FACULTY OF SCIENCE AND TECHNOLOGY

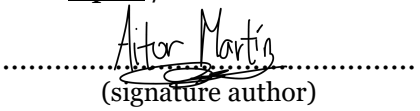
MASTER THESIS

Study programme / specialisation:
Electrical Engineering and Computer Science

The spring semester, 2022

Open / Confidential

Author: Aitor Martín Rodríguez



(signature author)

Course coordinator: Head of Department Tom Ryen

Supervisor(s): Associate Professor Naeem Khademi

Thesis title: Enhancing QUIC over Satellite Networks

Credits (ECTS): 30

Keywords: QUIC, SATCOM, satellite, transport layer, congestion control, BBR

Pages: 79

+ appendix: 25

Stavanger, 15.06.2022
date/year

Approved by the Dean 30 Sep 21
Faculty of Science and Technology

University of Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Enhancing QUIC over Satellite Networks

Master's Thesis in Computer Science
by

Aitor Martín

Supervisor

Assoc. Prof. Naeem Khademi

June 14, 2022

Abstract

The use of Satellite Communication (SATCOM) networks for broadband connectivity has recently seen an increase in popularity due to, among other factors, the rise of the latest generations of cellular networks (5G/6G) and the deployment of high-throughput satellites. In parallel, major advances have been witnessed in the context of the transport layer: first, the standardization and early deployment of QUIC, a new-generation and general-purpose transport protocol; and second, modern congestion control proposals such as the Bottleneck Bandwidth and Round-trip propagation time (BBR) algorithm. Even though satellite links introduce several challenges for transport layer mechanisms, mainly due to their long propagation delay, satellite Internet providers have relied on TCP connection-splitting solutions implemented by Performance-Enhancing Proxies (PEPs) to greatly overcome many of these challenges. However, due to QUIC's fully encrypted nature, these performance-boosting solutions become nearly impossible for QUIC traffic, leaving it in great disadvantage when competing against TCP-PEP. In this context, IETF QUIC WG contributors are currently investigating this matter and suggesting new solutions that can help improve QUIC's performance over SATCOM. This thesis aims to study some of these proposals and evaluate them through experimentation using a real network testbed and an emulated satellite link.

Acknowledgements

Thanks to my supervisor Associate Professor Naeem Khademi, for his guidance and dedication. Thanks to my family, for their unconditional support and love. Thanks to the people of Stavanger and all around the world, for sharing wonderful moments with me.

Contents

Abbreviations

ACK	Acknowledgement
AQM	Active Queue Management
BDP	Bandwidth-Delay Product
BBR	Bottleneck-Bandwidth and Round-Trip Time
CC	Congestion Control
cwnd	Congestion Window
E2E	End-to-end
ECN	Explicit Congestion Notification
GEO	Geosynchronous Equatorial Orbit
ISP	Internet Service Provider
LEO	Low Earth Orbit
NAT	Network Address Translation
NTP	Network Time Protocol
PEP	Performance Enhancing Proxy
RTT	Round-Trip Time
SATCOM	Satellite Communications
SSH	Secure Shell
TCP	Tranmission Control Protocol
TLS	Transport Layer Security
QoE	Quality of Experience
QUIC	Quick UDP Internet Connections
UDP	User Datagram Protocol

Chapter 1

Introduction

This document describes the work carried out during the spring semester of 2022 at the Faculty of Science and Technology at the University of Stavanger (UiS), for the Master Thesis in Computer Science titled 'Enhancing QUIC over Satellite Networks'.

1.1 Motivation

The transport layer is instrumental for establishing logical end-to-end communications over the Internet. Transport protocols provide network application developers with abstraction regarding what networks their traffic has to go across, in some cases providing reliability and security. TCP (Transmission Control Protocol) [1] and UDP (User Datagram Protocol) [2] have been fundamental pillars for the development of Internet communications for decades, and they are still used for most of the Internet traffic [3].

TCP, as a connection oriented and reliable transport protocol, is key for applications where data transfers needs to be completed reliably, and where latency requirements are not strict (e.g., web browsing, file exchange or e-mail). On the other hand, UDP minimizes overhead and provides fast connectionless communications, which makes it fundamental for delay-sensitive applications (e.g., VoIP or videoconferencing). Other protocols such as SCTP (Scream Control Transmission Protocol) [4] and DCCP (Datagram Congestion Control Protocol) [5] have attempted to combine the message-oriented approach of UDP with the reliability of TCP, to provide a more appropriate transport for modern applications. Even though these protocols have proven to be suitable for many scenarios, they have not been able to compete with the well-established TCP and UDP. In the last decade, a new candidate has appeared, aiming to revolutionize the transport layer.

QUIC (Quick UDP Internet Connections), initially designed by Google [6], aims to substitute TCP, offering a secure, low-latency, reliable, multiplexed and general-purpose transport over UDP. After years of development, QUIC was finally standardized as a series of RFCs [7–10]. QUIC integrates TLS, offering full end-to-end encryption and aiming for secure Internet communications. This feature greatly benefits Internet users, however, it also introduces some new challenges for network operators and ISPs (Internet Service Providers), due to the decrease in observability.

In parallel, we have witnessed the development of High Throughput Satellites (HTS) [11] and the 5th and 6th generations of cellular networks, which contemplate the use of Satellite Communication (SATCOM) networks for access or backhauling purposes, and even potentially using a hybrid terrestrial-satellite scheme [12]. GEO (Geosynchronous Equatorial Orbit) satellites have been widely used for broadband services for decades due to the large coverage areas they can offer, and we are now witnessing the early development and experimental phases of new generation LEO (Low Earth Orbit) constellations, which also seem promising drivers for the Internet of the future [13]. This means that future Internet communications will benefit from the possibilities these solutions introduce; nevertheless, they will also have to be able to overcome the challenges introduced by satellite infrastructure.

In this context, this thesis aims to analyze the challenges of using encrypted transport protocols - specifically QUIC - over SATCOM networks, study their impact on performance and evaluate the suitability of different state-of-the-art solutions through network experimentation.

1.2 Use Cases

There are several use-cases where modern cellular networks can integrate satellite infrastructure [12]. Figure 1.1 shows two significant use cases for broadband access using GEO satellites.

The first use case contemplates the use of a GEO satellite an access link, to give coverage to users living in remote areas directly. With a single satellite terminal, users can reach the gateway set up by their Internet Service Provider (ISP) through the satellite link, which provides them with Internet connectivity.

The second use case is transparent to the user, and it suggests using a GEO satellite to establish a backhaul link between the Radio Access Network (RAN) and the Core Network (CN) of the cellular network architecture. This can benefit remote regions where there is a lack of terrestrial infrastructure and a long distance to reach the CN. This

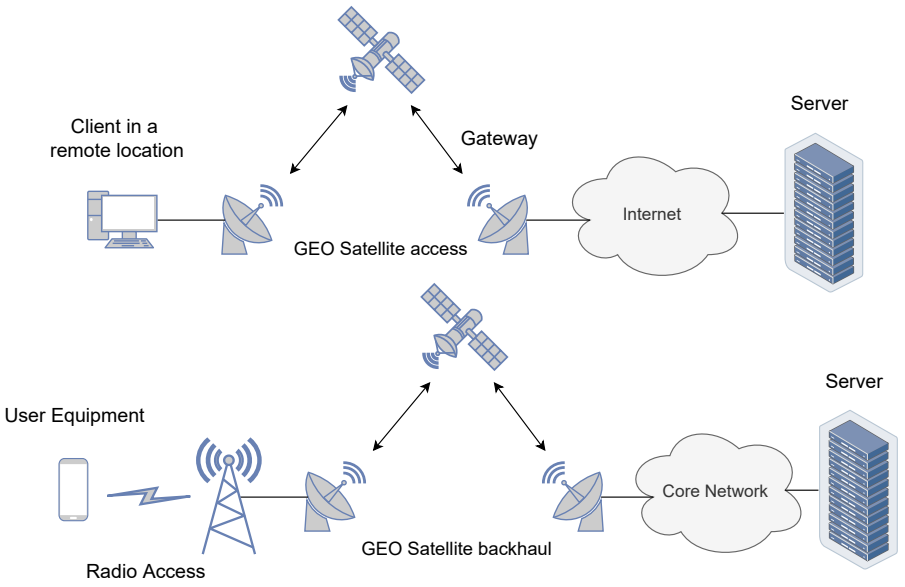


Figure 1.1: Satellite Internet use cases

use case also contemplates the possibility of using a dual-setup with a terrestrial and a satellite scenario: this would allow to perform load balancing and applying Quality of Service (QoS) through traffic prioritization over one link or the other depending on their, for example, latency or bandwidth requirements.

1.3 Problem Definition and Research Questions

Satellite links feature long propagation paths that contribute to the end-to-end latency. The distance from the Earth surface to GEO satellites can range from 36000 to 42000 km approximately, depending on the relative position between ground station and satellite, which translates into propagation delays that vary between 120 and 140 ms. In an end-to-end scheme where a GEO satellite link is in the path, one RTT (Round Trip Time) includes 4 trips between the Earth's surface and the satellite. This means that, if additional delays due to routing and processing in middleboxes are included, the RTT in the connection can easily go over 600 ms.

This introduces very significant challenges for the transport protocols [14]. First, protocol feedback suffers great delays. This means that mechanisms like loss recovery, congestion control or flow control take longer to get feedback (e.g., it takes a long time to detect packet loss or to receive acknowledgement packets). Second, the greater RTT increases the BDP (Bandwidth-Delay Product), which leads to a higher number of packets "in flight" (i.e., unacknowledged packets) and therefore requires larger buffers in the endpoints and

satellite transponders. Some other challenges also derive from the bandwidth asymmetry present in satellite links and propagation errors.

These challenges introduced by the satellite link have usually been overcome with the use of Performance-Enhancing Proxies (PEPs) [15], which often enhance TCP connections with the use of connection-splitting. Splitting connections into several segments allows local loss recovery and optimized congestion control in each segment. PEPs generally accelerate the TCP handshake by spoofing the SYN-ACK messages, without waiting for the response from the other side of the satellite link, to get a faster connection establishment. They can also implement satellite-optimized congestion algorithms, such as TCP Hybla [16].

However, QUIC's end-to-end encryption [9] completely disables connection-splitting, which leaves PEPs out of the picture. This leaves QUIC in great disadvantage, which even with the fast handshake is greatly outperformed by TCP-PEP solutions [17]. With the web quickly moving towards HTTP/3 and QUIC (to this date, 8% of the websites on the public Internet already use QUIC [18]), it becomes clear that new solutions need to be found to boost the performance of QUIC over satellite links.

Taking all of this into consideration, we formulate two main research questions:

- **RQ1**: *Can the performance of QUIC over SATCOM links be improved using transport protocol mechanisms?*
- **RQ2**: *If the answer to RQ1 is positive, are these mechanisms safe and feasible to implement?*

These research questions will be further detailed in Chapter 3.

1.4 Objectives

After defining the main research questions, a series of major objectives can be defined for this work:

1. Study the state-of-the-art surrounding QUIC transport over GEO SATCOM and identify solutions that could potentially boost performance.
2. Set up a network testbed that allows running automated QUIC experiments over an emulated satellite link, ensuring reproducibility and repeatability.

3. Define a series of metrics and scenarios to evaluate different aspects of network performance, in order to evaluate the suitability of the selected solutions.
4. Obtain performance results for these scenarios and observe their implications.
5. Reflect on the results and start a discussion about the impact of the selected solutions and their feasibility.

1.5 Outline

The thesis is structured as follows:

- Chapter 2 describes some theory regarding the QUIC protocol, congestion control algorithms and satellite communications (SATCOM), adding some insight into the challenges that satellite links introduce to the transport layer.
- Chapter 3 discusses the existing approaches to improve the performance of QUIC over satellite, and outlines the contributions of this work.
- Chapter 4 presents a brief description of the research methodology, adding insight into the chosen experimental approach, workflow and design principles.
- Chapter 5 offers an in-depth description of the design and implementation of the experimental testbed.
- Chapter 6 describes the experimental procedure, metrics and scenarios, and it presents the results of the experiments.
- Chapter 7 discusses the results, analyzing the possible reasons behind them, the impact of different aspects of the transport layer on performance and the feasibility of different mechanisms.
- Chapter 8 summarizes the outcome of this work, giving an answer to the research questions and hinting possible future research paths.

Chapter 2

Background

Before describing the solution proposal for the formulated problem statement, it is important to provide some background. This chapter aims to provide context for the work done in this thesis, describing the different pillars that it stands on and discussing the related works in the field.

The chapter starts by characterizing GEO SATCOM links, enumerating the challenges that these links introduce in the context of the Internet transport layer and briefly describing the solutions proposed over the years to mitigate these challenges for TCP traffic. These solutions include protocol optimizations and proxied appraoches, and it is essential to understand them in order to come up with new solutions for QUIC. This chapter also summarizes the published performance evaluation studies on QUIC over SATCOM and some early research studying different proposals for boosting performance.

Next, the QUIC protocol is introduced, describing its background and design principles, relevant features for SATCOM and the current development and deployment status of the different QUIC implementations. Having a basic understanding of the different mechanisms implemented in QUIC is important to design clever SATCOM optimizations.

Finally, major breakthroughs in congestion control are described, briefly explaining the development towards modern CC algorithms (i.e. BBR). Since many of BBR's properties make it a general-purpose CC, it is interesting to contemplate the feasibility of integrating BBR in the QUIC protocol and evaluating its performance over SATCOM.

2.1 Transport Layer over SATCOM

GEO SATCOM links introduce have a series of properties that introduce challenges for transport layer mechanisms. These challenges have lead to several TCP solutions, that

either optimize TCP over satellite by adding extensions to the protocol or propose the use of proxies that implement performance-enhancing mechanisms, usually transparent to the endpoints.

2.1.1 Challenges

There are mainly three relevant properties of the satellite link that are fundamental for this study: (1) the long propagation delay, (2) the asymmetry of the connection and (3) the propagation errors [19].

Firstly, the **long propagation delay** implies a long protocol feedback loop, which implies that the slow-start and congestion-avoidance mechanisms will need a long time to ramp up the *cwnd* and reach the available bandwidth (Challenge C#1), thus being inefficient in terms of link utilization. Loss recovery mechanisms and cumulative ACKs also suffer great delays, which can lead to unnecessary retransmissions and difficulty in the RTT measurements, which are essential for the correct setup of retransmission timers (Challenge C#2). In addition, a high propagation delay also increases the BDP, which increases the size of the buffers needed in both the endpoints (receiving windows) and in the satellite transponder (buffer size) to reach full link utilization (Challenge C#3).

Secondly, satellite links can be **bandwidth-asymmetric**, i.e. upstream bandwidth (from the user to the Internet) can be several times smaller than downstream bandwidth (from the Internet to the user). The bandwidth-asymmetric plans offered by satellite service providers are oriented towards web browsing traffic, which usually implies large amounts of data being downloaded from the Internet and not much traffic in the upstream directions (e.g., ACKs, cookies and other small uploads). However, this implies that, if the upstream traffic increases and the buffers become filled with ACKs, this will slow down the data transfers in the downstream link [20] (Challenge C#4).

Thirdly, satellite links are prone to **propagation errors**. Even though these links often maintain a free line-of-sight between satellite and ground stations, they are prone to bit errors caused by atmospheric attenuation and rain fading, which are present at the gigahertz frequency bands in which satellites communication satelites operate, - i.e. between 4 and 40 GHz [21] (Challenge C#5).

2.1.2 TCP Solutions

Since the challenges of using TCP over SATCOM links were identified, several mechanisms have been proposed to mitigate them over the years, and many of them have become fundamental TCP extensions to improve TCP's performance over heterogeneous networks.

Table 2.1 summarizes these solutions, classified as (1) protocol optimizations [22, 23], (2) proxy-enabled optimizations (with PEPs [15]) and (3) link layer solutions. The following subsections describe them in more detail.

Type	**Mechanism**	**Challenge**
Protocol optimizations	Better slow-start and congestion-avoidance	C#1
	Timestamps extension [24]	C#2
	Selective Acknowledgements [25]	C#2
	TCP Window Scale Option [24]	C#3
	Model-based CC	C#5
PEP optimizations	ACK Spoofing	C#1,2
	Handshake Spoofing	C#1
	Satellite-optimized CC [16]	C#1
	Local Loss Recovery	C#2
	ACK Aggregation	C#4
Link layer solutions	Forward Error Correction (FEC)	C#5

Table 2.1: Mechanisms to boost TCP performance over SATCOM

2.1.2.1 TCP Protocol Optimizations

Protocol optimizations are based on modifications or extensions of the TCP protocol that can be implemented directly on the sender or receiver. The following list briefly describes some of the main approaches implemented in the TCP protocol:

1. **The TCP Window Scale option** [24]. The original TCP specification [1] used a 16-bit field to set the size of the sending and receiving windows, which leads to a maximum of 64 kilobytes. When the bandwidths and latencies in Internet communications started to grow, this value stopped being enough to handle the large amounts of in-flight data. The TCP Window Scale option defines an exponent that can set the window value up to 2^{30} bytes, i.e. 1 gigabyte. This extension was key for support of high BDP paths.

2. **The Timestamps extension** [24]. A common approach to measure RTT in TCP endpoints is to measure the time between packet sent and corresponding ACK received. However, this method has some faults: (1) re-transmitted packets share the same sequence numbers, and (2) the sequence number space is limited to 2^{32}. These two items can lead to wrong RTT measurements. The Timestamps extension adds a header field that can be used to carry a clock timestamp, and enables the use of an *echo* where both endpoints share their timestamp, allowing to measure RTT with precision.

3. **Selective Acknowledgements (SACKs)** [25]. The first TCP specification used cumulative acknowledgements - i.e., ACKs indicate the last packet that has been received successfully. The main issue of this approach is that if one packet is lost, all the following packets will also need to be re-transmitted. Using SACKs allows to inform the sender of the specific packets that have been lost, so that the sender only needs to re-transmit the lost packets.

4. **Better slow-start and congestion-avoidance**. The increase of the *cwnd* in both the slow-start and congestion-avoidance phases becomes slower when the path RTT increases. Proposals like TCP Hybla [16] modify these phases so that they are independent of the RTT, compensating for the long delay and letting the *cwnd* ramp-up faster in satellite scenarios.

These solutions allow to improve TCP connections over long BDP links without needing to use any proxies. However, if these optimizations are too satellite-specific and too aggressive for other general-purpose scenarios (e.g., a satellite-optimized congestion control), there might be little interest in deploying them on the server-side. This is why many satellite service providers rely on proxied solutions, which are usually transparent to the endpoints.

2.1.2.2 Performance-Enhancing Proxies

PEPs are proxy-based solutions that allow to accelerate TCP connections by intercepting them somewhere along the network path [15], and they are widely used in both satellite and cellular use cases. PEPs usually rely on connection-splitting, which allows them to split a network path into several segments and manage separate connections in each of them, being able to optimize these segments individually without degrading performance in the others.

PEP implementations can be either integrated - i.e. based on one single proxy (e.g. PEPsal [26]) - or distributed i.e. based on two proxies, allowing to create a tunnel (e.g. QPEP [27]), as presented in Figure 2.1.

Depending on the selected PEP architecture, some performance-boosting mechanisms or others might be feasible. Although integrated solutions are cheaper and simpler, there are some mechanisms that they do not support - e.g. handshake acceleration. Table 2.2 summarizes these mechanisms, indicating which are possible in each PEP architecture.

The following list briefly describes some popular PEP mechanisms:

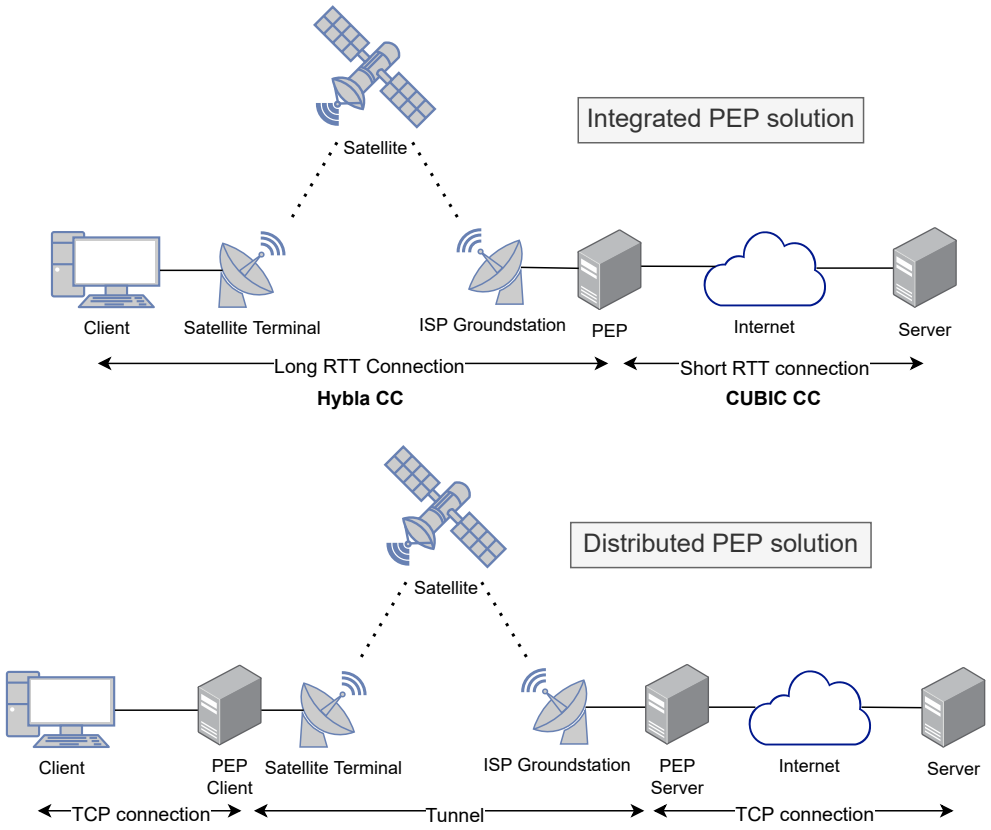


Figure 2.1: PEP implementations: integrated (top) and distributed (bottom)

Mechanism	Integrated	Distributed
ACK spoofing	✓	✓
ACK aggregation	✓	✓
CC optimization	✓	✓
Local loss recovery	✗	✓
Handshake acceleration	✗	✓

Table 2.2: Summary of PEP mechanisms

1. **ACK Spoofing**. A proxy can send ACKs to the sender pretending to be the receiver, to accelerate the increase of the *cwnd*.

2. **ACK Aggregation**. A proxy can reduce the rate of ACKs by aggregating them, to reduce congestion on the return link.

3. **CC optimization**. Via connection-splitting, a proxy on the server-side of the satellite link can use a satellite-optimized CC (e.g., Hybla) that is only applied on the satellite segment. This allows servers to use a general-purpose CC, without needing to know that there is a satellite within the network path.

4. **Local Loss Recovery**. A proxy on the receiver-side can store packets in a buffer to allow fast retransmission in case of packets being lost.

5. **Handshake Spoofing**. A proxy can pretend to be the server and respond to the initial connection handshake on the client-side, avoiding the satellite link.

2.1.3 QUIC over SATCOM

When QUIC is introduced into the picture of SATCOM broadband services, a major issue stands out. As a result of the fully-encrypted nature of the QUIC header, PEP optimizations become nearly impossible, because proxies cannot see the contents of the QUIC header that are fundamental for connection-splitting (e.g., packet numbers, stream IDs, etc.). Therefore, unless some cooperation between the network and the endpoints is introduced or some QUIC header fields are exposed to the middleboxes, PEPs as we know them are mostly out of the picture.

After identifying this major issue, several studies have shown that the inability to use PEPs leaves QUIC in great disadvantage when compared to TCP-PEP over SATCOM links. Table 2.3 summarizes the work carried out in these studies, which evaluate different aspects of transport layer performance through experimentation over simulated, emulated and real satellite links.

Ref.	Object of Study	Implementation	Experiments
[28]	Page Load Times with Google QUIC	chromium	R
[29]	Impact of Packet Loss on QUIC-SATCOM	chromium, quicly, ngtcp2	R, E
[30]	HTTP Browsing with Google QUIC	chromium	E
[31]	QUIC ACK Policies over SATCOM	chromium, quicly, picoquic	R, E
[32]	Effect of QUIC mechanisms for web browsing over SATCOM	chromium	E
[33]	QUIC-BBR over SATCOM	chromium	E
[34]	Web over GEO/LEO systems	chromium	S
[35]	FEC-QUIC over wireless links	rQUIC (quic-go)	S
[36]	Evaluation of BDP Extension	picoquic	R, E

Table 2.3: Summary of the research that evaluates the performance of QUIC over SATCOM links. Experiments column: R (real satellite), E (emulation) and S (simulation)

Authors in [28–30, 32] have thoroughly evaluated QUIC over SATCOM on real and emulated scenarios, and they have all claimed that TCP-PEP solutions greatly outperform QUIC. The work carried out in [28] points out that low performance is magnified for large downloads - when downloads are short, QUIC's fast handshake compensates for the high latency; but when downloads become larger, QUIC's CC's slow convergence makes it difficult to use the available link bandwidth efficiently. As shown in [30], the presence of packet loss does not change the picture, and TCP-PEP keeps clearly beating QUIC. Results in [29] also point out some performance differences between QUIC implementations and satellite operators, likely due to high heterogeneity in the

QUIC endpoints, the satellite's low layer mechanisms and the satellite operators' policies. Researchers in [32] show the benefits of QUIC's 0-RTT resumption, stream multiplexing and connection control when compared against no-PEP TCP. Other works such as [34] extend performance studies to integrated GEO-LEO satellite networks.

After QUIC's drawbacks over SATCOM where clearly shown, some studies began research tasks on some performance-boosting solutions: (1) the use of custom ACK policies [31] to reduce traffic in the return link under bandwidth asymmetry, which showed promising results; (2) the benefits of using BBR congestion control [33] on lossy high RTT links; (3) applying custom forward error correction to reduce the impact of propagation errors [35] and (4) remembering path parameters to accelerate CC convergence [36].

The implications and potential benefits of using a PEP that splits QUIC connections have also been evaluated in [37], showing that QUIC-PEP can be faster than TCP-PEP. However, such a solution breaks end-to-end encryption and gives the proxy complete read access to the payload, failing to maintain QUIC's security principle. This study will not consider such a solution, and it will focus on solutions that do not involve breaking the end-to-end confidentiality.

2.2 QUIC protocol

QUIC was first proposed by Google in 2012 [6], defined as a general-purpose transport protocol running over UDP that can inherit the best of TCP and UDP, while also integrating end-to-end encryption with Transport Layer Security (TLS) [38], providing reliable stream multiplexing and also reducing latency. QUIC is also designed to run in the user-space, avoiding the problem of misbehaving-middleboxes and transport layer ossification [39] and also enabling quick iteration of the protocol. At the same time, this protocol aims to solve a series of shortcomings of TCP [40], such as (1) high latency in connection establishment; (2) only being able to send a single request/response per segment in HTTP/1.1 due to the lack of multiplexing and (3) the problem of head-of-line (HOL) blocking.

During the last decade, maintenance, development and specification of the QUIC protocol has been carried out by the IETF QUIC Working Group, and it has lead to a series of RFCs [7–10] that define the first standardized version of QUIC. This version of QUIC is usually referred to as IETF QUIC, to distinguish it from Google QUIC, which is developed independently and implemented in Chromium [41].

2.2.1 New Features of QUIC

The QUIC protocol introduces a long list of features and mechanisms that are explained in depth in [8]. The following list enumerates some of the most fundamental mechanisms in the specification in the context of this study:

1. **Faster connection establishment**. The traditional TCP connection establishment uses a 3-way handshake that takes 1.5 RTTs to finish. If TLS is also used, unless any mechanisms such as the TCP Fast Open [42] of the TLS False Start [43] are implemented, it takes 3 RTTs to establish the secure connection. However, QUIC allows a faster handshake with integrated TLS: in the first connection to a certain server, QUIC uses a 1-RTT handshake; for further connections, QUIC can benefit from 0-RTT connection resumption, given that client and server have cached information about each other, significantly reducing the latency penalty of starting a new connection. This can be highly benefitial for GEO satellite links, which can introduce up to 600 miliseconds of latency for each RTT.

   A comparison between QUIC and TCP+TLS handshake is shown in Figure 2.2.

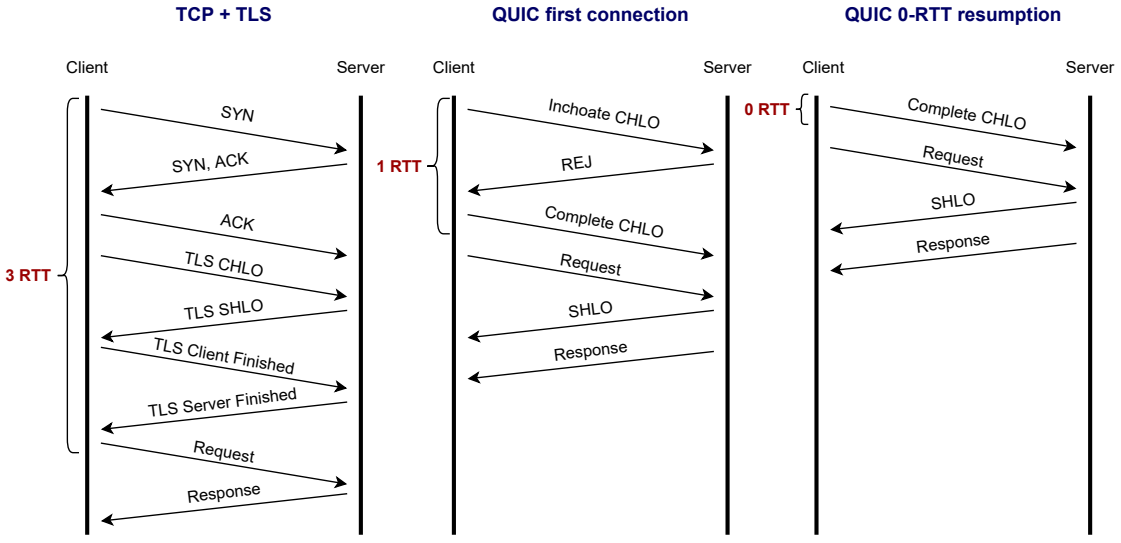


Figure 2.2: QUIC handshakes in comparison to TCP+TLS

2. **Connection Identifiers.** Instead of using the common 5-tuples from TCP (2 IP addresses, 2 ports and the higher layer protocol), QUIC uses a 64 bit connection identifier, randomly chosen by both endpoints. This enables connection mobility across IP addresses and UDP ports.

