This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3134562, IEEE Transactions on Network and Service Management

Evaluating QUIC Performance over Web, Cloud Storage and Video Workloads

Tanya Shreedhar*, Rohit Panda†, Sergey Podanev† and Vaibhav Bajpai†
*IIIT-Delhi, India †TUM, Germany

***Abstract*—QUIC was launched in 2013 with a goal to provide reliable, connection-oriented and end-to-end encrypted transport and is recently standardized in May 2021 by the Internet Engineering Task Force (IETF). This work evaluates QUIC performance over the web, cloud storage, and video workloads and compares them to traditional TLS/TCP. To this end, we have designed tests (`quic_perf`, `tls_perf` and `video`) and conducted measurements from 2018 – 2021 using multiple vantage points: an educational network, a high-bandwidth low-RTT residential link in Germany and a low-bandwidth high-RTT residential link in India. We target Alexa Top-1M for web workloads and probe them towards the support for QUIC, TLS 1.2 and TLS 1.3. By measuring >5.7K websites that support QUIC, we observe that QUIC has up to ≈140% lower mean connection times than TLS 1.2/1.3 over TCP for low-bandwidth and high-RTT networks. When comparing different versions of QUIC, we observe that IETF QUIC connection times are slightly better than different versions (`Q050`, `Q046`, `Q044`, `Q043`, `Q039` and `Q035`) of gQUIC. For cloud storage workloads, we observe that TLS 1.2 over TCP exhibits higher throughput for larger file sizes (>20 MB up to 2 GB), while QUIC exhibits higher throughput for smaller file sizes (≤20 MB) while downloading files from Google Drive. At the same time, QUIC has much higher CPU utilization than TLS 1.2 over TCP, almost double while downloading a large file (200 MB) from Google Drive due to in-kernel optimizations that benefit TCP. For video workloads, we observe that QUIC is 534 ms faster than TLS 1.2 over TCP from India (406 ms from Germany) in establishing a connection to YouTube media servers. Although we witness that (similar to cloud storage workloads) the overall download rate is higher over TLS, QUIC still tends to depict better video content delivery with reduced stall events and up to 50% lower stall durations due to its lower latency overheads. To support reproducibility, the developed tests and the collected data are made publicly available to the community.**

***Index Terms*—Content Delivery Networks (CDN), Transmission Control Protocol (TCP), Transport Layer Security (TLS), QUIC, Internet Engineering Task Force (IETF), Application Workloads.**

I. Introduction

To meet the ever-growing low-latency demands, Google proposed QUIC [1], a new transport protocol that is recently standardized by the Internet Engineering Task Force (IETF) [2]–[6]. The design of QUIC came with the experience gathered from SPDY [7], which later got standardized as HTTP/2 [8]. QUIC is a new protocol because deploying extensions to TCP to support latency-sensitive applications has seen diminishing returns over the years. This is due to the ossification [9] of the Internet caused by intervening middleboxes on the path that inhibit wider deployment of extensions to TCP such as TCP Fast Open [10] or protocols such as Stream Control Transmission Protocol (SCTP) [11] and Multipath TCP [12].

QUIC uses UDP as a substrate and encrypts protocol headers (and associated payload) to prevent middleboxes from making modifications. QUIC is written in user-space (see: §II) to allow rapid deployment of protocol updates that are not tied to regular OS release cycles. Google Chrome and Google apps on Android, when possible, allow interaction with Google services (such as YouTube *et al.*) over QUIC. This has led to increased adoption of QUIC to reach 8% of global Internet traffic, with 4.6% of all websites being QUIC-capable [13] as of July 2020. Google [1] reports that QUIC (using `Q035`, deployed in 2016) has been able to reduce search latency by 8% for desktop users and 3.6% for mobile users.

For rapid testing of newly released versions, Piraux *et. al.* [14] developed a test suite that interacts with public QUIC servers and verifies the conformance with the IETF specifications. However, due to the rapid development and release cycle, performance studies analyzing QUIC quickly become dated. There has been little work (see: §III) that independently (of Google) evaluates the benefits of using QUIC in uncontrolled settings and different networks using more recent deployments of QUIC. Understanding protocol behavior in realistic workloads and real network settings is pertinent for the large-scale adoption and deployment of QUIC over the Internet. In this paper, we close this research gap by investigating the benefits of using QUIC for different (web, cloud storage and video) workloads over traditional TLS/TCP transport in different networks. The measurements are performed using a VM and a Raspberry Pi from a high-bandwidth, low-RTT residential link in Germany and a low-bandwidth, high-RTT residential link in India.

The key **contributions** of this paper are two-fold: a) design and implementation of `quic_perf`, `tls_perf` and `video` tests (see: §IV) with QUIC support, which are open-sourced to the community and b) findings reported from the analysis of the collected dataset from 2018–2021 (also publicly released) as summarized below –

`tls_perf` and `quic_perf`, active measurement tests written in C. The tests evaluate the performance (latency and throughput) of TLS/TCP and QUIC connections, respectively. `tls_perf` test uses `libcurl` [15] underneath, while `quic_perf` uses `lsquic` [16], an open-source implementation of QUIC used internally by LiteSpeed CDN. For measuring latency, the tests connect to a given web-server from the Alexa Top-1M websites list and capture various connection-level metrics such as protocol version, DNS lookup time,

1932-4537 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on December 11,2021 at 11:39:12 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3134562, IEEE Transactions on Network and Service Management

connection times, time to first byte (TTFB) and download time. We find that only 5.7K websites amongst Alexa Top-1M actively support QUIC. For measuring throughput, we uploaded files of different sizes (namely 1 KB to 2 GB) to Google Drive and repeatedly downloaded them using the tests with various QUIC versions and TLS 1.2 over TCP (since TLS 1.3 is not supported by Google Drive yet).

`video_download`, an active measurement test written in C. The test downloads and mimics the playout of YouTube videos on the command line. The test uses `libcurl` and `lsquic` underneath to provide TLS/TCP and QUIC support. We have added monitoring points in the `video_download` test to measure connection establishment times, achievable throughput and CPU utilization as key performance indicators when downloading YouTube videos both over QUIC and TLS 1.2 over TCP (since TLS 1.3 is not supported by YouTube yet). In `video_streaming`, we utilize an adaptive streaming test-suite, *VideoMon*, that streams pre-defined videos from YouTube over QUIC and TLS/TCP [17]. The test is written in Python, and we run it in a container environment in a university VM. To emulate real-world environments, we introduce different packet losses over the links. Overall, we collect several performance metrics that affect a user's QoE while streaming, e.g., startup delay, number of quality switches, number and the duration of stalls. Our findings are –

Web Workloads – QUIC reports lower handshake times than TLS 1.2 and 1.3 over TCP for both IPv4 and IPv6, with IETF QUIC reporting ≈50% lower latency than gQUIC versions for half of the samples over IPv6. Amongst the gQUIC versions we tested, `Q050` performs better than others. However, QUIC exhibits diminishing returns in latency as the connection state prolongs, with latency benefits declining when TTFB and download times are compared with that of TCP/TLS. We observe that Google CDN serves the largest sample ($\approx 68\%$) of (> 5.7K) websites that offer QUIC support both over IPv4 and IPv6. Surprisingly, EdgeCast CDN seems to offer the lowest handshake times over IPv4, although it does not (yet) provide QUIC services over IPv6 (see: §V).

Cloud Storage Workloads – For downloading files from Google Drive, we observe that the mean throughput of QUIC is higher for small file sizes, but for larger file sizes (> 20 MB up to 2 GB), TLS/TCP performs better. It is because, for smaller file sizes, connection times and TTFB dominate the total download time, while the gain diminishes as the file size increases. In terms of resource utilization, QUIC has high CPU usage as the in-kernel optimizations, such as large receive offload (LRO) that currently benefit TCP, are not available for UDP flows in the Linux kernel we tested (see: §VI).

Video Workloads – For `video_download`, QUIC provides much better improvements in India than in Germany for connection times, with a reduction of 550 ms (410 ms in Germany) compared to TLS 1.2 for half of the samples. The overall download rate for TLS 1.2 is higher than QUIC (similar observation to that of cloud storage workloads), and the download rate for `Q035` is higher than other QUIC versions for both India and Germany. For YouTube `video_streaming`, TLS/TCP incurs a larger startup delay compared to QUIC, and the gap in the performance increases in a lossy network. Despite lower overall download rate, QUIC has a better video content delivery with reduced stall events and lower stall durations due to its reduced latency overheads and better loss recovery mechanism (see: §VII).

To encourage reproducibility [18] of our work, `tls_perf`, `quic_perf` and `video` tests with the added QUIC support are open-sourced and the collected dataset is publicly available to the research community [19].

II. Background

Google's SPDY [7] improved latency by introducing features like concurrent multiplexing of requests over a single TCP connection, HTTP header compression, request prioritization, server push, and server hint. SPDY's stream multiplexing feature gives it an edge over HTTP/1.1 [20]. However, being built on top of TCP, applications still experience high handshake latency and head-of-line blocking (all SPDY streams multiplexed on the same TCP connection will be blocked by a lost or out-of-order packet). SPDY led to the design of HTTP/2 [8], which was released in 2015. It differs from SPDY in header compression. HTTP/2 has improved performance over HTTP/1.1, and the performance benefits of HTTP/2 over HTTP/1.1 were consistently high for a range of network delays tested in [21]. However, HTTP/2 also suffers from the TCP head-of-line blocking problem. The biggest challenge in deploying changes to the transport layer is middleboxes, as they tend to block any unfamiliar flow for security reasons [22]. Several proposed TCP extensions such as TCP Fast Open [10] or protocols such as SCTP [11] and Multipath TCP [12], [23] have not seen wide deployment due to the middleboxes blocking the flows. This led Google to develop QUIC, a new transport protocol designed on top of UDP. QUIC support was initially added to Chrome in June 2013 for the development team, and in early 2014 a tiny number of users were allowed early access. Gradually the number of users increased, and eventually, by January 2017, QUIC was enabled for all users of Chrome and the Android YouTube app [1]. Rüth *et. al* [24] studied QUIC usage in the wild and found 161K out of 150M scanned websites support QUIC. This adoption was attributed to support from Google and Akamai. QUIC accounted for up to 9.1% of the Internet traffic and Google's egress traffic over QUIC was 42.1%. The mobile YouTube increased Google's egress traffic over QUIC from 15% to 30%. Since using UDP as a base promises Internet-scale deployability, many emerging protocols are using UDP [25] (and even QUIC) as a substrate to target specific application use-cases [26].

QUIC Features – QUIC is a user-space protocol that allows for faster development and deployment. QUIC replaces the traditional (TCP, TLS and HTTP/2) HTTP(S) stack. QUIC employs flow control both at the connection level as well as the stream level. While connection-level flow control restricts the buffer size that the sender can exhaust aggregated over all streams, stream-level flow control restricts the buffer size that the sender can exhaust over a single stream.

QUIC provides improvements over TLS 1.2/1.3 over TCP in terms of connection establishment times. TLS 1.2 and 1.3

1932-4537 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on December 11,2021 at 11:39:12 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3134562, IEEE Transactions on Network and Service Management

(over TCP) use 3-RTT and 2-RTT connection establishment times, whereas QUIC uses 1-RTT for the first time connections. For the subsequent connections, the handshake times are 2-RTT, 1-RTT and 0-RTT for TLS 1.2, 1.3 (over TCP) and QUIC, respectively. This is because QUIC by design has TLS negotiation built into its protocol, whereas TCP needs to complete a transport handshake before initiating a TLS handshake. Thus, QUIC is *atleast* 1-RTT faster than TLS/TCP in terms of connection establishment times.

Another design feature of QUIC is stream multiplexing in transport, allowing data to be delivered *in order* at the stream level. Thus a lost packet affects only the streams whose data it carried and not other streams. This is unlike TCP, which requires packets to be delivered in order at the connection level and can lead to a head-of-line (HoL) blocking problem.

QUIC's loss-recovery mechanisms builds on and simplifies TCP's recovery mechanisms like TCP Selective Acknowledgements (SACKs) [27], Fast Retransmit [28], Early Retransmit [29], Recent Acknowledgement (RACK) [30], etc. QUIC also has monotonically increasing packet numbers which help in distinguishing original and retransmitted packets and a more accurate RTT measurement. Also, unlike TCP, QUIC supports multiple ACK ranges, which helps in speeding up recovery and reducing spurious retransmissions. In addition, QUIC ACKs are irrevocable, aiding in simpler implementation and reducing the memory pressure on the sender [2] [3].

