When BBR beats FEC coded CUBIC flows over SATCOM

Matthieu Petrou, David Pradas, Nicolas Kuhn, Mickaël Royer, Emmanuel Lochin

When BBR beats FEC coded CUBIC flows over SATCOM

Matthieu Petrou
Viveris Technologies
Toulouse, FRANCE
matthieu.petrou@viveris.fr

David Pradas
Viveris Technologies
Toulouse, FRANCE
david.pradas@viveris.fr

Nicolas Kuhn
CNES
Toulouse, FRANCE
nicolas.kuhn@cnes.fr

Mickaël ROYER
ENAC
Toulouse, FRANCE
mickael.royer@enac.fr

Emmanuel Lochin
ENAC
Toulouse, FRANCE
emmanuel.lochin@enac.fr

***Abstract*—This paper assesses the benefits of using Sliding Windows Forward error codes (SWF) to protect transport protocol sessions over a SATCOM link within IP tunnels. We consider two commonly deployed protocols and congestion control algorithms : TCP/CUBIC, currently deployed by default inside most of recent OS kernels and a QUIC/BBR implementation named Picoquic. Our objective is to evaluate the performance of these protocols in challenging SATCOM environments and to assess if SWF can contribute to improve their performance. We consider two different scenarios based on real loss mobility patterns played over the OpenSAND satellite emulator. Results show that using SWF tunnels can hide losses to a CUBIC server: this reduces the download time of 20MB by more than 90%. However, the main finding is that SWF does not contribute to the download time reduction for BBR, making useless its deployment. We conclude that the use of BBR over SATCOM could be an efficient way to perform communications over unreliable links, resulting from a high mobility context for instance, considering BBR flows are managed by an adequate QoS allocation.**

***Index Terms*—TCP, QUIC, SATCOM, Sliding Window FEC, CUBIC, BBR**

I. Introduction

Satellite links represent an important part of the global network. They provide Internet for rural and isolated areas with no/low terrestrial network connectivity (fiber, ADSL or mobile networks), but also for maritime and airborne areas. High latency is one of the main characteristics of GEO satellite networks, due to the distance of geosynchronous orbits which induces a round-trip time (RTT) around $600\,ms$. This latency reduces the capability of congestion windows of congestion-controlled transport protocols to grow rapidly at the start of the transfer, or after deleterious events. As a matter of fact, these events such as packets loss, have a major impact on the overall transport performance and on the Quality of Experience (QoE) of users [1], [2].

Performance Enhancing Proxies (PEP) [3] have been designed to mitigate this high latency problem and speed up TCP over SATCOM. However, QUIC protocol cannot benefit from PEPs acceleration and retransmission. QUIC protocol has been designed to perform on top of UDP and following its privacy policy, encrypts data packet and control information, preventing PEP technology to identify QUIC flows. Previous studies over SATCOM [4], [5] pointed out QUIC issues due to the absence of PEP acceleration. Furthermore, without PEP, any lost packets will trigger retransmission that will cross the whole path. QUIC global traffic share will certainly grow, as HTTP3 is expected to use QUIC [6]. Therefore, there is a need for solution for QUIC over SATCOM [7].

A potential solution for QUIC, or to prevent TCP retransmissions, can be to use Sliding Window Forward Erasure Coding tunnels [8] (denoted in the following Sliding Window FEC or SWF). This idea follows certain Internet Engineering Task Force (IETF) groups working on the possibility to use tunnels over satellite link [9], [10]. Their objective is to robustify transport sessions by enabling SWF tunnels, or integrating SWF directly inside the QUIC protocol [11]. SWF brings interesting decoding properties compared to the standard block FEC scheme [8]. We explain in Section II-C why this scheme is of interest over long delay link, and this motivates the present study that aims to weight up the gain obtained by using SWF tunnels over a satellite link.

The IRTF draft "Coding and congestion control in transport" [12], focuses on the deployment of FEC schemes variants in congestion-controlled transport protocols. This paper contributes to answer some of the points raised by this IRTF draft. We also compare two congestion control algorithms (loss based and delay based) to answer the Research Recommendation #3 that proposes to discuss about the benefits between using FEC (or in our case SWF) and replacing the congestion control algorithm. Note that in our tests, SWF is always placed below the layer transport protocol, which means that recovered losses are hidden to the transport layer.

In this paper, we investigate the benefits of using a SWF tunnel to protect TCP/CUBIC and QUIC/BBR (i.e. Picoquic implementation of BBR) over GEO satellite links. As TCP/CUBIC is the default congestion control algorithm in GNU/Linux servers [13], we consider TCP/CUBIC as a reference protocol for loss-based congestion control.

Picoquic implements its own BBR variant and enables this congestion control by default. We choose Picoquic/BBR compared to other implementations because the code is stable and mature. Furthermore, Picoquic/BBR has been designed with long-delay use-cases in-mind and demonstrates good performance over SATCOM [14]. By the way, Picoquic has been deeply discussed in the EToSat IRTF working group[1]. For the

[1]See: https://ietf.topicbox-scratch.com/groups/etosat/T88ef7ba0fac5b010/side-meeting-on-quic-performance-over-high-bdp-networks

whole paper, QUIC/BBR refers to *"Picoquic implementation with BBR congestion control"* while BBR alone, to the BBR congestion control. Respectively, TCP/CUBIC refers to the *"GNU/Linux kernel implementation of CUBIC"* while CUBIC alone, to the CUBIC congestion control.

The rest of this document is organized as follows: Section II details related research, Section III presents our experimental methodology and test cases, Section IV presents and analyzes our results, and Section V summarizes our work.

