Exploring Proxying QUIC and HTTP/3 for Satellite Communication

Mike Kosek*, Hendrik Cech*, Vaibhav Bajpai†, Jörg Ott*
*Technical University of Munich, Germany
[kosek | cech | ott]@in.tum.de
†CISPA Helmholtz Center for Information Security
bajpai@cispa.de

***Abstract*—Low-Earth Orbit satellites have gained momentum to provide Internet connectivity, augmenting those in the long-established geostationary orbits. At the same time, QUIC has been developed as the new transport protocol for the web. While QUIC traffic is fully encrypted, intermediaries such as performance enhancing proxies (PEPs) – in the past essential for Internet over satellite performance – can no longer tamper with and optimize transport connections. In this paper, we present a satellite emulation testbed and use it to compare QUIC and TCP as well as HTTP/3 and HTTP/1.1 with and without minimal PEP functionality. Evaluating *goodput* over time, we find that the slow start threshold is reached up to 2s faster for QUIC PEP in comparison to QUIC Non-PEP. Moreover, we find that HTTP/3 and HTTP/3-PEP outperform HTTP/1.1 and HTTP/1.1-PEP in multiple *web performance* scenarios, where HTTP/3-PEP improves over HTTP/3 for *Page Load Time* by over 7s in edge cases. Hence, our findings hint that these performance gains may warrant exploring PEPs for QUIC.**

I. Introduction

Internet access via satellite has experienced a revival with the advent of Starlink by SpaceX [1], Kuiper by Amazon [2], Oneweb [3], and Telesat [4], which leverage massive satellite constellations in low earth orbits (LEO) at 300–2000 km altitude. This is contrast to early services, such as Hughes DirecTV and Viasat Connexion for aircraft, among others, that mostly used geostationary (GEO) satellites at an altitude of 35,785km to provide Internet connectivity in the late 1990s and (early) 2000s, with Iridium [5] as one notable exception that has also relied on LEO satellites.

Using GEO satellites, such early Internet access services have experienced a one-way delay of around 250ms for the satellite hop. This called for introducing intermediaries, often dubbed *performance enhancing proxies (PEPs)* [6], in order to improve (interactive) content access. PEPs may generally provide two classes of functionality: **I.** Transport layer connection splitting and other transport layer optimizations aim at decoupling the congestion and error control loops of different path segments, usually isolating the "challenged" (i.e., typically satellite) link [6]. These functions can be performed even if higher layer security protocols (e.g., TLS) are used as TCP headers are in the clear and allow the manipulation of connections. **II.** Application layer functions such as HTTP prefetching and content caching aim at reducing the impact of high RTTs on application layer protocols. These functions need access to the application data and thus won't work unless secure connections are terminated at an intermediary. While, for the regular Internet, Content Delivery Networks (CDNs) may help by replicating services and contents close to the user, this is generally not applicable when the challenged link or path segment includes the last hop.

Today, the advent of QUIC [7]–[9] as a secure transport protocol that also encrypts control information prevents intermediaries from accessing header fields, and thus challenges the implementation and use of the transport layer PEP functionality (I.). Yet, connection splitting remains possible if proxies do not operate transparently but are explicitly included in QUIC connection setup as, e.g., discussed in the MASQUE WG of the IETF [10]. But would QUIC PEPs actually make a difference – for both traditional GEO-based satellite services and for the recent LEO-based ones?

This question arises for (at least) two reasons: (1) QUIC integrates transport setup and TLS 1.3 security and thus reduces the number of round-trips required for connection establishment, possibly even eliminating them entirely with 0-RTT setup for recurring connections to the same server. (2) QUIC supports stream multiplexing within a connection without head-of-line blocking so that only a single connection per server is needed, eliminating repeated setup costs.

In this paper, we seek to explore if these two performance improvements built into QUIC by design are sufficient to offset the need for PEPs in satellite networks. Since end-to-end encryption using TLS/TCP or QUIC rules out application layer PEP operations (II.), we focus on I., i.e., transport layer optimizations. To this end, we make two main contributions:

(1) We present a satellite communication emulation testbed (§ II) which enables reproducible measurements over satellite networks by using our specifically designed QUIC PEP as well as QUIC performance measurement implementations.

(2) We carry out an extensive emulation study (§ III) assuming propagation delays of LEO and GEO satellites and explore various combinations of link characteristics. We report on connection *goodput* over time and, as initial indicators on web performance, *Response Start*, *First Contentful Paint*, and *Page Load Time*.

We compare QUIC vs. TCP, and HTTP/3 (which uses QUIC) vs. HTTP/1.1 (which uses TCP), with and with-

ISBN 978-3-903176-48-5© 2022 IFIP

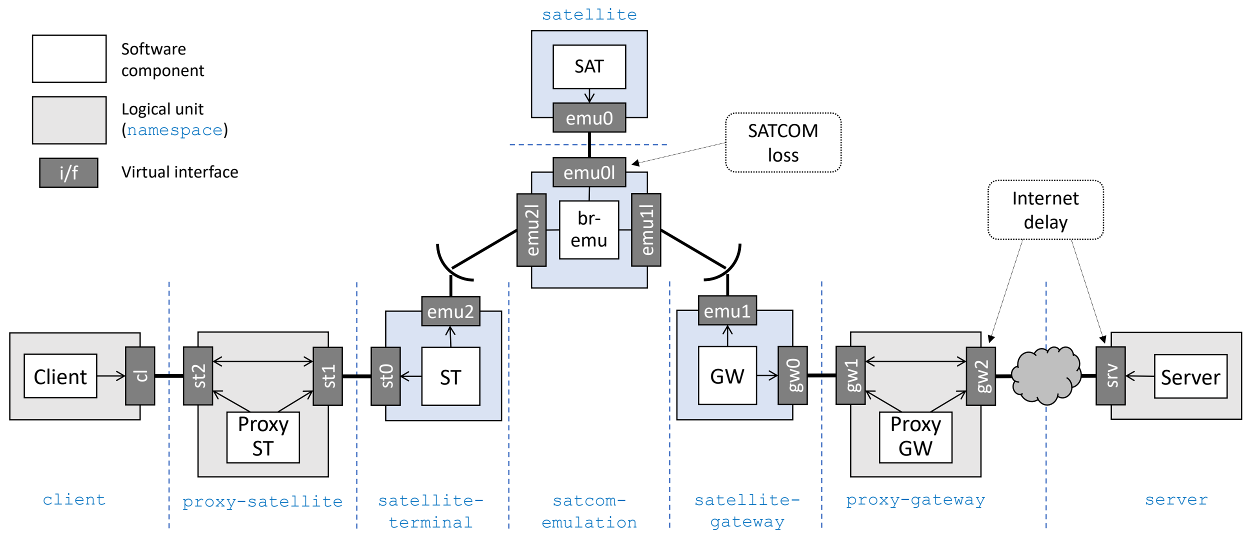

Fig. 1. The SATCOM emulation testbed comprises eight logical units based on Linux network namespaces depicted as light shaded rectangles, where the blue shaded ones make up the OpenSAND SATCOM components (*ST, SAT, GW*). *Client* and *Server* are the measurement endpoints and two PEPs (*Proxy-ST, Proxy-GW*) may be included in the path or bypassed. To induce *SATCOM loss*, the forwarding properties of the indicated interface are modified by `netem` emulation components. In addition, `netem` is also used to create *Internet delay* between the satellite ground station hosting the *Proxy-GW* and the target server. Arrows from components to interfaces indicate network access through these interfaces, while double-sided arrows represent routing between networks. Bold lines connect two interfaces and thin lines indicate the affiliation with a bridge.

