FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Lehrstuhl Informatik 7

Rechnernetze und Kommunikationssysteme

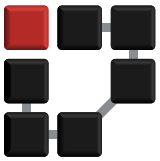
Sebastian Endres

Evaluation and Optimization of the IETF QUIC Protocol for Satellite Networks

Masterarbeit im Fach Informatik

12. Januar 2022

Please cite as:
Sebastian Endres, “Evaluation and Optimization of the IETF QUIC Protocol for Satellite Networks,” Master’s Thesis (Masterarbeit), Univ. of Erlangen-Nürnberg, Computer Science 7, January 2022.



Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Lehrstuhl Informatik 7
Rechnernetze und Kommunikationssysteme)

Martensstr. 3 · 91058 Erlangen · Deutschland

https://www.cs7.tf.fau.de/

Evaluation and Optimization of the IETF QUIC Protocol for Satellite Networks

Masterarbeit im Fach Informatik

vorgelegt von

Sebastian Endres

geb. am 02. Februar 1995
in Roth

angefertigt am

Lehrstuhl Informatik 7
Rechnernetze und Kommunikationssysteme

Department Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

Betreuer: **Prof. Dr-Ing. Reinhard German**
Jörg Deutschmann, M.Sc.

Abgabe der Arbeit: **12. Januar 2022**

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.
Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties.
I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Sebastian Endres)
Erlangen, 12. Januar 2022

Abstract

Due to the poor performance of TCP on high latency paths, in networks with geostationary satellite links Performance Enhancing Proxies are used. However, because of encrypted headers, this is no longer possible with QUIC, so that the achieved throughput is usually low. At the same time, with features such as 0-RTT connection establishment and new congestion control mechanisms, QUIC offers fast data transmissions even under challenging conditions.

So far, there has been little research on the performance of the IETF QUIC in satellite communications and only few implementations have been considered. In this thesis, we present an automated test environment based on the QUIC-Interop-Runner used by the IETF. It allows performing and evaluating uniform measurements with the main implementations over both emulated and real satellite links.

The measurement results show large differences between implementations and that the speed depends on both the server and the client in use. We can verify that the average link utilization is very low. We also notice that with some implementations it is impossible to transfer larger files under such conditions. Based on the comparison of different real-world satellite accesses, we conclude that a higher available data rate alone does not always lead to a better throughput.

Kurzfassung

Aufgrund der schlechten Leistung von TCP bei Pfaden mit hoher Latenz werden in Netzwerken mit geostationären Satellitenverbindungen Performance Enhancing Proxies eingesetzt. Das ist bei QUIC wegen verschlüsselter Header jedoch nicht mehr möglich, weshalb der erreichte Durchsatz meist niedrig ist. Gleichzeitig bietet QUIC durch Features wie 0-RTT Verbindungsaufbau und neue Congestion-Control-Mechanismen die Chance, auch unter herausfordernden Bedingungen schnelle Datenübertragungen zu gewährleisten.

Bisher gibt es kaum Forschung zur Leistung von IETF-QUIC in der Satellitenkommunikation und es wurden nur wenige Implementierungen betrachtet. Wir stellen eine automatisierte Testumgebung vor, die auf dem von der IETF eingesetzten QUIC-Interop-Runner basiert. Sie ermöglicht es, einheitliche Messungen mit den wichtigsten Implementierungen über sowohl emulierte als auch reale Satellitenverbindungen durchzuführen und auszuwerten.

Die Messergebnisse zeigen große Unterschiede zwischen den Implementierungen und dass die Geschwindigkeit sowohl vom verwendeten Server als auch vom Client abhängt. Wir können verifizieren, dass die durchschnittliche Ausnutzung der Links sehr niedrig ist. Wir stellen auch fest, dass es unter derartigen Bedingungen mit einigen Implementierungen nicht möglich ist, größere Dateien zu übertragen. Aus dem Vergleich verschiedener realer Satellitenzugänge folgern wir, dass allein eine höhere verfügbare Datenrate nicht immer zu einem besseren Durchsatz führt.

Contents

Chapter 1

Introduction

> Once you've been in space, you appreciate
> how small and fragile the Earth is.
>
> *Valentina Tereshkova*

Much has changed in space since 1945, when Arthur C. Clarke proposed the usage of "Extra-terrestrial Relays" that orbit around Earth and act like bent-pipes in radio communications to make it possible to communicate across continents [1]. Since then, satellite communications (SATCOM) have become the most relevant field of space activity. Being commercially profitable it allows spreading information all over the world. Both benefits are important for western market-based democracies [2]. In 2020, there are almost 2700 operational satellites in space with almost half of them used for communications[1]. With the construction of Low Earth Orbit (LEO) mega-constellations, this number is currently rapidly increasing. Nowadays, the predominant application of communication satellites is providing users on Earth with Internet access. Even in case of natural disasters they can reliably supply large areas and help to fulfill the 9th Sustainable Development Goal (SDG), to "build resilient infrastructure"[2]. Satellite based Internet access is typically used to connect remote and rural locations, create backup links, rapidly deploy new services and provide mobile environments, like vessels, ships, and aircraft [3]. For a couple of years, a lot of attention has been on the new LEO mega-constellations. However, geostationary satellites will remain relevant for satellite Internet for the foreseeable future due to their advantages, like fixed position in the sky, longer lifetime and higher coverage. This thesis is focussing on geostationary satellites.

[1] Visualizing All of Earth's Satellites: Who Owns Our Orbit? `https://www.visualcapitalist.com/visualizing-all-of-earths-satellites/` *(visited on 2021-12-21)*

[2] SDG9: "Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation" `https://sdgs.un.org/goals/goal9` *(visited on 2021-12-21)*

Major transformations are currently also taking place in the field of transport protocols. The new QUIC protocol is about to replace the aging TCP at least in web stacks. This is mainly because it usually achieves a higher performance [4]. Middleboxes prevent the deployment of new mechanisms on TCP and lead to an ossified Internet. Therefore, their presence on the Internet is consistently tackled by encrypting transport layer information.

This presents a threat to SATCOM operators who use middleboxes named Performance Enhancing Proxies (PEPs) so far to transparently enhance the performance of end-to-end TCP connections on the satellite segment. They are required because satellite path characteristics differ significantly from those of terrestrial paths, mainly in higher latency. This has an impact on the efficiency of transport protocols [3]. For QUIC connections without path-awareness, this results in very poor throughput on links via geostationary satellites, as current research shows. Downgrading to TCP by blocking QUIC is not a long-term solution. Instead, the opportunities for extensibility and features, such as 0-RTT handshakes and pluggable Congestion Control Algorithms (CCAs) appears favorable.

However, to this point, there has been very little research on the performance of IETF QUIC in SATCOM. In addition, only very few implementations of QUIC have been taken into account yet, although it must be assumed that the performance heavily depends on the choice of implementation.

By presenting an automated test bed that allows running uniform measurements with numerous client and server implementations via both emulated and real satellite links, we are able to assess the influence of the implementation on the performance of a QUIC connection. We can also to identify implementations that achieve slightly better performance on satellite links than others, and analyze weaknesses of poorly performing ones by inspecting their traces in detail. These conclusions might also apply to some extent to other networks with similar properties, like on-board Wi-Fi in trains, though, this will require future research.

In Chapter 2, we introduce to SATCOM and explain challenges of links via geostationary satellite links. Additionally, we present an overview of the most relevant features of QUIC in the context of SATCOM. After that, in Chapter 3, we give an overview of related research with the focus on measurements of QUIC via geostationary satellites. Chapter 4 documents the newly developed test environment for massive evaluation of QUIC implementations. The results of our measurements, using this tool, are presented and evaluated in Chapter 5. Finally, Chapter 6 summarizes the results and names future work.

Chapter 2

Fundamentals

> [...] I cannot imagine a better instrument of counter-revolution
>
> *With these words, Josef Stalin rejected a plan to improve the Soviet telephone system in 1953.*

In this chapter, we outline the technical concepts and basis of our work. First we give a short overview about SATCOM in general, followed by the characteristics of communication channels with satellites in the Geostationary Earth Orbit. Measures already currently applied to TCP to achieve high efficiency over such challenging links are also presented. In the second part, most important Congestion Control Algorithm are briefly introduced. The last part of this chapter deals with QUIC, with its current and proposed features and their importance for SATCOM.

2.1 Satellite Communications

When a satellite is involved in at least one part of a telecommunication link, this is called satellite communications (SATCOM). While telephony used to be the main focus of interest formerly, the "predominate current use today is to support Internet Protocols" [3], while Internet browsing dominates the traffic [5]. The properties of SATCOM links highly depend on the kind of satellites that are used to establish the connection. The different kinds are explained in the next sections.

2.1.1 Orbits

Most artificial satellites are orbiting the Earth in a pre-defined elliptical orbit which follows the Kepler's law of planetary motion. The law sets the dimensions of the

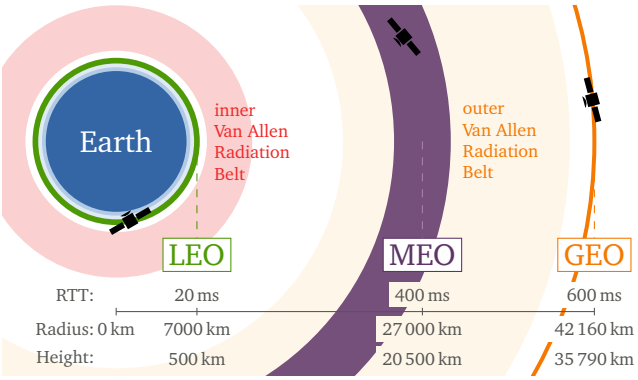


Figure 2.1 – The Most Important Earth Orbits:
Geostationary Earth Orbit (GEO), Low Earth Orbit (LEO), Medium Earth Orbit (MEO), and the inner and outer Van Allen radiation belts, zones of energetic charged particles, which makes it impracticable for satellites.

axes in relation to the speed of the satellite. The following parameters define an orbit around an astronomical body [2]:

- Inclination (i)
- Longitude of the ascending node (Ω)
- Argument of periapsis (ω)
- Eccentricity (e)
- Semi-major axis (a)
- Mean anomaly at epoch ($M0$).

For SATCOM, the most important parameter is a, the dimension of the semi-major axis. As orbits are often almost circular ($e \approx 0$), a equals the radius of the orbit. To simplify it further, we can also use the altitude above the surface of Earth, because right now all communication satellites are orbiting the Earth. The required speed of the satellite depends on a and is higher for lower orbits.

That being said, only three categories of orbits are important for SATCOM. They are visualized in Figure 2.1 and explained in the following sections.

2.1.1.1 Low Earth Orbit

The nearest orbit to the Earth's surface is the Low Earth Orbit (LEO) with an altitude below 2000 km. Additionally, orbits below 450 km are called Very Low Earth Orbit (VLEO) orbits [6]. The small distance to Earth leads to a low signal attenuation and additionally to a small propagation delay for electromagnetic signals. This makes it interesting for SATCOM because depending on the exact scenario the delay is comparable to terrestrial optical fibers.

The following equations calculate the propagation delays d_{prop} of an exemplary communication of two endpoints on Earth via a LEO satellite and a terrestrial fiber.

The endpoints for the SATCOM scenario are close the point on Earth's surface directly below the satellite. This point is called nadir. We set the height of the satellite to $h = 750\,\text{km}$ in Equation (2.1). The signal must travel this distance twice. For Equation (2.2), a fiber length of 1000 km is used with speed of light $c_{\text{fiber}} \approx \frac{2}{3} \cdot c_{\text{free}}$ with $c_{\text{free}} \approx 3 \cdot 10^8\,\text{m/s}$ [7]:

$$d_{\text{prop, LEO}} = \frac{h \cdot 2}{c_{\text{free}}} \approx \frac{750\,\text{km} \cdot 2}{3 \cdot 10^8\,\text{m/s}} = 5\,\text{ms} \tag{2.1}$$

$$d_{\text{prop, fiber}} = \frac{\text{length}}{c_{\text{fiber}}} \approx \frac{1000\,\text{km}}{2 \cdot 10^8\,\text{m/s}} = 5\,\text{ms} \tag{2.2}$$

On the other hand, there are some reasons that made this orbit less interesting for many years: The *high velocity* of the satellites required by Newton's law (see Section 2.1.1.3) which makes it hard to align the customer's antenna and which leads to distortion because of the Doppler effect. As the line of sight persists only for minutes, *handovers* between satellites are required as soon as the one, the ground station is currently connected to, crosses the horizon. The shifting relative positions also result in a *high variance* of delays [8]. Additionally, due to the small distance a large *amount of satellites* is required to cover a significant area of the Earth's surface and thus reach many customers. This makes it costly as well as complex since all of them must be arranged in a stable constellation.

However, changes occurred in the last few decades. The space has been commercialized which drives down costs. Phased array antennas have developed and can be used to readjust the antenna beam to follow the satellite's trajectory. And once deployed the large amount of satellites forms a very fault-tolerant network. *Starlink* by *SpaceX* is a major player in trying to provide Internet access via LEO using mega-constellations that consists of thousands of satellites. With promising data rates between 50 and 150 Mbit/s at a low delay of 20 to 40 ms[3], new opportunities are arising: People that nowadays have no access to the Internet might be supplied with high speed Internet one day, provided that the monthly costs are reduced in the future. It is also discussed integrating satellite services into cellular communication standards [6].

But also new challenges emerge with mega-constellations: To achieve good performance routing within the satellite network is required [6]. Therefore, satellites are usually equipped with Inter-Satellite Link (ISL) using laser beams. From protocol perspective, issues like out-of-order delivery caused by the highly varying network have to be addressed [3]. And of course the political questions exists, who is allowed to launch such a large amount of satellites and how to handle space debris. The latter one has to be avoided to prevent chain reactions of uncontrolled collisions

[3] Announced specifications on the website of Starlink: `https://starlink.com/` *(visited on 2022-01-05)*

of objects in space. This effect is called Kessler Syndrome [9] and poses a severe danger to space activities.

2.1.1.2 Medium Earth Orbit

Everything between LEO and GEO is considered to be a Medium Earth Orbit (MEO). While often used by navigation satellites like *GPS*, *GLONASS*, *Galileo* and *BeiDou* it is rarely used for communication satellites. The larger delays compared to LEO and the variable ground track of MEO satellites makes this orbit less suitable for communication satellites. But there are also advantages: Caused by the larger distance, much fewer satellites are required to cover the Earth's surface and handovers have to take place less frequently, e.g., every 45 min for *O3b*. It is operated by the space company *SES* and deployed at an altitude of 8063 km [3]. With only 20 satellites it provides broadband Internet connectivity to the "other three billion" people without Internet access. To summarize, using MEO is a trade-off between GEO and LEO with acceptable latencies and a limited number of satellites required.

2.1.1.3 Geostationary Earth Orbit

The Geostationary Earth Orbit (GEO) is used to place satellites on a fixed longitude above the Earth's equator. This property is very useful for SATCOM because ground antennas can be installed statically, and no automatic tracking is required. The inclination of the orbit has to be $a = 0°$ and the direction of rotation has to equal the Earth's rotation. The radius of the orbit is $r_{\text{GEO}} \approx 42\,160\,\text{km}$, which results from the balance of the centrifugal and centripetal forces. The first one is caused by the circular movement of the satellite and the latter one is the Newton's gravity. Combining both formulas results in Equation (2.3) which can be resolved to the radius, as shown in Equation (2.4) [2]. For GEO, the standard gravitational parameter of the Earth $G \cdot M = \mu \approx 3.986 \cdot 10^{14}\,\text{m}^3/\text{s}^2$ and the sidereal day $T_{\text{GEO}} \approx 1436\,\text{min}$ has to be used:

$$T_{\text{GEO}} = 2\pi\sqrt{\frac{r_{\text{GEO}}^3}{G \cdot M}} \tag{2.3}$$

$$\Rightarrow r_{\text{GEO}} = \sqrt[3]{\frac{G \cdot M \cdot T^2}{(2\pi)^2}} \approx \sqrt[3]{\frac{3.986 \cdot 10^{14}\,\text{m}^3/\text{s}^2 \cdot (1436\,\text{min})^2}{(2\pi)^2}} \approx 42\,160\,\text{km} \tag{2.4}$$

With the radius of the Earth being about $r_{\text{Earth}} \approx 6370\,\text{km}$ the altitude above the surface is $h_{\text{GEO}} \approx 35\,790\,\text{km}$.

A big advantage for SATCOM of GEO compared to the other orbits we introduced before, is that in theory only three satellites are required to cover the entire Earth's surface, except the polar regions which can't be covered using GEO satel-

lites. Practically often six satellites are used because while in theory the beam of one satellite covers 40 % of the surface, in only 20 % of the area the signal meets the desired quality [8]. The footprint is also large enough that no handovers are required. The large distance from earth however leads to high signal attenuation which makes communications challenging. Also installing new satellites in that orbit is very expensive owing to the distance.

Nevertheless, this orbit is one of the most desired ones for broadcasting and communications because of the fixed location of the satellites above the Earth. According to the Union of Concerned Scientists Satellite Database[4] there are currently 565 GEO satellites arranged on a ring on the equatorial plane, making this area very crowded. Therefore, it is very costly to get a license and an assigned position. That is the reason GEO satellites are built for long lifetimes and high reliability [2]. To be able to provide many customers with one satellite, spot beams are used, while each beam serves one area on the ground. Neighbored beams use different frequencies in the Ka-band (27–40 GHz) which are reused with enough distance. The total capacity of modern High Throughput Satellites (HTSs) is in the range of some 100 Gbit/s to some Tbit/s. It is increasing for newer generations [10].

2.1.2 Components of an IP SATCOM System

The task of a classic GEO satellite basically is to receive signals, amplify, modulate them to another frequency and retransmit them to earth. As it usually operates on the physical layer and in particular does not process packets, a communication channel via such a satellite is referred to as *bent pipe*. Links are established bidirectionally, both from the providers Source Ground Station (SGS) to the User Segment (referred to as *forward link* or *downlink*) and in the reverse direction (referred to as *return link* or *uplink*). Connections between forward and return links differ to the extent that

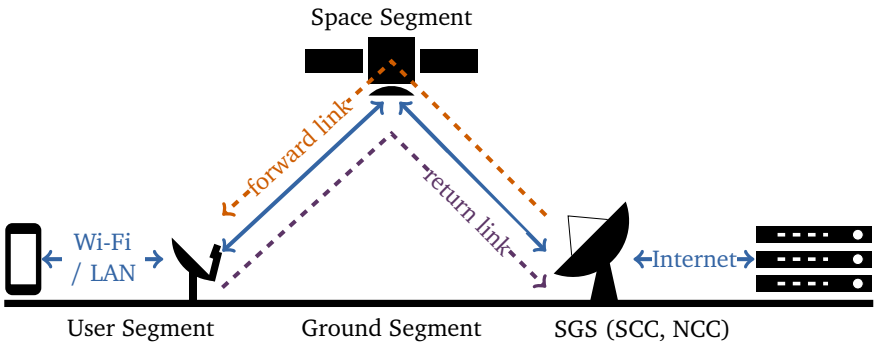


Figure 2.2 – Main Components of a SATCOM System

[4]Union of Concerned Scientists Satellite Database: `https://www.ucsusa.org/resources/satellite-database#.W7WcwpMza9Y` *(visited on 2022-01-04)*

all connections in the forward link can be managed centrally. E.g., Time Division Multiple Access (TDMA) can be used to multiplex the connections in the forward link, whereas other methods must be used in the return link [3]. While both antennas on the *Ground Segment* are parabolic antennas, satellite operators usually use larger ones to achieve a better directivity. Besides of the antenna, customers of IP services are equipped with a modem offering carrier-grade NAT or public (legacy) IPv4 addresses. IPv6 addresses, as they would be state of the art, are rarely provided when using consumer-grade tariffs [5]. Users connect to the modem using Ethernet or Wi-Fi. On the provider side there are usually two operation centers: The Network Control Center (NCC) providing the connection to the Internet backbone (also called *feeder* in the broadcasting sector) and the Satellite Control Center (SCC) which controls the satellite by sending and receiving Telemetry, Tracking and Control (TT&C) commands.

2.1.3 Characteristics of Network Links via GEO Satellites and Mitigations

IP networks using GEO satellites differ from terrestrial networks in some properties. The most relevant ones are described in the following sections.

2.1.3.1 High Delay

The most relevant characteristic of a GEO link is the remarkably high latency caused by the propagation of electromagnetic signals. Similar to Equation (2.1) we can calculate the minimum delay from one ground station to the other by using the height of the satellite $h_{\text{GEO}} \approx 35\,790\,\text{km}$ in Equation (2.5). The minimum Round-Trip Time (RTT) for GEO links calculated in Equation (2.6) is this delay doubled since it affects the forward and the return link.

$$d_{\text{prop, GEO}} = \frac{h_{\text{GEO}} \cdot 2}{c_{\text{free}}} \approx \frac{35\,790\,\text{km} \cdot 2}{3 \cdot 10^8\,\text{m/s}} \approx 240\,\text{ms} \tag{2.5}$$

$$RTT_{\text{min, GEO}} = 2 \cdot d_{\text{prop, GEO}} \approx 480\,\text{ms} \tag{2.6}$$

When both GSs are not located directly below the satellite but at the edge of the coverage, the signal travels $2 \cdot 41\,760\,\text{km}$, which results in an RTT of roughly 557 ms [3]. However, the endpoints usually experience an RTT of at least 600 ms caused by additional processing delays like Forward Error Correction (FEC) encoding, which is explained in Section 2.1.3.5. Extending the path at both sides (e.g., by counting in the hops from the providers Ground Station (GS) to the target web server and by using a LAN on the users side) further increases the delay. At times

when the utilization of the network is high, even RTTs of more than 1 s are possible (see Section 2.1.3.3).

2.1.3.2 Asymmetric Data Rates

As mentioned before, space in the geostationary orbit is limited and only one satellite is responsible for quite a large part of the earth. Although HTSs have an overall high capacity in total, it has to be shared with many users. Additionally, the available frequency spectrum, allocated by the International Telecommunication Union, Radiocommunication Sector (ITU-R), is limited but has to be used in the forward and the return link. Adapted to the typical usage profile the bandwidth and thus the available data rates in the forward link is usually larger than in the return link, which results in faster downloads than uploads [3]. A characteristic asymmetry factor is ten. For instance the Astra access, we have used, advertises 20 Mbit/s in the forward link and 2 Mbit/s in the return link (see Section 4.2.2.1). The data rate assigned per user in the forward direction is typically in the range of 1 to 100 Mbit/s [10]. While usually more data is transferred via the forward link, the return link is always used for signaling (see Section 2.1.3.5). When the asymmetry gets too high, `ACK` traffic reduction mechanisms can be used to achieve high download rates [3].

2.1.3.3 Variations in Delay and Data Rate

One of the advantages of GEO above LEO is that there is not a high jitter caused e.g., by mobility issues and handovers between satellites, given that there are no moving obstacles in the line of sight. Yet the signals must pass through the atmosphere which makes the link quality and capacity dependent on the weather. It was observed that heavy snowfall degrades the throughput and reliability of the connection [11]. Apart from that, the throughput is also influenced by the utilization of the shared satellite, which is usually higher during the prime time between about 6 p.m. and 11 p.m. [11]

A reason for this is that packages are queued in network devices before they are being processed and forwarded. Historically the maximum size of these queues grew proportionally to the development of the network speed. As the average length of queues increases on high demand, nowadays, large queuing delays may occur, which leads to less throughput per user. This phenomenon is referred to as **Bufferbloat** [12]. It was predicted by John Nagle in 1985 [13] and became relevant more than 20 years later.

2.1.3.4 High Bandwidth Delay Product

Given the unidirectional delay and the data rate we can calculate the so called **Bandwidth Delay Product (BDP)**[5]. This product is often used to classify networks because it defines the amount of unacknowledged data a transport protocol is allowed to have "in flight", which in return defines the minimum size of send and receive buffers [14] [15]. Additionally, the BDP equals the size, a transmitter could send, instead of waiting for feedback from the client, which is a lot for GEO networks. See Section 2.1.3.6 for information about congestion control and Automatic Repeat-Request (ARQ) protocols. However, it should be mentioned that the BDP alone does not describe a network sufficiently. Networks with the same BDP but different delays differ greatly.

The BDP for satellite links is larger, than for terrestrial fiber networks, for its higher delay, as shown in Equations (2.7) and (2.8). Such networks are generally referred to as high BDP networks or Long Fat Networks (LFNs) [3] [16][6].

$$\mathrm{BDP}_{\mathrm{GEO}} = d_{\mathrm{prop,\,GEO}} \cdot R \approx 240\,\mathrm{ms} \cdot 50\,\mathrm{Mbit/s} = 1.5\,\mathrm{MB} \tag{2.7}$$

$$\mathrm{BDP}_{\mathrm{terr}} = d_{\mathrm{prop,\,fiber}} \cdot R \approx 5\,\mathrm{ms} \cdot 1\,\mathrm{Gbit/s} = 625\,\mathrm{kB} \tag{2.8}$$

Figure 2.3 explains the relationship between BDP and buffer sizes. On the left side the transmit-buffers and on the right side the receive-buffers are displayed for four points in time. Between the channel fill state is visualized. Both buffers have an ideal size of $2 \cdot \mathrm{BDP}$. At the start of the transmission the channel and the

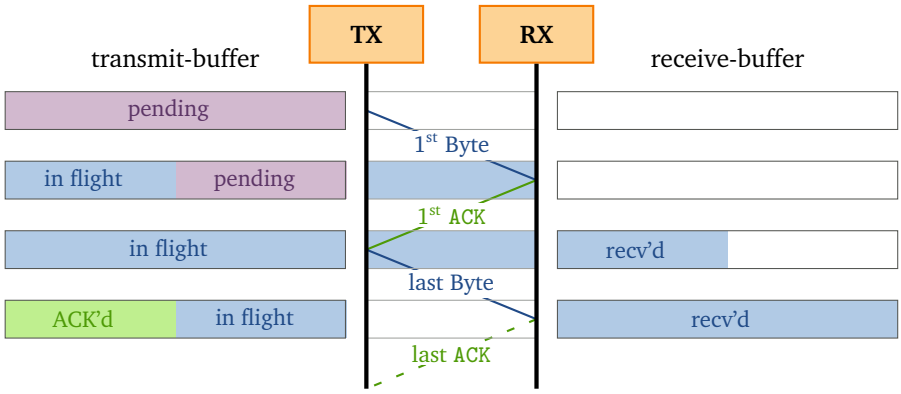


Figure 2.3 – Sequence Diagram of a Transmission Between a Transmitter (TX) and a Receiver (RX) with Ideal Buffer Sizes According to BDP

[5] Usually we prefer using the word "data rate" instead of "bandwidth" for the speed of transferring data because bandwidth is used for frequency spectrum. We make an exception for "Bandwidth Delay Product" because "data rate delay product" is not used in literature.

[6] It should be noted that although according to RFC 1072 both networks are referred to as high BDP networks, they can hardly be compared. Instead, the threshold value of $1 \cdot 10^5$ bit in RFC 1072 is outdated.

receive-buffers are empty. When the first packet arrives at the receiver the channel is entirely filled with $1 \cdot \text{BDP}$ Bytes and the receiver transmits the first `ACK` back to the transmitter. At the time this `ACK` arrives at the transmitter, all data in the transmit-buffer is sent, but not yet acknowledged. At this moment the transmitter stops sending new data in the example. When the last packet arrives at the receiver, the ideal receive-buffer is entirely filled, while `ACKs` for only the half of the transmitted data arrived at the transmitter.

2.1.3.5 Packet Errors

Since packet errors and losses are to be expected with every link, transport protocols that implement ARQ are used on the Internet. However, since DVB-S2 is used on the physical layer, losses are quite rare on satellite links under good conditions. It provides a quasi error-free (QEF) channel by adapting the modulation rate depending on the link quality and by exploiting FEC with variable coding rate [10]. Thus, the Bit Error Ratio (BER) exposed to upper layers is usually less than $1 \cdot 10^{-7}$ [3]. But the satellite path is usually only one part of the end-to-end link from the user to the server. Users often use Wi-Fi to connect to the modem. While there are countermeasures to reduce loss rates in such wireless networks as well, the error rates there can become very high, and errors usually appear in bursts. A simplified Gilbert-Elliot [17] loss model can be used to emulate packet errors for such a network, at which a probability of $P(b|g) = 0.018$ to switch from the "good" to the "bad" state is often used [18]. Retransmitting packets via the satellite link takes very long because of the long RTT. If a packet gets lost and has to be retransmitted once, the time delta between first transmission and successful reception is 1.5 RTT, which is at least 900 ms.

2.1.3.6 Congestion Control

There are two mechanisms that control the transmission rate and influence the utilization of the channel: *Flow control* and *congestion control*. The *flow control* depends on how fast the receiver can process the data buffered in the receive-buffer. It can be used to signal the sender that it should throttle the transmission rate, when the receiver is too slow. Using a satellite connection instead of a terrestrial has virtually no effect on it.

This is different to *congestion control*, which is used to avoid overload in the network. Its efficiency usually highly depends on the RTT because it exploits information inferred from `ACK` packets sent from the client to the server to estimate the bottleneck bandwidth[7] of the channel. Once estimated, the congestion window (cwin) is adapted in order to control, how many bytes are "in flight", i.e.,

[7] Again, we are using the term "bottleneck *bandwidth*" because "bottleneck *data rate*", which would be correct, is not used in literature

sent but not yet acknowledged. If the bottleneck bandwidth is underestimated, the channel is not used to full capacity; if it is overestimated, buffers overflow and the network gets congested (see Section 2.1.3.3). Finding the optimum is quite hard because it can change at the moment, when a new flow starts or stops competing for data rate. After the *congestion collapse* in 1986 [19], when no Congestion Control Algorithm (CCA) has been used on the Internet, plenty of algorithms have been developed. They are visualized in Figure C.1 in the appendix. All of them usually start transmitting with a small initial congestion window (iwin) that is increased (usually exponentially) upon the reception of ACKs. This mechanism is called *slow start*. It works well on networks with small RTT, when the transmitter receives the ACK shortly after transmitting the packet. On links with high delays, like GEO links, conventional CCAs may need very long to increase the cwin, given that they do not run in timeouts when such high RTTs are not expected.

Regarding decreasing the cwin due to congestion there are two major categories of CCAs as shown in Figure C.1: *Loss based*, which are the classic ones, and *delay based*, which are more modern (see Section 2.2.4). Loss based CCAs, like (New)Reno and CUBIC, start to reduce the transmission after experiencing packet losses, i.e., when the buffers overflow (see Sections 2.2.1 and 2.2.3). The late reaction to the overloaded network has severe impact on the performance: Usually the transmission rate is heavily reduced (e.g., by factor $\frac{1}{2}$) and slowly increased upon reception of ACKs [19], which takes much time on GEO links, as said before. Another problem of loss based algorithms is that every packet error is taken as indication of congestion. But especially in wireless networks errors might also be caused by temporary effects on physical layer like collision with packets of other clients.

In this context, it is nowadays common to use more verbose Selective Acknowledgements (SACKs) instead of classic cumulative ACKs. They are defined as an extension for TCP [20]. Traditional cumulative ACKs are used, to signal the successful reception of all previous packets until a given packet number. If the sender gets a cumulative ACK for a packet number less than the maximum packet number it has transmitted and when it has to assume that the packet got lost, it has to retransmit the lost packet and all subsequent packets. SACKs allow to indicate the reception of ranges of packets, while single lost packets in between can be re-requested selectively. This allows the sender to continue sending new packets in addition to the lost ones, rather than repeatedly sending packets that have already been successfully received. Thus, the performance of the transport protocol over LFNs with a high BDP can be significantly increased.

