A Performance Perspective on Web Optimized Protocol Stacks: TCP+TLS+HTTP/2 vs. QUIC

Konrad Wolsing, Jan Rüth, Klaus Wehrle, Oliver Hohlfeld*
RWTH Aachen University, Germany
{wolsing,rueth,wehrle,hohlfeld}@comsys.rwth-aachen.de

ABSTRACT

Existing performance comparisons of QUIC and TCP compared an optimized QUIC to an unoptimized TCP stack. By neglecting available TCP improvements inherently included in QUIC, comparisons do not shed light on the performance of current web stacks. In this paper, we can show that tuning TCP parameters is not negligible and directly yields significant improvements. Nevertheless, QUIC still outperforms even our tuned variant of TCP. This performance advantage is mostly caused by QUIC's reduced RTT design during connection establishment, and, in case of lossy networks due to its ability to circumvent head-of-line blocking.

CCS CONCEPTS

• **Networks → Network measurement**;

ACM Reference Format:
Konrad Wolsing, Jan Rüth, Klaus Wehrle, and Oliver Hohlfeld. 2019. A Performance Perspective on Web Optimized Protocol Stacks: TCP+TLS+HTTP/2 vs. QUIC. In *ANRW '19: Applied Networking Research Workshop (ANRW '19), July 22, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3340301.3341123

1 INTRODUCTION

The advancement of Web application and services resulted in an ongoing evolution of the Web protocol stack. Driving reasons are security and privacy or the realization of latency-sensitive Web services. Today, the typical Web stack involves using HTTP/2 over TLS over TCP, making it practically one (ossified) protocol. While parts of the protocols have been designed to account for the others, this protocol stacking still suffers from inefficiencies, e.g., head-of-line blocking. Even though protocol extensions promise higher efficiency (e.g., TLS 1.3 early-data [24] or TCP Fast Open [23]), the ossification around the initial designs challenges their deployment.

QUIC [15] (as used in HTTP/3) combines the concepts of TCP, TLS, HTTP/2, tightly coupled into a new protocol that enables to utilize cross-layer information and to evolve without ossification. While it fixes some of TCP's shortcomings like head-of-line blocking when used with HTTP, its design, in the first place, should enable evolution.

A number of studies showed that QUIC outperforms the *classical* TCP-based stack [2, 7, 8, 13, 17, 30]—that is by comparing QUIC to an unoptimized TCP-based stack; a limitation that we address in this paper. Current QUIC implementations were specifically designed and parameterized for the Web. In contrast, stock TCP implementations, as in the Linux kernel, are not specialized and are built to perform well on a large set of devices, networks, and workloads. However, we have shown [26] that large content providers fine-tune their TCP stacks (e.g., by tuning the initial window size) to improve content delivery. All studies known to us neglect this fact and indeed compare an out-of-the-box TCP with a highly-tuned QUIC Web stack and show that the optimized version is superior. Furthermore, they often utilize simple Web performance metrics like page load time (PLT) to reason about the page loading speed, even though it is long known that PLT does not correlate to user-perceived speeds [3, 14, 31].

In this paper, we seek to close this gap by parameterizing TCP similar to QUIC to enable a fair comparison. This includes increasing the initial congestion window, enabling pacing, setting no slow start after idle, and tuning the kernel buffers to match QUIC's defaults. We further enable BBR instead of the CUBIC as the congestion control algorithm in one scenario. We show that this previously neglected tuning of TCP impacts its performance. We find that for broadband access, QUIC's RTT-optimized connection establishment indeed increases the loading speed, but otherwise compares to TCP. If optimizations such as TLS 1.3 early-data or TCP Fast Open were deployed, QUIC and TCP would compare well. In lossy networks, QUIC clearly outperforms the current Web stack, which we mainly attribute to its ability to progress

*Is now at Brandenburg University of Technology

ANRW '19, July 22, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6848-3/19/07…$15.00
https://doi.org/10.1145/3340301.3341123

streams independently of head-of-line blocking. Our comparison is based on visual Web performance metrics that better correlate to human perception than traditionally used loading times. To evaluate real-world websites, we extend the Mahimahi framework to utilize the Google QUIC Web stack to perform reproducible comparisons between TCP and QUIC on a large scale of settings. This work does not raise any ethical issues and makes the following contributions:

- We provide the first study that performs an eye-level comparison of TCP+TLS+HTTP/2 and QUIC.
- Our study highlights that QUIC can indeed outperform TCP in a variety of settings but so does a tuned TCP.
- Tuning TCP closes the gap to QUIC and shows that TCP is still very competitive to QUIC.
- Our study further highlights the immense impact of choice of congestion control, especially in lossy environments.
- We add QUIC support to Mahimahi to enable reproducible QUIC research. It replays real-world websites in a testbed subject to different protocols and network settings.