II. Background

In this section, we briefly introduce QUIC, the congestion control algorithms tested, and the sliding window FEC scheme.

A. QUIC

As presented in Section I, HTTP3 is expected to lay on QUIC protocol [6], and the IETF standardized QUIC in May 2021 [15]. Nowadays, both Firefox and Chrome enable HTTP3 or QUIC by default.

QUIC protocol is built on top of UDP, therefore, QUIC datagrams are encapsulated in UDP frames. However, in contrast to UDP, QUIC uses some TCP features such as the acknowledgment of received packets, flow and congestion control. The overall goal behind QUIC is to make the Internet faster and more secure. To provide a quicker connection, QUIC uses a single UDP socket to send several QUIC streams handling different files. All these streams are scheduled to ensure fairness and enable priorities between them. By comparison, TCP would need to have multiple concurrent TCP flows, using multiple sockets, and the fairness would depend on the congestion control algorithms of those flows without real priority system. As UDP does not provide packet encryption, QUIC brings cryptographic functions to secure communications. Therefore, each QUIC stream, from a given UDP flow, can be decrypted separately, and does not increase the risk of head of line blocking. To reduce startup latency, QUIC needs only one round trip to both set up the connection and secure it, while TCP needs at least two or three round trip to perform the TCP handshake and the TLS negotiation. Furthermore, QUIC implements a 0-RTT feature which allows to send data immediately when a previous communication has already been established to prevent a new connection negotiation.

As defined today, RFC9000 [15] is letting the possibility to developers to make consistent choice among various options while remaining compliant. A recent study [16] confirmed this, where 45 different QUIC transport parameters settings were found on Internet. Picoquic [17] is a good illustration of an implementation of the QUIC protocol. As shown by the amount of pull request of the Picoquic project, its developers made various optimizations while remaining compliant with the RFC.

In this paper, we consider QUIC only as a transport protocol framework and thus we mainly focus on the performance of the congestion control algorithm.

B. Congestion control algorithms

CUBIC was proposed in 2008 [18] as a step forward for congestion control algorithms. CUBIC is used by default in GNU/Linux since 2006 and in Windows Server since 2017. With this congestion control algorithm, a packet loss is used as an indicator of congestion. Therefore, the congestion window size is reduced whenever a loss occurs. However, in case of error-link losses, CUBIC limits the flows throughput to prevent an inadequate reaction to non existent congestion.

By default in GNU/Linux, TCP/CUBIC starts with a specific algorithm named Hybrid Slow start (Hystart), proposed in 2011 [19]. Hystart algorithm estimates when to leave the slow start phase and when to enter the congestion avoidance phase. To assess the limit of the available bandwidth, Hystart mainly checks the RTT increases. With variable or high latency, Hystart may exit prematurely the slow start phase [20], [21]. We evaluate its impact in Section III-C1.

BBRv1 was introduced by Cardwell *et al.* [22]. In 2019, the same authors introduced BBRv2 [23], [24], a fairer version of BBRv1 [25]. BBR assesses maximum available bandwidth and the minimum measured RTT to determine the congestion window size. With a throughput equals to the measured available bandwidth, BBR does not excessively and unnecessarily fills bottleneck buffer. Therefore, BBR prevents the bufferbloat and reduces the overall latency.

To probe the available bandwidth, BBR tries to send data faster, then slower and finally at the same rate than its current throughput. When the available bandwidth is evaluated and every five seconds (as a default value), BBR probes the RTT. To probe the RTT, it drains the bottleneck buffer by reducing the throughput to half of the evaluated bandwidth. After one round trip, BBR considers the buffer empty and measures the minimum RTT. Basically, BBR alternates between these two phases. Contrarily to BBRv1, BBRv2 also enables an in-flight data limitation, which forces the reduction of the congestion window when losses occur or when Explicit Congestion Notification (ECN) rate is too high. In the IRTF draft of BBRv2 [24], the default values of losses rate threshold and ECN rate threshold are 2% and 50%, respectively. However, these thresholds could be modified to make BBR less sensitive to losses.

As seen previously, BBR implementation is still in a standardization process [24] and is still strongly discussed. This leads to several variants and implementations proposed. For example, Picoquic implements its own version of BBR. Based on BBRv1 implementation, Picoquic BBR was modified to match with BBRv2. However, some difference remains such as a Hystart algorithm during its start phase, or the in-flight data limitation which is not handled. In others words, we can consider that the implementation of BBR in Picoquic does not set the loss threshold from BBRv2. Finally, Picoquic BBR has an evolving ACK rate during the connection, which is also not present in Google BBR.

In 2018, 22.4% of Alexa top 250 sites were using CUBIC and 25.2% were using BBRv1 [13]. Those results depict

the variety of the implementations deployed over Internet, composed by a mix of different congestion control algorithms. Furthermore, BBR is not, to our knowledge, present in Windows server nor as a congestion control in GNU/Linux.

We recall that we choose CUBIC as a reference protocol, and Picoquic sets by default with its own BBR implementation. As our goal is not to compare CUBIC and BBR but to evaluate impact of SWF according to the congestion control algorithm under test, it is important to keep in mind that on a high RTT and a certain packet loss ratio, BBR would have a significant advantage over CUBIC [26]. However, recent tests done over a real satellite link, with almost no losses, showed that the difference was not that important [27].