2.1.3.7 Performance Enhancing Proxies

When end-to-end CCAs are used, lost packets are always retransmitted via the satellite link, even when the error happened in the user's local Wi-Fi network. This leads to large delays and unnecessary utilization of the satellite link. Additionally, as mentioned before, TCP implementations with conventional CCAs usually do not perform well on such connections. These are the reasons, why satellite operators deploy Performance Enhancing Proxies (PEPs) (sometimes also referred to as Performance Enhancement Proxies, split proxies or split-TCP) [21]. It can be differentiated between one-sided and two-sided (distributed) PEPs, while one-sided ones are often used in cellular communications and two-sided in SATCOM. Except PEPsal [22], which is open-source, most PEP implementations are proprietary, and it is unknown, how they work exactly. However, the main idea is to split the connection between client and server into multiple smaller paths with one CCA control loop for each segment instead of one loop for the end-to-end connection. For two-sided satellite PEPs, one proxy is deployed in the user's modem and the other one at the gateway

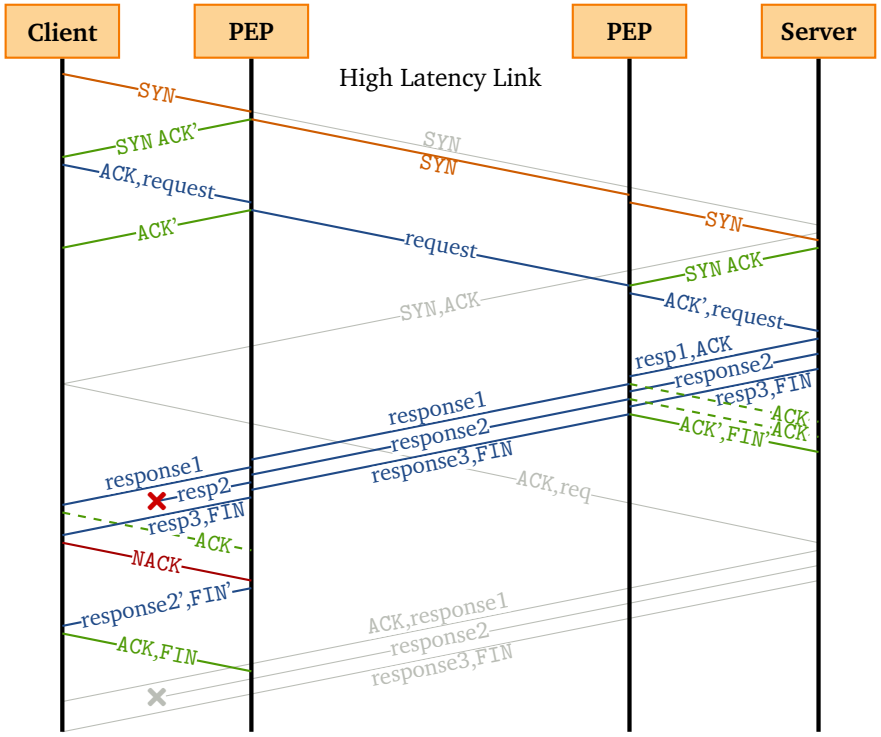


Figure 2.4 – Basic Functionality of a PEP. `SYN`, `ACK` and `FIN` refer to names of flags in TCP headers. `ACK`s and `SYN`s are spoofed, data is buffered in PEPs and retransmissions are handled locally. The transmission without using PEPs is drawn shaded and clipped because it would take more than twice as long.

of the provider. An exemplary accelerated TCP communication is visualized in Figure 2.4. PEPs can work transparently for TCP because the transport information is entirely unencrypted and unauthenticated. Thus, even users that may know the presence of a satellite service, but do not deploy special applications, can benefit from them. While ordinary CCAs can be used on the first and last segments, multiple measures can be taken in between by either modifying the TCP headers or wrapping the packets or payload into proprietary frames:

Local loss recovery: Buffer every packet locally and retransmit from the local buffer, which is much larger than ordinary transmit-buffers, upon requests for retransmissions.

Promise signaling: "Spoof" `SYN`s and `ACK`s to make the endpoints send data permanently to fill local buffers.

Suppress Duplicate Acknowledgements (DupACKs): When retransmissions are handled locally, the congestion controller on the client side gets no DupACKs, avoiding heavy reduction of transmission rate, as explained in Section 2.2.1.

Use an adapted CCA, that is optimized for the satellite scenario, like TCP Hybla, which is explained in Section 2.2.2, or CCAs with higher timeouts, larger iwin and larger maximum cwin

Compress data to reduce bandwidth.

Keeping TCP connections open, that would be closed because of larger timeouts, especially when the link degrades for a longer period of time.

Prioritize traffic.

In theory, TCP traffic and transport layer information could also be encrypted to secure it against eavesdropping that is possible because of the large spot beam footprint, as explained in Section 2.1.1.3. However, no operator seems to have implemented this [8].

While PEPs lead to a high speedup, like related research in Section 3.2 shows, it is often criticized that they are violating the end-to-end principle of the transport layer [23]. On the one hand this is acceptable because no endpoint modification is required. On the other hand, this leads to *ossification* of Internet protocols. By making special assumptions on the TCP behavior and packet format, new features, like TCP RACK-TLP [24], TFO (Section 2.3.2.2) and ECN (Section 2.3.2.6) are not supported [3]. PEPs are also limited to TCP, while VPN connections, UDP traffic and modern transport protocols, like QUIC (see Section 2.3), can't be accelerated.

2.1.3.8 Pacing

Another important measure at the transport protocol level that should be mentioned is pacing [25]. It is used for both terrestrial and satellite links, and is especially relevant,

when the BDP of the network is high [26]. When an ACK arrives at the sender, the congestion controller can initiate the transmission of new data. If this data is sent at a higher rate than the bottleneck bandwidth, so-called microbursts can occur, which overload the network for a short time and cause buffer overflows, resulting in severe throttling of the transmission rate. To avoid this, the pacing mechanism is added after the congestion controller to smooth the packet rate. Especially in satellite connections, where packet losses and retransmissions are expensive, data should always be sent at a rate that is not higher than the estimated bottleneck bandwidth.

2.2 Important Congestion Control Algorithms

The question of the performance of transport protocols is often a question of the applied CCA. Therefore, a brief overview of the most significant algorithms for general purpose transport protocols in the SATCOM context is presented in the following section. Most of them are built for TCP while theoretically they can also be ported to QUIC. It has to be mentioned that different CCAs exist for multimedia streaming protocols. Unlike generic transport protocols, such as TCP and QUIC, these are able to adjust the source data rate, for example by transmitting lower resolution videos.

2.2.1 NewReno

As successor of Reno, which is one of the first CCAs, NewReno [27] belongs to the classic algorithms. The main contribution of Reno [28] is to skip slow start and use fast retransmit followed by a *fast recovery* phase, when three *Duplicate Acknowledgements (DupACKs)* arrive at the sender. NewReno additionally specifies the usage of partial ACKs to allow signaling gaps in a succession of successfully transmitted packets. While both have only a little practical relevance nowadays it is still used for academic purposes, and it is even chosen for some QUIC implementations because of its simplicity, as it is shown in Table 4.2.

2.2.2 Hybla

To overcome limitations of NewReno for high-latency links, Hybla [29] works similar to its predecessor but modifies the calculation rules for the increase of cwin by making them dependent on the estimated RTT. It was not widely adopted by satellite operators because they already relied on PEPs [30].

2.2.3 CUBIC

As networks developed first BIC, then CUBIC was invented for fast long-distance networks [31]. Compared to NewReno only the sender side changed and uses a cubic instead of a linear function over time to increase the cwin. It is now the default CCA in Linux and seems to reach fair performance over satellite links [30]. To overcome the limitations of slow start phase, **HyStart++** is currently being developed which exploits heuristics "to exit slow start early while also mitigating poor performance" [32]. The QUIC implementation of Cloudflare, called *quiche* uses this combination[8].

2.2.4 BBR

In order to make the Internet faster, Google has developed a new algorithm called BBR, which stands for "Bottleneck Bandwidth and Round-trip propagation time". The model based approach takes the experienced delay into account when estimating the network utilization. When buffers are filled more, the delays get higher (see Section 2.1.3.3). Thus, higher delays are an indication of higher network load. Work is currently ongoing to standardize of BBRv2 that also considers ECN flags, which are explained in Section 2.3.2.6, and packet losses [33]. Under some circumstances it outperforms CUBIC on satellite links, especially when the Packet Loss Rate (PLR) is high [34]. For even better performance in the start-up phase, it can be optionally used in combination with HyStart. Typically, new CCAs need to be more aggressive than those already in widespread use in order to be deployed on a large scale. However, BBR is pretty fair to CUBIC [34] and yet is already in use on more than 18 % of the Alexa Top 20 000 websites [35]. A weakness of BBR seems to be the periodic probing phase, where throughput periodically drops for a short time, granting data rate to other competing connections [34].

2.3 QUIC

Even before BBR, Google started to work on a new transport protocol, called QUIC, to replace the ossified and aging TCP. While the name was usually used as acronym for "Quick UDP Internet Connections"[9], the Internet Engineering Task Force (IETF) decided that "QUIC is a name, not an acronym" [36].

After giving a short overview of the biggest milestones in history of QUIC, we introduce features and extensions that are most relevant for SATCOM.

[8] Cloudflare blog post announcing to use CUBIC with HyStart++ in *quiche*: `https://blog.cloudflare.com/cubic-and-hystart-support-in-quiche/` *(visited on 2021-12-22)*

[9] Presentation of QUIC by Jim Roskind at the conference IETF 88 (transport area): `https://www.ietf.org/proceedings/88/slides/slides-88-tsvarea-10.pdf` *(visited on 2021-12-06)*

2.3.1 History of QUIC

2.3.1.1 Prehistory

For many years, there was no other relevant transport protocol deployed on the Internet than TCP (for reliable transmission) and UDP (for unreliable transmission). In 2000, SCTP was standardized [37] with many progressive features. However, apart from WebRTC, which uses it as substrate protocol for data channels on top of UDP, it was not widely adopted [38], but many principles have influenced the design of QUIC later.

Also, with the desire in mind to make web browsing faster, Google developed the Application protocol SPDY (pronounced “speedy”), which was later adopted by the IETF and standardized as HTTP/2 [39]. Among many features that also have not been broadly adopted, it introduces multiplexing of multiple streams using the same TCP connection. QUIC adopted this connection pooling.

2.3.1.2 Google QUIC

Google started development of QUIC in 2012 as part of the Chromium project and in 2013 they already made early field studies [40]. It was designed as substrate protocol for future generations of HTTP, with “the goal to provide secure and reliable low-latency end-to-end transport” [38]. By in turn using UDP as slim substrate protocol, it benefits from its wide-ranging support. As consequence the entire QUIC stack has to be implemented as user space application on common systems. This is justified by the fact that updates can be deployed faster. Similarly to HTTP/2 it provides reliable bidirectional byte streams that allow multiplexing multiple requests on a single connection. Better security compared to TCP is achieved by encrypting the header as far as possible. This also hides transport information from middleboxes in order to enforce the end-to-end principle of the transport layer. To further avoid ossification, future extendibility is ensured by being part of the design. Many modern techniques that are state of the science have been implemented, which enables QUIC to perform better than TCP, under the most common circumstances (see Section 2.3.2) [40].

Until end of 2016 Google experience already about 30 % of its inbound traffic to be QUIC while it was not even standardized yet [40].

2.3.1.3 IETF QUIC

In order to create an open standard, the project was transferred to IETF in 2015[10]. Major tasks were modularizing the components of the design and replacing the Google crypto, which was embedded into QUIC, with TLS1.3 [41] [42], on which

[10]First draft of QUIC standard: `https://datatracker.ietf.org/doc/draft-tsvwg-quic-protocol/00/` *(visited on 2021-12-06)*

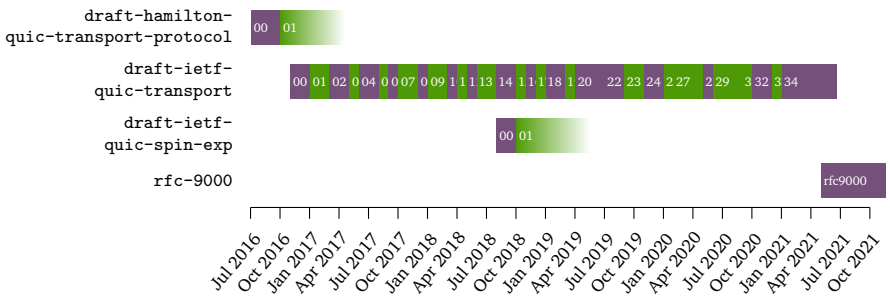


Figure 2.5 – The Timeline of RFC 9000, the Main RFC of QUIC

standardization work was going on at the same time [40]. In 2016, the IETF founded the new QUIC Working Group to coordinate the work on the new protocol, which underlines the importance of this project. Over time, everything was adapted once again, so that in the end no detail, no bit order was the same as Google's original design. Only the basic principles were adopted. Although the work had been going on for a long time, it was not until 2017 / 2018 that the first satellite operators realized that the encryption of QUIC hides required information from PEPs, which causes QUIC to be inefficient over satellite links[11]. In May 2020, work on QUICv1 was considered to be done, and it was released as RFCs 8999 to 9002 [36]. Until then, it was already deployed by many other companies, like Cloudflare, which was already seeing 12 % of traffic running over QUIC[12].

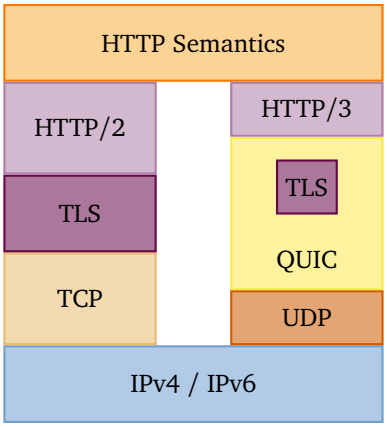


Figure 2.6 – Protocol Stack of HTTP/3 compared to HTTP/2

The official standard for mapping HTTP semantics on QUIC is still pending, but is about to be released as HTTP/3 [43]. The entire web protocol stack, using UDP as substrate protocol of QUIC, TLS1.3 and HTTP/3 on top of it, is displayed in Figure 2.6.

The emergence of QUIC shows that global players do increasingly more research and development than academic staff. If the latter ones are still involved into standardization at all, it is only when it is wanted by industry. That is why participation

[11] Presentation by Tom Jones at *2nd QUIC and Satellite Open Stakeholder Meeting* on 2021-12-02 `https://erg.abdn.ac.uk/video/2nd%20ESA%20MTAILS%20Satellite%20Stakeholder%20Meeting/1.1%20Evolution%20of%20QUIC%20and%20Satellite%20over%20the%20Last%203%20Years.pdf` *(visited on 2022-01-03)*

[12] Blog post of Lucas Pardue from Cloudflare, one of the main contributors to QUIC, about the release of QUICv1: `https://blog.cloudflare.com/quic-version-1-is-live-on-cloudflare/` *(visited on 2021-12-06)*

in standardization bodies, such as the IETF, is very important for universities and research institutions. And, of course, the experience of companies is warmly welcomed. In this case, satellite internet operators have not yet been involved, but their concerns should nevertheless be taken into account.

In the following, the term QUIC is used as a synonym for IETF QUIC as of RFC 9000, and gQUIC as a synonym for the older and by now obsolete Google QUIC.

2.3.2 Features of QUIC and their Relation to SATCOM

Like already mentioned, QUIC comes with a lot of progressive techniques, and it was designed with state-of-the-art mechanisms in mind. Not all features that are listed here are new. However, their adoption is facilitated by the fact that QUIC in contrast to TCP is new and not ossified and designed to be extensible. Some presented features are not yet standardized, but work is already in progress.

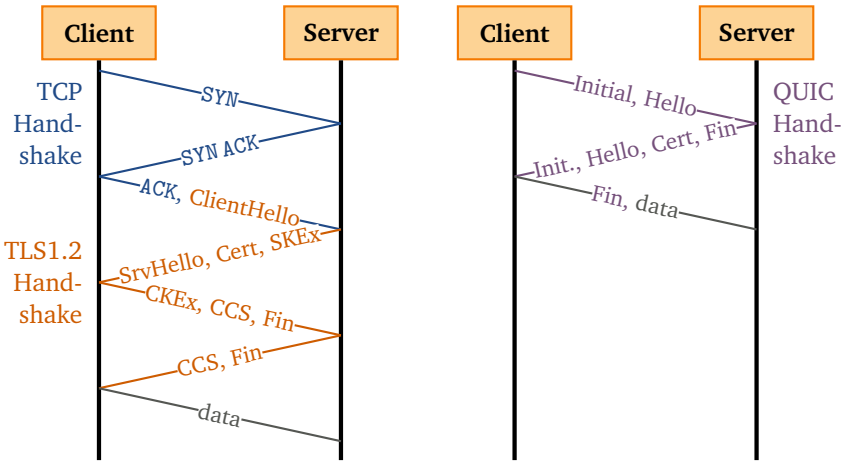
2.3.2.1 Eliminating Head-of-Line-blocking

With the principle of multiple streams at transport layer, QUIC solves the issue of Head-of-Line-blocking (HoLB). As TCP strictly maintains the order of packets, following problem arises: When multiple resources are being transmitted in parallel and if packet loss prevents the transmission of one from proceeding, then all streams are affected and cannot proceed. This is not the case for QUIC since the reliability mechanisms are applied on stream level, which allows for example that a website is being transmitted while the transmission of a single embedded object, like an image, encounters packet loss. This design approach was copied from SCTP, and also HTTP/2 tried to address this issue but only solved it on application layer because TCP is used as substrate protocol.

On connections with high delays, like satellite connections, solving this issue enhances the Quality of Experience (QoE), since it is perceived as if websites load faster [4].

2.3.2.2 1-RTT, 0-RTT and 0-RTT-BDP Handshakes

To reduce the Time to first Byte (TTFB) it is crucial to open connections quickly. TCP requires a 3-way handshake (consisting of `SYN`, `SYN-ACK`, `ACK`, while the last `ACK` packet can already carry data). TLS1.2, which is deployed on top of TCP, when in use, requires in the default case an additional 2-RTT handshake to negotiate cryptographic parameters. Thus, it takes at least $3 \cdot RTT$ to open a connection (see Figure 2.7a). QUIC manages to open connections using 1-RTT handshakes by allowing attaching data to the last packet of the handshake and by integrating TLS1.3 into the transport layer [36] [42] (see Figure 2.7b). TLS1.3 requires only



(a) TCP and TLS handshake **(b)** QUIC 1-RTT handshake

Figure 2.7 – Significantly Improved Time to Perform a Default Handshake in QUIC Compared to TCP with TLS by Using a 1-RTT Handshake.
SKEx: Server Key Exchange; *CKEx*: Client Key Exchange; *CCS*: Change Cipher Spec; *Cert*: Transmission of Certificate;

one round-trip for new connections and zero round-trips for subsequent ones. In addition, QUIC can be extended to allow resumed connections to be initiated with a so-called 0-RTT handshake, which means that parameters from previous connections are stored and reused. Both endpoints agree on them in the first packet while data is already attached to it. A mechanism similar to QUIC's 0-RTT was already developed for TCP as TCP Fast Open (TFO) [44]. However, due of the ossification of the Internet it is not yet widely deployed.

In the satellite context with one RTT being at least 600 ms, the time for a default handshake could be reduced from 1800 ms to 600 ms, which is remarkable and seems to have a big impact at least for small websites and files (see Section 3.2).

As described before, ramping up the congestion control packet rate and estimating path parameters, like BDP, takes some time, especially on paths with high delay, such as satellite links. However, the parameters are required to allow efficiently utilization of the path. The estimations could be skipped on connection resumption if the endpoints used previously estimated parameters again. To achieve this, work is underway on an extension called 0-RTT-BDP [45]. The idea is, to negotiate previously determined parameters during connection establishment.

2.3.2.3 Path MTU Discovery

In order to optimally utilize the path, estimating the Maximum Transmission Unit (MTU) supported by the end-to-end link by using Path MTU Discovery (PMTUD) is

highly recommended[13]. The RFC allows choosing between explicit PMTUD, which uses ICMP messages and is defined for IPv6 [46] and IPv4 [47], and Datagram Packetization Layer PMTUD (DPLPMTUD), which uses probe packets [48]. These techniques already existed before, but as they are highly advised for QUIC, they will more likely be adopted. When satellite operators choose to provide a high MTU, good QUIC implementations will recognize it and achieve a better performance.

2.3.2.4 `ACK` Decimation

There are two drafts, [49] and [50] that are currently being worked on to decrease the `ACK` traffic in cases where the return path is limiting the performance because of the asymmetry of the link, as explained in Section 2.1.3.2. The idea is to define a ratio and send only one packet with `ACK` frames per n data packets or at fixed time intervals. While this is not yet ready to use, the future extendibility of QUIC makes it at least possible to implement these features.

2.3.2.5 Improved Congestion Control

Many details in the congestion control area have been tuned, but the principles are not new. E.g., QUIC makes use of already existing CCAs, but does not define which ones to use. However, the modularized design makes it possible to switch them easily and deploy more modern ones, like BBR (see Section 2.2.4), when the implementation supports it.

Among others the following things were adjusted:

Eliminating the retransmission ambiguity: When a sender retransmits a packet in TCP it uses the same packet number again. Thus, the receiver can not determine, whether an `ACK` packet belongs to the original or the retransmitted packet, and it can not estimate the current RTT. This is solved by using strictly monotonic packet numbers, even for retransmissions [25].

Increasing the `ACK` range: This allows to re-request packets more precisely [25]. For TCP, there are only three SACK ranges which is very little for high BDP paths with high loss and leads to dreaded timeouts.

Pacing: Pacing is explained in Section 2.1.3.8. It is now strongly advised, to use it in combination with QUIC's congestion controller[14].

Probe Timeout: Following the design of Recent Acknowledgement-Tail Loss Probe (RACK-TLP [24]), QUIC uses a mechanism called Probe Timeout (PTO) to send probe packets after a certain timespan, when it suspects that a packet has been lost. This is especially important at the end of a transmission, when no

[13] "An endpoint SHOULD use DPLPMTUD or PMTUD" [36]

[14] "A sender SHOULD pace sending of all in-flight packets based on input from the congestion controller." [25]

subsequent packets are sent anymore and losses can't be detected by holes in `ACK` ranges.

2.3.2.6 Explicit Congestion Notification

As mentioned in Section 2.1.3.6, loss based CCAs use packet losses as indication of congestion. To avoid this level of escalation and to make CCAs throttle their transmit rate earlier in times of Bufferbloat, Active Queue Management (AQM) algorithms like Random Early Detection (RED) [51] or derivates of it have been proposed to actively drop packets out of the queue when the load increases. However, this measure still seems relatively harsh to many network administrators, which is why it was rarely activated in the past[15].

Another mechanism to signal congestion more explicitly is Explicit Congestion Notification (ECN) [53]. Again, this already exists independently of QUIC, but RFC 9002 makes the usage of ECN messages mandatory for QUIC [25]. When the network devices support it, they can set the `ECN-CE` ("congestion experienced") flag in the IP header of packets that are sent to the receiver, when the buffer levels are high. The QUIC implementation at the receiver side will in turn inform the sender to throttle down the transmission rate.

2.3.2.7 Packet Level Forward Error Correction

Not yet standardized, but suggested multiple times ([54], [55]), is the idea to introduce Forward Error Correction (FEC) on the packet level. A coding rate $R_C = \frac{k}{n}$ can be defined to complement data packets with parity packets. The scheme ensures that when $(n-k)$ parity packets are added to k data packets, and at most $(n-k)$ packets get lost or corrupted, all data can be reconstructed at the receiver. Whether the additional data overhead is useful, must be decided on a case-by-case basis, since FEC is already present on the physical layer (see Section 2.1.3.5). This can be useful for example for real-time applications where a round-trip dependent recovery is unacceptable [55]. And especially in the SATCOM context it can help to further reduce expensive retransmissions by improving robustness of single streams.

[15] Jim Gettys summed up reasons, why RED was often not activated in 2010: `https://gettys.wordpress.com/2010/12/17/red-in-a-different-light/` *(visited on 2021-12-11)*. Another reason might be that the paper "RED in a Different Light" by Jacobson et al. [52] was never accepted because it contains an "offensive image" of a toilet.

Chapter 3

Related Work

> Those who cannot remember the past are condemned to repeat it.
>
> *George Santayana*

As we have seen in Section 2.1.3.7, PEPs can not accelerate QUIC because all the relevant transport layer information is encrypted and thus inaccessible for middleboxes. Without any external tuning, it is up to QUIC to determine path properties reliably and to perform well. In this chapter, we first explain common terms and metrics and then review previous research findings on the performance of QUIC in SATCOM context. Evaluations of QUIC via ordinary links are not part of this summary.

3.1 Terms and Metrics

Because QUIC was designed as transport protocol for browsers, the built-in browser metrics are used for many measurements. The metrics follow the interface specification of *Navigation Timing*[16]. It defines points in time that are relevant for the loading process of a website. Some of them depend on the performance of the transport protocol. Figure 3.1 visualizes a very simplified loading process of a website. On the timeline on the client side, the relevant points in time as defined in the specification are written in orange. Alternative terms used in contexts other than browser metrics are highlighted in purple. The points in time for the server side are not officially defined, and we assigned custom names.

[16] W3C® *Navigation Timing* Specification: `https://www.w3.org/TR/navigation-timing/` *(visited on 2021-12-17)*

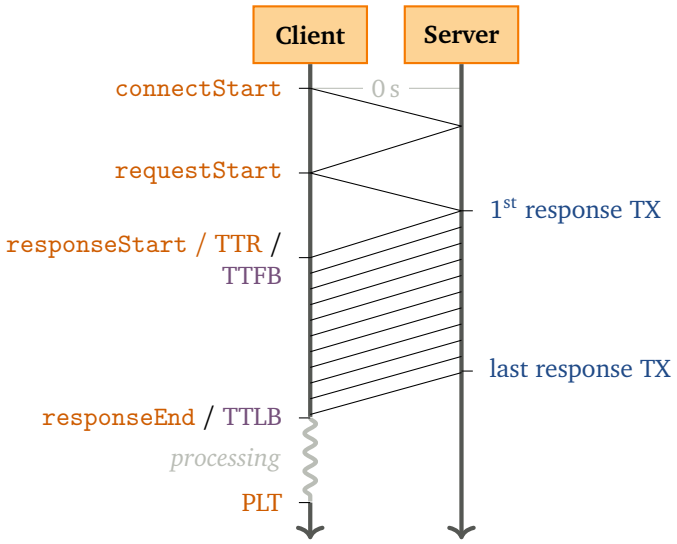


Figure 3.1 – Common Metrics for Transport Protocol Measurements in the Context of Browsers

When a user navigates to a new website, the browser will do plenty of tasks, such as querying the local cache or resolving DNS names, which are neglected here. After that, the browser will open a new connection on the transport layer to the target server at $t =$ `connectStart`. Normally, the client can send the HTTP request to the server about one RTT later at $t =$ `requestStart`. The server will then process the request and send back the first response packet at $t = 1^{st}$ *response TX*, which arrives at $t =$ `responseStart` at the client. This point in time is also referred to as *Time to* `responseStart` *(TTR)* or *Time to first Byte (TTFB)*. After that, the requested data is being transferred to the client. The last packet is sent out at $t =$ *last response TX* and arrives at $t =$ `responseEnd`, which is also called *Time to last Byte (TTLB)*. Depending on the browser, the data will already be processed during reception. The processing is done at $t =$ *Page Load Time (PLT)* and the website is displayed completely.

For modern interactive websites with dynamic content, however, these metrics have hardly any relevance for the performance as it is perceived by users. In that case, especially the PLT does not correlate with the user-perceived speed [4]. Instead, there are metrics that take visual impression of websites into account, for example the points in time of the first and last visual change of the page or when the page is loaded for 85 % because then it is often perceived as completely loaded. To collect these metrics, entire systems consisting of proper browsers and representative websites, have to be measured, and the results have to be analyzed statistically. For more low-level approaches, it is usually enough to transmit test data and measure the *time to completion*. During the transmission, the gross data rate, i.e., the rate of

transferred user data, which is also referred to as *goodput*, is also representative for the performance of the transport protocol.

3.2 Related Papers and Measurements

Table 3.1 gives a very compact overview of relevant research on this topic. A more verbose version of this table is attached at Appendix D and also available online[17].

Table 3.1 – Overview of Related Research on QUIC Performance via Satellite Links.

Reference	QUIC Version	Client	Server	Type of Benchmark	Type of Link
Ⓐ Rula 2018 [56]	?	?	?	Websites	emul.
Ⓑ Yang 2018 [57]	gQUIC Q035	*chrome*	*quic-go*	Website	emul.
Ⓒ Zhang 2018 [58]	gQUIC Q039	*chrome*	Google QUIC test server	Websites	emul.
Ⓓ Wang 2018 [34]	gQUIC Q039	*chrome*	Google QUIC test server	Websites	emul.
Ⓔ Thomas 2019 [59]	gQUIC Q039	*chrome* 67	Google Server	File	real
Ⓕ Deutschmann 2019 [5]	gQUIC Q043	*chrome* 69	*chrome* *quic-go*	File	real
Ⓖ Fairhurst 2019 [60]	draft-20	*quicly* v20		Files	real
Ⓗ Wolsing 2019 [4]	gQUIC Q043	*chrome* 70	Google QUIC test server	real Websites	emul.
Ⓘ Mogildea 2019 [61]	Q046 draft-22 draft-22	*chrome* *quicly* *ngtcp2*		File	real, emul.
Ⓙ Border 2020 [62]	gQUIC Q046	*chrome* 77	Google Drive	File	real, emul.
Ⓚ Kuhn 2020 [18]	gQUIC ?	*chrome* 67	Google Server	Websites	real
Ⓛ Custura 2020 [63]	draft-27 draft-26	*quicly* *chrome*		File	real, emul.
Ⓜ Kuhn 2021 [64]	?	*picoquic*		File	emul.

[17] Full table of related research at GitHub: https://github.com/sedrubal/QUIC_HIGH_BDP/blob/master/research_overview.md

Published papers and presentations at meetings of the IETF have been taken into account. The table and the sections below are ordered chronologically and linked with the letter in the first column. A "gQUIC" entry in the *version* column means that the old Google QUIC was used (see Section 2.3.1.2), while versions starting with "`draft-`" reference the version of the IETF draft of QUIC (see Section 2.3.1.3). We are not yet aware of any research that uses QUICv1. "Website(s)" in the column *Type of Benchmark* indicates that real world or artificial websites with embedded objects have been transferred using QUIC, while "File(s)" means that only single objects of a given size have been transmitted. In the column *Type of Link*, we differentiate between emulated or real satellite links. Information that could not be obtained is marked with "Ⓘ".

3.2.1 Ⓐ Mile High WiFi: A First Look At In-Flight Internet Connectivity

In this paper, Rula et al. analyze in-flight communication systems, which are currently deployed to provide Wi-Fi in airplanes and usually perform poorly [56]. One technology, called Mobile Satellite Services (MSS), uses satellites as relays to connect to the GS. This causes challenging path characteristics with an RTT of roughly 760 ms and a loss rate of 6 %. (Symmetric) data rates are usually at about 1.89 Mbit/s. Compared to satellite connections, where the user's equipment is fixed, the loss rate seem to be higher in the airplane scenario and the data rate seems to be lower than what is usually offered by providers. The test bed for the measurements consists of an emulated link using NetEm (see Section 4.1.2). The CDFs of PLTs of downloads of different artificial web pages is compared between HTTP/1.1, HTTP/2 and QUIC. Rula et al. conclude that QUIC outperforms HTTP/1.1 in large parts, and HTTP/2 performs poorly at high loss rates. However, apart from proxies that cache content, PEPs do not seem to have been considered.

3.2.2 Ⓑ Performance Analysis of QUIC Protocol in Integrated Satellites and Terrestrial Networks

Yang et al. simulate several network scenarios involving terrestrial and satellite links to assess the performance of QUIC in such integrated networks [57]. The networks contain LEO and GEO satellites and are implemented in MATLAB and the Satellite Tool Kit (STK). A range of RTTs up to 500 ms are simulated, which is less than the usually observed RTTs. The data rate is 10 Mbit/s at different loss rates in the range of 0 to 3 %, which is quite realistic. Google Chrome is used as client and *quic-go* as server to establish the gQUIC connection. Again, the resulting CDFs of PLTs are compared to HTTP/2 over TCP. The conclusion is that QUIC outperforms TCP in all

scenarios, except during handovers in LEO networks because of a faster handshake. However, this conclusion is questionable, as the complexity of the simulated network is very high and important parameters and configurations are either not given or confusing. Additionally, the comparison to TCP is misleading because most likely PEPs were not used in these test environments.

3.2.3 Ⓒ How Quick Is QUIC in Satellite Networks

Zhang et al. claim to provide the first measurements of QUIC performance via satellite links [58]. They use NetEm to emulate GEO links with RTTs of 200, 400 or 600 ms, data rates of 256 kbit/s, 512 kbit/s or 1 Mbit/s and different BERs. Except of the slightly low data rate, these values are appropriate. Again, Google Chromium is used as client and the Google QUIC test server that was part of the `proto-quic` repository, as server, to download websites of different sizes. Unfortunately, the resulting PLTs are compared to an HTTP stacks using TCP without PEP. This puts the finding into perspective that QUIC outperforms HTTP/TCP, especially when propagation delays are large and PLRs are high.