Structure. Section 2 examines related work, highlights the evaluation metrics and introduces to the Mahimahi framework. Section 3 explains our testbed, network configuration, and protocol considerations. Section 4 shows the results of the measurement. Finally, Section 5 concludes this paper.

2 RELATED WORK AND BACKGROUND

QUIC is subject to a body of studies [2, 7, 8, 13, 17, 20, 30], most compare QUIC against some combination of TCP+TLS+HTTP/1.1 or HTTP/2. But to the best of our knowledge, all use stock TCP configurations measuring a likely unoptimized TCP version to a QUIC version that inherently contains available TCP optimizations. Yu et al. [30] is the only study on the impact of packet pacing for QUIC as a tuning option. However, no further comparison to TCP is made.

Generally, the related work can be divided into two categories depending on their measurement approach. One body of research [8, 17, 27] measures against websites hosted on public servers utilizing both QUIC and TCP—however, usually operated by Google. Thus, they do not have any access to the servers, which makes tuning the protocol impossible and the configurations in use are unknown. The second body [2, 7, 13, 20] uses self-hosted servers, in principle allowing for tuning, however, none of them does so.

One critical difference between TCP and QUIC is their connection establishment since QUIC by design needs fewer RTTs than traditional TCP+TLS until actual website payload can be exchanged. Cook et al. [8] already take into account that there is a difference between first and repeated connections that require each one less RTT for both protocols. Nevertheless, QUIC still has a one RTT advantage in both connections, repeated as well as first, and again this fact is not dealt with any further.

Since today's websites consist of various resources hosted by several providers, many connections to different servers are established even for fetching a single website. Many studies consider websites with varying resources but deployed by a single server only [2, 7, 17]. To study realistic Web sites, the Mahimahi framework [21] was designed to replicate this multi-server nature of current websites into a testbed (see Section 2.2). Nepomuceno et al. [20] perform a study with Mahimahi but find that QUIC is outperformed by TCP which does not coincide with our and related work. We believe this is due to the use of the Caddy QUIC server, which is known to not (yet) perform very well [19]. Also, they did not configure any bandwidth limitations.

2.1 Web Performance Metrics

We aim to evaluate the performance of a different protocol stack on a broad set of standard Web performance metrics. Besides network characteristics like goodput or link utilization as measured in [7, 30], *Page Load Time* (PLT) is the most used metric. But PLT does not always match user-perceived performance [3, 14, 31], e.g., it includes the loading performance of *below*-the-fold content that is not displayed and thus not reflected in end-user perception. This is why we decide to focus more closely on state-of-the-art visual metrics that are known to better correlate with human perception. These metrics are derived from video recordings of the pages loading process *above*-the-fold as recommended by [5, 9].

Metrics of interest are the time of the *First Visual Change* (FVC), *Last Visual Change* (LVC), and time the website reaches visual completeness of a desired threshold in percent. In our case, *Visual Complete 85* (VC85), which corresponds to the point in time measured from the navigation start when the currently rendered website's above-the-fold matches to 85% the final website picture. Only navigation start can be used as start point since visual metrics are derived from video recordings only (see Section 3.2 how we deal with DNS impacting the measurement). Lastly, we also take into account the *Speed Index* (SI) [11].

2.2 Website Replay with Mahimahi

Mahimahi [21] is a framework designed to replicate real-world websites with their multi-server structure in a testbed. It uses HTTP traffic recordings that are later replayed. Mahimahi preserves the multi-server nature with the help of virtualized Web servers. Mahimahi is built upon multiple shell commands that can be stacked to create a virtual network. Each shell allows for modifying a single aspect of the traversing network flow, e.g., generating loss or limiting the bandwidth. Mahimahi yields realistic conditions for performance measurements [21]. This way, it enables repeatable and controllable studies with real-world websites.

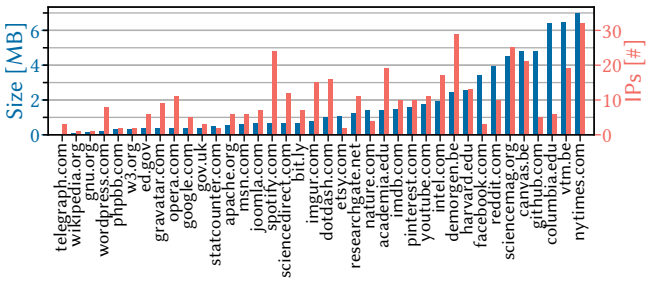

Figure 1: This figure depicts the download size of the replayed websites (blue) and the number of unique IP addresses that need to be contacted for resources (red).

3 TESTBED SETUP

We will now continue to explain how we design our testbed to perform eye-level comparisons of TCP and QUIC.