3. **Stream Multiplexing.** Within a single connection multiple streams can be sent, all identified by the same connection ID. Streams have a stream identifier (stream ID), and they can be established by both the client (using even numbers) and

the server (using odd numbers). QUIC provides flow control on both stream and connection level.

4. **Monotonically increasing sequence numbers.** TCP uses sequence numbers to identify unique segments; however, if the same segment is retransmitted, it uses the same sequence number, not allowing the receiver totell the difference between the original and the retransmissions. QUIC, on the contrary, uses monotonically increasing sequence numbers - i.e., the sequence number increases even for retransmitted segments. This helps to estimate the path RTT more accurately, which becomes more important when the base RTT is really high, e.g. in the satellite link.

5. **Packets and Frames.** QUIC endpoints communicate with each other using **packets**, which are transported over UDP datagrams. The specification defines two types of headers for QUIC packets: the **long** header, which is used for packets sent before the 1-RTT keys have been exchanged, and the **short** header, which is used to minimize the overhead in data exchange after the handshake is completed. QUIC packets can carry multiple **frames** in the payload field. All packets except for Version Negotiation packets (the ones used to negotiate which version of QUIC is gonna be used) have some level of cryptographic protection [8]. The provided confidentiality and integrity mechanisms are great additions for satellite networks, where usually little attention is paid to security concerns, due to the usually wrongly assumed trade-off between performance and security [44].

2.2.2 HTTP/3

The definition of QUIC also came along with the standardization work of HTTP/3 [45], which is essentially the translation of the HTTP semantics [46] to the QUIC transport.

Since web browsing is one of the major use cases for satellite broadband connectivity, it seems relevant to look at the evolution of the HTTP protocol during the last few decades. HTTP/1.1 [47] lacked a multiplexing layer, so in order to avoid HOL blocking, it was necessary to open one new TCP connection for each parallel request to be sent, which has a negative impact on congestion control and network efficiency. With HTTP/2 [48], a multiplexing layer was introduced, which allowed to multiplex various requests on a single packet. However, the parallel nature of HTTP/2 multiplexing is not visible to the TCP loss recovery mechanism, and therefore, in the event of a lost or reordered packet, all active transactions can experience a stall regardless of whether each of them were affected by this event.

The arrival of QUIC comes with many benefits for the HTTP protocol. QUIC incorporates stream per-stream flow control and reliability in the transport layer, as well as congestion control across the entire connection. In the HTTP context, means being able to launch multiplexed HTTP requests in a single connection, allowing congestion control to operate equally over all of them.

This leads to the new HTTP/3-QUIC protocol stack, presented in Figure 2.3 in comparison to the previous HTTP/2-TCP stack. HTTP/3 relies on QUIC for data confidentiality, integrity and peer authentication (previously provided by TLS over TCP), for multi-streaming (previously implemented on HTTP/2) and for reliability (previously provided by TCP).

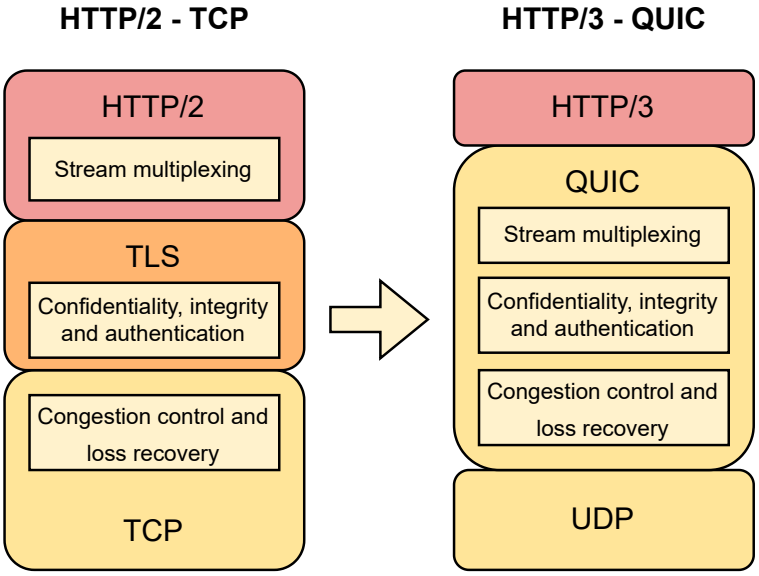


Figure 2.3: The HTTP/3 protocol stack

The main operation scheme between HTTP/3 client and server can be summarized as the following:

- An HTTP/3 client opens a QUIC connection, which provides protocol negotiation, stream-based multiplexing and flow control.
- The HTTP/3 client can multiple various HTTP requests on a single QUIC connection. Each request/response pair goes on a different QUIC stream, avoiding HOL blocking.
- The HTTP/3 server can perform a **server push**, to send HTTP objects without needing to wait for the client's request.
- HTTP/3 endpoints can benefit of the use of QPACK (Header Compression for HTTP/3) [49], replacing the previous HPACK.

2.2.3 Relevant QUIC Extensions

After the release of the QUICv1 specification [8], contributors have been working on a series of extensions for QUIC. The following list briefly describes some QUIC extensions that are relevant for the future implementations of QUIC over SATCOM:

1. **Unreliable Datagram Extension** [50]. QUIC uses *STREAM* frames for reliable data transmission. However, in applications with real-time requirements and high packet loss tolerance (e.g., videoconferencing), it is interesting to be able to transmit data unreliably. To address this issue, the specification in [50] introduces a *DATAGRAM* frame. QUIC datagrams are not retransmitted upon loss, and neither flow-controlled. The ability to carry multiple frames in a single packet allows reliable and unreliable transmissions to coexist in the same QUIC connection. This extension can be fundamental to alleviate the impact of latency for applications with real-time requirements, e.g. VoIP or videoconferencing over satellite.

2. **Multipath QUIC Extension** [51]. This extension aims to provide QUIC with the ability to manage multiple simultaneous network paths on a single connection. This implies per-path congestion control, RTT measurement and Maximum Transmission Unit (MTU) discovery. This extension is still work in progress, and there are currently some debates on how it should be implemented - e.g. the packet number space debate [52], which discusses the pros and cons of using a global or a per-path packet number space. The multipath extension will be key for the deployment of hybrid terrestrial-satellite solutions in 5G networks, such as the one proposed in [53].

3. **BDP Frame Extension** [54, 55]. This extension allows endpoints to exchange path parameters when resuming a connection, to accelerate the slow process of discovering them. It is further described in Chapter 3.

4. **ACK Frequency Extension** [56]. This extension allows QUIC clients to negotiate the rate at which they send acknowledgement frames. It is further described in Chapter 3.

2.2.4 QUIC Implementations

There is a wide range of QUIC implementations available for experimentation and use. The most popular implementations are presented in Table 2.4, which summarizes some relevant information about them and the companies or individuals that carry out their development.

As shown in the table, many different actors have shown interest in QUIC and developed their own QUIC implementations for their services: Content Delivery Network (CDN) providers - e.g., Akamai and Cloudflare -; web service solution providers such as LiteSpeed and big technological companies such as Apple, Microsoft, Google and Facebook. IETF QUIC WG enthusiasts have also independently implemented QUIC for experimenting with new QUIC features, such as *aioquic*, *ngtcp2* and *picoquic*. We can see that most implementatins already support HTTP/3.

Even though most of them are based on the same specification, high heterogeneity in the implementations has been reported [57]. In this context, the QUIC Interop Runner project [58] was developed by several IETF QUIC WG contributors in order to benchmark the performance and interoperability between QUIC implementations. This project has recently been extended to GEO satellite links by authors in [59].

Implementation	HTTP/3	Comments
aioquic [60]	✓	Implementation for Asyncio RTC
Apple QUIC	✓	Proprietary implementation by Apple for iOS
Akamai QUIC	✓	Powers Akamai CDN services
lsquic [61]	✓	Used in LiteSpeed web server solutions
msquic [62]	✗	Developed by Microsoft
mvfst [63]	✗	Deployed in Instagram and Facebook [64]
Neqo [65]	✓	Developed by Mozilla
ngtcp2 [66]	✓	Mainly developed by Tatsuhiro Tsujikawa
picoquic [67]	✓	Mainly developed by Christian Huitema
quiche (chromium) [68]	✓	Default transport in Google Chrome
quiche (cloudflare) [69]	✓	Powers Cloudflare's edge network
quicly [70]	✗	Developed for H2O Web Servers
quic-go [71]	✗	Used in several projects (see [71])

Table 2.4: Summary of some QUIC implementations

2.3 Congestion Control

Congestion Control (CC) is a fundamental mechanism for the well-being of the Internet. The connectionless design of the Internet with the IP protocol provides great robustness and flexibility; however, it also requires endpoints to apply mechanisms that control the traffic that they introduce into the network, to avoid overloading the Internet [72]. When congestion controlled traffic goes through a satellite link, performance drops due to the long protocol feedback loop, which slows down convergence and can potentially introduce fairness issues, as already described in Section 2.1.1.

Traditional standardized TCP CC algorithms rely on packet loss as an indicator of network congestion, e.g. NewReno [73] and CUBIC [74]. Other CC algorithms use

different input for detecting congestion; for instance, the model-based TCP Vegas [75] uses packet delay as a signal of congestion. Recent research into CC strategies has led to the definition of the Bottleneck-Bandwidth and Round-trip propagation time (BBR) algorithm [76], a model-based CC algorithm that models congestion by measuring the bottleneck-bandwidth and RTT in the network path, described in Section 2.3.2.

Other alternatives rely on the network itself notifying that there is actual congestion. These include Explicit Congestion Notification (ECN) [77], which allows routers to notify congestion by marking packets using a flag in the TCP header, and Active Queue Management (AQM) [78], which allows to actively drop packets to notify congestion to the sender.

QUIC specifies by default a congestion controller based on NewReno; however, many QUIC implementations have adapted TCP CC for QUIC support. This thesis will compare QUIC performance using the loss-based CUBIC and the model-based BBR, which are briefly described in the following subsections.

2.3.1 Loss-based CC: CUBIC

Loss-based CC algorithms use packet loss as an indicator that the network is congested. CUBIC [74], designed to utilize bandwidth more efficiently in paths with high speed and latency, is nowadays the default CC algorithm for TCP in Linux kernels since version 2.6.19.

The CUBIC algorithm starts with the standard TCP slow-start mechanism [79], which exponentially increases the *cwnd* starting from a low value. Then, it follows a cubic function, which allows to (1) fastly ramp up to the *cwnd* value set before the last congestion event and (2) slowly increase the *cwnd* to probe for more bandwidth after the inflection point of the cubic function is surpassed.

An example of a sender using CUBIC over an ideal GEO satellite link with a 600 ms RTT and 20 Mbps downstream bottleneck bandwidth (i.e., BDP = 1.5 Mbytes) is shown in Figure 2.4. After the slow start phase, the normal CUBIC behavior is visible, but due to the long RTT the function takes several seconds to increase the *cwnd* up to its maximum value (this value depends on the link BDP and the bottleneck buffer size - in this case, with a buffer size of 1BDP, the *cwnd* can reach a value of 2 BDP, i.e. 3 Mbytes). This involves the available bandwidth being underutilized for long periods of time.

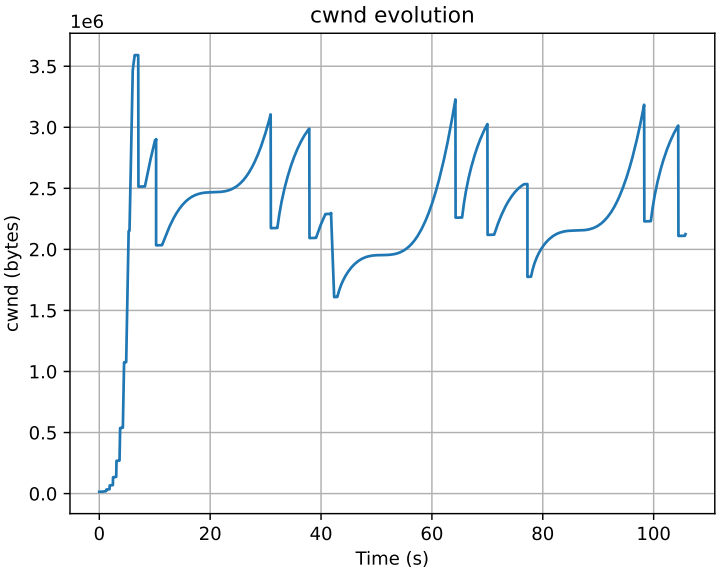


Figure 2.4: An example showing CUBIC behavior over a high latency GEO link (RTT = 600 ms)

2.3.2 Model-based CC: BBR

A well-known issue about loss-based CC is that they assume that all packet loss is consequence of network congestion. In the modern communications era, where wireless network are highly present, packet loss due to propagation errors is very common, which can lead to inefficient usage of the available bandwidth. In this context, BBR [80] aims to improve link utilization by trying to find an optimal *cwnd* value that maximizes throughput while trying to keep the connection RTT as low as possible and avoiding bottleneck buffer overload.

The BBR algorithm can be summarized into four main phases:

1. The **STARTUP** phase increases the *cwnd* exponentially to fill the bottleneck queue quickly and to measure the available bandwidth on the network path.
2. The **DRAIN** phase drains the bottleneck queue to remove the congestion introduced in the STARTUP phase.
3. The **PROBE_BW** phase, in which the algorithm cycles through different pacing rate values, aiming to continuously maximize the use of available bandwidth, but also draining the queue regularly in order to maintain fairness towards other parallel flows.
4. The **PROBE-RTT** phase allows the sender to re-measure the minimum RTT value within regular intervals.

An example of a sender using BBR over an ideal GEO satellite link with a 600 ms RTT and 20 Mbps downstream bottleneck bandwidth is shown in Figure 2.5. After the STARTUP and DRAIN, the *cwnd* stays on the optimal point, utilizing the available bandwidth better and draining the queue on regular intervals to remove congestion.

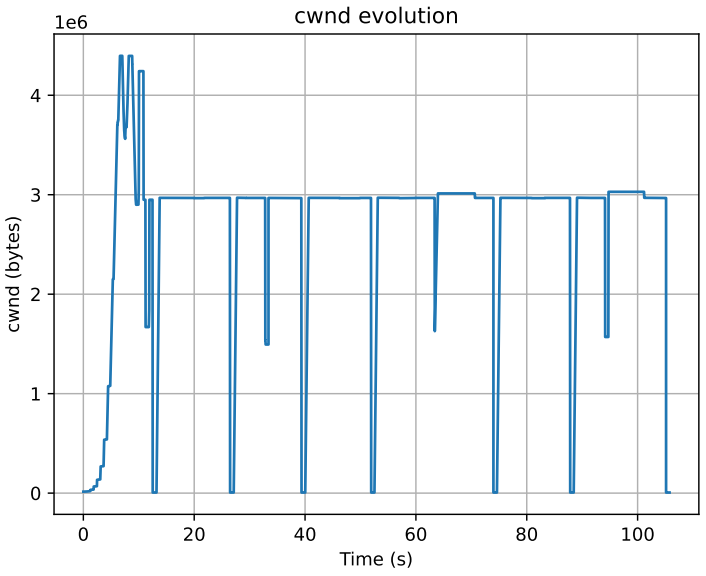


Figure 2.5: An example showing BBR behavior over a high latency GEO link (RTT = 600 ms)

This algorithm has been experimentally shown to outperform CUBIC in many scenarios, especially under high packet loss conditions [81]. Nevertheless, several studies have pointed out several unfairness issues: (1) unfairness between parallel BBR flows, (2) high aggressiveness towards loss-based CC and (3) RTT unfairness [82–84]. These studies attribute these issues to the highly aggressive nature of the early STARTUP phase and the following bandwidth probing phases.

After these problems were identified, an updated version of the algorithm started to be specified in 2019, later defined as BBRv2 [76]. This update adds additional complexity to the BBR bandwidth probing algorithm, with the goal of maintaining BBRv1's high bandwidth utilization while making it less aggressive towards other flows sharing the link. This is partly achieved by making the BBR state machine use packet loss and ECN information as an input, which reduces aggressiveness and improves coexistence with loss-based CC. BBRv2 also tweaks the probing algorithms to make RTT fluctuations smoother.

BBRv2 has already been thoroughly investigated with TCP over terrestrial networks in several studies [85–88]. Authors in [85] have shown clear improvements in the fairness towards CUBIC with BBRv2; however, other studies still found intra-protocol convergence

issues between BBRv2 flows [86, 87] and some misbehavior of the algorithm when network conditions and bandwidth dynamics change [88], leading to some proposals to improve the algorithm even further.

As stated before, BBRv2 has been exhaustively evaluated over TCP traffic going through low-medium RTT paths. Nevertheless, there is a lack of studies that evaluate BBRv2 over QUIC and that investigate BBRv2's performance over scenarios with high BDP values, such as those in satellite links. This thesis will study these aspects in-depth.

Chapter 3

Proposed Solutions

This chapter describes a series of approaches that have been proposed by IETF QUIC WG contributors and other researchers in order to improve QUIC over satellite.

3.1 Introduction

The proposed solutions can be classified as (1) end-to-end solutions - which can be client-side, server-side or both - that aim to improve performance over satellite by improving QUIC; and (2) proxied solutions that aim to offer performance enhancing mechanisms through cooperation between endpoints and network proxies.

The following sections introduce each of these categories, pointing out the advantages and drawbacks of each, and describe some of the major solutions that are currently under investigation.

3.2 End-to-end solutions

End-to-end solutions might allow improving the performance of QUIC over satellite without the need of any proxies. Many contributors advocate for these solutions, since they only affect the endpoints, they avoid the ossification caused by middleboxes (e.g., PEPs) and they maintain the end-to-end principle. Table 3.1 links together the transport layer challenges pointed out in Chapter 2 with the different end-to-end solutions that are described in this Chapter.

Challenge	Solution(s)
Slow connection startup	BDP Frame Extension [54]
ACK Congestion on return path	ACK Frequency Extension [56]
Packet loss recovery and CC convergence	Congestion-based CC: BBRv1/v2 [76] Forward Error Coding (FEC) for QUIC [89]

Table 3.1: Challenges and proposals for QUIC over SATCOM

3.2.1 Better Congestion Control

By default, the QUIC specification defines a CC mechanism based on NewReno [10]. The mechanism begins with a a Slow Start phase, where the *cwnd* increases exponentially with each ACK received; a Recovery phase to reduce the *cwnd* when packet loss is detected (either by receiving 3 duplicate ACKs or detecting the expiration of a retransmission timeout) or an increase in the ECN-CE counter is detected; and a Congestion Avoidance phase, in which the *cwnd* is increased linearly using an Additive Increase Multiplicative Decrease (AIMD) approach.

Even if QUIC specifies NewReno by default, different QUIC implementations have implemented other TCP CC algorithms over QUIC:

- CUBIC is supported by most QUIC implementations
- BBRv1 is supported by *lsquic*, *mvfst*, *ngtcp2*, *picoquic* and *xquic*
- BBRv2 is only supported by *chrome* and *ngtcp2*

As justified in Chapter 2, using model-based CC algorithms such as BBR might be helpful in satellite links, since it can help to improve bandwidth utilization and minimize the increase in path RTT due to queue congestion. Especially in high packet loss conditions, BBR might prove an advantage in comparison with NewReno or CUBIC.

Therefore, in this study we compare both BBRv1 and BBRv2 with CUBIC in terms of bandwidth utilization, congestion load and fairness.

3.2.2 Accelerate path parameter discovery

When clients resume a session to download a large object, CC algorithms require time to ramp up the data rate - especially if the path RTT is high (e.g., satellite). To solve this, the proposal in [54] suggests that endpoints can save path parameters - i.e. the base RTT and the bottleneck-bandwidth - from previous sessions and use them in the following connections started between the same pair of IP addresses. This would accelerate the

slow process of discovering path parameters - which is especially slow in the presence of a satellite link in the path.

There have been several proposals on how to implement this, following different security rationale [55]:

(a) The server learns path parameters in the first 1-RTT connection. In following connections, it waits for 1 RTT to check if the current RTT is similar enough to the previous RTT. If this safety check is passed, the current path parameter values are replaced by the ones stored from the previous connection. In this solution, path parameters are **stored in the server**, and **never sent to the client**.

(b) The server sends an encrypted packet to the server, containing path parameters in a NEW_TOKEN frame. The client can then send it back to the server in the next sessions. Path parameters are **shared through the link**, but they are **never revealed to the client**.

(c) The server builds a BDP Frame [54] including path parameters, client IP address and a lifetime value, and sends it to the client. If the client accepts the BDP extension negotiation, it can send this frame back to the server in following connections to enable parameter acceleration. Path parameters are **shared through the link**, and they are **revealed to the client**.

None of these proposals allows the client to modify path parameters. Proposals (b) and (c) need to implement a series of safety checks before activating this feature, to make sure that network conditions or network path have not changed, and to avoid any possible malicious clients [55].

In this study, we evaluate this solution using the BDP Frame proposal (proposal (C)), which is currently implemented in *picoquic* [67]. An illustration of this approach is shown in Figure 3.1.

3.2.3 Reduce ACK load in the return link

The asymmetry in satellite links can lead to ACK congestion in the return path, as discussed in Chapter 2, limiting the throughput in the forward direction [20].

For example, consider the following simplified example scenario: we are using a bottleneck link with a bandwidth of 10 Mbps on the downstream and 50 kbps on the upstream. The downstream link mainly carries data packets of 1000 bytes, leading to a maximum of 10,000 packets per second. The upstream mainly carries ACK packets of e.g. 40 bytes,

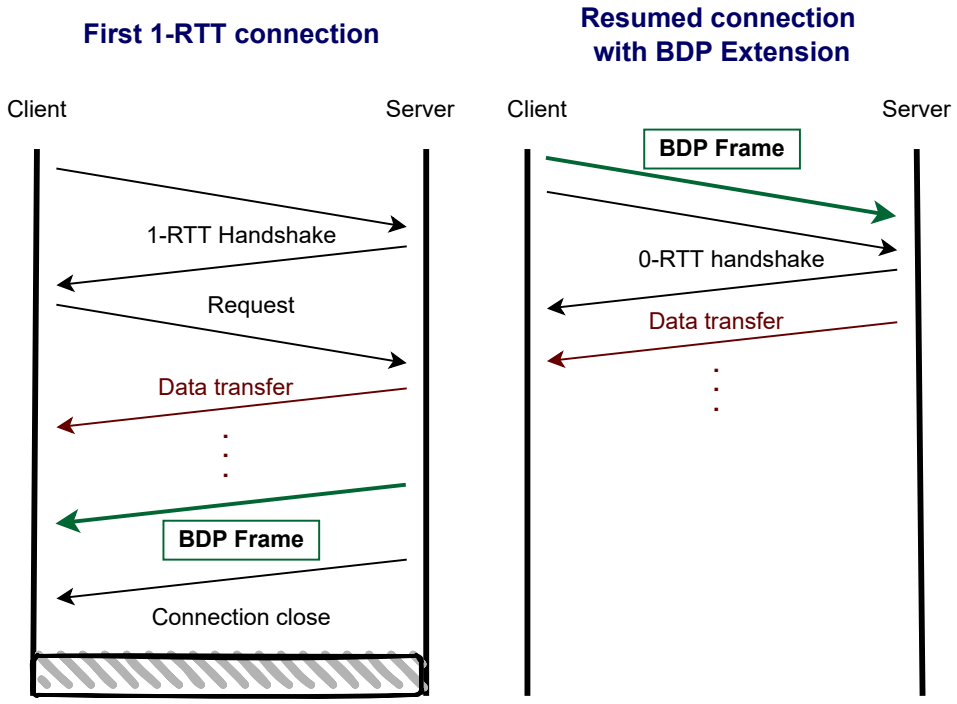


Figure 3.1: Illustration of BDP extension, using the BDP Frame approach

leading to a maximum of 1,250 ACKs per second. Therefore, we can define a "threshold" of $k = 10,000/1,250 = 8$, meaning that 1 ACK can be sent per each 8 data packets. If the receiver sends ACKs more frequently than this, the return path will get congested, limiting forward traffic.

The QUIC specification [10] specifies a default ACK ratio of 1:2 - i.e. 1 ACK per each 2 packets. In this context, the ACK Frequency extension [56] allows QUIC endpoints to negotiate this ratio, in order to reduce the ACK overload. The extension introduces a new transport parameter (**min_ack_delay**), which advertises support of the extension, and defines a minimum value for this ratio. It also defines two frames:

- The **ACK_FREQUENCY** frame, which can be sent by the receiver to the sender to specify the rate at which it wants to send ACKs.
- an **IMMEDIATE_ACK** frame, which can be sent by the sender to the receiver to ask for an ACK, in order to reduce the feedback delay in specific situations, or to measure RTT with a PING frame.

These mechanisms allow QUIC endpoints to define and negotiate custom ACK policies, which can be optimized according to parameters such as the link RTT and the bandwidth asymmetry.

3.2.4 Forward Error Correction

There is also some interest in introducing Forward Error Correction (FEC) on the transport layer [89]. FEC is based on the premise of adding some overhead to the sent QUIC packets that allows receivers to fix errors and therefore reduce packet loss.

On the one hand, this solution adds robustness and reduces packet loss events, minimizing the impact that long protocol feedback in the satellite links has on loss recovery mechanisms. On the other hand, the additional overhead needed for FEC coding also significantly reduces the amount of data that can be carried in each QUIC packet. This tradeoff needs to be thoroughly addressed and evaluated through different FEC solutions. Authors in [35] provide a FEC solution over QUIC to improve performance over different wireless links.

Nevertheless, due to the lack of current research effort into this solution, there is a lack of QUIC-FEC implementations, and thus it is not gonna be studied in this work.

3.3 Application Proxies

Another alternative is to use application proxies that can help optimize the QUIC connections over the satellite link. In this context, some IETF QUIC WG contributors contemplate the use of Multiplexed Application Substrate over QUIC Encryption (MASQUE) [90] as a potential solution, a proposal that extends the HTTP CONNECT method for compatibility with UDP and QUIC, allowing to proxy QUIC connections.

3.3.1 MASQUE

In 2020, the IETF MASQUE WG was created [91], aiming to produce a series of HTTP specifications that allow running multiplexed applications using QUIC streams and datagrams inside an HTTPS connection.

MASQUE is based on the HTTP CONNECT method defined in [47]. This method allows using a proxy to establish secure end-to-end tunneled connections between a client and a server, guaranteeing that the proxy cannot read the data being exchanged. This solution allows having proxies that only establish connections towards secure servers. However, this method has two major shortcomings: (1) it only supports tunneling TCP connections, which means that it cannot tunnel UDP datagrams; and (2) the lack of multiplexing, since each CONNECT command opens a new TCP connection.

MASQUE aims to define UDP CONNECT [92], which extends HTTP CONNECT to be able to tunnel UDP datagrams. Additionally, it aims to be able to tunnel QUIC Datagrams inside an HTTP connection [93]. Therefore, MASQUE could allow using QUIC to tunnel another QUIC connection, as shown in Figure 3.2.

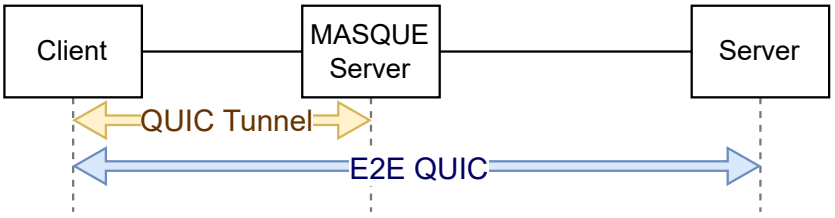


Figure 3.2: Illustration of the use of MASQUE to tunnel QUIC connections

These solutions could enable use cases where MASQUE can provide cooperation between endpoints and network proxies to enhance performance, as proposed by authors in [94] for 5G networks, allowing mechanisms such as local loss recovery or "promise signaling", which involves the proxy being able to "promise" having received frames to the client but delaying their delivery for the sake of performance. Preliminary results in [95] show promising performance improvements using the local loss recovery mechanism when the packet loss in the client-proxy path becomes significant.

This hints towards potential use cases of MASQUE for SATCOM links. It has been demonstrated that the lack of a satellite-optimized CC for QUIC over SATCOM [28] is one of the main factors that decreases its performance. In the absence of proxies, CC will be suboptimal to some of the path segments: e.g., a satellite-optimized CC might work well on the satellite segment, but it might not be good for the rest of the link. However, if the QUIC connection is proxied, it might be possible to use a QUIC tunnel over the satellite link with an appropriately optimized CC.

Even though this solution looks promising, it introduces the challenge of having two nested congestion control mechanisms, which might not be optimal. In addition, it is still in early development and there is a lack of MASQUE implementations that are easy to integrate in the current QUIC implementation ecosystem. This is why this thesis will not study this approach, but it opens up very interesting research paths for the future.

3.4 Selected Solutions

Taking all of the above into consideration, this study will focus on the following proposals: using better congestion control (Section 3.2.1), accelerating path parameter discovery with the use of the BDP Extension (Section 3.2.2) and using custom ACK policies on the return link (Section 3.2.3).

Therefore, the research question **RQ1** can be subdivided into three more specific research questions:

- **RQ1**: *Can the performance of QUIC over SATCOM links be improved using transport protocol mechanisms?*
  - **RQ1.1**: *Can better congestion control algorithms improve bandwidth utilization, download speed and fairness?*
  - **RQ1.2**: *Can the early exchange of BDP information between endpoints using the BDP Extension improve congestion control convergence?*
  - **RQ1.3**: *Can custom ACK policies on the client-side improve performance on asymmetric satellite links?*

Chapter 4

Research Methodology

This chapter aims to describe the research methodology followed during this work. This involves discussing the different possible approaches for satellite experimentation, specifying which one was chosen for this work, the reasons why it was chosen and its implications. The following section presents the general methodology followed for designing experiments, running them and extracting conclusions. Some comments are also included on the design principles of the network testbed and the importance of reproducibility, repeatability and test automation.

4.1 Satellite Experimentation

Any experimental setup can provide realistic results up to some extent. Just like many other real-life systems, performing experiments with satellite links with high precision can be challenging, since there are many factors to consider - e.g., the electronics in the satellite transponder in outer-space conditions, the signal propagation through the atmosphere, disturbances such as atmospheric attenuation or hydrometeors, etc.