QUIC supports pluggable congestion control and is designed to support different congestion control algorithms. QUIC uses a modification of TCP NewReno [31] as a default congestion control algorithm [3].

QUIC as an enabler for HTTP/3 – The IETF QUIC working group was formed in Oct 2016, and QUIC was standardized in May 2021. HTTP/3 [32] is the mapping of HTTP semantics over QUIC, which takes full advantage of QUIC's stream-multiplexing capabilities. QUIC streams are different from HTTP/2 streams and are independent of each other, so HTTP/2 header compression cannot be used without the issue of head-of-line blocking. Thus, a new header compression algorithm QPACK [4] is used instead of HTTP/2 style header compression, HPACK [33]. HTTP/3 also has server push similar to HTTP/2. But unlike HTTP2, HTTP/3 has no prioritization, which is considered to be too complicated [34]. This simplistic design and the advantages of faster handshakes (courtesy QUIC) give HTTP/3 an edge over HTTP/2. A recent study [35] shows an increased adoption of HTTP/3 over the Internet indicating a promising future.

III. Related Work

Early studies on QUIC [36]–[38] (measuring Q02X) have become dated with time due to the drop in the amount of incoming traffic that Google receives from clients using such older versions (early 2016 or before) of QUIC. As such, we focus the related work on more recent studies on QUIC. Langley *et al.* [1] present the first large-scale measurements of QUIC across its various versions. They design an extensive test and evaluation environment for the development of gQUIC using the knowledge and control over the Chrome browser and its web services. They use measurements collected (late 2016 and early 2017) using Q035 to show that QUIC reduces search and video latency by 8% (due to lower number of handshakes) and video rebuffer rates by 18% (due to better loss-recovery mechanisms) on an average for desktop users. They show that these latency benefits vary by geography, wherein higher latency benefits are visible to users in high-RTT and lossy networks. Thus in this work, we specifically measure from a developed (DE) and developing (IN) region to verify and provide observations independent from Google to this phenomenon. The control over client and server software, combined with a large user base, allowed fast and regression-free iterations of the protocol. Cook *et al.* [39] evaluate QUIC's page load time in local controlled testbed environments and find that QUIC has improved performance over TCP in unstable networks (wireless/mobile) but not so much in stable networks. Nepomuceno *et al.* [40] also used the page load time metric to compare QUIC with HTTP/2 and TCP with HTTP/1.1 and found QUIC to perform worse than TCP at different values of RTT and packet loss ratios, contrary to the related work and even our results (§V). This is because they use the Caddy QUIC server, which does not perform well [41]. In addition, they do not take bandwidth limitations into account for their evaluations. Kakhki *et al.* [42] compared QUIC and TCP in different emulated controlled environments (desktop and mobile) and network settings and discussed the performance improvement of different QUIC versions over time. Yu *et al.* [43] discussed QUIC's packet pacing mechanism as a tuning option but did not evaluate it in comparison to TCP. For a fairer comparison, Wang *et al.* [44] implemented QUIC in the Linux kernel and compared its performance with TCP. Rüth *et al.* [24] show the usage of QUIC in the wild. Their scans on the entire IPv4 address space showed 617.59K QUIC-capable IPs and 161K out of 150M domains scanned supported QUIC. Google and Akamai seemed to be the driving force behind QUIC adoption, in which QUIC accounts for 2.6% to 9.1% of the Internet traffic depending on the vantage point. Wolsing *et al.* [45] tune TCP parameters to improve its performance and compare this tuned TCP version with QUIC using the Mahimahi emulation framework. QUIC still outperforms TCP due to its lower connection times and the ability to reduce head-of-line blocking.

Lately, there has also been an interest in applying machine learning (ML) techniques to classify QUIC traffic. For instance, Mazhar *et al.* [46] propose an ML approach to monitor quality of experience (QoE) metrics (such as startup delay and rebuffering events) for encrypted video traffic delivered over HTTPS and QUIC. There have also been proposals to incorporate new features into QUIC. For instance, Coninck *et al.* [47] propose to add multipath capabilities to QUIC. Wu *et al.* [48] propose a learning-based multipath scheduler for Multipath QUIC. Diego *et al.* [49] analyze QUIC traffic from mobile end-user devices and show an increase in the number of Android apps using QUIC. Perkins *et al.* [50] propose a minimal set of extensions to QUIC to support real-time media. Eggert [51] discusses the feasibility of deploying QUIC directly on resource constraint IoT devices. Vaere *et al.* [52] proposes to add a spin signal using three bits in

1932-4537 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on December 11,2021 at 11:39:12 UTC from IEEE Xplore. Restrictions apply.

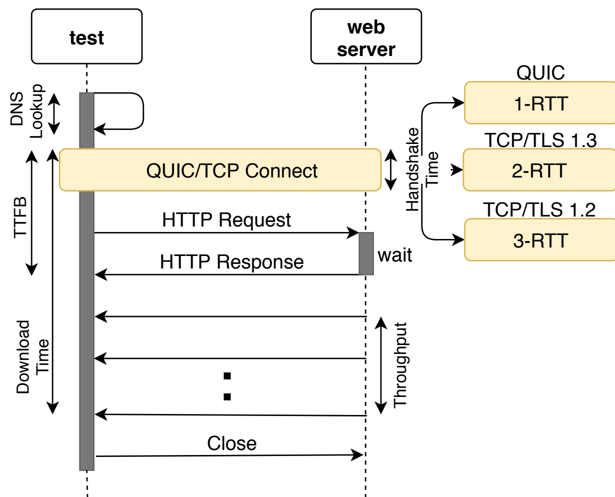


Fig. 1: *A sequence diagram of the* `tls_perf` *and* `quic_perf` *tests and stages where metrics are collected. The connection establishment time for QUIC, TCP/TLS 1.3 and TCP/TLS 1.2 are 1-RTT, 2-RTT and 3-RTT respectively.*

the QUIC header to add explicit support for passive latency measurements. Palmer *et al.* [53] proposes to add unreliable streams to improve QoE of video streaming with QUIC.

There also has been work [42], [54] on measuring YouTube video streaming over QUIC in a controlled environment by emulating varying network conditions. For instance, Bhat *et al.* [55] compare QoE using different Dynamic Adaptive Streaming over HTTP (DASH) algorithms running on top of QUIC by streaming videos from an Amazon EC2 server instance. In [56], the authors primarily focus on QoE, specifically measuring bitrate and quality switches. Seufert *et al.* [57], [58] evaluate YouTube video streaming QoE metrics such as video quality and stalls but find no evidence for QoE improvements of QUIC over TCP. In [59], the authors discuss the insensitivity and unfairness of QUIC protocol to the varying network bandwidth and addition of other TCP subflows on their HAS (HTTP-based Adaptive Streaming) testbed. Rajiullah *et al.* [60] evaluate QUIC using the MONROE platform and conclude that overall QUIC has a negligible impact. Rüth *et al.* [61] performed a user study to evaluate QUIC and TCP's perceived performance for the user and conclude that they are indistinguishable. In this work, we measure different workloads in real-world settings targeting Alexa Top-1M websites, Google Drive cloud storage, and YouTube video downloads and streaming using QUIC and TLS 1.2/1.3 over the Internet. All the related studies to this work either measure QUIC performance in emulated settings or using custom server deployments such as streaming its own video from a DASH server installed on EC2 instances. Hence, we consider them orthogonal to our work. To the best of our knowledge, this is the first study to comprehensively evaluate varying workloads, namely web, cloud storage, and video, over more recent deployments of QUIC (Q035 and newer, including IETF) using residential links at two different vantage points (India and Germany) in an uncontrolled setting.

Version	Features	Release Date
`Q035`	Endpoints can set stream limit	Jun 2016
`Q039`	Uses big endian format	Apr 2017
`Q043`	PRIORITY frames	Feb 2018
`Q044`	IETF header format	Jun 2018
`Q046`	Q044 and demux bit	Feb 2019
`Q050`	Header protection, initial obfuscators	Aug 2019
`IETF`	IETF Draft-24 [62]	Nov 2019

TABLE I: *QUIC protocol versions observed during the course of our measurements. The release date is when the version was introduced in the Chromium repository.*

IV. Methodology

We use a Virtual Machine running Ubuntu 16.04.5 LTS equipped with a 1Gbps LAN connection connected to the Leibniz Supercomputing Centre (LRZ) network. We also use a Raspberry Pi 3 Model B (RPi) with 1 GB RAM and 32 GB storage connected to a 100 Mbps line installed at a residential location in Munich, Germany. We also installed another RPi in Bhubaneswar, India, equipped with a 20 Mbps connection. Our VM-based setup represents a near-ideal network environment, whereas the RPi-based setup allows us to perform measurements in conventional residential networks. We devised specific tests namely `quic_perf`, `tls_perf` and `video` for our evaluation. The tests leverage LiteSpeed `lsquic` [16] and `libcurl` [15] libraries to provide QUIC and TLS/TCP support respectively.

The target websites for the measurements are taken from [24], [63], which maintains a list of websites supporting QUIC grouped by Alexa Top-1M list and IPv4 `zmap` scans. Our tests connect to targeted web servers and measure application and transport-level metrics of the established connection. From our measurements, we find that only 5722 websites out of Alexa Top-1M support QUIC. We target this set of websites to analyze the performance of QUIC for web transport over the Internet. We have also created Debian packages of our tests using `cmake` and `cpack` to facilitate their installation on a Raspbian environment running on a Raspberry Pi. We publicly release the source code and Debian packages of these tests to the community. The whole process can be orchestrated by `crontab`, which runs the `bash` script every three hours. To ensure that the previous test cycle is over, a test run is limited with a `timeout` utility to complete the script within one hour. The data was collected between November 2018 and July 2021. We make the data publicly available to the community. We further describe the tests in detail below.

A. `tls_perf` and `quic_perf`

We developed a `tls_perf` test (written in C) to measure TLS/TCP connections. The test uses `libcurl` [15] library, which is used to configure and execute connections with different parameters such as URL, port and timeout. `libcurl` is also used for logging the metadata about the connection, which can be accessed after the connection terminates.

Similarly, we also developed `quic_perf` test (written in C) to measure QUIC performance. In order to add QUIC support, we evaluated existing QUIC libraries [64], [65] and

1932-4537 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on December 11,2021 at 11:39:12 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3134562, IEEE Transactions on Network and Service Management

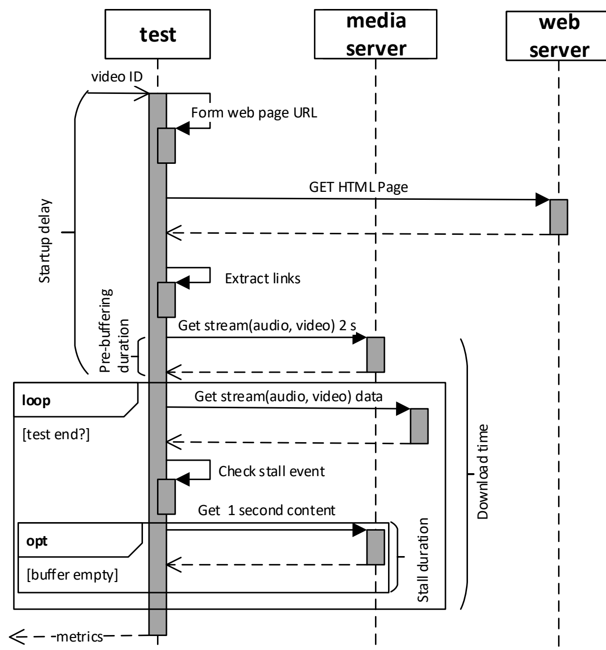


Fig. 2: *A sequence diagram of the* `video_download` *test and stages where metrics (handshake times, throughput, utilization) are collected.*

chose `lsquic`, an open-source QUIC client library [16] for integration purposes. `lsquic`[1] is developed and maintained by LiteSpeed CDN with a promise for long term support for QUIC development and has wide support of gQUIC protocols currently deployed on the Internet and IETF QUIC draft versions currently under development. Being written in C, `lsquic` allows easy pluggability with our existing measurement tools (such as the `video` tests that are written in C).