3.2.4 Ⓓ Performance Evaluation of QUIC with BBR in Satellite Internet

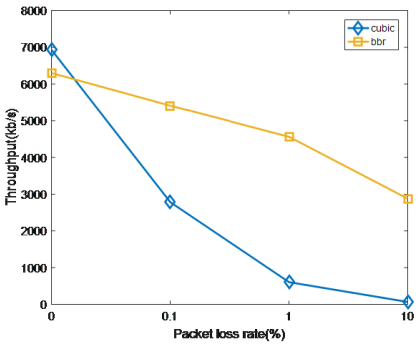


Figure 3.2 – Throughput of QUIC with CUBIC vs. QUIC with BBR at different PLRs at a data rate of 10 Mbit/s, a file size of 10 MB and an RTT of 600 ms. Taken from [34]

This is a follow-up paper of Ⓒ, but now with the focus on gQUIC in combination with the CCA BBR (Section 2.2.4) instead of CUBIC (Section 2.2.3), as in the previous measurement [34]. Wang et al. use the same setup as before, but they further increase the BER up to $2 \cdot 10^{-1}$ and increase the data rate up to 10 Mbit/s. This time, the achieved goodput is compared between Chrome with BBR and Chrome with CUBIC, with the result that BBR outperforms CUBIC by far in high loss scenarios (see

Figure 3.2). However, the periodic probing phases of BBR reduce the performance significantly, and they give CUBIC the opportunity to steal data rate, when both are in competition with each other, like Wang et al. show in a second test bed.

3.2.5 Ⓔ Google QUIC performance over a public SATCOM access

Thomas et al. provide measurements in a realistic scenario by using a real GEO satellite access (KA-SAT PRO25Go), Google Chrome as client, the Google 404 website (with embedded objects) and some image served by Google (5.3 MB) as payload, and a PEP-setup to accelerate TCP connections that is used in production. By enabling TFO (see Section 2.3.2.2) in the "HTT" stack (HTTP/2, TLS1.2, split-TCP), the protocol stacks are comparable. 0-RTT in the "HQU" (HTTP/2, gQUICv39 with BBR, UDP) stack is enabled, but was not used. The satellite link has 25 Mbit/s in the forward link and 5 Mbit/s in the return link at an RTT of 750 ms. Browser metrics (PLT, TTR and elapsed time) are used to compare the protocol stacks, and every measurement was repeated on a 4G connection for reference. Additionally, sequence-number plots are provided (see Section 5.6.1). The authors conclude that the performance of QUIC in the satellite scenario is poor because of the non-delegated congestion controller, and the faster connection establishment does not compensate this issue.

3.2.6 Ⓕ Satellite Internet Performance Measurements

Another paper using real satellite links is this one by Deutschmann et al. [5]. The main focus is on evaluating the performance of different HTTP versions via three different end user satellite Internet solutions (Avanti PLC, SES Astra and Eutelsat Tooway). The asymmetric links have data rates of 6/2 Mbit/s or 15/30 Mbit/s (forward / return link capacity), but they were black box tested to assess the Quality of Service (QoS) as experienced by users. Measurements show that all operators reach the advertised throughput values, while 2 operators show high UDP delay variations, which results in bad QoS for QUIC. Different combinations of connections with or without TLS, with different HTTP versions or QUIC, and with or without VPN as substrate protocol, have been tested. The authors assume that PEPs accelerate TCP connections without VPN. For QUIC, Google Chromium was used as client and the Google Chrome test server and *quic-go* were used as servers. Some users of satellite internet use OpenVPN to encrypt the traffic in order to mitigate the security threat presented by large satellite footprints. However, it also prevents PEPs from accessing TCP headers. Both a large file and artificial websites are used as payload and boxplots of PLTs are compared. Plenty of insights can be drawn from the results shown in Figure 3.3: The two more round-trips required by TLS1.2 are clearly visible compared to unencrypted HTTP/1.1. When using VPN below TCP, the

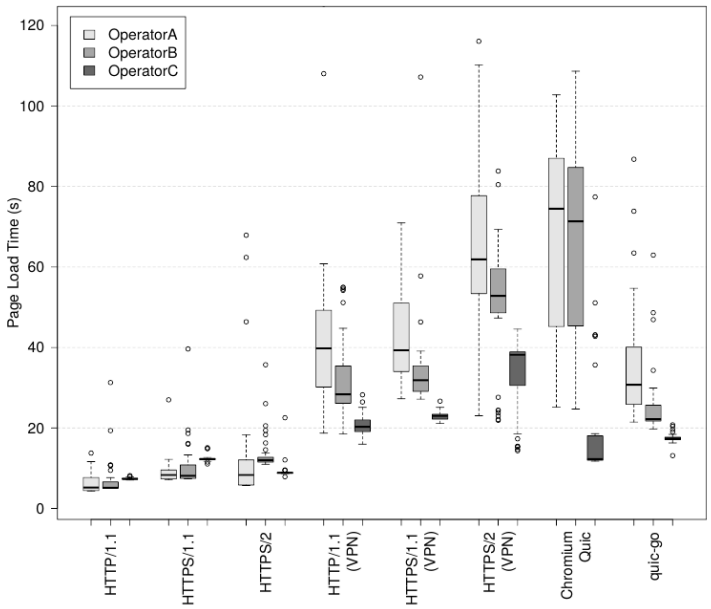


Figure 3.3 – PLT of a Large-Sized Website Using Different Protocols and Operators. Taken from [5]

performance is worse than the performance of plain TCP connections accelerated by PEPs. One operator could be identified who throttles the data rate of single flows, which leads to worse performance of HTTP/2 compared to HTTP/1.1 because browsers create only a single TCP flow per server, when HTTP/2 is used. Providers should consider this limitation for QUIC flows as well. The PLT for QUIC connections is—as expected—worse than the PLT for TCP via PEP connections, but better than for TCP via VPN connections, which also uses UDP as substrate protocol.

3.2.7 Ⓖ Measuring QUIC Dynamics over a High Delay Path

Fairhurst et al. presented interesting measurement results at the IETF 105 conference, but we are not aware of an associated scientific publication [60]. This is the first measurement in this collection which uses an early draft version of IETF QUIC instead of gQUIC. They measured the elapsed time and created packet-number-vs-time-plots (see Section 5.6.9.2) for transmitting files of different sizes using *quicly* (with Reno), TCP (with CUBIC, SACK and TLS1.2 / TLS1.3) and TCP (same configuration) via OpenVPN. Again, a real satellite link was used, which was measured beforehand. The observed throughput is 8.5/1.4 Mbit/s and the RTT is 639 ms on average. The impact of the faster handshake of TLS1.3 could be verified for small transfers and also the advantage of plain TCP over QUIC in presence of PEPs.

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

Comparing different protocols is difficult because the results highly depend on the level of optimization of the implementations. On account of this, Wolsing et al. use an optimized version of TCP, which is already quite similar to QUIC, and compare it with Google QUIC (Chromium & Google Chrome test server) [4]. Following features have been used and optimizations have been done on the TCP stack: TLS1.3 (with shorter handshakes as TLS1.2); BBR with pacing; increased iwins; tuned buffers; slow-start-after-idle disabled. The main focus of the paper is not only SATCOM, but various challenging scenarios, which also include MSS, as it was described in Ⓐ. The same path parameters are used again to simulate the network and again no PEP is used for TCP connections. By simulating web browsing on real websites using the Mahimahi framework, visual browser metrics, which reflect the user's perception, are collected. The main result is that QUIC still outperforms the optimized TCP, as far as browser metrics are concerned because of the reduced RTT design and the elimination of HoLB. And in the MSS scenario, where a PLR of 6 % is common, BBR performs better than CUBIC.

3.2.9 Ⓘ QUIC over Satellite: Introduction and Performance Measurements

To highlight the differences between different QUIC implementations and also between different satellite operators, Mogildea et al. present a test matrix of three real satellite links (SES Astra, Avanti PLC and Eutelsat Tooway) plus one emulated connection (using dummynet, see Section 4.1.2) and three QUIC implementations (*chrome*, *quicly* and *ngtcp2*) plus an HTTP/2 stack using TCP [61]. While *chrome* uses the old gQUIC, the latter ones are using an early draft version of IETF QUIC. The implementations also differ in the CCA. The providers advertise their links with a data rate between 16 to 30 Mbit/s in the forward link and 2 to 3 Mbit/s in the return link. All of them have PEPs installed. The emulated link is configured for an RTT of 600 ms and an asymmetric data rate of 20/2 Mbit/s. Sequence/offset-number-plots (see Section 5.6.1) are provided for the transmission of a file of 1 MB for a loss-free and a lossy scenario with a PLR of 1 %. While the performance depends strongly on the implementation and also a bit on the provider, it decreases significantly for implementation in the lossy scenario.

3.2.10 Ⓙ Evaluating QUIC's Performance Against Performance Enhancing Proxy over Satellite Link

Like others did before, Border et al. present satellite link measurements of the in the meantime outdated gQUIC with HTTP/2 and PEP-accelerated HTTP/1.1 [62]. This time, very large file downloads (1 GiB) from Google Drive, where QUIC was already deployed in production at that time, are analyzed by using box plots of the achieved goodput. Using servers that are not under own control has the disadvantage that only limited information about the configuration can be provided. Two different test beds are considered: One with real satellite equipment and a link emulator, which introduces losses, between the client PC and the client terminal, to emulate the case when Wi-Fi is used on this path. The second one is entirely emulated and does not contain a PEP. Both connections have an RTT of roughly 600 ms and the PLRs are between 0 and 1 %. The results are in line with the previous ones that gQUIC outperforms plain TCP connections, especially at higher loss rates, but it can by far not reach the performance of PEP-accelerated TCP connections.

3.2.11 Ⓚ QUIC: Opportunities and threats in SATCOM

With the target audience of Satellite Internet Service Providers (ISPs) in mind, Kuhn et al. deliver a strengths, weaknesses, opportunities, and threads (SWOT) analysis of end-to-end QUIC from satellite-operators point-of-view. [18]. In the paper, they also provide measurement results by using the same setup as described in Ⓔ. Different websites are downloaded via a real satellite link and sequence plots (see Figure 3.4) and PLTs are used to analyze the results. The analysis confirms previous findings that QUIC performs better than TCP for small files since fewer round-trips are required to establish the connection. For larger files, TCP outperforms QUIC by factor 2, while the TTR is slightly smaller for QUIC. QUIC with BBR seems to need more time to

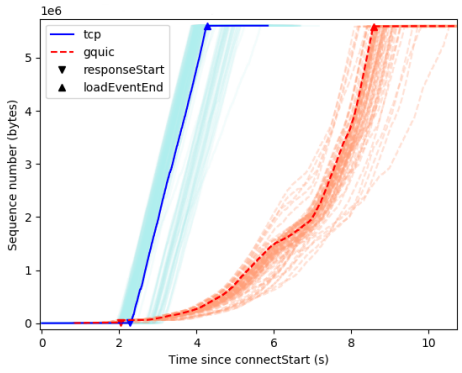


Figure 3.4 – Sequence Number Plots of TCP and QUIC Using a Real Satellite Link. Taken from [18]

get up to speed in loss free scenarios (4 s) because of the probing phase, while BBR with TCP takes only 50 ms as it only needs to probe the short link between the user and the PEP.

Regarding the SWOT analysis, Kuhn et al. draw following conclusions:

Strengths: QUIC brings performance improvements for small pages due to the reduced time required for handshakes.

Weaknesses:

- QUIC can not be made to recover losses locally (e.g., by using PEPs), and losses are first detected one round-trip later.
- Congestion and flow control at endpoints is sometimes unsuitable because thus link capacity is reached very slowly.

Opportunities:

- PEPs can be removed.
  - ⋆ Reduces cost of network infrastructure.
  - ⋆ New features can be deployed faster.
  - ⋆ End-to-end security is improved
  - ⋆ Resiliency of mobile scenarios is increased.
- Extensibility of QUIC

Threats: ISPs lose control of performance, which could lead to operators locking out QUIC traffic.

3.2.12 Ⓛ Impact of Acknowledgements using IETF QUIC on Satellite Performance

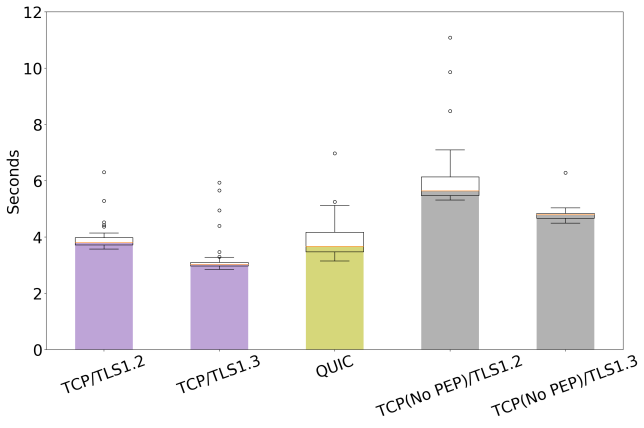


Figure 3.5 – Download Time for a 100 kB File Using TCP and QUIC Over a Broadband Satellite IPOS Service. Taken from [63]

In order to analyze the impact of `ACK` rates on the performance of IETF QUIC via asymmetric links, Custura et al. present measurements over emulated and real

satellite links [63]. *Quicly* (with Reno) and Google Chromium (with BBR), which switched to IETF QUIC at that time, are used as client and server. The emulated link has an RTT of 600 ms, an asymmetric data rate of 8.5/1.5 Mbit/s and a PLR of 1 %, while the real link was measured and has an RTT of about 630 ms, and an effective data rate of 8.5/1.5 Mbit/s. The results are compared to a TCP stack (Reno) with a PEP on the real satellite connection. While for the TCP stack HTTP/2 is used, the IETF QUIC stack uses HTTP/3 respectively an early version of it. The box plots provided for the PLTs for downloading a 100 kB file in Figure 3.5 are in line with the results shown before. Additionally, the authors conclude that a reduced rate of `ACKs` in the return link subsystem "can in some cases improve forward path performance, without impacting congestion control performance" [63].

3.2.13 Ⓜ Feedback from using QUIC's 0-RTT-BDP extension over SATCOM public access

The last measurement in this collection was presented by Kuhn et al. at the conference IETF 111 [64]. To evaluate the gain achieved by using the 0-RTT-BDP extension, which is currently being designed (see Section 2.3.2.2), they used *picoquic*[18] to transmit files in the range of 0.5 to 100 MB via a satellite connection that is emulated using the OpenBACH framework (see Section 4.1.1). Different RTTs up to 500 ms and a large range of data rates are emulated, and the elapsed time and the percentage of the utilized bandwidth are measured. The evaluation is very limited, and some questions remain unanswered since it is only a presentation and not a scientific publication. However, the effect of using 0-RTT instead of 1-RTT can be clearly seen and using 0-RTT-BDP instead of 0-RTT brings an additional gain of roughly the same magnitude. The gains, though, are highly dependent on the file size.

3.3 Summary of Related Measurements

Although several efforts have been invested to evaluate the performance of QUIC in the SATCOM context, the conclusions of many research papers are often limited: In many cases, either emulated or real satellite links are used, but they are rarely compared with each other. We also see that the test environments are very heterogeneous, and the metrics vary (visual browser metrics, time to completion, goodput). Even the methods of measurement hardly provide comparable results: Some use bulk

[18] *Picoquic* has support for the 0-RTT-BDP extension in two different variants:

- Storing parameters at the server: `https://github.com/private-octopus/picoquic/pull/1204`
- Using `BDP_FRAME`s: `https://github.com/private-octopus/picoquic/pull/1209`

(Both links visited on 2021-12-22)

transfers, others use websites, whereby particularly websites can be very different, as there is no such thing like a “typical website”. Some papers do not provide relevant parameters, like the CCA and other configurations. Another limitation is that mostly only single implementations were used, with Google Chrome / Chromium being the most common choice. Additionally, all implementations are under development and the QUIC protocol itself was not yet released for all measurements except the last. However, the results are always interpreted to be representative for QUIC as a protocol in general while the impact of differences between implementations is neglected, which is very high though, as we see in Chapter 5. While there have been many studies on the performance of gQUIC, few have used IETF QUIC, although a lot has changed: Web stacks with IETF QUIC use HTTP/3 instead of HTTP/2 and TLS1.3 instead of custom crypto. When comparing QUIC with TCP, one must also keep in mind that TCP has been around for many years and is more mature, while QUIC has not been optimized for so long. Furthermore, TCP has many parameters and extensions and comparing a new protocol with it is difficult, as the performance of TCP depends highly on the configuration.

However, most research draws the same conclusion that QUIC usually performs much better than TCP wrapped in an UDP-based VPN and slightly better than plain TCP, especially in lossy scenarios. But it performs much worse than PEP-accelerated TCP. The main reason for this seems to be the high delay of satellite links. For very small amounts of data, QUIC has an advantage because of the reduced handshake. The performance further depends on the CCA, while BBR seems to outperform CUBIC in lossy environments.

3.4 Suggested Improvements

Some papers provide suggestions on how the poor performance of QUIC via satellite links could be mitigated. Kuhn et al. recommend using a **more aggressive congestion control** on satellite links to ramp up the package rate much faster [18]. Based on the current experience, implementations should ensure using **0-RTT** whenever it is possible. Because of the high BDP, **increasing buffer sizes** helps to increase the link utilization. Furthermore, an efficient way of **local loss recovery** has still to be found. However, Kuhn suggests pushing forward **packet level FEC** (see Section 2.3.2.7), optionally in combination with proxies, as it will be explained in Section 6.1. Jones et al. recommend **increasing the maximum cwin** and the **iwin** and using **pacing** [3]. As soon as the return link becomes the limiting component, Custura et al. recommend **reducing the `ACK` ratio** [63]. For high loss scenarios, Wang et al. and others are suggesting **BBR** as CCA [34].

Chapter 4

Architecture

> Any sufficiently advanced technology
> is indistinguishable from magic.
>
> *Arthur C. Clarke*

As we saw in Section 3.3, there is a need in running comparable measurements with numerous QUIC implementations to assess the performance of the QUIC protocol. Therefore, an automated test bed and evaluation framework was developed, which is presented in this chapter.

4.1 Existing Tools and Benchmark Frameworks

Before describing our own work in Section 4.2, we briefly give an overview of existing tools and test frameworks that are commonly used for research on networking and protocol performance. Some of them form the basis of our work.

4.1.1 OpenSAND and OpenBACH

Open Satellite Network Demonstrator (OpenSAND)[19] and Open Benchmark Automation tools for Communication and Hypervision (OpenBACH)[20] are two open-source frameworks that are developed and often used by the French space agency cnes and Thales Alenia Space. Both tools are designed to be able to interact with each other.

OpenSAND is a satellite network emulator from the physical layer up. It covers the network stack from the physical layer like DVB-S2 upwards but can also be interconnected with real equipment and IP-based networks. It should act as a reference environment for scientific research with the focus on communications

[19] Project Homepage of OpenSAND: `https://opensand.org/` *(visited on 2022-01-03)*
[20] Project Homepage of OpenBACH: `http://openbach.org/` *(visited on 2022-01-03)*

engineering, and provides a user-friendly configuration and monitoring solution. However, it is generally unsuitable for research at the protocol layer because too many low-level details are taken into account, which sometimes cause higher layers to behave non-deterministically in our experiments.

OpenBACH can be used to supervise and control networks under test. It aims to provide modular, user-friendly and web-based tools to manage the deployment of SATCOM systems, and to design, configure, schedule, monitor and analyze benchmarks on them. Both real and emulated links (e.g., by using OpenSAND) are supported. For the nodes and components of the framework, either dedicated machines or VMs are required which seems very ponderous compared to the solution we present in Section 4.1.6. Under the hood, it uses production-grade open-source tools, like Ansible[21] for deployment and orchestration, Grafana[22], InfluxDB[23] and the ELK-stack[24] for data processing and result analysis. On the one hand, it is quite powerful and even sophisticated series of measurements can be configured, on the other hand, it supports only a limited set of benchmarks and implementations. We need to be able to run the same benchmarks with numerous implementations. While it would be possible to extend it, the benefit of having a graphical interface to configure the tests is not worth the work required.

4.1.2 NetEm and dummynet

NetEm [65] and dummynet [66] are two network emulators that act on packet level, which are similar in design and often used for research purposes, but also in production for bandwidth management. The first one is part of the Linux Kernel, the second one of FreeBSD and Mac OS X. Both can be used to emulate a link with poorer properties than the underlying physical or virtual one. The most relevant configurable parameters are data rate, delay and PLR. The networks can be configured using the command line tools `tc` or `ipfw`, which instruct the kernel to set up queues and pipes, respectively, with the desired properties. Because of their seamless integration into the systems both tools are a good choice for network emulation but require some knowledge about the network stacks of the kernels.

4.1.3 ns-3

ns-3[25] is an open-source discrete-event network simulator for Internet systems, which is designed to be used in research and education [67]. It provides a simulation core and modules that can be used to program custom network scenarios in C++ or

[21] Ansible: `https://www.ansible.com/` *(visited on 2021-12-11)*
[22] Grafana: `https://grafana.com/` *(visited on 2021-12-11)*
[23] InfluxDB: `https://www.influxdata.com/` *(visited on 2021-12-11)*
[24] ELK-stack: `https://www.elastic.co/de/what-is/elk-stack` *(visited on 2021-12-11)*
[25] ns-3: `https://www.nsnam.org/` *(visited on 2021-12-12)*

Python. Apart from routing algorithms and models for wireless communication, such as Wi-Fi, it also includes tracing capabilities allowing to record e.g., Pcap files. While it provides tools to set up entire virtual simulation environments, it can also be interconnected with the real world in a real-time mode, enabling the use of real-world protocol implementations. On the one hand, ns-3 is quite powerful, but on the other hand, it is rather complex to implement simple channels, like a point-to-point connection with a specific data rate, delay, and PLR. However, as it was already used by the QUIC-Interop-Runner presented in Section 4.1.6, we use ns-3 as network emulator in our setup.

4.1.4 Wireshark, TShark, Pyshark and Tcpdump

The Swiss army knife for network analysis, Wireshark[26], brings dissectors for gQUIC as well as IETF QUIC. While the support is still under development, the current version 3.6.0 has proven to be stable (and mostly feature complete) in our experiments. It can be used for live capturing and offline analysis of capture files, like Pcap and Pcap NG. Apart from the GUI program, there is also a set of command line tools, like `editcap` and `tshark`. We use the first one to bundle key-log and Pcap files into Pcap NG files, which makes it easier to handle them. Key-log files contain secrets and can be exported by most TLS libraries, like they are used by QUIC implementations. Wireshark can use them to decrypt QUIC headers and payload. There is a Python wrapper around `tshark`, called pyshark[27], which allows automated and headless analysis of capture files using the powerful dissectors and filters of Wireshark. We experienced some problems with it because parsing large traces is very slow and requires much memory.

In our decentralized test setup with real satellite links, we use tcpdump[28] instead of ns-3 to capture traces, as explained in Section 4.2.2. We also use it in the simulated setup for some experiments to overcome a bug in ns-3 tracing[29] that leads to cut-short Pcap files. Unfortunately, tcpdump has no support for filtering QUIC traffic as of version 4.99.1, which is not a big problem, as it allows filtering on protocol and port basis. More accurate filtering can be done using pyshark.

4.1.5 qlog and qvis

A downside of the encryption of QUIC is that it is harder to analyze and debug. Therefore, a uniform logging-format, called qlog, is currently being defined [68]. Client and server implementations need to implement support for it. The logs contain

[26] Wireshark: `https://www.wireshark.org/` *(visited on 2021-12-12)*
[27] pyshark: `https://github.com/KimiNewt/pyshark` *(visited on 2021-12-12)*
[28] Tcpdump: `https://www.tcpdump.org/` *(visited on 2021-12-12)*
[29] ns-3-issue on GitLab: `https://gitlab.com/nsnam/ns-3-dev/-/issues/102` *(visited on 2021-12-12)*

internal states and events of the endpoints, which are otherwise not accessible, when packets are intercepted on the wire, as it is the case for traditional packet capturing. While the specification is format-agnostic, mostly JSON is used as representation. And even if the logging-format was invented with QUIC as primary use-case, it is designed to be protocol-agnostic. The event definitions for QUIC are specified in [69]. By using a uniform and implementation independent format, it is possible to create reusable debugging tools and visualizations, like qvis[30]. This is a browser-based tool to generate a variety of visualizations for traces of QUIC and HTTP/3. It is for example possible to create interactive offset plots, like they are presented in Section 5.6.1, which are however very detailed and slightly overloaded. Using qlog for analyzing will be a good choice in the future because it works independently of Wireshark support, but right now not all QUIC implementations support logging of qlog-files reliably. And, as the specification is work-in-progress, not all implementations use the same version of the specification. Additionally, the timestamp of an event, when a packet is sent off or received, is generated in user-space, but the kernel or hardware may slightly delay the transmission and reception. This is not a problem when capturing packets on the wire.

4.1.6 The QUIC-Interop-Runner

In order to automate interoperability testing of publicly available QUIC implementations, Marten Seemann created the QUIC-Interop-Runner (QIR)[31]. Every client implementation is tested with every server implementation three times per day and several checks are performed, like if a handshake completes successfully, or if 0-RTT works (see Section 2.3.2.2). Additionally, very basic measurements have been added in order to assess the performance, when different implementations have to work together. The results are presented online as interactive tables. As we use this tool as a basis for our measurement environment, the following sections describe its method in more detail.

4.1.6.1 Terms and Mode of Operation

First, we define the terms that describe how the QIR works. The runner is invoked to fill the *result matrix*. This matrix has a column for each *server implementation* and a row for each *client implementation*. Thus, each cell belongs to a *combination of implementations*. For each *run* of the interoperability runner, either all or only a subset of *test cases* and *measurements* can be selected. *Test cases*, or simply *tests*, are of secondary relevance in this thesis, and we focus mainly on the *measurements*, as they are listed in Table 4.3. We also refer to them as *measurement test cases* or

[30] qvis: `https://qvis.quictools.info/` *(visited on 2021-12-22)*
[31] The QUIC-Interop-Runner: `https://interop.seemann.io` *(visited on 2021-12-13)*

scenarios because they define different scenarios implementations will be faced with. The *test cases* and *scenarios*, which are defined as Python classes, specify the desired setup, the link parameters, like data rate, delay and queue sizes, and the checks that are performed after the execution. As the *measurements* and *test cases* share the same setup code, we also sometimes refer to both simply as *test case*. In the following, we highlight the names of them using SMALL CAPITALS, to avoid confusion with other terms, like "goodput", which is sometimes used as the name of the *measurement* and sometimes as the metric of net data rate. We refer to the execution of a single *test case* with a tuple of implementations as *test run* or *experiment*. *Measurements* are repeated multiple times for the same *combination of implementations*. Every single iteration is also referred to as *experiment* or *measurement iteration*.

For each *test case* and each *combination of implementations*, a record is kept of whether it has been successful, unsuccessful or whether the *test case* is unsupported. For *measurement test cases*, the average of the achieved goodput over all iterations is recorded. The results are then added to the corresponding cell in the *result matrix*.

4.1.6.2 Architecture

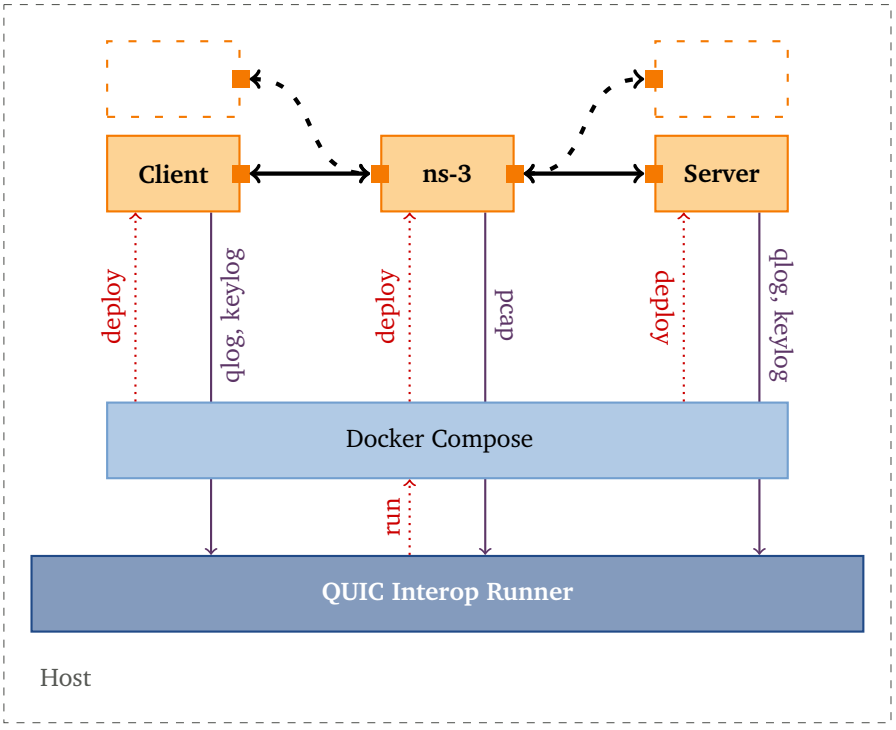


Figure 4.1 – Architecture of the Original QIR

The runner is open-source[32] and implemented in Python. It uses Docker[33] with virtual networks to deploy the entire test environment on a single host machine (see Figure 4.1). Most test cases require a setup of three containers: The QUIC client implementation, the QUIC server implementation and a link emulator in between. For special test cases, additional containers can be deployed, as it is the case for CROSSTRAFFIC where the performance is measured while other connections compete for data rate. Every creator of a QUIC implementation who wants its implementation to be added to the interoperability matrix must provide a Docker image that uniformly deploys their implementations. Parameters for the implementations are passed in through environment variables. Most images use wrapper scripts to translate the variables to command line flags for the implementations. Before each run, QIR checks the compatibility of each image, i.e., it checks, whether the images respect the environment variables and are using the correct exit codes. For the link emulator, different ns-3 scenarios (see Section 4.1.3) are used, which are also open-source[34]. When the tests are performed, Docker Compose[35] is used to deploy the scenario. Therefore, a `docker-compose.yml` template file is adapted by using environment variables. Once all containers are up and running, it relies on the images the creators of the implementations provide to use the TLS keys and certificates that are passed in through the volume `/certs`, and log to `/logs`. Servers have to serve files with random content from the volume `/www` and clients have to download them to the volume `/downloads`, preferably by using the very simple HTTP/0.9 protocol, which has a minimal overhead[36]. During the run, ns-3 is used to capture packets on the wire into Pcap files. The traces are then analyzed using pyshark and the results are exported to a JSON file.

Currently, Marten Seemann uses shared *GitHub Actions* runners to perform the official benchmarks. Getting accurate information about the infrastructure is difficult. According to the docs[37] Ubuntu 20.04 VMs are used with 2 CPU cores and 7 GB of RAM memory. The VMs are hosted in the Microsoft Azure cloud. Depending on the utilization of the machines, most notably on the CPU steal time, high variations in the measurement results are to be expected. Additionally, it is not known whether there are any modifications applied on the Kernel that are relevant for the performance of QUIC, like UDP-buffer parameters (`rmem_max`, `rmem_default`). Most likely it

[32] Repository of QIR on GitHub: `https://github.com/marten-seemann/quic-interop-runner` *(visited on 2021-12-13)*

[33] Docker: `https://www.docker.com/` *(visited on 2021-12-13)*

[34] Repository of Network Simulator for QUIC benchmarking at GitHub: `https://github.com/marten-seemann/quic-network-simulator` *(visited on 2021-12-13)*

[35] Docker Compose: `https://docs.docker.com/compose/` *(visited on 2021-12-13)*

[36] Discussion about using HTTP/0.9 as protocol: `https://github.com/marten-seemann/quic-interop-runner/issues/267` *(visited on 2021-12-13)*

[37] GitHub Actions Runner documentation: `https://docs.github.com/en/actions/using-github-hosted-runners/about-github-hosted-runners` *(visited on 2021-12-13)*

has no influence for functionality testing, but it is questionable if the measurement results can be compared with other measurements.