3.1 Selecting and Recording Websites

Websites. We want to choose websites that replicate a real-world picture of commonly used websites. The goal is to obtain a small set of domains diverse in size, resources, and involved servers. As there is no standard test set of such website, we use the domain collection from [29] consisting of 40 different websites from which we had to exclude two. One domain is a private project website and the other failed to record and reply properly. The domains originate from the Alexa [1] and Moz [18] ranking lists and were chosen in a way to obtain a good distribution of page size and resource counts [29], see Figure 1. The bars in red suggest the majority of our tested sites to use multi-server infrastructures, highlighting the relevance of replicating it with Mahimahi.
Recording. Downloading of the websites was not performed with the tool provided by Mahimahi. Instead, we utilize Mitmproxy with a custom script that dumps the raw HTTP responses of the server to disk. According to the Google QUIC server specification [10] the *transfer-encoding* header must be removed if its value is *chuncked*. The same holds for the *alternate-protocol* header for any value. Other than that, the recorded HTTP responses remain unchanged.

In post-processing, few resource files needed to be downloaded additionally, since, e.g., the header of the github.com website loads a random image from a fixed collection via JavaScript.

3.2 Replaying with Mahimahi

Mahimahi. To support a state-of-the-art QUIC in Mahimahi, we include Google's QUIC server from the Chromium sources utilizing QUIC Version 43. For TLS1.3 and HTTP/2, we replace the Mahimahi default Apache server with NGINX. All NGINX servers forward the requests to a single uWSGI proxy server that provides the previously recorded HTTP responses from main memory. Similarly, the QUIC servers use their built-in feature loading all responses from a folder into memory. Finally, we create a self-signed certificate authority (CA) and incorporate it to the Chrome browser's list of trusted CAs to circumvent any authentication errors.

Protocol	Description
TCP	Stock TCP (Linux): IW10, Cubic
TCP+	IW 32, Pacing, Cubic, tuned buffers, no slow start after idle
TCP+BBR	TCP+, but with BBR as congestion control
QUIC	Stock Google QUIC: IW 32, Pacing, Cubic
QUIC+BBR	QUIC, but with BBR as congestion control

Table 1: Protocol configuration used in our tests.

Enforcing QUIC or TCP+TLS1.3+H2. We want to be sure that only QUIC or TCP is used. On the one hand, we accomplish this using Chrome flags, to enforce QUIC, we set "–enable-quic –origin-to-force-quic-on=*" and "–disable-quic" for TCP respectively. On the other hand, the QUIC and NGINX servers never run at the same time. In the TCP case, each request is performed over TLS1.3 and HTTP/2. There are no resources that get transmitted unencrypted.
Protocol Tuning. To allow for a fair comparison between TCP and QUIC, we tune the stock TCP stack of a Linux kernel to more closely match QUIC's defaults. This is done by increasing the initial window to 32 segments, enabling pacing, setting no slow start after idle and tuning the kernel buffers. QUIC by default also uses an initial window of 32 and pacing. Since we expect the employed congestion control algorithm to significantly impact the measured performance, we incorporated one scenario for TCP and QUIC each utilizing BBR [6] instead of CUBIC [12]. An overview of the five protocol configurations is shown in Table 1.

TCP Fast Open [23] and TLS1.3 early-data [24] are two possible options to tune TCP/TLS even further. We decided against both techniques because of the following reasons. TLS1.3 early-data was not supported by the Chrome browser at the time of the measurement and as it is prone to replay attacks requires idempotency which further challenges its widespread use. TCP Fast Open is not widely deployed on the Internet today [16, 22]. Moreover, we always measure the website performance with a fresh browser and clean caches, thus QUIC has to perform an extra RTT for connection establishment as well and does not use 0-RTT connections.
Network Settings. For network emulation, the built-in tools from Mahimahi are used. We stack the following three network parameters from server to client with Mahimahi shells. First, a packet gets delayed in either direction, both adding up to the desired minimum latency. Second, the link shell implements a drop-tail buffer limiting the throughput per direction. Finally, the loss shell drops packets at random for both directions equally. The loss is configured, such that

Network	Uplink	Downlink	Delay	Loss
DSL	5 Mbps	25 Mbps	24 ms	0.0 %
LTE	2.8 Mbps	10.5 Mbps	74 ms	0.0 %
DA2GC	.468 Mbps	.468 Mbps	262 ms	3.3 %
MSS	1.89 Mbps	1.89 Mbps	760 ms	6.0 %

Table 2: Network configurations. Queue size is set to 200 ms except for DSL with 12 ms.