C. Sliding Window FEC (SWF)

Sliding Window FEC is a specific class of Forward Error Correction (FEC) based on a sliding encoding window [8]. FEC block codes applied below the transport layer is an efficient way to improve packet transmission over long delay link, as it prevents retransmissions of lost packets. This possibility to recover lost packets independently from the RTT is a strong advantage considering the feedback response time of a GEO link for instance. A FEC block code is built with source and redundant data packets [28]. The rebuilt of lost packets is only possible if their number is lesser than, or equal to the number of redundant packets [29] within the window. A good value for the block size is necessarily a balance between the maximum FEC decoding latency at the receiver, and the desired robustness against long loss bursts. SWF uses an encoding window with a fixed or variable size, that slides over the set of source data packets. FEC encoding is launched whenever needed from the set of source data packets currently present in the sliding encoding window at a given time. This approach significantly reduces FEC-related latency, since redundancy packets can be generated and passed to the transport layer on-the-fly at any time, and can be regularly received by receivers to quickly recover packet losses [29]. Using SWF codes is therefore highly beneficial to real-time flows [29] but also for data transfer long delay link. SWF also handles the problem of setting a block code size, making it a good candidate for IP tunnels where flows are aggregated between them.

We denote $S_{(k,z)}$, an SWF with parameters z the size of the sliding window; and k the step size in packets where k and $z \in \mathbb{N}^*$ and $k << z$. R_i^j is a redundancy packet combining a set of source packets P ranging from P_i to P_j with $i \in \{1, k+1, 2k+1, ...nk+1\}$ and $P_j \in \{i+z-1\}$. During the startup phase, the window grows up to z and then slides. Therefore, during this first round, we have R_1^j where $j \in \{k, 2k, 3k..., z\}$. A sliding window $S_{(4,20)}$ and its redundant packets are represented in Fig. 1. The computation complexity grows as the number of packets and the sliding window increase. This FEC scheme recently standardized in [8] follows the same principle developed previously by Martinian *et al.* in [30].

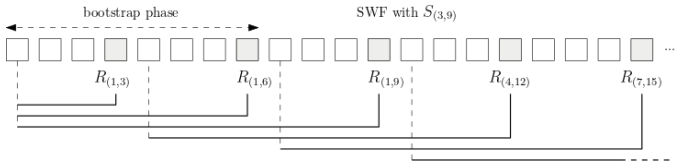

Fig. 1. Illustration with a Sliding Window FEC $S_{(3,9)}$, with a sliding window size (z) of 9 packets and a redundant packet every 3 packets (k). Redundant packets are colored in gray and source data packets in white. R_i^j represents redundant packets, which covers original packets between i and j included.

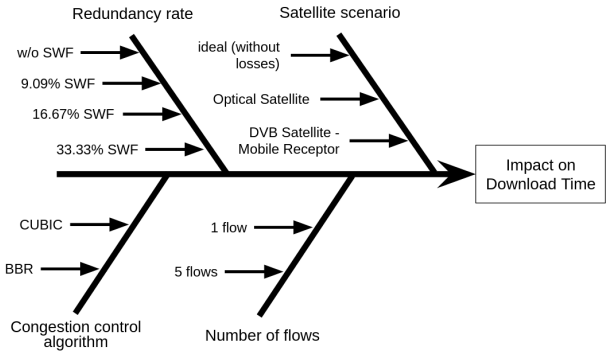

Fig. 2. Cause and effect diagram of our system. We evaluate the performance of the download time for different redundancy rates, scenarios, congestion control algorithms and number of concurrent flows.

III. Measurement methodology

We detail in this section the setup and scenarios chosen to evaluate the gain of using SWF to protect TCP/CUBIC or QUIC/BBR flows. We perform a sanity check (identified as ideal scenario) to validate our experimental setup. In this scenario there is no loss on the SATCOM link. Fig. 2 summarizes the input variables that must be considered as they impact on the download time.

A. Experimental Setup

The experimental setup is built with two end-to-end nodes (a client and a server) and three emulated devices : a satellite terminal, a satellite gateway and a GEO satellite, all of them emulated with OpenSAND [31], [32], an open-source end-to-end satellite communication system emulator. OpenSAND can be exploited to emulate SATCOM systems with a fair representation [33]. A client is connected to the satellite terminal, via either an emulated Wi-Fi link or a LAN network. A server hosting the content (i.e. the files to be downloaded) is placed behind the satellite gateway. Fig. 3 depicts this topology.

To set a realistic public satellite Internet access, we configure the OpenSAND satellite link with a forward bandwidth of 12Mb/s, a return bandwidth of 3Mb/s and a one-way delay of $250\,ms$. We use real SATCOM packets loss scenario further presented in this section. In addition, supplementary packet loss ratio on the Wi-Fi link are either set to zero or one percent. To prevent spurious retransmissions, we set the buffer size

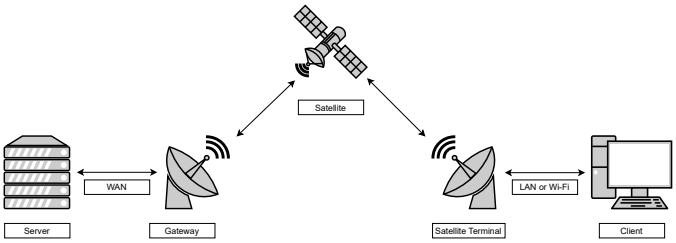

Fig. 3. Testbed topology with OpenSAND, the satellite emulator [31]–[33].

of the bottleneck to 1.2 times the Bandwidth Delay Product (BDP).

For the sake of reproducibility, all tests are orchestrated by OpenBACH [34], an open-source network metrology test bench. OpenBACH allows both the reproducibility of the platform and scenarios experimented by controlling the traffic generation (driven by iPerf3 [35] (v3.7+) for TCP/CUBIC or Picoquic (v0.34f) [17] for QUIC/BBR), as well as the collection and post-treatment of QoS/QoE data.

