantification of the protocol impact in real world scenarios.

The reminder of the paper is structured as follows. Section 2 details the different HTTP-based protocols and summarizes related work. The used evaluation methodology is summarized in Section 3, while the results for the initial delay are presented in Section 4. The paper is concluded in Section 5.

II. Background and Related Work

This section details HTTP-based network protocols and related studies.

A. HTTP-based Network Protocols

HTTP is an application level request/response protocol running on top of TCP [1]. The first formal standardization of HTTP was HTTP/1.1, which was released in January 1997 as RFC 2086 [2]. Common drawbacks of HTTP/1.1 are the sequential request/response exchange and the resulting head of line blocking on application layer, which negatively affects web page loading times. In mid-2009, Google announced their new experimental protocol called SPDY, which should solve some well-known HTTP/1.1 performance issues and therefore improve the web delivery latency. HTTP/2, which is based on SPDY and profits from the lessons learned during the design and operation of this protocol was standardized in May 2015 [3]. It overcomes the outlined head of line blocking on application level by dynamically multiplexing requests on one TCP connection. Besides the standardized protocols, Google designed an experimental protocol, QUIC based on UDP. It provides a couple of new features like faster connection establishment, connection migration, flexible congestion control and a support of cryptographic mechanisms by design.

In order to illustrate the fundamental difference between the protocols, Figure 1 shows their assignment to the network stack. As can be seen, HTTP/x are pure application layer protocols while QUIC is also partly located at transport layer. The main difference, is the particular underlying transport protocol.

Despite the obvious architectural differences, there are also different approaches in supporting a fast data transport, connection establishment, and encryption. HTTP/1.1 introduces keepalive connections, allowing to reuse already existing

978-3-901882-89-0 @2017 IFIP

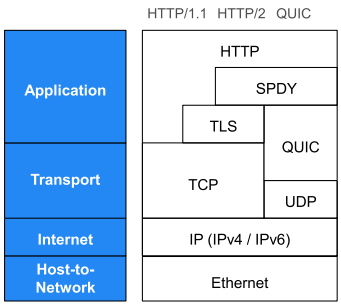


Fig. 1: Protocols Assigned to the Network Stack

connections and hence to benefit from the properties of TCP. To further take advantage of the TCP connection, HTTP/2 introduced its binary framing layer, which enables the possibility of multiplexing several streams over the same TCP tunnel. Even though QUIC is build on top of UDP, it uses stream multiplexing over a single connection just like HTTP/2.

HTTP/1.1 request pipelining was meant to allow multiple requests sent to the server simultaneously, however, it was never supported. Browser developers addressed this by opening up to six different TCP connections per origin. HTTP/2 got rid of the one request at a time constraint and introduced the concept of stream multiplexing over one connection. Hence, the browser can request multiple files simultaneously without the need of additional TCP connections. This multiplexing mechanism was also adopted by QUIC.

The main limiting factor concerning web latency is the RTT, thus it is desirable to reduce the required RTTs as much as possible. With the introduction of keepalive connections as default, HTTP/1.1 was able to get rid of additional RTTs caused by the constant re-establishment of TCP connections after file transfers. In order to further reduce the number of needed RTTs, web developers used tricks like resource inlining, image sprites or domain sharding. With the multiplexing, HTTP/2 did not only improve the keepalive concept, but also got rid of additional TCP connections, which also introduced additional RTTs. Beside the stream multiplexing, QUIC introduces two other mechanisms which help to reduce RTTs: its 0RTT connection establishment and its connection migration. With the 0RTT connection establishment QUIC provides the ability to open an encrypted connection in at most one RTT instead of 3 RTTs when using TCP with TLS. QUIC's connection migration helps to re-establish a connection to an already known server faster, so that the client can request new files immediately without having to wait for the client-server handshake.

B. Performance Comparison of HTTP-based Transport Protocols

In 2013, a first comparison between QUIC and HTTP/1.1 was performed [4]. The evaluations are based on a 10 MB file download for different packet loss, bandwidth, and RTT configurations. During the experiments, HTTP/1.1 performed better than QUIC. This is mainly due to missing optimizations in the early stage of the QUIC protocol at this time, and the missing TLS integration for the HTTP/1.1 experiments.