In order to set up satellite experiments, there are three possible approaches [96]: (1) having access to a real satellite link (2) emulating the satellite link using a testbed (e.g., OpenSAND [97]) or (3) through simulation (e.g., SNS3 [98], the satellite network extension to ns-3). All approaches can have their advantages and drawbacks. For instance, approach (1) provides very realistic results, but limits experimentation to the services and bandwidth plans provided by the satellite operator. An (2) emulated or (3) simulated setup provides higher flexibility for designing experiments; however, it also simplifies many physical phenomena and low layer mechanisms, which can reduce the reliability of the experimental results if not considered carefully.

In this study, due to the lack of access to real satellites at UiS, we rely on **satellite link emulation**. Emulation through a testbed is usually conceived as more realistic than simulation, since it involves physical equipment and real network links, aiming to imitate the conditions of the real scenario as close as possible. Emulating a satellite link with high precision could involve emulating physical layer (PHY) and medium access control (MAC) mechanisms - e.g., a very common PHY technique in satellite link is Adaptive Coding and Modulation (ACM), which allows to adapt coding and modulation schemes to the link conditions, using more robust schemes in case of high link degradation and faster schemes when the conditions are better.

Since we are studying satellite links from the perspective of the transport layer, it is possible to detach from these low layer mechanisms, since they play a less relevant role on the higher picture of the end-to-end scheme. However, these mechanisms should not be disregarded and their absence needs to be considered when both designing the experiments and interpreting the results.

4.2 Experimental Methodology

The research methodology followed in this work is summarized in Figure 4.1. The first step is to design and implement a network testbed that allows to set up a wide range of experimental scenarios through an emulated satellite link. This testbed should be flexible in topology and scalable, to be able to scale it up with more endpoints if needed. It should also allow test automation - i.e. to set up parameter sweeps and automate experiments, in order to limit the need of human operation as much as possible and save time resources.

It is essential that the testbed allows repeatability of the experiments - i.e., that experiments can be repeated under the same conditions every time, to avoid any external bias - and reproducibility, so anyone can repeat the same set of experiments and obtain the same results. This is instrumental in any scientific practice, to make the results credible and so that other institutions and individuals can verify the claims made.

After the testbed is set up, it needs to be validated. For this, a series of basic experiments is designed to analyze the impact of the satellite link delay, as well as the correct operation of the QUIC endpoints in aspects such as congestion control and flow control. This is an essential step to guarantee the value of the results obtained with the testbed. The testbed implementation is described in detail in Chapter 5.

After the testbed is validated and ready, a series of experimental scenarios need to be built. These scenarios need to be able to prove the feasibility of the selected solutions,

and they should aim to be as close as possible to real satellite scenarios, in terms of link characteristics and traffic conditions. After designing an initial set of scenarios and obtaining preliminary results, these scenarios should be fine-tuned according to the observations made. This involves an iterative process of trial and error and constant improvement of the scenarios, aiming to converge towards more realistic results.

Once appropriate results have been obtained, they need to be thoroughly studied and interpreted. This involves analyzing if the results match our expectations, and making hypotheses of possible reasons behind unexpected behaviors, which is why these should be accompanied by proper data visualization. This should lead to formulating a series of conclusions and answers to the research questions.

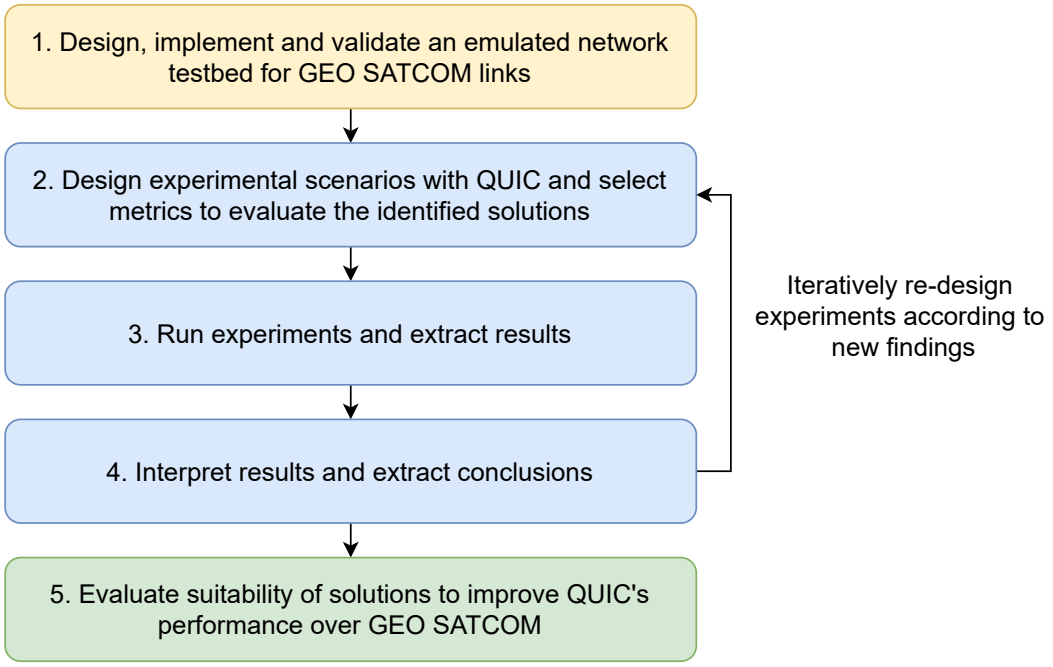


Figure 4.1: Summary of research methodology

4.3 Data Collection and Analysis

The experiments in this work generate large amounts of data, which needs to be properly extracted, processed and analyzed. Experiment logs should be compressed to save storage space and to accelerate remote downloads. The processing and visualization process should be completely automated, to ensure that the same treatment is applied to all the experimental data, and also saving time resources in the long term.

When going through the processing of the results, is essential to be aware of the following: even if the experiments are executed correctly, wrong conclusions can be extracted if the post-processing is not done properly. This is why close attention needs to be put into

these matters, which involve: (1) how to extract metrics from the logs generated at the endpoints and (2) how to present and visualize these metrics.

Chapter 5

Testbed Implementation

This chapter describes the UiS TEACUP testbed used to perform network experiments using QUIC. The testbed is based on the 'TCP Experiment Automation Controlled Using Python' (TEACUP) platform from CAIA [99], which allows to orchestrate and automate TCP network experiments. The UiS TEACUP testbed was first built in [100], and it has now been extended for QUIC support.

5.1 Testbed Overview

The complete testbed topology is shown in Figure 5.1. The testbed is composed by a controller and 5 experiment nodes, which include 4 hosts and a router.

- The **controller** takes care of executing commands remotely in the experiment nodes and extracting all the logs and traces they generate.
- **hostXX-Y** are the endpoints, which can act as client or servers:
  - **XX** indicates the third octet of the network address - i.e., 10 for network A and 11 for network B. Unless specified otherwise, hosts in network A act as clients, and hosts in network B act as servers.
  - **Y** enumerates the different endpoints in the same network (i.e., 1 and 2)
- The **router** is responsible of routing traffic between endpoints in networks A and B, and performing link emulation

The controller is connected to the experiment nodes through the Control Network (CN), with IP address 10.0.0.1/24. This network is used for remotely controlling the experiment

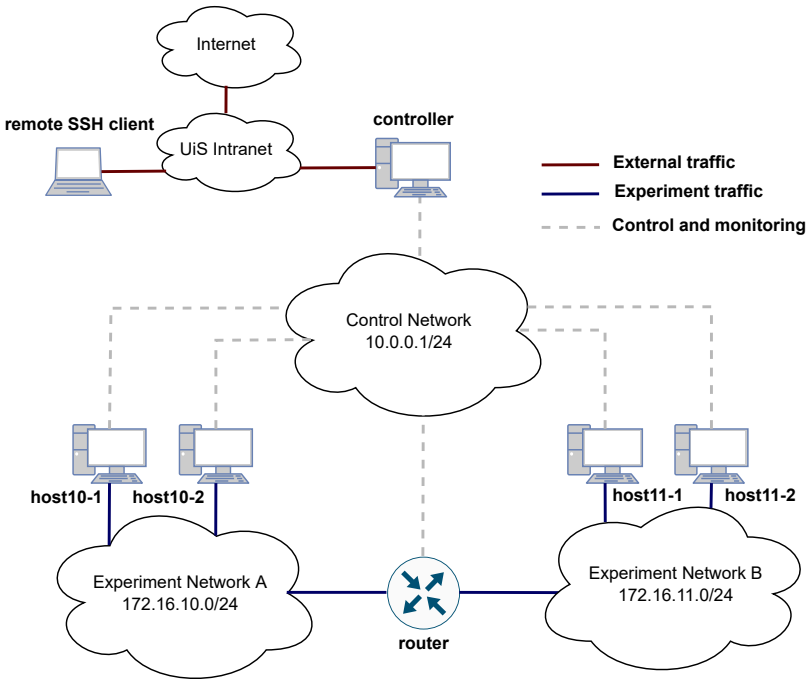


Figure 5.1: Architecture of the network testbed

nodes (endpoints and router), as well as downloading data from them (logs, dumps, traces, etc.). This is managed using the TEACUP platform.

The traffic generated in the experiments goes through the Experiment Networks (EN), with IP addresses 172.16.10.0/24 (network A) and 172.16.11.0/24 (network B), as shown in Figure 5.2. The experiment networks are isolated from the control network - this is essential to make sure that SSH sessions or other external traffic does not affect the experiments.

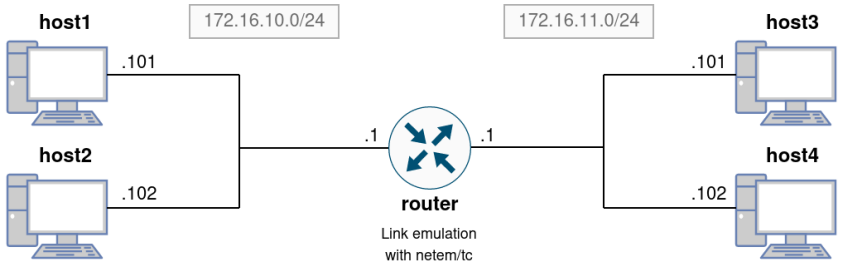


Figure 5.2: Topology of the Experiment Networks

All the experiment nodes run on the OpenSUSE Leap 15.1 Linux distribution, and their hardware is identical, as shown in Table 5.1. The controller runs on FreeBSD [101].

CPU	4 x Intel(R) Core i5-3470 @ 3.20GHz
RAM	16 GB Micron DDR3 1600 MHz
HDD	500 GB (100 GB for OS and Swap, 400 GB for /home)

Table 5.1: Testbed node specifications

5.1.1 Controller Setup

The FreeBSD controller host needs to be able to provide a series of basic services to experiment nodes: **time synchronization**, **gateway functionalities** for Internet access, and **remote access** to the nodes.

Time synchronization between the experiment nodes is essential for rigorous experimentation, and it is achieved using the Network Time Protocol (NTP). The controller acts as a NTP server, using the **ntpd** service in FreeBSD. The experiment nodes are configured to synchronize with the controller on startup. The TEACUP platform restarts the NTP service on every host before each experimental run, to fix any clock deviation.

Add this line to /etc/rc.conf:

```
ntpd_enable='YES'
```

In order to minimize the number of public IP addresses used, the testbed uses private addressing for the experiment nodes, and uses a single public IP address for the external interface in the controller. Therefore, the controller needs to perform **Network Address Translation (NAT) services**, to provide the experiment nodes with Internet connectivity. This also involves enabling packet forwarding in the controller.

To enable NAT, a custom FreeBSD kernel needs to be built with a specific set of options, as described in [1]. This also requires to enable IP forwarding and set the firewall policy as 'open':

Add these lines to /etc/rc.conf:

```
gateway_enable="YES"
firewall_enable="YES"
```

[1] https://docs.freebsd.org/en/books/handbook/kernelconfig/

```
firewall_type="OPEN"
natd_enable="YES"
natd_interface="re0"
```

It is also essential that the experiment nodes can be accessed from the controller using the **Secure Shell (SSH) protocol**. The testbed requires that the SSH connections do not require password, since they need to be accessed frequently by the TEACUP scripts experiment orchestration. To enable this, the **sshd** service in the experiment nodes needs to be configured to allow Public Key authentication, modifying /etc/ssh/sshd_config. Then, the controller needs to generate a SSH key pair, and share this key with the experiment nodes.

Run the following commands:

```
>> ssh-keygen
>> ssh-copy-id -i /root/.ssh/id_rsa.pub root@<server>
```

and repeat the second command for each SSH server.

This basic setup allows the use of TEACUP [99] in the controller.

5.1.2 Endpoint Setup

Network configuration in the endpoints needs to be configured according to the network topology. This implies static IP addresses and routes. Endpoints configure only two routes: one route to reach the opposite experiment network through the router (i.e., endpoints in network A configure route to reach endpoints in network B, and viceversa) and a default route through the gateway - i.e., the controller.

The endpoints need to be able to run all the commands that the TEACUP platform uses. This requires to install a series of packages:**net-tools-deprecated**, **ethtool**, **tcpdump**, **psmisc** and **ntpq**.

The endpoints need to install all the traffic generators and loggers desired. It also requires to install a patched version of iperf developed by CAIA, which allows to remove the cap in flow control windows.

5.1.3 Router Setup

The router has three network interfaces: one for the control network, and two for the experiment networks. It needs to provide routing between both experiment networks, by setting up one route to reach network B from network A, and another route to reach network A from network B.

The router also allows the use of the Linux network emulator (**netem**), which allows to implement queues to emulate delays and buffers on the network interfaces and emulate the effects of channel disturbances (e.g. packet loss). Netem is included in the **iproute2** linux package.

Necessary tools for logging TCP traffic traversing the router are also installed, such as **tcpdump**.

5.2 Experiment Orchestration with TEACUP

For any study that includes heavy experimentation, it is essential to be able to automate experiments. This involves being able to define a series of experimental scenarios with variable parameters, so that consecutive experiments can be launched automatically without the need of human operation. This improves time management and facilitates experiment reproducibility and repeatability.

TEACUP consists of a series of Python scripts that use the *Fabric* library [102] to configure experiment nodes and launch experiments. Experiments are orchestrated using a *fabfile*, which launches experiments according using the functions defined in a series of Python scripts, as shown in Figure 5.3. Experimental scenarios are defined using a configuration file (**config.py**), which allows to modify the network topology, set static values to experiment parameters and define parametric sweeps.

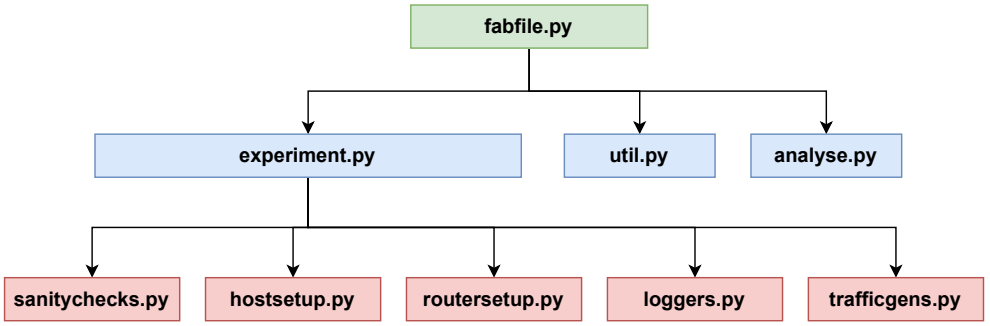


Figure 5.3: Architecture of main TEACUP scripts

For each parameter combination and experiment run, TEACUP performs a series of steps defined in **experiment.py** [103]. These steps can be summarized as the following:

1. Configure topology, assigning hosts to networks A and B.
2. Run sanity checks according to **sanitychecks.py**: check for necessary tools in the hosts, check connectivity and kill old processes.
3. Initialize hosts according to **hostsetup.py**: network configuration, modify kernel variables (e.g., to choose the TCP congestion control algorithm or set the size of receiving windows), synchronize clocks, etc.
4. Initialize router according to **routersetup.py**: configure routing and set link emulation queues.
5. Start loggers according to **loggers.py**.
6. Start traffic generators according to **trafficgens.py**.
7. Wait for experiments to end, and stop all processes.
8. Collect logs.

TEACUP also includes a series of scripts that analyze TCP experiment results and generate plots using **R** plotting libraries and **pdfjam**, which automatically generates PDF files containing the desired results. Analysis functions are defined in **analyse.py**.

5.2.1 Extending TEACUP for QUIC support

The TEACUP platform was initially designed for experimenting with TCP. However, TEACUP is a very flexible platform and it can be modified to execute any low level commands in the experiment hosts, which makes it extremely customizable.

In this study, the platform has been extended to be able to experiment with QUIC implementations. This involves (1) adding new traffic generators for the QUIC implementations to be used, (2) adding new code to handle the extraction of QUIC statistics and (3) other modifications of the existing TEACUP scripts to handle specific experimental scenarios.

5.2.1.1 New traffic generators

TEACUP traffic generators are defined in **trafficgens.py**, and they define a series of methods that specify the commands to be run on the endpoints to start traffic exchange. Most traffic generators consist of three methods: (1) one that starts the server application, (2) one that starts the client application, and (3) a third wrapper method that executes both applications consecutively on each endpoint - first the server and then the client. The command line instructions are executed remotely from the controller through SSH access and using **nohup**, which allows to execute commands in the background and detach from them. The execution in the background is managed by a series of method defined in **runbg.py** and the **runbg_wrapper.sh** bash file.

For this work, we have introduced two new traffic generators for two QUIC implementations: **ngtcp2** [66], based on the 'examples/client' and 'examples/server' applications; and **picoquic** [67], based on its 'picoquic_sample' application. More details on these implementations are given in Section 5.4.

The QUIC traffic generators introduced first perform a series of checks, to make sure that the user has introduced a valid port and server host in the configuration file, and they specify the path where logs will be stored. Then, the functions build the command-line instruction to execute the server or client applications:

- The **server** method specifies the route to the certificate and key for authenticating itself to the client. It also allows to set the congestion control algorithm according to the *ccalgo* parameter in the configuration file.
- The **client** method allows to specify the size of the download to be started. For this, a series of files have to be created in the server path '/root/files' with different sizes (e.g., 'file100MB', 'file1GB', etc.). The client also allows to repeat the download multiple times, according to the *download_repeat* parameter.

The full code for the *ngtcp2* and *picoquic* traffic generators is available in Appendix B.

5.2.1.2 New loggers

QUIC implementations use *qlog* [104] in order to log all the events happening in the endpoints: transport parameters exchanged, metrics updated, packets received or sent, packets lost, etc (more details on Section 5.5). The QUIC implementations already include *qlog* support, but the necessary methods need to be implemented in TEACUP in order to handle *qlog* files and retrieve them from the endpoints.

For this, two main methods have been introduced in a new script called **qlog.py** (see the full code listing in Appendix C):

- The **clean_qlog()** method, which removes all previous *qlog* files from the endpoints before each experiment run. Since *qlog* files can be quite heavy, this allows to prevent qlog files stacking and using a lot of storage in the endpoints. This function is called in the beginning of each run, from **experiment.py**.
- The **get_qlog()** method, which takes care of (1) compressing the generated *qlog* files using **gzip**, (2) renaming them according to their role and a counter (which is used when many multiple client-server pairs are run simultaneously) and (3) downloading them. This function is called in the end of the experiment run on **experiment.py**, after all traffic generators and other processes have been terminated.

These functions also need to account for the differences in the *qlog* implementation between *ngtcp2* and *picoquic*; e.g., *picoquic* generates a binary file which needs to be processed to make it text-based and readable.

5.2.1.3 Others

Other modifications have been made to TEACUP script to support some QUIC scenarios.

First, some minor functionality items where added, such as the possibility to enable and disable *qlog* logging through a parameter in configuration files was introduced, by setting 'TPCONF_enable_qlog' to '1' or '0' respectively.

Additionally, a missing functionality was identified when designing one set of experiments, which required some traffic generators being stopped at specific times one after the other. Since QUIC traffic generators cannot be told how long the download should be (they require to specify the download size in bytes), the approach we follow is to generate long connections and terminate them after some time. However, TEACUP only allows to terminate all traffic generators at the same time, which prevents designing some scenarios (e.g., latecomer fairness).

To solve this, an additional parameter named **kill_delay** was introduced in the functions named 'stop_process()' and 'stop_processes()' in **runbg.py**, which allows to specify how long the process termination needs to be delayed for each traffic generation instance. This allows to set up any possible multi-flow scenario on the configuration files, where flows start and terminate at different times.

5.3 Satellite Emulation

For this study, we contemplated two approaches for satellite emulation: first, emulating the satellite link in the testbed router using NetEm queues [105]; and second, using OpenSAND [97], a more advanced satellite emulation platform. Both of these approaches are described in the following subsections.

5.3.1 tc-netem

The Linux Network Emulator (NetEm or tc-netem) [105] allows to emulate the satellite link by implementing two queues in the testbed router, one for the downstream link and another one for the upstream link. Each of these queues can be defined with the following parameters:

- A **queue management method**, which determines how packets entering the queue are handled. We are using **FIFO**: First In, First Out.
- A **buffer size**, which limits the amount of packets that fit into the queue. In this study, we refer to the buffer size as a factor of the BDP.
- A fixed **queue delay**, which corresponds with the One-Way Delay (OWD).
- A **bottleneck bandwidth**, to limit the amount of traffic that can traverse the link.

NetEm also allows introduce artificial disturbances in the link - e.g. packet loss or reordering. In addition, it can be easily integrated in the TEACUP platform, since the queues can be set up and modified merely using command-line instructions.

The benefits of using NetEm are multiple: (1) it is natively implemented on the Linux kernel, (2) it can be implemented on any network node and (3) it can be configured through very simple command-line instructions. Nevertheless, it cannot be disregarded that using NetEm to emulate the satellite links lacks many low layer satellite mechanisms existing in real-life broadband satellites.

5.3.2 OpenSAND

OpenSAND [97] is a satellite emulation platform developed by Thales Alenia Space and the french government space agency (CNES). It provides a series of features on different layers, from radio resource management and PHY techniques to advanced routing and

Quality of Service (QoS) enforcing. OpenSAND is set up with three machines: a satellite terminal on the user side, a satellite emulator and a satellite gateway on the external side.

Our initial purpose was to introduce OpenSAND into the TEACUP testbed, according to Figure 5.4. However, a few challenges came up that left OpenSAND out of the scope for now. First, introducing OpenSAND requires a change in topology, which implies restructuring the experiment networks, changing the roles of the experiment nodes and adding new ones, while making it all compatible with TEACUP. Additionally, the lack of user-friendly documentation for the OpenSAND setup, the compatibility issues with OpenSUSE Linux distributions and the scarce available time made it difficult to implement it on time to meet the deadlines.

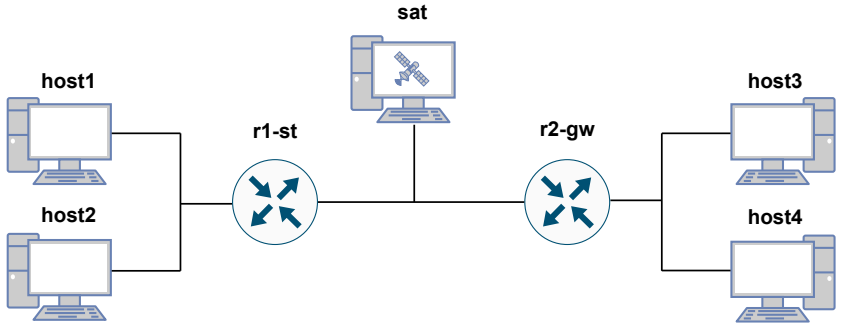


Figure 5.4: Proposed experiment network topology for OpenSAND

However, we encourage the future integration of OpenSAND in the UiS TEACUP testbed, since it would allow to repeat the experiments designed in this study and to compare results between both emulation methods.

5.4 QUIC implementations

From the publicly available QUIC implementations (already discussed in Chapter 2), two have been selected for this work: *ngtcp2* [66], mainly because it implements the newest version of BBR (i.e. BBRv2), and *picoquic* [67], since it allows to experiment with satellite-enhancing features.

These two implementations are compared in Table 5.2, which looks into their default parameters and available features. Even though it would be really interesting to look at more implementations and compare them, it takes time to learn their inner structure and how to work with them, and even more to implement them in the TEACUP platform.

These implementations need to be installed in all the endpoints. Bash scripts for installation of these implementations are listed in Appendix D.

	ngtcp2	**picoquic**
Initial Receiving Window	1 MB	1 MB
Max ACK Delay	25	10
Available CC	BBRv2/v1, Cubic	BBRv1, Cubic
HTTP/3	Yes	Yes
Path MTU Discovery	No	Yes
ACK Frequency Extension	No	Yes
BDP Frame Extension	No	Yes

Table 5.2: Available features and default parameters for selected QUIC implementations

5.5 Event Logging for QUIC

Extracting detailed statistics from TCP experiments requires the use of kernel tools such as *web10g* for Linux [106] and *SIFTR* for FreeBSD [107]. However, since QUIC operates completely on the user space, no kernel tools are necessary to extract event statistics.

Instead, IETF QUIC WG contributors have been working on a series of drafts that define *qlog*, a logging format for QUIC and HTTP/3 [104, 108, 109]. *qlog* is defined as a logging scheme that it is streamable, event-based and format-agnostic [104] - i.e., it can be serialized in many formats (JSON/JSON-SEQ, CSV, protobuf, etc.). Event definitions for QUIC, HTTP/3 and QPACK are listed in [108, 109].

Figure 5.5 shows the first lines of a *qlog* file generated by a *picoquic* connection. As it can be seen, the header shows relevant information such as the *qlog* version it uses, the QUIC implementation that generated it and timestamp for the start of the connection. It also shows the IP addresses, ports and connection IDs involved, and the transport parameters set by each one of the endpoints. Then, a long list of all the events registered during the connection can be found, including CC parameter updates, packets exchanged and received and frames included in each of them, ACKs and signaling, etc.

This work derives most of the performance evaluation metric values from *qlog* files.

5.6 Github Repository

The testbed is maintained in the **UiS-IDE-NG/network-testbed** Github repository [2]. Figure 5.6 shows a summary of the contents of this repository, which are organized as follows:

[2] https://github.com/UiS-IDE-NG/network-testbed

```
{ "qlog_version": "draft-00", "title": "picoquic", "traces": [
{ "vantage_point": { "name": "backend-67", "type": "server" },
"title": "picoquic", "description": "229a113b70d57bdb","event_fields": ["relative_time", "category", "event", "data"],
"configuration": {"time_units": "us"},
"common_fields": { "protocol_type": "QUIC_HTTP3", "reference_time": "1649776376544392"},
"events": [
[0, "transport", "datagram_received", { "byte_length": 1252, "addr_from" : {"ip_v4": "172.16.10.101", "port_v4":52997}
[0, "transport", "packet_received", { "packet_type": "initial", "header": { "packet_size": 1252, "packet_number": 0, "
    "frame_type": "crypto", "offset": 0, "length": 289}, {
    "frame_type": "padding"}]}],
[356, "info", "message", { "message": "ALPN[0] matches default alpn (picoquic_sample)"}],
[360, "transport", "parameters_set", {
    "owner": "remote",
    "sni": "test.example.com",
    "proposed_alpn": ["picoquic_sample"],
    "alpn": "picoquic_sample"}],
[364, "transport", "parameters_set", {
    "owner": "remote",
    "initial_max_stream_data_bidi_local" : 2097152,
    "initial_max_data" : 1048576,
    "initial_max_streams_bidi" : 513,
    "idle_timeout" : 30000,
    "max_packet_size" : 1440,
    "initial_max_streams_uni" : 513,
    "initial_max_stream_data_bidi_remote" : 65635,
    "initial_max_stream_data_uni" : 65535,
    "active_connection_id_limit" : 8,
    "max_ack_delay" : 10,
    "handshake_connection_id": "9aec103ffb99b861",
    "min_ack_delay" : 1000}],
```


Figure 5.5: Example qlog-JSON file generated by *picoquic*

- The **config** folder contains configuration files related to the network topology, IP addressing and routing, for the experiment nodes and the controller.
- The **documentation** folder contains all documentation produced for this testbed in the past for the first version of the testbed, as well as the documentation generated during this work.
- The **experiments** folder contains the TEACUP configuration files for all the experimental scenarios designed for this thesis, which are described in Chapter 6. They allow reproducing the experiments (see more on Appendix A).
- The **results** folder contains all post-processed results, including excel sheets, .csv files and pdf plots.
- The **teacup** folder contains the scripts of the extended TEACUP developed for this work.
- The **visualization** folder contains all the scripts written for post-processing of the *qlog* files. Statistics and metrics are extracted using **Python** and **R** is used for generating plots.

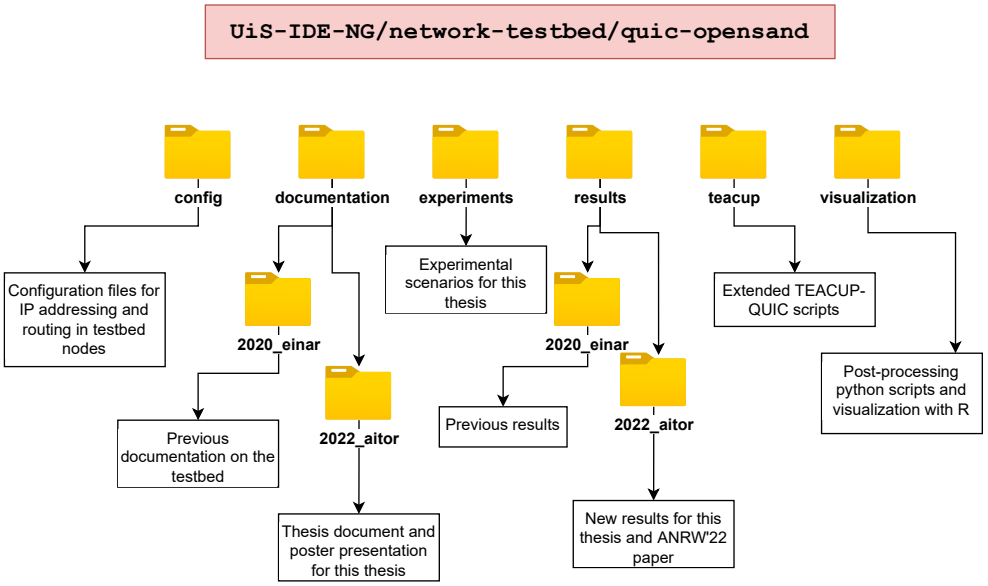


Figure 5.6: Overview of the contents in the UiS Network Testbed repository

Chapter 6

Experiments and Results

This section describes the experimental design and the different scenarios built, and it presents the obtained results. First, some general guidelines for all experiments are given; then the chosen metrics and scenarios are defined and justified; and finally, the results are presented using plots, tables and descriptions.