We realize 30 iterations of two sets of tests, with either one or five concurrent flows. Each flow runs within its own thread to prevent a fair-share bias from the sending application and ensure that the transport layer is the only one to manage the congestion window. Either with one or five flows, each flow transfers 20 MB of data to ensure that the download time is long enough to obtain a noticeable difference between various configurations. This size also prevents to mitigate the slow start phase impact.

We use a SWF tunnel based on the SWIF-CODEC proposed by IRTF-NWCRG [36]. To verify our hypothesis, we consider four configurations. Only the rate of redundant packets varies between the configurations, the sliding window size stays constant to simplify the comparison:

1) without SWF (w/o SWF);
2) $S_{(10,100)}$: one redundant packet every ten source packets, with a sliding window of one hundred packets, which represents 9.09% of redundancy;
3) $S_{(5,100)}$: one redundant packet every five source packets, with a sliding window of one hundred packets, which represents 16.67% of redundancy;
4) $S_{(2,100)}$: one redundant packet every two source packets, with a sliding window of one hundred packets, which represents 33.33% of redundancy.

With a capacity sets to $12\,Mb/s$, and a packet size limited to $1400\,B$ (to prevent fragmentation due to the tunneling), the sliding window is set to 100 packets, which covers around $100\,ms$. This configuration should cover our longest packet losses burst of the tested scenarios.

B. Scenarios

We consider three different losses scenarios which cover a fair wide variety of use-cases described below.

1) Optical scenario: This scenario is the optical satellite use case. To be closer to a real optical satellite, we consider collected data which were summarized on a Gilbert-Elliot model [37]. Based on this model, we generate time based traces with parameters $p = 0.01$ and $q = 0.167$. OpenSAND scheduling step is 10ms, meaning that a switch between a good and bad GE state cannot be done below a 10ms window. The loss burst length varies from 10 ms to 43 ms, with a mean loss burst length at 16.70 ms. Over 30 seconds of traces, 0.768 s corresponds to losses (2.56%). Over a 10-minute tests, a UDP stream of 10 Mb/s gets 2.70% packets dropped. Fig. 4 shows the losses over time.

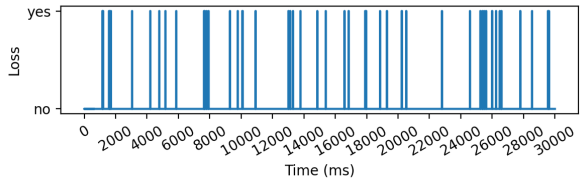

Fig. 4. Losses over time of the optical satellite scenario. Based on collected data [37], traces are generated using a Gilbert-Elliot model (p = 0.01 and q = 0.167).

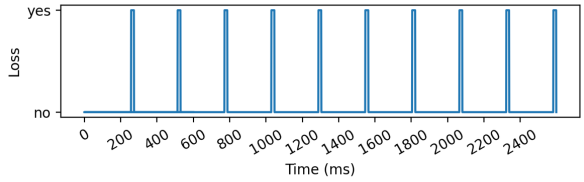

Fig. 5. Losses over time of DVB satellite - mobile receptor scenario. These traces have been collected on a moving train connected to DVB satellite.

2) DVB Mobile scenario: This scenario is based on collected traces on a moving train connected through a DVB satellite. Losses are mainly due to catenaries. Based on traces previously used in other CNES[2] studies, the loss packet bursts are regular. They occur around every 258 ms and last for 15-16 ms. Outside of those burst, the link is perfectly stable and without losses. Overall, the loss time represents 5.89% of the traces. With a UDP stream of 10 Mb/s for 10 minutes, 6.32% of the packets are lost. Fig. 5 shows the losses over time.

3) Sub-scenarios: We add two sub-scenarios to the *Optical* and *DVB Mobile* scenarios. The first one is "*without Wi-Fi*", hence we do not add any losses on the user LAN. The second one is "*with Wi-Fi*", and we add 1% of uniform random losses between the satellite terminal and the client.

C. Ideal scenario: the case-control study

As previously explained, we validate the experimental setup with no loss on the satellite or the Wi-Fi link. With this "ideal scenario", we ensure that our experimental setup is working as intended. As Hystart is known to be inefficient with high RTT and large jitter (cf. Section II), first and foremost we first compare CUBIC and BBR with and without Hystart enabled. To estimate the overhead of the tunnel, we perform a

[2]The National Centre for Space Studies is the French government space agency.

test within the tunnel and without SWF enabled. Finally, we compare the various configurations to evaluate the redundancy impact on a scenario where SWF is useless. Table I collects all the results from the ideal scenario.

1) Hystart: We investigated the impact of Hystart [19]. We noticed some performance problems when enabled and as recently reported in [38]. As a side note, we wish to point out that the Hystart Picoquic implementation is quite different from the TCP/CUBIC one. Furthermore, BBRv1 and BBRv2 do not use Hystart algorithm during start phase.

Our tests confirm that Hystart has a deleterious effect on the TCP/CUBIC performance while beneficial with QUIC/BBR. This effect is diluted over an aggregate for both protocols, but it still negatively impacts on TCP/CUBIC standard deviation (Table I). Considering these results, we deactivate Hystart on TCP/CUBIC for the rest of our tests, and let it enabled on QUIC/BBR.

2) Tunnel overhead: This set of tests ensure that the tunnel, without redundancy, has no impact on the download time. Results for both protocols show that the tunnel does not alter significantly the download time (less than 3% for TCP/CUBIC and QUIC/BBR) nor the standard deviation (less than 0.2 seconds for one flow and 0.9 seconds or less for 5 flows).