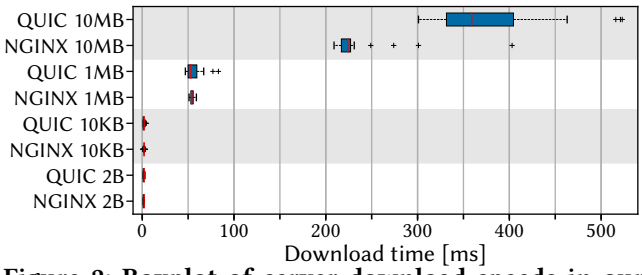

Figure 2: Boxplot of server download speeds in our testbed (31 repetitions and no bandwidth limitation).

the chance for two packets, e.g., request and response, getting transmitted successfully equals $1 - p$ with p being the desired loss rate. The implemented values are shown in Table 2. Bandwidth and delay values for DSL and LTE are taken from [4], we assume no additional loss here. The last two networks emulate slow links measured from in-flight WLAN services [25]. Except for the DSL link with 12ms maximal queueing delay, we assume rather bloated buffers of 200 ms. Thus, our configured delay is the minimum delay and queuing further adds jitter up to the configured buffer size.

Validation. Before conducting measurements, we validate the implemented testbed regarding the network and protocol parameters ensuring the correct protocol choice. We found that the Chromium browser's DNS timeout of 5 s significantly distorts a measurement when a DNS packet is lost and thus moved the DNS server such that no traffic shaping is applied to DNS traffic. Moreover, Figure 2 shows that both server variants yield similar performance for files ≤ 1 MB. This suggests that our results are not biased by the servers' implementations. For this test, we repeated 31 downloads of a single file with the Chromium browser under static network conditions—only 10 ms minimum delay, no loss, and no bandwidth limits. The gap between NGINX and QUIC server emerging at a file size of 10 MB is not relevant since our website sizes are much smaller (see Figure 1). Independent resources are even smaller, the largest being 4 MB.

3.3 Performing Measurements

The actual measurements are performed inside a virtual machine equipped with 6 cores and 8 GB of memory running Arch Linux kernel Version 4.18.16. To measure a single setting consisting of one website, network, and protocol configuration, a Mahimahi replay shell with the described network stack is used. A single setting gets measured over 31 runs to gain statistical significance and at the same time keep the number of runs/videos manageable. We utilize the Browsertime [28] framework to instrument the browser. It records videos of the loading process that we subsequently evaluate for the visual metrics. For each run, Browsertime opens up a fresh Chromium browser Version 70.0.3538.77. In total, this leads to 760 configurations (38 domains, 4 network, and 5 protocol settings). We validated that each run completed successfully by reviewing the video recordings manually.

4 QUIC VS. TCP PERFORMANCE

We evaluate the performance difference with all metrics in the different network settings (across all tested websites) by means of a performance gain. The following equation explains the calculation of the performance gain between a reference protocol, e.g., TCP and a protocol to compare with like QUIC. $\overline{X}$ correspond to the mean over the 31 runs.

$$performance\ gain^{TCP}_{QUIC} = \frac{\overline{X}_{QUIC} - \overline{X}_{TCP}}{\overline{X}_{TCP}}$$

If not stated otherwise, numbers provided in the text are mean performance gains over all websites for SI. Besides comparing means we also utilize an ANOVA test to tell whether there is a statistically significant difference in the distribution of the 31 runs of two protocols. If the ANOVA test for two settings is $p < 0.01$ (significance level), we count the setting with the lower mean as significantly faster otherwise no conclusion can be drawn. The results of our measurements are depicted in Figure 3. We show the CDFs of the performance gain of the different metrics comparing stock TCP to the other protocol stacks. LVC is left out in this figure because in contrast to PLT there is no relevant difference visible.

DSL and LTE. For the lossless DSL and LTE scenarios, the protocols separate into two groups both yielding similar performance gains. TCP+ (DSL: -0.05$^{TCP+}_{TCP}$, LTE: -0.08$^{TCP+}_{TCP}$) and TCP+BBR (DSL: -0.05$^{TCP+BBR}_{TCP}$, LTE: -0.09$^{TCP+BBR}_{TCP}$) perform almost indistinguishable but against TCP, there is a noticeable improvement visible throughout all metrics. Similarly, QUIC (DSL: -0.09$^{QUIC}_{TCP+}$, LTE: -0.14$^{QUIC}_{TCP+}$) and QUIC+BBR (DSL: -0.09$^{QUIC+BBR}_{TCP+BBR}$, LTE: -0.13$^{QUIC+BBR}_{TCP+BBR}$) perform equally but are still quite a bit faster than the two tuned TCP variants. For these two networks, the congestion control choice does not make a significant difference, which is likely due to the small queue. Stock TCP indeed lags behind all other protocols showing that stock TCP should not be used to compare against QUIC here. QUIC achieves to decrease the average SI by -131.3 ms$^{QUIC}_{TCP}$ (DSL) and -344.9 ms$^{QUIC}_{TCP}$ (LTE), but also against TCP+ by still -87.1 ms$^{QUIC}_{TCP+}$ (DSL) and -215.9ms$^{QUIC}_{TCP+}$ (LTE).