out PEPs, representing the past and future web. Evaluating *goodput* over time, we find that the slow start threshold is reached up to 2s faster for QUIC PEP in comparison to QUIC Non-PEP, where the improvements are more pronounced on connections with higher RTTs. Moreover, we find that HTTP/3 and HTTP/3-PEP outperform HTTP/1.1 and HTTP/1.1-PEP in multiple scenarios which we attribute to QUIC's multiplexing capabilities. In addition, HTTP/3-PEP improves over HTTP/3 for *Page Load Time* in GEO orbits: With a reduction of ~330ms for real world conditions, and over 7s in edge cases, we observe a benefit of PEPs for QUIC connections.

In order to enable the reproduction of our findings, we make the developed tools, the raw data of our measurements, as well as the analysis scripts and supplementary files publicly available[1]. We note that our current QUIC PEP evaluation assumes a simplistic setup in which the PEP terminates a QUIC connection and then relays the user data, exposing it to the proxy. While other designs are conceivable, this choice does not affect the purpose of our measurements: understanding if QUIC could benefit from PEPs. While we discuss this and further limitations of our current work in § IV, related work will be detailed in § V, and § VI concludes the paper.

II. SATCOM Emulation Testbed

The *Satellite Communication (SATCOM) emulation testbed* enables reproducible transport as well as application layer measurements over SATCOM networks, leveraging OpenSAND [11] for the emulation of the satellite components. OpenSAND is an established open-source tool for the emulation of SATCOM networks, featuring link-layer emulation based on the DVB-RCS2 and DVB-S2 standards [12]. While OpenSAND shows a high degree of accuracy [13] and is widely used [13]–[16], its complex setup and parametrization is considered a major obstacle [13]. We therefore abstract the parametrization of the *SATCOM emulation testbed*, creating a controlled emulation environment executed on a single Linux system using different *scenarios*. A *scenario* represents the combination of different testbed (e.g., delay, loss, attenuation) as well as transport layer (e.g., congestion control, initial window) parameters used by the emulation. Each *scenario* can be run a specified number of times, where the testbed is gracefully shut down and newly started on every emulation run to rule out any influence of previous runs to a subsequent one. Within each *scenario*, multiple transport and application layer *measurement types* are performed: using QUIC, TCP, HTTP/3, as well as HTTP/1.1, each with and without the aid of a Performance Enhancing Proxy (PEP). By combining multiple *scenarios* within one emulation configuration, the automated visualization enables a holistic view and detailed analysis of transport and application layer performance over all measured *scenarios*.

In this section, § II-A details the *SATCOM emulation testbed design*, followed by a comprehensive description of the *scenario* parameters in § II-B. While § II-C introduces the *measurement types*, § II-D presents the *QUIC PEP* implementation in order to proxy QUIC connections. Following the introduction of the *QUIC performance measurement tool* in § II-E, we conclude with a *validation* of the emulation testbed in § II-F.

A. Design

Fig. 1 depicts the SATCOM emulation testbed design. The emulation comprises eight logical units based on Linux

[1]https://github.com/kosekmi/2022-ifip-nw-quic-proxies

network namespaces depicted als shaded rectangles. On the left-hand side, the *Client* is deployed, running the client-side measurement tools for the respective *measurement type*. The `client` namespace is connected to the `proxy-satellite` namespace, where, depending on the *measurement type*, the *Proxy-ST* proxies the data, or an IP route will forward the data to the `satellite-terminal` namespace. For the emulation of the satellite components (blue shaded rectangles), we leverage OpenSAND. Within the `satellite-terminal` namespace, the interface `st0` forwards the packets to the emulated *satellite terminal ST*. *ST* encapsulates the packets into DVB-RCS2 RLE (Return Link Encaspulation) frames and forwards the data to the `satcom-emulation` namespace which interconnects the three OpenSAND components *ST*, *SAT*, and *GW* (blue shaded rectangles) via the `bridge`-interface `br-emu`. The *SAT* component performs the satellite emulation (e.g., packet delay, signal attenuation), where loss is also emulated on the satellite connection (interface `emu0l`) using `netem`. Following the satellite emulation, packets are sent to the `satellite-gateway` namespace, where the *satellite gateway GW* de-encapsulates the DVB-RCS2 RLE frames, and forwards the data to the `proxy-gateway` namespace. Depending on the *measurement type*, the *Proxy-GW* proxies the packets, or an IP route forwards the data. Using either option, the `gw2` interface subsequently emulates the one-way delay between the `proxy-gateway` and the *Server* using `netem`, i.e., the delay between the satellite ground station hosting the *Proxy-GW* and the target server. As a final step, the *Server* runs the server-side measurement tools for the respective *measurement type* and replies to the measurement requests initiated by the *Client*. While the communication flow in the opposite direction, i.e., from *Server* to *Client*, is mainly identical, the *Internet delay* is added on `srv` instead of `gw2`, the encapsulation/de-encapsulation of *Proxy-GW* and *Proxy-ST* are interchanged, and the packets are encapsulated using DVB-S2 GSE (Generic Stream Encapsulation) instead of DVB-RCS2 RLE.

The depicted *SATCOM testbed design* represents a typical SATCOM use-case: A client connects via a local network to a *satellite-terminal*, which communicates via a *satellite* to a *satellite-gateway*, from where a connection to a server is established. In this use-case, the *satellite-terminal* represents the access router, where the *satellite-gateway* represents the ground station – both operated by the SATCOM network provider. Moreover, both *Proxy-ST* and *Proxy-GW* are also operated by the SATCOM network provider in order to optimize the SATCOM connections using PEPs.

B. Scenarios

A *scenario* represents the combination of different testbed and transport layer parameters used by the emulation, which are presented in Tab. I and detailed in the following.

Testbed parameters. The *Internet delay* states the one-way delay between the satellite ground station and the target server. In contrast, the *SATCOM delay* represents the one-way delay of the SATCOM connection, where a static value, or a list of values stating the change over time, can be configured. Moreover, *loss* configures the loss of the SATCOM connection, and *attenuation* the signal damping.