3) SWF overhead: When there is no loss, the SWF redundancy packets have an overhead in terms of available bandwidth. As expected, download time medians worsen as the redundancy grows. Results also show that the negative effect of SWF is stronger on QUIC/BBR than on TCP/CUBIC. To clarify these results, we compare both steady-state throughput. As a matter of fact, it seems that TCP/CUBIC without SWF does not use the full available bandwidth. TCP/CUBIC without SWF converges to a mean throughput equals to 10.62 Mb/s while QUIC/BBR obtains 11.71 Mb/s. To go further, once TCP/CUBIC flow converges, the throughput still has some important drops compare to QUIC/BBR, or protected TCP/CUBIC flows.

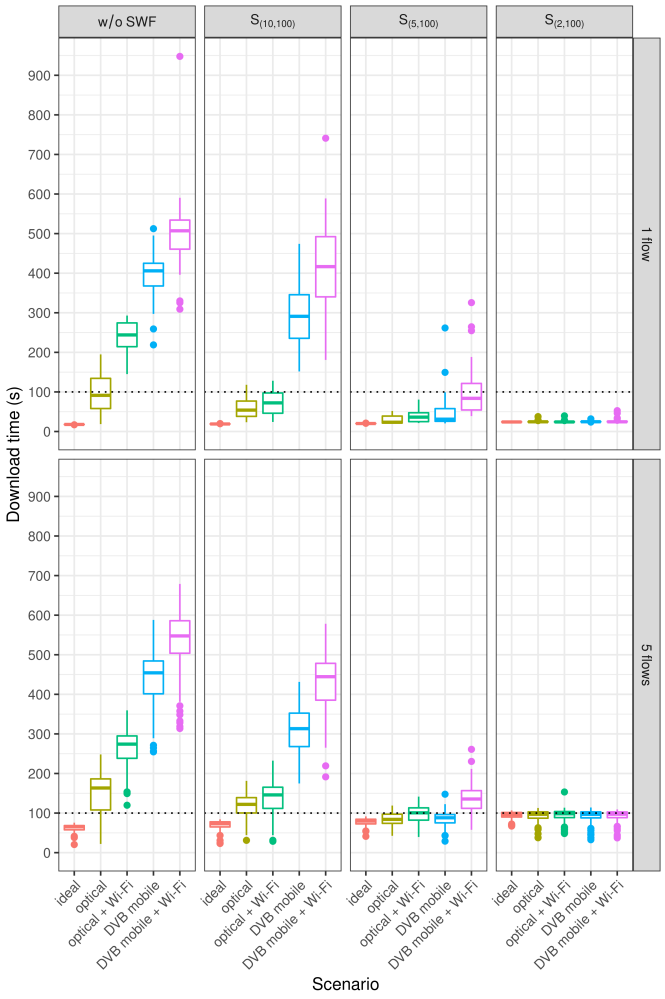

Fig. 6. TCP/CUBIC download time results for each scenario and configuration. Number of sample: 30 for 1 flow and 150 for 5 flows.

IV. Results

We measured the download time over the various scenarios previously defined (cf. Section III) along with different configurations of SWF. Table II compiles all download time median results, as well as the percentage difference between the results without SWF and those with SWF. Fig. 6 and Fig. 7 presents respectively TCP/CUBIC and QUIC/BBR results displayed as whisker boxes, depending on scenarios and configurations.

A. Optical scenario

By way of reminder, with the optical satellite scenario, losses pulse by burst following a Gilbert-Elliot model (cf. Scenarios section).

1) Optical satellite without Wi-Fi: Losses in this scenario negatively impact on TCP/CUBIC download time, i.e., without losses on the link, one unprotected flow takes only 17.84 seconds to download 20 MB, and in this scenario it takes 91.42 seconds to download the same amount of data, which is more than 4 times longer. With 16.67% or 33.33% of redundancy, respectively $S_{(5,100)}$ and $S_{(2,100)}$, we get closer to the ideal scenario. On the other hand, the same losses do not have a strong impact on QUIC/BBR. Even worse, SWF reduces the effectiveness of QUIC/BBR.

2) Optical satellite with Wi-Fi: In the same scenario but with Wi-Fi, which adds 1% of random packet losses on the client LAN, TCP/CUBIC results are even worse. With one and five TCP/CUBIC flows, the $S_{(2,100)}$ configuration becomes necessary to stay close to the ideal scenario results. With Wi-Fi, QUIC/BBR does not seem impacted, or at least not enough to get benefits from tested SWF configurations. Therefore, we did some additional tests on QUIC/BBR with $S_{(20,100)}$ and $S_{(50,100)}$, i.e., 4.76% and 1.96% of redundancy respectively. The results are not better with $S_{(20,100)}$ with 20.92 seconds and 85.84 seconds, respectively for one and five flows. $S_{(50,100)}$ configuration results median slightly improve one flow download time (20.51 seconds), but worsen five flows (83.03 seconds).

TABLE I
DOWNLOAD TIME MEDIAN AND STANDARD DEVIATION OBTAIN WITH THE IDEAL SCENARIO

Ideal Scenario			TCP/CUBIC median download time (s)	TCP/CUBIC standard deviation (s)	QUIC/BBR median download time (s)	QUIC/BBR standard deviation (s)
without SWF with Hystart		1 flow	21.27	3.81	17.38	0.13
		5 flows	65.67	10.26	76.63	11.18
without SWF without Hystart		1 flow	17.84	0.34	19.24	7.36
		5 flows	65.29	8.73	76.18	11.26
TCP w/o Hystart and QUIC with Hystart	w/o SWF with tunnel	1 flow	17.94	0.34	17.66	0.14
		5 flows	66.73	7.83	75.97	11.93
	$S_{(10,100)}$	1 flow	19.04	0.34	19.23	0.15
		5 flows	73.02	10.28	86.15	13.86
	$S_{(5,100)}$	1 flow	20.17	0.32	20.73	0.14
		5 flows	79.78	9.00	93.98	15.47
	$S_{(2,100)}$	1 flow	24.12	0.29	25.32	0.10
		5 flows	96.26	9.67	116.54	17.44

TABLE II
DOWNLOAD TIME MEDIAN (IN SECONDS) AND PERCENTAGE DIFFERENCE BETWEEN THE WITHOUT SWF (W/O SWF) CONFIGURATIONS AND SWF CONFIGURATION, ON IDEAL, OPTICAL SATELLITE AND THE DVB SATELLITE - MOBILE RECEPTOR SCENARIOS.