Using `lsquic`, we added support for the gQUIC (`Q035`–`Q050`) and IETF QUIC (`ID-24`) versions (Table I) in `quic_perf` spanning two years (2018–2020) of QUIC development. We also added features such as support for concurrent streams, a new HPACK interface, a compression algorithm that is used to compress HTTP headers in HTTP/2 [66], support for `0-RTT`, improved fall-back mechanism and finally `ls-qpack`.

Figure 1 provides a sequence diagram of the operation of both tests and metrics they collect in the process. Both the tests connect to a given server on a given port. With each measurement, they output various connection-level metrics such as DNS lookup time, handshake time, TTFB, HTTP response time and overall download time in the CSV format along with the metadata associated with each measurement such as timestamp, hostname, path, IP address, port and protocol version. With these tests, we measure QUIC performance over web (§V) and cloud storage (§VI) workloads.

[1]While developing the tools for our study, we also actively interacted with the main developer of lsquic and have incorporated his feedback on the paper, which supports active collaboration with this codebase.

B. `video` tests

`video_download` – We have developed a `video_download` test (written in C) that downloads and mimics the non-adaptive and step-down playout of YouTube videos. This test aims to compare TLS/QUIC in their abilities to deliver the best quality of video without disruptions. The parameters of the `video_download` test are based on our (and our collaborator's) previous work [67], [68]. Figure 2 shows the operation of the `video_download` test. The test takes a YouTube URL as input and scrapes the fetched HTML page to extract the list of container formats, available resolutions and URL locations of media servers. The test then establishes two concurrent HTTP sessions (or one HTTP session over a QUIC connection with two streams) to fetch audio and video in the desired format and resolution. The test ensures temporal synchronization between the audio and video streams by de-muxing timestamps. The test does not render content at any time, but it only reads the container format to extract frame timestamps. The payload is subsequently discarded. Ahsan *et al.* [67] have shown that active measurement tests towards YouTube should run for a minimum of 1 minute (with a recommended value of 3 minutes). The `video_download` test runs for 3 minutes. We added QUIC support to the `video_download` test using the `lsquic` library (similar to the `quic_perf` test). The library is written in C similar to the `video` tests that makes integration simpler. The `video_download` test also uses `curl` for connection management when making connections over TLS and `lsquic` when making connections over QUIC. For instance, `lsquic` creates an HTTP session over a QUIC connection using one or more streams. It processes the incoming streams (performs packet reordering and decryption) over the QUIC connection and hands over the in-order byte stream to the `video_download` test for further processing using callback events.

`video_streaming` – We evaluate the performance of QUIC and TLS/TCP for adaptive YouTube video streaming by performing active measurements using the *VideoMon* framework [17]. *VideoMon* is an open-source containerized toolchain that allows researchers to headlessly stream user-defined YouTube videos and record key QoE metrics of the playtime. We follow a similar measurement design as `video_download` test, as we (i) randomly stream one of the trending videos in the vantage points' region and (ii) stream every video for 3 minutes. As a result, our `video_streaming` experiments provide a complementary (and more realistic) viewpoint to our `video_download` test as the video stream quality switches between different encodings to adapt to changes in the underlying network. Throughout our tests, we collect key QoE performance indicators, e.g., startup delay, duration and number of video stalls, buffer sizes and number (and resolutions) of quality switches for different loss settings. We use Linux system utility `tc` along with network emulator [69] to introduce losses in the network. The utility adds a queue discipline on the specified link, shaping the link-layer traffic based on the configuration. The rule affects both ingress and egress

1932-4537 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on December 11,2021 at 11:39:12 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3134562, IEEE Transactions on Network and Service Management

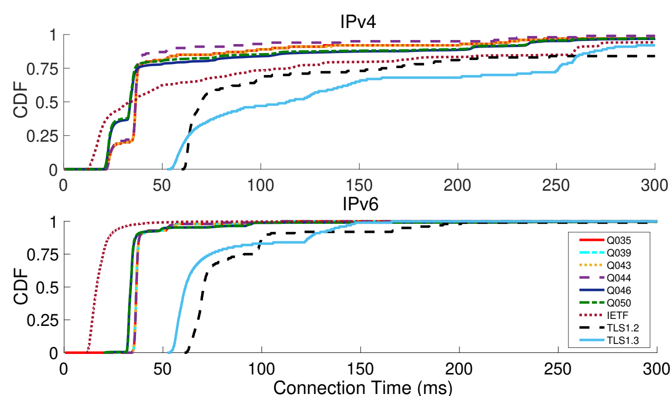


Fig. 3: *CDFs of connection times for QUIC and TCP/TLS versions over IPv4 and IPv6 from an LRZ Network, Germany. QUIC has lower connection times than TCP/TLS 1.2 and 1.3 for both IPv4 and IPv6.*

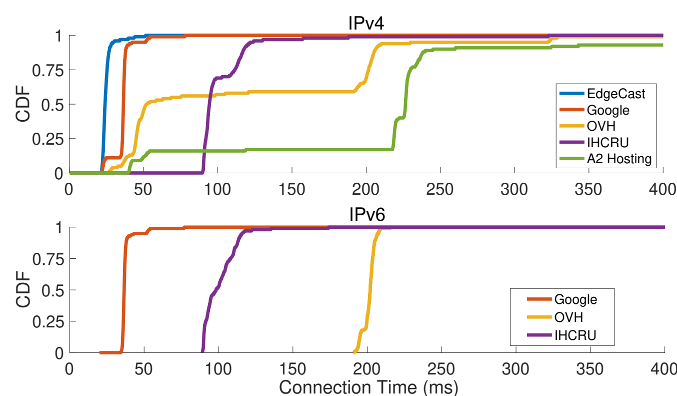


Fig. 4: *CDFs of connection times for all QUIC versions combined categorized by serving AS over IPv4 and IPv6 as observed from LRZ Network, Germany. EdgeCast performs better than Google CDN over IPv4. Connection time at 50th percentile (≈36 ms) closely match that of Google AS.*

traffic on the link. Together, our `video tests` provides a well-rounded performance comparison between QUIC and TCP/TLS when interacting with YouTube servers (see §VII).

V. Web Workloads

In this section we observe connection times, TTFB and download times for IETF QUIC (`ID-24`), gQUIC (`Q050`–`Q035`) (Table I) and TLS 1.2/1.3 over TCP for web workloads. For QUIC, we measure the `1-RTT` handshake, whereas for TLS 1.3/1.2 over TCP, we measure `2-RTT` and `3-RTT` respectively. We consider each measurement as a first-time connection to the websites since we do not reuse any previous session information. Since most of the websites we tested do not support `0-RTT` by default [2] [3], the performance analysis when connections are reused (QUIC `0-RTT` and TLS 1.3 `1-RTT`) is left as a consideration for future work. The analysis was performed using data collected by the VM and RPis (2018–2020) towards the target list (>5.7K) of websites out of Alexa Top-1M that supports QUIC (§IV). We also analyze IPv4 and IPv6 separately as all the websites in our target list do not support QUIC over IPv6 yet.

A. *Connection Times*

`tls_perf` uses `CURLINFO_APPCONNECT_TIME` (an option that can be retrieved by calling `curl_easy_getinfo()` on a `libcurl` handle) to measure the time from start until the TLS connect/handshake to the remote host is complete (Figure 1). It is to be noted, this time also contains the DNS lookup time, which may be cached on making repeated requests and would affect subsequent measurements. For this purpose, we subtract the value of `CURLINFO_NAMELOOKUP_TIME`, which is the time from start until name resolution is complete. For QUIC, we measure the time from the start of the handshake to the time of its completion. `lsquic` provides `quic_perf` with a callback `on_hsk_done` to capture when the handshake is complete and whether or not the connection attempt was successful. Similar to TLS/TCP, QUIC measurements also do not include the DNS lookup time in the connection establishment times.

Figure 3 shows the CDF of connection times for different versions of QUIC and TLS/TCP over IPv4 and IPv6 on a high bandwidth LRZ Network, Germany. It can be observed that QUIC has lower connection times than TLS 1.2 and TLS 1.3. IETF QUIC `ID-24` performs slightly better than other gQUIC versions. QUIC versions have similar connection times with a median value of ≈35 ms over IPv4. However, over IPv6, IETF QUIC has lower connection times than any other version by ≈50%. We also notice that `Q050` has slightly better performance than other gQUIC versions with ≈34 ms median connection time. The slight changes in connection times in different gQUIC versions may be attributed to functional changes in the protocol [70]. We plan to investigate the impact of such advances in the protocol on its overall performance in future works. Meanwhile, TLS 1.2 and TLS 1.3 have median connection times of ≈71 ms and ≈62 ms, respectively. Interestingly, we observe that TLS 1.2 performs at par (and sometimes even better) than TLS 1.3, even though theoretically it requires an additional RTT for connection establishment. While analyzing our data, we found that the websites with TLS 1.2 are slightly more widely deployed by CDNs than TLS 1.3, and hence its connection times are sometimes marginally better than TLS 1.3. We confirm this by analyzing the connection establishment times of the websites that support both TLS versions and TLS 1.3 consistently outperforms TLS 1.2. The step-wise distribution in Figure 3 is due to different serving ASes in our target websites. We discuss this aspect below.

Performance based on Serving AS – We compare the latency measurements at the AS level and associate each website to the destination AS (using the destination IP endpoint returned by both tests) and then select the top five ASes ordered by the number of websites with QUIC support for further

[2]https://cloud.google.com/blog/products/networking/tls-1-3-is-now-on-by-default-for-google-cloud-services

[3]https://blog.cloudflare.com/even-faster-connection-establishment-with-quic-0-rtt-resumption/

1932-4537 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on December 11,2021 at 11:39:12 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3134562, IEEE Transactions on Network and Service Management

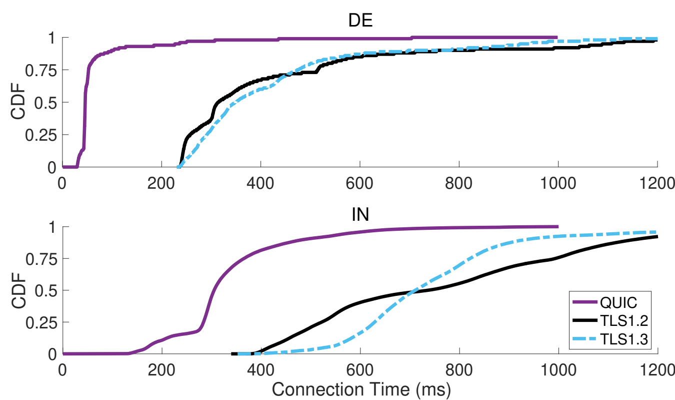


Fig. 5: *CDF of handshake times over QUIC and TLS as observed from RPi in DE and IN. QUIC is 445 ms (270 ms from DE) faster than TLS 1.2 and 409 ms (300 ms from DE) faster than TLS 1.3 from IN at the 50th percentile.*

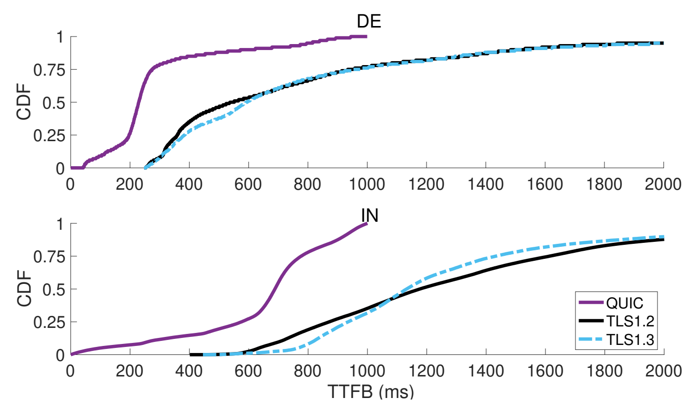


Fig. 6: *CDFs of time to first byte over QUIC and TLS as observed from RPi in DE and IN. QUIC shows greater improvement in low bandwidth, high RTT networks in IN. QUIC is 497 ms faster than TLS 1.2 and 448 ms faster than TLS 1.3 from IN at the 50th percentile.*

analysis. These five ASes cover 4887/5722 target websites resulting in a coverage of ≈85% with Google AS serving an overwhelming majority of 3863 websites. We analyzed the connection times of all QUIC versions (individually and combined) categorized by serving AS over IPv4 and IPv6. Figure 4 shows the CDFs of connection times over IPv4 and IPv6 for all QUIC versions combined split by destination AS as observed from LRZ Network, Germany. We can see that EdgeCast AS performs better than Google AS, and these two ASes perform much better than other ASes. EdgeCast performs ≈50% better than Google for half of the samples, while A2 Hosting exhibits the worst connection times. We also find that EdgeCast and A2 Hostings do not support QUIC over IPv6, while the connection times over IPv6 for Google, OVH and IHCRU are less than ≈36 ms, ≈98 ms and ≈201 ms respectively for half of the samples. Meanwhile, connection times for all samples combined at the 50th percentile is ≈36 ms, which closely matches that of Google AS. Furthermore, we investigate the performance of different QUIC versions served within the same AS. We find that IETF QUIC generally outperforms gQUIC within all ASes. However, we do not observe much difference between different gQUIC versions, especially within Google AS (plot not shown). We observe differences in connection establishment time across different ASes which likely attributes to the variations and the long tails in Figure 3. This analysis highlights that QUIC's performance in the wild is widely affected by the CDN that serves the content. As such, given Google AS has the largest number of websites with QUIC support, the overall distribution for the metrics largely mimics the performance as seen by Google AS.