An brief user manual to reproduce all the experiments experiments is presented in Appendix A.

6.1 Experiment Design

We define various type of links for our experimental setup: the satellite links (SAT), which define different types of SATCOM links with different bandwidth values and asymmetry ratios, and the terrestrial link (TERR), which serves as a baseline for comparison, to analyze the impact of long delays and possible asymmetry.

The default values for link parameters are presented in Table 6.1. The values for the Bottleneck Buffer Size are defined as a fraction of the BDP. Unless specified otherwise, a default value of 1 BDP is assumed for this parameter.

	SAT	**TERR**
One Way Delay (OWD)	300 ms	50 ms
Downstream Bandwidth	20 Mbps	
Upstream Bandwidth	20 Mbps	
Bottleneck Buffer Size	0.25|0.5|**1.0**|2.0 x BDP	

Table 6.1: Link emulation parameters

All experiments in this study are repeated 10 times, to provide statistical insight.

6.2 Metrics

In order to numerically evaluate performance in the different scenarios to be built, we make use a of a series of metrics:

1. **Goodput**: measures the application level throughput - i.e., the ratio between the number of application bytes sent and the amount of time it took to send them, as shown in Equation 6.1. We derive this metric from QLOG traces in the receiver, checking the progress of the download offset and event timestamps.

$$Goodput(bytes/s) = \frac{bytesDelivered}{time} \tag{6.1}$$

2. **Link Utilization**: measures how well the available bandwidth is utilized, with a dimensionless ratio. In this work, we measure link utilization using Goodput values, and it is calculated using Equation 6.2.

$$Utilization = \frac{Goodput}{Bandwidth} \tag{6.2}$$

3. **Jain's Fairness Index (JFI)** [110]: measures the amount of fairness between multiple flows, taking values from 0 (completely unfair) to 1 (completely fair). The JFI can be calculated using Equation 6.3,

$$J(x_1, x_2, ..., x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2} \tag{6.3}$$

   where n is the number of parallel flows and x_i are the goodput values for each flow.

4. **Download Time**: measures the time it takes to complete the download of a single object or a series of objects. This metric is obtained from the QLOG traces in the client, by taking the timestamp of the reception of the last data segment in the download.

6.3 Scenarios

For this study, a series of experimental scenarios have been built. These have been grouped in three blocks, as shown in Figure 6.1, according to the solution they aim to evaluate: (A) using better congestion control, (B) accelerating the discovery of path parameters and (C) reducing congestion in the return link.

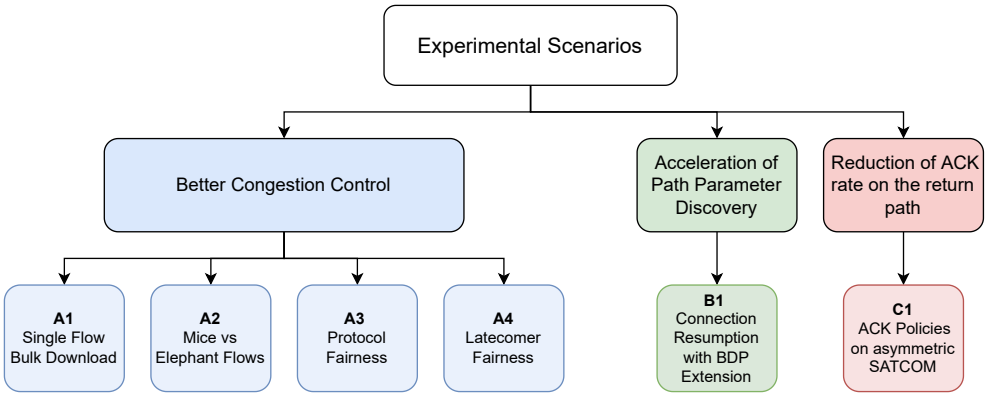


Figure 6.1: Overview of experimental scenarios

These scenarios are listed in Table 6.2, showing which experimental parameters they vary and which metrics are used in each one. All the scenarios are further detailed and justified in the following subsections.

Scenario	Description	Parameters varied	Metrics
A1	Single-Flow Bulk Download	CC, Buffer Size, PLR	Goodput
A2	Mice vs Elephant Flows	CC, Object Size, Number of Objects	Download Time
A3	Multi-Flow Fairness	CC, Number of flows	JFI, link utilization
A4	Latecomer Issue	CC, RTT	Goodput
B1	BDP Extension	Object Size, Resumption strategy	Download Time
C1	Asymmetric SATCOM links	ACK Policy, PLR	Goodput

Table 6.2: Summary of Experimental Scenarios

6.3.1 Block A: Better Congestion Control

This block of results aim to study the suitability of BBR congestion control over GEO SATCOM links, under different scenarios. These scenarios evaluate performance and fairness under different congestion and loss circumstances, testing how well different CC algorithms cope with them. The following subsections describe each of these scenarios.

6.3.1.1 Scenario A1: Single-Flow Bulk Download

Scenario A1 aims to measure bulk download performance with different QUIC implementations and CC algorithms, in both the satellite and the terrestrial scenario, to evaluate how well the link is utilized in the presence of elephant flows. This is achieved by initiating the download of a 1GB file, requested from the client to the server, as illustrated in Figure 6.2. The download is terminated after 180 seconds, and the average goodput is measured.

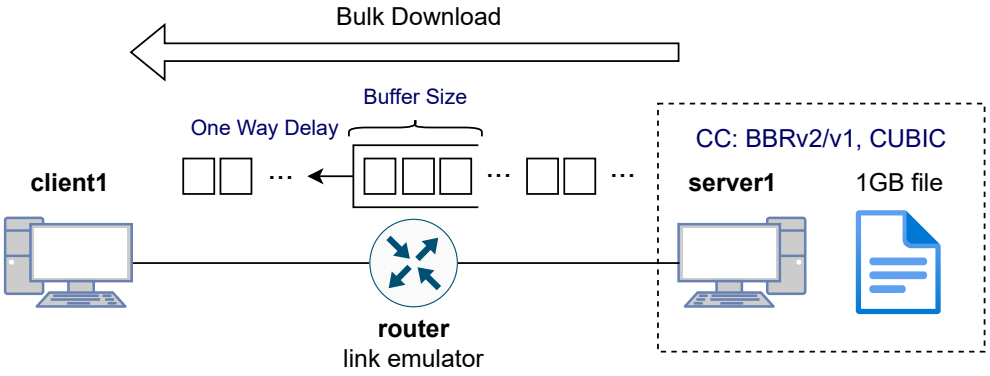


Figure 6.2: Illustration of scenario A1

This scenario helps to identify which QUIC implementations perform better under the satellite link, as well as the impact of CC under different packet loss conditions. The experiments will be repeated for multiple buffer sizes, and packet loss will be introduced artificially in the bottleneck queue.

6.3.1.2 Scenario A2: Mice vs Elephant Flows

Scenario A2 aims to measure the download time of different series of objects of different sizes. These experiments are run with the presence of background traffic, for a more realistic scenario and for evaluating the impact of CC for mice flows competing against elephant flows. A simple illustration of this scenario is shown in Figure 6.3.

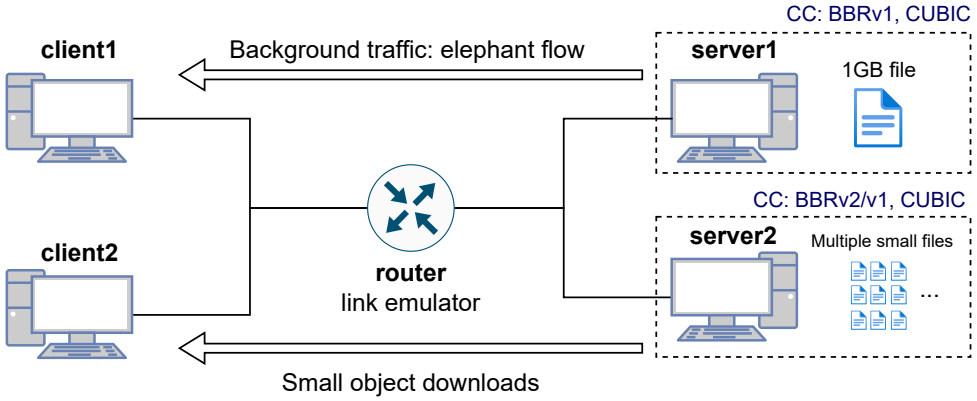


Figure 6.3: Illustration of scenario A2

The background elephant flow is generated between 'client1' and 'server1' using the same strategy as in scenario A1. 60 seconds after starting the elephant flow, the download of small objects is started between 'client2' and 'server2'. The experiments are repeated for different object sizes - 1KB, 10KB, 100KB and 1MB - and for different numbers of objects - 1, 10 and 100 objects. The possible influence of the CC in the background flow on the downloads is also tested by repeating the experiments with BBRv1 and

CUBIC. Performance is evaluated by measuring the download time of the different series of objects.

6.3.1.3 Scenario A3: Multi-Flow Fairness

Scenario A3 aims to evaluate the fairness of different CC algorithms in multiple-flow scenarios. Fairness is measured using Jain's Fairness Index (JFI), which is an accurate measure of fairness even when the number of flows increases. Flows are initiated simultaneously between two pairs of hosts, and they are terminated after 180 seconds. Odd-numbered flows run between 'client1' and 'server1', and even-numbered flows run between 'client2' and 'server2' -, as illustrated in Figure 6.4. This ensures an equal impact of CPU load on all flows.

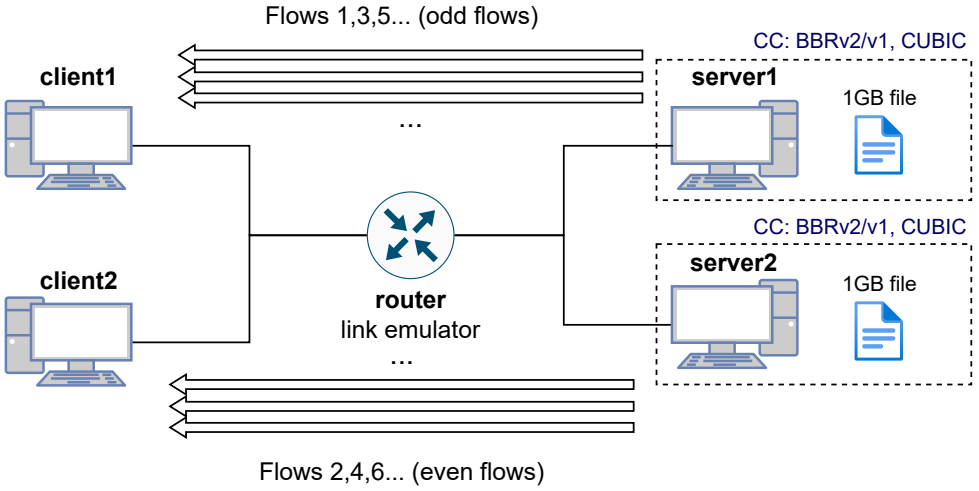


Figure 6.4: Illustration of scenario A3

This scenario allows to study two aspects of fairness: (1) the intra-protocol fairness - i.e., between flows that use the same CC algorithm; and (2) the inter-protocol - i.e., between flows that use different CC algorithms. Intra-protocol fairness is measured for 2, 4, 8, 16, 32 and 64 flows. Inter-protocol fairness is measured for different CC combinations of 2 and 4 flows. For this set of experiments, only *ngtcp2* will be evaluated, since *picoquic* does not implement BBRv2.

6.3.1.4 Scenario A4: Latecomer Issue

In the context of fairness, scenario A4 looks into the latecomer issue. This problem implies latecomer flows either (1) not being able to compete with other flows that are already utilizing the link or (2) being too aggressive towards them.

To evaluate this issue, a 4-flow scenario was designed where each flow starts 40 seconds after the other at a different time - at 0, 40, 80 and 120 seconds. As in the previous scenario, odd-numbered flows and even-numbered flows are started between different pairs of hosts, as shown in Figure 6.5. All flows are terminated 180 seconds after they are started.

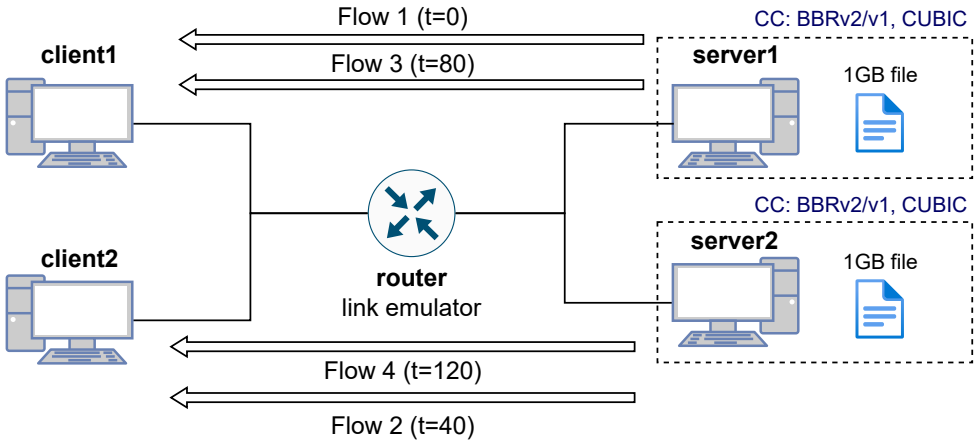


Figure 6.5: Illustration of scenario A4

To evaluate different performance aspects of these scenario, four items are considered:

1. **Convergence speed**. Latecomer flows SHOULD be as fast as possible at getting their correspondent bandwidth share.
2. **Aggressiveness towards existing flows**. Latecomer flows SHOULD NOT steal more bandwidth than they are corresponded with to existing flows.
3. **Long-term fairness**. All flows SHOULD converge to a fair bandwidth share.
4. **Recovery of available bandwidth after other flows end**. Once other flows are terminated and some bandwidth is freed, latecomer flows SHOULD be able to get the new available bandwidth.

6.3.2 Block B: Faster path parameter discovery

The second approach aims to study the possible benefits of using the BDP Frame extension implemented in *picoquic* for achieving faster CC convergence. This could improve not only short flows, by allowing a faster *cwnd* ramp-up for small downloads, but also longer flows, since this faster ramp-up allows to converge to the maximum available bandwidth faster, thus benefiting flows in the long term.

6.3.2.1 Scenario B1: Connection Resumption with BDP Extension

This scenario which allows to measure the download time for different object sizes on three different contexts:

- **The first connection**. The server and client do not know each other, so the default 1-RTT handshake is performed. In the end of this connection, the server sends a token to the client.
- **Zero-rtt resumption**. The client initiates a connection with a known server, sending the token in a 0-RTT packet. If the token is verified and accepted by the server, the 0-RTT handshake goes through, immediately starting the data transfer.
- **Zero-rtt resumption with BDP extension**. With the BDP extension enabled, the server sends a BDP frame in the end of the first connection. This BDP frame is sent back to the server by the client when attempting to resume a connection, allowing the server to optimize its CC parameters according to the BDP frame data.

Each experiment run for both connection resumption approaches is carried out following the next steps:

1. Start first 1-RTT connection client and server, with the BDP extension disabled or enabled.
2. Resume connection using the received token and start the download.
3. Compare the download time in both the initial connection and the resumed connection.

6.3.3 Block C: ACK policies for reducing congestion in return link

The third approach aims to analyze the impact of asymmetry on QUIC performance, and study the possible benefit of using custom ACK policies to reduce congestion in return link.

For this block, we define the **Asymmetry Ratio** to represent the asymmetry between the downstream and the upstream links, as shown in Equation 6.4,

$$AsymmetryRatio(\%) = 1 - \frac{BW_u}{BW_d} \tag{6.4}$$

where BW_u and BW_d are the bottleneck bandwidth values for the upstream and downstream respectively. Table 6.3 shows the Asymmetry Ratio values used for these experiments and their corresponding bandwidths.

Asymmetry Ratio (%)	Downstream Bandwidth	Upstream Bandwidth
0	20 Mbps	20 Mbps
50		10 Mbps
75		5 Mbps
90		2 Mbps
95		1 Mbps
97.5		500 kbps
99		200 kbps

Table 6.3: Correspondence between asymmetry ratio values and link bandwidth

In order to study the impact of the asymmetry, we measure the achieved goodput relative to the symmetric case. Results will be obtained with both *ngtcp2* - which does not use any ACK policy to benefit asymmetrical links - and *picoquic* - which implements the ACK Frequency extension by default and applies an ACK policy.

6.3.3.1 Scenario C1: Bulk download on asymmetric SATCOM

Scenario C1 measures the bulk download goodput achieved through the download of a large file, similarly to scenario A1. However, in this case there are two main differences: (1) the link is asymmetric and (2) different ACK policies are used.

Since an aggressive delayed ACK policy might have a negative impact on loss recovery, since it delays loss detection, the experiments will be repeated with the presence of packet loss. The experiments will also be run with the presence of cross-traffic on the upstream, which can be a typical scenario if there is heavy bidirectional traffic (e.g., videoconferencing traffic), and it can aggravate the issue of upstream congestion even more.

6.4 Results

This section presents and describes the results obtained in the different scenarios for blocks A, B and C. The results are presented using tables and graphs, produced from experimental data using **R** and its *plotly* library.

6.4.1 Block A results

6.4.1.1 A1: Bulk download results

Figures 6.6, 6.7 and 6.8 summarize the goodput results in the current scenario, for values of 0%, 0.1% and 1% packet loss (PLR) respectively.

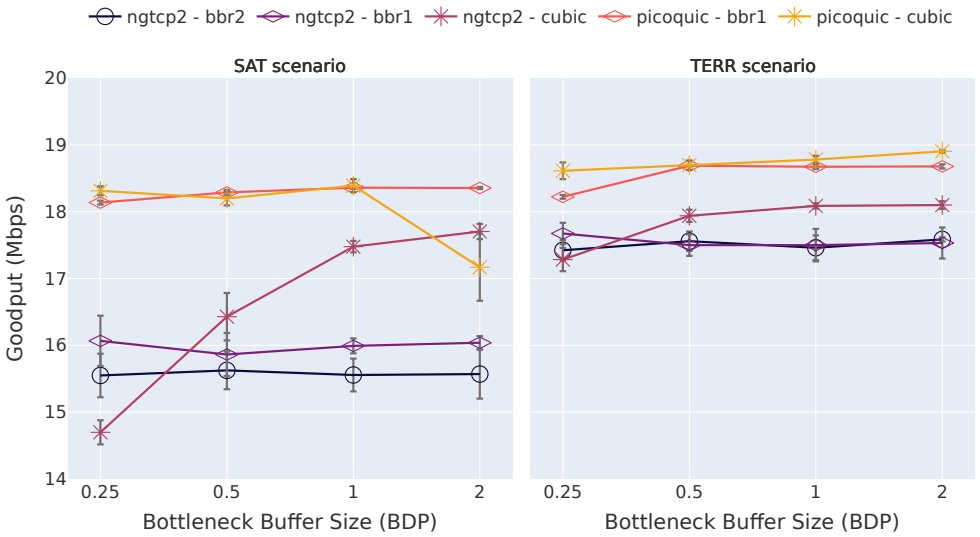


Figure 6.6: Bulk download goodput over an ideal link (PLR=0)

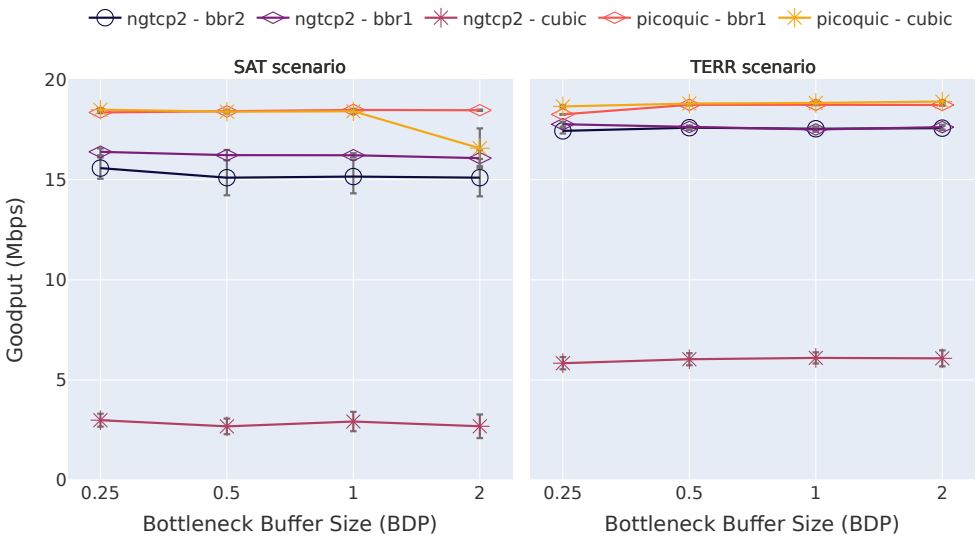


Figure 6.7: Bulk download goodput with losses (PLR=0.1%)

First, we focus on the ideal scenario with no packet loss, i.e. Figure 6.6. As expected, results show that performance over the satellite link is worse than over the terrestrial link in all cases, due to the long RTT. Results also show *picoquic* performing significantly better than *ngtcp2*, especially in the satellite scenario, where the performance boost provided by the implementation is more significant.

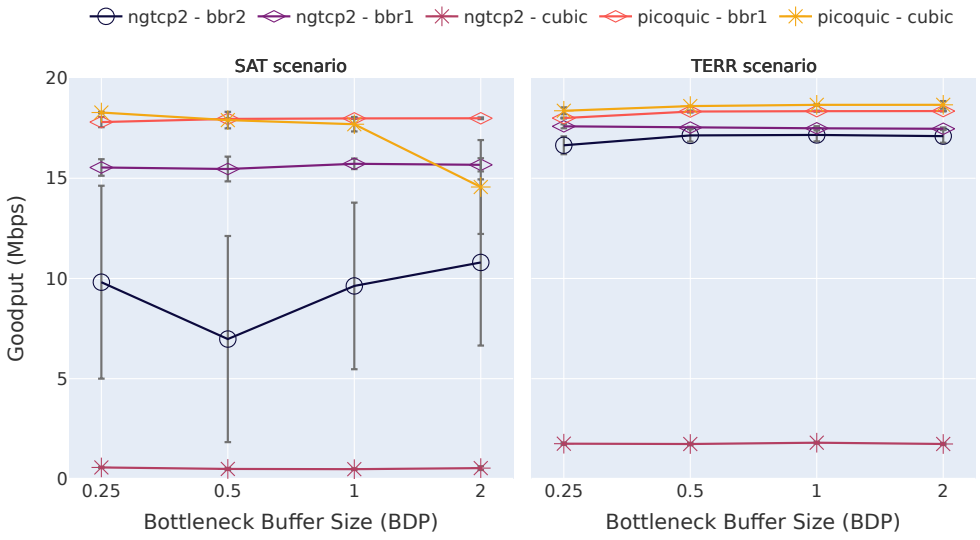


Figure 6.8: Bulk download goodput with losses (PLR=1%)

The choice of CC algorithm also proves to make a difference. For appropriate buffer sizes (i.e. over 1BDP) CUBIC appears to do better than both versions of BBR. BBRv1 and BBRv2 do not show significant improvements from the use of bigger buffers, and they perform slightly worse than CUBIC in the ideal scenario.

When packet loss is introduced in the link, as shown in Figures 6.7 (PLR=0.1%) and 6.8 (PLR=1%), we see CUBIC providing a dramatically lower goodput, down to around 3 Mbps in the case with 0.1% PLR and going under 1 Mbps in the 1% PLR case. This is a consequence of the loss-based nature of CUBIC, which performs poorly in lossy scenarios in the absence of lower-layer mechanisms that add robustness (e.g., FEC or Adaptive Coding and Modulation (ACM)).

Nevertheless, BBR keeps providing a similar performance in the presence of packet loss - even with a PLR of 1%, BBRv1 achieves goodput values over 15Mbps. Since BBRv1 sets the pacing rate only based on the bottleneck-bandwidth and the RTT, it maintains the pacing rate no matter the amount packet loss experienced, and the QUIC loss recovery mechanisms take care of retransmitting the lost packets. While BBRv2 shows a similar performance to the ideal scenario with 0.1% PLR, performance drops significantly in the high PLR experiments. This is potentially a result of the packet loss level surpassing the threshold set by the BBRv2 sender, after which the packet loss starts to have an impact on the CC behaviour.

It is possible to get a greater understanding on the reasons behind these results by looking at congestion window and RTT data. Figure 6.9 shows the *cwnd* and RTT evolution over time for a randomly picked run, for each CC algorithm available in *ngtcp2*, in the ideal (PLR=0) and high loss (PLR=1%) scenarios. Results clearly show how, while BBRv1 is

barely affected by the presence of packet loss in terms of *cwnd* and RTT dynamics, the impact on BBRv2 and CUBIC is more significant, especially in the latter.

6.4.1.2 A2: Mice-flow results

Figure 6.10 shows the download times for different numbers of objects and object sizes, with each CC algorithm available. Experiments are run in the presence of an elephant flow in the background: BBR background traffic (plots on the first row of Figure 6.10) and also using CUBIC background traffic (plots on the second row of Figure 6.10).

First of all, results show an exponential increase of the download time as the object size increases, and also as the number of objects increases. This is expected, as a result of the exponential increase of the *cwnd* in the start of the connection.

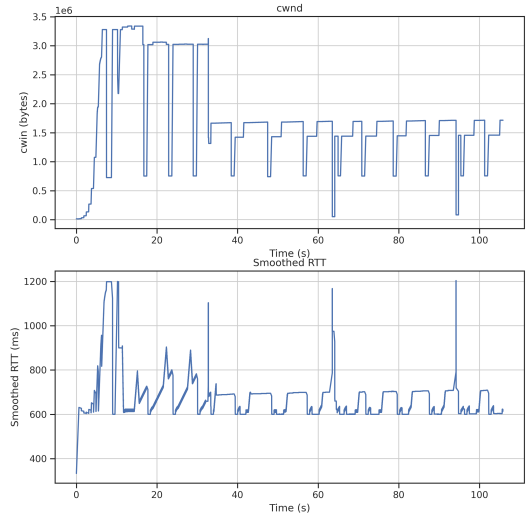
Results in the presence of BBR background traffic show that BBRv1 achieves the shortest download times, especially for heavier object sizes, while CUBIC takes around twice as long as BBRv1 in the 1MB case. BBRv2 itself also outperforms CUBIC, but it performs a bit slower than BBRv1, as already hinted in the previous results from scenario A1.

When the background traffic is CUBIC, we see download times increasing in most cases. BBRv2 suffers a dramatic performance drop when competing against a CUBIC elephant flow. BBRv1 also sees an increase in download times; however, it still beats CUBIC. This demonstrates that the fact that BBRv1 beats CUBIC is not a consequence of BBRv1 being too agressive towards it.

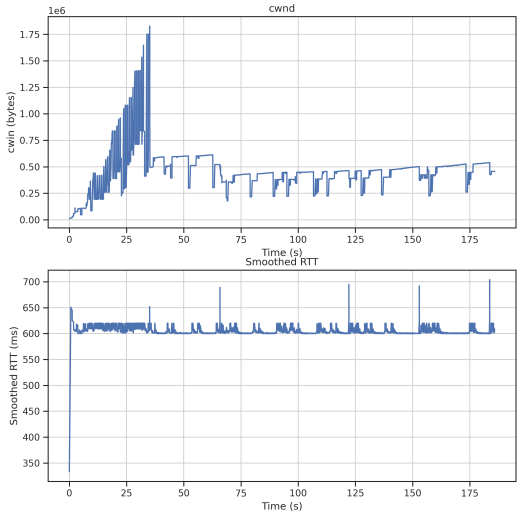
6.4.1.3 A3: Multi-flow fairness results

Figure 6.11 shows the results of the intra-protocol fairness tests with 2 flows. The figure show a clear winner - BBRv1 is the fairest of all the candidates -, followed by CUBIC and BBRv1. In addition, the size of the bottleneck queue does not appear to have a clear impact on fairness.

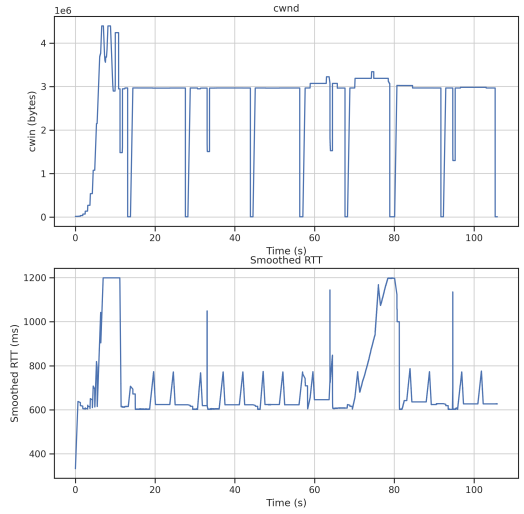
Results for intra-protocol fairness tests with multiple parallel flows are shown in Figure 6.12. Bandwidth utilization for these tests is also shown in Figure 6.13. The results shows that, although bandwidth utilization improves, fairness in BBRv1 and BBRv2 starts to drop significantly as the number of flows increases. Even though BBRv2 appears to behave more fairly than BBRv1 for a high number of flows, the JFI scores achieved by both BBR versions are not appropriate. Meanwhile, CUBIC achieves an outstanding fairness score even for 64 parallel flows.