In a second step, we take a look at the ANOVA test results focussing on DSL (LTE yields equivalent results). When comparing the runs of TCP+ and QUIC in DSL with PLT

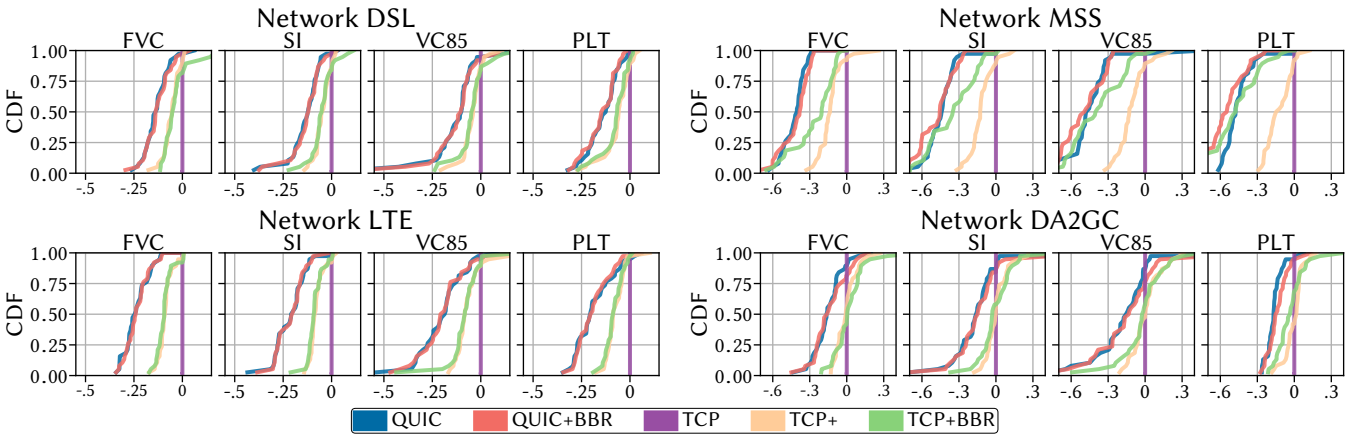

Figure 3: CDF of the performance gain over all websites with TCP as reference protocol. If the performance gain is < 0 (left side of plot) then the compared protocol is faster than TCP.

as the metric with each other, 30 of the 38 websites yield a significant improvement with QUIC. For the remaining 8 websites, none was significantly faster than TCP+. For SI even 32 are faster and only 6 show no significant difference. Similar results can be seen when comparing QUIC+BBR with TCP+BBR this way. For TCP+ and TCP in the same scenario with PLT as the metric, 25 websites are faster with TCP+, for 12 there is no significant difference and only 1 website was significantly slower. Again when comparing TCP+BBR with TCP+ and similarly QUIC+BBR with QUIC for DSL and LTE throughout all metrics, we find for the majority of the websites no difference. These results line up with the results shown in Figure 3. Moreover, the steep incline of the CDFs for QUIC and TCP+ indicate that the website size or structure seems to have little influence on the achievable gain. Only looking at SI and VC85, we see a small percentage of measurements where QUIC has a significantly higher gain.

In-Flight Wifi. For the networks MSS and DA2GC, the overall picture is quite similar—meaning QUIC as well as QUIC+BBR, are usually faster than TCP+ (MSS: -0.36$^{\text{QUIC}}_{\text{TCP+}}$, DA2GC: -0.14$^{\text{QUIC}}_{\text{TCP+}}$) and TCP+BBR (MSS: -0.18$^{\text{QUIC+BBR}}_{\text{TCP+BBR}}$, DA2GC: -0.10$^{\text{QUIC+BBR}}_{\text{TCP+BBR}}$). But there are some important differences, for the MSS link with the highest loss rate (6 %), TCP+BBR operates much better than TCP+ (-0.26$^{\text{TCP+BBR}}_{\text{TCP+}}$). Since BBR does not use loss as a congestion signal it increases its rate regardless of this random loss. This means that in this case, the choice in congestion control has a greater impact on the performance than the protocol choice itself. At the time of the FVC, TCP+BBR is already -2866.2 ms (avg.) quicker than TCP+ but with each later metric, the gap widens so that at PLT, TCP+BBR can keep up the pace even against QUIC and is 11395.4 ms (0.21×) quicker. This shows that TCP with BBR needs some time to catch up and thus affects the FVC much more than the later PLT. For the QUIC protocols, the picture is similar. At first, QUIC and QUIC+BBR are similarly fast and mostly better than TCP+BBR. But as the loading process commences QUIC+BBR outperforms QUIC slightly, e.g., -1828.3 ms$^{\text{QUIC+BBR}}_{\text{QUIC}}$ better SI. QUIC with CUBIC, nevertheless, is reasonably fast being still a legit option to use. The shape of the performance gain CDFs of QUIC+BBR and TCP+BBR are very similar especially for PLT highlighting the influence of the congestion control once again. We believe that QUIC with CUBIC is still competitive due to QUIC's ability to circumvent head-of-line blocking and its large SACK ranges. For the MSS network, QUIC reduces the SI by -8364.8 ms$^{\text{QUIC}}_{\text{TCP+}}$ (avg.) compared to TCP+ and by -2091.5 ms$^{\text{QUIC+BBR}}_{\text{TCP+BBR}}$ when taking both BBR protocols into account.