While the VM measurements allowed us to benchmark the performance of the various QUIC versions in a well-provisioned network, measurements from the Raspberry Pi allows us to evaluate QUIC in a more conventional residential network from different vantage points. To this end, we have considered a 100 Mbps high-bandwidth, low-RTT residential link in Germany (DE) and a 20 Mbps low-bandwidth and high-RTT link in India (IN). Given the focus of this study to evaluate QUIC performance for different workloads in an uncontrolled setting and its comparison with traditional transports, henceforth we discuss the results from the RPi. From here on, we show the best performing QUIC version for increased clarity of results.

Figure 5 shows the CDF of connection times over QUIC and TLS as observed from RPi in DE and IN. In comparison to the results on the university VM (Figure 3), QUIC provides larger benefits over the uncontrolled residential links. In a low-bandwidth and high-RTT network (RPi IN), QUIC provides ≈175% and ≈125% improvement over TLS 1.2 and TLS 1.3 respectively at the 75th percentile. The large improvement is due to QUIC utilizing `1-RTT` connection time in comparison to `3-RTT` and `2-RTT` for TLS 1.2 and TLS 1.3 respectively in a high RTT and lossy network.

B. *Time to First Byte*

Time to First Byte (TTFB) is defined as the time taken from initiating the user's request to receiving the first byte of the object from the server (Figure 1). For TCP/TLS, it is the time between the start of TCP handshake and the arrival of an HTTP response, which includes (i) completion of TCP and TLS handshake, (ii) sending of an HTTP request, (iii) server processing the HTTP request and (iv) receiving the response. To measure TTFB for TCP/TLS, `tls_perf` uses `CURLINFO_STARTTRANSFER_TIME` provided by `libcurl`, which includes the pre-transfer commands and negotiations and the time it takes for the server to calculate the result. We remove the DNS lookup time as before. For QUIC, `quic_perf` uses the callback function `on_read` provided by `lsquic_stream` to process the bytes received after the handshake is complete. This callback is invoked for the first time when `quic_perf` receives the first byte of the response from the server. TTFB is measured from start to this time and is composed of the time taken for the QUIC handshake, sending the HTTP `GET` request, processing time at the server and receiving the response.

1932-4537 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on December 11,2021 at 11:39:12 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3134562, IEEE Transactions on Network and Service Management

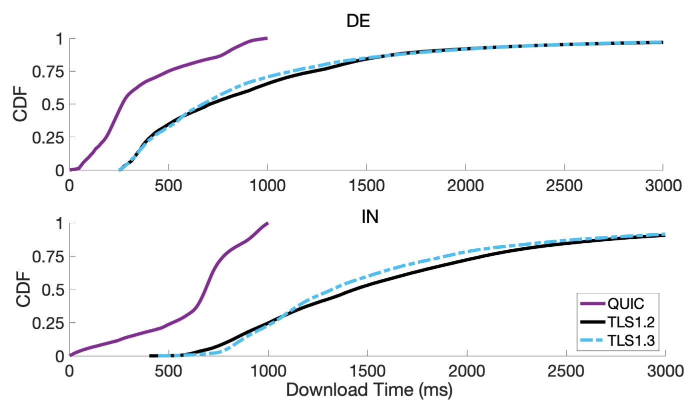


Fig. 7: *CDFs of download time over QUIC and TLS as observed from RPi in DE and IN. QUIC shows greater improvement in low bandwidth, high RTT networks in IN. QUIC is 760 ms faster than TLS 1.2 and 640 ms faster than TLS 1.3 from IN at the 50th percentile.*

Figure 6 shows the CDF of TTFB for QUIC and TCP/TLS from RPi in DE and IN. At 50th percentile TTFB measurements over QUIC conducted from RPi, DE are ≈230 ms. In contrast, for measurements conducted from RPi, India, TTFB at 50th percentile is almost 3× (≈690 ms), with ≈70% and ≈63% improvement over TCP/TLS 1.2 and 1.3 respectively. Figure 6 reveals a diminishing gap between QUIC and TLS/TCP (compared to Figure 5) indicating that QUIC tends to lose some of its improvements gained by `1-RTT` connection establishment time. We observe that the gain of `1-RTT` seems to have a smaller impact as the connection state proceeds.

C. *Total Download Time*

To calculate the total download time, the tests send an HTTP `GET` request to download the landing page (Figure 1). In the case of TLS/TCP, `tls_perf` uses option `CURLINFO_TOTAL_TIME` provided by `libcurl` which returns the time from start to when the response is complete (i.e., `FIN` is sent). We remove the DNS resolution time as before. For QUIC, `quic_perf` uses the callback on `_conn_closed`. It is invoked by `lsquic` when all data has been received and all the processes/connection with the server is closed.

Figure 7 shows the CDF of download time for QUIC and TCP/TLS from RPi in DE and IN. In a low-bandwidth and high-RTT network (RPi IN), QUIC provides ≈62.5% and ≈60% improvement over TLS 1.2 and TLS 1.3 respectively at 75th percentile with a download time of ≈780 ms. At 50th percentile, download times over QUIC from RPi, DE are ≈270 ms. In contrast, for measurements conducted from RPi, India, download times at 50th percentile are almost 2.6× (≈700 ms). Since we only download the website's landing page, the total number of downloaded bytes is low (with a median value of ≈ 40KB) and hence the shape of the CDF plot for total download time is similar to TTFB in Figure 6.

Takeaway: We find that QUIC has lower handshake times than TLS 1.2 and 1.3 over TCP for both IPv4 and IPv6 for Web workloads. IETF QUIC `ID-24` performs better than all the gQUIC versions we tested. Google AS serves 68% of the websites supporting QUIC. Consequently, QUIC's connection establishment times observed in our dataset closely mimic that of Google CDN. QUIC's lower connection times over TCP/TLS show higher benefits in a low-bandwidth high-RTT residential link in India when compared to a high-bandwidth low-RTT link in Germany. However, the gains by `1-RTT` connection times diminish for TTFB and download times.

VI. Cloud Storage Workloads

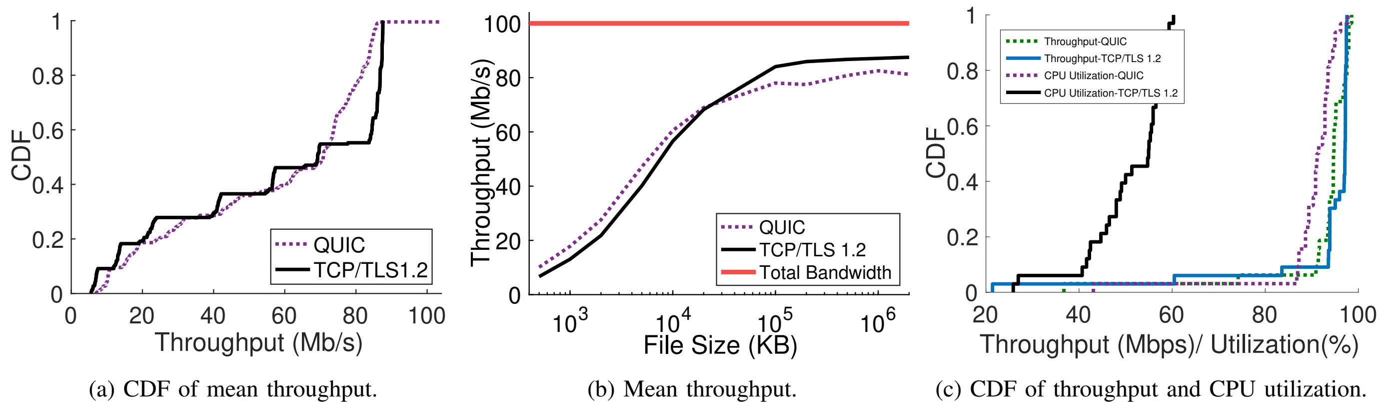
To evaluate the performance of QUIC in cloud storage workloads, we download files of different sizes varying from 1KB to 2GB from Google Drive using tests `quic_perf` and `tls_perf`. We observe throughput and CPU utilization when downloading files over QUIC and TLS/TCP. The results shared in this section are measured from the RPi over IPv4 in Germany and were repeated 20 times. Google Drive currently does not support TLS 1.3. As such, the performance comparison is made between QUIC and TLS 1.2.

A. *Throughput*

We measure attained throughput by dividing the file size by the total time it takes to download it. We upload files of different sizes (namely 1KB, 2KB, 5KB, 1MB, 2MB, 5MB, 10MB, 20MB, 50MB, 100MB, 200MB, 500MB, 1GB and 2GB) and download them with various QUIC versions and TLS 1.2. Usually, downloading large files from Google Drive prompts a warning screen that requires manual confirmation. We automate this step by submitting an HTTP `GET` request using `curl` and grabbing the cookie, followed by a second request appended with a *confirm* parameter in the URL. For greater granularity, we used `nettop` [71]. This tool intercepts all TCP/UDP packets using `libpcap` [72] and extracts IP and port information. In another thread it parses `/proc/[pid]/fd/` to identify the `inode` of the socket for a particular process, using this `inode` we can investigate the `/proc/net/tcp` and `/proc/net/udp` files. These files hold a dump of the TCP and UDP socket table, which contains information about the local address and port used by the socket. Using this, we assign each packet to a process and thus monitor the network used by each process. We sample the bandwidth usage every 500 ms. A benefit of `nettop` is that we control the timestamps at which the throughput can be measured to allow finer correlation with CPU utilization. Figure 8a shows CDF of mean throughput for QUIC and TCP/TLS `1.2` while downloading files of different sizes from Google Drive consecutively. We observe that QUIC achieves higher throughput than TCP for 55% of the measurements. For the rest, TCP is better. We observed similar throughput for different QUIC versions and hence do not show the differentiation in the plot for clarity. To dig deeper, we plot the mean throughput of QUIC and TCP for each file size (see Figure 8b). We observe that the mean throughput of TCP is higher

1932-4537 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on December 11,2021 at 11:39:12 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3134562, IEEE Transactions on Network and Service Management



(a) CDF of mean throughput.

(b) Mean throughput.

(c) CDF of throughput and CPU utilization.

Fig. 8: *(a) CDF of mean throughput and (b) mean throughput when downloading files of different sizes from Google Drive over QUIC and TCP/TLS, (c) CDF plot of throughput and CPU utilization for 200 MB file download. QUIC has higher throughput than TCP for smaller file sizes but the trend reverses for higher file sizes. For achieving similar throughputs, QUIC utilizes* $2\times$ *CPU resources compared to TCP.*

than QUIC for larger file sizes ($>$ 20 MB up to 2 GB), but for smaller file sizes ($\leq$ 20 MB), QUIC has higher throughput. This is because, for smaller file sizes, the connection times and TTFB dominate the total download times. QUIC's lower connection time leads to higher throughput for small file sizes. However, connection times constitute only a tiny fraction of the download time for large file sizes.