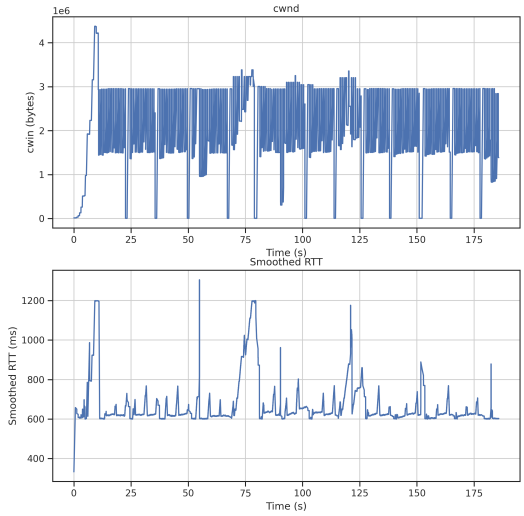
(a) BBRv2, PLR=0%



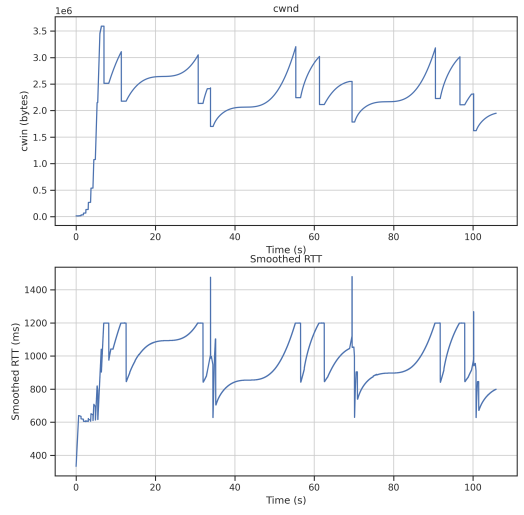
(b) BBRv2, PLR=1%



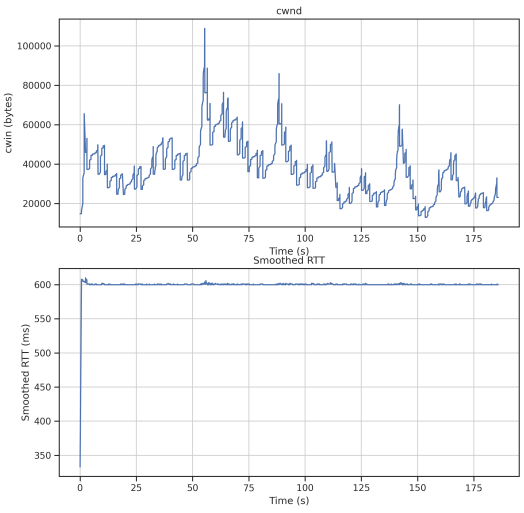
(a) BBRv1, PLR=0%



(b) BBRv1, PLR=1%

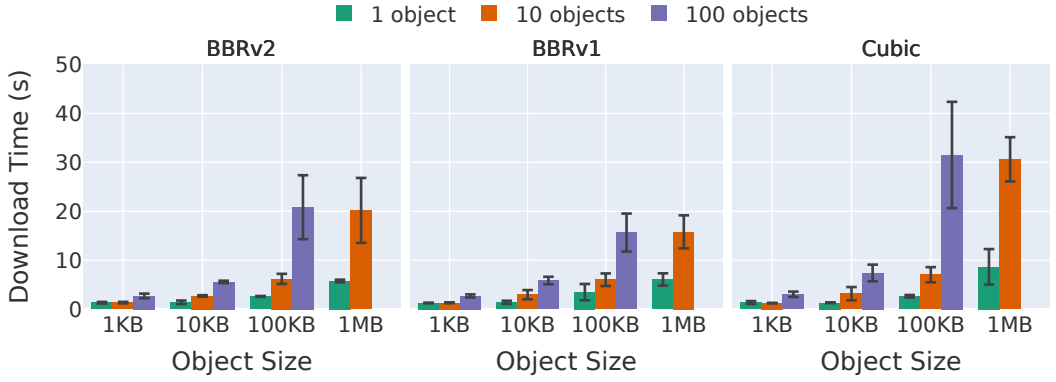


(a) CUBIC, PLR=0%

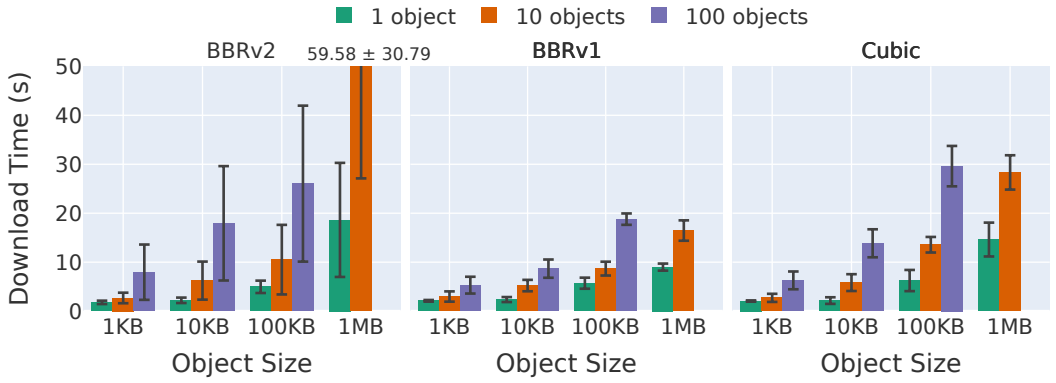


(b) CUBIC, PLR=1%

Figure 6.9: Congestion window and RTT evolution for a randomly picked run, for each CC algorithm, in the ideal (PLR=0) and high loss (PLR=1%) scenarios.



(a) BBR background traffic



(b) CUBIC background traffic

Figure 6.10: Mice flow experiment results for BBR and CUBIC background traffic

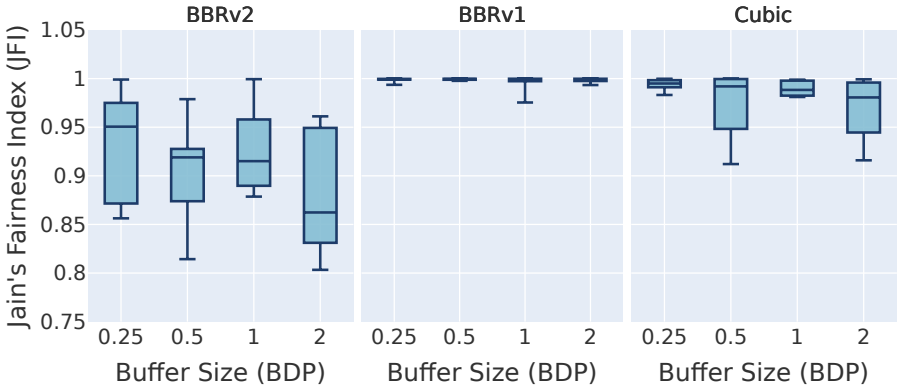


Figure 6.11: Intra-protocol fairness tests with two parallel flows, with *ngtcp2*

Afterwards, we look at the inter-protocol fairness with three different scenarios using different CC combinations: (A) BBRv2 vs BBRv1, (B) BBRv2 vs CUBIC and (C) BBRv1 vs CUBIC. Figure 6.14 shows the JFI obtained for all possible CC permutations with 2 and 4 parallel flows. The JFI is computed using all flows individually and shows the fairness score between all of them, but it does not reveal which CC algorithm is dominating. To solve this, the relative bandwidth share of each flow is shown in the pie

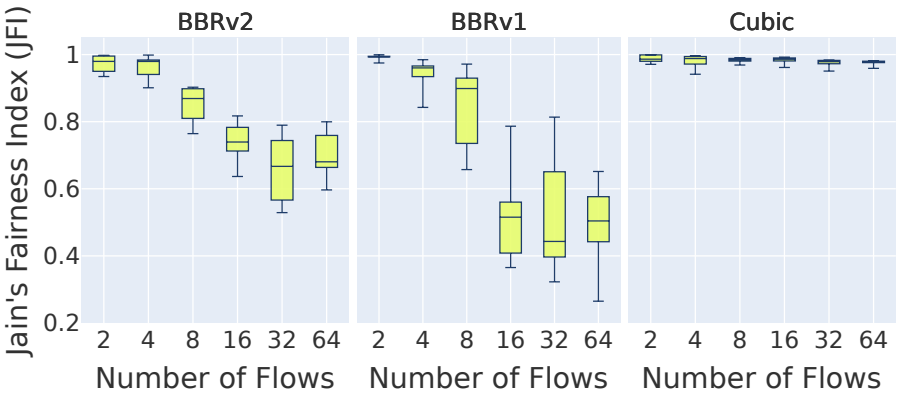


Figure 6.12: Intra-protocol fairness tests with different numbers of parallel flows, with *ngtcp2*

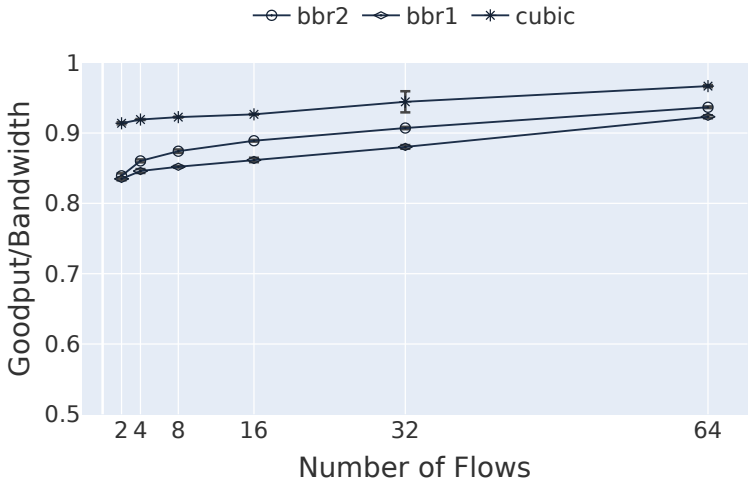


Figure 6.13: Bandwidth utilization for intra-protocol fairness tests

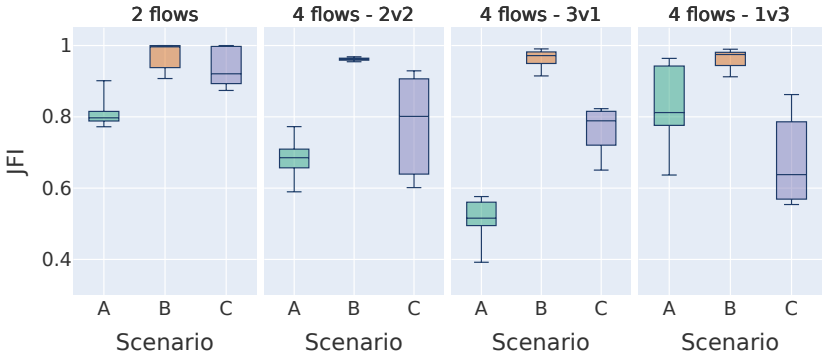
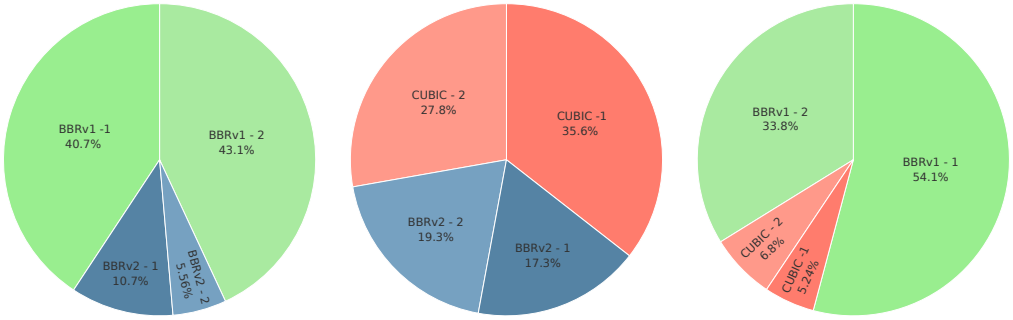


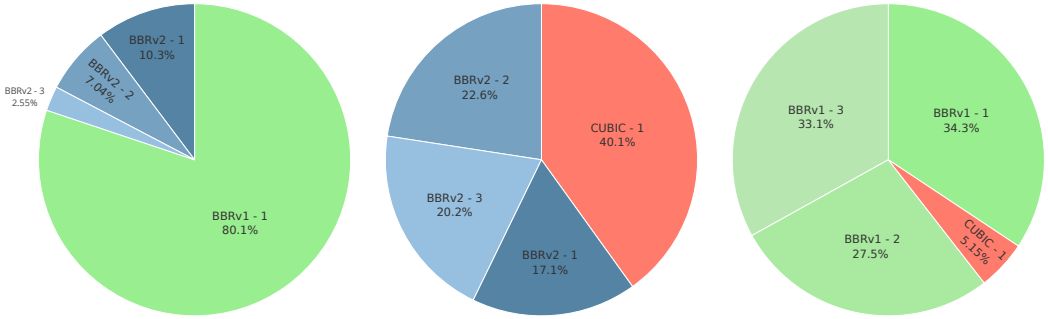
Figure 6.14: Inter-protocol fairness tests for different CC combinations, with *ngtcp2*

charts in Figure 6.15.

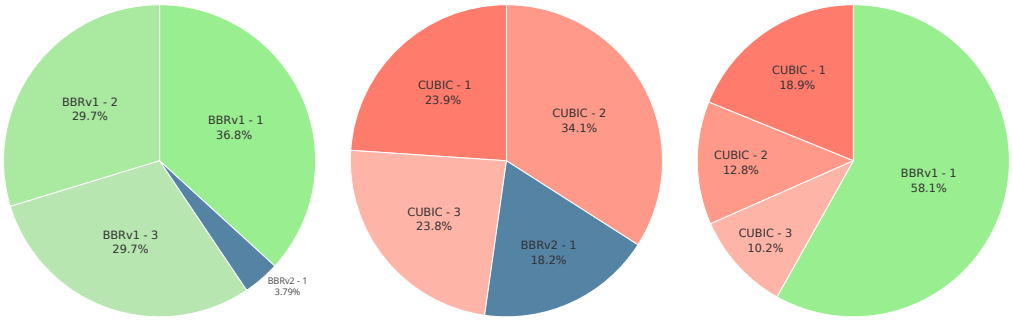
Results show that, while BBRv1 behaves very aggressively towards CUBIC, leaving CUBIC flows with a small bandwidth share (see scenario C and corresponding pie charts), BBRv2 provides a fairer behavior towards CUBIC in all cases (see scenario B), with JFI



(a) 2v2 inter-protocol scenarios



(b) 3v1 inter-protocol scenarios



(c) 1v3 inter-protocol scenarios

Figure 6.15: Relative bandwidth shares for 4-flow inter-protocol experiments with *ngtcp2*

scores over 90%. When the two BBR versions compete against each other, BBRv2 gets great disadvantage.

6.4.1.4 A4: Latecomer test results

Figure 6.16 summarizes the results of the latecomer experiments for both the satellite and terrestrial links, showing the goodput achieved by each flow over time, as well as the

aggregate goodput and the smoothed RTT measured by each flow. Overall, we see the endpoints in all flows measuring the RTT with very close results. In terms of bandwidth utilization, we observe quite stable aggregate goodput over time over the terrestrial link, while the satellite link makes it oscillate more due to the long feedback loop, especially with BBR CC.

When looking at flows individually and studying how they enter the link, we can draw several observations, summarized in Table 6.4. In terms of convergence speed, we see CUBIC being the slowest candidate, especially in the SATCOM scenario, due to the long RTT. Since both BBR versions share a very similar *cwnd* ramp-up approach, they both converge at a similar pace.

Looking at the short-term fairness to evaluate the aggressiveness towards existing flows, we see BBRv1 latecomers stealing an excessive amount of bandwidth from previous flows. BBRv2 latecomers, however, join the link more gently and do not greatly damage existing flows. In terms of long-term fairness, all the candidates do a good job, but especially CUBIC and BBRv2.

We also identify an issue where BBRv2 fails to recover the available bandwidth after the rest of the flows are terminated (see item (a) in Figure 6.16). This issue is only present in the SATCOM scenario, which hints that it is a product of the long RTT not allowing the CC orchestrator to fully utilize the available bandwidth.

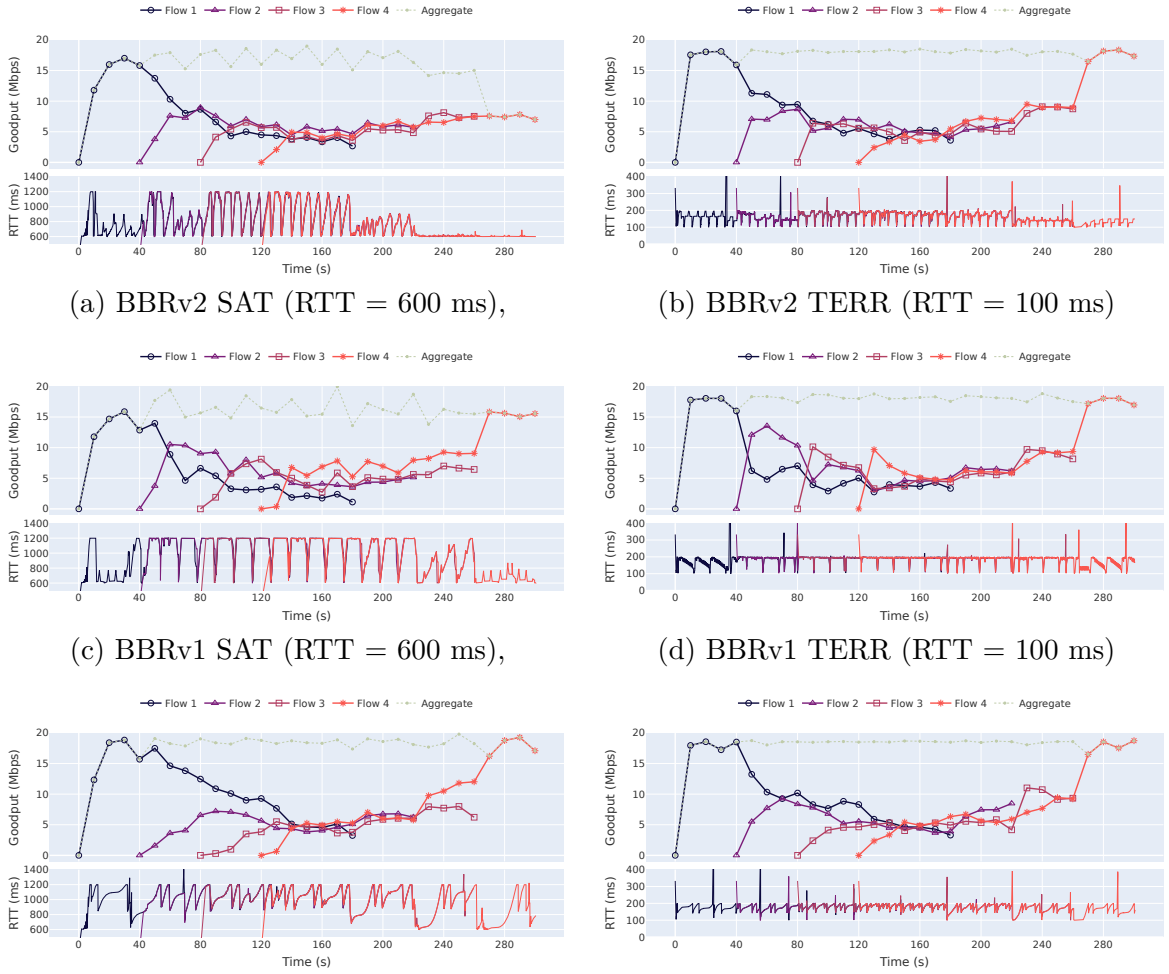
Qualitative Metric	BBRv2	BBRv1	CUBIC
Convergence speed	Good	Good	Poor
Short-term fairness	Best	Poor	Good
Long-term fairness	Good	Decent	Best
Recovery of available bandwidth	Poor*	Good	Good

Table 6.4: Qualitative latecomer fairness evaluation for BBRv2, BBRv1 and CUBIC. * For the SATCOM scenario

6.4.2 Scenario B1 results

Figure 6.17 shows the download times for different object sizes using *picoquic*, comparing the first connection with the two approaches for connection resumption (without and with the exchange of the BDP Frame). Download times for smaller objects were not shown because connection resumption does not seem to have an impact on download times in their downloads.

Results show that the connection resumption allows to save around 1 RTT (600 ms) in the download, and around 1 second for 50MB objects. However, our results do not show



(a) BBRv2 SAT (RTT = 600 ms), (b) BBRv2 TERR (RTT = 100 ms)

(c) BBRv1 SAT (RTT = 600 ms), (d) BBRv1 TERR (RTT = 100 ms)

(e) CUBIC SAT (RTT = 600 ms), (f) CUBIC TERR (RTT = 100 ms)

Figure 6.16: Latecomer experiment results for different CC in both the satellite and terrestrial scenarios

any additional advantage with the exchange of the BDP frame, no matter the object size. Experiments were also repeated using different CC algorithms, but no noticeable change in behavior was observed.

6.4.3 Scenario C1 results

Table 6.5 shows the goodput ratio in relation to the symmetric scenario, for different asymmetry ratio values. Results show that the lack of a satellite-optimized ACK policy in *ngtcp2* introduces a notable performance degradation as the asymmetry ratio increases, especially after hitting the 95% mark. Meanwhile, *picoquic*'s performance remains constant even with very high asymmetry.

Experiments were repeated with the presence of cross-traffic in the upstream link, producing the results shown in Table 6.6. Results show that the presence of upstream

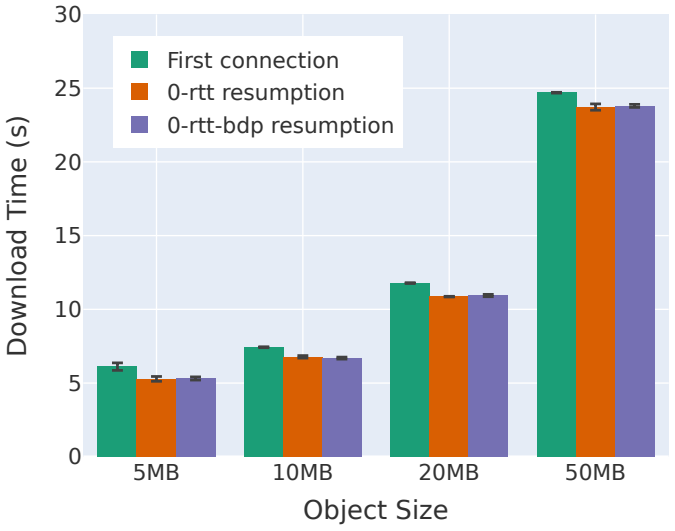


Figure 6.17: Download times for different object sizes, using 0-rtt and 0-rtt-bdp connection resumption approaches

traffic aggravates the issue: with only 50% of asymmetry we already observe a performance decrease of around 12%.

Statistics about the number of ACKs sent by the receiver and STREAM (data) frames received were also extracted from the experiments, and presented in Figures 6.18 (without loss) and 6.19 (with 1% PLR). Results reveal that *ngtcp2*'s lack of ACK policy is limited to the default QUIC 1:2 ratio (i.e., 1 ACK sent for every two packets); meanwhile, *picoquic* uses an ACK ratio of around 1:17. Results also show that this ACK policy implemented by *picoquic* does not depend on the level of asymmetry, since it uses the same ACK ratio regardless of the asymmetry ratio.

			Asymmetry Ratio					
			50%	75%	90%	95%	97.5%	99%
PLR	ACK Policy		Goodput ratio (%)					
0%	None (ngtcp2)	Mean	99.52	99.40	99.27	95.15	66.70	27.88
		Stdev	1.37	1.35	2.18	2.70	0.97	0.34
	picoquic	Mean	99.92	99.97	100.06	100.05	100.03	100.06
		Stdev	0.31	0.33	0.21	0.21	0.19	0.22
1%	None (ngtcp2)	Mean	100.60	98.77	100.37	75.81	41.74	19.36
		Stdev	2.57	3.47	2.82	2.41	1.21	0.53
	picoquic	Mean	99.91	99.99	99.58	99.99	100.01	99.63
		Stdev	0.42	1.19	0.38	0.47	0.19	0.22

Table 6.5: Goodput ratio relative to symmetric scenario

			Asymmetry Ratio					
			50%	75%	90%	95%	97.5%	99%
PLR	ACK Policy		Goodput ratio (%)					
0%	None (ngtcp2)	Mean	88.90	73.41	58.85	30.56	13.61	4.31
		Stdev	8.58	14.63	7.39	8.08	2.18	2.43
	picoquic	Mean	100.01	100.04	99.95	99.92	99.92	99.96
		Stdev	0.24	0.20	0.22	0.20	0.22	0.20
1%	None (ngtcp2)	Mean	99.17	78.02	53.14	24.27	13.22	5.81
		Stdev	7.09	10.26	11.29	5.85	2.73	3.45
	picoquic	Mean	99.67	99.90	99.85	99.86	99.80	99.90
		Stdev	0.30	0.38	0.43	0.36	0.34	0.34

Table 6.6: Goodput ratio relative to symmetric scenario, with cross-traffic on the upstream

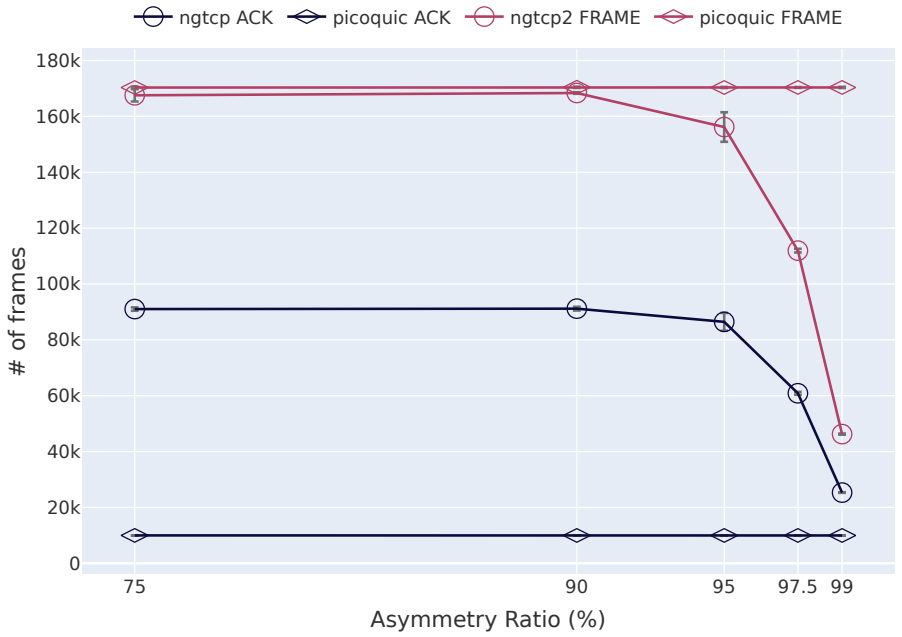


Figure 6.18: Number of ACK frames sent and STREAM frames received by the client, on the ideal scenario (PLR=0)

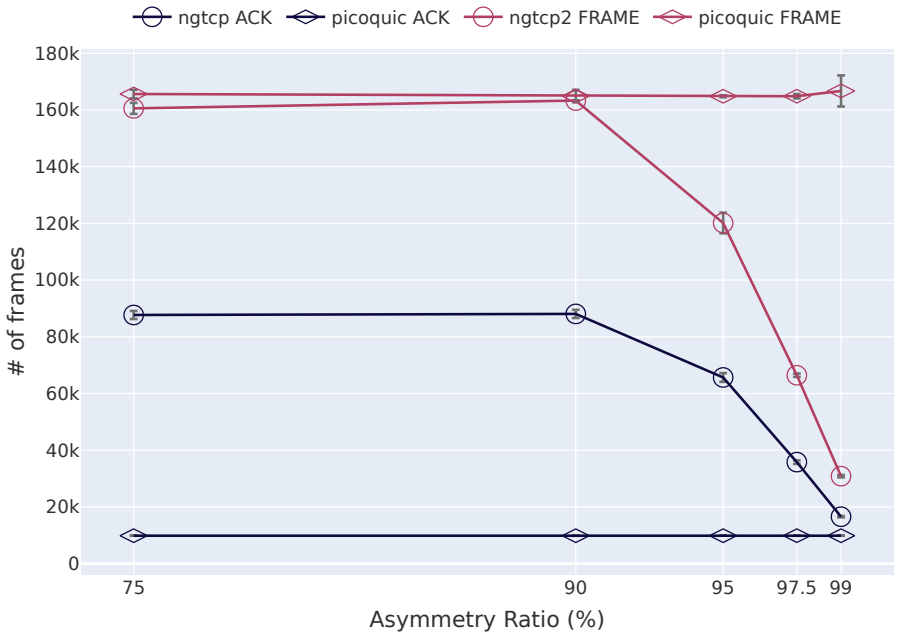


Figure 6.19: Number of ACK frames sent and STREAM frames received by the client, under loss conditions (PLR=1%)

Chapter 7

Discussion

The obtained set of the results for the different proposed solutions and experimental scenarios has led to a series of observations that can generate a very interesting discussion. This discussion aims to not only provide a broader view on the implications of the results, but also to comment on the practical implementation feasibility of the solutions, and to hint potential future research paths related to the different mechanisms proposed.

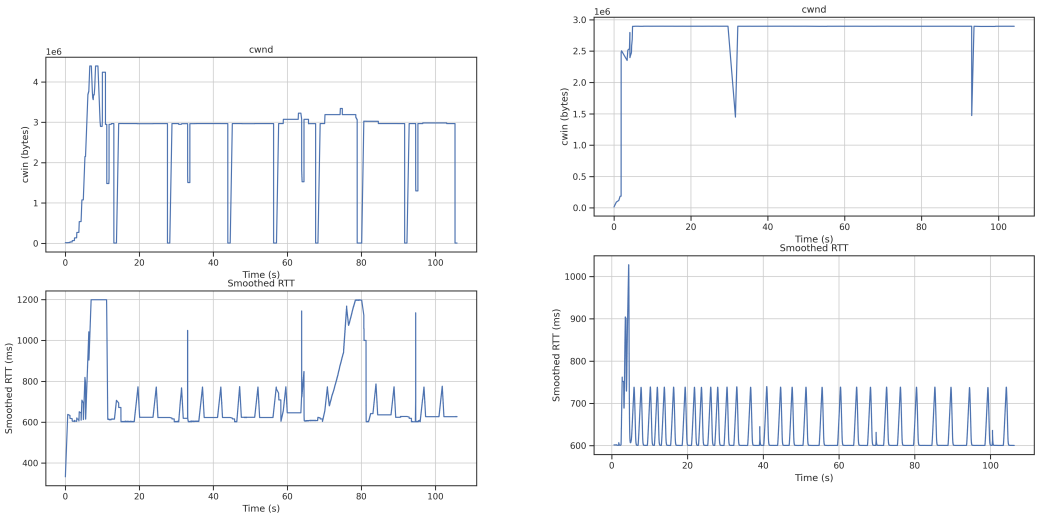
7.1 Impact of QUIC Implementation

Before starting a discussion on the different solutions proposed, some insight needs to be added into the heterogeneity observed between the QUIC implementations used in this study (*ngtcp2* and *picoquic*). Even though both implementations are built based on the same IETF QUIC specification, they are still under development, and small differences in software design might introduce different behavior patterns under certain conditions.