The last network, DA2GC, also has a high loss rate (3.3 %) but a much lower bandwidth. This is the only scenario where we observe no significant difference for most websites among all TCP configurations even with the ANOVA test. We also see that in a small fraction of our measurements stock TCP outperforms QUIC and the tuned TCP variants. Nevertheless, again the QUIC variants are generally significantly faster with a higher performance gain at the FVC (e.g., -0.14$^{\text{QUIC}}_{\text{TCP+}}$) that persists towards the PLT (e.g., -0.16$^{\text{QUIC}}_{\text{TCP+}}$). The choice of the congestion control algorithm does not seem to have a significant impact here likely due to the low bandwidth. Only for PLT we find QUIC with CUBIC to be slightly superior over QUIC with BBR. There is not a single website where QUIC+BBR yields a significantly faster performance. The SI decreases with QUIC by -2632.5 ms$^{\text{QUIC}}_{\text{TCP+}}$ vs. TCP+ and by -1372.5 ms$^{\text{QUIC+BBR}}_{\text{TCP+BRR}}$ for BBR.

Discussing Metrics. Some of the websites exhibit very poor performance regarding the visual metrics VC85 and SI. We observe this behavior especially for the DA2GC network with performance gains of up to +1.0 compared to stock TCP (not shown, plots cropped for readability). The reason for these outliers is that the protocol choice has such a substantial

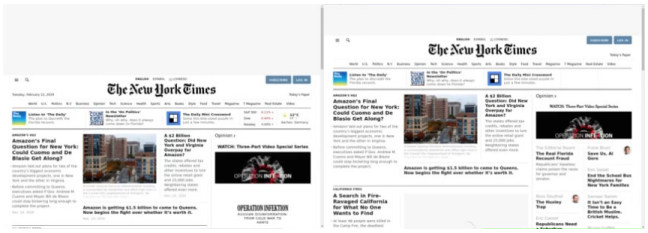

Figure 4: Screenshot during the loading process of the nytimes.com website. Left TCP right QUIC. QUIC in comparison delays a top banner leading to bad scores in visual metrics compared to the final website.

impact on some websites that their resources load in different orders resulting in very distinct rendering sequences.

Figure 4 shows such a scenario exemplary for the nytimes.com website in the DA2GC network. Here TCP reaches VC85 after ~48s whereas QUIC needs ~124s even though the PLT for QUIC (~141s) is much faster than for TCP (~170s). For TCP the upper part of the website loads comparably early such that the lower elements are already rendered at their final positions. In contrast to that QUIC manages to receive the lower contents first. Later, when the upper banner completes loading it shifts the whole website down. Therefore, VC85 fails to express this setting given the large shift. Similarly, SI is affected since it integrates over visual completeness over time. Thus, it critically depends on the website, the browser's loading order, and a user's preference for how a website should load to know which metric to use.

Protocol Design Impact. Within our testbed, any TCP configuration needs to fulfill two complete RTTs before the actual HTTP request can be sent out to the server—TCP handshake plus TLS setup. In contrast, QUIC requires only one RTT to do so—the first CHLO gets rejected by the server since the server certificates are unknown to the client. We are interested in whether this 1 RTT difference can explain the remaining performance gap between QUIC and TCP+. However, the complex interactions with multiple servers complicate an analysis since these connections are interleaved simply subtracting 1 RTT is not possible. We, therefore, take a look at two websites served only via a single IP (see Figure 1): wikipedia.org and gnu.org. We subtract one RTT from the FVC, as the earliest metric and one RTT from the PLT as the latest completing metric. Table 3 shows the results in the different network settings for TCP+ and QUIC and additionally for MSS using the BBR variants of both.