B. CPU Utilization

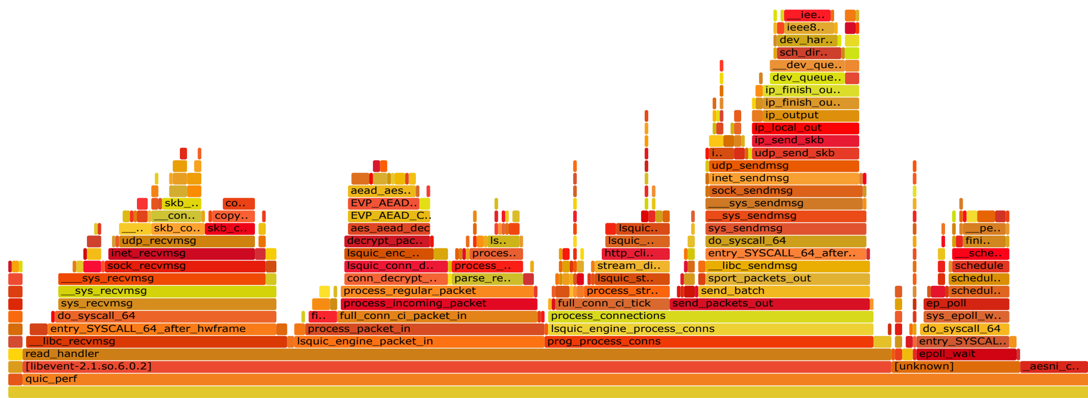
To measure the CPU utilization for a specific process we wrote a custom script `cpuutil` [73] which parses the data from `/proc/[PID]/stat` and `/proc/stat`. Assessing CPU usage from `/proc` is a widely used methodology in several recent research works [74], [75]. Our script records the number of jiffies executed by the process and the CPU, respectively. By sampling the same data again after a fixed interval (500 ms), we can calculate the process's CPU usage over the sampling time. `strace` is used to record the time spent in each system call, while the `perf` tool provides the CPU usage profile.

Figure 8c shows the CPU utilization (%) and throughput (Mbps) while downloading a 200 MB file from Google Drive. Please note that both the metrics are sharing the x-axis. It can be seen that QUIC utilizes twice as much CPU as TCP to achieve similar throughput values. CPU utilization for QUIC is $\approx$90% and remains consistent throughout the progress. TLS/TCP CPU utilization is $\approx$50%. The high CPU utilization of QUIC is because it is a user-space protocol built on top of UDP. QUIC transmission is limited to a maximum packet size of 1370 bytes for IPv4 and 1350 bytes for IPv6 to avoid packet fragmentation. On the other hand, TCP can utilize larger packet sizes than UDP. This means that UDP consumes higher resources for the same throughput in making expensive system calls to `sendmsg/recvmsg` operation. Thus, QUIC can achieve high performance only at the cost of high CPU utilization. We visualize the results as a flame graph in Figure 9, which allows us to identify the frequently used code paths. Rectangles on the X-axis corresponds to a stack frame, and the width of each rectangle represents its frequency in the stacks. The Y-axis shows the depth of the stack starting from 0 at the bottom. The top-level shows which system call is executed on the CPU. Figure 9a and Figure 9b shows the CPU utilization of `quic_perf` and `tls_perf` for same Google Drive file download. We observe that the width of the `__sys_recvmsg` and `__sys_sendmsg` for `quic_perf` is quite large, indicating that the bulk of CPU usage is in these paths. Also, we note that the stack for the `sendmsg` is quite tall, showing many function calls to send a UDP message. For our download using Google Drive, QUIC spends most of its time (76.89%) on the recvmsg system call (see: Table II). The Linux kernel supports Large Receive Offload (LRO)/ Large Segmentation Offload (LSO) for TCP, allowing incoming packets from a single stream to be aggregated first before packets are pushed up the networking stack. This greatly reduces the number of packets that have to be processed at the upper layers [76]. The Linux kernel (starting with v4.18) has recently added support for UDP generic receive offload (GRO)/ generic segmentation offload (GSO). Recent studies [77], [78] show that QUIC with generic segmentation offload (GSO) enabled can be computationally as efficient as TCP. However, Raspbian (used in this study) is yet to incorporate this kernel release. Therefore, QUIC (UDP) requires every packet to traverse up the stack. Note, QUIC is designed for serving traffic to clients, a large proportion of which may not always run a bleeding edge of the Linux kernel. As such, evaluating QUIC on kernels that do not yet support GRO/GSO is still useful for quantifying the resource utilization overhead of QUIC compared to TLS/TCP. A recent work [79] also measured the CPU usage breakdown of different implementations of QUIC for packet I/O, crypto, ACK, packet reordering and processing and their subsequent effects on performance degradation. The key takeaway of their results is that the majority share of QUIC's CPU overhead is due to packet I/O and crypto operations.

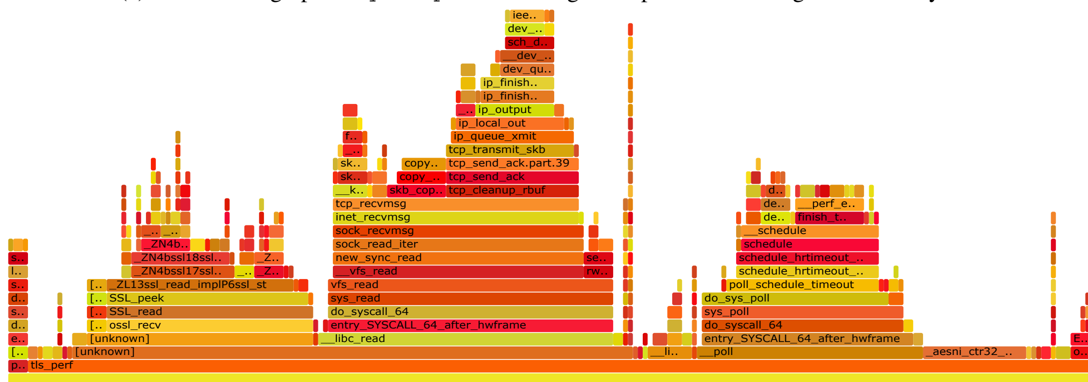
Takeaway: For cloud storage workloads, we observe that

1932-4537 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on December 11,2021 at 11:39:12 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3134562, IEEE Transactions on Network and Service Management



(a) CPU flame graph of `quic_perf` showing code paths consuming maximum cycles.



(b) CPU flame graph of `tls_perf` showing code paths consuming maximum cycles.

Fig. 9: *CPU usage of (a) QUIC and (b) TLS/TCP when downloading a large file from Google Drive. The width of the _sys_recvmsg and _sys_sendmsg is quite large for* `quic_perf` *indicating a higher CPU utilization in these paths.*

Index	%time	usecs/call	calls	time(s)	syscall
1	76.89	24	80800	1.95	recv_msg
2	17.33	201	2191	0.44	epoll_wait
3	4.79	57	2141	0.12	send_msg

TABLE II: *Top-3 system calls for QUIC while downloading file from Google drive from* `strace`*.*

the mean throughput of QUIC when downloading files from Google Drive is higher for small file sizes (≤20 MB) but for larger file sizes (>20 MB up to 2 GB), TCP's throughput is higher. It is because connection times and TTFB dominate the total download time for smaller file sizes. This gain diminishes as the file size increases. QUIC downloading a large file from Google Drive has high CPU usage because of the large number of `send/recv` calls.

VII. Video Workloads

We developed `bash` scripts to fetch the top 50 most popular videos over the day using the YouTube API [80] and ran the `video` tests toward the first video that succeeded. The API takes the regional diversity of the popularity into account and the video list is refreshed every day. In `video_download`, the test is performed once over TLS and subsequently over QUIC using multiple versions configurable on the command line. In situations where the vantage point is dual-stacked, the test runs over both IPv4 and IPv6. In `video_streaming`, the test subsequently streams the YouTube video over a headless Chrome browser using TLS/TCP, followed by QUIC. Each streaming experiment lasts for 3 minutes and we perform multiple runs throughout the day.

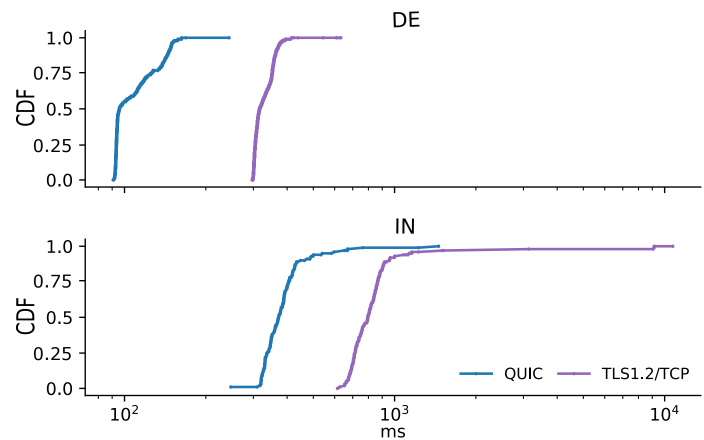


Fig. 10: *CDF of handshake times towards YouTube website over QUIC and TLS 1.2 from DE and IN. QUIC is 424 ms (221 ms from DE) faster than TLS from IN at the 50th percentile.*

We performed memory profiling using the `massif` (`valgrind`) heap analyzer to evaluate the dynamic total memory consumption of the `video` test (§VII-A3). We

1932-4537 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on December 11,2021 at 11:39:12 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3134562, IEEE Transactions on Network and Service Management

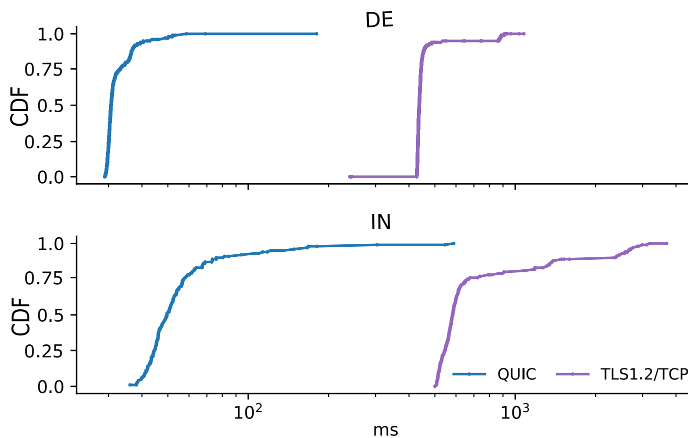


Fig. 11: *CDF of handshake times towards YouTube video servers over QUIC and TLS 1.2. QUIC is 534 ms (406 ms from DE) faster than TLS from IN at the 50th percentile.*

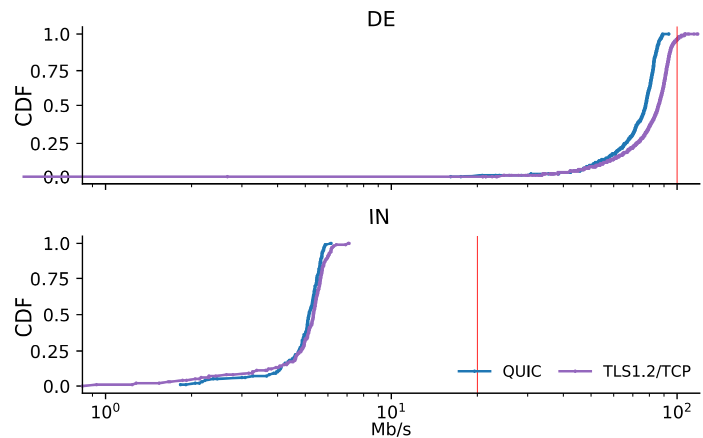


Fig. 12: *CDF of overall download rate from DE and IN. The vertical markers indicate the advertised data rate of the connection. QUIC in IN performs nearly similar to TLS.*

observed a peak memory usage of <5MB when downloading a 4K video for 1 minute, thereby allowing the `video_download` test to be deployed on a Raspberry Pi. We perform the `video_download` test from the same two vantage points to observe content delivery towards YouTube over QUIC from both developed (high-bandwidth, low-RTT) and developing (low-bandwidth, high-RTT) regions in a residential setting in Germany and India, respectively. Since these vantage points are single-stacked, our results only contain data on IPv4. YouTube currently does not support video streaming over TLS 1.3 [81], [82] therefore we only compare the performance between QUIC and TLS 1.2. Immediately after the `video_download` test completes, we perform `paris-traceroute` [83] using `scamper` [84] towards the destination IP endpoints identified by the test. The IP endpoints can either be media servers hosted within the CDN or caches deployed by the ISP. We used the `scamper` measurements to analyze the IP path lengths towards the media servers streaming the video. The routers on the path did not respond to UDP traceroutes when measured from India (IN). Meanwhile, the measurements from Germany (DE) revealed that it takes around 16 IP hops (3 AS hops) to reach the media servers hosted at Google (`AS15169`). We perform `video_streaming` tests on the university VM. To emulate more realistic environments, we also examine video streaming performance from YouTube servers with 1%, 5% and 10% network losses. We use `tc` to introduce losses in the network [69]. In the following analysis, we focus on the latency, throughput and QoE metrics of video delivery and YouTube streaming over QUIC and TLS/TCP.