Download time median in seconds		1 flow				5 flows			
		w/o SWF	$S_{(10,100)}$	$S_{(5,100)}$	$S_{(2,100)}$	w/o SWF	$S_{(10,100)}$	$S_{(5,100)}$	$S_{(2,100)}$
Ideal scenario	TCP/CUBIC	17.84	19.04 **+6.73%**	20.17 **+13.06%**	24.12 **+35.20%**	65.67	73.02 **+11.19%**	79.78 **+21.49%**	96.26 **+46.58%**
	QUIC/BBR	17.38	19.23 **+10.64%**	20.73 **+19.28%**	25.32 **+45.68%**	76.63	86.15 **+12.42%**	93.98 **+22.64%**	116.54 **+52.08%**
Optical satellite without Wi-Fi	TCP/CUBIC	91.42	54.03 **-40.89%**	24.42 **-74.38%**	24.55 **-73.15%**	163.35	121.99 **-25.32%**	83.99 **-48.58%**	96.89 **-40.68%**
	QUIC/BBR	19.90	21.29 **+6.99%**	21.72 **+9.15%**	25.53 **+28.28%**	80.90	91.44 **+13.03%**	94.46 **+16.76%**	116.86 **+44.46%**
Optical satellite with Wi-Fi	TCP/CUBIC	244.01	72.37 **-70.34%**	36.30 **-85.12%**	24.19 **-90.09%**	274.18	145.91 **-46.78%**	100.61 **-63.31%**	98.24 **-64.17%**
	QUIC/BBR	20.61	20.92 **+1.53%**	22.30 **+8.23%**	25.45 **+23.50%**	80.44	88.93 **+10.55%**	96.15 **+19.53%**	113.78 **+41.44%**
DVB satellite - mobile receptor without Wi-Fi	TCP/CUBIC	405.91	291.02 **-28.30%**	31.11 **-92.34%**	24.67 **-93.92%**	454.30	313.26 **-31.04%**	88.61 **-80.61%**	96.74 **-78.71%**
	QUIC/BBR	23.56	23.67 **+0.45%**	23.68 **+0.51%**	25.51 **+8.28%**	83.49	93.56 **+12.07%**	87.78 **+17.13%**	114.67 **+37.35%**
DVB satellite - mobile receptor with Wi-Fi	TCP/CUBIC	507.10	416.56 **-17.85%**	83.82 **-83.47%**	24.52 **-95.16%**	547.56	444.55 **-18.81%**	135.51 **-75.25%**	97.13 **-82.23%**
	QUIC/BBR	24.96	24.98 **+0.07%**	24.09 **-3.50%**	25.35 **+1.54%**	84.76	92.86 **+9.56%**	102.92 **+21.41%**	115.18 **+35.88%**

B. DVB mobile scenario

We recall that in this scenario losses appear as regular bursts (cf. III-B).

1) DVB satellite - mobile receptor without Wi-Fi: Unprotected TCP/CUBIC flows are highly affected by the previous scenario, they fall even more in the DVB satellite - mobile receptor scenario. With one flow, only the $S_{(2,100)}$ configuration reduces the download times to the ideal scenario level. In the case of five flows, less redundancy, like the $S_{(5,100)}$ configuration, is enough.On the other corner, unprotected QUIC/BBR results are still not that far from the ideal ones. As expected, the download time worsen as long as percentage of losses grows. However, this represents a difference ranging from 5 to 6 seconds from the ideal scenario, while the gap for TCP/CUBIC is 380 seconds. As in the previous scenario, SWF does not improve the results. Even though $S_{(10,100)}$ and $S_{(5,100)}$ results are close to the unprotected results for one flow. Nevertheless, it is not the case for five protected flows.

2) DVB satellite - mobile receptor with Wi-Fi: When the Wi-Fi is added to the test, unsurprisingly, unprotected TCP/CUBIC flows results are worse. The $S_{(2,100)}$ redundancy configuration are the only solution which approaches the ideal scenario results. It divides the download time of unprotected flows by a bit more than 20 for one flow, and by more than 5 for five flows. Like in the previous scenarios, additional losses do not affect BBR results. For one flow, redundancy configurations bring little change compared to unprotected flows. The median download time is not impacted, only the result variations reduce as the redundancy increases. Even with that much losses, with five flows, the unprotected QUIC/BBR configuration stays the more efficient one. To try with lower redundancy, we reused the $S_{(20,100)}$ configuration (4.76% of redundancy). This configuration brings little improvement to the unprotected one BBR flow, with 24.20 seconds to download time. For five flows, SWF protection worsen the results for QUIC/BBR.