TABLE I
EMULATION PARAMETERS FOR *scenario* CONFIGURATIONS

Category	Parameter	Values
Testbed	Internet delay	*ms*, static
	SATCOM delay	*ms*, static or dynamic
	SATCOM loss	*percentage*
	SATCOM attenuation	*db*
Transport	Congestion Control	*CUBIC*, *NewReno*
	Initial Window	*maximum packets*
	ACK Frequency (QUIC)	*ack-freq parameters*

Transport parameters. The transport parameters are specific to the measurement components *Client* and *Server*, as well as the PEPs *Proxy-ST* and *Proxy-GW*. Hence, every parameter can be individually specified for each transport component, which enables the optimization of the SATCOM connection using *Proxy-ST* and *Proxy-GW* independently of the *Client* and *Server* parametrization. The *Congestion Control* parameter configures the Congestion Control Algorithm (CCA), where `Cubic` and `NewReno` are the available options. While `Cubic` is the default CCA in most of today's operating systems, `NewReno` is preferred on high RTT connections as experienced in SATCOM networks due to its less aggressive *congestion window* growth [17]. However, more optimized CCA implementations (e.g., `Hybla, BBR`) exist for satellite networks [18]. Since the QUIC implementation we use is currently limited to `Cubic` and `NewReno` (see § II-D and § II-E), we restrict the available options in order to achieve comparable results between QUIC and TCP (see § IV). Moreover, the *Initial Window (IW)* parameter configures the initial window, while the *ACK Frequency* parameter is specific to QUIC and enables support for the *QUIC Acknowledgement Frequency* extension [19].

C. Measurement Types

Within each *scenario*, multiple measurements are performed using QUIC, TCP, HTTP/3 (which uses QUIC), and HTTP/1.1 (which uses TCP), each with and without the aid of a PEP. We acknowledge that the number of websites supporting HTTP/2 is rising [20], and we will additionally evaluate HTTP/2 in a future study (see § IV).

QUIC. For the *QUIC* measurement type, we developed the *QUIC performance measurement tool* (see § II-E) which measures the *connection establishment time*, the *time to first byte*, the *congestion window*, and the *goodput* with and without the usage of the *QUIC PEP* (see § II-D).

TCP. The *TCP* measurement type uses *iperf3* to measure the *congestion window* on the server-side, and the *goodput* on the client-side, optionally proxied by the open-source TCP PEP *PEPsal* [21]. Moreover, the *connection establishment time* and the *time to first byte* are measured using *curl* on the client connecting to a *nginx* web server on the server.

HTTP/3. The *HTTP/3* measurement type leverages the *H2O* web server. While *H2O* serves an arbitrary website, it is accessed using *HTTP/3* by a *Chromium* web browser controlled by *Selenium* on the client-side, with or without using the *QUIC PEP* (see § II-D). Using the *Performance Navigation Timing* API [22], various *web performance metrics* like *Response Start*, *First Contentful Paint*, and *Page Load Time*, are measured.

HTTP/1.1. For *HTTP/1.1*, we measure the same web performance metrics using the same tools as with *HTTP/3*, optionally proxied by the *PEPsal* TCP PEP.

D. QUIC PEP

With TCP headers unencrypted, TCP PEPs can be deployed in SATCOM networks for transport layer connection optimizations based on connection splitting (see § I). With its mandatory header encryption, connection splitting is not applicable to QUIC in a similar fashion and, thus, QUIC connections are inherently end-to-end. To evaluate if QUIC over SATCOM networks could benefit from transport layer optimizations as performed by TCP PEPs, we develop a proxy to enable connection splitting of QUIC connections: The proxy receives incoming connections, establishes a new connection to a predefined destination, and forwards data between both connections while directly mapping the stream IDs. This simplistic design breaks end-to-end encryption and gives the proxy access to the decrypted user data. We therefore explicitly note that the implementation of *QUIC PEP* is a proof-of-concept to evaluate if QUIC over SATCOM networks could benefit from transport layer optimizations; see § IV for a detailed discussion. This design facilitates concatenating multiple proxies and decouples the congestion control loops of different path segments; hence, transport layer connections can be optimized for their respective path segment properties.

As a basis for *QUIC PEP*, the *quicly* [23] implementation is used. In the *default* operation mode, which is used throughout the paper until otherwise noted, the *QUIC PEP* performs the handshakes of incoming connections in parallel with the connection establishment with the next hop, which can be an upstream proxy, or a target server. Using this design, the connection establishment is parallelized, and the client is able to send data before the connection to the target server is established; hence, the time until the server, and subsequently the client, receive the data, is improved. To enable the proxying of HTTP/3 connections, the *h3-capable* operation mode disables this parallelization, and requires a handshake with the next hop to be completed before completing the handshake with the previous hop. This requirement traces back to the server-initiated unidirectional HTTP/3 *control* as well as the *header compression* encode/decode streams, which must be received by the client before the HTTP/3 request is sent in order to avoid connection failure caused by state mismatch between server and client.

For transport layer optimization, *QUIC PEP* offers the parametrization of the *CCA*, the *IW*, and the *QUIC version*, and also supports the *QUIC Acknowledgement Frequency* extension [19].

E. QUIC Performance Measurement Tool

In order to evaluate QUIC performance, we develop a *QUIC performance measurement tool* consisting of a client and a server module. Using a client-initiated connection, the *connection establishment time* as well as the *time to first byte* between client and server is measured. Subsequently, the server sends arbitrary data back to the client, where the corresponding *congestion window* is evaluated on the server-side, and the resulting *goodput* on the client-side. The measurement tool is also based on *quicly* and offers parametrization options similar to *QUIC PEP*: the *CCA*, the *IW*, and the *QUIC version*. Moreover, it offers support for the *QUIC Acknowledgement Frequency* extension [19], and incorporates the *qlog* [24] logging schema to facilitate measurement analysis.

F. Validation

We evaluate the SATCOM emulation testbed in order to validate its *functionality*, *accuracy*, and *reproducibility*. We perform a functional validation by running the SATCOM emulation testbed with different *scenario* configurations for every *measurement type*, where each *scenario* is run 100 times. Before each *scenario* is run, we execute *ICMP* control measurements, and capture the packets on both the client and server side using *tcpdump* during the emulation. Evaluating the *ICMP* measurements as well as the packet captures, we find that the *loss* and *delay* characteristics are accurately emulated. However, we observe an increased SATCOM *delay* for *connection establishment time* and *time to first byte* of ∼10ms for TCP-based connections and ∼25ms for QUIC-based connections. While we are not able to attribute this observation to a single cause, an analysis identified that the delay is added by the OpenSAND SATCOM emulation. Following the successful completion of the emulation runs, we evaluate the *accuracy* via the automated visualization of all *measurement types*, augmented by a manual analysis of the packet captures, the client, server, and proxy logs, and the CPU and RAM utilization metadata. We find, that the emulation is not limited by neither CPU nor RAM utilization, and that the *goodput* converges to the maximum link-layer goodput as configured by the SATCOM components. Moreover, we validate the *reproducibility* of the SATCOM emulation testbed by comparing the results of 3 different Linux systems, finding identical results for all *scenarios* and *measurement types*.