A. `video_download` Performance

*1) **Connection Time**:* We define handshake time as the time to establish a TLS 1.2 session over TCP (`3-RTT`) (Figure 2) and the time to establish a QUIC connection (`1-RTT`). We did not measure `0-RTT` (with QUIC) and `2-RTT` (with TLS 1.2) for subsequent connections because we wanted to measure the performance characteristics by considering each connection as a new connection attempt.

Figure 10 shows the CDF of handshake times towards the YouTube website. QUIC is 424 ms (221 ms from DE) faster than TLS from IN in half of the samples. Figure 11 shows the CDF of handshake times toward media server destinations serving the YouTube video. These destinations can also include caches (whenever available) deployed inside the ISP. We witness that from IN, QUIC is 534 ms (406 ms from DE) faster in half of the samples than TLS and 2s (421 ms from DE) faster in 90% of the samples. These observations are similar to the handshake times of the web workload (§V) towards Google AS. When comparing between QUIC versions, we did not observe a difference in handshake times for half of the samples. At the 90th percentile, we observe that different versions of QUIC are 2 ms apart when observed from DE. While, `Q035`, the most widely deployed version of QUIC [24] can establish handshakes faster than newer versions of QUIC in 90% of the samples when observed from IN. This observation also holds true for connections made towards the YouTube website. For increased clarity of results, we show only the best performing QUIC version in the results. Overall, we witness that (besides DE) handshake times towards the media server are usually less than the webserver over QUIC and TLS. We suspect that this observation is due to ISP content caches that cache [85] popular videos closer to the user.

*2) **Throughput**:* Figure 12 shows the overall download rate, which includes the combined download rate of audio and video chunks. It can be seen that the overall download rate using TLS/TCP is higher compared with QUIC. When measured from IN, QUIC performs similar (3.1% difference) to TLS/TCP. While downloading files of different sizes (1KB to 2GB) from Google Drive (which also supports QUIC) (§VI-A), it was observed that for larger file sizes (20MB or above, such as the videos we measured), the achieved throughput over QUIC is lower than that of TLS/TCP. We suspect that QUIC sustains latency benefits over TLS/TCP, but tends to trade throughput with long flows as also witnessed in previous studies [42]. We suspect that this observation is partly due to kernel optimizations such as large receive offload (LRO) that are available for the TCP stack.

1932-4537 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on December 11,2021 at 11:39:12 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3134562, IEEE Transactions on Network and Service Management

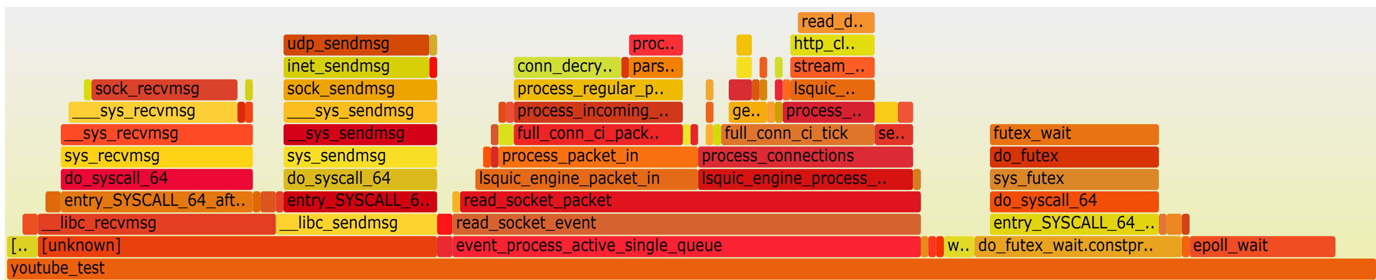


Fig. 13: *CPU sampling of the* `video_download` *test with* `perf`*. Rectangles on the X-axis corresponds to a stack frame and the width of the rectangle represents how often this function is present in the stack. We witness that* `_libc_recvmsg`*,* `_libc_sendmsg`*,* `packet_processing`*,* `futex_wait`*,* `epoll_wait` *represent 16.49%, 11.17%, 32.45%, 14.36%, 10.11% of the total sample respectively. The colors are used to distinguish frames.*

*3) **CPU Utilization:*** Figure 13 shows the CPU profile of the `video_download` test over QUIC performed using `perf`. Rectangles on the X-axis corresponds to a stack frame and the width of each rectangle represents its frequency in the stacks. The Y-axis shows the depth of the stack starting from 0 at the bottom. The top-level shows which system call is executed on the CPU. QUIC, as a user-space protocol, allows for faster changes and deployment, but it makes expensive system calls to send and receive UDP packets (`___sys_recvmsg` and `___sys_sendmsg`). Additionally, QUIC also suffers from a higher computational cost of ACK processing than TCP primarily due to two reasons. First, because ACK processing is done in user-space for QUIC (resulting in more data copies in user-kernel boundary and higher context switching), whereas TCP ACKs are handled in the kernel. Second, TCP ACK is plain text, whereas QUIC uses encrypted ACKs, further increasing the computational cost. Moreover, the kernel maintains the TCP connection state and reuses the state for all packets sent on the connection. As an example, the kernel applies a firewall rule at the start of the connection. However, the kernel has no connection state for QUIC connections; such kernel operations are performed for every QUIC packet [78]. As a result, most of the `video` processing time is currently taken for sending and receiving UDP packets, similar to observations when downloading large files from Google Drive (§VI-B) and QUIC packet processing (`packet_processing`), while the rest of the considerable workload is occupied by scheduling operations with `semaphores` and events. Lower packet processing overhead is seen in the CPU profile of the `video_download` test over TCP/TLS. We omit the flame graph of TCP/TLS due to space restrictions. In related work, [79] also discusses the CPU usage breakdown of QUIC for packet I/O, crypto, ACK, packet reordering and processing and their effects on the application performance.

B. `video_streaming` Performance

*1) **Startup Delay:*** Startup delay is the time measured from the video request to the start of the playback. This includes the DNS resolution and handshake times as it is an application layer metric. Figure 14 shows the CDF of the startup delay at various loss percentages over QUIC and TLS/TCP. As expected, increasing the loss increases the startup delay for both

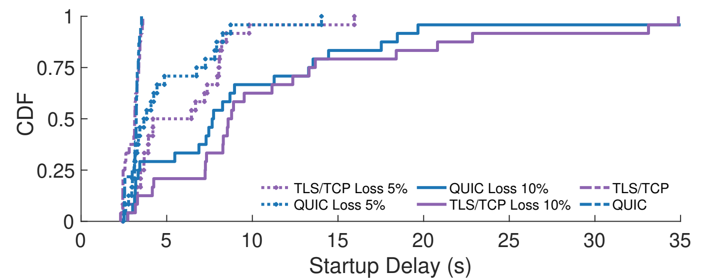


Fig. 14: *Distribution of startup delay over QUIC and TLS/TCP 1.2. QUIC experiences lower startup delays than TLS/TCP in lossy networks.*

protocols. The plot indicates that in non-lossy networks, both TLS/TCP and QUIC show similar startup delay performances. However, QUIC overtakes TLS/TCP by a large margin as soon as the network starts to experience packet losses. At 5% loss, QUIC is $\approx$ 40% faster than TLS/TCP at starting the YouTube video at the 50$^{\text{th}}$ percentile. For a loss of 10%, QUIC takes at most 7.65 ms to start video playback for half of the samples compared to 8.59 ms for TLS/TCP. The lower startup delay of QUIC in networks with packet losses can be attributed to the lower connection times and better loss recovery mechanisms compared to that of TLS/TCP [3].

*2) **Stall Rates and Stall Durations:*** A stall event is triggered during playback when a frame is not received before playout time. During stalls, video playback stops as the playback buffer drops to zero seconds. We observed higher stalls in networks with higher losses for both protocols (see: Table III). In situations where a stall does occur, we also measure the duration of the stall. Figure 15 shows the CDF of the stall durations for QUIC and TLS/TCP for different losses. We observe that at median, TLS/TCP exhibits upto 50% longer stall durations compared to QUIC at 10% losses. As such, even though higher throughput is achieved with TLS/TCP (Figure 12), QUIC depicts a better video streaming experience with reduced stalls and shorter stall durations. This behavior is even more pronounced in lossy networks. We regard the performance improvements of QUIC over TLS/TCP to its improved and fine-grained loss recovery mechanisms, which allows it to avoid retransmission ambiguities [3].

1932-4537 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on December 11,2021 at 11:39:12 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3134562, IEEE Transactions on Network and Service Management

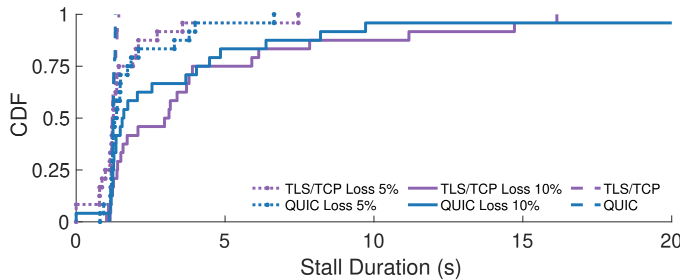


Fig. 15: *CDF of stall durations for QUIC and TLS/TCP. Adaptive video streaming over QUIC experiences lower stall durations than TLS/TCP, especially in loss-prone networks.*

Fig. 16: *Correlation between number of quality switches and stall duration (seconds). Y-axis is clipped to 15 s to improve readability.*

		No Loss	Loss 1%	Loss 5%	Loss 10%
#Stalls	QUIC	2.0	2.0	2.0	2.08
	TCP	2.0	2.0	1.96	2.2
#Switches	QUIC	1.0	1.0	1.0	2.58
	TCP	1.0	1.07	1.13	2.13

TABLE III: *Effect of losses on the average number of stalls and quality switches for QUIC and TLS/TCP.*

3) ***Quality Switches****:* Table III shows the quality switches for QUIC and TLS/TCP for different loss percentages. As expected, with increasing packet loss, we experience more stalls and more quality switches. The number of quality switches is higher for QUIC in comparison to TCP for a high loss network. In networks experiencing zero/low packet losses, both QUIC and TCP use the highest quality of 720p and a lowest of 480p. For higher losses (5-10%), the lowest quality drops to 144p for both protocols. To understand the effect of the number of quality switches on the QoE, we map the number of quality switches to corresponding stall durations in Figure 16. Note the clustering of data points in the low stall duration and the low number of quality switches. In zero/low packet losses, YouTube's adaptive video streaming mechanism maintains the optimal user QoE by upscaling the video to the highest possible resolution and keeping stalls as low as possible. However, as the packet losses in the network increase, the client experiences many more quality switches, irrespective of the transport protocol, adapting to network changes. In this case, streaming over QUIC results in better QoE than TLS/TCP since the average stall durations are much lower (see Figure 15). This is likely an effect of QUIC's superior loss recovery mechanism compared to TCP as it adopts and builds on approaches like F-RTO, early retransmit, TCP-RACK algorithm, etc. [3].

Takeaway: We observe that QUIC provides significant improvements over TLS/TCP in low-bandwidth and high-RTT regions for video downloads. QUIC handshakes towards YouTube media servers offer an improvement of 534 ms (IN) and 406 ms (DE) when compared with TLS/TCP. We also observe that the overall download rate for TLS/TCP is higher than QUIC partly due to kernel optimizations such as LRO available for the TCP stack. QUIC provides a better video streaming experience with a lesser number and duration of stall events compared to TLS/TCP. We observe that TLS/TCP exhibits up to 50% longer stall durations compared to QUIC at 50th percentile for high loss networks.