Observations from scenario A1 (section 6.4.1.1) have shown *picoquic* clearly outperforming the other candidate in terms of bandwidth utilization, especially under high BDP conditions, providing a goodput increase of around 16% in the BBRv1 case. It is also possible to see *picoquic* performing really well under high packet loss conditions with CUBIC, as opposed to *ngtcp2*'s CUBIC.

To gain deeper insight, we can compare the *cwnd* and Smoothed RTT at the sender with both implementations. Figure 7.1 presents these parameters over time on randomly chosen runs, using BBRv1 CC, 1 BDP buffer size and absence of packet loss. The data shows different BBR behaviors: even though *ngtcp2* BBRv1 drains the queue by reducing the *cwin* to 0 on regular intervals, *picoquic* BBRv1 maintains a lower average RTT on

the path while keeping a high goodput. We also observe *picoquic*'s slow-start increasing the *cwnd* faster.



(a) ngtcp2 (b) picoquic

Figure 7.1: *cwnd* and Smoothed RTT on the sender for single-flow experiments (BBRv1, 1BDP, 0% PLR)

Results also indicate that this performance difference is not only a matter of CC choice, and *picoquic*'s advantage might be a result of other aspects of the transport layer. Even though for these experiments flow control window limits have been uncapped, it is possible that the flow control tuning algorithm in some QUIC implementations might be limiting performance when the BDP is high. It is also likely that other mechanisms unique to *picoquic* might be improving performance in this scenario, such as Path MTU Discovery (PMTUD) or ACK policies fine-tuned for satellites (see more on 7.4).

All these observations hint that the next following years will witness several iterations in the development of QUIC implementations and IETF specification: some implementations will include experimental features that introduce improvements to the protocol under SATCOM scenarios, and these features will likely be adapted by other agents.

7.2 Impact of Congestion Control

The choice of congestion control for QUIC over SATCOM has demonstrated to play a big role in many aspects of network performance and fairness, as observed in Block A results. Since observations do not show a clear winner, it is reasonable to do follow a qualitative comparison that evaluates how well each CC algorithm performs in different performance and fairness aspects.

Table 7.1 summarizes the observed performance and behavior of each CC algorithm, in the different studied aspects of performance and fairness, in line with the experimental scenarios designed for Block A. In this discussion section, we are focusing on *ngtcp2*, which implements the latest iteration of BBR, allowing to study if it solves the issues of previous versions and if it introduces any other additional challenges.

Qualitative Metric	**BBRv2**	**BBRv1**	**CUBIC**
Bulk Download over ideal link	Good	Good	**Best**
Bulk Download over lossy link	Decent	**Best**	Poor
Mice Flow Performance	Good	**Best**	Poor
Intra-Protocol fairness	Poor	Poor	**Best**
Fairness towards CUBIC	**Best**	Poor	-
Latecomer fairness	**Best**	Decent	Good

Table 7.1: Qualitative comparison of CC algorithms with *ngtcp2*

Our observations have shown that BBR congestion can be highly suitable for SATCOM links, providing better bulk download performance under lossy links - which can help mitigate the impact of satellite propagation errors - and faster downloads with mice flows. This suggests that BBR might be key for improving QoE in Internet applications that involve large downloads (e.g., those in video streaming clients) and also smaller but multiple concurrent downloads (e.g., web browsing). Using the latest BBRv2 also seems to be the right choice for improving fairness, which is fundamental for the well-being on the Internet, especially when the number of flows on the same link increases. It has also been shown that link utilization appears to increase with a large number of simultaneous flows - in some cases it might be more efficient to open multiple QUIC connections.

However, there are still clear issues with BBRv2: (1) the intra-protocol convergence scores are still not acceptable when the number of concurrent flows grows, which can be problematic when BBR traffic starts to gain more presence on the Internet; (2) the unfairness between BBRv2 and BBRv1 (although it is expected that BBRv2 replaces its previous version) and (3) an issue with BBRv2's capability to use the available bandwidth efficiently after congestion events for high RTT links. These should be further investigated and addressed in upcoming BBR versions.

Another matter yet to be investigated is the impact of CC in multi-streaming scenarios. With QUIC's capability to multiplex streams with a common congestion controller, but with stream-specific flow control, the CC choice might play an important role. This can be studied through applications running over QUIC that make smart use of the stream multiplexing feature.

7.3 Impact of BDP Frame Strategy

Even though the BDP frame strategy seemed to be a promising candidate for improving QUIC's performance over SATCOM, our results have been able to demonstrate any benefits from its use, unlike the study in [36].

We suggest three possible reasons why our results might not be showing any improvements:

- The experimental scenario might not be properly designed to evaluate this strategy.
- The safety mechanisms to avoid CC being too aggressive might be preventing optimization under the proposed satellite link emulation.
- The satellite emulation might be missing some lower layer mechanisms that play an important role in CC convergence.

Anyways, this motivates further research into how the path parameter data exchanged through BDP frames is actually used in the CC optimization.

7.4 Impact of ACK Frequency Strategy

The experimental evaluation in Block C has shown significant performance degradation when link bandwidth asymmetry is present, which is very common in satellite broadband services. As shown in Table 7.2, satellite broadband access providers such as HughesNet or Viasat in USA or Eutelsat in Europe offer plans with high asymmetry, often surpassing an asymmetry ratio of 95%. Even though asymmetry might not have been a problem in the past, web browsing nowadays introduces significant uplink traffic (e.g., in the form of cookies), and even more in interactive live streaming or videoconferencing.

Our results have revealed that, for a 95% asymmetric link with the presence of upload traffic, download performance can get highly degraded due to congestion in the upstream bottleneck, losing around 70% of download goodput. Luckily, we have witnessed great benefits from the use of ACK policies powered by the ACK Frequency extension in *picoquic*, which manage to maintain high performance over highly asymmetric links by negotiating a reduced ACK delivery frequency. This motivates further study of these policies and addition of this extension to other QUIC implementations.

Since delaying ACK delivery could also be dangerous for the well-being of loss recovery mechanisms, these ACK policies should be carefully studied. An interesting approach would be to make these ACK policies be dynamically fine-tuned according to the

bandwidth asymmetry in the link, so that no extreme ACK delay is introduced if it is not necessary. Another possible approach would be to introduce ACK Congestion Control over QUIC, such as the one proposed for TCP in RFC 5690 [111], allowing to adapt the rate of sending ACKs according to the congestion in the uplink.

Provider	**Bandwidth**	**Asymmetry Ratio**
HughesNet (USA)	Up to 25/3 Mbps	88%
Viasat (USA)	Up to 100/3 Mbps	97%
Eutelsat (Europe)	Up to 75/3 Mbps	96%

Table 7.2: Satellite broadband plans offered by some popular providers in USA and Europe

Chapter 8

Conclusion

This thesis has investigated the convergence between major breakthroughs in the Internet transport layer (i.e., QUIC and BBR) and new use cases for SATCOM broadband access, identifying the challenges involved and evaluating performance-boosting solutions.

Our proposal has studied three different strategies to potentially improve the performance of QUIC over SATCOM: (1) the use of BBR congestion control, (2) the early exchange of path parameters between endpoints using the BDP extension and (3) the use of custom ACK policies for reducing ACK congestion in return links. Performance results have been obtained using a physical network testbed located at UiS and an emulated satellite link based on *netem/tc*, with a 600 ms round-trip time and 20 Mbps of bottleneck downstream bandwidth. Experiment orchestration and automation has been facilitated by the development of an extended version of TEACUP, which allows to design experiments using different QUIC implementations.

The complete set of experiment results allows us to answer the research questions formulated in the beginning of this document. They also open up a series of questions which might lead to future research paths which can be important for QUIC's further standardization and performance improvements over SATCOM broadband access.

Part of this work has also derived into an accepted publication for the Applied Networking Research Workshop 2022 (ANRW'22). Further details on this publication, as well as a pre-print for the paper, can be found in Appendix E.

8.1 Answers to the Research Questions

RQ1: *Can the performance of QUIC over SATCOM links be improved using transport protocol mechanisms?*

Yes. This study has proven that it is possible to improve QUIC's performance over SATCOM without the need of any proxies that compromise the end-to-end principle, and merely by implementing a series of mechanisms in QUIC endpoints that help to mitigate the challenges created by the satellite link.

- **RQ1.1**: *Can better congestion control algorithms improve bandwidth utilization, download speed and fairness?*

  Yes. Results have proven that using BBR congestion control instead of CUBIC can greatly improve download performance in both large and short downloads over SATCOM links, especially under the presence of propagation errors. Using the latest version of BBR (BBRv2) has also demonstrated being able to improve fairness towards CUBIC flows and latecomer convergence. However, a series of issues with BBR CC have also been identified: (1) unfair behavior between BBR flows and (2) inability of BBRv2 to recover available bandwidth after congestion events on long RTT links.

- **RQ1.2**: *Can the early exchange of BDP information between endpoints using the BDP Extension improve congestion control convergence?*

  It is unsure. While some early studies have shown a potential performance gain with the use of this extension, our results do not show any clear benefits from its use.

- **RQ1.3**: *Can custom ACK policies on the client-side improve performance on asymmetric satellite links?*

  Yes. Our results have proven that satellite-optimized ACK policies can help minimize the performance degradation on highly asymmetric links, especially under the presence uplink traffic.

RQ2: *If the answer to RQ1 is positive, are these mechanisms safe and feasible to implement?*

They can be, with proper design. Some of the solutions evaluated and discussed can be benefitial for SATCOM broadband but also a bit aggressive.

While BBR seems to be appropriate for both terrestrial and satellite links, ACK policies or BDP Frame acceleration could introduce additional issues unless proper safety checks are implemented along with them.

8.2 Future Directions

The outcome of this work also opens up a series of future research paths.

- **Improve BBRv2 over SATCOM**. Even though BBRv2 has proven to be great for SATCOM, the algorithm needs to be further refined in order to tackle the fairness issues identified. This involves investigating the BBRv2 bandwidth probing state machine and experimenting with it in order to find solutions, potentially contributing to further versions. It is also possible that these issues are not product of the BBRv2 algorithm itself, but caused by its implementation over QUIC.
- **Investigate CC ramp-up strategies using BDP Frame data.** Even though the experiments in this thesis did not show favorable results for the BDP Frame extension, we believe that it should be further investigated, and that this approach has potential for being a fundamental mechanism for QUIC over SATCOM.
- **Investigate potential dynamic ACK policies**. This thesis has demonstrated the benefits of using custom ACK policies to improve performance under bandwidth-asymmetric links. In this context, we suggest the design of ACK policies that adapt to bandwidth asymmetry and return link congestion dynamically.
- **Proxy-based solutions: MASQUE**. As described in Chapter 3, MASQUE could potentially be useful to implement SATCOM optimizations without breaking E2E encryption and revealing minimal information to the proxies, through cooperation between the endpoints and the network. We believe this will be a trendy study topic in the next few years.
- **More realistic emulation with OpenSAND**. The testbed can be extended to introduce OpenSAND satellite emulation, as indicated in Chapter 3. OpenSAND implements many low layer mechanisms that are present in real satellite transponders and gateways, allowing to study QUIC performance over SATCOM over more realistic scenarios.

List of Figures

List of Tables

Appendix A

User manual for experiment reproduction

This appendix aims to guide anyone with access to the UiS network testbed to reproduce the results described in this work. To reproduce the experiments, three steps need to be followed: (1) check the status of the testbed and perform sanity checks on it, (2) select an scenario to reproduce and run the experiments, (3) perform the post-processing of results.

A.1 Testbed sanity checks

After getting access to the testbed, a series of sanity checks need to be performed. The following list can be used as a check-list of tasks to make sure that the testbed is correctly set up and healthy.

- Check that the controller and all testbed nodes are running.
- Access the controller and perform a series of tasks:
  - Ping the experiment nodes to test that they are reachable.
  - Check that SSH access to the testbed nodes without password is working.
  - Check that experiments nodes are properly time-synchronized.
  - Check that there is Internet connectivity in all the nodes.
- Make sure that the controller is SSH-able remotely through the UiS Intranet.

- Run an example QUIC experiment (e.g., the one located in 'experiments/quic-experiment') and see that everything is going smoothly according to the configuration file. If something is missing, debug according to experiment logs.

A.2 Running an experimental scenario

After the status of the testbed has been validated, select the experimental scenario to be reproduced, and access the corresponding folder from the controller, as shown in Table A.1

Scenario	Folder
A1: Single-Flow Bulk Download	experiments/single-flow
A2: Mice vs Elephant Flows	experiments/multi-file
A3: Multi-Flow Fairness	experiments/bbr2_fairness_multiflow
A4: Latecomer Issue	experiments/bbr2_fairness_latecomers
B1: BDP Extension	experiments/zero-rtt-bdp
C1: Asymmetric SATCOM links	experiments/asymmetric

Table A.1: Guide to find the configuration files for the scenarios described in this work

Once moved to the corresponding folder, check the TEACUP configuration file **config.py**, to get a clear understanding of how the scenario is set up. This configuration file is easily readable and modifiable, and it defines all default values for experiments and parameter sweeps. The main parameters to be modified in these experiments are listed in Table A.2.

Parameter	Description
TPCONF_enable_qlog	'1' to enable *qlog*, '0' to disable it
TPCONF_runs	Number of experiment runs executed
TPCONF_duration	Duration in seconds of one experiment run
TPCONF_delays	One-way delay introduced in the router queue in miliseconds
TPCONF_loss_rates	Packet loss rates artificially introduced in the queue
TPCONF_bandwidths	Bandwidth setups for downstream and upstream
TPCONF_aqms	Queue management strategy (for this work we only use 'bfifo')
TPCONF_buffer_sizes	Buffer size values (if AQM is 'bfifo', in bytes)
TPCONF_ccalgos	CC algorithm for the sender: 'bbr2', 'bbr' or 'cubic'

Table A.2: List of most used parameters in TEACUP configuration files

In order to run an experiment, execute the following instruction in the command-line,

```
$ ./run.sh
```

or to run it in the background,

```
$ screen -d -m ./run.sh
```

and wait for the experiments to conclude. One can keep track of the progress by looking into the 'experiments_started.txt' and 'experiments_completed.txt' files.

Once the experiments are completed, download the qlog files generated under the generated experiment folder (named 'exp_...'). This can be done remotely through SSH using the **scp** tool.

A.3 Post-processing

Once the qlog files are available locally, the Python script for post-processing located in the **visualization** folder need to be used, named 'visualize_qlog.py'. This python script contains a long list of methods to extract different statistics from *qlog* files. Most of these methods make use of the **read_qlog()** method, which takes the path to a compressed *qlog* file, decompresses it using the **gzip** library, and extracts all events using the **json** and **jsonseq** libraries.

Table A.3 specifies which methods have been used to generate the results for each of the scenarios. The end of the Python script can be used to set up processing tasks by calling these methods for the different sets of results, since *qlog* files are heavy and it takes time to analyze them.

Scenario	Folder
A1: Single-Flow Bulk Download	analyze_goodput_bdp_single()
A2: Mice vs Elephant Flows	analyze_download_time()
A3: Multi-Flow Fairness	analyze_jfi_multiflow()
A4: Latecomer Issue	plot_goodput_time()
B1: BDP Extension	analyze_download_time()
C1: Asymmetric SATCOM links	analyze_goodput_bdp_single()

Table A.3: Guide to find the configuration files for the scenarios described in this work

The results obtained can then be pasted into the R visualization scripts for generating the plots and exporting them as PDF files.

Appendix B

QUIC traffic generators for TEACUP

```
#
# picoquic
#

## Start picoquic server
#  @param

def start_picoquic_server(counter='1', file_prefix='', remote_dir='', port='',
                          srv_host='', extra_params='', check='', wait='',
                          kill_delay=0, ccalgo='', kill=''):
    if port == '':
        abort('Must specify port')
    if srv_host == '':
        abort('Must specify server host')

    # start picoquic
    logfile = remote_dir + file_prefix + '_' + \
        env.host_string.replace(':', '_') + '_' + counter + '_picoquic.log'

    picoquic_cmd = '/home/aitor/picoquic/picoquic_sample server'
    picoquic_cmd += ' %s /root/certs/cert.pem' % (port)
    picoquic_cmd += ' /root/certs/key.pem /root/files %s' % (ccalgo)


    if extra_params != '':
        picoquic_cmd += ' ' + extra_params
    wait_time = wait
```

```
    pid = runbg(command=picoquic_cmd, wait=wait_time, out_file=logfile)

    bgproc.register_proc(env.host_string, 'picoquic', counter, pid, logfile, kill_delay)



## Start picoquic client
#  @param

def start_picoquic_client(counter='1', file_prefix='', remote_dir='', port='',
                      srv_host='', extra_params='', check='',
                      wait='', kill_delay=0, download_size='', download_repeat=1):

    if port == '':
        abort('Must specify port')
    if srv_host == '':
        abort('Must specify server host')

    # start picoquic
    logfile = remote_dir + file_prefix + '_' + \
        env.host_string.replace(':', '_') +'_' + counter + '_picoquic.log'

    picoquic_cmd = '/home/aitor/picoquic/picoquic_sample client %s %s' %(srv_host, port)
    if download_size != '':
        picoquic_cmd += ' /tmp file'+download_size

    # Multi-file download
    if download_repeat > 1:
        for i in range(download_repeat):
            picoquic_cmd += ' file'+download_size

    if extra_params != '':
        picoquic_cmd += ' ' + extra_params

    wait_time = wait
    pid = runbg(command=picoquic_cmd, wait=wait_time, out_file=logfile)

    bgproc.register_proc(env.host_string, 'picoquic', counter, pid, logfile, kill_delay)


## Start picoquic sender and receiver
## For parameters see start_picoquic_client() and start_picoquic_server()
def start_picoquic(counter='1', file_prefix='', remote_dir='', local_dir='',
               port='', client='', server='', ccalgo='', download_size='',
```

```
                download_repeat=1, extra_params_client='', extra_params_server='',
                check='', wait='', kill_delay=0):
    "Start picoquic traffic sender and receiver"

    server, server_internal = get_address_pair(server)
    client, dummy = get_address_pair(client)
    execute(start_picoquic_server, counter, file_prefix, remote_dir, port,
            server_internal, extra_params_server, check, wait, kill_delay,
            ccalgo, hosts=[server])
    execute(start_picoquic_client, counter, file_prefix, remote_dir, port,
            server_internal, extra_params_client, check, wait, kill_delay,
            download_size, download_repeat, hosts=[client])



#
# ngtcp2
#

## Start ngtcp2 server
#  @param

def start_ngtcp2_server(counter='1', file_prefix='', remote_dir='', port='',
                        srv_host='', extra_params='', check='', wait='', kill_delay=0,
                        ccalgo=''):
    if port == '':
        abort('Must specify port')
    if srv_host == '':
        abort('Must specify server host')

    # Implement check here

    # kill previous idle processes

    #run('pkill -f \'examples/server\'')

    # start ngtcp2
    logfile = remote_dir + file_prefix + '_' + \
        env.host_string.replace(':', '_') + '_' + counter + '_ngtcp2.log'
    ngtcp2_cmd = '/home/aitor/ngtcp2/examples/server --quiet'

    if config.TPCONF_enable_qlog =='1':
        ngtcp2_cmd += ' --qlog-dir=/root/qlog'
```

```
    if extra_params != '':
        ngtcp2_cmd += ' ' + extra_params

    if ccalgo != '':
        ngtcp2_cmd += ' --cc %s' %(ccalgo)

    # Uncap flow control windows
    ngtcp2_cmd += ' --max-data 33554432 --max-stream-data-bidi-local 33554432'
    ngtcp2_cmd += ' --max-stream-data-bidi-remote 33554432 --max-stream-data-uni 33554432'
    ngtcp2_cmd += ' --max-window 0 --max-stream-window 0'

    ngtcp2_cmd +=  ' %s %s /root/certs/key.pem /root/certs/cert.pem' %(srv_host, port)
    wait_time = wait
    pid = runbg(command=ngtcp2_cmd, wait=wait_time, out_file=logfile)

    bgproc.register_proc(env.host_string, 'ngtcp2', counter, pid, logfile, kill_delay)



## Start ngtcp2 client
#  @param

def start_ngtcp2_client(counter='1', file_prefix='', remote_dir='', port='',
                        srv_host='', extra_params='', check='',
                        wait='', kill_delay=0, ccalgo='', download_size='',
                        download_repeat=1, zerortt=0):

    if port == '':
        abort('Must specify port')
    if srv_host == '':
        abort('Must specify server host')
    # Implement check here

    # kill previous idle processes

    #run('pkill -f \'examples/client\'')

    # start ngtcp2
    logfile = remote_dir + file_prefix + '_' + \
        env.host_string.replace(':', '_')+ '_' + counter + '_ngtcp2.log'
    uri = ''
    if download_size != '':
        uri = 'file://root/files/file'+download_size
        # Multi-file download
```

```
        if download_repeat > 1:
            for i in range(download_repeat-1):
                uri += ' file://root/files/file'+download_size



    ngtcp2_cmd = '/home/aitor/ngtcp2/examples/client --quiet
    ngtcp2_cmd += ' --tp-file=/root/tp-file --download=/root/downloads'
    if zerortt == 1:
        ngtcp2_cmd += ' --session-file=/root/session-file'


    if config.TPCONF_enable_qlog =='1':
        ngtcp2_cmd += ' --qlog-dir=/root/qlog'
    if extra_params != '':
        ngtcp2_cmd += ' ' + extra_params

    if ccalgo != '':
        ngtcp2_cmd += ' --cc %s' %(ccalgo)

    # Uncap flow control windows
    ngtcp2_cmd += ' --max-data 33554432 --max-stream-data-bidi-local 33554432'
    ngtcp2_cmd += ' --max-stream-data-bidi-remote 33554432 --max-stream-data-uni 33554432'
    ngtcp2_cmd += ' --max-window 0 --max-stream-window 0'

    ngtcp2_cmd += ' %s %s %s' %(srv_host, port, uri)

    wait_time = wait
    pid = runbg(command=ngtcp2_cmd, wait=wait_time, out_file=logfile)

    bgproc.register_proc(env.host_string, 'ngtcp2', counter, pid, logfile, kill_delay)


## Start ngtcp2 sender and receiver
## For parameters see start_ngtcp2_client() and start_ngtcp2_server()
def start_ngtcp2(counter='1', file_prefix='', remote_dir='', local_dir='',
                 port='', client='', server='', extra_params_client='',
                 extra_params_server='', ccalgo='', download_size='',
                 download_repeat=1, zerortt=0, check='', wait='', kill_delay=0):
    "Start ngtcp2 traffic sender and receiver"

    server, server_internal = get_address_pair(server)
    client, dummy = get_address_pair(client)
    execute(start_ngtcp2_server, counter, file_prefix, remote_dir, port,
```

```
            server_internal, extra_params_server, check, wait,
            kill_delay, ccalgo, hosts=[server])
    execute(start_ngtcp2_client, counter, file_prefix, remote_dir, port,
            server_internal, extra_params_client, check, wait,
            kill_delay, ccalgo, download_size, download_repeat, zerortt, hosts=[client])
```

Appendix C

QUIC loggers for TEACUP

```
def clean_qlog():

    # Remove qlog files from previous experiments

    run('rm -r /root/qlog')
    run('mkdir /root/qlog')


def get_qlog(path=''):

    # Get the name of generated qlog file(s)
    output = run('ls /root/qlog')
    files = output.split()
    counter = 1
    role = 'client'
    if(env.host_string.endswith('11-1') or env.host_string.endswith('11-2')):
        role = 'server'

    if(env.host_string.endswith('10-1') or env.host_string.endswith('11-1')):
        counter = 1
    else:
        counter = 2

    for file in files:
        output_file = role+str(counter)
        if(file.endswith('.sqlog')):
            # ngtcp2
            run('mv /root/qlog/' + file + ' /root/qlog/' + output_file  + '.sqlog')
            # Compress
```

```
        run('gzip -f /root/qlog/' + output_file + '.sqlog')
        # Get file
        get('/root/qlog/' + output_file  + '.sqlog.gz', path)
    else:
        # picoquic

        if(file.endswith('.log')):
            # Convert binary to if it's a binary
            run('/home/aitor/picoquic/picolog_t -f qlog -o /root/qlog /root/qlog/'+file,7
             warn_only=True)
            # Remove binary file
            run('rm /root/qlog/'+file)
            splitted = file.split('.')
            file = splitted[0]+'.qlog'

        # change name
        run('mv /root/qlog/' + file + ' /root/qlog/' + output_file + '.qlog')
        # Compress
        run('gzip -f /root/qlog/' + output_file + '.qlog')
        # Get file
        get("/root/qlog/"+ output_file + '.qlog.gz', path)

    counter += 2
```

Appendix D

Installation scripts for QUIC implementations

This appendix contains the listings of the bash scripts that automate the process of installing the QUIC implementations used in this work. These are intended to accelerate the process of setting up the testbed if it needs to be re-installed.

```
#########################
# ngtcp2 Installation #
#########################

export home_dir="/home/aitor"

# Set library path

export LD_LIBRARY_PATH="/usr/local/lib:/usr/local/lib64"

# Set it automatically on startup

cp files/libraries.sh /etc/profile.d/

# Install dependencies

zypper install -y pkg-config autoconf automake make cmake libtool
zypper install cunit-devel libev-devel gcc-c++ gcc8 gcc8-c++
zypper refresh
zypper install -y autotools-dev

# Install OpenSSL

cd $home_dir
git clone --depth 1 -b OpenSSL_1_1_1m+quic https://github.com/quictls/openssl
cd openssl
./config enable-tls1_3 --prefix=$PWD/build
make -j$(nproc)
make install_sw
cd ..

# Install nghttp3

git clone https://github.com/ngtcp2/nghttp3
cd nghttp3

cmake .
make all
make install

autoreconf -i
./configure --prefix=$PWD/build --enable-lib-only
make -j$(nproc) check
make install

cd ..

# Install ngtcp2

git clone https://github.com/ngtcp2/ngtcp2
cd ngtcp2

cmake .
make all
make install
autoreconf -i

# Gotta compile with gcc 8 or higher, otherwise it does not work

./configure PKG_CONFIG_PATH=$PWD/../openssl/build/lib/pkgconfig:$PWD/../nghttp3/build/lib/pkgconfig
make -j$(nproc) check
```

```
#########################
# picoquic Installation #
#########################

# Move to home directory
cd /home/aitor

# Install required packages
zypper install -y cmake openss-devel

# Clone and build picotls
git clone https://github.com/h2o/picotls.git
cd picotls
git submodule init
git submodule update
cmake .
make
make check

# Clone and build picoquic
cd /home/aitor
git clone https://github.com/private-octopus/picoquic
cd picoquic
cmake .
make

# Copy 'picoquicdemo' to path
cp picoquicdemo /usr/bin

# Copy certificate folder to root directory
cp -r certs /root
```

Appendix E

Accepted publication for ANRW'22

Some of the results obtained in this thesis have derived into an accepted publication on the **Applied Networking Research Workshop (ANRW) 2022** [112], which takes place on July 23-29 at Philadelphia (USA), along with the IETF-114 meeting. This academic workshop, sponsored by ACM SIGCOMM and the Internet Research Task Force (IRTF), encourages researchers to present their work on applied networking research with new Internet protocols.

This paper titled 'On the Suitability of BBR Congestion Control for QUIC over GEO SATCOM Networks', includes some partial results of Block A: Better Congestion Control.

The following pages contain a pre-print of the paper.

On the Suitability of BBR Congestion Control for QUIC over GEO SATCOM Networks

Aitor Martin
Department of Electrical
Engineering & Computer Science
University of Stavanger, Norway
a.martin@stud.uis.no

Naeem Khademi
Department of Electrical
Engineering & Computer Science
University of Stavanger, Norway
naeem.khademi@uis.no

ABSTRACT

Satellite broadband connectivity has received significant level of interest in recent years, partly due to the emergence of 5th and 6th generations of cellular networks and their novel use-cases. High-throughput GEO satellites are therefore expected to become an integral part of such networks both for access and backhauling. Almost simultaneously, major breakthroughs have occurred within the Internet transport layer and its congestion control mechanisms – i.e., with the increasing deployment of user-space QUIC protocol and BBR congestion control. The impact of these innovations and their overall performance on the Internet paths traversing over satellite links is yet to be investigated. Although traditionally TCP-splitting methods with Performance-Enhancing-Proxies (PEPs) were used to boost the transport performance over the satellite links, such approaches are proven to be extremely hard if not impossible for QUIC due to its encrypted nature. This leads to QUIC's poor performance over satellite links, which is currently being investigated by the IETF's QUIC WG. In addition, the transport performance depends on the choice of congestion control and QUIC implementation. In this work we will explore these aspects and the suitability of BBR congestion control for QUIC over SATCOM networks through real-life experimentation in an emulated testbed environment.

CCS CONCEPTS

• **Networks** → **Transport protocols**; **Network experimentation**.

ANRW '22, July 23-29, 2022, Philadelphia, USA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

KEYWORDS

BBRv2, congestion control, satellite, QUIC

ACM Reference Format:
Aitor Martin and Naeem Khademi. 2022. On the Suitability of BBR Congestion Control for QUIC over GEO SATCOM Networks . In *Proceedings of Applied Networking Research Workshop (ANRW '22).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Although they have been instrumental for delivering continuous broadband services to remote locations over vast areas of the globe for several decades, geosynchronous (GEO) satellite networks have just begun to receive significant interest in recent years. This is both due to the emergence of high-throughput satellites (HTS) and 5th/6th generation of cellular networks (5G/6G) and their novel use-cases. In such networks, broadband services can either be delivered by the ISPs to the end-user either directly or through a SATCOM-enabled cellular network in locations where terrestrial connectivity is limited or non-existent.