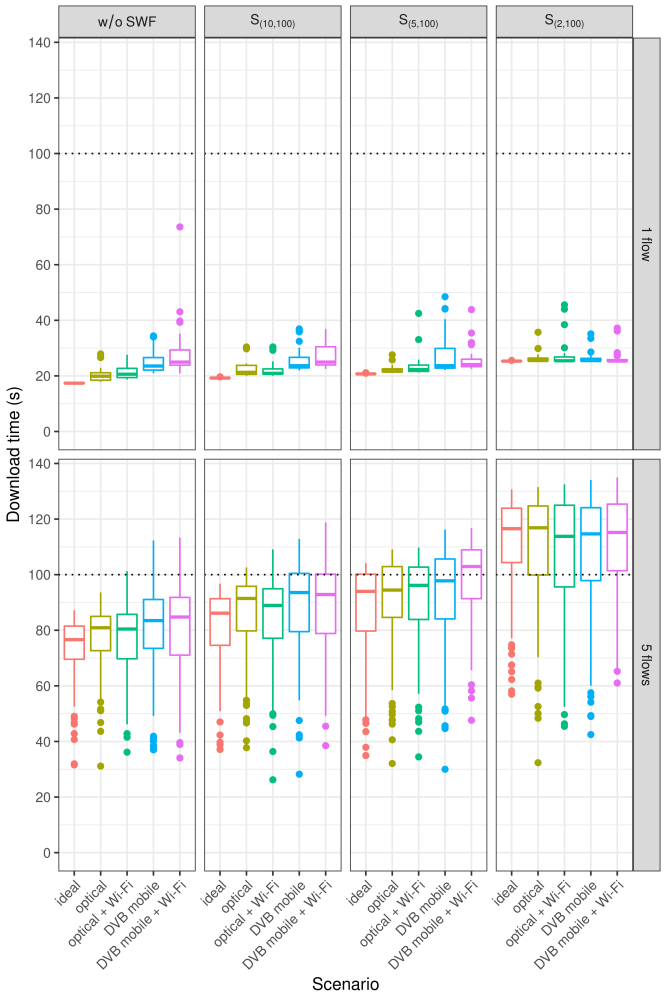

Fig. 7. QUIC/BBR download time results over each scenario and configuration. Number of sample: 30 for 1 flow and 150 for 5 flows.

C. Discussion about our results

The measurements presented allow us to discuss several aspects both on CUBIC and BBR congestion controls. We observe that, unlike CUBIC, BBR flows download time does not get improvement from SWF. Indeed, a loss-based congestion control algorithm, such as CUBIC, identifies all link losses as congestion. Therefore, it might appear obvious that the use of a FEC scheme over CUBIC flows would improve the throughput performance in case of error-link losses. However, we learned that Sliding Window FEC scheme deployed within a tunnel greatly enhances CUBIC performance without complex configuration as for FEC block codes. Furthermore, such tunnel can be easily deployed to protect from loss either on a subnetwork of the path or on the end-to-end network. This solution has been envisioned in [9], [10]. Thus, our study confirms this possibility.

Regarding BBR, this congestion control takes both the loss (considering the loss threshold set as in BBRv2) and the delay as a congestion indication. It should be recalled that BBR evaluates the available bandwidth, based on the RTT evolution, and sends just enough data to reach the available bandwidth, without unnecessarily filling the bottleneck buffer. Therefore, BBR measures a decrease of the available bandwidth when redundancy packets are added. However, BBRv2 implements a loss threshold (default: 2%) which reduces the throughput when reached. We can suppose that SWF could be a solution to protect BBRv2 over unreliable link, which would require some test with BBRv2.

Finally, table III attempts to summarize the potential gain of using SWF in the context of SATCOM.

TABLE III
CONCLUSION ON POTENTIAL GAINS AND COMPROMISES FROM SWF PROTECTION OVER FLOW PERFORMANCE AND DEPLOYMENT

	Gain	Compromise
Performance	- CUBIC performance with one and multiple flows	- Cost of capacity (network point of view) - Does not improve QUIC/BBR - Cost of data (user point of view) - Deleterious on a reliable link or with oversized redundancy
Deployment	- Adapted to encrypted protocol - Could be deployed only on a defaulting segment - Still lot of TCP/CUBIC servers	- Congestion control parameters can be tuned in QUIC - SWF needs an end-to-end tunnel - Unknown number of QUIC/CUBIC servers - Require to apply only on loss based CC

V. CONCLUSION

In this paper, we evaluate the benefit of using SWF tunnel to protect TCP/CUBIC and QUIC/BBR flows from losses and improve their download time over several SATCOM mobile scenarios. The results show that TCP/CUBIC benefits from SWF over unreliable links. SWF protection over TCP/CUBIC can divide by twenty the download time, compared to unprotected flows. Whereas QUIC/BBR is much more resilient to losses and consequently does not gain from SWF protection. Even if this study exhibits little benefits for BBR over SATCOM, finding the right redundancy rate is not trivial and could lead to unfairness along the whole packets path if badly sized. As future work, we propose to study the impact of SWF on BBRv2 flows to put to the test our suspicion of its benefits over the loss threshold.

VI. ACKNOWLEDGEMENTS

The authors would like to thank Jérôme Lacan for several discussions on this work.

REFERENCES

[1] Luis A. Sanchez, Mark Allman, and Dr. Dan Glover. Enhancing TCP Over Satellite Channels using Standard Mechanisms. RFC 2488, January 1999.
[2] 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. Work in Progress.
[3] 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.

[4] Cristian Mogildea, Jörg Deutschmann, Kai-Steffen Hielscher, and Reinhard German. Quic over satellite: Introduction and performance measurements. 09 2019.

[5] Ludovic Thomas, Dubois Emmanuel, Kuhn Nicolas, and Emmanuel Lochin. Google quic performance over a public satcom access. International Journal of Satellite Communications and Networking, 37(6):601–611, November 2019.