4.1.6.3 Implementations

At time of writing, QIR supports the most significant and publicly available implementations of QUIC, as listed on the wiki page of the IETF QUIC working group[38]. They are managed in a JSON inventory and shown in Table 4.1. All implementations with *both* in the column *Role* can work as client and as server, which is the majority. There are two exceptions: The popular browser *chrome*, which can only act as client, and the well-known web server *nginx*, which can only act as server. The leading `https://github.com/` was omitted for GitHub repository URLs. As we collected the versions by inspecting the Docker images by hand, not all versions could be determined for sure. These entries are marked with a (?). Hexadecimal versions with a leading `@` are commit hashes in the corresponding git-repository.

Table 4.1 – Implementations used in QIR

Name	Role	Repository on GitHub / URL	Version
aioquic	both	`aiortc/aioquic`	0.9.17
chrome	client	`marten-seemann/chrome-quic-interop-runner`	ChromeDriver 89.0.4389.23
kwik	both	`ptrd/kwik`	`@a9e478c` (Nov 2021, 03)
lsquic	both	`litespeedtech/lsquic`	(?)
msquic	both	`microsoft/msquic`	(?)
mvfst	both	`facebookincubator/mvfst`	(?)
neqo	both	`mozilla/neqo`	0.4.21
nginx	server	`https://quic.nginx.org/`	1.21.4
ngtcp2	both	`ngtcp2/ngtcp2`	0.1.0-DEV (?)
picoquic	both	`private-octopus/picoquic`	`@8dfc4776` (Oct 2021, 23)
quant	both	`NTAP/quant`	0.0.34 (?)
quic-go	both	`lucas-clemente/quic-go`	0.23.0
quiche	both	`cloudflare/quiche`	(?)
quicly	both	`h2o/quicly`	`@699e2564` (Nov 2021, 10)
xquic	both	`Kulsk/xquic`	(?)

[38]The wiki page of the IETF QUIC working group that lists important implementations of QUIC. `https://github.com/quicwg/base-drafts/wiki/implementations` *(visited on 2021-12-13)*

Table 4.2 – Implementations and their CCAs

Name	CCA	HyStart
aioquic	**NewReno**	✗
chrome	BBRv2, CUBIC	✓
kwik	**NewReno**	✗
lsquic	**BBR**, CUBIC	✗
msquic	**CUBIC**	✗
mvfst	BBR, **CUBIC**, NewReno, …	✓
neqo	CUBIC, **NewReno**	✗
nginx	?	✗
ngtcp2	BBRv2, BBR, **CUBIC**, Reno	✗
picoquic	**BBR**, CUBIC	✓
quant	**NewReno**	✗
quic-go	CUBIC ?, Reno ?	✗
quiche	**CUBIC**	✓
quicly	CUBIC, **Reno**, pico	✗
xquic	**BBR**, CUBIC, Reno	✗

It has to be mentioned that some implementations are not built for performance. Some target energy efficiency in the context of IoT, others are built for debugging purposes, like the server implementation of *neqo*, and others are just drafts or proof-of-concepts. While they are not representative for the performance of the QUIC protocol itself, using multiple implementations is better than picking a single one and making general conclusions.

Accordingly, the choice of the CCA is quite diverse. While RFC 9002 [25] suggests using a modified version of TCP NewReno, it is also allowed to implement any other CCA (see Section 2.2). Many implementations, especially the more optimized ones, employ CUBIC or BBR or even support multiple algorithms, as it is show in Table 4.2. The default CCAs of each implementation, which is (most likely) used in our measurements, is written in **bold**. We determined the algorithms and defaults by analyzing the source code, the command line flags and the log output by hand. Thus, there might be some missing or wrong entries. Some implementations might also use modified versions of the algorithms we determined. Missing entries or entries with high uncertainty are marked with a ?.

4.2 Modifications to QUIC-Interop-Runner

In order to use QIR as a performance measurement test bed which also supports real links, some things had to be changed. In this section, we mention the most important modifications and new features we introduced.

4.2.1 New Simulated Measurement Test Cases

There is already one SATCOM related test case included in the official QIR. It is called LONGRTT, and it checks whether a QUIC connection can be established via a link with a large RTT of 1500 ms. This is significantly larger than RTTs, which usually occur on links via geostationary satellite links. However, in worst case scenarios such RTTs are possible, as explained in Section 2.1.3.3. The simulated symmetric delay, buffer queue sizes and data rate of 10 Mbit/s in both directions are artificial and unlikely to be found in a real scenario though.

In addition, there are only two measurement test cases, i.e., tests where the performance rather than the success of tests matters: One is called GOODPUT. In this case, the average goodput is measured for the transfer of a large file. We use these values as reference for our new measurements. The other one is CROSSTRAFFIC. There the goodput is measured in the case, when another transmission, in this case `iperf`, is competing for data rate. Because the channel parameters chosen for this scenario are not related to SATCOM and because the goodput highly depends on the type of competing traffic (CCA, number of parallel transmissions, ...), we disregard this measurement.

4.2.1.1 New ns-3 Scenario

To emulate links that are similar to satellite links, we had to implement an asymmetric ns-3 scenario, which supports setting different parameters in the forward and return

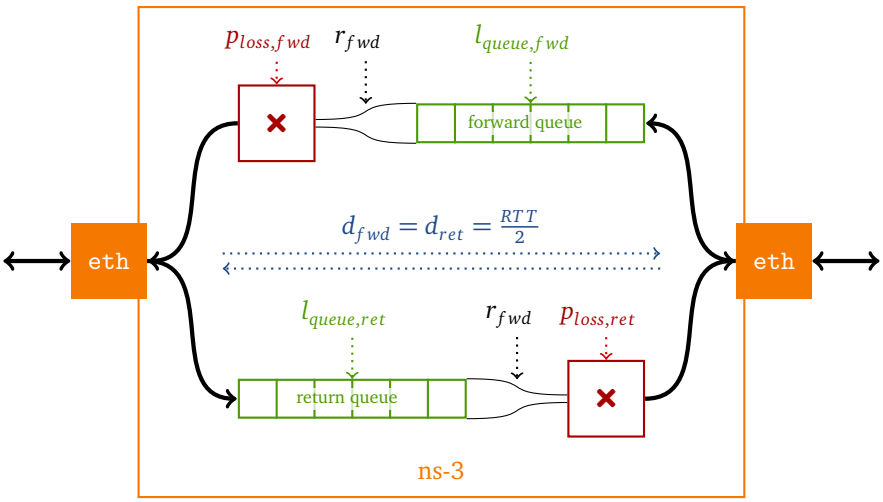


Figure 4.2 – Visualization of the Asymmetric ns-3 Scenario

path. It is open-source and available online[39]. Figure 4.2 illustrates the mechanism of this scenario. The two interfaces, one of which is connected to the client and one to the server, are connected with two unidirectional links. For each of them, the queue size l_{queue}, the data rate r, and the packet drop probability p_{drop} can be configured. Additionally, the delay d, which is half of the RTT, is set for both directions. Using this model we added the emulated measurement scenarios SAT and SATLOSS.

4.2.1.2 Parameters for New Measurements

The parameters of all test cases and measurement scenarios are displayed in Table 4.3. SATLOSS is basically the same as SAT, but with a PLR of 1 %. The artificial losses are distributed uniformly (we do not use a sophisticated error loss model). In both cases, the data rate is 20 Mbit/s in the forward link and 2 Mbit/s in the return link, so that it corresponds to the advertised data rates of ASTRA. Unfortunately, this is different to the data rate of GOODPUT where 10 Mbit/s are used in both directions. We did not want to change that parameter because we wanted to be able to compare our results with the results that are produced using the original QIR. Though, as typical satellite links are asymmetric, we had to choose different values for the satellite scenarios, anyway, and 20/2 Mbit/s are at least in the same magnitude. The RTT is set to 600 ms, which is the default used in literature presented in Chapter 3. Given these parameters, we can calculate the BDP as $\frac{\text{RTT}}{2} \cdot r_{\text{fwd}}$ for each scenario, as shown in the table. In all measurements, i.e., each scenario in the table except LONGRTT, a 10 MiB file is transferred. While this is roughly 100 times as large as the BDP in the GOODPUT scenario, it is only twice the BDP in the SAT, SATLOSS and ASTRA

Table 4.3 – QIR Test Cases and Measurements with Relevance for SATCOM.

Name	RTT	Data Rate	PLR	Time-out	Min. Goodput	Min. Efficiency	BDP
	[ms]	[Mbit/s]	[%]	[s]	[Mbit/s]	[%]	[MiB]
LONGRTT	1500	10	0	60	—	—	7.2
GOODPUT	30	10	0	60	1.4	14	0.1
SAT	600	20/2	0	120	0.7	3	5.7
SATLOSS	600	20/2	1	360	0.2	1	5.7
ASTRA	$\gtrapprox$600	20/2	$\lessapprox$0.1	120	0.7	3	5.7
EUTELSAT	$\gtrapprox$600	50/5	$\lessapprox$0.1	120	0.7	1	14.3

[39] Network Simulator for QUIC benchmarking with an additional asymmetric link scenario. `https://gitlab.cs.fau.de/sedrubal/masterarbeit/quic-network-simulator/-/tree/feature-asymmetric-p2p/` *(visited on 2021-12-14)*

scenarios and even less than $1 \cdot \text{BDP}$ in the EUTELSAT scenario. The test payload is much larger than a usual website, but it is a realistic value for file transfers. While QUIC was designed for HTTP, it will most likely also be used for other purposes in the future, like file transfer, WebRTC or even IP-Tunnels (see Section 6.1), which is more similar to the scenario of bulk transfers. Furthermore, the aim of using large files was about being able to analyze both large and small file transfers, with the transfer of a smaller file being roughly equivalent to the first few seconds of the transfer of a large one. However, the probability of an error is higher for larger files. Therefore, the comparability is limited.

4.2.1.3 Timeouts

The execution of all tests in the official interoperability matrix takes about two hours. Most experiments are functionality tests that only take a few seconds to run. As we added a lot of long-running measurements, the execution time to fill the entire matrix rose significantly to several days. Therefore, each test run will be terminated after a certain period of time. Because of the challenging parameters of the new measurement test cases, much higher timeouts were chosen compared to the timeouts used for the already existing measurements, as it is listed in Table 4.3. The relatively small value for the scenario called GOODPUT is adopted from the official QIR. There are different reasons for measurements to time out: Either the endpoints may be inactive or in a faulty state and nothing will change, when the timeout is increased. Or the performance is so bad that it takes too long to transfer the test data. Table 4.3 also calculates the minimum required goodput to pass the test cases. As the timeouts are chosen quite generously, it is safe to quit the execution after the timeout is reached, and consider the test case as failed, as such a poor goodput will hardly be useful in real world scenarios.

4.2.1.4 Measurement Iterations

In order to get more reliable results, we increased the amount of iterations of each simulated measurement test case for the same combination of server and client from 5 to 10. Measurements using a real satellite connection, as explained in the next section, are only repeated 5 times to avoid running into rate limits. Unfortunately, increasing the iteration count has a negative side effect: As a series of measurements for the same implementation combination and the same scenario is aborted upon the first failed iteration, the probability that there is at least one failed one, is increased by increasing the amount of iterations. This leads to slightly more failed entries in the result matrix. In the future, the runner should be modified to execute the same number of iterations for each combination and simply count the failed experiments.

4.2.2 Distributed Deployment for Non-Emulated Links

Besides SAT and SATLOSS we also added two scenarios, which use a real satellite link instead of a simulated one. There is one main computer which is used as controller and as host for the server implementation, and there is one client computer per satellite access where the client implementation has to be executed. Docker Compose works only for a single host if it is used the way it was used before. Therefore, it was necessary to transform the part of QIR, which handles the deployment of the Docker containers. We replaced it with direct API calls to the Docker daemon by using the Python Docker library docker-py[40]. It comes with built-in support for remote Docker daemons that can be accessed via SSH. Each test case can now optionally specify an SSH-connect-URL for the deployment of the client implementation. The new approach for distributed scenarios is shown in Figure 4.3. Instead of virtualized networks, the containers for the endpoints are now bridged to the host network. The server is then accessible via the public IP address of the host. In order to find the public IP address that can be reached from the host of the user implementation, all available addresses are probed automatically by spinning up a minimal server on the host before starting the measurements. This works automatically by using the existing SSH connection opened by docker-py. The determined IP addresses are also used for the check of the captured traces after the measurement.

While the packets can be captured by ns-3 in simulated scenarios, we have to use a second tcpdump container per endpoint, which joins the network of implementation under test, to capture all packets that pass through the network. The original design of QIR depended on Docker bind-mounts to access traces, logs and other files from

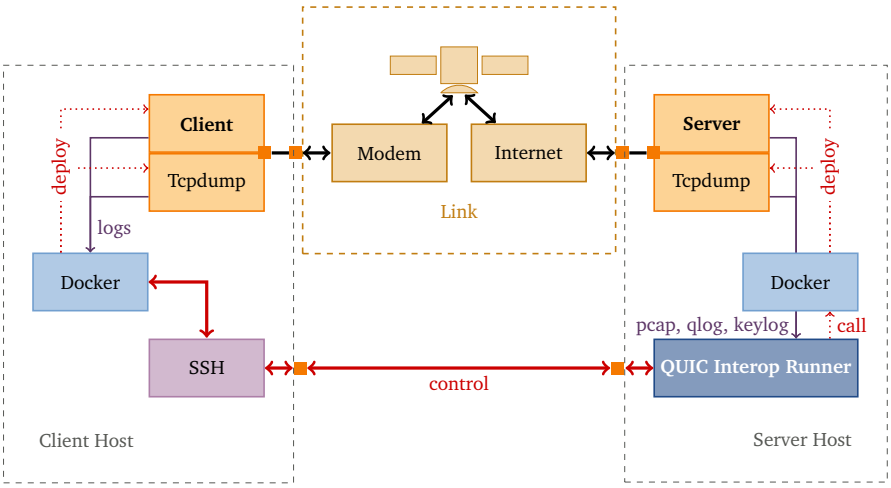


Figure 4.3 – Distributed Architecture of Modified QIR

[40] docker-py on GitHub `https://github.com/docker/docker-py` *(visited on 2021-12-14)*

the containers. As bind-mounts also only work for a single host, we replaced them by "`docker container cp`"-API calls though docker-py. Thus, the Python library seamlessly handles the transfer via SSH for remote hosts.

We experienced some difficulties with the default limit for parallel sessions in OpenSSH. The value of `MaxSessions` had to be increased from 10 (default) to about 50.

By using this approach we extended the set of measurements with ASTRA and EUTELSAT. Some details about the satellite accesses are shown in Table 4.4 and in the sections below. More detailed analysis of the links can be found in [11]. We additionally added support for *Starlink*, but running measurements on it and evaluating the results is out of scope of this thesis.

All vendors provide IPv4 connectivity using carrier-grade NAT, but none supports IPv6 so far. The client computers, the modems, and the dishes are deployed at the roof of the tower for computer science of the University of Erlangen (Martensstr. 3, D-91058 Erlangen), and no obstacles are in the line of sight. According to previous measurements [11] the RTT for both ASTRA and EUTELSAT is usually in the range of 600 to 1000 ms and the PLRs are less usually less than 0.1 %. The advertised net data rates are reached. Unfortunately, these values are not monitored during the measurements, but no indications for deviations have been found, as explained in Section 5.7.4.

4.2.2.1 The ASTRA Measurement Test Case

For ASTRA, we are using *Novostream Astra Connect L+*[41]. While this is with roughly 55€ per month the largest offer, there are also cheaper ones with traffic limits. The geostationary satellites are operated by *SES Astra* and deployed at 28.2° East. They provide coverage all over Europe.

Table 4.4 – Measurements with Real Satellite Access

Test Case	Provider & Tariff	Advertised Data Rate	Traffic Limit	Modem	IPv6
ASTRA	**Novostream** Astra Connect L+	20/2 Mbit/s	—	Gilat SkyEdge II-c	✗
EUTELSAT	**Konnect** Zen	50/5 Mbit/s	prioritization up to 60 GB	Hughes HT2000W	✗

[41] Novostream Astra Connect sales website: `https://www.novostream.de/produkt/astra-connect/` *(visited on 2021-12-15)*

4.2.2.2 The EUTELSAT Measurement Test Case

The satellite access for the measurement EUTELSAT is *Eutelsat Konnect Zen*[42] which is operated by Eutelsat. To provide broadband services in Europe and Sub-Saharan Africa it uses a modern HTS satellite at 7.2° East. It was built by Thales Alenia Space and became operational in 2020[43]. The planned life cycle is 15 years, so QUIC will likely be one of the most used internet protocols during that time. We are using the medium-sized offer for about 50€ per month with a soft traffic limit of 60 GB of prioritized traffic. As mentioned in Section 5.7.4, we did not exceed this limit with our measurements.

4.2.3 Efficiency Metric

The original version of QIR uses the goodput that was achieved by the combination of implementations, as result of a measurement. It is defined by the size of the transmitted file divided by the time between the reception of the first and the last packet, as shown in Equation (4.1). Since the differences in data rates between the scenarios are very large, as can be seen in Table 4.3, the absolute goodput value is not useful anymore to compare the performance in different scenarios. Therefore, we have implemented the measure of efficiency η in the QIR as defined in Equation (4.2), which is simply the achieved goodput normalized to the maximum possible net throughput of that forward link.

$$\text{goodput} = \frac{\text{file size}}{t_{\text{rec, last packet to client}} - t_{\text{rec, first packet to client}}} \tag{4.1}$$

$$\eta = \frac{\text{goodput}}{r_{fwd}} \tag{4.2}$$

We are assuming that the available data rate of the return link has no impact on the goodput and can be neglected, which seems to hold according to our observation in Section 5.6.9.1. The website that visualizes the measurement results uses the efficiency to highlight the results in color. Figure A.1 shows a partial screenshot of the modified website.

In order to be more precise, the formula could be extended to Equation (4.3), to respect the influence of non-zero PLRs. However, for the sake of simplicity this minimal difference is omitted.

$$\eta_{exact} = \frac{\text{goodput}}{r_{fwd} \cdot \left(1 - p_{loss,fwd}\right)} \tag{4.3}$$

[42] Eutelsat Konnect sales website: `https://konnect.com/` *(visited on 2021-12-15)*

[43] Eutelsat Konnect information website: `https://www.eutelsat.com/en/satellites/eutelsat-7-east.html#eutelsat-konnect` *(visited on 2021-12-15)*

A weakness of this metric is that it neglects a commonly observed phenomenon: The impact of increasing the RTT by a given factor is usually worse than decreasing the data rate by the same factor[44]. However, we are able to qualitatively verify this phenomenon when GOODPUT is compared to SAT or when SAT is compared to EUTELSAT in Chapter 5.

4.2.4 Additional Features and Functionality

A lot of further small modifications have been applied to the codebase of QIR. Noteworthy is the ability to **shuffle test runs and measurement iterations** to avoid that temporal interferences negatively influence all measurement results of a single implementation combination. Instead, such temporal effects now have virtually no influence on the average results.

To avoid the periods of higher utilization of the satellites in the evenings, as explained in Section 2.1.3.3, we added the possibility to **pause measurements during a given time period**. This is very useful because the measurement of all implementations takes several days.

The result matrix is written to a slightly complex JSON file. We added a powerful **parser** library that is furthermore able to modify the content of the results. In that way, we could modify the runner to **resume** aborted runs, which is very helpful for development purposes, but also because the execution takes very long due to the new measurement scenarios. It is also possible to retry failed experiments. This feature should only be used when a measurement failed because of bugs or reasons that are not caused by the implementation. Otherwise, the test results will be falsified, especially when measurements are retries which timed out before.

For reasons of **reproducibility**, we additionally store information about the docker images of the implementations, like the image ID and the image repository digests, which distinctively describe the exact image. On resumption, it is checked that the same images as before are used again.

To make the output of the runner on the terminal more clear, it was restructured, and it is now **colored**. Many components of the modified QIR can also be run as standalone programs, which spawn interactive shells for debugging.

For evaluation of the large measurement data, we developed tools to **render plots** for each and every trace in the interoperability matrix of an interoperability run. The tools are able to generate different plots. However, the most useful ones are offset number plots, as explained in Section 5.6.1. To create them, we parse all Pcaps that are captured on the client and on the server side using pyshark. In order to infer additional information from the traces, like the TTFB, TTLB and the

[44] Document by Mike Belshe: "More bandwidth doesn't matter (much)" `https://goo.gl/X8rE6Q` *(visited on 2021-12-20)*

points in time, when the first and the last packet was sent by the server, we try to match the packets from both traces. If that works, we also try to find eligible packets to determine the current RTT. The heuristically determined information is used to additionally annotate the plots. The tools are built to use multiple processes, but analyzing all Pcaps and rendering all plots still took several weeks. The reason is that it is very slow to load the traces into Python objects.

4.2.5 Environment

All measurement mentioned in the next chapter have been performed on the following systems:

The simulated scenarios have been executed on an Ubuntu 20.04.3 LTS server with Kernel 5.4.0. Docker (version 20.10.10) was configured to use overlay2 on *extfs* as storage driver, *runc* version v1.0.2 as container runtime, *apparmor* and *seccomp* (Profile: default) as security options (no SELinux), and cgroupsv1. The server is powered by an *Intel® Xeon® X5650* CPU launched in Q1 2010 which supports SSE4.2 but not AVX. The system has 16 GiB of RAM installed.

For measurements with real satellite connections, the server implementations have been executed on the aforementioned machine. The clients have been run on other computers, which are connected to the modems of the satellite link. These computers run Ubuntu 18.04.6 LTS with kernel 5.4.0. Docker (version 20.10.7) is configured similar to the main server but with *runc* v1.0.0. Both computers are powered by an Intel® Core™ i5 4590 CPU (launch date Q2 2014 with support for SSE4.2 and AVX2) and 8 GiB RAM each.

Chapter 5

Analysis

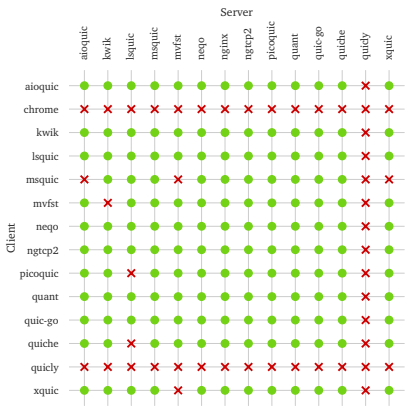
> The first time you do something, it's science.
> The second time, it's engineering.
> The third time, it's just being a technician.
>
> *Clifford Stoll*

Using the modified QIR described in Section 4.1.6 we ran measurements for the most relevant QUIC implementations that are currently publicly available. In this chapter, we first present the result matrices in the style of the result tables of the original QIR. After discussing reasons for failures, we first evaluate the measurement results on a statistical basis, then we examine traces of individual implementations. An evaluation of the quality of the results concludes the chapter.

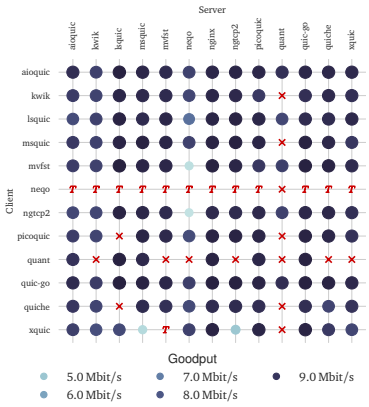
5.1 Measurement Result Matrices

As explained in Section 4.2.1, among the test cases of the original QIR, one named LONGRTT is relevant for SATCOM. To be able to compare the own measurement results with that ones from the official setup, the GOODPUT measurement was used for reference. Additionally, we added four new measurement scenarios, one simulated scenario without losses, one with an artificial loss rate of 1 %, one using SES Astra as real satellite link and one using Eutelsat.

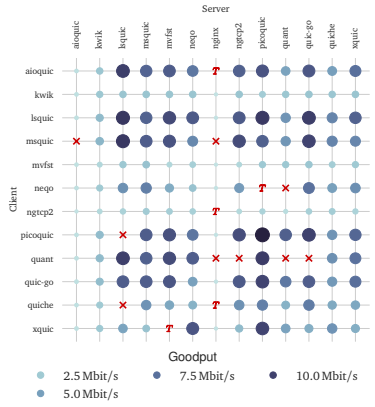
The results of our tests and measurements are visualized as heatmap in Figure 5.1. For the LONGRTT test case, the binary results (✅ for "succeeded" and ❌ for "failed" or "not supported") are shown in Figure 5.1a. All other matrices belong to measurements. The average of the achieved goodput values for multiple iterations of the same scenario with the same combination of implementations are displayed as cells. Failed measurements are marked with ❌.



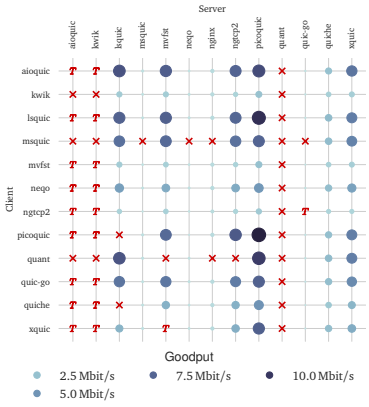
(a) Test Results of Test LONGRTT



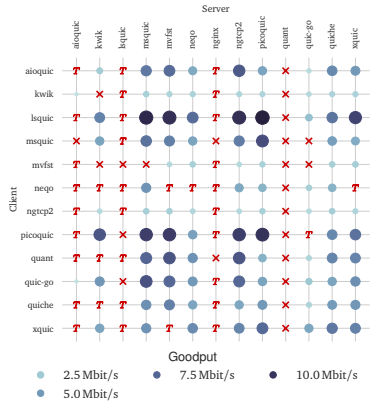
(b) Average Goodput of Measurement GOODPUT



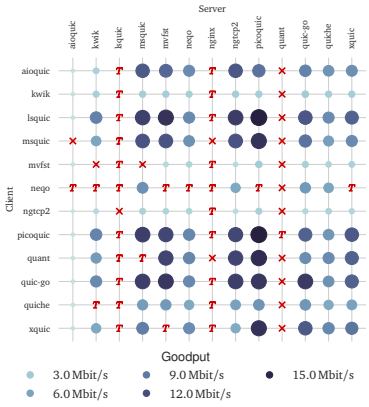
(c) Average Goodput of Measurement SAT



(d) Average Goodput of Measurement SATLOSS



(e) Average Goodput of Measurement ASTRA



(f) Average Goodput of Measurement EUTELSAT

Figure 5.1 – Result Matrices

We tried to automatically determine the reason for failed measurements because unfortunately the QIR does not store this information in a machine-readable form. Measurements that most likely failed because of timeouts, are marked with ***T***. The reasons for other failures are explained in the next section. For better readability, we omit *chrome* and *quicly* in Figures 5.1c to 5.1f, since they always fail, as explained below. We analyze these results in detail in the following sections.

5.2 Failed Experiments

Before analyzing the succeeding test and measurement results we briefly have a look at the failed ones.

5.2.1 Unsupported Implementations

5.2.1.1 *Chrome*

The wrapper script `run_endpoint.sh`, which is packaged in the Docker image for *chrome*, filters out unsupported test cases. As defined by QIR all implementations have to exit with return code 127 if they do not support a specific test case type. At time of writing, *chrome* supports only the HTTP3 test case even with the official QIR. It is probably very difficult to support a broader range of test cases because *ChromeDriver*, which is used to run the Chrome browser in a headless mode, only exposes a limited set of flags to control the QUIC behavior. Measurements using *chrome* are of high interest since according to statistics about usage share of web browsers[45] numerous users use this implementation for everyday Internet browsing. On the other hand, *Google Chrome* as QUIC client has already been often used for measurements, as shown in Chapter 3.

5.2.1.2 *Quicly*

According to the logs that are available at `https://interop.seemann.io` the test cases for *quicly* usually fail during the compliance check for various reasons. In our experiments, the implementation does not even start regardless whether it is used as client or server. Instead, it immediately crashes with the message “Illegal instruction (core dumped)”. As it works on other machines, the old hardware described in Section 4.2.5 used to run the experiments might cause this issue. However, compiling *quicly* on the target machine by building the Docker image there does not solve the problem. It would be out of scope to debug this issue, especially because we expect these test cases to fail anyway, as indicated by the official interoperability matrix.

[45] Browser statistics of Wikimedia: `https://analytics.wikimedia.org/dashboards/browsers/#all-sites-by-browser` *(visited on 2021-12-13)*

For measurements using real satellite links, a computer with a more modern CPU was used for running the client (see Section 4.2.5). On these machines the `Illegal Instruction`-error does not appear, and the implementation fails for other reasons, as it is the case on the official QIR website.

Measurements using *quicly*, however, can be found in [61].

5.2.1.3 *Quant*

Unfortunately, the pre-built server and client binaries of *quant* also crash on our hardware, most likely for the same reason as *quicly* does, as explained in the section before. Building the Docker image on the server used for the experiments, and thus compiling *quant* for the aged target hardware architecture, solves this issue. But many measurements time out in our setup, while they are usually succeeding for the official QIR. As timeouts seem to happen less frequently, when *quant* runs on the more modern hardware, which is used for the client in combination with real satellite accesses, missing hardware acceleration features that are explained in Section 4.2.5 may cause the bad performance.

5.2.1.4 *Lsquic*

The Docker container of *lsquic* depends on IPv6 to be available inside the container, when used as server. The wrapper script passes an argument to the server executable to listen on `::1`. Both satellite operators used for measurements don't support IPv6 and the client PCs are configured accordingly. Thus, all measurements with real satellite links using *lsquic* as server failed with the message "bind failed: Cannot assign requested address". The Dockerfile for the QIR image is not publicly available, so it is hard to implement a workaround.

5.2.1.5 *NGINX*

While the QUIC preview of the web server *NGINX* often succeeds for emulated scenarios, it fails for all measurements via real satellite links. Most of them fail due to running in timeouts. It is unclear, why this happens reproducibly for real satellite connections but not for emulated ones.

5.2.2 Other Reasons for Single Failed Experiments

After each test run, some checks will be done to verify that the experiment succeeded. The most frequent reasons for failed experiments are named below.

5.2.2.1 Downloaded Files are Missing or Empty

All clients have to store the files, they downloaded from the server, to a given directory. The downloaded files are then compared to the original files. When files are missing or empty, the experiment fails. This indicates malfunction of one of the endpoints, which must be debugged by the developers. This happens strikingly often, when *msquic* is used as client (about 20 % of all measurements).

5.2.2.2 Key-log Files are Missing

After checking the downloaded files the Pcap traces are analyzed in order to verify that a QUIC connection was established with the given configuration as explained in Section 4.1.6. Additionally, for all measurements the time of arrival of the first and last packet has to be determined to calculate the goodput, like it was shown with Equation (4.1). To be able to decrypt the QUIC packets and access the transport layer information, key-log files are required, as explained in Section 4.1.4. When neither the server nor the client logged the TLS secrets, it is impossible to get this information and the test case is considered as failed. As all implementations are capable of logging key-logs, the lack of them indicates malfunction of the endpoints, e.g., a failure during handshake.

5.2.2.3 Timeout

There are several experiments that time out for measurements on simulated and real satellite links. The challenging scenarios render some implementations to fail transmitting the test payload, or they only achieve very poor performance. More details about timeouts are given in Section 4.2.1.3. It should be noted that if an implementation fails in our scenarios it does not mean that it can not be used via a SATCOM access at all, but it has difficulties to transfer a large file over such links. E.g., using it as HTTP API server or client might work fine. Additionally, as already noted in Section 4.2.1.4, timeouts which are caused by spontaneous effects, like temporary link degradation, makes QIR mark all experiments of a tuple of implementations for the same type of measurement as failed, even if the experiments before succeeded. This should be changed in future versions of QIR.

Neqo

It has to be noted that *neqo*, a client implementation by Mozilla written in Rust, runs into the timeout for all executions of the GOODPUT measurement. Despite the results being rather poor, it almost always succeeds for other measurement test cases. While we could reproduce the behavior on other machines, it appears to work on the official QIR. We could not figure out why this is the case.

5.3 Statistical Results by Measurement

After determining the reasons for failed test cases, we now analyze the results in more detail. First, we do that on a statistical basis differentiated by measurement.

In Figure 5.2, the measurement results of all individual experiment runs are presented in boxplots. On the left subplot the absolute goodput values as defined in Equation (4.1) are used whereas on the right side the values are normalized using the relative efficiency metric as proposed in Equation (4.2).