III. Evaluation

With a validated emulation testbed in place, we now proceed to evaluate if QUIC benefits from transport layer optimizations through PEPs, analyzing *goodput* as well as *web performance* characteristics over multiple SATCOM network configurations. The emulation is run on an Ubuntu 18.04 system with Kernel 5.4.0, featuring 2 Intel Xeon E5-2643 6-Core CPUs and 128GB of RAM. The SATCOM components are configured with a clear-sky Signal-to-Noise Ratio of 20dB, a constant QPSK 1/4 modulation, a roll-off factor of 0.25, a return-band (client to server direction) bandwidth of 20MHz, and a forward-band (server to client direction) bandwidth of 50MHz, resulting in a maximum forward-band link-layer goodput of 20Mbps. Using the validation (see § II-F), we ensure that the emulation is not influenced by neither the hardware nor the configuration.

Using the *scenario* model (see § II-B), we differentiate between the two satellite orbits GEO and LEO, where we set the *SATCOM* one-way delay to 250ms for GEO as derived from the speed of light in a vacuum. In order to determine a typical one-way delay for LEO orbits, we perform ∼600k RTT measurements from a vantage point in Central Europe over a period of one week using Starlink [1], where we find a median first hop RTT of ∼32ms (mean ∼33ms). While we acknowledge that the first hop RTT in LEO constellations changes due to the movement of the satellites, we observe relatively constant RTTs for at least 30 consecutive seconds, which is in line with the observations of Kassing et al. [25] and Pavur et al. [14]. As our measurements do not exceed a duration of 20s, we set the *SATCOM* LEO one-way delay to a static value of 16ms (1/2 RTT). In addition to the *SATCOM* delay, we set the *Internet* one-way delay to 40ms for both GEO and LEO orbits in order to emulate the terrestrial distance between the satellite ground station and the target server. Hence, the delay configuration results in a GEO RTT of 580ms and a LEO RTT of 112ms. Lastly, the *attenuation* is configured with 0dB, i.e., no signal damping, and the packet *loss* rate is chosen from 0, 0.01, 0.1, and 1%. While the transport layer loss is considered to range from nearly 0% (where almost all errors are corrected by the link-layer) up to 0.01% in real world satellite conditions [14], we include 0.1 and 1% loss conditions in order to evaluate edge cases.

We further differentiate between the Non-PEP and PEP *transport* parameter configurations. For Non-PEP, we configure both client and server with default values used in both QUIC and TCP stacks, where the *CCA* is set to `Cubic` and the *IW* to 10. For PEP, we use the identical settings as for Non-Pep for client and server, but optimize the SATCOM connection using PEPs by setting the *CCA* to `NewReno` and the *IW* to 100 for both QUIC and TCP. Finally, we use QUIC version 1 for all QUIC configurations.

With the above *scenarios*, we emulate a typical SATCOM use-case with varying orbits as well as loss characteristics, enabling a direct comparison between PEP and Non-PEP connections using QUIC, TCP, HTTP/3, and HTTP/1.1. While the emulation testbed also offers more advanced *scenario* configurations (see § II), we limit our evaluation to the configurations presented above to get a deeper understanding of the *CCA* and *IW* optimization potential, which are traditionally optimized by TCP-PEPs [6].

A. Goodput

We evaluate the *goodput* over time presented in Fig. 2 by analyzing the ratio of bytes received between PEP and Non-PEP (main plots) and absolute goodput (embedded plots) for GEO (top row) as well as LEO (bottom row) satellite orbits, each with 0, 0.01, 0.1, and 1% loss (columns from left to right). With a duration of 15s, a typical file-download use-case is evaluated. The measurements are repeated 100 times and we present the averages over all measurement runs. The main plots show the relative difference of bytes received over time between PEP and Non-PEP connections. A factor of 1 (dashed line) represents no improvement, a factor of >1 shows a benefit by the usage of a PEP, and a factor of <1 represents a degradation of PEP in comparison to Non-PEP connections. Moreover, the embedded plots show the absolute goodput over time for QUIC, QUIC-PEP, TCP, and TCP-PEP connections.

Evaluating the GEO measurements (Fig. 2 top row, *a* to *d*), we observe a benefit of PEPs over all loss configurations, where the factor of bytes received increases up to 10× for PEP QUIC connections (magenta line). The benefit is more pronounced in the first 5 seconds; we attribute this improvement to the *IW* optimization of the PEPs. On high RTT connections as experienced in SATCOM networks, the slow start can become ACK-locked: While the bytes in flight are limited by the *congestion window*, the sender has to pause the transmission until an ACK is received. When we increase the *IW* by 10×, more data can initially be sent before the receipt of an ACK increases the *congestion window*, resulting in a faster slow start until the slow start threshold is reached. This benefit can also be observed in the embedded plots of the top row showing the absolute goodput over time: The slow start is considerably faster using PEPs (magenta and cyan lines), improving over Non-PEPs (green and red lines) for 0 and 0.01% loss configurations by up to 2s (Fig. 2 *a* and *b*). While the edge case loss configurations of 0.1 and 1% (*c* and *d*) also show benefits from the increased *IW* of the PEP connections (magenta and cyan lines), we observe a degradation in goodput after reaching the slow start threshold for both PEP and Non-PEP connections. The overall goodput is drastically reduced; however, the PEP connections reach the slow start threshold faster. In addition, we observe a benefit for QUIC connections (green and magenta lines) in comparison to TCP (red and cyan lines), where both QUIC connections result in higher goodput following slow start. Evaluating 1% loss (Fig. 2 *d*) for TCP connections, we observe an increase for TCP-PEP of up to 85× within the first second of the measurement (cyan line). While this factor exceeds our increase in *IW* of 10×, we attribute this outlier to the *iperf3* measurement tool: While we sample the bytes received in 0.1s intervals, *iperf3* regularly omits the first 1–8 intervals in the 1% loss configuration, resulting in the observed inaccuracies within the first second.