[6] Mike Bishop. Hypertext Transfer Protocol Version 3 (HTTP/3). Internet-Draft draft-ietf-quic-http-34, Internet Engineering Task Force, February 2021. Work in Progress.

[7] Nicolas Kuhn, François Michel, Ludovic Thomas, Emmanuel Dubois, Emmanuel Lochin, Francklin Simo, and David Pradas. Quic: Opportunities and threats in satcom. International Journal of Satellite Communications and Networking, n/a(n/a).

[8] Vincent Roca and Ali C. Begen. Forward Error Correction (FEC) Framework Extension to Sliding Window Codes. RFC 8680, January 2020.

[9] Yizhou Li, Xingwang Zhou, Mohamed Boucadair, Jianglong Wang, and Fengwei Qin. LOOPS (Localized Optimizations on Path Segments) Problem Statement and Opportunities for Network-Assisted Performance Enhancement. Internet-Draft draft-li-tsvwg-loops-problem-opportunities-06, Internet Engineering Task Force, July 2020. Work in Progress.

[10] Nicolas Kuhn and Emmanuel Lochin. RFC 8975 Network Coding for Satellite Systems, January 2021. Internet Research Task Force, Request For Comments (RFC) 8975, https://datatracker.ietf.org/doc/rfc8975/.

[11] Vincent Roca, François Michel, Ian Swett, and Marie-Jose Montpetit. Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Schemes for QUIC. Internet-Draft draft-roca-nwcrg-rlc-fec-scheme-for-quic-03, Internet Engineering Task Force, March 2020. Work in Progress.

[12] Nicolas Kuhn, Emmanuel Lochin, François Michel, and Michael Welzl. Coding and congestion control in transport. Internet-Draft draft-irtf-nwcrg-coding-and-congestion-09, Internet Engineering Task Force, June 2021. Work in Progress.

[13] Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, and Ben Leong. The Great Internet TCP Congestion Control Census. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 3(3):1–24, December 2019.

[14] Sebastian Endres, Jörg Deutschmann, Kai-Steffen Hielscher, and Reinhard German. Performance of quic implementations over geostationary satellite links, 2022.

[15] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, May 2021.

[16] Johannes Zirngibl, Philippe Buschmann, Patrick Sattler, Benedikt Jaeger, Juliane Aulbach, and Georg Carle. It's Over 9000: Analyzing Early QUIC Deployments with the Standardization on the Horizon. page 15, 2021.

[17] picoquic. https://github.com/private-octopus/picoquic.

[18] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: A new tcp-friendly high-speed tcp variant. SIGOPS Oper. Syst. Rev., 42(5):64–74, July 2008.

[19] Sangtae Ha and Injong Rhee. Taming the elephants: New tcp slow start. Comput. Netw., 55(9):2092–2110, June 2011.

[20] Romuald Corbel, Stéphane Tuffin, Annie Gravey, Arnaud Braud, and Xavier Marjou. Impact of quic on fairness in mobile networks. pages 82–89. 2019 10th International Conference on Networks of the Future (NoF), 2019.

[21] Eneko Atxutegi, Åke Arvidsson, Fidel Liberal, Karl-Johan Grinnemo, and Anna Brunstrom. TCP Performance over Current Cellular Access: A Comprehensive Analysis, pages 371–400. Springer International Publishing, Cham, 2018.

[22] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR Congestion-Based Congestion Control. page 34.

[23] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Priyaranjan Jha, Yousuk Seung, Kevin Yang, Ian Swett, Victor Vasiliev, Bin Wu, Luke Hsiao, Matt Mathis, and Van Jacobson. BBR v2: A Model-based Congestion Control Performance Optimizations. page 32.

[24] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Ian Swett, and Van Jacobson. BBR Congestion Control. Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-01, Internet Engineering Task Force, November 2021. Work in Progress.

[25] 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.

[26] 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, Huntsville, AL, USA, December 2018. IEEE.

[27] Saahil Claypool, Jae Chung, and Mark Claypool. Measurements Comparing TCP Cubic and TCP BBR over a Satellite Network. In 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC), pages 1–4, Las Vegas, NV, USA, January 2021. IEEE.

[28] M. Watson, A. Begen, and V. Roca. Forward Error Correction (FEC) Framework. RFC 6363 (Proposed Standard), October 2011.

[29] Pierre Ugo Tournoux, Emmanuel Lochin, Jérôme Lacan, Amine Bouabdallah, and Vincent Roca. On-the-fly erasure coding for real-time video applications. IEEE Transactions on Multimedia, 13(4):797–812, 2011.

[30] E. Martinian and C.-E.W. Sundberg. Burst erasure correction codes with low decoding delay. IEEE Transactions on Information Theory, 50(10):2494–2502, 2004.

[31] Opensand. https://www.opensand.org/.

[32] E. Dubois et al. Opensand, an open source satcom emulator. Kaconf, 2017.

[33] Antoine Auger, Emmanuel Lochin, and Nicolas Kuhn. Making Trustable Satellite Experiments: an Application to a VoIP Scenario. In 89th IEEE Vehicular Technology Conference, pages 1–5, Kuala Lumpur, Malaysia, April 2019. IEEE.

[34] Openbach. https://www.openbach.org/.

[35] iperf3. https://software.es.net/iperf/.

[36] swif-codec. https://github.com/irtf-nwcrg/swif-codec.

[37] Lucien Canuet. Reliability of satellite-to-ground optical communication. PhD thesis, University of Toulouse, Toulouse, France, 2018.

[38] Benjamin Peters, Pinhan Zhao, Jae Won Chung, and Mark Claypool. Tcp hystart performance over a satellite network. In In Proceedings of the 0x15 NetDev Conference, Virtual Conference, July 2021.