Simultaneously, recent years have witnessed major innovations within the Internet transport layer. This has mainly been focused on two areas: (1) de-ossifying transport protocol stack by deploying QUIC – a user-space transport protocol with a UDP substrate – to traverse the misbehaving middleboxes which made the transport evolution impossible [1]; (2) improving congestion control (CC) mechanisms to better utilize the available bandwidth while maintaining low latency through deployment of Bottleneck Bandwidth and Round-trip propagation time (BBR) congestion control [2]. Initially designed by Google and currently under standardization at the IETF, QUIC is a UDP-based, reliable, multiplexed and fully-encrypted transport protocol, aiming to solve many of TCP's shortcomings [3][4][5]. Although the IETF's QUIC standard specifies a CC similar to the TCP NewReno [5], different implementations of QUIC offer support for BBR CC. BBR's objective is to adapt the congestion window (cwnd) according to the actual level of congestion on the path by taking into consideration the increase in Round-Trip Time (RTT) and maximum achievable bandwidth – i.e., by mostly

maintaining cwnd around an optimal value above which an increase in cwnd ceases to yield more bandwith yet incurs more latency [2]. Initial investigation of BBR's performance over TCP and its related coexistence issues with loss-based CCs [6] has lead to the introduction of BBRv2 [7]. Unlike BBR, which was deemed to be *too aggressive* due to lack of reaction to packet loss and thus penalizing competing loss-based CCs, BBRv2 aims to score a better level of fairness by reacting to loss beyond a certain threshold.

Preliminary investigations of web performance with BBR-enabled QUIC over satellite links show that BBR can outperform the loss-based CUBIC in presence of packet loss [8]. While the work in [8] focuses on small and medium size web objects, a more thorough and general investigation of these CCs over satellite links particularly with the addition of BBRv2 is yet to be performed. Satellite links are shown to pose significant challenges to the transport protocols and congestion control algorithms [9] due to their high bandwidth-delay product (BDP) and likely asymmetric nature. While these issues were traditionally mitigated through the use of connection splitting methods and Performance Enhancing Proxies (PEPs) for TCP traffic [10], this has been proven to be very challenging for QUIC due to its end-to-end encrypted nature limiting the level of optimization methods that can be implemented on the middleboxes (e.g., SATCOM gateways). Recent discussions at the IETF aiming to address this issue involve several potential approaches: (a) improving end-to-end QUIC performance tailored for SATCOM environments, e.g., providing signaling of essential path characteristics (e.g., BDP) between the QUIC end-points [11]; and (b) voluntarily exposing certain header fields in QUIC to the middleboxes [12].

This paper aims to thoroughly explore the suitability of BBR and BBRv2 for QUIC over satellite broadband links, which are yet to be fully investigated. The contributions of this paper are four-fold as follows:

(1) to the best of our knowledge, it is the first work to investigate the performance of QUIC with BBRv2 in satellite environments using real-life experiments in an emulated network testbed.
(2) it evaluates several important aspects of CC performance over long-haul satellite links such as inter- and intra-protocol fairness, latercomer fairness issues, and mice versus elephant flows.
(3) investigates the impact of QUIC implementation on the transport performance over satellite links by employing two commonly used QUIC implementations.
(4) elaborates on the suitability of BBR for QUIC over satellite links, current issues and potential solutions.

The remainder of this paper is structured as follows: Section 2 outlines the related works on QUIC over satellite networks and BBR congestion control; Section 3 presents in detail the network testbed setup used for our real-life evaluations; Section 4 presents our evaluation scenarios and their experimental results; Section 5 provides a discussion based on our evaluation results on the suitability of BBR for QUIC over satellite links, its implications and potential solutions; and finally, Section 6 concludes the paper.

2 BACKGROUND

This section offers a context for our work by providing background on common transport layer issues over satellite links both for TCP and QUIC, and existing and proposed solutions to mitigate such issues. In addition, it presents prior works on the evaluation of BBR CC particularly over satellite links and also investigates the works performed on the performance of different QUIC implementations.

2.1 Transport Layer over Satellite

The presence of a GEO satellite link along the network path introduces several challenges for transport layer mechanisms [13], often resulting in underutilized links. **Firstly**, as a consequence of large delays introduced by the signal propagation to GEO, the path's RTT increases significantly – e.g., to an order of 0.5 sec. This increases the transport's feedback loop, leading to the slow *cwnd* growth during both TCP slow-start and congestion avoidance phases hence leading to poor link utilization. A high RTT thus implies a higher BDP, requiring buffers and windows in both the endpoints and the satellite transponder to be large, to be able to handle the large amounts of in-flight unacknowledged data [13]. **Secondly**, satellite links can introduce higher bit error rate than terrestrial paths. This can be problematic for loss-based CCs such as NewReno or CUBIC which take packet loss as an indication of network congestion. **Thirdly**, satellite links can be asymmetrical, which can result in the upstream buffers becoming filled with ACKs. This can reduce downstream performance, since it adds extra delay to protocol feedback and triggers unnecessary retransmissions [14].

These challenges can be confronted by optimizing TCP mechanisms for satellite environments – for instance, satellite-optimized CCs (e.g., TCP Hybla [15]), window scaling solutions (e.g., the TCP window scale option [16]) or different ACK handling strategies. However, most satellite service providers have relied on PEPs to mitigate the issues of TCP over satellite [10].

Although PEP solutions can be diverse in terms of both topology and performance-boosting mechanisms, most of them rely on connection splitting. Split-TCP connections allow optimizations such as local loss recovery, ACK segment spoofing to accelerate the TCP initial handshake [10], or the use of satellite-optimized CC only for the satellite segment. Open-source PEP implementations such as PEPsal [17] have been fundamental for the research of TCP solutions over

satellite. Other solutions like QPEP [18] propose a distributed PEP topology – i.e., with one proxy at each end of the satellite link – to tunnel TCP connections using QUIC.

Nevertheless, more challenges arise QUIC traffic is introduced to the satellite networks [9]. Given the current trend with QUIC's deployment on the Internet (7.9% of all websites are already using QUIC [19]), the wide use of QUIC over satellite links is to be expected. Since QUIC advocates for complete end-to-end confidentiality with full header encryption, these PEP solutions become unfeasible, unless there is some cooperation mechanism between the endpoints and the network or some header information exposed. The inability to use PEPs with QUIC has been shown to dramatically degrade performance in several studies [20, 21, 22].

Since then, efforts have been made at IETF's QUIC WG to study newly proposed experimental QUIC features that could boost its end-to-end performance over satellite links [9]. These include: (1) the BDP Frame extension [11], which suggests a mechanism that allows endpoints to remember path characteristic parameters (i.e., the RTT and bottleneck bandwidth) from previous connections and reuse them to accelerate the startup of following connections to the same server; (2) the ACK Frequency extension [23], which allows receivers to modify the ACK sending rate and avoid introducing congestion in the upstream link; and (3) the use of Forward Error Correction (FEC) to mitigate the effect of transmission errors in the satellite link [24].

2.2 BBRv1 and BBRv2

Traditional TCP CCs such as CUBIC or NewReno use packet loss as a sign of network congestion, which can lead to poor link utilization on lossy network paths. On the contrary, BBR aims to improve link utilization by searching for the optimal *cwnd* value that maximizes the bandwidth achieved while trying to minimize RTT.

The BBR algorithm can therefore be summarized into four main phases: (1) the STARTUP phase, which increases the *cwnd* exponentially to quickly fill the bottleneck queue and measures how much bandwidth is available on the network path; (2) the DRAIN phase, in which the bottleneck queue is drained to remove the congestion introduced during STARTUP phase; (3) the PROBE-BW phase, in which the algorithm cycles through different pacing rate values, aiming to continuously search the maximum available bandwidth, but also regularly draining the queue in order to not dominate over other competing flows; (4) the PROBE-RTT, which allows the sender to re-measure the minimum RTT value within regular intervals.

Some early studies have shown that the first version of BBR [2] is able to significantly outperform CUBIC in many scenarios, especially when packet loss becomes significant [25]. However, later studies identify that BBR can be unfair towards loss-based CC, as a consequence of the aggressive STARTUP and PROBE-BW phases [6, 26, 27]. These works have also identified RTT unfairness with BBR – i.e., parallel flows with different minimum RTTs do not share the available bandwidth fairly.

These issues have led to an update to the algorithm, first proposed in 2019 - and referred to as BBRv2 hereafter - that aims to find a balance between BBR and loss-based CCs, to reach high link utilization yet behaving less aggressively when sharing the link with CCs such as CUBIC, through the use a more complex probing algorithm [7], which we will not fully explain here for the sake of brevity. While BBRv1 did not react to packet loss at all, BBRv2 uses packet loss and Explicit Congestion Notification (ECN) signaling as inputs for the probing mechanism. BBRv2 also reduces the *cwnd* decrease rate in the PROBE-RTT phase, for a less aggressive throughput fluctuation. Several works have already evaluated BBRv2 for TCP over terrestrial networks [28, 29, 30, 31]. Authors in [28] show that BBRv2 significantly improves fairness when in competition with loss-based CC. Additionally, results in [29] and [30] provide in-depth analysis of intra-protocol and inter-protocol fairness, and point out convergence problems between BBRv2 flows when using large bottleneck buffers. Researchers in [31] also identified issues regarding BBRv2's capability of adapting to changing network conditions - e.g., bandwidth dynamics or random losses - and suggest new mechanisms to alleviate them.

Even though BBRv2 has been deeply evaluated experimentally over terrestrial networks and with TCP, none of these studies: (1) investigate its performance, inter- and intra-protocol fairness when traversing over high-BDP links (e.g., satellite links); and (2) investigate BBRv2 performance over QUIC. This paper investigates these two previously unexplored areas in detail.

2.3 QUIC implementations

Currently, there are several open-source IETF QUIC implementations available for experimentation. However, even though they are based on the same reference specifications, these implementations are highly heterogeneous [32]. Thus, QUIC performance evaluation results can be highly dependent on the particular implementation. As researchers in [33] show through the use simulated and real satellite links, some implementations can perform weekly when used as a client, as a server or even both - e.g., their results show the *ngtcp2* implementation performing poorly as a client, but better as a server.

3 EXPERIMENTAL TESTBED SETUP

Our approach in this work is based on real-life experimentation in an emulated network environment. We make use of a physical network testbed located at the University of

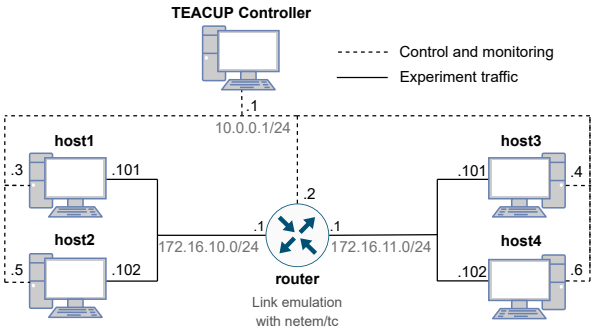


Figure 1: UiS TEACUP network testbed topology

CPU	4 x Intel(R) Core i5-3470 @ 3.20GHz
RAM	16 GB Micron DDR3 1600 MHz
OS	OpenSUSE Leap 15.1 Linux (Kernel 5.4.0)
HDD	500 GB (100 GB for OS and Swap, 400 GB for /home)

Table 1: UiS TEACUP testbed hardware specification

Stavanger (UiS) with a dumbbell topology as shown in Figure 1. The testbed contains two pairs of hosts that act as endpoints: hosts 1 and 2 in network act as clients and hosts 3 and 4 in network act as servers, Another machine acts as a *router* between the two networks, and performs satellite link emulation.

All hosts and the *router* in the networks are commodity off-the-shelf products and have identical hardware specifications as presented in Table 1. Each endpoint has two Ethernet interfaces, one for the control network traffic and another for the experimental traffic. The *router* has an extra Ethernet interface, to perform routing between the two experiment networks. The nodes are interconnected using a Gigabit Ethernet switch.

3.1 Experiment Orchestration with TEACUP

TCP Experiment Automation Controlled Using Python (i.e., TEACUP) developed by CAIA [34], is designed to orchestrate and run TCP experiments on emulated network testbeds. It is based on a set of Python scripts that allow for easy experiment design and automation. The scripts are run from a *controller* node running FreeBSD, which uses the *Fabric* Python library to control the experiment nodes remotely and extract statistics from them. The TEACUP architecture separates the control network from the experiment network to isolate the experimental traffic from control traffic. For this work, we have extended the TEACUP to run experiments and extract statistics with multiple QUIC implementations.

3.2 Link emulation with *netem/tc*

To perform our evaluations at the transport layer level, we emulate a network path with typical characteristics of a satellite link using *netem/tc* in the *router*. Table 2 shows the parameter values used for two scenarios: the satellite scenario (SAT) and the terrestrial scenario (TERR).

	SAT	TERR
One Way Delay (OWD)	300 ms	50 ms
Bottleneck Bandwidth	20 Mbps	20 Mbps
Bottleneck Buffer Size	0.25|0.5|1.0|2.0 x BDP	

Table 2: Network path emulation parameters

	ngtcp2	picoquic
Initial Window	1 MB	1 MB
Max ACK Delay	25	10
Available CC	BBRv2/v1, Cubic	BBRv1, Cubic
HTTP/3	Yes	Yes

Table 3: Default settings and parameters for QUIC implementations

3.3 QUIC Traffic generation and logging

In our evaluations, we make use of two major QUIC implementations: *ngtcp2* [35] and *picoquic* [36]. Table 3 shows the default parameters and available features for both implementations. Since QUIC runs in user-space, no kernel tool was needed to extract transport layer statistics. We therefore used *qlog*, an easily structured, human-readable and standardized event-logging format for QUIC [37] that is offered by most QUIC implementations.

4 EXPERIMENTS AND RESULTS

This chapter describes a series of scenarios designed to evaluate different aspects of CC performance, along with their corresponding results. Experiments were run varying different parameters: the bottleneck buffer size, the number of flows, object sizes and the number of objects. Experiments are repeated with all the available CC algorithms available in QUIC implementations. All experiments are repeated 10 times, and run over the SAT link scenario unless specified otherwise.

4.1 Bulk Download Performance

In order to test the bulk download performance, the client start the download of a 1GB file stored in the server, the download is terminated after 2 minutes and the average goodput is measured. Figure 2 shows the average goodput obtained in this scenario using different QUIC implementations and CC algorithms, and setting the bottleneck buffer size to different fractions of the BDP, for both the satellite and terrestrial scenarios.

Results show that, as expected, link utilization is worse in the satellite scenario. It can be seen that *picoquic* performs better, especially in the satellite scenario. Irrespective of the buffer size, BBRv1 provides a slightly higher goodput than

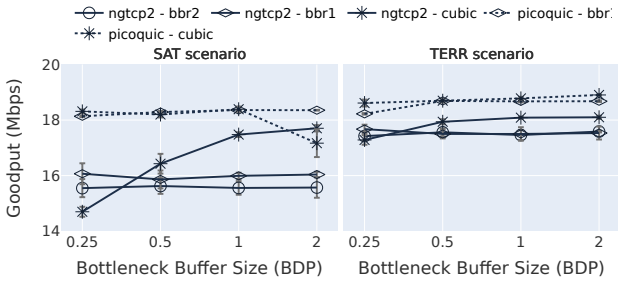


Figure 2: Bulk download average goodput with different implementations and CC. Left: SAT; Right: TERR

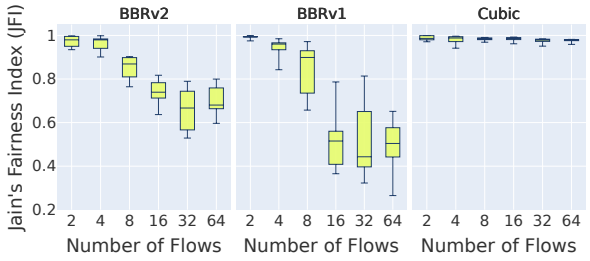


Figure 3: JFI for multi-flow intra-protocol fairness tests with ngtcp2

BBRv2. However, CUBIC has an advantage in comparison with both BBR versions for high buffer sizes.

4.2 Intra- and Inter-Protocol Fairness

When studying CC, it is essential to evaluate the level of fairness between multiple parallel flows. This scenario launches a number of simultaneous long downloads, and terminates them after 5 minutes. In order to measure fairness between parallel flows, we use Jain's Fairness Index (JFI) [38]. Due to the abscence of a BBRv2 implementation in *picoquic*, we evaluate fairness using only *ngtcp2*.

First, intra-protocol fairness (i.e., fairness between same CC flows) is evaluated. Figure 3 shows the obtained JFI for different numbers of parallel flows, for a bottleneck buffer size to 1 BDP. We omitted presenting fairness results with different buffer sizes, since they did not appear to have a significant impact in our results.

Results in Figure 3 show that BBRv1 achieves the best fairness score for the 2-flow scenario, never going below a JFI value of 0.970 and staying very close to 1. As the number of flows increases, while CUBIC is achieving outstanding fairness levels even with 64 flows, with a JFI value of 0.977 ± 0.006, both BBR versions show a very significant decrease in the fairness score, reaching JFI values below 0.4 in the case of BBRv1. Despite these fairness issues, we observed an improvement in link utilization for higher number of flows for all CC algorithms.

To evaluate inter-protocol fairness (i.e.- fairness between competing flows with different CC) three different scenarios have been defined for different CC combinations: (A) BBRv2 vs BBRv1, (B) BBRv2 vs CUBIC and (C) BBRv1 vs CUBIC. Figure 4 shows the obtained JFI for the different scenarios with 2 and 4 flows, and all possible distributions. It reveals that BBRv2 behaves very fairly towards CUBIC in all configurations, in contrast to the low JFI values in scenario C (BBRv1 vs CUBIC). We also see fairness issues between BBRv1 and BBRv2; however, this may not be too critical since BBRv2 is intended to replace BBRv1.

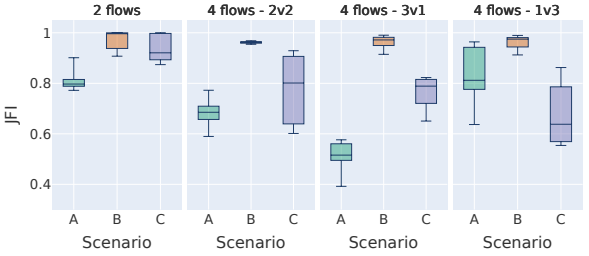


Figure 4: JFI for multi-flow inter-protocol fairness tests with ngtcp2

4.3 Latecomer Issue

In the context of CC fairness, the latecomer issue is another common problem flows, which implies latecomer flows either (1) being penalized by previously started flows, and thus not receiving a fair bandwidth share, or (2) penalizing existing flows. To evaluate this issue, a scenario was designed with 4 flows start at 0, 40, 80 and 120 seconds respectively, all of them lasting for 180 seconds.

Figures 5, 6 and 7 show the goodput achieved with each flow over time, as well as the aggregate goodput and the smoothed RTTs, for BBRv2, BBRv1 and CUBIC respectively. While results with CUBIC show latecomer flows underperforming, they show good convergence and fairness in the long term coexistence. BBRv1 shows the most aggressive performance: latecomer flows gain a great share of available bandwidth faster, and they overtake previous flows; meanwhile, BBRv2 shows a less aggressive behaviour. BBRv2 results also show that the remaining Flow number 4 fails to recover the available bandwidth in the last tens of seconds, which might be a problem in the *ngtcp2* BBRv2 implementation. Looking at the smoothed RTT over time for both BBR versions, we observe the expected behavior with the queue drains and the changes in the PROBE-RTT phase, which makes the RTT fluctuate more smoothly in the BBRv2 update.

4.4 Mice versus Elephant Flows

Since satellites are widely used for Internet access, it is also relevant to study the download time of small objects, which lead to the called 'mice flows'. Figure 8 shows the download time for different object sizes and numbers of objects,

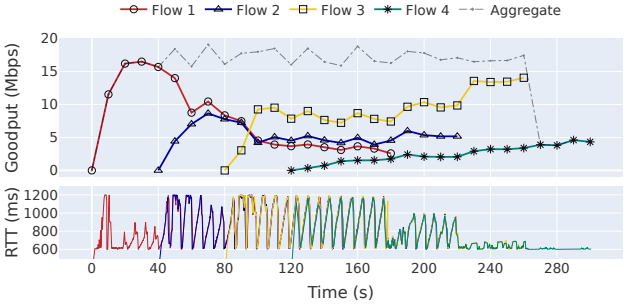


Figure 5: Latecomer fairness with 4 BBRv2 flows

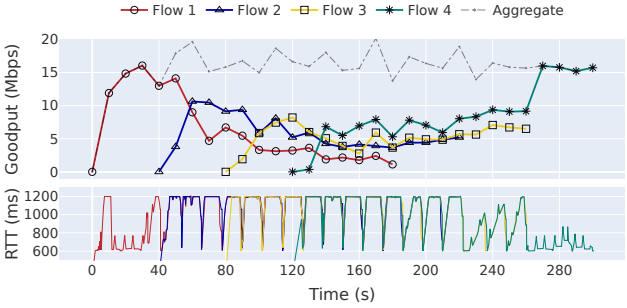


Figure 6: Latecomer fairness with 4 BBRv1 flows

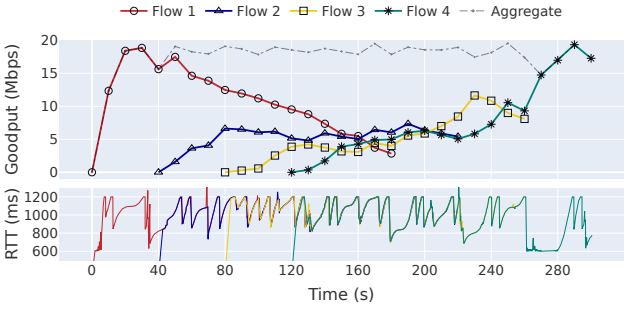


Figure 7: Latecomer fairness with 4 CUBIC flows

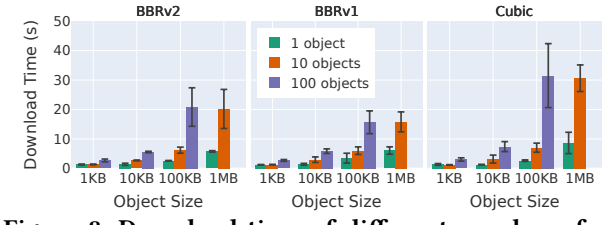


Figure 8: Download time of different number of objects, for various object sizes

with *ngtcp2*. These experiments are run with the presence of background BBR traffic, to emulate a more realistic scenario. The 100x1MB case was omitted for the sake of better visualization. Results show an exponential relationship between the download time and the amount of objects downloaded, which is a consequence of the exponential increase of the cwnd in the CC startup. It is also noticeable that download times are higher with CUBIC, especially for larger object sizes and amounts of objects - e.g., for the 100x100KB case, download time with CUBIC is around double the amount obtained with BBRv1.

5 DISCUSSION

Our study has revealed that heterogeneity in the different QUIC implementations affects performance on different levels, since we clearly see *picoquic* providing better utilization over satellite links. We believe that this is consequence of some features introduced in *picoquic* by default that make clients and servers that recognize high BDP links, and adapt the QUIC sockets to the needs of satellite links (e.g., delaying ACKs in clients, or allowing flow control windows to increase higher than usual). We also believe that the lack of Path MTU Discovery in *ngtcp2* is having an impact on the results. These solutions are expected to become instrumental in the context of encrypted transport over satellite.

Multi-flow fairness experiments also reveal some important matters. Firstly, we see very high unfairness between competing BBR flows over satellite, especially when the number of flows grows, which could be especially problematic for satellite backhauling scenarios. Secondly, we can observe that the newest BBR version solves the previous fairness issues against CUBIC, which is fundamental since BBR traffic will have coexist with CUBIC traffic in the Internet.

When looking at the mice flow results, it becomes clear that BBRv1 and BBRv2 outperform CUBIC when downloading a large number of files. This stresses the relevance of using BBR to minimize user-perceived latency in the satellite scenario, which could be key not only for web browsing, but also for interactive applications.

Nevertheless, being unable to use PEP optimizations still leaves QUIC in great disadvantage in comparison to TCP, especially for mice flow traffic. This raises some interest to evaluate QUIC application proxy solutions, such as Multiplexed Application Substrate over QUIC Encryption (MASQUE) [12], which could introduce performance-enhancing solutions.

6 CONCLUSION

This work has studied the suitability of BBR congestion control over emulated satellite networks and using different QUIC implementations. Our results suggest BBR can significantly boost mice flow performance and that BBRv2 solves fairness issues against CUBIC. However, we could spot a series of drawbacks: (1) CUBIC is performing better than BBR in many cases; (2) both BBR versions still show clear intra- protocol fairness issues and (3) the latecomer issue is still present.

Future work will look at more realistic web browsing scenarios, where the download of complex websites will be emulated in our testbed, following their tree-shaped structure.

REFERENCES

[1] Giorgos Papastergiou, Gorry Fairhurst, David Ros, Anna Brunstrom, Karl-Johan Grinnemo, Per Hurtig, Naeem Khademi, Michael Tüxen, Michael Welzl, Dragana Damjanovic, and Simone Mangiante. 2017. De-ossifying the internet transport layer: a survey and future perspectives. *IEEE Communications Surveys Tutorials*, 19, 1, 619–639. DOI: 10.1109/COMST.2016.2626780.

[2] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. Bbr: congestion-based congestion control. *ACM Queue*, 14, September-October, 20 –53. http://queue.acm.org/detail.cfm?id=3022184.

[3] Jana Iyengar and Martin Thomson. 2021. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000. (May 2021). DOI: 10.17487/RFC9000. https://www.rfc-editor.org/info/rfc9000.

[4] Martin Thomson and Sean Turner. 2021. Using TLS to Secure QUIC. RFC 9001. (May 2021). DOI: 10.17487/RFC9001. https://www.rfc-editor.org/info/rfc9001.

[5] Jana Iyengar and Ian Swett. 2021. QUIC Loss Detection and Congestion Control. RFC 9002. (May 2021). DOI: 10.17487/RFC9002. https://www.rfc-editor.org/info/rfc9002.

[6] Mario Hock, Roland Bless, and Martina Zitterbart. 2017. Experimental evaluation of BBR congestion control. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. IEEE, Toronto, ON, (October 2017), 1–10. ISBN: 978-1-5090-6501-1. DOI: 10.1109/ICNP.2017.8117540. Retrieved 04/04/2022 from http://ieeexplore.ieee.org/document/8117540/.

[7] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Ian Swett, and Van Jacobson. 2022. BBR Congestion Control. Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-02. Work in Progress. Internet Engineering Task Force, (March 2022). 66 pages. https://datatracker.ietf.org/doc/html/draft-cardwell-iccrg-bbr-congestion-control-02.

[8] Yue Wang, Kanglian Zhao, Wenfeng Li, Juan Fraire, Zhili Sun, and Yuan Fang. 2018. Performance Evaluation of QUIC with BBR in Satellite Internet. In *2018 6th IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*. ISSN: 2380-7636. (December 2018), 195–199. DOI: 10.1109/WiSEE.2018.8637347.

[9] Tom Jones, Gorry Fairhurst, Nicolas Kuhn, John Border, and Stephan Emile. 2021. Enhancing Transport Protocols over Satellite Networks. Internet-Draft draft-jones-tsvwg-transport-for-satellite-02. Work in Progress. Internet Engineering Task Force, (October 2021). 29 pages. https://datatracker.ietf.org/doc/html/draft-jones-tsvwg-transport-for-satellite-02.

[10] Jim Griner, John Border, Markku Kojo, Zach D. Shelby, and Gabriel Montenegro. 2001. Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations. RFC 3135. (June 2001). DOI: 10.17487/RFC3135. https://www.rfc-editor.org/info/rfc3135.

[11] Nicolas Kuhn, Francklin Simo, David Pradas, and Emile Stephan. 2021. Evaluating BDP FRAME extension for QUIC. en. *arXiv:2112.05450 [cs]*, (December 2021). arXiv: 2112.05450. Retrieved 01/18/2022 from http://arxiv.org/abs/2112.05450.

[12] Tommy Pauly and David Schinazi. 2022. QUIC-Aware Proxying Using HTTP. Internet-Draft draft-pauly-masque-quic-proxy-03. Work in Progress. Internet Engineering Task Force, (March 2022). 19 pages. https://datatracker.ietf.org/doc/html/draft-pauly-masque-quic-proxy-03.

[13] Luis A. Sanchez, Mark Allman, and Dr. Dan Glover. 1999. Enhancing TCP Over Satellite Channels using Standard Mechanisms. RFC 2488. (January 1999). DOI: 10.17487/RFC2488. https://www.rfc-editor.org/info/rfc2488.

[14] S Oueslati-Boulahia, A Serhrouchni, and S Tohm. 2000. TCP Over Satellite Links : Problems and Solutions. en, 12.

[15] Carlo Caini and Rosario Firrincieli. 2004. TCP Hybla: a TCP enhancement for heterogeneous networks. en. *International Journal of Satellite Communications and Networking*, 22, 5, (September 2004), 547–566. ISSN: 1542-0973, 1542-0981. DOI: 10.1002/sat.799. Retrieved 04/26/2022 from https://onlinelibrary.wiley.com/doi/10.1002/sat.799.

[16] David Borman, Robert T. Braden, Van Jacobson, and Richard Scheffenegger. 2014. TCP Extensions for High Performance. RFC 7323. (September 2014). DOI: 10.17487/RFC7323. https://www.rfc-editor.org/info/rfc7323.

[17] C. Caini, R. Firrincieli, and D. Lacamera. 2006. Pepsal: a performance enhancing proxy designed for tcp satellite connections. In *2006 IEEE 63rd Vehicular Technology Conference*. Volume 6, 2607–2611. DOI: 10.1109/VETECS.2006.1683339.

[18] James Pavur, Martin Strohmeier, Vincent Lenders, and Ivan Martinovic. 2020. Qpep: a quic-based approach to encrypted performance enhancing proxies for high-latency satellite broadband. *ArXiv*, abs/2002.05091.

[19] [n. d.] Usage Statistics of QUIC for Websites, May 2022. (). Retrieved 05/03/2022 from https://w3techs.com/technologies/details/ce-quic.

[20] Nicolas Kuhn, François Michel, Ludovic Thomas, Emmanuel Dubois, and Emmanuel Lochin. 2020. 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)*. ISSN: 2326-5949. (October 2020), 1–7. DOI: 10.1109/ASMS/SPSC48805.2020.9268814.