The author of [5] picked up on the results of [4] in 2014. He repeats the measurements in order to investigate the improvement of QUIC by comparing the results of the different versions. Additionally, the runs are also performed using HTTPS. In case of packet loss, the newer QUIC version clearly prevails against HTTP/1.1 and HTTPS regarding goodput. Looking at the increasing RTT measurements, QUIC is able to improve its performance on low latency links yielding in a result equal to HTTP. However, RTTs higher than 200 ms still have a huge negative influence on QUIC's goodput. Beside these comparisons, the author also evaluates the multiplexing mechanism of QUIC by requesting 10 files of 100 KB each over a single connection. Experiments with the multiplexing mechanisms reveal that the goodput of QUIC continues to grow with available bandwidth, while the goodput of HTTP/1.1 and HTTPS is capped due to the fact that they cannot request more than one file at a time. When experiencing jitter on packet delays, QUIC is highly negatively affected by even small values like 0.5 ms standard deviation to the normal RTT.

In [6], the author compares the performance of QUIC, SPDY, and HTTP/1.1 using web page emulation. The evaluation of the mean page load time of all scenarios shows that SPDY performs almost in any case better than HTTP/1.1. On links with low bandwidths QUIC preveals over HTTP/1.1 due to its multiplexing and compression features. Altogether, it can be seen that QUIC improves as the RTT increases and for RTTs higher than 210 ms QUIC clearly outperforms HTTP/1.1. Nevertheless, HTTP/1.1 performs better than QUIC on low-RTT links with high bandwidth.

The performance of QUIC on transport and on application level is investigated by the authors of [7]. On transport level, QUIC's flow dynamics as well as the friendliness of QUIC's and TCP's congestion control algorithms are evaluated. In order to investigate the flow dynamics, the authors look at the impact of the link capacity, random losses, and enabled FEC on the channel utilization. Beside QUIC with and without FEC, the measurements are also performed using TCP. It turns out that the overhead introduced by FEC worsens the overall performance of QUIC. Further, for 1 % as well as for 2 % induced losses, the TCP goodput is significantly reduced, while the QUIC link utilization is not. A performance evaluation on application level is done by measuring the page load time of websites using QUIC, SPDY over TLS, and HTTPS. As metric the percentage page load time improvement with respect to HTTPS is chosen. For the scenarios without losses, both SPDY and QUIC outperform HTTPS.

The authors of [8] examine the effects of HTTP/1.1, SPDY and QUIC on page load times. For this, they deploy four simple websites on Google Sites, a web hosting service by Google. The authors conclude that page load time is decreased significantly in more than 40 % of the scenarios when using QUIC. Especially on links with high RTT, QUIC seems to help the most, while it does not perform well when downloading a large amount of data due to its packet pacing. Nevertheless, HTTP/1.1 performs best for downloading large objects.

Altogether it can be said, that several aspects regarding the performance of QUIC have been evaluated including several metrics on transport and application level. This paper enhances the state of the art by investigating the network transport delay for real world deployments of these protocols using

Event	Description
`abort`	Loading of an audio/video is aborted
`can play`	Browser can start playing the audio/video
`error`	Error occurred during loading of audio/video
`ended`	Current playlist is ended
`loadedmetadata`	Browser has loaded meta data for audio/video
`loadstart`	Browser starts looking for audio/video
`progress`	Browser is downloading audio/video
`pause`	Audio/video has been paused
`playing`	Video is playing after being paused or stopped for buffering
`stalled`	Browser tries to get media data, but data is not available
`suspend`	Browser is intentionally not getting media data
`waiting`	Video stops because it needs to buffer the next frame

TABLE I: HTML5 Video Events

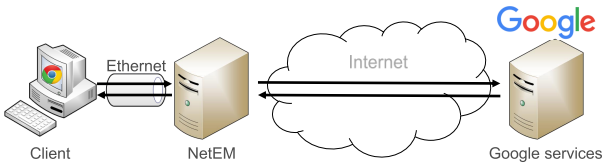


Fig. 2: Measurement Setup

Parameter	Used Values
Bandwidth (Mbps)	{2, 6, 16}
Additional RTT (ms)	{20, 100, 200}
Random Packet Loss (%)	{0, 1, 2}
0RTT CE of QUIC	{with, without}

TABLE II: Evaluation Parameters

Google web services. Further, our evaluations include a close look on the impact of QUICS 0RTT connection establishment mechanism.