VIII. Implications for Network Management

The conventional methods used by ISPs to measure performance by passively monitoring the network will not work with QUIC anymore since the flows are end-to-end encrypted. Although there are preliminary studies on how to passively measure the performance of encrypted QUIC flows, a stable ground truth is still needed for validation, which the paper provides by measuring QUIC performance of different workloads from the endpoint of the client (independently) toward many content delivery networks (CDNs) that currently support QUIC. We believe that this study is of prime relevance to three parties. Firstly, our analysis can be utilized by network and content providers who are waiting to adopt QUIC and would want to understand its behavior in different network conditions. Secondly, this study can be of interest to the IETF as QUIC has been very recently standardized [2] and its performance behavior is of significant interest to the community. Our study helps provide an empirical grounding on the performance of QUIC in the real world. Finally, our work is of importance for the networking research community who would value the tools and datasets we open-source and make publicly available to further extend the understanding of this emerging protocol over the Internet.

IX. Limitations and Future Work

In this work, we evaluated gQUIC (`Q035-Q050`) and IETF QUIC (`ID-24`). We plan to continue the development of the tests to track both minor and significant changes in QUIC in the future. Observations on QUIC presented in this study are a function of the `lsquic` implementation, which the tests use underneath. The different implementations of QUIC can lead to different results. There has been some preliminary work understanding the behavior of different implementations in this space, and we refer the inclined reader to [86]. The IETF is discussing the need for a standardized QUIC API that would enable applications to plug different implementations. This will enable repeating the experiments with the tests across implementations in the future.

1932-4537 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on December 11,2021 at 11:39:12 UTC from IEEE Xplore. Restrictions apply.

The measurements are performed from a limited sample of (diverse) vantage points. However, the goal of this work is not to sample a large number of clients, but to sample a large number of destinations and different application workloads covering short-, long- flows, resource utilization. Our aim is to extend our measurements to large-scale measurement platforms such as SamKnows and CAIDA Ark. Such platforms utilize heavily constrained hardware probes that are deployed at users' homes. We have confirmed that our tests are possible to deploy on such environments.

X. Conclusion

We measured and analyzed different QUIC versions over both short flows and long flows to mimic the web, cloud storage and video workloads. We developed specific tests `quic_perf`, `tls_perf` and `video` for such evaluation. Using these tests, we conducted our measurements from different vantage points that allowed us to compare the benefits of using QUIC in a lossy residential versus a well-provisioned university network. We observed that QUIC performs well for short flows (such as small file downloads, browsing websites) as the total download time is dominated by connection time. However, since QUIC is built on the top of UDP, it exhibits high CPU utilization due to a high number of `send/recv` function calls that reduce the throughput over long flows (such as large file downloads, streaming YouTube videos). QUIC handshakes towards YouTube media servers provide an improvement of ≈90% for both IN and DE when compared with TLS. This reduces the startup delay over QUIC by 1s for high loss (10%) networks. Even though the download rate is lower over QUIC, the lower latency leads to better video content delivery due to reduced stall rates and stall durations when streaming videos.

Acknowledgements – We thank Dmitri Tikhonov who actively supported integration of `lsquic` with `quic_perf` and `video` tests. We also thank ChengLun Li, Aakash Kamble and Bernhard Jaeger for early exploration of this idea.

References

[1] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. R. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W. Chang, and Z. Shi, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," in *SIGCOMM*. ACM, 2017. [Online]. Available: https://doi.org/10.1145/3098822.3098842

[2] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021. [Online]. Available: https://rfc-editor.org/rfc/rfc9000.txt

[3] J. Iyengar and I. Swett, "QUIC Loss Detection and Congestion Control," RFC 9002, May 2021. [Online]. Available: https://rfc-editor.org/rfc/rfc9002.txt

[4] C. B. Krasic, M. Bishop, and A. Frindell, "QPACK: Header Compression for HTTP over QUIC," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-qpack-21, Feb. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-qpack-21

[5] M. Thomson and S. Turner, "Using TLS to Secure QUIC," RFC 9001, May 2021. [Online]. Available: https://rfc-editor.org/rfc/rfc9001.txt

[6] Y. Cui, T. Li, C. Liu, X. Wang, and M. Kühlewind, "Innovating Transport with QUIC: Design Approaches and Research Challenges," *IEEE Internet Computing*, vol. 21, no. 2, pp. 72–76, 2017. [Online]. Available: https://doi.org/10.1109/MIC.2017.44

[7] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How Speedy is SPDY?" in *USENIX NSDI*, 2014. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/wang

[8] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540, pp. 1–96, May 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7540.txt

[9] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?" in *IMC*, 2011, pp. 181–194. [Online]. Available: https://doi.org/10.1145/2068816.2068834

[10] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain, "TCP Fast Open," *RFC 7413*, pp. 1–26, 2014. [Online]. Available: https://doi.org/10.17487/RFC7413

[11] R. R. Stewart, "Stream Control Transmission Protocol," *RFC 4960*, pp. 1–152, 2007. [Online]. Available: https://doi.org/10.17487/RFC4960

[12] A. Ford, C. Raiciu, M. J. Handley, O. Bonaventure, and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 8684, Mar. 2020. [Online]. Available: https://rfc-editor.org/rfc/rfc8684.txt

[13] QUIC Usage Statistics. [Online]. Available: https://w3techs.com/technologies/details/ce-quic

[14] M. Piraux, Q. D. Coninck, and O. Bonaventure, "Observing the Evolution of QUIC Implementations," ser. CONEXT EPIQ Workshop, 2018. [Online]. Available: https://doi.org/10.1145/3284850.3284852

[15] `libcurl`. [Online]. Available: https://curl.haxx.se/libcurl/

[16] LiteSpeed Technologies. (2021) Github repository. `lsquic`. [Online]. Available: https://github.com/litespeedtech/lsquic

[17] A. Schwind, C. Midoglu, Ö. Alay, C. Griwodz, and F. Wamser, "Dissecting the performance of YouTube video streaming in mobile networks," p. e2058, 2020. [Online]. Available: https://doi.org/10.1002/nem.2058

[18] V. Bajpai, A. Brunström, A. Feldmann, W. Kellerer, A. Pras, H. Schulzrinne, G. Smaragdakis, M. Wählisch, and K. Wehrle, "The Dagstuhl Beginners Guide to Reproducibility for Experimental Networking Research," ser. SIGCOMM CCR, vol. 49, no. 1, 2019. [Online]. Available: https://doi.org/10.1145/3314212.3314217

[19] Github repository. Reproducibility Dataset and Scripts. [Online]. Available: https://github.com/tanyashreedhar/tnsm-2021-quic

[20] Y. El-khatib, G. Tyson, and M. Welzl, "Can SPDY really make the web faster?" ser. IFIP Networking Conference, 2014, pp. 1–9. [Online]. Available: https://doi.org/10.1109/IFIPNetworking.2014.6857089

[21] R. Corbel, E. Stephan, and N. Omnès, "HTTP/1.1 pipelining vs HTTP2 in-the-clear: Performance comparison," ser. IEEE NOTERE, 2016. [Online]. Available: https://doi.org/10.1109/NOTERE.2016.7745823

[22] F. Aschenbrenner, T. Shreedhar, O. Gasser, N. Mohan, and J. Ott, "From single lane to highways: Analyzing the adoption of multipath tcp in the internet," in *2021 IFIP Networking Conference (IFIP Networking)*, 2021, pp. 1–9. [Online]. Available: https://doi.org/10.23919/IFIPNetworking52078.2021.9472785

[23] T. Shreedhar, N. Mohan, S. K. Kaul, and J. Kangasharju, "Qaware: A cross-layer approach to mptcp scheduling," in *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, 2018, pp. 1–9. [Online]. Available: https://doi.org/10.23919/IFIPNetworking.2018.8696843

[24] J. Rüth, I. Poese, C. Dietzel, and O. Hohlfeld, "A First Look at QUIC in the Wild," ser. PAM Conference, vol. 10771, 2018, pp. 255–268. [Online]. Available: https://doi.org/10.1007/978-3-319-76481-8_19

[25] T. Shreedhar, S. K. Kaul, and R. D. Yates, "An age control transport protocol for delivering fresh updates in the internet-of-things," in *2019 IEEE 20th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, 2019, pp. 1–7. [Online]. Available: https://doi.org/10.1109/WoWMoM.2019.8793011

[26] M. Kosek, T. Shreedhar, and V. Bajpai, "Beyond QUIC v1: A First Look at Recent Transport Layer IETF Standardization Efforts," *IEEE Communications Magazine*, vol. 59, no. 4, pp. 24–29, 2021. [Online]. Available: https://doi.org/10.1109/MCOM.001.2000877

[27] S. Floyd, J. Mahdavi, M. Mathis, and D. A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, Oct. 1996. [Online]. Available: https://rfc-editor.org/rfc/rfc2018.txt

[28] E. Blanton, D. V. Paxson, and M. Allman, "TCP Congestion Control," RFC 5681, Sep. 2009. [Online]. Available: https://rfc-editor.org/rfc/rfc5681.txt

[29] J. Blanton, P. Hurtig, U. Ayesta, K. Avrachenkov, and M. Allman, "Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP)," RFC 5827, Apr. 2010. [Online]. Available: https://rfc-editor.org/rfc/rfc5827.txt

[30] Y. Cheng, N. Cardwell, N. Dukkipati, and P. Jha, "The RACK-TLP Loss Detection Algorithm for TCP," RFC 8985, Feb. 2021. [Online]. Available: https://rfc-editor.org/rfc/rfc8985.txt

[31] A. Gurtov, T. Henderson, S. Floyd, and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 6582, Apr. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6582.txt

1932-4537 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on December 11,2021 at 11:39:12 UTC from IEEE Xplore. Restrictions apply.

[32] M. Bishop, "Hypertext Transfer Protocol Version 3 (HTTP/3)," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-http-18, Jan. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-18

[33] R. Peon and H. Ruellan, "HPACK: Header Compression for HTTP/2," RFC 7541, May 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7541.txt

[34] D. Stenberg, "Comparison with HTTP/2," 2019, https://http3-explained.haxx.se/en/h3/h3-h2.

[35] M. Trevisan, D. Giordano, I. Drago, and A. S. Khatouni, "Measuring HTTP/3: Adoption and Performance," *CoRR*, vol. abs/2102.12358, 2021. [Online]. Available: https://arxiv.org/abs/2102.12358

[36] P. Biswal and O. Gnawali, "Does QUIC Make the Web Faster?" in *IEEE Global Communications Conference*, ser. IEEE GLOBECOM, 2016, pp. 1–6. [Online]. Available: https://doi.org/10.1109/GLOCOM.2016.7841749

[37] G. Carlucci, L. D. Cicco, and S. Mascolo, "HTTP over UDP: an experimental investigation of QUIC," in *ACM Symposium on Applied Computing*, ser. ACM Symposium on Applied Computing. ACM, 2015, pp. 609–614. [Online]. Available: https://doi.org/10.1145/2695664.2695706

[38] P. Megyesi, Z. Kramer, and S. Molnár, "How quick is QUIC?" in *ICC*, ser. ICC, 2016, pp. 1–6. [Online]. Available: https://doi.org/10.1109/ICC.2016.7510788

[39] S. Cook, B. Mathieu, P. Truong, and I. Hamchaoui, "QUIC: Better for what and for whom?" ser. IEEE ICC, 2017, pp. 1–6. [Online]. Available: https://doi.org/10.1109/ICC.2017.7997281

[40] K. Nepomuceno, I. N. d. Oliveira, R. R. Aschoff, D. Bezerra, M. S. Ito, W. Melo, D. Sadok, and G. Szabó, "QUIC and TCP: A Performance Evaluation," ser. ISCC, Jun. 2018. [Online]. Available: https://dx.doi.org/10.1109/ISCC.2018.8538687

[41] F. Mütsch, "Caddy - a modern web server (vs. nginx)," 2017, https://hackernoon.com/caddy-a-modern-web-server-vs-nginx-e9e4abc443e.