Evaluating the LEO measurements (Fig. 2 bottom row, *e* to *h*), we again observe a benefit of PEPs over all loss configurations, where the factor of bytes received increases up to 3× for PEP QUIC connections (magenta line). This benefit is more pronounced within the first 2 seconds; however, while the increase in *IW* results in a faster slow start, the slow start threshold is reached at the same time for PEP (magenta and cyan) and Non-PEP (green and red) connections (bottom row embedded plots). Moreover, the benefit is less pronounced in comparison to GEO. Evaluating the 0.1 and 1% loss edge cases (*g* and *h*), we observe similar trends in comparison to GEO while the overall goodput is drastically reduced. Furthermore,

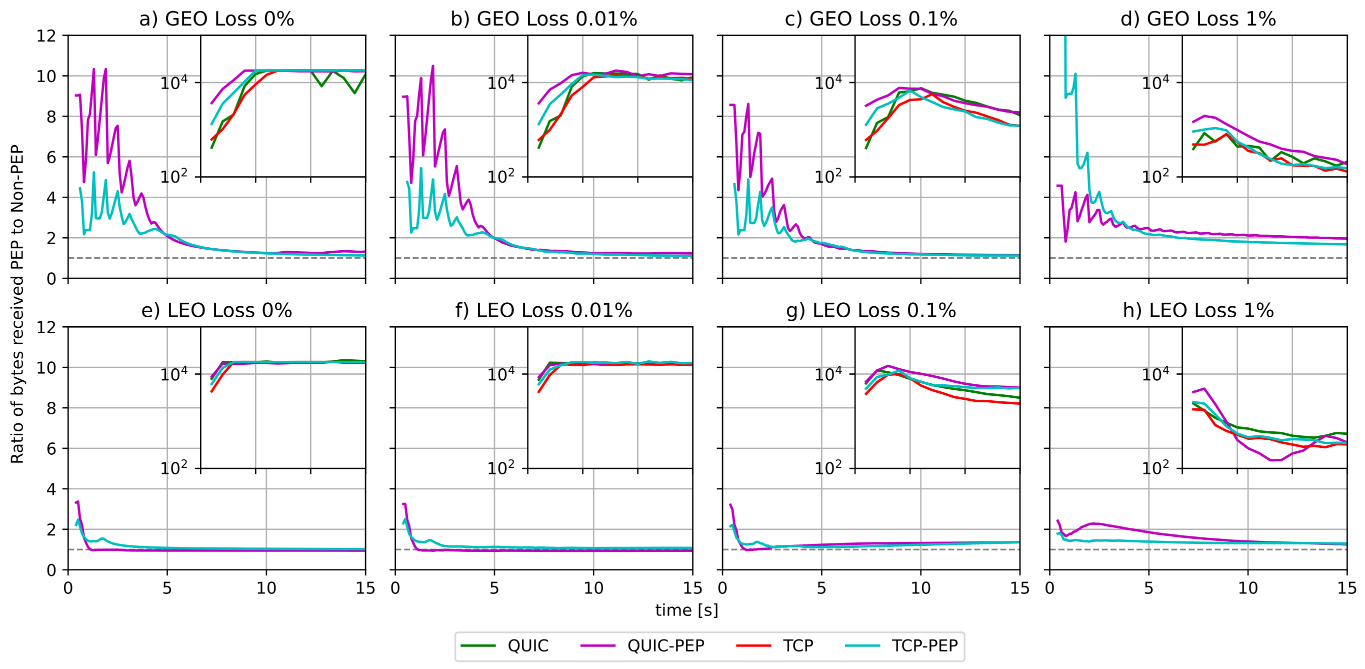

Fig. 2. Ratio of bytes received PEP to Non-PEP (main plots) and absolute goodput (embedded plots) over 15s for GEO (top row) as well as LEO (bottom row) satellite orbits for real world (0, 0.01%) and edge case (0.1, 1%) loss conditions (columns from left to right). A factor of 1 represents no improvement, a factor of >1 shows a benefit by the usage of a PEP, and a factor of <1 represents a degradation of PEP in comparison to Non-PEP connections.

we see a benefit of PEP connections almost over the complete measurement for 0.1% loss (Fig. 2 *g*), where QUIC-PEP shows a fast degradation on 1% loss following slow start (Fig. 2 *h*). ***Takeaway:*** *While we observe benefits for PEP connections over all orbits and loss configurations, the improvements are, expectedly, more pronounced on connections with higher RTTs and less loss, where the slow start threshold is reached up to 2s faster in comparison to Non-PEP connections. The benefits are primarily limited to the slow start phase in real world conditions, which results from the* IW *optimization of the PEPs. However, PEP connections can also lead to a degradation in goodput on SATCOM connections with shorter RTTs.*

B. Web Performance

We evaluate the web performance by analyzing the median values of *Response Start (RS)*, *First Contentful Paint (FCP)*, and *Page Load Time (PLT)* over 100 measurement runs for every orbit, loss, and protocol combination presented in milliseconds in Tab. II. To enable the proxying of HTTP/3 connections, the *h3-capable* operation mode of *QUIC PEP* is used (see § II-D); hence, a handshake with the next hop has to be completed before completing the handshake with the previous hop, resulting in a sequential connection establishment. Moreover, we use HTTP/1.1 without encryption in contrast to the TLS 1.3 encrypted HTTP/3. While the overhead of the TLS encryption adds 1 (in case of TLS 1.3), respective 2 (in case of TLS 1.2) RTTs to the TCP connection establishment of HTTP/1.1, the overhead is systematic. Hence, we leverage unencrypted HTTP/1.1 as a performance oriented baseline for our comparison to HTTP/3, where our results can be extrapolated for TLS 1.3 / TLS 1.2 encrypted HTTP/1.1 by adding 1, respective 2, RTTs.

For our analysis, we use the *ETSI Kepler Web Reference Page* [26] that aims to represent a "typical" website. While we acknowledge that an objective characterization of a typical website is debatable considering the heterogeneity of the web, choosing this "representative" website can still serve as a first indication if QUIC is able to benefit from proxies. A more diverse set of websites will be evaluated in future work (see § IV). The default settings of the *Chromium* client and of the *H2O* server are used to transport the website's 75 objects (∼880kb) either over a single QUIC connection with 6 streams in case of HTTP/3 (h3), or over 6 distinct TCP connections in case of HTTP/1.1 (h1). QUIC version 1 and HTTP/3 version draft-29 are used.

First, we take a look at the *Response Start (RS)* web performance metric, which represents the time that passes between the sending of the first packet of the transport handshake and the reception of the first byte of the HTTP response by the client [27]. Hence, the *RS* is expected to resemble 2 RTTs plus the static one-way overhead added by OpenSAND of ∼10ms for TCP and ∼25ms for QUIC (see § II-F). Analyzing the GEO *RS* presented in Tab. II, we observe ∼1.2s for the TCP-based h1 and h1-PEP protocols, while the QUIC-based h3 is moderately (∼1.3s) and h3-PEP considerably (∼2.7s) slower in comparison. The duration of *RS* is largely independent of the packet loss rate. The protocols h1, h1-PEP and h3 show the expected *RS* that roughly equals 2 GEO RTTs of 580ms plus the static overhead added by OpenSAND. The increased *RS* of h3-PEP (∼2.7s) traces back to the *h3-capable* operation mode of *QUIC PEP*: Because the connections are established

TABLE II

MEDIAN *Response Start (RS)*, *First Contentful Paint (FCP)*, AND *Page Load Time (PLT)* IN MILLISECONDS OF PEP AND NON-PEP HTTP/3 (H3) AS WELL AS HTTP/1.1 (H1) CONNECTION FOR GEO (TOP) AS WELL AS LEO (BOTTOM) SATELLITE ORBITS FOR REAL WORLD (0, 0.01%) AND EDGE CASE (0.1, 1%) LOSS CONDITIONS (COLUMNS FROM LEFT TO RIGHT).