All boxplots have their minimum value at 0 because failed experiment runs are included with a goodput value of 0 Mbit/s respectively an efficiency of 0 %, and there are failed experiments in each measurement as it can be seen in Figure 5.1. There is a large difference between GOODPUT and SAT respectively SATLOSS, which is not visible for the absolute values, but for the efficiency values on the right side of the plot. The different proportions between the left and the right subplots are caused by the fact that the emulated data rate in the forward path is twice as high for SAT and SATLOSS as for GOODPUT (20 Mbit/s respectively 10 Mbit/s). The limits are visualized using red lines. Similarly, the absolute values reached for EUTELSAT are the highest goodput values among all scenarios (about 17.5 Mbit/s). But normalized to the theoretical throughput of 50 Mbit/s in the forward path as defined in the Service Level Agreement (SLA), the maximum efficiency is with less than 40 % the smallest within all measurements. The boxplots of SAT, SATLOSS and ASTRA are

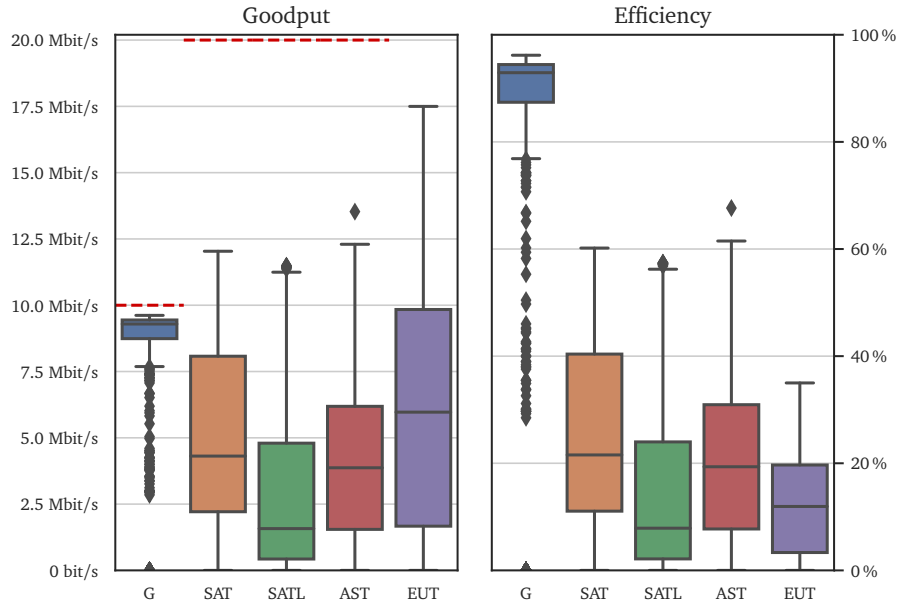


Figure 5.2 – Distribution of Results of all Measurements

scaled by the same factor on the right side because the simulated data rate equals the advertised data rate of the SES Astra connection used.

Table 5.1 shows the maximum and the mean goodput and efficiencies for each scenario again, but expressed in numerical values. The percentage of implementation combinations that did not succeed are listed as well. The reasons for the fails are explained in Section 5.2. Additionally, we added the percentage of implementation tuples that failed because they most likely ran into a timeout at least once. As explained before, the reason for a failed test is not saved in machine-readable form. Thus, we analyzed the log files, to calculate this value.

It can be clearly seen that, on average, the efficiency decreases significantly if the link exhibits characteristics typical for geostationary satellites. Introducing losses further decreases the efficiency. Switching from Astra to Eutelsat, which advertises a connection with a throughput more than twice, does not automatically double the goodput. Instead, the mean efficiencies reached on Astra are even about 66 % higher than on Eutelsat. After all it must be noted that the performance of QUIC via both simulated and real satellite links is on average extremely poor.

Besides bad results, there is also a very large number of combinations that failed to transfer a 10 MiB file in those scenarios—for the test cases SATLOSS, ASTRA and EUTELSAT about half of them. While at about 30 % of the fails in each measurement scenario can be explained by missing compatibility or other reasons explained in Section 5.2, the percentage of combinations that fail due to timeouts varies per measurement type. The majority of timeouts in GOODPUT are caused by faulty behavior of *neqo* client which was described in Section 5.2.2.3. While there are only a few timeouts in the SAT scenario, the percentage increases drastically up to 10 % in the lossy SATLOSS scenario. For both real satellite scenarios ASTRA and EUTELSAT, the percentage even increases further up to about 18 % resp. 16 %. For SATLOSS, *aioquic* and *kwik* as server cause most timeouts whereas for ASTRA and EUTELSAT, *aioquic*, *lsquic*, *nginx*, and sometimes *kwik* as server seem to be the cause.

Table 5.1 – Overview of Measurement Results
For the columns **Mean** and **Maximum**, the absolute goodput values are in the left column are and the relative efficiency values in the right column.

Measurement	Mean		Maximum		Failed	Timeout
	[Mbit/s]	[%]	[Mbit/s]	[%]	[%]	[%]
GOODPUT	8.54	85	9.6	96	35	7
SAT	4.88	24	12.0	60	28	3
SATLOSS	2.86	14	11.5	57	44	10
ASTRA	3.98	20	13.5	68	51	18
EUTELSAT	6.01	12	17.5	35	45	16

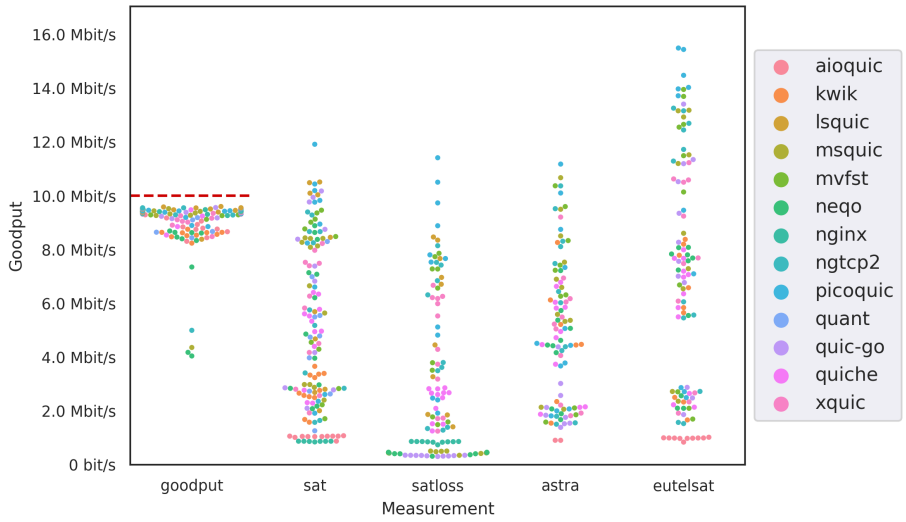


Figure 5.3 – Swarm Plots of Mean Goodput Results for Different Measurements

The visualization of the statistical values above hides the underlying distribution of the measurement results. In Figure 5.3, the mean goodput of each combination is plotted separately for each measurement. Each point is colored according to the server. Only values of successful measurements are included and for the sake of clarity some points of the GOODPUT scenario are omitted. Also be aware that absolute values are displayed and the emulated network in the GOODPUT scenario provides a much smaller data rate (red line), than the others (see Table 4.3). This reveals an interesting difference between the measurements: Apart from a very few outliers, there is more or less only one large group of implementations in GOODPUT, which all together achieve rather good results. However, in SAT and SATLOSS, the points spread across a large range of goodput values. The distribution of measurement results seems to be divided into two groups when using real satellite links, a more compact group achieving very poor values and a stretched group achieving slightly better to medium efficiencies. This phenomenon can be seen better at the EUTELSAT scenario. While it is very hard to figure out the reason for this behavior, it might be an indication that some implementations are not able to negotiate some connection parameters or QUIC features. Being at the top of every distribution for satellite scenarios, *picoquic* reaches the best and results. At least for SAT and EUTELSAT, *aioquic* performs the worst.

5.4 Analysis by Implementation

The high variances in the measurement results, we have seen before, makes it difficult to draw general conclusions. A more detailed analysis that differentiates between implementations has to be done.

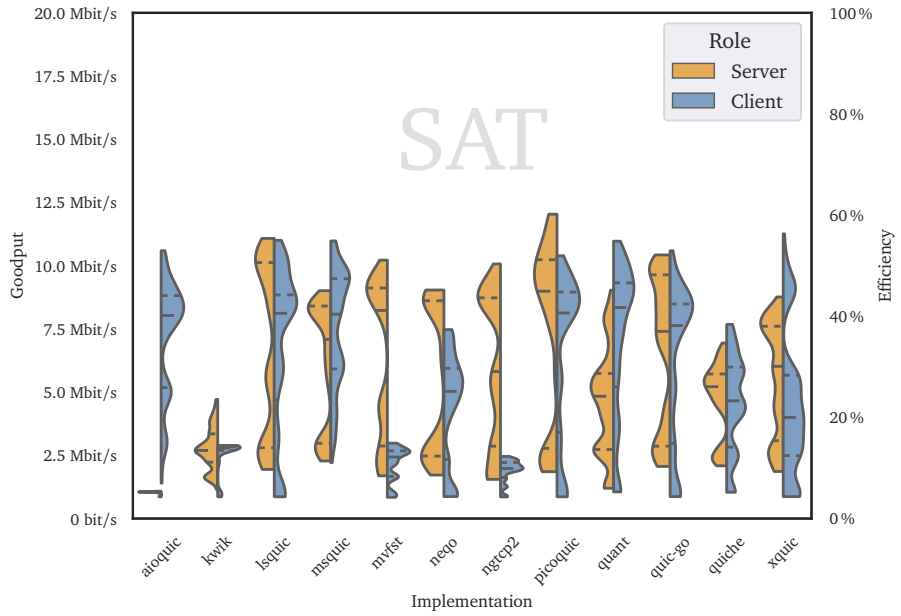
In order to simplify the analysis, we want to verify the assumption that the performance is mainly determined by the server implementation and that the client is of secondary relevance. This would make sense as the server has to transmit much more data and has to estimate the channel and especially the bottleneck bandwidth. If this hypothesis had been correct, we would have been able to marginalize the result matrices onto the server dimension. For this purpose, we plot the distribution of the achieved goodput values in the SAT and SATLOSS scenarios into Figure 5.4 by using violin plots separately for each implementation. The plots for the other measurements are attached in Appendix B. We differentiate between the results that were achieved when the implementation was used as server (🟠 orange) or as client (🔵 blue). To estimate the distribution of the results based on the underlying histogram we use *kernel density estimation (KDE)* provided by seaborn[46], which is a Python library for data visualization. In order to avoid displaying impossible values, like negative data rates, we clip the distributions at the maximum and minimum observed value. Additionally, we omit failed experiments[47] and implementations that did not succeed in a significant amount of experiments for both roles.

Most server distributions (🟠 orange) show either one maximum at a very low data rate, like *aioquic*, or in the case of *lsquic*, *msquic*, *mvfst*, *neqo*, *picoquic* and *quic-go* two maxima with one being at a lower and the other one at a higher data rate. In the SATLOSS case in Figure 5.4b, the upper bulge disappears for *msquic*, *neqo* and *quic-go* and the performance is low overall. The narrow distribution of *aioquic* can be explained by the very deterministic behavior that is shown in Section 5.6.4. It is evident that it constantly achieves similar data rates regardless which client was used. The distributions with two peaks indicate that the server implementation is generally able to achieve better results. However, the server cannot compensate a poorly performing client.

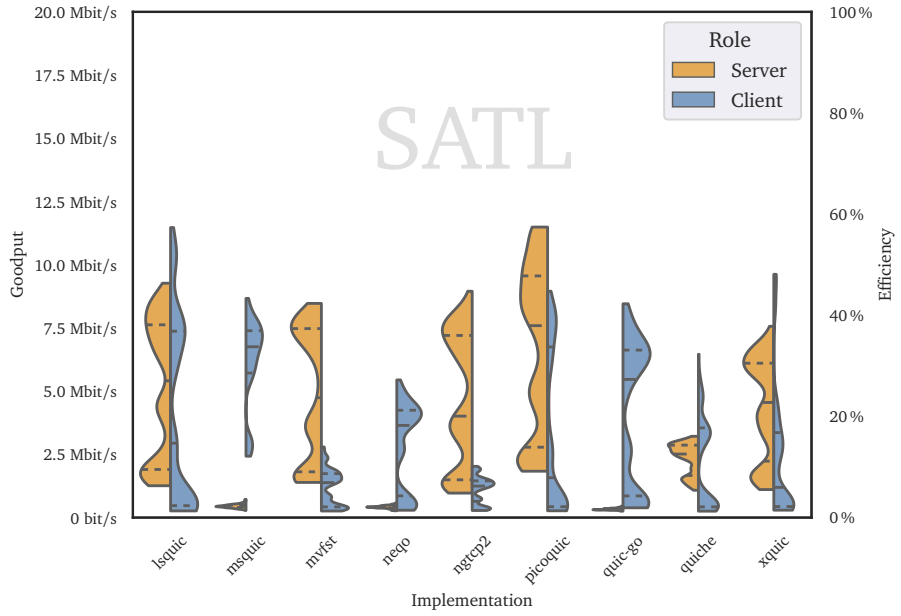
On the client side (🔵 blue), the variance is often very high. *Lsquic*, *picoquic*, *quant* and *quic-go* seem to have most of their measurement results at a slightly higher goodput as the rest. Examples for clients that constantly achieve poor results, while the range of values achieved by the corresponding server implementation are quite large, are *kwik*, *mvfst* and *ngtcp2*.

[46] Seaborn Documentation for violin plots that employ KDE. `https://seaborn.pydata.org/generated/seaborn.violinplot.html` *(visited on 2021-12-17)*

[47] We could include measurements that have reached timeouts with a data rate of 0 Mbit/s. However, the results are unfortunately not available in such a form that the reason for a failed test could be easily determined. This should be improved for future versions of QIR.



(a) SAT



(b) SATLOSS

Figure 5.4 – Distribution of Measurement Results by Implementation for SAT and SATLOSS.
The horizontal lines represent the quartiles Q_1, Q_2 and Q_3.

Hence, it seems that both the client and the server contribute to the overall performance of a QUIC connection—contrary to our original assumption. In most cases, it really depends on the combination of implementations. It has been observed that some servers perform better than others. However, there is no guarantee that the selection of a server of superior performance enhances the performance regardless of the choice of the client. Likewise, using a good client may result in poor results in combination with a bad server. Additionally, an implementation can achieve good results in one scenario and poor ones in another. The dependency of the performance on both endpoints is also shown by the fact that in the result matrix in Figure 5.1 you cannot see that there are rows that clearly dominate over columns, nor the other way around. Nicolas Kuhn makes the same conclusion that both the server and the client have a significant influence.[48] Technically the impact of the client is mainly manifested by the way it sends `ACK`s or by the set of features and parameters it offers to the server.

5.5 Correlation of Measurement Results

In this section, we want to analyze the correlations of the results between different measurements in order to find trends and interrelations of the behavior of the implementations.

5.5.1 Behavior of Implementations in Different Scenarios

First, we relate the average efficiency values of each combination of implementations for pairs of measurement scenarios each. Each column and each row of Figure 5.5 belongs to one measurement. The cells in the lower triangle of the grid represent the relation of the measurement results of the correspondent test cases. The points are colored depending on the server, due to the analysis in Section 5.5.2. On the diagonal, where row and column belongs to the same test case, a histogram of the achieved efficiency of that test case is plotted. The upper triangle of the grid is left empty, as it would show the same relations as the lower left half but transposed.

First we have a look on the diagonal where the histograms of the measurement values of each test case are located. While almost all implementation combinations reach an efficiency value of over 80 % in GOODPUT (G), none of the server-client-tuples reaches an efficiency greater than 55 % on the ASTRA measurement (AST), 30 % on EUTELSAT (EUT) and 60 % on SAT (SAT) and SATLOSS (SATL). There is a very little amount of combinations that don't perform well in the GOODPUT

[48] Early research results by Nicolas Kuhn about performance and interoperability presented at *2nd QUIC and Satellite Open Stakeholder Meeting* on 2021-12-02: `https://erg.abdn.ac.uk/video/2nd%20ESA%20MTAILS%20Satellite%20Stakeholder%20Meeting/1.3%20Performance%20implications%20of%20interoperability.pdf` slide 3 *(visited on 2022-01-03)*

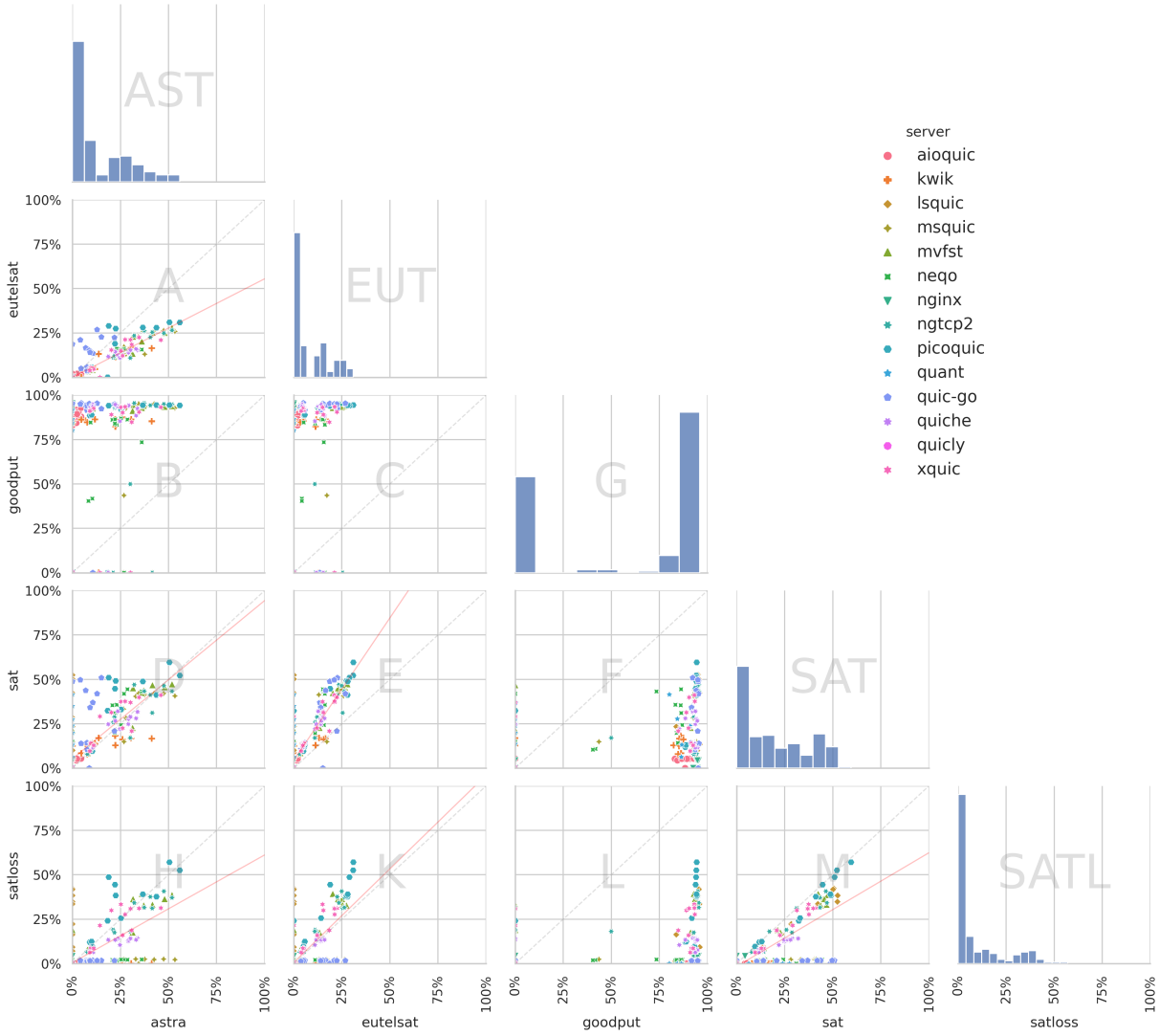


Figure 5.5 – Relation of the Average Efficiency of the Same Implementation Combinations Between Different Measurements

measurement case, including multiple combinations with *neqo* as server, which can be seen in the subplots below. The high values for the GOODPUT measurement prove that QUIC performs well on an average network path as encountered in everyday scenarios. It also demonstrates that the implementations used for the measurements are kind of optimized for such links. It has to be mentioned that there is a certain limit for the efficiency values because IP, UDP and QUIC headers are always required. A noteworthy detail already mentioned before is that the distributions for ASTRA, EUTELSAT and SATLOSS measurements have two maxima: A larger group of combinations that perform very poor and a smaller one that performs better, but still not well at all as noted before.

Now we are inspecting the relational scatter plots in the lower left triangle.

Considering the subplots Ⓑ, Ⓒ, Ⓕ and Ⓛ, it is noticeable that the values of the GOODPUT measurement are rather uncorrelated with the values of all other measurements shown. This may indicate that an implementation optimized for a regular terrestrial network path, does not automatically perform well on a satellite path with a geostationary satellite.

The scatter plot Ⓐ of ASTRA and EUTELSAT measurement results shows a higher correlation than we have seen before. Apart from a few combinations using *picoquic* and *quic-go* as server, which achieve a better efficiency on EUTELSAT than on ASTRA, most implementations tend to achieve a higher efficiency on ASTRA than on EUTELSAT. But as said before, all efficiencies on simulated or real satellite links are very poor. The efficiency values on the real satellite links, ASTRA and EUTELSAT, however, are calculated from the advertised data rates of the commercial products. In reality, they could be much lower or vary considerably.

The relational plot between SAT and ASTRA (Ⓓ) suggests correlation, but less than the plot between SAT and EUTELSAT (Ⓔ). The high correlation of the latter one highlights the similarity of the test results from EUTELSAT measurement compared to the simulated test results from SAT test case, which means that the simulated scenario emulates the real scenario quite well. Similar to the comparison of EUTELSAT and ASTRA, there are some combinations with *picoquic* and *quic-go* as servers that perform better in our emulated environment (SAT) than in the ASTRA scenario. These values are above the diagonal of equilibrium.

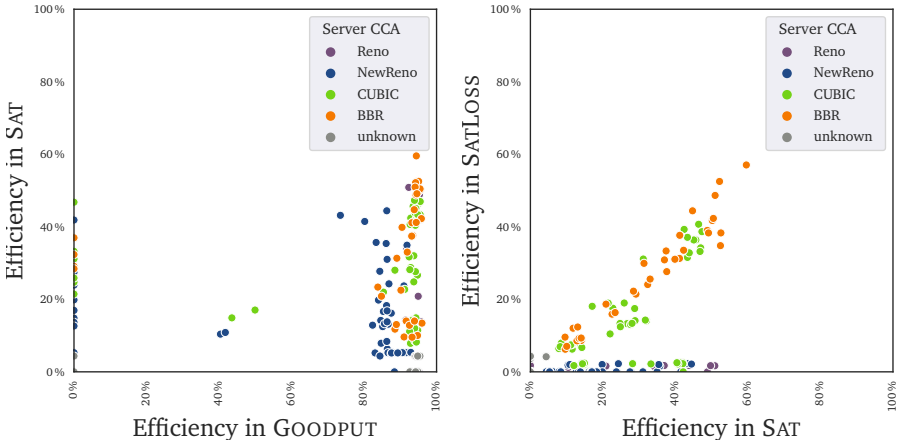
On the plots for SATLOSS (Ⓗ, Ⓚ, Ⓜ) the set of data points can be divided into two distinct subgroups: One showing quite a high correlation with the results of ASTRA, EUTELSAT and SAT test cases and one constantly performing very bad in the SATLOSS scenario but better in the other scenarios. The latter group contains combinations using *msquic*, *neqo* or *quic-go* as server. On the other side *picoquic* performs remarkably better on SATLOSS than on ASTRA and EUTELSAT. The reason might be that the PLR is higher on SATLOSS than on the real satellite

links. No implementation performs significantly better on SATLOSS than on SAT. This seems reasonable, as the additional loss in SATLOSS makes it more challenging.

5.5.2 Distinction by CCA

Since we expect that the CCA has a high influence on the performance, we inspect two subplots from Figure 5.5 in more detail. We colorize the points in the scatter plot according to the associated default CCA of the server implementation that was analyzed in Table 4.2. In Figure 5.6, we choose the scatter plots for GOODPUT with SAT (Figure 5.6a) and SAT with SATLOSS (Figure 5.6b), respectively.

In the first case, we can not find clusters of measurement results. While servers with CUBIC and BBR seem to achieve slightly better results in GOODPUT, compared to servers using Reno and NewReno, the same implementations achieve a large range of efficiency values in the SAT case. No conclusions can be drawn from this relational plot. In the comparison plot for SAT and SATLOSS, we again see the two groups of measurement results, as explained in the section before: One group of measurement results showing a correlation between both scenarios and one achieving a larger range of goodput values in the SAT case while not performing well in the SATLOSS case. It is evident that no servers using Reno or NewReno are represented in the first group, which means that they don't perform well in the lossy scenario. This can have two possible reasons: If we assume poor correlation, this can be a side effect that implementations employing more complex CCAs like CUBIC and BBR are also higher optimized. On the one hand, if we assume causality, this could be an indication that QUIC performs poorly in lossy scenarios in combination with Reno and NewReno.



(a) Correlation between GOODPUT and SAT **(b)** Correlation between SAT and SATLOSS

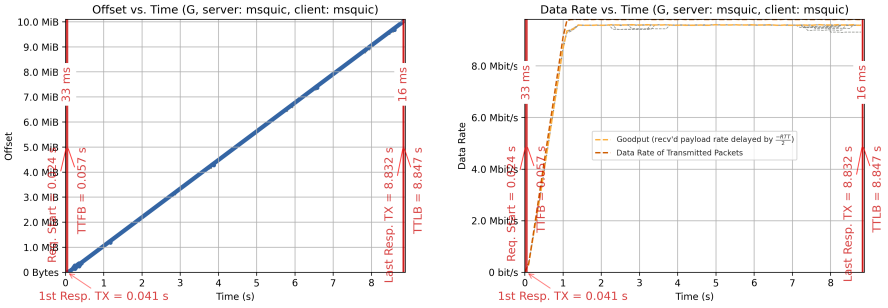
Figure 5.6 – Correlation of Results Colored by CCA

5.6 Analysis of Traces

Additionally, to the analysis of overall measurement results, we created automated scripts to draw several detailed plots of single file transfers, as already explained in Section 4.2.4. In the following subsections, we discuss the most interesting ones in which a particularly striking behavior of the applications can be identified. Other plots are available online[49]. Unfortunately, plots are not available for every measurement, since the mechanism depends on the implementations creating key-log files at the specified location (see Section 5.2.2.2), on integer Pcap files (see Section 4.1.4), on Wireshark, being able to parse and decrypt all QUIC packets (see Section 4.1.4), and on some other constraints. E.g., it is not always the case that implementations use the specified UDP ports and HTTP/0.9.

5.6.1 About the Offset Plots

The traces of most implementations in the GOODPUT scenario are very close to an ideal behavior. Therefore, they are well suited to explain the plots which we have automatically generated from the captured Pcap files. Figure 5.7 shows the offset versus time plot and the corresponding data rate plot. To generate the plots we normalize the timestamps of all packets to the timestamp of the first packet from the client to the server, which is assumed to carry the (1-RTT) handshake to open a new QUIC connection. So the first packet gets $t = 0\,\text{s}$, no matter how long the client took to start and send the first packet. Each `STREAM` frame contains a chunk of the file that is being transferred. To allow the client to reassemble the file from the chunks even when they arrive out-of-order, each frame is labeled with the offset number



(a) Offset versus Time Plot **(b)** The corresponding Data Rate Plot

Figure 5.7 – A Virtually Ideal Trace Using *Msquic* as Client and Server in GOODPUT Scenario

[49] Results of the experiments using our fork of QIR: `https://interop.sedrubal.de/` *(visited on 2021-12-18)*

within the file. We use this offset information to trace the transmission by plotting this offset number over the time we capture it.

The goodput data rate shown in orange in Figure 5.7b corresponds to the sum of payload data averaged over a window of 1 s. It can be thought of the derivation of the blue offset trace curve of the left plot if there are no retransmissions and all packets are transmitted in order. The line in red represents the raw data rate, which includes IP, UDP and QUIC headers. In the plot above, both lines are too close to see any difference between them.

When the information can be obtained from the Pcap files, interesting events, like they are explained in Section 3.1, are additionally annotated in red. The trace with median `TTLB` is displayed colored, while the other traces that belong to other iterations of the same measurement, are displayed in gray.

5.6.2 The Ideal Trace

These plots can be used to investigate the behavior of the implementations in detail. Therefore, we have to understand how an ideal trace looks like.

The delay between `requestStart` and `TTFB` should be very close to the minimum possible delay, which is $1 \cdot \text{RTT}$. Otherwise, this is an indication that the request is spread over several packages or that the server takes too long to respond. After that, the server has to start the transmission with a startup phase. It is totally fine to start at a low data rate and ramp it up in the first few milliseconds because otherwise networks with low BDP might congest. This slow start phase should be very short, like in the example above where it is not even visible. At least $1 \cdot \text{RTT}$ later however, after the arrival of the first `ACK` that signals that no packet was lost, the congestion windows should be increased quickly, and the offset curve should show a strong left curvature. Once the bottleneck bandwidth is reached, the server should continue to transmit at a constant data rate without interruption until the last packet. This behavior equals a straight line in the offset plot. The corresponding *data rate plot* should hence ramp up in the first milliseconds until a data rate near the channel capacity is reached. Then, the data rate should remain constant.

5.6.3 *Picoquic*

As mentioned before, *picoquic* seems to handle SATCOM scenarios very well. So we can choose it as positive example, and have a look at its offset plot. Firstly, with about 9 s, the time to completion is very short compared to the results of other combinations in the SATLOSS scenario that is shown in Figure 5.8. A short slow start phase can be observed. After being able to estimate the RTT and the bottleneck bandwidth, the server starts to transmit at an almost constant and high data rate of about 17 Mbit/s. Even losses are handled fine by just retransmitting the lost packets

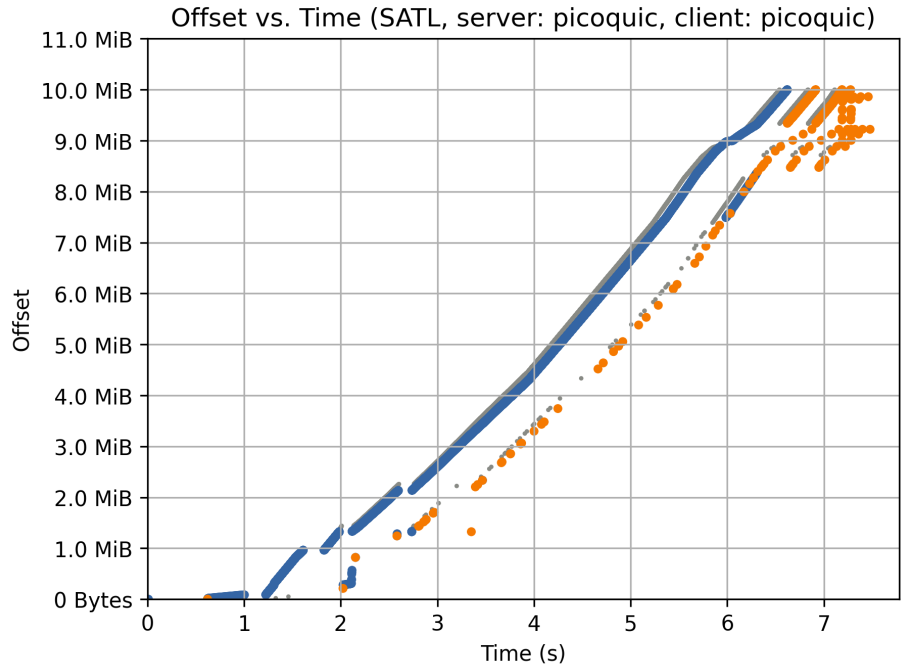


Figure 5.8 – *Picoquic* Showing High Performance Via a Lossy Satellite Link. Packets of the trace with median TTLB are colored blue respectively orange, when they are re-transmitted. Packets of other iterations of the same combination of implementations in the same scenario are colored gray.

without decreasing the congestion window. At the end of the transmission it does not wait for `ACKs` from the client, but immediately retransmits the last few bytes multiple times, to get the most out of the lossy channel (see Section 2.3.2.5). We can clearly see that *picoquic* seems to have some specific "SATCOM scenarios" in mind in its design, which is what it is known for in the community.