[42] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, "Taking a Long Look at QUIC: An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols," in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC '17. New York, NY, USA: ACM, 2017, p. 290–303. [Online]. Available: https://doi.org/10.1145/3131365.3131368

[43] Y. Yu, M. Xu, and Y. Yang, "When QUIC meets TCP: An experimental study," ser. IEEE IPCCC, 2017, pp. 1–8. [Online]. Available: https://doi.org/10.1109/PCCC.2017.8280429

[44] P. Wang, C. Bianco, J. Riihijärvi, and M. Petrova, "Implementation and Performance Evaluation of the QUIC Protocol in Linux Kernel," ser. ACM MSWiM, 2018, pp. 227–234. [Online]. Available: https://doi.org/10.1145/3242102.3242106

[45] K. Wolsing, J. Rüth, K. Wehrle, and O. Hohlfeld, "A Performance Perspective on Web Optimized Protocol Stacks: TCP+TLS+HTTP/2 vs. QUIC," ser. IRTF ANRW Workshop, 2019, p. 1–7. [Online]. Available: https://doi.org/10.1145/3340301.3341123

[46] M. H. Mazhar and Z. Shafiq, "Real-time Video Quality of Experience Monitoring for HTTPS and QUIC," ser. IEEE INFOCOM, 2018. [Online]. Available: https://doi.org/10.1109/INFOCOM.2018.8486321

[47] Q. D. Coninck and O. Bonaventure, "Multipath QUIC: Design and Evaluation," in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2017, Incheon, Republic of Korea, December 12 - 15, 2017*, ser. CoNEXT. ACM, 2017, pp. 160–166. [Online]. Available: https://doi.org/10.1145/3143361.3143370

[48] H. Wu, Ö. Alay, A. Brunstrom, S. Ferlin, and G. Caso, "Peekaboo: Learning-based Multipath Scheduling for Dynamic Heterogeneous Environments," ser. IEEE JSAC, 2020. [Online]. Available: https://doi.org/10.1109/JSAC.2020.3000365

[49] D. Madariaga, L. Torrealba, J. Madariaga, J. Bermúdez, and J. Bustos-Jiménez, "Analyzing the Adoption of QUIC From a Mobile Development Perspective," ser. SIGCOMM EPIQ Workshop, 2020, p. 35–41. [Online]. Available: https://doi.org/10.1145/3405796.3405830

[50] C. Perkins and J. Ott, "Real-time Audio-Visual Media Transport over QUIC," in *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC, EPIQ@CoNEXT 2018, Heraklion, Greece, December 4, 2018*, ser. CONEXT EPIQ Workshop. ACM, 2018, pp. 36–42. [Online]. Available: https://doi.org/10.1145/3284850.3284856

[51] L. Eggert, "Towards Securing the Internet of Things with QUIC," ser. NDSS Symposium, 2020. [Online]. Available: https://easychair.org/publications/preprint/68D2

[52] P. D. Vaere, T. Bühler, M. Kühlewind, and B. Trammell, "Three Bits Suffice: Explicit Support for Passive Measurement of Internet Latency in QUIC and TCP," ser. IMC, 2018, pp. 22–28. [Online]. Available: https://dl.acm.org/citation.cfm?id=3278535

[53] M. Palmer, T. Krüger, B. Chandrasekaran, and A. Feldmann, "The QUIC Fix for Optimal Video Streaming," in *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC, EPIQ@CoNEXT 2018, Heraklion, Greece, December 4, 2018*, ser. CONEXT EPIQ Workshop. New York, NY, USA: ACM, 2018, pp. 43–49. [Online]. Available: https://doi.org/10.1145/3284850.3284857

[54] T. Zinner, S. Geissler, F. Helmschrott, and V. Burger, "Comparison of the initial delay for video playout start for different HTTP-based transport protocols," ser. IFIP/IEEE IM, 2017, pp. 1027–1030. [Online]. Available: https://doi.org/10.23919/INM.2017.7987428

[55] D. Bhat, A. Rizk, and M. Zink, "Not so QUIC: A Performance Study of DASH over QUIC," in *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV 2017, Taipei, Taiwan, June 20-23, 2017*, ser. NOSSDAV Workshop. New York, NY, USA: ACM, 2017, pp. 13–18. [Online]. Available: https://doi.org/10.1145/3083165.3083175

[56] D. Bhat, R. Deshmukh, and M. Zink, "Improving QoE of ABR Streaming Sessions through QUIC Retransmissions," in *Proceedings of the 26th ACM International Conference on Multimedia*, ser. MM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1616–1624. [Online]. Available: https://doi.org/10.1145/3240508.3240664

[57] M. Seufert, R. Schatz, N. Wehner, B. Gardlo, and P. Casas, "Is QUIC becoming the New TCP? On the Potential Impact of a New Protocol on Networked Multimedia QoE," ser. QoMEX Conference, 2019, pp. 1–6. [Online]. Available: https://doi.org/10.1109/QoMEX.2019.8743223

[58] M. Seufert, R. Schatz, N. Wehner, and P. Casas, "QUICker or not? -an Empirical Analysis of QUIC vs TCP for Video Streaming QoE Provisioning," ser. ICIN Conference, 2019, pp. 7–12. [Online]. Available: https://doi.org/10.1109/ICIN.2019.8685913

[59] I. Ayad, Y. Im, E. Keller, and S. Ha, "A Practical Evaluation of Rate Adaptation Algorithms in HTTP-based Adaptive Streaming," *Computer Networks*, vol. 133, pp. 90–103, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128618300288

[60] M. Rajiullah, A. Lutu, A. S. Khatouni, M.-R. Fida, M. Mellia, A. Brunstrom, O. Alay, S. Alfredsson, and V. Mancuso, "Web Experience in Mobile Networks: Lessons from Two Million Page Visits," ser. WWW Conference, 2019, p. 1532–1543. [Online]. Available: https://doi.org/10.1145/3308558.3313606

[61] J. Rüth, K. Wolsing, K. Wehrle, and O. Hohlfeld, "Perceiving QUIC: Do Users Notice or Even Care?" ser. CoNEXT, 2019, p. 144–150. [Online]. Available: https://doi.org/10.1145/3359989.3365416

[62] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-transport-24, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-24

[63] QUIC Research@COMSYS. [Online]. Available: https://quic.comsys.rwth-aachen.de/data.php

[64] M. Kühlewind and B. Trammell, "Applicability of the QUIC Transport Protocol," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-applicability-13, Sep. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-applicability-13

[65] IETF QUIC Implementations. [Online]. Available: https://github.com/quicwg/base-drafts/wiki/Implementations

[66] C. Benfield, "HPACK Documentation," 2020, https://readthedocs.org/projects/hpack/downloads/pdf/latest/.

[67] S. Ahsan, V. Singh, and J. Ott, "Impact of Duration on Active Video Testing," ser. NOSSDAV Workshop, 2016, pp. 9:1–9:6. [Online]. Available: https://doi.org/10.1145/2910642.2910651

[68] V. Bajpai, S. Ahsan, J. Schönwälder, and J. Ott, "Measuring YouTube Content Delivery over IPv6," *Computer Communication Review*, vol. 47, no. 5, pp. 2–11, 2017. [Online]. Available: https://doi.org/10.1145/3155055.3155057

[69] Linux Manual Page, "tc-netem," https://man7.org/linux/man-pages/man8/tc-netem.8.html.

[70] "QUIC Version Description," https://source.chromium.org/chromium/chromium/src/+/master:net/third_party/quiche/src/quic/core/quic_versions.h?originalUrl=https:%2F%2Fcs.chromium.org%2Fchromium%2Fsrc%2Fnet%2Fthird_party%2Fquiche%2Fsrc%2Fquic%2Fcore%2Fquic_versions.h, 2018, last Accessed: 2021-11-30.

[71] Github repository. `nettop`. [Online]. Available: https://github.com/RohitPanda/nettop

1932-4537 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on December 11,2021 at 11:39:12 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3134562, IEEE Transactions on Network and Service Management

[72] Github repository. libpcap. [Online]. Available: https://github.com/the-tcpdump-group/libpcap
[73] Github repository. CPU Utilization. [Online]. Available: https://github.com/RohitPanda/CPU_utilization/blob/master/main.c
[74] X. Chen, N. Ding, A. Jindal, Y. C. Hu, M. Gupta, and R. Vannithamby, "Smartphone energy drain in the wild: Analysis and Implications," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 1, pp. 151–164, 2015. [Online]. Available: https://doi.org/10.1145/2796314.2745875
[75] F. Li, Y. Shi, A. Shinde, J. Ye, and W. Song, "Enhanced cyber-physical security in internet of things through energy auditing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5224–5231, 2019. [Online]. Available: https://doi.org/10.1109/JIOT.2019.2899492
[76] A. Menon and W. Zwaenepoel, "Optimizing TCP Receive Performance," ser. USENIX ATC, 2008, pp. 85–98. [Online]. Available: http://www.usenix.org/events/usenix08/tech/fullpapers/menon/menon.pdf
[77] W. de Bruijn and E. Dumazet, "Optimizing UDP for content delivery: GSO, pacing and zerocopy," in *Linux Plumbers Conference*, 2018.
[78] K. Oku and J. Iyengar, "Can QUIC match TCP's computational efficiency?" 2020, https://www.fastly.com/blog/measuring-quic-vs-tcp-computational-efficiency.
[79] X. Yang, L. Eggert, J. Ott, S. Uhlig, Z. Sun, and G. Antichi, "Making QUIC Quicker With NIC Offload," in *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, ser. EPIQ '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 21–27. [Online]. Available: https://doi.org/10.1145/3405796.3405827
[80] Github repository. youtube test. [Online]. Available: https://github.com/RohitPanda/Quic-Test
[81] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," *RFC 8446*, pp. 1–160, 2018. [Online]. Available: https://doi.org/10.17487/RFC8446
[82] R. Holz, J. Hiller, J. Amann, A. Razaghpanah, T. Jost, N. Vallina-Rodriguez, and O. Hohlfeld, "Tracking the Deployment of TLS 1.3 on the Web: A Story of Experimentation and Centralization," *SIGCOMM Comput. Commun. Rev.*, vol. 50, no. 3, p. 3–15, jul 2020. [Online]. Available: https://doi.org/10.1145/3411740.3411742
[83] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira, "Avoiding Traceroute Anomalies with Paris Traceroute," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 153–158. [Online]. Available: https://doi.org/10.1145/1177080.1177100
[84] M. J. Luckie, "Scamper: A Scalable and Extensible Packet Prober for Active Measurement of the Internet," ser. IMC, 2010, pp. 239–245. [Online]. Available: https://doi.org/10.1145/1879141.1879171
[85] T. V. Doan, L. Pajevic, V. Bajpai, and J. Ott, "Tracing the Path to YouTube: A Quantification of Path Lengths and Latencies Toward Content Caches," ser. IEEE Communications Magazine, 2019. [Online]. Available: https://doi.org/10.1109/MCOM.2018.1800132
[86] R. Marx, J. Herbots, W. Lamotte, and P. Quax, "Same Standards, Different Decisions: A Study of QUIC and HTTP/3 Implementation Diversity," in *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, ser. EPIQ '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 14–20. [Online]. Available: https://doi.org/10.1145/3405796.3405828

Tanya Shreedhar is a doctoral candidate at IIIT-Delhi, India. She received her B.E. with a University Gold Medal from Panjab University, Chandigarh. She was a visiting researcher at TUM, Germany (2020). She has been awarded several competitive fellowships, including TCS Research Scholarship and Overseas Research Fellowship. Her research interests lie broadly in the area of next-generation transport protocols, specifically but not limited to MPTCP, QUIC and Age Control Protocol.

Rohit Panda received his Masters in Infomatics (2019) from TUM, Germany and Bachelors degree in Computer Science (2011) from Kalinga Institute of Industrial Technology, Bhubaneswar. He is an experienced software engineer and is interested in working on high performance and low latency systems. He is currently working as a software developer at SOFHA GmbH.

Sergey Podanev received his Masters in Infomatics (2019) at TUM, Germany and Networks and Telecommunications (2018) at TPU, Russia. He is interested in distributed computing techniques including cloud technologies. He is currently working as a software engineer with the focus on microservices at Kafka processing.

Vaibhav Bajpai is a senior researcher at TUM, Germany. He received his PhD (2016) and Masters (2012) degrees in Computer Science from Jacobs University Bremen, Germany. He is the recipient of the best of CCR award (2019), ACM SIGCOMM best paper award (2018), and IEEE COMSOC award (2017) for the best dissertation in network and service management. His current research focuses on the performance and management of next-generation networked systems.

1932-4537 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on December 11,2021 at 11:39:12 UTC from IEEE Xplore. Restrictions apply.