III. Evaluation Methodology

This section highlights the implemented tools, the utilized metrics and the measurement setup and the corresponding course of events.

A. Browser Plugin

To gather the relevant data on application layer we develop a plugin for the Google Chrome browser. Since the browser itself already provides extensive network logging and debugging capabilities, the plugin can simply hook into the logging process by listening to event calls provided by the browser engine.

The plugin uses the network analysis tool provided by Google Chrome *DevTools*[1] in order to measure the video streaming behavior and webpage loading times. Furthermore, the DevTools allow the export of a JSON log containing all requests and corresponding responses.

In order to access information on the video playout, a reference to the HTML5 video element is set and listeners for its events are bound. The used events and their descriptions are listed in Table I. Each time one of these events gets fired, the relevant data is logged. Instead of using the `stalled` event, which does not correlate well with video stops, the extension checks periodically every 300 ms if the playout is currently interrupted due to stalling. All relevant data is logged using *net-internals* and *DevTools*.

B. Evaluation Metrics

The Time to Playout Start (TTPS) specifies the amount of time the player needs to start the video playout. It is calculated as the difference between the point in time the player triggers the `playing` event and the `navigationStart` timestamp. In addition we calculate the *Speedup*

$$S = (\frac{E[m(p_1)]}{E[m(p_2)]} - 1) \times 100.$$

Thereby m represents the metric and p_1 represents the baseline protocol p_2 is compared to. The speedup allows the comparison of the performance of different protocols.

C. Measurement Setup and Course of Events

The measurements performed in the context of this work have been done in a dedicated testbed comprised of two physical machines that are directly connected to the Internet. The measurement setup is depicted in Figure 2.

The middle machine, labeled *NetEM*, is running the Network Emulator Kernel module and shapes passing network traffic using the `tc` tool. This machine is responsible for introducing additional delay, packet loss and limiting the available bandwidth according to the respective measurement scenario. The evaluated influence factors and the used values for each of the parameters are listed in Table II.

The *Client* machine on the left is running the Google Chrome browser with our plugin. It generates the website requests, monitors responses and extracts and stores relevant metric data.

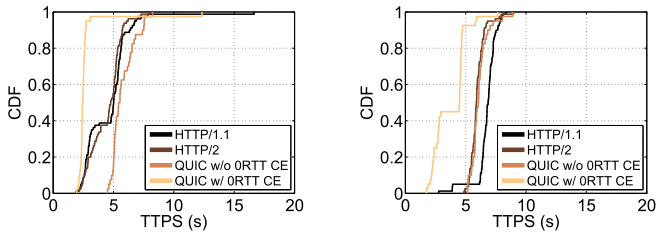
The measurement workflow itself is comprised of two essential steps. First, the respective network parameters are configured using network emulation via `tc`. This enables the simulation of different network characteristics which influence the performance of the evaluated network protocols.

Second, the client machine requests a video from Google's YouTube video streaming platform and monitors the respective metric values mentioned in the previous section. In order to eliminate service load differences as well as network parameters we can not control directly, each parameter and protocol combination is measured 20 times, while the video remains the same for all repetitions. Important to note is that the measurements for each protocol are done in an alternating manner. After two consecutive video requests the used protocol is changed. This results in a spread of the measurement points in time. Thus, the impact of varying network and web service behavior, which cannot be controlled or quantified since we are querying a live system via the Internet, is distributed over the different protocols thus allowing a fair comparison.

[1] https://developers.google.com/web/tools/chrome-devtools/

	2 Mbps		6 Mbps	
	0 % Loss	2 % Loss	0 % Loss	2 % Loss
50 ms RTT				
HTTP/2	3.85 %	−0.86 %	−0.06 %	23.44 %
QUIC w/o 0RTT CE	−19.81 %	−22.75 %	2.57 %	21.85 %
QUIC w/ 0RTT CE	71.72 %	69.14 %	63.42 %	120.20 %
100 ms RTT				
HTTP/2	13.02 %	24.71 %	45.28 %	47.55 %
QUIC w/o 0RTT CE	10.21 %	5.63 %	60.88 %	60.14 %
QUIC w/ 0RTT CE	80.04 %	101.15 %	128.41 %	194.43 %

TABLE III: TTPS Speedup with Regard to HTTP/1.1 Different Network Configurations



(a) 50 ms RTT (b) 100 ms RTT

Fig. 3: Comparison of Time to Playout Start for 2 Mbps Bandwidth

IV. Evaluation of Initial Delay For YouTube Video Streaming

In this section, the results for the impact of different network protocols on user-centric quality indicators for video streaming are discussed.