5.6.4 *Aioquic*

It is very eye-catching that the offset plots of *aioquic*, when used as server, show a reproducible clear, exponentially increasing curve, no matter which client is used. For Figure 5.9, *mvfst* is chosen as client, but it performs quite similar with other implementations and even via real satellite links.

In the displayed example, no retransmission was identified, and each offset is transmitted one after another with a slightly exponential increasing rate reaching its limit of 1.6 Mbit/s at the end. This increase is caused by NewReno (the CCA of *aioquic*) which iteratively tries to approximate the link capacity based on the knowledge inferred from `ACKs`. But the bottleneck bandwidth, as plotted in green, is never reached and the CCA never switches to steady state. Instead, with more than

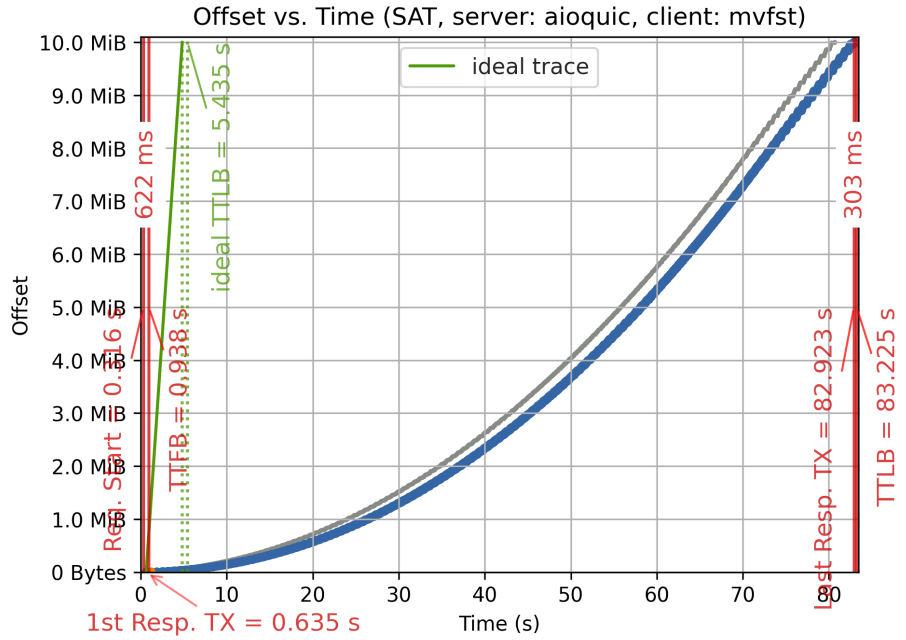


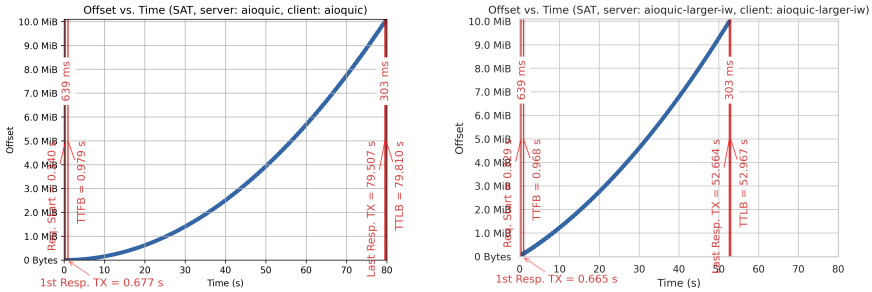
Figure 5.9 – Offset versus Time of a Transmission of a 10 MiB file using *Aioquic* as Server and *mvfst* as Client in SAT Scenario

80 s the transmission is remarkably slow. Losses (not shown here) are treated very exemplary by *aioquic* as long as they are very rare. There are a few measurements where only a single loss happened, which leads to an immediate decrease of the transmission rate followed by a clean exponential increase. Higher loss rates as in the SATLOSS or ASTRA case, cause *aioquic* to fail.

5.6.4.1 Increasing the Initial Window

To improve the performance and getting up to speed more quickly, multiple measures can be taken, as presented in Section 3.4. One of them is using a priori knowledge about the channel capacity and starting with a higher transmission rate by increasing the iwin. In Figure 5.10b, the iwin of *aioquic* was increased by factor 5, from 10 to $50 \cdot \text{MSS}$. Figure 5.10a is basically the same as Figure 5.9, which was shown before. Due to this modification the time to completion can be reduced by 30 % because the transmission already starts with a rate that is by factor 5 higher than before (about $\frac{1.2\,\text{MiB}}{10\,\text{s}} \approx 1\,\text{Mbit/s}$ instead of $\frac{0.25\,\text{MiB}}{10\,\text{s}} \approx 0.2\,\text{Mbit/s}$). In the end, roughly 2.2 Mbit/s $\left(\frac{1\,\text{MiB}}{3.8\,\text{s}}\right)$ are reached, which, however, is still far from the bottleneck bandwidth. Of course, the iwin can not generally be blindly increased because for normal links with lower BDP it would lead to overshooting the bottleneck capacity, especially when there are other competing connections. This example demonstrates the impact of

starting with the preferably previously estimated channel parameters, which is one of the ideas of 0-RTT-BDP, as described in Section 2.3.2.2.



(a) Unmodified *Aioquic* **(b)** *Aioquic* with larger initial window

Figure 5.10 – Offset versus Time Plot of a Transmission of a 10 MiB file using Two Different Variants of *Aioquic* in SAT Scenario

5.6.5 Bend in Offset Plot and Pacing Behavior

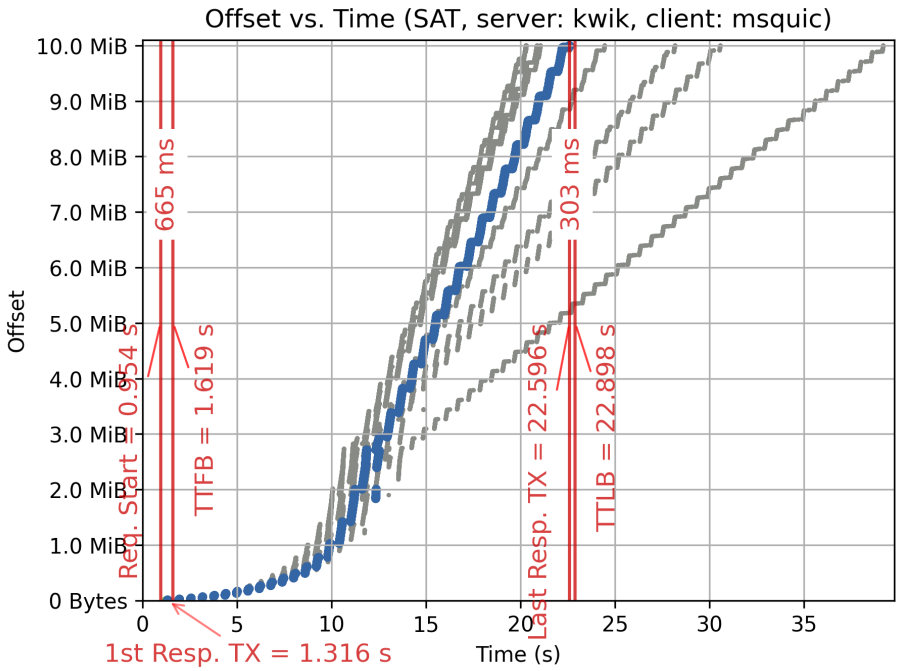


Figure 5.11 – Bend in Offset Plot of *Kwik* & *Msquic* in SAT Scenario

Some traces show a noticeable bend in the middle of the transmission like the one in Figure 5.11. The trace was generated by *kwik* as server and *msquic* as client. As this did not occur in every iteration the TTLB varies by factor 2 between about

20 s and 40 s. Note that this is an emulated no loss scenario. No reason could be found for the nondeterministic behavior.

A second flaw can also be detected in this trace: If the server had used proper pacing, there would not have been any bursts of packet transmissions, as they are clearly visible in the trace. The gaps between the bursts can be closed, when the packets are transmitted more smoothly with the estimated channel rate. Instead, *kwik* seems to send the packets in bursts and then stops to wait for `ACK`s from the client. This issue arises for most traces in the SAT scenario, when *kwik* is used as server.

5.6.6 Retransmissions

Using the packets captured at server side for generating the offset traces, retransmissions can be detected, when the same offset number appears repeatedly. These packets are marked in orange. In this section, different behavior caused by retransmissions is shown.

In Figure 5.12, we can identify different reactions to artificial losses in SATLOSS scenario by the same implementations. The shadowed trace with the shortest time to completion shows that the endpoints manage to transfer the chunks almost in-order at a constantly increasing rate while resending single missing packets as soon as it is

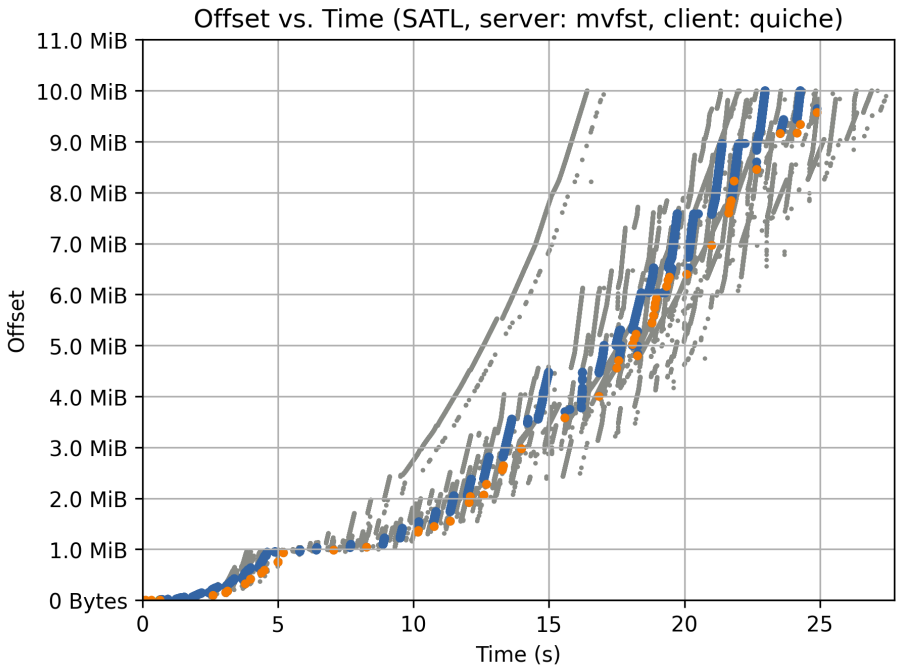


Figure 5.12 – Different Reaction to Losses of Same Implementation in SATLOSS

known that they did not reach the client 1 · RTT later. In this trace, SACK seem to perform well. However, during all other iterations the implementations seem to fail to increase the transmission rate and there are many gaps between bursts of packets sent to the client. It seems that the server does not continue sending new packets until ACKs for packets shortly sent before have arrived.

Figure 5.13 shows a common behavior of measurements with very poor goodput values in SATLOSS. Each iteration belongs to a different line with very variable transmission rates during the experiment. It seems that the server fails to estimate the correct bottleneck bandwidth of the channel while it mistakes the losses for indications of network congestion and buffer overflows, which is the approach of classical loss-based CCAs. In combination with such a high RTT, it results in a very long time to completion.

The transmission in Figure 5.14 starts quite usual with an exponential slow start phase, but after about 6 s, after sending about 6 MiB, the server starts to transmit packets with offset numbers between 3 and 4 MiB again. While it looks like a normal retransmission at first glance, parts of these offset numbers have not been sent before. Otherwise, they would be marked orange by our analysis. Instead, single packets seem to be skipped first and sent later. This can be seen in Figure 5.15 which shows a highly magnified area of the trace. If possible interleaving data packets should be avoided because otherwise the ACK range will be fragmented, which in return results

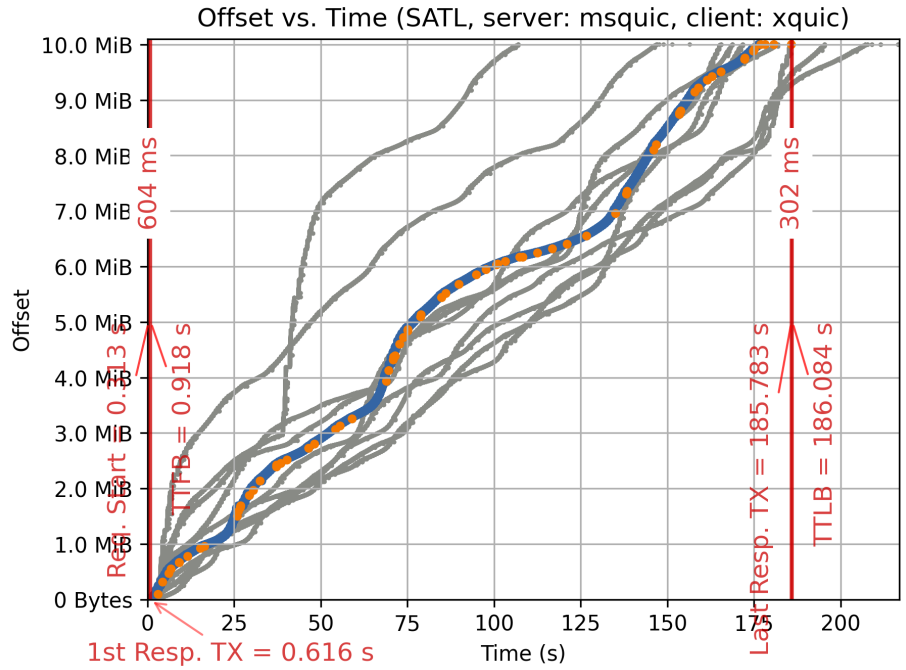


Figure 5.13 – Slow but Ordered Progress in SATLOSS

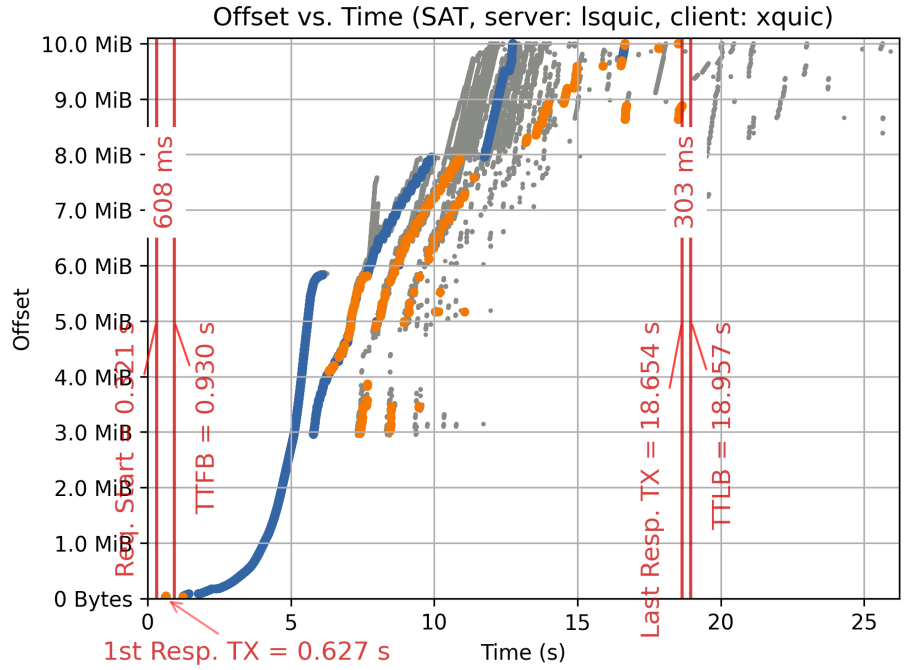


Figure 5.14 – Out-of-Order Transmission in SAT

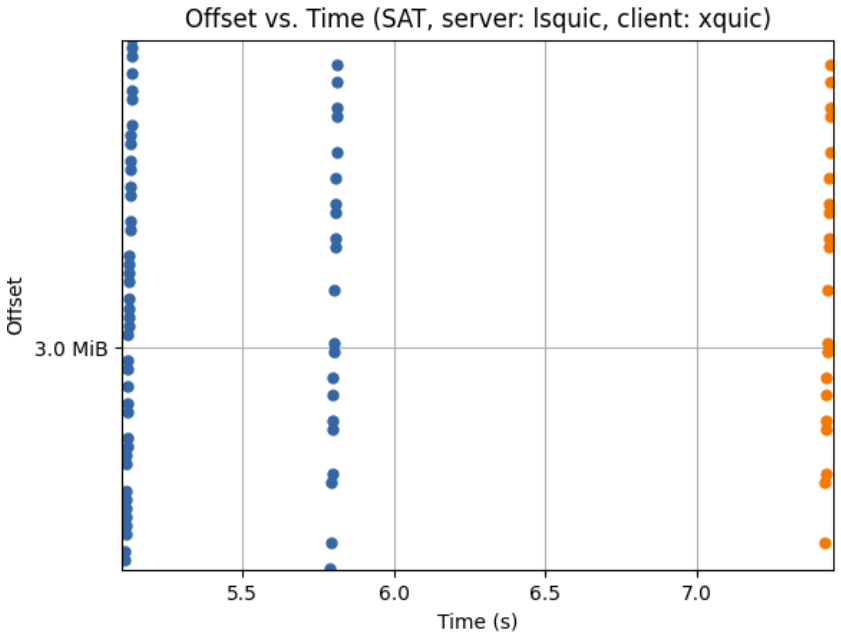


Figure 5.15 – A Detail of Figure 5.14 Highly Magnified.
Note: The markers in the second vertical trace are not retransmissions.

in fragmented buffers and larger ACK packets. Hereafter many retransmissions occur with a delay of slightly more than $1 \cdot \text{RTT}$. Since no artificial losses are introduced in SAT and since no retransmissions occur when other implementation are used, non-artificial, system-related losses seem to be unlikely. Instead, inappropriate timeouts chosen by the *lsquic* server whose traces show patterns like this often may cause this behavior which increases the time to completion dramatically.

The test case GOODPUT, which is not related to SATCOM, but still interesting, is also a no loss scenario. The trace of *msquic* with *xquic* in Figure 5.16 shows an interesting behavior, though: The transmissions of all iterations passes off quite fine with a constant goodput of 9.5 Mbit/s, but the last packets are always retransmitted multiple times. This indicates a bug in handling the end of transmissions. Waiting for the tail is very problematic because packet losses are only recognized after the loss timeout has passed since they can not be detected with the help of DupACKs or SACK packets for subsequent packets. To circumvent this issue, TLP was proposed (see Section 2.3.2.5). In this case, the problems at the end double the time to completion.

5.6.7 Early Retransmissions

A special pattern can be observed with *picoquic* as server in some ASTRA and EUTELSAT measurements. Figure 5.17 shows *picoquic* with *ngtcp2* in the ASTRA scenario. It

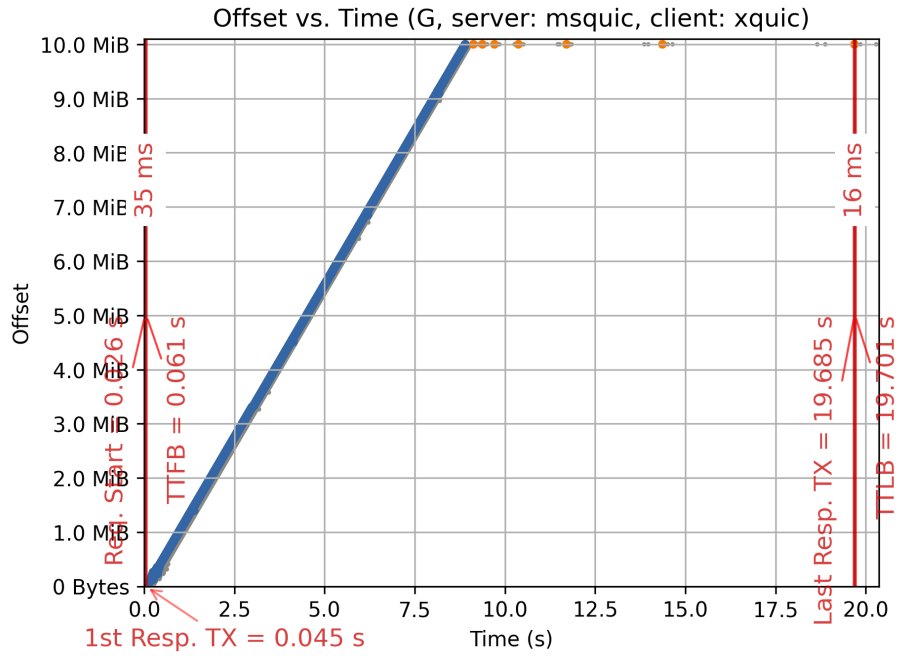


Figure 5.16 – Multiple Retransmission at End in GOODPUT

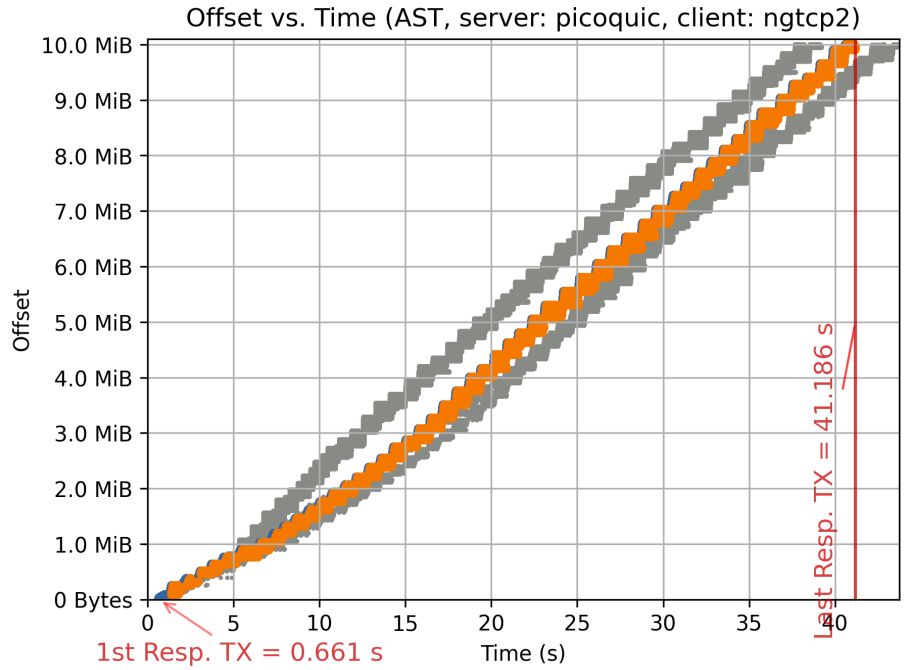
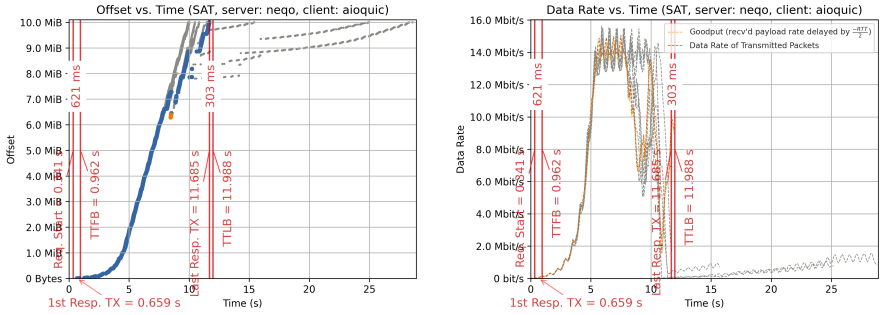


Figure 5.17 – Trace of *Picoquic* and *Ngtcp2* in ASTRA Scenario with a Conspicuous Number of Retransmissions

seems that the server retransmits each and every packet right after transmitting it the first time—less than one RTT later. Thus, the server cannot know whether the packet was lost or not. For the entire file of 10 MiB, about 80 MiB were transmitted in total, which equals a data rate of 16 Mbit/s while the goodput is only 2 Mbit/s. This very much fills the channel with a capacity of 20 Mbit/s. As *picoquic* is usually quite good at estimating the bottleneck bandwidth, this behavior might be intentional. When it realizes that the client does not allow filling the channel with data by setting the Receive Window (rcwin) to a too small value, it might send each packet repeatedly in hope that at least one of them arrives at the client undamaged. Thus, retransmissions requested by the client, which would result in a major delay, are avoided. Of course, we can only guess whether this is actually the case.

5.6.8 Long Tail Latency

Furthermore, some traces show a sudden drop of the data rate at a certain point in time. The transmission proceeds quite well until then and for smaller files everything would be fine. In Figure 5.18, during the highlighted iteration the server retransmits packets after 7 s, then after 10 s it even stops and waits for some time until it resumes sending at the same data rate as before. In some other iterations of the measurement,



(a) Offset Plot **(b)** Data Rate

Figure 5.18 – Trace of *Neqo* and *Aioquic* in SAT Scenario with a High Drop of Data Rate

the rate drops to a very low value followed by a very slow exponential growth. This growth is similar to the startup phase, with the difference that it doesn't reach a steady state shortly after. The plot on the right side visualizes the data rate, i.e., the derivation of the offset plot as explained before. Both exponential increases can be seen very clearly on the gray traces, but the second exponential phase has a much smaller gradient. While both implementations sometimes perform well together, the occasional drops lead to a high tail latency.

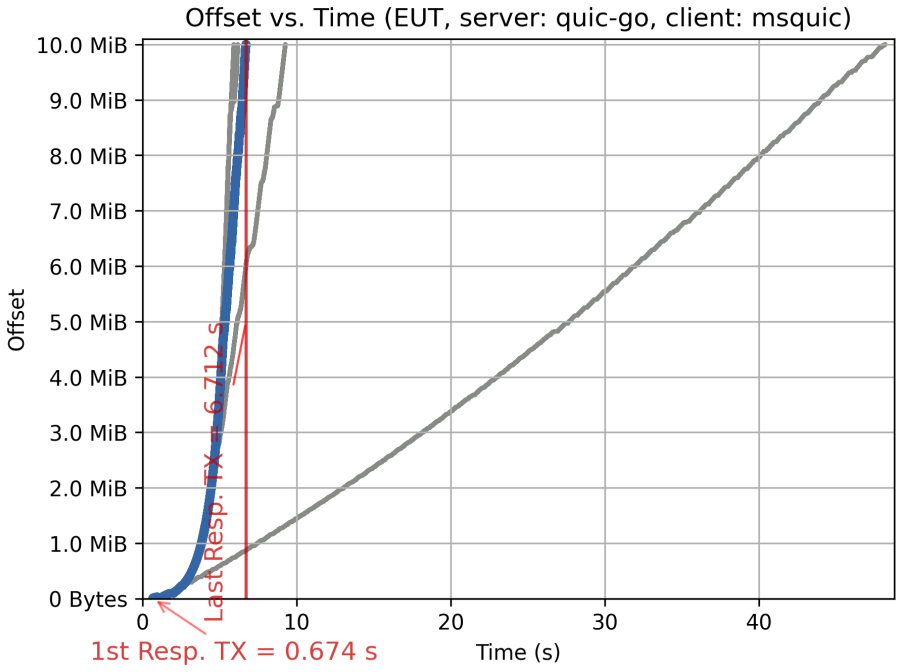


Figure 5.19 – *Quic-go* and *Msquic* with High Tail Latency

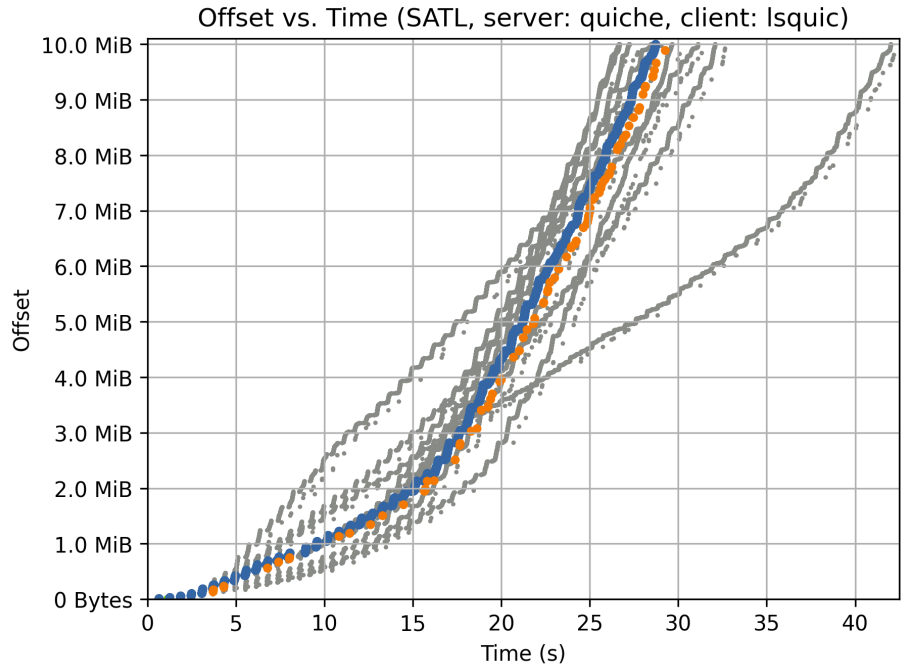


Figure 5.20 – *Quiche* and *Lsquic* with High Tail Latency

Figures 5.19 and 5.20 present two other examples for combinations with high tail latency that share a common pattern. On average the channel capacity is estimated more or less correctly, but sometimes the transmission remains at a lower data rate. Both outlier curves bend slightly to the left after some more seconds. At least in the case of *quiche* with *lsquic* in SATLOSS shown in Figure 5.20, it reaches the same data rate as in the other iterations at the end. However, the time to completion is about 25 % higher than on average in this measurement and even about 7 times higher in the measurement of *quic-go* with *msquic* via EUTELSAT.

5.6.9 Other Plots and Evaluations

The tools presented in Section 4.2.4 are capable of generating other traces besides offset plots. In this section, we briefly summarize the findings we gained from them.

5.6.9.1 Data Rate on Return Link

We analyzed the data rate on the return link, i.e., the link from client to server by creating plots similar to Figure 5.7b in order to check whether the return link might limit the transmission on the forward link. While data traffic travels from server to the client, the client has to answer with `ACK` packets in the reverse direction. When high load on the reverse paths slows down the flow of `ACK`s, the server can not transmit

data fast. This issue was analyzed by Ana Custura et al. [63] / Section 3.2.12. The range of data rates we observed on the return link is quite wide, between a few dozen of kbit/s and a maximum of 600 kbit/s. This is much less than the available path bottleneck, which is at least 2 Mbit/s (in SAT, SATLOSS and ASTRA). So the ACK rate should not influence the overall performance in our case. It is interesting that there does not seem to be a uniform pattern. Some transmissions generate high return link utilization at the beginning of the transmission, for some it scales with the utilization of the forward link, and sometimes there are even significant peeks right in the middle of the transmission.

5.6.9.2 Packet Number Plots

While TCP packets contain only a sequence number, QUIC packets carry a packet number and STREAM frames an offset number. We used the latter one to generate the offset plots as shown before. When the packet number is plotted over time, the resulting graphs usually look quite similar, when the implementation use consecutive numbers. One difference is that the offset number is limited to the file size while packet numbers can become arbitrary large, when there are retransmissions. We found one implementation handling packet numbers differently than the others: *Mvfst* uses a random but constant packet number for the entire transmission. This contradicts the specification: “Packet numbers [...] start at packet number 0. Subsequent packets [...] MUST increase the packet number by at least one. [...] A QUIC endpoint MUST NOT reuse a packet number [...] in one connection. ” [36]

5.6.9.3 Packet Sizes

We also plotted the QUIC packet size over time and identified some implementations, like *aioquic* that use values of slightly more than 1200 B, which would be the minimum value as specified in RFC 9000 [36]. Some implementations, like *lsquic*, *msquic*, *neqo*, *ngtcp2*, *picoquic* increase the packet size during transmission usually to a value around 1400 B which indicates the usage of PMTUD or DPLPMTUD as described in Section 2.3.2.3. There are also some implementations, especially *nginx* and *xquic*, which drop either regularly or only for individual packets to a packet size that is only fractions of the MTU. This can have numerous reasons, which were not analyzed in more detail. Generally it is advised to use large packets on high BDP paths in order to fill it more easily—at least in low loss scenarios.