		Response Start (RS)				**First Contentful Paint (FCP)**				**Page Load Time (PLT)**			
Orbit	**Protocol / Loss**	**0%**	**0.01%**	**0.1%**	**1%**	**0%**	**0.01%**	**0.1%**	**1%**	**0%**	**0.01%**	**0.1%**	**1%**
GEO	h3	1314	1310	1312	1321	2759	2760	2773	2861	7669	7671	7739	16865
	h3-PEP	2717	2715	2717	2727	3581	3575	3581	4196	7344	7336	7522	9409
	h1	1222	1220	1214	1221	3841	3844	3832	4172	10806	10835	11176	15802
	h1-PEP	1215	1215	1214	1215	3852	3842	3836	4404	11144	11124	11175	13994
LEO	h3	434	436	435	447	1408	1410	1415	1489	2612	2617	2649	7010
	h3-PEP	1418	1414	1419	1416	1880	1868	1882	1914	3172	3173	3264	3872
	h1	291	288	294	303	1138	1135	1145	1329	3218	3227	3302	4573
	h1-pep	289	286	287	301	1095	1085	1084	1356	3308	3321	3367	4298

sequentially, and multiple message exchanges are required for each connection setup, the h3-PEP *RS* is considerably higher. Evaluating h3 and h3-PEP, we find additional overheads for QUIC by the processing on the server and the proxies. Analyzing the *quicly* library used by *QUIC PEP* as well as the *H2O* web server (see § II), we find that the additional overheads can be attributed to the `quicly_accept()` function which accepts new connections, resulting in an increase of 2–6ms per connection establishment. Looking at the *RS* for the LEO network over all loss configurations, we observe the same trends as in GEO: While h1 and h1-PEP are on par with ∼290ms, h3 is moderately (∼435ms), and h3-PEP considerably (∼1.4s) slower in comparison. Our results therefore show that the *RS* of the TCP-based protocols h1 and h1-PEP are faster in comparison to the QUIC-based protocols h3 and h3-PEP.

As a second web performance metric we evaluate *First Contentful Paint (FCP)*, which measures the time between the sending of the first packet of the transport handshake and the rendering of the first element by the browser; hence, it represents the user-perceived load speed of a website [28]. Evaluating the GEO *FCP* presented in Tab. II, we observe identical values per protocol over the 0 and 0.01% loss configurations with ∼2.8s for h3, ∼3.6s for h3-PEP, as well as ∼3.8s for both h1 and h1-PEP. While the QUIC-based protocols h3 and h3-PEP show a slower *RS*, both improve over their TCP counterparts for *FCP*. We attribute this improvement to QUIC's multiplexing capabilities: While 6 distinct TCP connections are established by *Chromium* for h1 and h1-PEP to avoid head-of-line blocking, 6 streams are sent over a single QUIC connection for h3 and h3-PEP. Hence, the overhead of connection establishment is considerably reduced, leading to faster *FCP* for h3 and h3-PEP in comparison to h1 and h1-PEP. Moreover, we also observe a relative improvement comparing *RS* and *FCP* for h3 and h3-PEP: While the overhead of the sequential connection establishment of h3-PEP results in a *RS* slowdown of ∼1.4s, this difference is reduced to ∼820ms for *FCP*. Looking at GEO 0.1% loss, we observe identical values in comparison to 0 and 0.01%, while 1% loss shows an increase of up to ∼600ms for h3-PEP as well as h1-PEP. We observe similar trends in the *FCP* of GEO and LEO orbits for 0 and 0.01% loss. However, due to the reduced RTT in the LEO orbit, h3-PEP can not yet overcome the initial overhead of the sequential connection establishment in comparison to h1 and h1-PEP. With a *FCP* of ∼1.9s, h3-PEP falls short of h1 and h1-PEP (∼1.1s). Moreover, h3-PEP also falls short of h3 by ∼470ms, but still improves over their *RS* difference of ∼1s. Considering LEO 0.1% loss, we observe slightly increased values in comparison to 0 and 0.01%, where 1% loss shows an increase of up to ∼260ms for h1-PEP.

Our third web performance metric is the *Page Load Time (PLT)* (see Tab. II), which represents the time between the sending of the first packet of the transport handshake until all content of the website is received by the browser [29]. Looking at GEO's 0 and 0.01% loss configurations, we observe the fastest *PLT* for h3-PEP with ∼7.3s, followed by h3 (∼7.7s), h1 (∼10.8s), and h1-PEP (∼11.1s). We again attribute the performance benefit of the QUIC-based h3 and h3-PEP connections to QUIC's stream multiplexing feature. Moreover, we observe the break-even of the initial overhead of the sequential connection establishment for h3-PEP, outperforming h3 by ∼330ms, and h1 as well as h1-PEP by more than 3s; hence, we see a benefit of PEPs for QUIC connections in GEO orbits. At higher loss rates, the h3-PEP performance advantage increases: With 1% loss, h3-PEP finishes loading more than 7s earlier than h3, and ∼4.5s earlier than h1-PEP. Comparing GEO h1 and h1-PEP, we observe that h1 is slightly faster for 0 and 0.01% loss configurations, while they are on par for 0.1%; however, h1-PEP shows an improvement of ∼2s over h1 for 1% loss. Evaluating *PLT* for LEO orbits for 0 and 0.01% loss, we observe similar trends in comparison to GEO. While h1 (∼3220ms) and h1-PEP (∼3315ms) show the slowest *PLT*, h3-PEP is faster with ∼3170ms, only outperformed by h3 with ∼2615ms. Considering the reduced RTT in LEO orbits, h3-PEP is not able to overcome the initial overhead of the sequential connection establishment in comparison to h3, but improves on h1 and h1-PEP by ∼50ms, respective ∼145ms. Looking at 0.1% loss, this improvement becomes more distinct, where on 1% loss h3-PEP outperforms h1 and h1-PEP by more than 400ms, and even improves over h3 by more than 3s.