Even though quality changes and stalling events were monitored throughout the measurements, none of the used parameter combinations resulted in a significant occurrence of any of these events. A possible reason for this could be the short duration of the used video which only amounts 138 seconds. In the following, we highlight the results of our measurements for the initial delay until the video playout starts.

Figure 3 shows the empirical CDFs for a 2 Mbps bandwidth with 50 ms RTT and 100 ms RTT. On the x-axis, the TTPS can be seen, while the values of the corresponding CDF are depicted on the y-axis. For both scenarios, the particular plot illustrates the CDF for each protocol. Considering a RTT of 50 ms, the resulting TTPS values can be viewed in Figure 3a. In this case, QUIC with 0RTT CE clearly performs best. In 97.5 % of the cases the playout start appears by far earlier than the ones of the other protocols.Besides, it can be said that QUIC without 0RTT CE performs the worst. The results for HTTP/1.1 and HTTP/2 on the other hand, behave mostly similar. When looking at the results for the 100 ms RTT scenario, which is depicted in Figure 3b, the performance of QUIC with 0RTT CE clearly stands out: all of its requested videos start the playout earlier than the others. In 5 % of the cases, the video playout starts always earlier when using HTTP/1.1 than when using HTTP/2 and QUIC without 0RTT CE. However, for the remaining cases, the inital delay is typically larger. The behavior of HTTP/2 and QUIC without the 0RTT mechanism in this scenario is similar.

In order to confirm these trends for the other scenarios, Table III lists the speedups with regard to HTTP/1.1 for the 2 Mbps and the 6 Mbps scenarios, each with 50 ms RTT and 100 ms RTT, as well as 0 % and 2 % randomly induced losses. Throughout the measurements, QUIC with 0RTT CE always performs best considering the TTPS metric. Another trend that can be observed is, that with increasing RTT the performance gain with regard to HTTP/1.1 is also improving. Further, the speedup of QUIC using its 0RTT mechanism seems to also improve with increasing loss probability, with exception of the 2 % loss scenario for 2 Mbps bandwidth and 50 ms RTT.

V. Conclusion

This paper features a measurement study on the impact of different HTTP-based network protocols on the initial delay until video play out is started.

Even though the amount of parameter combinations and performed measurements were limited, the obtained results show some clear trends. For the time to playout start, QUIC with 0RTT CE clearly performs better than the other protocols within the context of the investigated scenarios. Moreover, the improvement with regard to HTTP/1.1 gets better with increasing RTT. A reason for this might be its 0RTT CE. Altogether, it can be said that using QUIC with 0RTT CE within the context of streaming videos on YouTube seems to provide several advantages compared to HTTP/1.1 and HTTP/2. However, the complete magnitude of it cannot be determined by the measurements performed and is therefore left open as starting point for future research. The impact of the measurement results on the perceived QoE of the user also remains for future work.

References

[1] I. Grigorik, *High Performance Browser Networking: What every web developer should know about networking and web performance*. "O'Reilly Media, Inc.", 2013.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Rfc 2068: Hypertext transfer protocol-HTTP1.1," *Status: PROPOSED STANDARD*, January 1997.

[3] M. Belshe, M. Thomson, and R. Peon, "Rfc 7540: Hypertext transfer protocol version 2 (HTTP/2)," 2015.

[4] Alex Gizis (Connectify), "Taking Google's QUIC For a Test Drive," 07. November 2013. [Online]. Available: http://www.connectify.me/blog/taking-google-quic-for-a-test-drive/

[5] A. Vernersson, "Analysis of udp-based reliable transport using network emulation," Master's thesis, Lulea University of Technology, 2015.

[6] S. R. Das, "Evaluation of quic on web page performance," Ph.D. dissertation, Massachusetts Institute of Technology, 2014.

[7] G. Carlucci, L. De Cicco, and S. Mascolo, "Http over udp: an experimental investigation of quic," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 609–614.

[8] P. Megyesi, Z. Krämer, and S. Molnár, "How quick is quic?" in *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.