5.7 Verification of Test Results

In the last section, we want to evaluate the validity of the measurement results.

5.7.1 Comparison with Results of Official QIR

After porting the QIR setup to own machines and after modifying the source code of the runner it is crucial to verify that the test and measurement results are still identical to the results before.

5.7.1.1 The LONGRTT Test Case

The symmetric channel and the chosen parameters of the LONGRTT test case are not typical for a SATCOM access using geostationary satellites, as explained in Section 4.2.1. However, as it is already part of the official QIR, we use it anyway, to verify that our modified setup still produces the same results. In our setup as well as in that by Marten Seemann, apart from the currently unsupported implementations that are mentioned in Section 5.2.1, all implementations manage to complete the handshake in such challenging scenarios.

Long Term Evaluation of Test Case LONGRTT

To monitor the activity of the developers of QUIC implementations and to assess the reliability of the measurement values, we have been downloading and storing the test results of the official runner since 2021-05-03 because the results will be deleted after a few days on the website. Analyzing the data set for the LONGRTT test case gives following insights:

- 65 % of the combinations always or almost always succeed
- 23 % always failed. Among them are combinations with *chrome* (see Section 5.2.1.1) or *quicly* (see Section 5.2.1.2), but also the same single combinations of implementations that fail in our setup[50].
- Most of the other 12 % are combinations with *mvfst* where most of them got fixed either on 2021-08-29 or 2021-08-26 and are succeeding since then.

On 2021-10-29 experiments using *ngtcp2* as server broke for three repetitions, but after that it succeeded again. The combination *kwik-mvfst* broke in the mid of October 2021. It is hard to say whether these changes are caused by changes in the code base of the implementations or by changes in QIR. However, it can be said that the results produced by QIR are trustworthy and not heavily influenced by coincidence.

[50] *aioquic-msquic*, *lsquic-picoquic*, *mvfst-msquic*, *mvfst-xquic*, *xquic-msquic*

5.7.2 The GOODPUT Measurement Test Case

In Figure 5.21, the distribution of average measurement results of the GOODPUT measurement test case are visualized using boxplots. On the left side, the latest values that are available on the official QIR website at time of writing (2022-01-07), are used. On the right side our values are shown. For this plot, we only considered values of experiments that succeeded both locally and on the official interoperability server.

The mean of the results on the left side is about 9.1 Mbit/s, while the mean goodput value of our measurements, is at about 9.0 Mbit/s. With less than 1 % deviation this is quite close. The median and maximum values are equal, and also the variance of the measurement results is in the same order of magnitude.

We can hence conclude that the results of our setup for the GOODPUT measurement are comparable to the results of the official QIR setup.

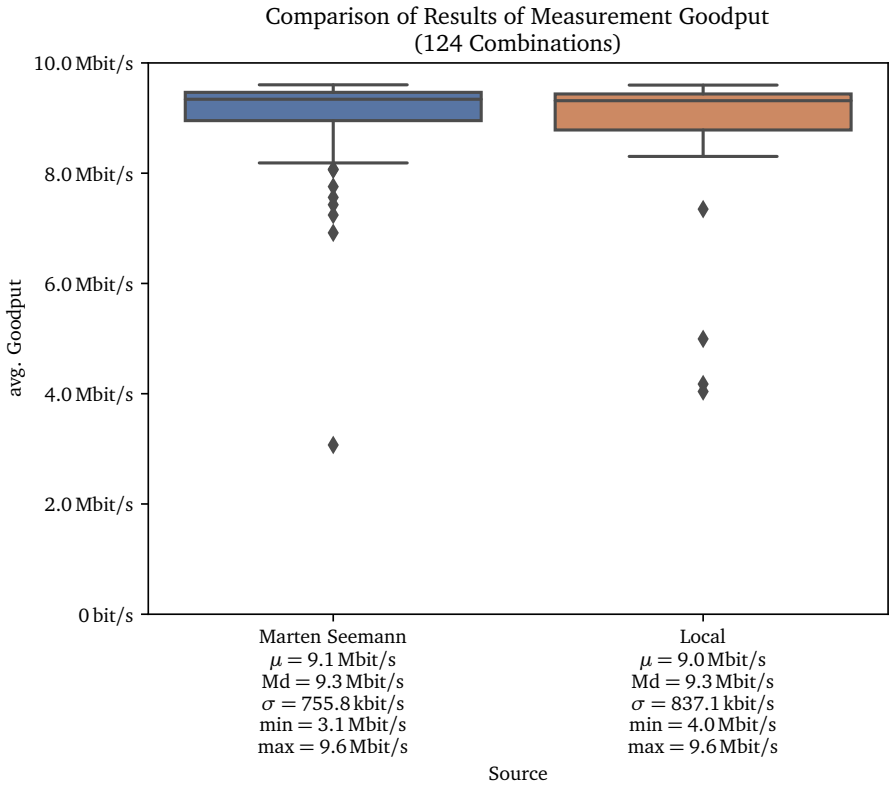


Figure 5.21 – Measurement Results of the GOODPUT Measurement When Executed on Own Setup Compared to the Latest Results Available Online

5.7.2.1 Long Term Evaluation of Measurement Test Case GOODPUT

To assess the significance of the measurement values in Figure 5.21 we inspected the GOODPUT results of the official QIR over several months, like before. In Figure 5.22, the average measurement values of each implementation combination are aggregated by their server and plotted over the time. For visual reasons, all values are also aggregated by each day, which slightly reduces the variance of each series.

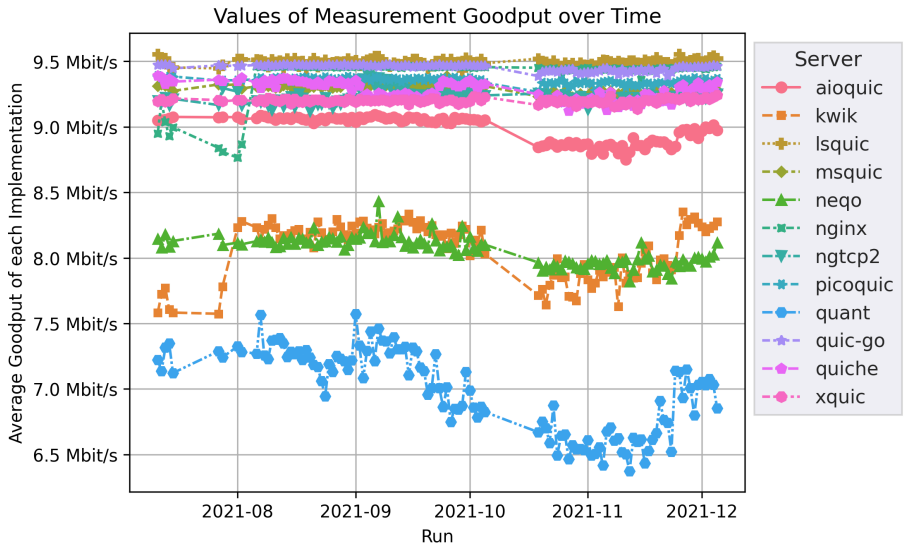


Figure 5.22 – Values of Measurement GOODPUT of the Official QIR Over Several Months

Over the last months the order of magnitude of the performance of most implementations remains stable.[51] During the inspected time range only *nginx* and *kwik* made a leap of about +0.75 Mbit/s at the beginning of August. Around the beginning of October the performance of multiple implementations drifted downwards though. It may be doubted that this was caused by changes in the code base of the implementations. Higher utilization of the GitHub Actions servers seems more likely, as explained in Section 4.1.6.2. However, we can ascertain that the measurement results are usable for qualitative performance determination.

[51] Early research results as presented by Tom Jones at *2nd QUIC and Satellite Open Stakeholder Meeting* on 2021-12-02 show that at least *quicly* improved over the past 3 years with only a little enhancement during the last year. This evaluation covers only the last 3 month and only the performance in the environment of the interoperability runner which might differ from the actual performance because of different configuration of the implementations for the test scenarios. `https://erg.abdn.ac.uk/video/2nd%20ESA%20MTAILS%20Satellite%20Stakeholder%20Meeting/1.1%20Evolution%20of%20QUIC%20and%20Satellite%20over%20the%20Last%203%20Years.pdf` slide 13 *(visited on 2022-01-03)*

5.7.3 Comparison with Results of Other Research

In this section, we try to verify if our results are in line to the results of previous research activities. Suitable values from [3] and from the papers presented in Section 3.2 with link parameters similar to our scenarios are presented in Table 5.2. The payload size is annotated with 🗎, when websites have been used as payload. In the last column, the corresponding goodput values are calculated, to make it easier to compare it with our results.

Jones et al. present some empirical performance values of QUIC in the appendix of [3]. The larger link with 50 Mbit/s in the forward direction is comparable to the link that we have used in EUTELSAT. As TTLB, 5 s are specified, which equals a goodput of 16.78 Mbit/s. On the one hand, this is more than twice as high as the average goodput which we have observed (see Table 5.1). On the other hand, *picoquic*, according to our results one of the best performing implementations, achieved on average roughly 15.5 Mbit/s, when used as client and as server. This is in the same order of magnitude. Our results of *picoquic* for ASTRA are with 10.1 Mbit/s only slightly better than the results in mentioned in the draft for a 10/2 Mbit/s connection (second row of the table). However, the data rate of the forward link is in our case with 20 Mbit/s twice as high as in the draft. It has to be noted that the reliability of the values in the draft is questionable, as no further statements about the measurement setup are provided.

In Ⓕ, Deutschmann et al. used three different satellite operators [5]. While quite similar results are achieved for the first and the second one, the third one, which provides the lowest data rate, performs best. The paper concludes that the performance depends not only on the transport protocol, but also highly on the satellite access. We could not use *chrome*, as explained in Section 5.2.1.1,

Table 5.2 – Selected Measurement Results from Literature

Source	Implementation	Link Rate [Mbit/s]	RTT [ms]	PLR	Size [MiB]	PLT / TTLB [s]	Goodput [Mbit/s]
Jones 2021 [3]	?	50/10	650	—	10	5.0	16.78
	?	10/2	650	—	10	10.0	8.39
Ⓕ Deutschmann 2019 [5]	*chrome*	20/2	*real Link*		10 🗎	75.0	1.12
	chrome	30/2	*real Link*		10 🗎	73.0	1.15
	chrome	15/3	*real Link*		10 🗎	15.0	5.59
Ⓘ Mogildea 2019 [61]	*ngtcp2*	20/2	*real Link*		1	8.5	0.99
	ngtcp2	20/2	600	0 %	1	8.0	1.05

but the average results for SAT (4.88 Mbit/s), SATLOSS (2.86 Mbit/s) and ASTRA (3.98 Mbit/s), as they are listed in Table 5.1, are between the results of the three operators. For all these measurements, the data rate is 20/2 Mbit/s, which is more or less in the same order, at least taken into account that data rate does not matter that much compared to RTT, as explained in Section 4.2.3. In contrast to the paper, we use IETF QUIC instead of gQUIC and use bulk transfer instead of transferring websites, which might cause differences.

Mogildea et al. compared *chrome*, *quicly* and *ngtcp2* with different satellite accesses and with a simulated link in ① [61]. We could unfortunately not get any results for *chrome* and *quicly*, as explained in Section 5.2. However, as the same access was used, like we did in ASTRA, we can compare the difference of *ngtcp2* in this and the emulated scenario. The payload size in the paper is 1 MiB unlike our measurements which use 10 MiB. Because of the typical exponential start-up phase and the long time required for the handshake, like it can be seen in the traces shown in Section 5.6, the results can not be compared using absolute values. In our measurements, *ngtcp2* when used as server and as client achieved an average goodput of 1.554 Mbit/s in ASTRA and 1.572 Mbit/s in SAT, which is a deviation of less than 2 %. In the paper, the difference is roughly 5 %, which is comparable. The absolute goodput values are, as expected, lower than ours because for larger files the slow start phase weighs less.

5.7.4 Evaluation of Satellite Links

When running measurements with real links instead of simulated networks, it is important to ensure that the link quality is stable. In literature, the RTT is often monitored during measurements to proof that it is more or less constant. Unfortunately, no such data is available for our measurements. However, previous measurements on our satellite accesses show that the RTT is usually between 600 and 1000 ms for the accesses we use and PLRs are quite constant, as explained in Section 4.2.2.

To limit the influence of the utilization, measurements have been automatically paused during prime time, i.e., between 6 p.m. and 11 p.m. (see Section 2.1.3.3). Weather was monitored for the region of the user terminal. It was mostly constant cloudy with no major precipitation and a fairly constant temperature around 6 °C as shown in Figure 5.23.

Figure 5.24 visualizes all goodputs values over time which have been measured via both satellite Internet accesses, *Eutelsat Konnect* and *Novostream Astra Connect*. Each combination of client and server was scheduled 5 times and each scheduled execution was randomly shuffled with the execution of other combinations. Assuming that the values of goodput only depend on the performance of the implementations and not on the link quality, this would result in a stochastically stationary random

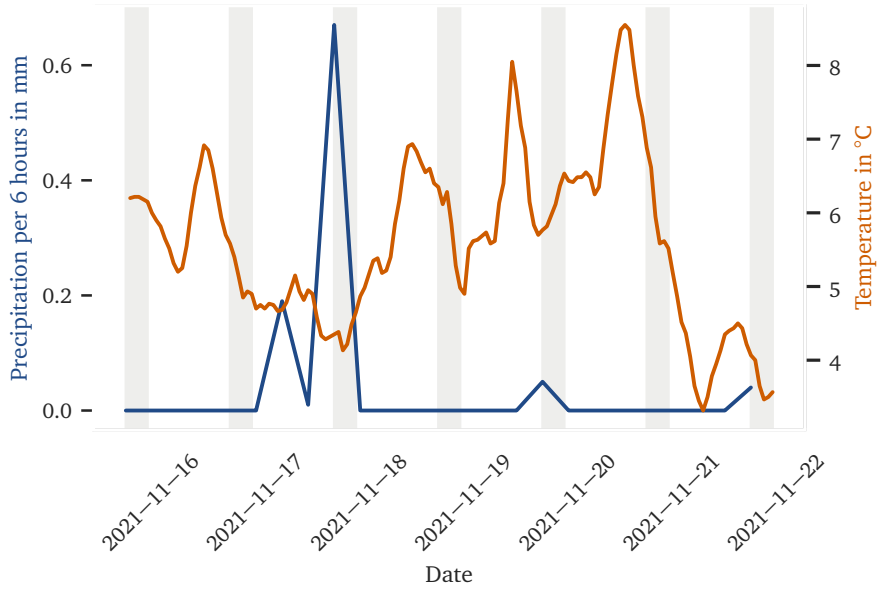


Figure 5.23 – Weather in Nuremberg At the Time of Mesurement
The data for the station №3668 of the Deutscher Wetterdienst was fetched from https://opendata.dwd.de/ on 2021-11-22. Measurements were taken between 2021-11-15 and 2021-11-22.

process. The regression lines, added to the plots, show no significant trend. Especially in the plot on the right side there seems to be no timely variations over the day. Instead, the results indeed seem to be independent of the time they were recorded. Thus, it is safe to assume that the utilization of the satellite was quite constant during time of measuring.

We can also confirm that we didn't exceed the traffic limit of 60 GiB/month for the *Eutelsat Konnect Zen* access. According to `vnstat`, about 10 GiB were received and about 2.3 GiB were transmitted on the sole client computer connected to the Eutelsat modem during the month in which the experiments were performed.

In both plots, there is a gap between 6 p.m. and 11 p.m. as mentioned before. The additional gap between 6 a.m. and 8 a.m. is a coincidence, caused by the fact that the measurements completed at about 6 a.m. and new measurements have been scheduled at about 8 a.m. Most EUTELSAT measurements have been postponed after ASTRA measurements because of a technical defect. It is unclear whether there was some outage at Eutelsat or simply a technical defect in the test setup. The single data points on November 16th and 17th were caused by the fact, that, after running some experiments, we detected a bug in the setup which resulted in broken Pcap traces, as described in Section 4.1.4. It was fixed before running the majority of the measurements. This bug had no effect on the measurement results, but some

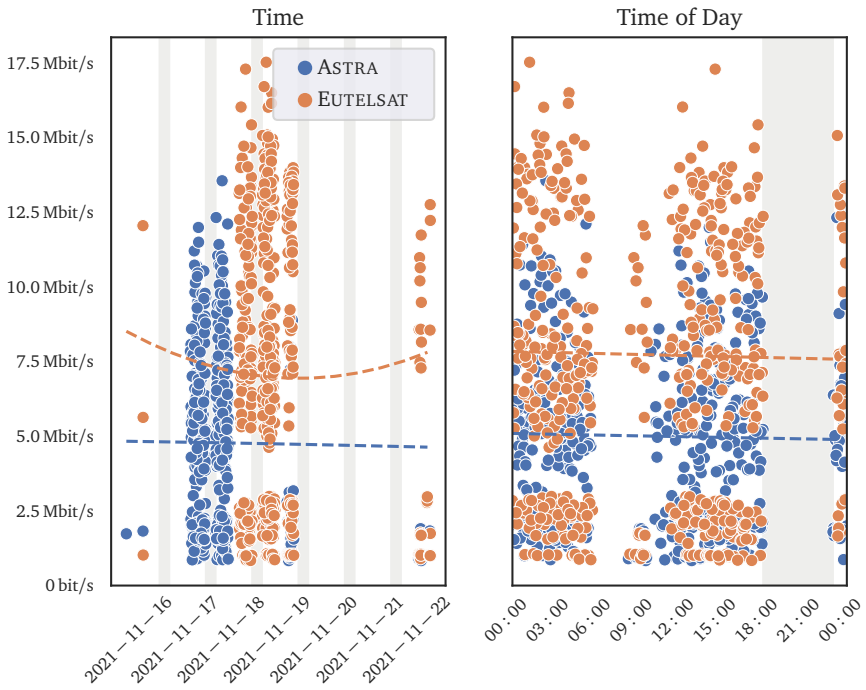


Figure 5.24 – All Measurement Values of Using Real Satellite Links Over Time. On the left side the entire time span is used on the x-axis. On the right side the time of day is displayed on the x-axis.

experiments had to be repeated later, which can be seen on the right side of the left plot.

Chapter 6

Conclusion

In this thesis, we have analyzed the challenges for transport protocols on Internet links via geostationary satellites and presented countermeasures already taken to accelerate TCP. Especially PEPs are used, which break the end-to-end semantic of TCP connections. We have also summarized the most relevant features in the context of SATCOM of the new transport protocol QUIC, which is built to replace the ossified TCP in the HTTP stack. The reduced handshake, the elimination of HoLB and the extensibility are interesting for SATCOM. While the built-in encryption of the headers enhances privacy, it also prevents satellite operators from using PEPs.

The analysis of current research on QUIC and SATCOM shows that QUIC exhibits lower performance than TCP in combination with PEPs. The main reason is the high delay on satellite links, which most non-specialized CCAs cannot handle. However, previous research neglects the influence on the performance of the implementation chosen to run the measurements. Additionally, many research papers use *chrome* with gQUIC, but only few use IETF QUIC.

Therefore, we have presented a test bed that allows running measurements with numerous implementations of QUIC. It is a modified version of the QUIC-Interop-Runner used by the IETF to perform interoperability tests of available implementations to check if developers are able to understand and implement the specification correctly. We have extended the runner with measurement scenarios that emulate different satellite links. Additionally, it is now possible to use real satellite Internet accesses for measurements.

Finally, we have presented measurement results of many popular implementations of IETF QUIC. While some implementations supported by the official runner failed to run on our setup for technical reasons, we still could use most of them. Many QUIC servers and clients fail to download a medium-sized file over satellite links because they either do not manage to transfer it in a very generous time interval, or they end up in a faulty state. While the successful implementations achieve similarly

good results in scenarios with low RTTs, almost all reach a rather low link utilization in satellite scenarios, while the differences between implementations is very high. Increasing the net data rate does not automatically increase the goodput in the same ratio. We tried to determine if the client or the server has a larger impact on the overall performance, but we have to conclude that both contribute to it. There are also some implementations that perform well as client, but the corresponding server implementation performs poorly or vice versa. Additionally, some are achieving slightly better results on satellite links without loss but only very poor results at a PLR of 1 %. A detailed analysis of the reasons for this was out of scope of this work, but the Congestion Control Algorithm seems to be a good starting point. By analyzing individual traces of transmissions, we found flaws that have to be addressed by the developers.

We were able to show that IETF QUIC can be used in GEO networks, but the performance is on average very poor. However, downgrading to TCP, like some operators do it right now by blocking QUIC traffic, is not a long-term solution. Instead, other solutions have to be found, and the endpoints have to take such challenging scenarios into account.

6.1 Future Work

QUIC parameters and current implementations should be optimized for paths with high latency. This could involve extensions such as 0-RTT-BDP [45].

Otherwise, a possible solution might also be to introduce proxies similar to PEPs for QUIC to enhance the performance of end-to-end connections. Currently, an IETF working group is designing explicit proxies for QUIC, called Multiplexed Application Substrate over QUIC Encryption (MASQUE)[52]. Apple already deployed similar techniques at large scale to build iCloud private relays[53]. They might be used to achieve path-wise congestion control for sub-paths with different properties. As mobile operators also experience related issues in cellular networks, research is already underway to deploy such proxies in 5G networks [70]. They call it multi-domain congestion control, but only little information is available yet. E.g., it is not clear how the problem of having a CCA inside a congestion-controlled connection is solved. However, newly introduced middleboxes should be specified very precisely to avoid a renewed ossification of the Internet. Mechanisms in endpoints that make it possible to achieve high performance while preserving the end-to-end principle of the transport protocol would be more desirable.

[52] MASQUE working group: `https://datatracker.ietf.org/wg/masque/` *(visited on 2022-01-06)*

[53] Documentation about iCloud Private Relay for Network Operators: `https://developer.apple.com/support/prepare-your-network-for-icloud-private-relay` *(visited on 2021-12-21)*

Appendix A

Screenshot of Result Website of QIR

The measurement results, files, and plots are available online. The data is presented on an interactive website, which is available on: `https://interop.sedrubal.de/` *(visited on 2021-12-23)*

Figure A.1 is a partial screenshot of the website.

Figure A.1 – Partial Screenshot of Result Website of QIR Showing the Measurement Results

Appendix B

Distribution of Measurement Results by Implementation

In this section, the estimated distribution of measurement results by implementation for the scenarios GOODPUT, ASTRA and EUTELSAT are attached (Figures B.1 to B.3). The plots for SAT and SATLOSS are embedded above (see Figures 5.4a and 5.4b).

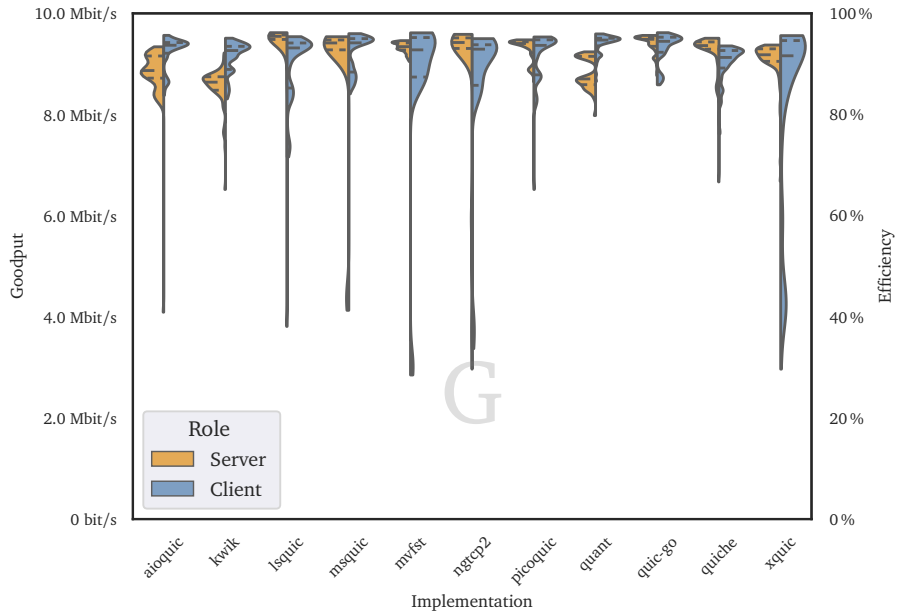


Figure B.1 – Distribution of Results by Implementation for GOODPUT

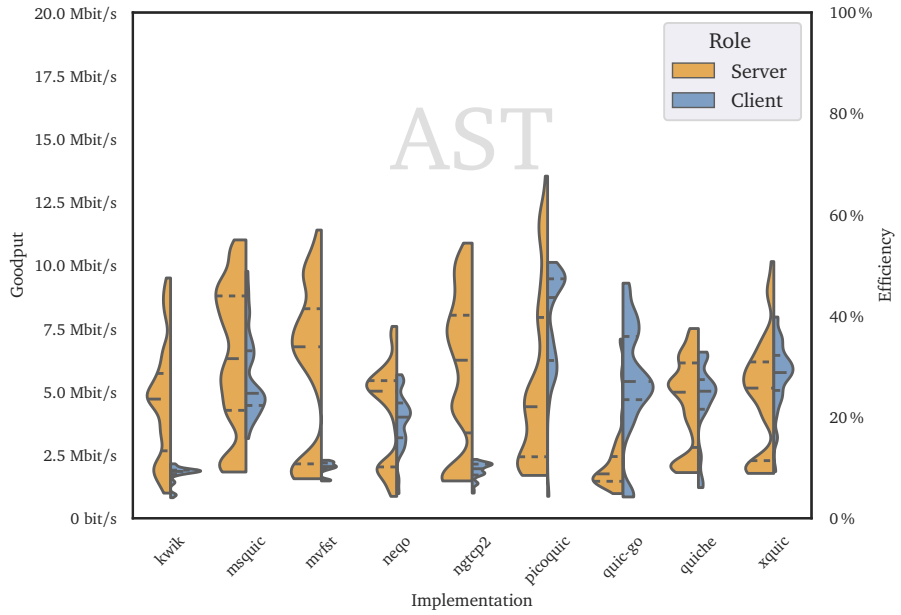


Figure B.2 – Distribution of Results by Implementation for ASTRA

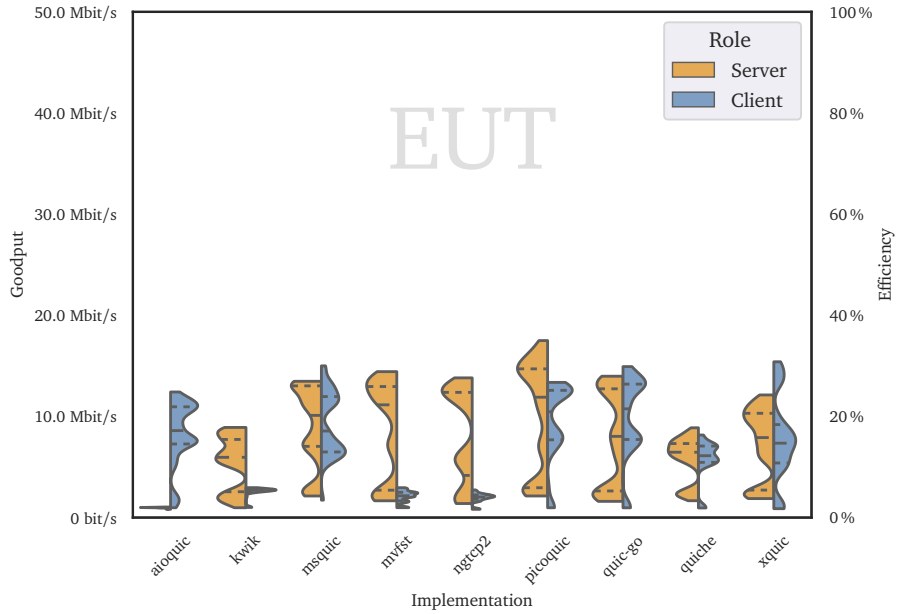


Figure B.3 – Distribution of Results by Implementation for EUTELSAT

Appendix C

Evolution of Congestion Control Algorithms

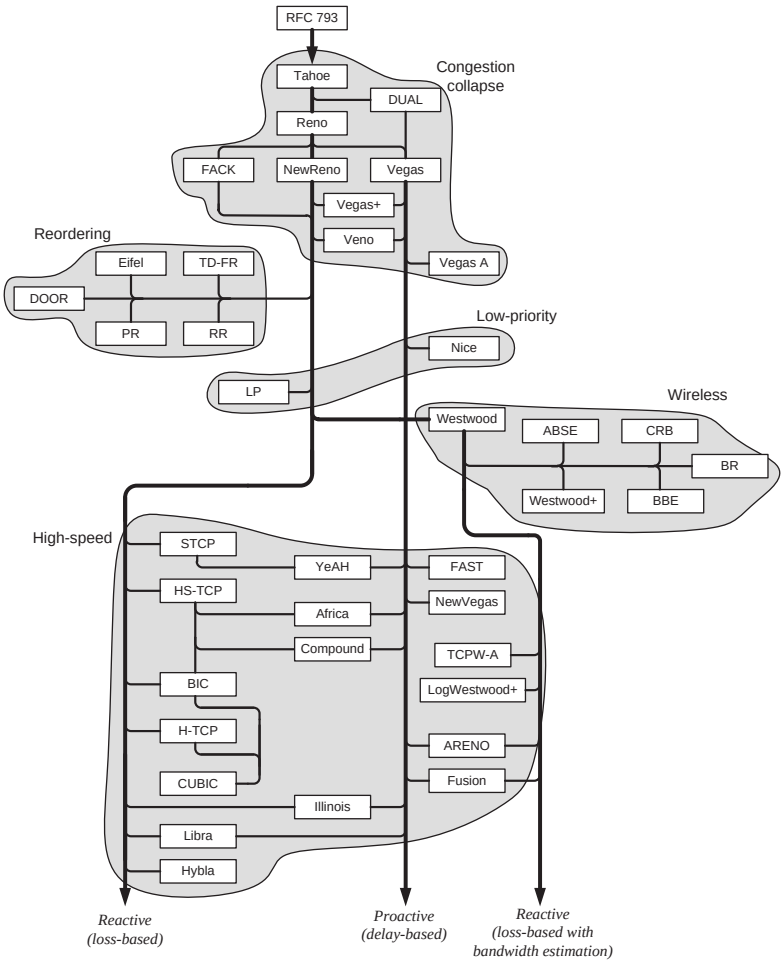


Figure C.1 – Evolutionary Graph of Variants of TCP Congestion Control. Taken from [71]. Since this graph is created in 2010, BBR is missing. It belongs to the group of delay based algorithms.