Takeaway: *Due to QUIC's multiplexing capabilities, we* *observe an improvement of the QUIC-based protocols h3 and h3-PEP over both h1 and h1-PEP, where h3 and h3-PEP achieve faster* FCP *and* PLT *in GEO orbits, as well as faster* PLT *in LEO orbits. Moreover, while the initial overhead of the sequential connection establishment of h3-PEP leads to slower* RS *and* FCP *in comparison to h3, h3-PEP improves over h3 for* PLT *in GEO orbits: With a reduction of ∼330ms for real world conditions, and over 7s in edge cases, we observe a benefit of PEPs for QUIC connections.*

IV. Limitations and Future Work

While our *QUIC PEP* realizes transport layer optimizations by means of connection splitting, the implementation is considered a proof-of-concept as the proxy is able to access the decrypted data, yet sufficient for the purpose of exploring possible benefits of QUIC PEPs. Hence, we are currently exploring schemes where the QUIC payload remains end-to-end encrypted and only selected control information are exposed to the proxies. Traffic tunneling mechanisms as discussed in the IETF MASQUE WG [30] preserve encryption and allow for an independent congestion control loop between these PEPs, but still run nested congestion control end-to-end, not providing connection splitting. Yet, the basic signaling mechanisms provide a well-defined means for interacting with proxies and thus could be leveraged for extensions towards QUIC PEPs, with further mechanisms for preserving end-to-end encryption to be explored.

Moreover, the PEPed QUIC connections are currently established per client request, offering additional optimization potential: By leveraging *0-RTT* between the proxies, the time required for the connection establishment can be further reduced. Considering the CCAs, both *QUIC PEP* and the *QUIC performance measurement tool* currently only offer `Cubic` and `NewReno`. While more optimized implementations exist for satellite networks (e.g., `Hybla, BBR`), we will integrate and evaluate additional *CCAs* in a future study. We also plan to incorporate the *Acknowledgement Frequency* extension [19] in order to reduce the number of acknowledgements sent, as well as the QUIC *BDP Frame* extension [31] to accelerate the goodput ramp-up on repeated connections; both show promising results for SATCOM networks.

While our findings revealed that QUIC connections are able to benefit from proxies through transport layer optimization for both *goodput* and *web performance*, we acknowledge that our findings on *web performance* are (naturally) influenced by the website selection. Therefore, we seek to evaluate a more diverse set of websites in a future study in order to generalize our findings, where we will also incorporate HTTP/2.

V. Related Work

Several papers [32]–[37] investigate the usage of QUIC in satellite networks. They find that QUIC realizes lower Page Load Times on the web compared to TCP, mainly due to its faster connection setup [32], [33]. Also, the utility of PEPs for improving the performance of TCP over long-delay satellite links has long been known [38]. The studies that compare the performance of TCP using PEPs (TCP-PEP) with QUIC, find that TCP-PEP generally outperforms QUIC for larger transfers [34]–[37].

Some authors argue that specific tuning, such as an increase of the initial window, can improve QUIC's performance [34], [36], [37]. Custura et al. [37] investigate how decreasing the acknowledgement frequency can reduce the control overhead and in some cases improve performance. Kuhn et al. [31] introduce the QUIC *BDP Frame* extension that accelerates the throughput ramp-up on repeated connections over long-delay satellite links.

We take a different approach to examine the impact of transport-layer optimizations on the performance of QUIC over GEO and LEO links by developing a proof-of-concept QUIC-PEP. Previous work on intermediaries with QUIC for SATCOM is sparse and the closest related work is from Pavur et al. [14] on QPEP, who multiplex TCP connections over a single QUIC connection between ground terminals and thereby significantly reduce Page Load Times. They, however, study a long-standing QUIC connection and therefore exclude the connection setup. While `netem` is commonly applied at a single point to model a satellite link [32], [33], [39], we use OpenSAND [11] in conjunction with netem and induce delay at different points along the network path to model our emulation closer to reality.

A large fraction of previous studies [40]–[43] examines the performance of Google's QUIC flavor (gQUIC) that differs in fundamental points, such as the cryptographic handshake, from the IETF specification of QUIC [44]. Recent web performance evaluations show that HTTP/3 using IETF QUIC does not necessarily perform better than HTTP/2, which builds upon TCP [45], [46]. Saif et al. [45] find that TCP performs better except if loss is present. In this case, QUIC's design shows its strength by reducing the impact of head-of-line blocking. Similarly, Yu et al. [46] present mixed findings and highlight the impact of configuration choices on QUIC's performance. We add a SATCOM perspective to the research space of HTTP/3 performance by analyzing common web performance metrics such as the Page Load Time.

VI. Conclusion

In this paper, we presented a satellite emulation testbed which enables reproducible QUIC, TCP, HTTP/3, and HTTP/1.1 measurements by using our specifically designed QUIC PEP as well as QUIC performance measurement implementations. Using the emulation testbed, we carried out an extensive emulation study for LEO and GEO satellites, exploring various combinations of link characteristics. We found, that the slow start threshold is reached up to 2s faster for QUIC PEP in comparison to QUIC Non-PEP, where the improvements are more pronounced on connections with higher RTTs. Moreover, we showed that HTTP/3 and HTTP/3-PEP outperform HTTP/1.1 and HTTP/1.1-PEP in multiple *web performance* scenarios which we attribute to QUIC's multiplexing capabilities. In addition, HTTP/3-PEP also improves over HTTP/3 for *Page Load Time* in GEO orbits by over 7s in edge cases. Hence, our findings show that PEPs can be beneficial for QUIC connections and warrant further exploration: While the presented QUIC PEP is considered a proof-of-concept, the basic signaling mechanisms discussed in the IETF MASQUE WG provide well-defined means for interacting with proxies and thus could be leveraged for extensions towards end-to-end encrypted QUIC PEPs.

Acknowledgements

We thank Robert Brüning, Felix Beil, and Curt Polack for their valuable efforts, as well as the anonymous reviewers for their insightful feedback.

References

[1] "Starlink," [Accessed 2022-Apr-30]. [Online]. Available: https://www.starlink.com

[2] A. Boyle, "Amazon to offer broadband access from orbit with 3,236-satellite 'project kuiper' constellation," [Accessed 2022-Apr-30]. [Online]. Available: https://www.geekwire.com/2019/amazon-project-kuiper-broadband-satellite/

[3] OneWeb, "Oneweb: Connect with ease," [Accessed 2022-Apr-30]. [Online]. Available: https://oneweb.net/connect_with_ease

[4] Telesat, "Telesat: Global satellite operators," [Accessed 2022-Apr-30]. [Online]. Available: https://www.telesat.com/

[5] Iridium, "Staying connected," [Accessed 2022-Apr-30]. [Online]. Available: https://www.iridium.com

[6] J. Griner *et al.*, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," RFC 3135, Jun. 2001. [Online]. Available: https://www.rfc-editor.org/info/rfc3135

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

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

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

[10] "Multiplexed Application Substrate over QUIC Encryption (masque)," [Accessed 2022-Apr-30]. [Online]. Available: https://datatracker.ietf.org/wg/masque/about/

[11] "OpenSAND SATCOM emulation," [Accessed 2022-Apr-30]. [Online]. Available: https://opensand.org/