For DSL and LTE the corrected difference is below one RTT and there are three cases where even TCP+ is slightly faster now. For MSS in all cases with CUBIC as the congestion control, QUIC is faster but only to a maximum of 1.4× RTT. Since within this network congestion control has a huge impact, we consider also BBR here. Overall in MSS with BBR,

Net	Website	Metric	[ms]	[RTT]
DSL	gnu.org	FVC	0.5	0.020
DSL	wikipedia.org	FVC	-8.2	-0.341
DSL	gnu.org	PLT	1.6	0.066
DSL	wikipedia.org	PLT	-3.1	-0.128
LTE	gnu.org	FVC	0.6	0.008
LTE	wikipedia.org	FVC	-40	-0.538
LTE	gnu.org	PLT	-30	-0.412
LTE	wikipedia.org	PLT	-13	-0.175
MSS	gnu.org	FVC	-196	-0.258
MSS	wikipedia.org	FVC	-412	-0.542
MSS	gnu.org	PLT	-1100	-1.447
MSS	wikipedia.org	PLT	-529	-0.696
DA2GC	gnu.org	FVC	-130	-0.497
DA2GC	wikipedia.org	FVC	-1384	-5.283
DA2GC	gnu.org	PLT	39	0.150
DA2GC	wikipedia.org	PLT	-1005	-3.834
MSS	gnu.org	FVC	-404	-0.532
MSS	wikipedia.org	FVC	-143	-0.189
MSS	gnu.org	PLT	-477	-0.628
MSS	wikipedia.org	PLT	451	0.593

Table 3: Difference between the means over the 31 runs of QUIC and TCP+ when subtracting one RTT. Values <0 denote that QUIC was faster. The lower MSS table compares QUIC+BBR and TCP+BBR.

the difference is also below of one RTT and for wikipedia.org and PLT even TCP+BBR is faster. Instead with DA2GC, the outcome is clearly for QUIC for the wikipedia.org website. Table 3 shows nicely that QUIC's RTT reducing design clearly improves the performance. Even though, TCP Fast Open and TLS 1.3 early-data would close the gap, especially Fast Open remains challenging to deploy. Furthermore, having no head-of-line blocking could still be a reason why in the majority of the cases QUIC is still slightly faster, especially, when the networks are lossy. We expect further improvements when using 0-RTT connection establishment with QUIC.

5 CONCLUSION

Comparisons between TCP and QUIC have often been biased up until now. In this paper, we extended the Mahimahi framework to support QUIC and perform reproducible performance measurements of 38 websites under different protocol and network scenarios. We show that tuning TCP parameters has a tremendous impact on the results for performance comparisons which can not be neglected when comparing TCP and QUIC. Yet, in many settings, QUIC's performance is still superior but the gap gets narrower. Moreover, we find that QUIC's higher performance is caused mostly due to its superior design during the connection establishment. We assume that besides the RTT reducing design, features like no head-of-line blocking increase QUIC's performance, especially in lossy networks. In those lossy networks, we also find that the choice of the congestion control algorithm has a much larger impact than the protocol itself. In our opinion, QUIC is still the preferred protocol for the future Web since it paves the way for continuous evolution.

ACKNOWLEDGMENTS

This work has been funded by the DFG as part of the CRC 1053 MAKI and SPP 1914 REFLEXES.

REFERENCES

[1] Alexa. 2019. Alexa Top 500 Global Sites. https://www.alexa.com/topsites.

[2] Prasenjeet Biswal and Omprakash Gnawali. 2016. Does QUIC Make the Web Faster?. In *IEEE Global Communications Conference (GLOBECOM)*. https://doi.org/10.1109/GLOCOM.2016.7841749

[3] Enrico Bocchi, Luca De Cicco, Marco Mellia, and Dario Rossi. 2017. The Web, the Users, and the MOS: Influence of HTTP/2 on User Experience. In *Springer Passive and Active Measurement (PAM)*. https://doi.org/10.1007/978-3-319-54328-4$_4$

[4] Breitbandmessung. 2018. Breitbandmessung Ergebnisse als interaktive Darstellung. https://web.archive.org/web/20181115105855/https://breitbandmessung.de/interaktive-darstellung.

[5] Jake Brutlag, Zoe Abrams, and Pat Meenan. 2011. Above the Fold Time: Measuring Web Page Performance Visually. In *Velocity: Web Performance and Operations Conference*. http://conferences.oreilly.com/velocity/velocity-mar2011/public/schedule/detail/18692.

[6] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. 2016. BBR: Congestion-Based Congestion Control. *ACM Queue* 14, 5 (2016). https://doi.org/10.1145/3012426.3022184

[7] Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. 2015. HTTP over UDP: An Experimental Investigation of QUIC. In *ACM Symposium on Applied Computing (SAC)*. https://doi.org/10.1145/2695664.2695706

[8] S. Cook, B. Mathieu, P. Truong, and I. Hamchaoui. 2017. QUIC: Better for what and for whom?. In *IEEE International Conference on Communications (ICC)*. https://doi.org/10.1109/ICC.2017.7997281

[9] Qingzhu Gao, Prasenjit Dey, and Parvez Ahammad. 2017. Perceived Performance of Top Retail Webpages In the Wild: Insights from Large-scale Crowdsourcing of Above-the-Fold QoE. In *ACM Workshop on QoE-based Analysis and Management of Data Communication Networks (Internet-QoE)*. https://doi.org/10.1145/3098603.3098606

[10] Google. 2019. Playing with QUIC - The Chromium Projects. https://www.chromium.org/quic/playing-with-quic.