Appendix D

Related Research

A web version of the table on the next pages is available online:
`https://github.com/sedrubal/QUIC_HIGH_BDP/blob/76ab9cc036a0a039e5a550a02263aad198428c06/research_overview.md`
(visited on 2021-12-23)

Measurements of QUIC implementations via Satellite Link

Author	Title	Date	Type of Research[1]	Research Focus	QUIC Ver.	QUIC Client	QUIC Server	QUIC Settings	HTTP in QUIC	Type of Benchmark	Type of Evaluation	Link Type[2]	Link Parameters	Comparison with non-QUIC protocols	TCP Settings	Remarks
John Rula et al.	Mile High WiFi: A First Look At In-Flight Internet Connectivity	04/2018	Paper	Targeting poor In-Flight Communications performance	*unspecified*	*unspecified*	*unspecified*	*unspecified*	*unspecified*	Websites (1; 2; 5; 10 obj. a 100; 200; 500; 1000 KB)	PLT	emul.	RTT 761; 380.5 ms rate (sym.) 1.89 3.78 Mbps PLR 6; 3 %	*no PEP* TCP ??? HTTP/1.1 and 2	*unspecified*	QUIC measurements are quire rudimentary but trends are visible.
Siyu Yang et al.	Performance Analysis of QUIC Protocol in Integrated Satellites and Terrestrial Networks	06/2018	Paper	Performance of QUIC in space-terrestrial integrated netw.	gQUIC Q035	Google Chrome	quic-go	*unspecified*	[5]	Website (400 KB)	CDF of PLT	emul.	RTT <20; 40; 300; 500 ms rate (sym.) 10 Mbps PLR 0; 1; 10; >12; 1.6; 3 %	**no PEP** TCP TLS? HTTP/2	"Cubic Reno", w/o & with ECN ("TCP", "TCP+")	Quite a messy paper
Han Zhang et al.	How Quick Is QUIC in Satellite Networks	06/2018	Paper	First Measurements of QUIC Perf. via Sat Link	gQUIC Q039	Google Chromium	Google QUIC test server (was part of proto-quic)	CUBIC, 0RTT, MUX	[5]	Websites (344 KB; 784 KB; 2.3 MB)	PLT	emul.	RTT 200; 400; 600 ms rate (sym.) 256 kbps; 512 kbps; 1 Mbps BER 10^-7; 10^-6; 10^-5	**no PEP** TCP TLS 1.2 HTTP/1.1 & 2	MTU=1500 B IW=10 *default*	
Wang Yue et al.	Performance Evaluation of QUIC with BBR in Satellite Internet	12/2018	Paper	Performance of QUIC with BBR as cc algorithm in GEO netw.	gQUIC Q039	Google Chromium	Google QUIC test server	BBR	[5]	Websites (344 KB; 784 KB, 2.3 MB)	goodput (diff. PLRs & over time)	emul.	RTT 200..600 (or 1000?) ms; rate (sym.) 1M; 10M PLR 10^-5..2*10^-1	*TCP setup is described, but no measurements using TCP are provided*		
Ludovic Thomas et al.	Google QUIC performance over a public SATCOM access	02/2019	Paper	Measurements over real sat link compared to 4G	gQUIC Q039	Google Chrome 67	Google Server (404 page & some image)	BBR, 0RTT, IW=32	HTTP/2	File (5.3 MB); Website (11 KB)	elapsed time (box plot); time-sequence	real	RTT 750 ms rate 25/5 Mbps	PEP TCP TLS 1.2 HTTP/2 ("ChromeNoQuic")	TFO	
Jörg Deutschmann et al.	Satellite Internet Performance Measurements	03/2019	Paper, IETF	Measurements of different HTTP vers. via sat link	gQUIC Q043	Google Chrome 69	Chromium QUIC; quic-go	*default*	[5]	File (10 MB); Websites (1.4 MB; 10 MB)	PLT (box plot)	real[3]	RTT 600 - >700 ms rate 5-15/2-6 Mbps	PEP & OpenVPN TCP no TLS & TLS? HTTP/1.1 & 2 diff. Operators	CUBIC SACK W scaling IW=10 no ECN	
Gorry Fairhurst et al.	Measuring QUIC Dynamics over a High Delay Path	07/2019	IETF	Trigger discussion about poor QUIC performance at IETF	draft-20	quicly v20	quicly v20	Reno, IW=10, MSS=1460	[5]	Files (100 KB; 1 MB)	elapsed time; time-sequence plot	real	RTT >550ms rate 8.5/1.4 Mbps	PEP & OpenVPN TCP TLS 1.2 & 1.3 HTTP/?	CUBIC SACK W Scaling IW=20/10 MSS=1460/1358	
Konrad Wolsing et al.	A Performance Perspective on Web Optimized Protocol Stacks: TCP+TLS+HTTP/2 vs. QUIC	07/2019	Paper	Fair comparisons of opt. TCP and QUIC (not only for SATCOM)	gQUIC Q043	Google Chromium 70	Google QUIC test server	*def.:* IW=32, pacing, CUBIC; BBR	HTTP/3	real Websites	FVC, SI, VC85, PLT (CDF of gain against TCP)	emul.	For "MSS": RTT 760 ms rate (sym.) 1.89 Mbps PLR 6 %	*no PEP* TCP TLS 1.3 HTTP/2	CUBIC & BBR IW=10 & 32 pacing on & off tuned buffers, no slow start after idle	Different scenarios have been evaluated; "MSS" is the only relevant scenario for

Author	Title	Date	Type of Research[1]	Research Focus	QUIC Ver.	QUIC Client	QUIC Server	QUIC Settings	HTTP in QUIC	Type of Benchmark	Type of Evaluation	Link Type[2]	Link Parameters	Comparison with non-QUIC protocols	TCP Settings	Remarks
																sat com.
Cristian Mogildea et al.	QUIC over Satellite: Introduction and Performance Measurements	09/2019	Paper	Summarize status quo; measurements of implementations	Q046; draft-22; draft-22	Chromium QUIC; quicly; ngtcp2	*same as client*	CUBIC; Reno; unspecified	[5]	File (1 MB)	time-sequence plot	real, emul.	RTT 600 ms; >600 ms rate 20/2 Mbps; 16-30/2-3 Mbps PLR 1 %	PEP & no PEP TCP TLS ? HTTP/2	CUBIC SACK W scaling no ECN	
John Border et al.	Evaluating QUIC's Performance Against Performance Enhancing Proxy over Satellite Link	06/2020	Paper, IETF	Compare performance of QUIC with older HTTP versions	gQUIC Q046	Google Chrome 77	Google Drive (no details)	*unspecified*	HTTP/2	File (1 GB)	goodput (box plots)	real, emul.	RTT ~600 ms PLR 0 %; 0.1 %; 1 %	PEP TCP ??? HTTP/1.1 & 2	*default*	
Nicolas Kuhn et al.	QUIC: Opportunities and threats in SATCOM	10/2020	Paper	Highlight opportunities and threats of QUIC in SATCOM	gQUIC ?	Google Chrome 67	Google Server (no details)	*unspecified*	HTTP/2[5]	Websites (11 KB; 5.3 MB; <2 obj.)	time-sequence plot	real[4]	*unspecified*	PEP TCP TLS 1.3? ???	*default*	Also goodput analysis via lossy channel (PLR? 0.01%; 0.05%; 0.1%; 0.5%); omitted, because of lack of details
Ana Custura et al.	Impact of Acknowledgements using IETF QUIC on Satellite Performance	10/2020	Paper	Influence of ACKs on performance via asymmetric links	draft-27; draft-26	quicly; Chromium QUIC cli	*same as client*	Reno; BBR	HTTP/3	File (100 KB)	elapsed time (box plot)	real, emul.	RTT 600 ms; ~630 ms rate 8.5/1.5 Mbps; 10/2 Mbps (nominal) / 8.5/1.5 Mbps (avail.) PLR 1 %; no artif. PLR	PEP & no PEP TCP TLS 1.2 & 1.3 HTTP/2	Reno SACK W scaling	Measurement data of PicoQUIC was also provided by someone else and results have been compared with PicoQUIC
Nicolas Kuhn et el.	Feedback from using QUIC's 0-RTT-BDP extension over SATCOM public access	07/2021	IETF	Evaluate gain of 0-RTT-BDP extension	*unspecified*	picoquic	picoquic	BBR; 0-RTT-BDP (local & frame)	yes	File (0.5; 1; 10; 100 MB)	Used Bandwidth in %; elapsed time (table)	emul.	RTT 100..500 ms rate 1/0.1; 10/2; 50/25; 200/100 Mbps			

1. `Paper`: Scientific paper; `IETF`: Presentation at IETF meeting
2. `emul.`: Emulated Link; `real`: real Satellite link
3. SES Astra, Avanti PLC, and Eutelsat Tooway
4. Eutelsat KA-SAT PRO25Go
5. HTTP versions are not specified, but older gQUIC implementations usually use HTTP/2 over gQUIC while more recent gQUIC versions and IETF QUIC implementations usually use HTTP/3. No hints have been found that raw QUIC without HTTP was used.

Appendix E

List of Acronyms

ACK	Acknowledgement
API	Application Programming Interface
AQM	Active Queue Management
ARQ	Automatic Repeat-Request
AVX	Advanced Vector Extensions
BBR	Bottleneck Bandwidth and Round-trip propagation time
BDP	Bandwidth Delay Product
BER	Bit Error Ratio
CCA	Congestion Control Algorithm
CDF	Cumulative Distribution Function
cnes	Centre national d'études spatiales
CPU	Central Processing Unit
cwin	congestion window
DNS	Domain Name System
DPLPMTUD	Datagram Packetization Layer PMTUD
DupACK	Duplicate Acknowledgement
DVB	Digital Video Broadcasting
ECN	Explicit Congestion Notification
ELK-stack	Elasticsearch, Logstash, Kibana
FEC	Forward Error Correction
GEO	Geostationary Earth Orbit
GLONASS	Global Navigation Satellite System

GPS	Global Positioning System
GS	Ground Station
GUI	Graphical User Interface
HoLB	Head-of-Line-blocking
HTS	High Throughput Satellite
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
ID	Identifier
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
ISL	Inter-Satellite Link
ISP	Internet Service Provider
ITU-R	International Telecommunication Union, Radiocommunication Sector
iwin	initial congestion window
JSON	JavaScript Object Notation
KDE	kernel density estimation
LAN	Local Area Network
LEO	Low Earth Orbit
LFN	Long Fat Network
MASQUE	Multiplexed Application Substrate over QUIC Encryption
MEO	Medium Earth Orbit
MSS	Mobile Satellite Services
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NCC	Network Control Center
OpenBACH	Open Benchmark Automation tools for Communication and Hypervision
OpenSAND	Open Satellite Network Demonstrator
PC	Personal Computer

PEP	Performance Enhancing Proxy
PLR	Packet Loss Rate
PLT	Page Load Time
PMTUD	Path MTU Discovery
PTO	Probe Timeout
QEF	quasi error-free
QIR	QUIC-Interop-Runner
QoE	Quality of Experience
QoS	Quality of Service
RACK	Recent Acknowledgement
RAM	Random Access Memory
rcwin	Receive Window
RED	Random Early Detection
RFC	Request for Comments
RTT	Round-Trip Time
SACK	Selective Acknowledgement
SATCOM	satellite communications
SCC	Satellite Control Center
SCTP	Stream Control Transmission Protocol
SDG	Sustainable Development Goal
SGS	Source Ground Station
SIMD	Single Instruction Multiple Data
SLA	Service Level Agreement
SSE	Streaming SIMD Extensions
SSH	Secure Shell
STK	Satellite Tool Kit
SWOT	strengths, weaknesses, opportunities, and threads
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TFO	TCP Fast Open
TLP	Tail Loss Probe
TLS	Transport Layer Security

TT&C	Telemetry, Tracking and Control
TTFB	Time to first Byte
TTLB	Time to last Byte
TTR	Time to `responseStart`
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VLEO	Very Low Earth Orbit
VM	Virtual Machine
VPN	Virtual Private Network
WebRTC	Web Real-Time Communication

List of Figures

List of Tables

Bibliography

[1] A. C. Clark, "Extra-Terrestrial Relays — Can Rocket Stations Give World-wide Radio Coverage?," Wireless World, vol. 51, no. 10, pp. 305–308, Oct. 1945, https://lakdiva.org/clarke/1945ww/1945ww_oct_305-308.html. [Online]. Available: https://worldradiohistory.com/UK/Wireless-World/40s/Wireless-World-1945-10.pdf (visited on 01/06/2022).

[2] H. Dodel and S. Eberle, *Satellitenkommunikation*. Springer, 2007. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-3-540-29576-1.pdf.

[3] T. Jones, G. Fairhurst, N. Kuhn, J. Border, and S. Emile, "Enhancing Transport Protocols over Satellite Networks," Internet Engineering Task Force, Internet Draft draft-jones-tsvwg-transport-for-satellite-02, Oct. 2021, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-jones-tsvwg-transport-for-satellite/02/ (visited on 12/10/2021).

[4] K. Wolsing, J. Rüth, K. Wehrle, and O. Hohlfeld, "A Performance Perspective on Web Optimized Protocol Stacks: TCP+TLS+HTTP/2 vs. QUIC," in *Proceedings of the Applied Networking Research Workshop*, ser. ANRW '19, New York, NY, USA: Association for Computing Machinery, Jul. 2019, pp. 1–7, ISBN: 978-1-4503-6848-3. DOI: 10.1145/3340301.3341123. [Online]. Available: https://dl.acm.org/doi/10.1145/3340301.3341123.

[5] J. Deutschmann, K.-S. Hielscher, and R. German, "Satellite Internet Performance Measurements," in *2019 International Conference on Networked Systems (NetSys)*, Mar. 2019, pp. 1–4. DOI: 10.1109/NetSys.2019.8854494. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8854494.

[6] L. Han and R. Li, "Problems and Requirements of Satellite Constellation for Internet," Internet Engineering Task Force, Internet Draft draft-lhan-problems-requirements-satellite-net-01, Oct. 2021, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-lhan-problems-requirements-satellite-net/01/ (visited on 12/10/2021).

[7] W. Stallings, *Data and Computer Communications*, Ninth, ser. William Stallings Books on Computer and Data Communications Technology. Boston: Pearson, 2011, ISBN: 978-0-13-217217-2. [Online]. Available: https://books.google.de/books?id=ZDY-bwAACAAJ.

[8] J. Pavur, M. Strohmeier, V. Lenders, and I. Martinovic, "QPEP: An actionable approach to secure and performant broadband from geostationary orbit," Internet Society, 2021. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2021-074-paper.pdf.

[9] D. Kessler, N. Johnson, J.-C. Liou, and M. Matney, "The Kessler Syndrome: Implications to Future Space operations," *Advances in the Astronautical Sciences*, vol. 137, Jan. 2010. [Online]. Available: https://www.threecountrytrustedbroker.com/media/kesseler_syndrome.pdf (visited on 12/22/2021).

[10] J. Deutschmann, T. Heyn, C. Rohde, K.-S. Hielscher, and R. German, "Broadband Internet Access via Satellite: State-of-the-Art and Future Directions," in *Broadband Coverage in Germany; 15th ITG-Symposium*, Mar. 2021, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9399712/.

[11] J. Deutschmann, K.-S. Hielscher, and R. German, "Bewertung der Leistungsfähigkeit von Internet über Satellit," Lehrstuhl für Informatik 7 (Rechnernetze und Kommunikationssysteme) Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Tech. Rep., Jul. 2021, p. 36. [Online]. Available: https://www.cs7.tf.fau.de/forschung/quality-of-service/forschungsprojekte/sat-internet-performance/ (visited on 12/11/2021).

[12] C. Kulatunga, N. Kuhn, G. Fairhurst, and D. Ros, "Tackling Bufferbloat in capacity-limited networks," in *2015 European Conference on Networks and Communications (EuCNC)*, Jun. 2015, pp. 381–385. DOI: 10.1109/EuCNC.2015.7194103.

[13] J. Nagle, "On Packet Switches With Infinite Storage," Internet Engineering Task Force, Request for Comments RFC 970, Dec. 1985. DOI: 10.17487/RFC0970. [Online]. Available: https://datatracker.ietf.org/doc/rfc970 (visited on 12/05/2021).

[14] R. Schrage, G. Forget, R. Geib, and B. Constantine, "Framework for TCP Throughput Testing," Internet Engineering Task Force, Request for Comments RFC 6349, Aug. 2011. DOI: 10.17487/RFC6349. [Online]. Available: https://datatracker.ietf.org/doc/rfc6349 (visited on 12/04/2021).

[15] W. Stallings, *Computer Networking with Internet Protocols and Technology*, ser. "The" William Stallings Books on Computer and Data Communications Technology. Upper Saddle River, NJ: Pearson Prentice Hall, 2004, ISBN: 0-13-191155-4.

[16] V. Jacobson and R. Braden, "TCP extensions for long-delay paths," Internet Engineering Task Force, Request for Comments RFC 1072, Oct. 1988. DOI: 10.17487/RFC1072. [Online]. Available: https://datatracker.ietf.org/doc/rfc1072 (visited on 12/04/2021).

[17] E. N. Gilbert, "Capacity of a Burst-Noise Channel," *Bell System Technical Journal*, vol. 39, no. 5, pp. 1253–1265, 1960, ISSN: 1538-7305. DOI: 10.1002/j.1538-7305.1960.tb03959.x. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1960.tb03959.x (visited on 12/05/2021).

[18] N. Kuhn, F. Michel, L. Thomas, E. Dubois, and E. Lochin, "QUIC: Opportunities and threats in SATCOM," in *2020 10th Advanced Satellite Multimedia Systems Conference and the 16th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, Oct. 2020, pp. 1–7. DOI: 10.1109/ASMS/SPSC48805.2020.9268814. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9268814.

[19] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM Computer Communication Review*, vol. 18, no. 4, pp. 314–329, Aug. 1988, ISSN: 0146-4833. DOI: 10.1145/52325.52356. [Online]. Available: https://doi.org/10.1145/52325.52356 (visited on 12/05/2021).

[20] S. Floyd, J. Mahdavi, M. Mathis, and A. Romanow, "TCP Selective Acknowledgment Options," Internet Engineering Task Force, Request for Comments RFC 2018, Oct. 1996. DOI: 10.17487/RFC2018. [Online]. Available: https://datatracker.ietf.org/doc/rfc2018 (visited on 12/18/2021).

[21] J. Griner, J. Border, M. Kojo, Z. D. Shelby, and G. Montenegro, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," Internet Engineering Task Force, Request for Comments RFC 3135, Jun. 2001. DOI: 10.17487/RFC3135. [Online]. Available: https://datatracker.ietf.org/doc/rfc3135 (visited on 12/05/2021).

[22] C. Caini, R. Firrincieli, and D. Lacamera, "PEPsal: A Performance Enhancing Proxy designed for TCP satellite connections," in *2006 IEEE 63rd Vehicular Technology Conference*, vol. 6, May 2006, pp. 2607–2611. DOI: 10.1109/VETECS.2006.1683339.

[23] G. Papastergiou, G. Fairhurst, D. Ros, A. Brunstrom, K.-J. Grinnemo, P. Hurtig, N. Khademi, M. Tüxen, M. Welzl, D. Damjanovic, and S. Mangiante, "De-Ossifying the Internet Transport Layer: A Survey and Future Perspectives," *IEEE Communications Surveys Tutorials*, vol. 19, no. 1, pp. 619–639, 2017, ISSN: 1553-877X. DOI: 10.1109/COMST.2016.2626780.

[24] Y. Cheng, N. Cardwell, N. Dukkipati, and P. Jha, *The RACK-TLP Loss Detection Algorithm for TCP*, Feb. 2021. DOI: 10.17487/RFC8985. [Online]. Available: https://datatracker.ietf.org/doc/rfc8985.

[25] J. Iyengar and I. Swett, *QUIC Loss Detection and Congestion Control*, May 2021. DOI: 10.17487/RFC9002. [Online]. Available: https://datatracker.ietf.org/doc/rfc9002/.

[26] H. Jiang and C. Dovrolis, "Why is the internet traffic bursty in short time scales?" In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '05, New York, NY, USA: Association for Computing Machinery, Jun. 2005, pp. 241–252, ISBN: 978-1-59593-022-4. DOI: 10.1145/1064212.1064240. [Online]. Available: https://doi.org/10.1145/1064212.1064240 (visited on 12/11/2021).

[27] A. Gurtov, T. Henderson, S. Floyd, and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm," Internet Engineering Task Force, Request for Comments RFC 6582, Apr. 2012. DOI: 10.17487/RFC6582. [Online]. Available: https://datatracker.ietf.org/doc/rfc6582 (visited on 12/06/2021).

[28] W. R. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," Internet Engineering Task Force, Request for Comments RFC 2001, Jan. 1997. DOI: 10.17487/RFC2001. [Online]. Available: https://datatracker.ietf.org/doc/rfc2001 (visited on 12/06/2021).

[29] C. Caini and R. Firrincieli, "TCP Hybla: A TCP enhancement for heterogeneous networks," *International Journal of Satellite Communications and Networking*, vol. 22, no. 5, pp. 547–566, 2004, ISSN: 1542-0981. DOI: 10.1002/sat.799. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sat.799 (visited on 12/06/2021).

[30] S. Claypool, J. Chung, and M. Claypool, "Comparison of TCP Congestion Control Performance over a Satellite Network," in *Passive and Active Measurement*, O. Hohlfeld, A. Lutu, and D. Levin, Eds., Cham: Springer International Publishing, 2021, pp. 499–512, ISBN: 978-3-030-72582-2. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-72582-2_29.

[31] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks," Internet Engineering Task Force, Request for Comments RFC 8312, Feb. 2018. DOI: 10.17487/RFC8312. [Online]. Available: https://datatracker.ietf.org/doc/rfc8312 (visited on 12/06/2021).

[32] P. Balasubramanian, Y. Huang, and M. Olson, "HyStart++: Modified Slow Start for TCP," Internet Engineering Task Force, Internet Draft draft-ietf-tcpm-hystartplusplus-03, Jul. 2021, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-tcpm-hystartplusplus/03/ (visited on 12/06/2021).

[33] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, and V. Jacobson, "BBR Congestion Control," Internet Engineering Task Force, Internet Draft draft-cardwell-iccrg-bbr-congestion-control-01, Nov. 2021, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-cardwell-iccrg-bbr-congestion-control/01/ (visited on 12/06/2021).

[34] Y. Wang, K. Zhao, W. Li, J. Fraire, Z. Sun, and Y. Fang, "Performance Evaluation of QUIC with BBR in Satellite Internet," in *2018 6th IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, Dec. 2018, pp. 195–199. DOI: 10.1109/WiSEE.2018.8637347. [Online]. Available: https://ieeexplore.ieee.org/document/8637347.

[35] A. Mishra, X. Sun, A. Jain, S. Pande, R. Joshi, and B. Leong, "The Great Internet TCP Congestion Control Census," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, 45:1–45:24, Dec. 2019. DOI: 10.1145/3366693. [Online]. Available: https://doi.org/10.1145/3366693 (visited on 12/06/2021).

[36] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," Internet Engineering Task Force, Request for Comments RFC 9000, May 2021. DOI: 10.17487/RFC9000. [Online]. Available: https://datatracker.ietf.org/doc/rfc9000 (visited on 12/02/2021).

[37] R. R. Stewart, "Stream Control Transmission Protocol," Internet Engineering Task Force, Request for Comments RFC 4960, Sep. 2007. DOI: 10.17487/RFC4960. [Online]. Available: https://datatracker.ietf.org/doc/rfc4960 (visited on 12/06/2021).

[38] 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, Apr. 2021, ISSN: 1558-1896. DOI: 10.1109/MCOM.001.2000877. [Online]. Available: https://doi.org/10.1109/MCOM.001.2000877.

[39] M. Belshe, R. Peon, and M. Thomson, *Hypertext Transfer Protocol Version 2 (HTTP/2)*, ser. Request for Comments 7540. RFC Editor, May 2015. DOI: 10.17487/RFC7540. [Online]. Available: https://datatracker.ietf.org/doc/rfc7540/.

[40] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, New York, NY, USA: Association for Computing Machinery, Aug. 2017, pp. 183–196, ISBN: 978-1-4503-4653-5. DOI: 10.1145/3098822.3098842. [Online]. Available: https://doi.org/10.1145/3098822.3098842 (visited on 12/06/2021).

[41] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Internet Engineering Task Force, Request for Comments RFC 8446, Aug. 2018. DOI: 10.17487/RFC8446. [Online]. Available: https://datatracker.ietf.org/doc/rfc8446 (visited on 12/06/2021).

[42] M. Thomson and S. Turner, "Using TLS to Secure QUIC," Internet Engineering Task Force, Request for Comments RFC 9001, May 2021. DOI: 10.17487/RFC9001. [Online]. Available: https://datatracker.ietf.org/doc/rfc9001 (visited on 12/11/2021).

[43] M. Bishop, "Hypertext Transfer Protocol Version 3 (HTTP/3)," Internet Engineering Task Force, Internet Draft draft-ietf-quic-http-34, Feb. 2021, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-quic-http/34/ (visited on 12/06/2021).

[44] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain, "TCP Fast Open," Internet Engineering Task Force, Request for Comments RFC 7413, Dec. 2014. DOI: 10.17487/RFC7413. [Online]. Available: https://datatracker.ietf.org/doc/rfc7413 (visited on 12/18/2021).

[45] N. Kuhn, S. Emile, G. Fairhurst, T. Jones, and C. Huitema, "Transport parameters for 0-RTT connections," Internet Engineering Task Force, Internet Draft draft-kuhn-quic-0rtt-bdp-11, Oct. 2021, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-kuhn-quic-0rtt-bdp/11/ (visited on 12/10/2021).

[46] J. McCann, S. E. Deering, J. Mogul, and B. Hinden, "Path MTU Discovery for IP version 6," Internet Engineering Task Force, Request for Comments RFC 8201, Jul. 2017. DOI: 10.17487/RFC8201. [Online]. Available: https://datatracker.ietf.org/doc/rfc8201 (visited on 12/07/2021).

[47] S. E. Deering and J. Mogul, "Path MTU discovery," Internet Engineering Task Force, Request for Comments RFC 1191, Nov. 1990. DOI: 10.17487/RFC1191. [Online]. Available: https://datatracker.ietf.org/doc/rfc1191 (visited on 12/07/2021).

[48] G. Fairhurst, T. Jones, M. Tüxen, I. Ruengeler, and T. Völker, "Packetization Layer Path MTU Discovery for Datagram Transports," Internet Engineering Task Force, Request for Comments RFC 8899, Sep. 2020. DOI: 10.17487/RFC8899. [Online]. Available: https://datatracker.ietf.org/doc/rfc8899 (visited on 12/07/2021).

[49] G. Fairhurst, A. Custura, and T. Jones, "Changing the Default QUIC ACK Policy," Internet Engineering Task Force, Internet Draft draft-fairhurst-quic-ack-scaling-04, Mar. 2021, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-fairhurst-quic-ack-scaling/04/ (visited on 12/07/2021).

[50] J. Iyengar and I. Swett, "Sender Control of Acknowledgement Delays in QUIC," Internet Engineering Task Force, Internet Draft draft-iyengar-quic-delayed-ack-02, Nov. 2020, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-iyengar-quic-delayed-ack/02/ (visited on 12/07/2021).

[51] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, Aug. 1993, ISSN: 1558-2566. DOI: 10.1109/90.251892. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/251892.

[52] V. Jacobson, K. Nichols, K. Poduri, and C. Systems, "RED in a Different Light," Tech. Rep., 1999.

[53] S. Floyd, K. K. Ramakrishnan, and D. L. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," Internet Engineering Task Force, Request for Comments RFC 3168, Sep. 2001. DOI: 10.17487/RFC3168. [Online]. Available: https://datatracker.ietf.org/doc/rfc3168 (visited on 12/07/2021).

[54] I. Swett, M.-J. Montpetit, V. Roca, and F. Michel, "Coding for QUIC," Internet Engineering Task Force, Internet Draft draft-swett-nwcrg-coding-for-quic-04, Mar. 2020, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-swett-nwcrg-coding-for-quic/04/ (visited on 12/10/2021).

[55] V. Roca, F. Michel, I. Swett, and M.-J. Montpetit, "Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Schemes for QUIC," Internet Engineering Task Force, Internet Draft draft-roca-nwcrg-rlc-fec-scheme-for-quic-03, Mar. 2020, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-roca-nwcrg-rlc-fec-scheme-for-quic/03/ (visited on 12/07/2021).

[56] J. P. Rula, J. Newman, F. E. Bustamante, A. M. Kakhki, and D. Choffnes, "Mile High WiFi: A First Look At In-Flight Internet Connectivity," in *Proceedings of the 2018 World Wide Web Conference*, ser. WWW '18, Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, Apr. 2018, pp. 1449–1458, ISBN: 978-1-4503-5639-8. DOI: 10.1145/3178876.3186057. [Online]. Available: https://dl.acm.org/doi/10.1145/3178876.3186057.

[57] S. Yang, H. Li, and Q. Wu, "Performance Analysis of QUIC Protocol in Integrated Satellites and Terrestrial Networks," in *2018 14th International Wireless Communications Mobile Computing Conference (IWCMC)*, Jun. 2018, pp. 1425–1430. DOI: 10.1109/IWCMC.2018.8450388. [Online]. Available: https://ieeexplore.ieee.org/document/8450388.

[58] H. Zhang, T. Wang, Y. Tu, K. Zhao, and W. Li, "How Quick Is QUIC in Satellite Networks," in *Communications, Signal Processing, and Systems*, Q. Liang, J. Mu, M. Jia, W. Wang, X. Feng, and B. Zhang, Eds., ser. Communications, Signal Processing, and Systems, Singapore: Springer Singapore, Jun. 2018, pp. 387–394, ISBN: 978-981-10-6571-2. DOI: 10.1007/978-981-10-6571-2_47. [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-10-6571-2_47.

[59] L. Thomas, E. Dubois, N. Kuhn, and E. Lochin, "Google QUIC performance over a public SATCOM access," *International Journal of Satellite Communications and Networking*, vol. 37, no. 6, pp. 601–611, 2019, ISSN: 1542-0981. DOI: 10.1002/sat.1301. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sat.1301 (visited on 12/08/2021).

[60] G. Fairhurst, A. Custura, and T. Jones, *Measuring QUIC Dynamics over a High Delay Path*, Conference Presentation, Montreal, Jul. 2019. [Online]. Available: https://datatracker.ietf.org/meeting/105/materials/slides-105-maprg-measuring-quic-dynamics-over-a-high-delay-path-01 (visited on 07/12/2021).

[61] C. Mogildea, J. Deutschmann, K.-S. Hielscher, and R. German, "QUIC over Satellite: Introduction and Performance Measurements," in *KaConf*, Sep. 2019, p. 9. [Online]. Available: https://www.researchgate.net/

publication/351282748_QUIC_OVER_SATELLITE_INTRODUCTION_AND_PERFORMANCE_MEASUREMENTS.

[62] J. Border, B. Shah, C.-J. Su, and R. Torres, "Evaluating QUIC's Performance Against Performance Enhancing Proxy over Satellite Link," in *2020 IFIP Networking Conference (Networking)*, Jun. 2020, pp. 755–760. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9142718/ (visited on 07/19/2021).

[63] A. Custura, T. Jones, and G. Fairhurst, "Impact of Acknowledgements using IETF QUIC on Satellite Performance," in *2020 10th Advanced Satellite Multimedia Systems Conference and the 16th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, Oct. 2020, pp. 1–8. DOI: 10.1109/ASMS/SPSC48805.2020.9268894. [Online]. Available: https://ieeexplore.ieee.org/document/9268894.

[64] N. Kuhn, *Feedback from using QUIC's 0-RTT-BDP extension over SATCOM public access*, Conference Presentation, online, Jul. 2021. [Online]. Available: https://datatracker.ietf.org/meeting/111/materials/slides-111-maprg-feedback-from-using-quics-0-rtt-bdp-extension-over-satcom-public-access-00 (visited on 08/16/2021).

[65] S. Hemminger, "Network emulation with NetEm," in *Linux Conf Au*, Citeseer, vol. 5, 2005, p. 2005. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.67.1687.

[66] M. Carbone and L. Rizzo, "Dummynet revisited," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 2, pp. 12–20, Apr. 2010, ISSN: 0146-4833. DOI: 10.1145/1764873.1764876. [Online]. Available: https://doi.org/10.1145/1764873.1764876 (visited on 12/11/2021).

[67] G. F. Riley and T. R. Henderson, "The ns-3 Network Simulator," in *Modeling and Tools for Network Simulation*, K. Wehrle, M. Güneş, and J. Gross, Eds., Berlin, Heidelberg: Springer, 2010, pp. 15–34, ISBN: 978-3-642-12331-3. DOI: 10.1007/978-3-642-12331-3_2. [Online]. Available: https://doi.org/10.1007/978-3-642-12331-3_2 (visited on 12/12/2021).

[68] R. Marx, L. Niccolini, and M. Seemann, "Main logging schema for qlog," Internet Engineering Task Force, Internet Draft draft-ietf-quic-qlog-main-schema-01, Oct. 2021, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-quic-qlog-main-schema/01/ (visited on 12/12/2021).

[69] R. Marx, L. Niccolini, and M. Seemann, "QUIC event definitions for qlog," Internet Engineering Task Force, Internet Draft draft-ietf-quic-qlog-quic-events-00, Jun. 2021, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-quic-qlog-quic-events/00/ (visited on 12/12/2021).

[70] Z. Krämer, M. Kühlewind, M. Ihlar, and A. Mihály, "Cooperative Performance Enhancement Using QUIC Tunneling in 5G Cellular Networks," in *Proceedings of the Applied Networking Research Workshop*, ser. ANRW '21, New York, NY, USA: Association for Computing Machinery, 2021, pp. 49–51, ISBN: 978-1-4503-8618-0. DOI: 10.1145/3472305.3472320. [Online]. Available: https://dl.acm.org/doi/10.1145/3472305.3472320.

[71] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-Host Congestion Control for TCP," *IEEE Communications Surveys Tutorials*, vol. 12, no. 3, pp. 304–342, 2010, ISSN: 1553-877X. DOI: 10.1109/SURV.2010.042710.00114.