[21] Jorg Deutschmann, Kai-Steffen Hielscher, and Reinhard German. 2019. Satellite Internet Performance Measurements. en. In *2019 International Conference on Networked Systems (NetSys)*. IEEE, Munich, Germany, (March 2019), 1–4. ISBN: 978-1-72810-568-0. DOI: 10.1109/NetSys.2019.8854494. Retrieved 01/07/2022 from https://ieeexplore.ieee.org/document/8854494/.

[22] John Border, Bhavit Shah, Chi-Jiun Su, and Rob Torres. 2020. Evaluating QUIC's Performance Against Performance Enhancing Proxy over Satellite Link. In *2020 IFIP Networking Conference (Networking)*. (June 2020), 755–760.

[23] Jana Iyengar and Ian Swett. 2021. QUIC Acknowledgement Frequency. Internet-Draft draft-ietf-quic-ack-frequency-01. Work in Progress. Internet Engineering Task Force, (October 2021). 12 pages. https://datatracker.ietf.org/doc/html/draft-ietf-quic-ack-frequency-01.

[24] Ian Swett, Marie-Jose Montpetit, Vincent Roca, and François Michel. 2020. Coding for QUIC. Internet-Draft draft-swett-nwcrg-coding-for-quic-04. Work in Progress. Internet Engineering Task Force, (March 2020). 17 pages. https://datatracker.ietf.org/doc/html/draft-swett-nwcrg-coding-for-quic-04.

[25] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control: Measuring bottleneck bandwidth and round-trip propagation time. en. *Queue*, 14, 5, (October 2016), 20–53. ISSN: 1542-7730, 1542-7749. DOI: 10.1145/3012426.3022184. Retrieved 04/04/2022 from https://dl.acm.org/doi/10.1145/3012426.3022184.

[26] Benedikt Jaeger, Dominik Scholz, Daniel Raumer, Fabien Geyer, and Georg Carle. 2019. Reproducible measurements of tcp bbr congestion control. *Computer Communications*, 144, 31–43. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2019.05.011. https://www.sciencedirect.com/science/article/pii/S0140366419303470.

[27] Dominik Scholz, Benedikt Jaeger, Lukas Schwaighofer, Daniel Raumer, Fabien Geyer, and Georg Carle. 2018. Towards a deeper understanding of tcp bbr congestion control. In *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, 1–9. DOI: 10.23919/IFIPNetworking.2018.8696830.

[28] Jose Gomez, Elie Kfoury, Jorge Crichigno, Elias Bou-Harb, and Gautam Srivastava. 2020. A Performance Evaluation of TCP BBRv2 Alpha. In *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*. (July 2020), 309–312. DOI: 10.1109/TSP49548.2020.9163512.

[29] Yeong-Jun Song, Won-Ju Eom, Jeong-Keun Kim, Chang-Hoon Park, Geon-Hwan Kim, and You-Ze Cho. 2020. Intra-protocol Convergence Problem in BBRv2's Bandwidth Probing. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*. ISSN: 2162-1233. (October 2020), 1016–1018. DOI: 10.1109/ICTC49870.2020.9289384.

[30] Yeong-Jun Song, Geon-Hwan Kim, Imtiaz Mahmud, Won-Kyeong Seo, and You-Ze Cho. 2021. Understanding of BBRv2: Evaluation and Comparison With BBRv1 Congestion Control Algorithm. *IEEE Access*, 9, 37131–37145. Conference Name: IEEE Access. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3061696.

[31] Furong Yang, Qinghua Wu, Zhenyu Li, Yanmei Liu, Giovanni Pau, and Gaogang Xie. 2022. BBRv2+: Towards balancing aggressiveness and fairness with delay-based bandwidth probing. en. *Computer Networks*, 206, (April 2022), 108789. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2022.108789. Retrieved 02/22/2022 from https://www.sciencedirect.com/science/article/pii/S1389128622000226.

[32] Robin Marx, Joris Herbots, Wim Lamotte, and Peter Quax. 2020. Same Standards, Different Decisions: A Study of QUIC and HTTP/3 Implementation Diversity. en. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*. ACM, Virtual Event USA, (August 2020), 14–20. ISBN: 978-1-4503-8047-8. DOI: 10.1145/3405796.3405828. Retrieved 04/04/2022 from https://dl.acm.org/doi/10.1145/3405796.3405828.

[33] Sebastian Endres, Jörg Deutschmann, Kai-Steffen Hielscher, and Reinhard German. 2022. Performance of QUIC Implementations Over Geostationary Satellite Links. *arXiv:2202.08228 [cs]*, (February 2022). arXiv: 2202.08228. Retrieved 04/04/2022 from http://arxiv.org/abs/2202.08228.

[34] 2016. TCP Experiment Automation Controlled Using Python (TEACUP) – A Tool for Automated TCP Testbed Experiments. (2016). Retrieved 04/26/2022 from http://caia.swin.edu.au/tools/teacup/.

[35] 2022. Ngtcp2. original-date: 2017-06-25T08:28:58Z. (April 2022). Retrieved 04/26/2022 from https://github.com/ngtcp2/ngtcp2.

[36] 2022. Picoquic. original-date: 2017-06-26T19:08:37Z. (April 2022). Retrieved 04/26/2022 from https://github.com/private-octopus/picoquic.

[37] Robin Marx, Luca Niccolini, and Marten Seemann. 2022. Main logging schema for qlog. Internet-Draft draft-ietf-quic-qlog-main-schema-02. Work in Progress. Internet Engineering Task Force, (March 2022). 49 pages. https://datatracker.ietf.org/doc/html/draft-ietf-quic-qlog-main-schema-02.

[38] R. Jain, D. Chiu, and W. Hawe. 1998. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. *arXiv:cs/9809099*, (September 1998). arXiv: cs/9809099. Retrieved 04/10/2022 from http://arxiv.org/abs/cs/9809099.

Bibliography

[1] Transmission Control Protocol. RFC 793, September 1981. URL https://www.rfc-editor.org/info/rfc793.

[2] User Datagram Protocol. RFC 768, August 1980. URL https://www.rfc-editor.org/info/rfc768.

[3] Kurose James and Ross Keith. *Computer Networking: A Top-Down Approach*. Boston, June 2016. ISBN 978-0-13-359414-0.

[4] Randall R. Stewart. Stream Control Transmission Protocol. RFC 4960, September 2007. URL https://www.rfc-editor.org/info/rfc4960.

[5] Sally Floyd, Mark J. Handley, and Eddie Kohler. Datagram Congestion Control Protocol (DCCP). RFC 4340, March 2006. URL https://www.rfc-editor.org/info/rfc4340.

[6] QUIC: Design Document and Specification Rationale, . URL https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit?usp=embed_facebook.

[7] Martin Thomson. Version-Independent Properties of QUIC. RFC 8999, May 2021. URL https://www.rfc-editor.org/info/rfc8999.

[8] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, May 2021. URL https://www.rfc-editor.org/info/rfc9000.

[9] Martin Thomson and Sean Turner. Using TLS to Secure QUIC. RFC 9001, May 2021. URL https://www.rfc-editor.org/info/rfc9001.

[10] Jana Iyengar and Ian Swett. QUIC Loss Detection and Congestion Control. RFC 9002, May 2021. URL https://www.rfc-editor.org/info/rfc9002.

[11] Louis J. Ippolito. *High Throughput Satellites*, pages 398–422. 2017. doi: 10.1002/9781119259411.ch15.

[12] Key elements for integration of satellite systems into Next Generation Access Technologies. *Report ITU-R M.2460-0*, page 19, July 2019.

[13] Chris Daehnick, Isabelle Klinghoffer, Ben Maritz, and Bill Wiseman. Large LEO satellite constellations: Will it be different this time? page 13.

[14] Tom Jones, Gorry Fairhurst, Nicolas Kuhn, John Border, and Stephan Emile. Enhancing Transport Protocols over Satellite Networks. Internet-Draft draft-jones-tsvwg-transport-for-satellite-02, Internet Engineering Task Force, October 2021. URL https://datatracker.ietf.org/doc/html/draft-jones-tsvwg-transport-for-satellite-02. Work in Progress.

[15] Jim Griner, John Border, Markku Kojo, Zach D. Shelby, and Gabriel Montenegro. Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations. RFC 3135, June 2001. URL https://www.rfc-editor.org/info/rfc3135.

[16] Carlo Caini and Rosario Firrincieli. TCP Hybla: a TCP enhancement for heterogeneous networks. *International Journal of Satellite Communications and Networking*, 22(5):547–566, September 2004. ISSN 1542-0973. doi: 10.1002/sat.799. URL https://doi.org/10.1002/sat.799.

[17] Ludovic Thomas, Emmanuel Dubois, Nicolas Kuhn, and Emmanuel Lochin. Google QUIC performance over a public SATCOM access. *International Journal of Satellite Communications and Networking*, 37(6):601–611, 2019. ISSN 1542-0981. doi: 10.1002/sat.1301. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sat.1301. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sat.1301.

[18] Usage Statistics of QUIC for Websites, May 2022, . URL https://w3techs.com/technologies/details/ce-quic.

[19] S Oueslati-Boulahia, A Serhrouchni, and S Tohm. TCP Over Satellite Links : Problems and Solutions. page 12.

[20] Mahesh Sooriyabandara, Gorry Fairhurst, Venkata Padmanabhan, and Hari Balakrishnan. TCP Performance Implications of Network Path Asymmetry. RFC 3449, December 2002. URL https://www.rfc-editor.org/info/rfc3449.

[21] Satellite frequency bands. URL https://www.esa.int/Applications/Telecommunications_Integrated_Applications/Satellite_frequency_bands.

[22] TCP extensions for long-delay paths. RFC 1072, October 1988. URL https://www.rfc-editor.org/info/rfc1072.

[23] Luis A. Sanchez, Mark Allman, and Dr. Dan Glover. Enhancing TCP Over Satellite Channels using Standard Mechanisms. RFC 2488, January 1999. URL https://www.rfc-editor.org/info/rfc2488.

[24] David Borman, Robert T. Braden, Van Jacobson, and Richard Scheffenegger. TCP Extensions for High Performance. RFC 7323, September 2014. URL https://www.rfc-editor.org/info/rfc7323.

[25] Sally Floyd, Jamshid Mahdavi, Matt Mathis, and Dr. Allyn Romanow. TCP Selective Acknowledgment Options. RFC 2018, October 1996. URL https://www.rfc-editor.org/info/rfc2018.

[26] Carlo Caini, Rosario Firrincieli, and Daniele Lacamera. PEPsal: a Performance Enhancing Proxy for TCP satellite connections (and future research directions at UoB). page 22.

[27] James Pavur, Martin Strohmeier, Vincent Lenders, and Ivan Martinovic. QPEP: A QUIC-Based Approach to Encrypted Performance Enhancing Proxies for High-Latency Satellite Broadband. *arXiv:2002.05091 [cs]*, February 2020. URL http://arxiv.org/abs/2002.05091. arXiv: 2002.05091.

[28] Nicolas Kuhn, François Michel, Ludovic Thomas, Emmanuel Dubois, and Emmanuel 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)*, pages 1–7, October 2020. doi: 10.1109/ASMS/SPSC48805.2020.9268814. ISSN: 2326-5949.

[29] Cristian Mogildea, Jörg Deutschmann, Kai-Steffen Hielscher, and Reinhard German. QUIC OVER SATELLITE: INTRODUCTION AND PERFORMANCE MEASUREMENTS. page 10.

[30] John Border, Bhavit Shah, Chi-Jiun Su, and Rob Torres. Evaluating QUIC's Performance Against Performance Enhancing Proxy over Satellite Link. In *2020 IFIP Networking Conference (Networking)*, pages 755–760, June 2020.

[31] Ana Custura, Tom Jones, and Gorry 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)*, pages 1–8, October 2020. doi: 10.1109/ASMS/SPSC48805.2020.9268894. ISSN: 2326-5949.

[32] Han Zhang, Tianqi Wang, Yue Tu, Kanglian Zhao, and Wenfeng Li. How Quick Is QUIC in Satellite Networks. In Qilian Liang, Jiasong Mu, Min Jia, Wei Wang,

Xuhong Feng, and Baoju Zhang, editors, *Communications, Signal Processing, and Systems*, Lecture Notes in Electrical Engineering, pages 387–394, Singapore, 2019. Springer. ISBN 978-981-10-6571-2. doi: 10.1007/978-981-10-6571-2_47.

[33] Yue Wang, Kanglian Zhao, Wenfeng Li, Juan Fraire, Zhili Sun, and Yuan Fang. Performance Evaluation of QUIC with BBR in Satellite Internet. In *2018 6th IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, pages 195–199, December 2018. doi: 10.1109/WiSEE.2018.8637347. ISSN: 2380-7636.

[34] Siyu Yang, Hewu Li, and Qian Wu. Performance Analysis of QUIC Protocol in Integrated Satellites and Terrestrial Networks. In *2018 14th International Wireless Communications Mobile Computing Conference (IWCMC)*, pages 1425–1430, June 2018. doi: 10.1109/IWCMC.2018.8450388. ISSN: 2376-6506.

[35] Mihail Zverev, Pablo Garrido, Fátima Fernández, Josu Bilbao, Özgü Alay, Simone Ferlin, Anna Brunstrom, and Ramón Agüero. Robust QUIC: Integrating Practical Coding in a Low Latency Transport Protocol. *IEEE Access*, 9:138225–138244, 2021. ISSN 2169-3536. doi: 10.1109/ACCESS.2021.3118112. Conference Name: IEEE Access.

[36] Nicolas Kuhn, Francklin Simo, David Pradas, and Emile Stephan. Evaluating BDP FRAME extension for QUIC. *arXiv:2112.05450 [cs]*, December 2021. URL http://arxiv.org/abs/2112.05450. arXiv: 2112.05450.

[37] Mike Kosek, Hendrik Cech, Vaibhav Bajpai, and Jörg Ott. Exploring Proxying QUIC and HTTP/3 for Satellite Communication. Technical Report arXiv:2205.01554, arXiv, May 2022. URL http://arxiv.org/abs/2205.01554. arXiv:2205.01554 [cs] type: article.

[38] Eric Rescorla and Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008. URL https://www.rfc-editor.org/info/rfc5246.

[39] Giorgos Papastergiou, Gorry Fairhurst, David Ros, Anna Brunstrom, Karl-Johan Grinnemo, Per Hurtig, Naeem Khademi, Michael Tüxen, Michael Welzl, Dragana Damjanovic, and Simone Mangiante. De-ossifying the internet transport layer: A survey and future perspectives. *IEEE Communications Surveys Tutorials*, 19(1): 619–639, 2017. doi: 10.1109/COMST.2016.2626780.

[40] Florian Gratzer. QUIC - Quick UDP Internet Connections. page 8, 2016.

[41] QUIC, a multiplexed transport over UDP. URL https://www.chromium.org/quic/.

[42] Yuchung Cheng, Jerry Chu, Sivasankar Radhakrishnan, and Arvind Jain. TCP Fast Open. RFC 7413, December 2014. URL https://www.rfc-editor.org/info/rfc7413.

[43] Adam Langley, Nagendra Modadugu, and Bodo Moeller. Transport Layer Security (TLS) False Start. RFC 7918, August 2016. URL https://www.rfc-editor.org/info/rfc7918.

[44] A. Roy-Chowdhury, J.S. Baras, M. Hadjitheodosiou, and S. Papademetriou. Security issues in hybrid networks with a satellite component. *IEEE Wireless Communications*, 12(6):50–61, 2005. doi: 10.1109/MWC.2005.1561945.

[45] Mike Bishop. Hypertext Transfer Protocol Version 3 (HTTP/3). Internet-Draft draft-ietf-quic-http-34, Internet Engineering Task Force, February 2021. URL https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34. Work in Progress.

[46] Roy T. Fielding, Mark Nottingham, and Julian Reschke. HTTP Semantics. Internet-Draft draft-ietf-httpbis-semantics-19, Internet Engineering Task Force, September 2021. URL https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-19. Work in Progress.

[47] Henrik Nielsen, Jeffrey Mogul, Larry M Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999. URL https://www.rfc-editor.org/info/rfc2616.

[48] Mike Belshe, Roberto Peon, and Martin Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, May 2015. URL https://www.rfc-editor.org/info/rfc7540.

[49] Charles 'Buck' Krasic, Mike Bishop, and Alan Frindell. QPACK: Header Compression for HTTP/3. Internet-Draft draft-ietf-quic-qpack-21, Internet Engineering Task Force, February 2021. URL https://datatracker.ietf.org/doc/html/draft-ietf-quic-qpack-21. Work in Progress.

[50] Tommy Pauly, Eric Kinnear, and David Schinazi. An Unreliable Datagram Extension to QUIC. RFC 9221, March 2022. URL https://www.rfc-editor.org/info/rfc9221.

[51] Yanmei Liu, Yunfei Ma, Quentin De Coninck, Olivier Bonaventure, Christian Huitema, and Mirja Kühlewind. Multipath Extension for QUIC. Internet-Draft draft-ietf-quic-multipath-01, Internet Engineering Task Force, March 2022. URL https://datatracker.ietf.org/doc/html/draft-ietf-quic-multipath-01. Work in Progress.

[52] Quentin De Coninck. The packet number space debate in multipath QUIC. *CoRR*, abs/2112.01068, 2021. URL https://arxiv.org/abs/2112.01068.

[53] Xavier Artiga, Jose Nunez-Martinez, Ana Perez-Neira, Gorka Juan Lendrino Vela, Juan Mario Fare Garcia, and Georgios Ziaragkas. Terrestrial-satellite integration in dynamic 5g backhaul networks. In *2016 8th Advanced Satellite Multimedia Systems Conference and the 14th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, pages 1–6, 2016. doi: 10.1109/ASMS-SPSC.2016.7601470.

[54] Nicolas Kuhn, Stephan Emile, Gorry Fairhurst, Tom Jones, and Christian Huitema. BDP Frame Extension. Internet-Draft draft-kuhn-quic-bdpframe-extension-00, Internet Engineering Task Force, March 2022. URL https://datatracker.ietf.org/doc/html/draft-kuhn-quic-bdpframe-extension-00. Work in Progress.

[55] Nicolas Kuhn, Stephan Emile, Gorry Fairhurst, Tom Jones, and Christian Huitema. Carefully Resume QUIC Session. Internet-Draft draft-kuhn-quic-careful-resume-01, Internet Engineering Task Force, May 2022. URL https://datatracker.ietf.org/doc/html/draft-kuhn-quic-careful-resume-01. Work in Progress.

[56] Jana Iyengar and Ian Swett. QUIC Acknowledgement Frequency. Internet-Draft draft-ietf-quic-ack-frequency-01, Internet Engineering Task Force, October 2021. URL https://datatracker.ietf.org/doc/html/draft-ietf-quic-ack-frequency-01. Work in Progress.

[57] Robin Marx, Joris Herbots, Wim Lamotte, and Peter Quax. Same Standards, Different Decisions: A Study of QUIC and HTTP/3 Implementation Diversity. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, pages 14–20, Virtual Event USA, August 2020. ACM. ISBN 978-1-4503-8047-8. doi: 10.1145/3405796.3405828. URL https://dl.acm.org/doi/10.1145/3405796.3405828.

[58] Marten Seemann. Interop Test Runner, May 2022. URL https://github.com/marten-seemann/quic-interop-runner. original-date: 2019-10-15T21:36:56Z.

[59] Sebastian Endres, Jörg Deutschmann, Kai-Steffen Hielscher, and Reinhard German. Performance of QUIC Implementations Over Geostationary Satellite Links. *arXiv:2202.08228 [cs]*, February 2022. URL http://arxiv.org/abs/2202.08228. arXiv: 2202.08228.

[60] aioquic, May 2022. URL https://github.com/aiortc/aioquic. original-date: 2019-02-04T23:27:58Z.

[61] LiteSpeed Tech. LiteSpeed QUIC (LSQUIC) Library README, May 2022. URL https://github.com/litespeedtech/lsquic. original-date: 2017-09-22T20:33:18Z.

[62] microsoft/msquic, May 2022. URL https://github.com/microsoft/msquic. original-date: 2019-10-26T04:10:24Z.

[63] facebookincubator/mvfst, May 2022. URL https://github.com/facebookincubator/mvfst. original-date: 2018-04-09T22:49:15Z.

[64] How Facebook is bringing QUIC to billions, October 2020. URL https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/.

[65] Neqo, an Implementation of QUIC written in Rust, May 2022. URL https://github.com/mozilla/neqo. original-date: 2019-02-18T19:20:20Z.

[66] ngtcp2, May 2022. URL https://github.com/ngtcp2/ngtcp2. original-date: 2017-06-25T08:28:58Z.

[67] picoquic, May 2022. URL https://github.com/private-octopus/picoquic. original-date: 2017-06-26T19:08:37Z.

[68] QUICHE, May 2022. URL https://github.com/google/quiche. original-date: 2021-07-27T18:19:46Z.

[69] cloudflare/quiche, May 2022. URL https://github.com/cloudflare/quiche. original-date: 2018-09-29T18:22:05Z.

[70] quicly, May 2022. URL https://github.com/h2o/quicly. original-date: 2017-06-06T14:27:42Z.

[71] Lucas Clemente. A QUIC implementation in pure Go, May 2022. URL https://github.com/lucas-clemente/quic-go. original-date: 2016-04-06T20:16:27Z.

[72] Sally Floyd. Congestion Control Principles. RFC 2914, September 2000. URL https://www.rfc-editor.org/info/rfc2914.

[73] Tom Henderson and Sally Floyd. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582, April 1999. URL https://www.rfc-editor.org/info/rfc2582.

[74] Injong Rhee, Lisong Xu, Sangtae Ha, Alexander Zimmermann, Lars Eggert, and Richard Scheffenegger. CUBIC for Fast Long-Distance Networks. RFC 8312, February 2018. URL https://www.rfc-editor.org/info/rfc8312.

[75] Lawrence S. Brakmo, Sean W. O'malley, and Larry L. Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *In SIGCOMM*, 1994.

[76] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Ian Swett, and Van Jacobson. BBR Congestion Control. Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-02, Internet Engineering Task Force, March 2022. URL https://datatracker.ietf.org/doc/html/draft-cardwell-iccrg-bbr-congestion-control-02. Work in Progress.

[77] Sally Floyd, Dr. K. K. Ramakrishnan, and David L. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, September 2001. URL https://www.rfc-editor.org/info/rfc3168.

[78] Fred Baker and Gorry Fairhurst. IETF Recommendations Regarding Active Queue Management. RFC 7567, July 2015. URL https://www.rfc-editor.org/info/rfc7567.

[79] Ethan Blanton, Dr. Vern Paxson, and Mark Allman. TCP Congestion Control. RFC 5681, September 2009. URL https://www.rfc-editor.org/info/rfc5681.

[80] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *ACM Queue*, 14, September-October:20 – 53, 2016. URL http://queue.acm.org/detail.cfm?id=3022184.

[81] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: Congestion-Based Congestion Control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue*, 14(5):20–53, October 2016. ISSN 1542-7730, 1542-7749. doi: 10.1145/3012426.3022184. URL https://dl.acm.org/doi/10.1145/3012426.3022184.

[82] Mario Hock, Roland Bless, and Martina Zitterbart. Experimental evaluation of BBR congestion control. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, pages 1–10, Toronto, ON, October 2017. IEEE. ISBN 978-1-5090-6501-1. doi: 10.1109/ICNP.2017.8117540. URL http://ieeexplore.ieee.org/document/8117540/.

[83] Benedikt Jaeger, Dominik Scholz, Daniel Raumer, Fabien Geyer, and Georg Carle. Reproducible measurements of tcp bbr congestion control. *Computer Communications*, 144:31–43, 2019. ISSN 0140-3664. doi: https://doi.org/10.1016/j.comcom.2019.05.011. URL https://www.sciencedirect.com/science/article/pii/S0140366419303470.

[84] Dominik Scholz, Benedikt Jaeger, Lukas Schwaighofer, Daniel Raumer, Fabien Geyer, and Georg Carle. Towards a deeper understanding of tcp bbr congestion

control. In *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 1–9, 2018. doi: 10.23919/IFIPNetworking.2018.8696830.

[85] Jose Gomez, Elie Kfoury, Jorge Crichigno, Elias Bou-Harb, and Gautam Srivastava. A Performance Evaluation of TCP BBRv2 Alpha. In *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*, pages 309–312, July 2020. doi: 10.1109/TSP49548.2020.9163512.

[86] Yeong-Jun Song, Won-Ju Eom, Jeong-Keun Kim, Chang-Hoon Park, Geon-Hwan Kim, and You-Ze Cho. Intra-protocol Convergence Problem in BBRv2's Bandwidth Probing. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1016–1018, October 2020. doi: 10.1109/ICTC49870.2020.9289384. ISSN: 2162-1233.

[87] Yeong-Jun Song, Geon-Hwan Kim, Imtiaz Mahmud, Won-Kyeong Seo, and You-Ze Cho. Understanding of BBRv2: Evaluation and Comparison With BBRv1 Congestion Control Algorithm. *IEEE Access*, 9:37131–37145, 2021. ISSN 2169-3536. doi: 10.1109/ACCESS.2021.3061696. Conference Name: IEEE Access.

[88] Furong Yang, Qinghua Wu, Zhenyu Li, Yanmei Liu, Giovanni Pau, and Gaogang Xie. BBRv2+: Towards balancing aggressiveness and fairness with delay-based bandwidth probing. *Computer Networks*, 206:108789, April 2022. ISSN 1389-1286. doi: 10.1016/j.comnet.2022.108789. URL https://www.sciencedirect.com/science/article/pii/S1389128622000226.

[89] Ian Swett, Marie-Jose Montpetit, Vincent Roca, and François Michel. Coding for QUIC. Internet-Draft draft-swett-nwcrg-coding-for-quic-04, Internet Engineering Task Force, March 2020. URL https://datatracker.ietf.org/doc/html/draft-swett-nwcrg-coding-for-quic-04. Work in Progress.

[90] Tommy Pauly and David Schinazi. QUIC-Aware Proxying Using HTTP. Internet-Draft draft-pauly-masque-quic-proxy-03, Internet Engineering Task Force, March 2022. URL https://datatracker.ietf.org/doc/html/draft-pauly-masque-quic-proxy-03. Work in Progress.

[91] Multiplexed Application Substrate over QUIC Encryption (masque). URL https://datatracker.ietf.org/wg/masque/about/.

[92] David Schinazi. Proxying UDP in HTTP. Internet-Draft draft-ietf-masque-connect-udp-13, Internet Engineering Task Force, June 2022. URL https://datatracker.ietf.org/doc/html/draft-ietf-masque-connect-udp-13. Work in Progress.

[93] David Schinazi and Lucas Pardue. HTTP Datagrams and the Capsule Protocol. Internet-Draft draft-ietf-masque-h3-datagram-09, Internet Engineering

Task Force, April 2022. URL https://datatracker.ietf.org/doc/html/draft-ietf-masque-h3-datagram-09. Work in Progress.

[94] Zsolt Krämer, Mirja Kühlewind, Marcus Ihlar, and Attila Mihály. Cooperative performance enhancement using quic tunneling in 5g cellular networks. In *Proceedings of the Applied Networking Research Workshop*, ANRW '21, page 49–51, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450386180. doi: 10.1145/3472305.3472320. URL https://doi.org/10.1145/3472305.3472320.

[95] Mirja Kühlewind, Matias Carlander-Reuterfelt, Marcus Ihlar, and Magnus Westerlund. Evaluation of quic-based masque proxying. In *Proceedings of the 2021 Workshop on Evolution, Performance and Interoperability of QUIC*, EPIQ '21, page 29–34, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450391351. doi: 10.1145/3488660.3493806. URL https://doi.org/10.1145/3488660.3493806.

[96] Antoine Auger, Emmanuel Lochin, and Nicolas Kuhn. Making Trustable Satellite Experiments: an Application to a VoIP Scenario. pages 1–5, 2019. URL https://doi.org/10.1109/VTCSpring.2019.8746404.

[97] OpenSAND. URL https://www.opensand.org/.

[98] SNS3. URL https://www.sns3.org/content/home.php.

[99] TCP Experiment Automation Controlled Using Python (TEACUP) – A Tool for Automated TCP Testbed Experiments. URL http://caia.swin.edu.au/tools/teacup/.

[100] Einar Haeger Solfjell. *Networking Experiment Reproducibility: A Case of BBR Congestion Control*. Bachelor's Thesis in Computer Science, University of Stavanger, May 2020.

[101] The FreeBSD Project. URL https://www.freebsd.org/.

[102] Welcome to Fabric! — Fabric documentation. URL https://www.fabfile.org/.

[103] Sebastian Zander and Grenville Armitage. TEACUP v1.0 – A System for Automated TCP Testbed Experiments. page 45, 2015.

[104] Robin Marx, Luca Niccolini, and Marten Seemann. Main logging schema for qlog. Internet-Draft draft-ietf-quic-qlog-main-schema-02, Internet Engineering Task Force, March 2022. URL https://datatracker.ietf.org/doc/html/draft-ietf-quic-qlog-main-schema-02. Work in Progress.

[105] tc-netem(8) - Linux manual page. URL https://man7.org/linux/man-pages/man8/tc-netem.8.html.

[106] Chris Rapier. rapier1/web10g, October 2020. URL https://github.com/rapier1/web10g. original-date: 2013-10-22T19:15:01Z.

[107] FreeBSD. siftr(4). URL https://www.freebsd.org/cgi/man.cgi?query=siftr&sektion=4&manpath=FreeBSD+8.2-RELEASE.

[108] Robin Marx, Luca Niccolini, and Marten Seemann. QUIC event definitions for qlog. Internet-Draft draft-ietf-quic-qlog-quic-events-01, Internet Engineering Task Force, March 2022. URL https://datatracker.ietf.org/doc/html/draft-ietf-quic-qlog-quic-events-01. Work in Progress.

[109] Robin Marx, Luca Niccolini, and Marten Seemann. HTTP/3 and QPACK qlog event definitions. Internet-Draft draft-ietf-quic-qlog-h3-events-01, Internet Engineering Task Force, March 2022. URL https://datatracker.ietf.org/doc/html/draft-ietf-quic-qlog-h3-events-01. Work in Progress.

[110] R. Jain, D. Chiu, and W. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. *arXiv:cs/9809099*, September 1998. URL http://arxiv.org/abs/cs/9809099. arXiv: cs/9809099.

[111] Jana Iyengar, David Ros, Andres Arcia, and Sally Floyd. Adding Acknowledgement Congestion Control to TCP. RFC 5690, February 2010. URL https://www.rfc-editor.org/info/rfc5690.

[112] Applied Networking Research Workshop (ANRW '22). URL https://irtf.org/anrw/2022/.