[11] Google. 2019. Speed Index. https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index.

[12] S. Ha, I. Rhee, and L. Xu. 2008. CUBIC: A New TCP-friendly High-speed TCP Variant. *ACM SIGOPS Operating Systems Review (OSR)* 42, 5 (2008). https://doi.org/10.1145/1400097.1400105

[13] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove. 2017. Taking a Long Look at QUIC: An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols. In *ACM Internet Measurement Conference (IMC)*. https://doi.org/10.1145/3131365.3131368

[14] Conor Kelton, Jihoon Ryoo, Aruna Balasubramanian, and Samir R. Das. 2017. Improving User Perceived Page Load Times Using Gaze. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/kelton.

[15] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. 2017. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *ACM SIGCOMM*. https://doi.org/10.1145/3098822.3098842

[16] Anna Maria Mandalari, Marcelo Bagnulo, and Andra Lutu. 2015. TCP Fast Open: Initial Measurements. In *ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT) Student Workshop*. https://www.simula.no/publications/tcp-fast-open-initial-measurements.

[17] P. Megyesi, Z. Krämer, and S. Molnár. 2016. How quick is QUIC?. In *IEEE International Conference on Communications (ICC)*. https://doi.org/10.1109/ICC.2016.7510788

[18] Moz. 2019. Top Sites: The 500 Most Important Websites on the Internet. https://moz.com/top500.

[19] Ferdinand Mütsch. 2017. Caddy - a modern web server (vs. nginx). https://ferdinand-muetsch.de/caddy-a-modern-web-server-vs-nginx.html.

[20] K. Nepomuceno, I. N. d. Oliveira, R. R. Aschoff, D. Bezerra, M. S. Ito, W. Melo, D. Sadok, and G. Szabó. 2018. QUIC and TCP: A Performance Evaluation. In *IEEE Symposium on Computers and Communications (ISCC)*. https://doi.org/10.1109/ISCC.2018.8538687

[21] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate Record-and-Replay for HTTP. In *USENIX Annual Technical Conference (ATC)*. https://www.usenix.org/conference/atc15/technical-session/presentation/netravali.

[22] Christoph Paasch. 2016. Network support for TCP Fast Open. *Presentation at NANOG* 67 (2016).

[23] Sivasankar Radhakrishnan, Yuchung Cheng, Jerry Chu, Arvind Jain, and Barath Raghavan. 2011. TCP Fast Open. In *ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. https://doi.org/10.1145/2079296.2079317

[24] E. Rescorla. 2018. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. RFC Editor. http://www.rfc-editor.org/rfc/rfc8446.txt

[25] John P. Rula, James Newman, Fabián E. Bustamante, Arash Molavi Kakhki, and David Choffnes. 2018. Mile High WiFi: A First Look At In-Flight Internet Connectivity. In *IW3C2 World Wide Web Conference (WWW)*. https://doi.org/10.1145/3178876.3186057

[26] Jan Rüth and Oliver Hohlfeld. 2018. Demystifying TCP Initial Window Configurations of Content Distribution Networks. In *IFIP/IEEE Network Traffic Measurement and Analysis Conference (TMA)*. https://doi.org/10.23919/TMA.2018.8506549

[27] Michael Seufert, Raimund Schatz, Nikolas Wehner, Bruno Gardlo, and Pedro Casas. 2019. Is QUIC becoming the New TCP? On the Potential Impact of a New Protocol on Networked Multimedia QoE. In *IEEE International Conference on Quality of Multimedia Experience (QoMEX)*.

[28] sitespeed.io. 2019. Browsertime - Your browser, your page, your scripts! https://github.com/sitespeedio/browsertime.

[29] Maarten Wijnants, Robin Marx, Peter Quax, and Wim Lamotte. 2018. HTTP/2 Prioritization and Its Impact on Web Performance. In *IW3C2 World Wide Web Conference (WWW)*. https://doi.org/10.1145/3178876.3186181

[30] Y. Yu, M. Xu, and Y. Yang. 2017. When QUIC meets TCP: An Experimental Study. In *IEEE International Performance Computing and Communications Conference (IPCCC)*. https://doi.org/10.1109/PCCC.2017.8280429

[31] Torsten Zimmermann, Benedikt Wolters, and Oliver Hohlfeld. 2017. A QoE Perspective on HTTP/2 Server Push. In *ACM Workshop on QoE-based Analysis and Management of Data Communication Networks (Internet-QoE)*. https://doi.org/10.1145/3098603.3098604