[12] "DVB-S2 specifications," [Accessed 2022-Apr-30]. [Online]. Available: https://www.etsi.org/technologies/dvb-s-s2

[13] A. Auger *et al.*, "Making Trustable Satellite Experiments: An Application to a VoIP Scenario," in *VTC*, 2019-Spring. [Online]. Available: https://doi.org/10.1109/VTCSpring.2019.8746404

[14] J. Pavur *et al.*, "QPEP: An Actionable Approach to Secure and Performant Broadband From Geostationary Orbit," in *NDSS*, 2021. [Online]. Available: https://doi.org/10.14722/ndss.2021.24074

[15] C. Baudoin and F. Arnal, "Overview of Platine emulation testbed and its utilization to support DVB-RCS/S2 evolutions," *ASMS*, 2010. [Online]. Available: https://doi.org/10.1109/ASMS-SPSC.2010.5586897

[16] F. Arnal *et al.*, "Handover Management for Hybrid Satellite/Terrestrial Networks," *LNICST*, 2013. [Online]. Available: https://doi.org/10.1007/978-3-642-36787-8_12

[17] I. Rhee *et al.*, "CUBIC for Fast Long-Distance Networks," RFC 8312, Feb. 2018. [Online]. Available: https://doi.org/10.17487/RFC8312

[18] S. o. Claypool, "Comparison of TCP Congestion Control Performance over a Satellite Network," *PAM*, 2021. [Online]. Available: https://doi.org/10.1007/978-3-030-72582-2_29

[19] J. Iyengar and I. Swett, "QUIC Acknowledgement Frequency," IETF, Internet-Draft draft-ietf-quic-ack-frequency-01, Oct. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-quic-ack-frequency/

[20] "HTTP usage statistics," [Accessed 2022-Apr-30]. [Online]. Available: https://w3techs.com/technologies/history_overview/site_element/all/y

[21] C. Caini *et al.*, "PEPsal: a Performance Enhancing Proxy designed for TCP satellite connections," *VTC*, 2006. [Online]. Available: https://doi.org/10.1109/VETECS.2006.1683339

[22] "PerformanceNavigationTiming API," [Accessed 2022-Apr-30]. [Online]. Available: https://w3c.github.io/navigation-timing/#process

[23] "Quicly QUIC implementation," [Accessed 2022-Apr-30]. [Online]. Available: https://github.com/h2o/quicly

[24] R. Marx *et al.*, "Debugging QUIC and HTTP/3 with Qlog and Qvis," *ANRW*, 2020. [Online]. Available: https://doi.org/10.1145/3404868.3406663

[25] S. Kassing and other, "Exploring the "Internet from Space" with Hypatia," *IMC*, 2020. [Online]. Available: https://doi.org/10.1145/3419394.3423635

[26] "ETSI Kepler Web Reference Page," [Accessed 2022-Apr-30]. [Online]. Available: https://www.etsi.org/deliver/etsi_tr/102500_102599/102505/01.02.01_60/tr_102505v010201p.pdf

[27] "MDN Web Docs - Time to first byte," [Accessed 2022-Apr-30]. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/time_to_first_byte

[28] "MDN Web Docs - First contentful paint," [Accessed 2022-Apr-30]. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/First_contentful_paint

[29] "MDN Web Docs - Page load time," [Accessed 2022-Apr-30]. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Page_load_time

[30] D. Schinazi, "UDP Proxying Support for HTTP," IETF, Internet-Draft draft-ietf-masque-connect-udp-09, Oct. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-masque-connect-udp/

[31] N. Kuhn *et al.*, "Evaluating BDP FRAME extension for QUIC," [Accessed 2022-Apr-30]. [Online]. Available: https://arxiv.org/abs/2112.05450

[32] S. Yang *et al.*, "Performance Analysis of QUIC Protocol in Integrated Satellites and Terrestrial Networks," in *IWCMC*, 2018. [Online]. Available: http://doi.org/10.1109/IWCMC.2018.8450388

[33] H. Zhang *et al.*, "How Quick Is QUIC in Satellite Networks," *CSPS*, 2019. [Online]. Available: https://doi.org/10.1007/978-981-10-6571-2_47

[34] L. Thomas *et al.*, "Google quic performance over a public satcom access," *Int. J. Satell. Commun. Netw.*, vol. 37, pp. 601–611, 2019. [Online]. Available: https://doi.org/10.1002/SAT.1301

[35] J. Deutschmann *et al.*, "Satellite internet performance measurements," in *2019 International Conference on Networked Systems (NetSys)*, 2019, pp. 1–4. [Online]. Available: https://doi.org/10.1109/NetSys.2019.8854494

[36] N. Kuhn *et al.*, "QUIC: Opportunities and threats in SATCOM," *ASMS/SPSC*, 2020. [Online]. Available: https://doi.org/10.1109/ASMS/SPSC48805.2020.9268814

[37] A. Custura *et al.*, "Impact of Acknowledgements using IETF QUIC on Satellite Performance," *ASMS/SPSC*, 2020. [Online]. Available: https://doi.org/10.1109/ASMS/SPSC48805.2020.9268894

[38] M. Sooriyabandara *et al.*, "TCP Performance Implications of Network Path Asymmetry," RFC 3449. [Online]. Available: https://www.rfc-editor.org/info/rfc3449

[39] Y. Wang *et al.*, "Performance Evaluation of QUIC with BBR in Satellite Internet," in *WiSEE*, 2018. [Online]. Available: https://doi.org/10.1109/WiSEE.2018.8637347

[40] A. M. Kakhki *et al.*, "Taking a Long Look at QUIC: An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols," *IMC*, 2017. [Online]. Available: https://doi.org/10.1145/3131365.3131368

[41] G. Carlucci *et al.*, "HTTP over UDP: An Experimental Investigation of QUIC," in *SAC*, 2015. [Online]. Available: https://doi.org/10.1145/2695664.2695706

[42] Y. Yu *et al.*, "When QUIC meets TCP: An experimental study," in *IPCCC*, 2017. [Online]. Available: https://doi.org/10.1109/PCCC.2017.8280429

[43] S. Cook *et al.*, "QUIC: Better for what and for whom?" in *ICC*, 2017. [Online]. Available: https://doi.org/10.1109/ICC.2017.7997281

[44] M. Nottingham, "What's Happening with QUIC," [Accessed 2022-Apr-30]. [Online]. Available: https://www.ietf.org/blog/whats-happening-quic/

[45] D. Saif *et al.*, "An Early Benchmark of Quality of Experience Between HTTP/2 and HTTP/3 using Lighthouse," *ICC*, 2021. [Online]. Available: https://doi.org/10.1109/ICC42927.2021.9500258

[46] A. Yu and T. A. Benson, "Dissecting Performance of Production QUIC," *WWW*, 2021. [Online]. Available: https://doi.org/10.1145/3442381.3450103