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

Aitor Martin
Naeem Khademi

Department of Electrical Engineering and Computer Science
University of Stavanger, Norway
Introduction
Introduction

- Geosynchronous Satellite Communication (GEO SATCOM) networks are becoming popular candidates for providing broadband Internet connectivity in novel 5G/6G use-cases.

- In parallel, we are witnessing major breakthrough on the Internet transport layer:
  1) standardization and deployment of QUIC, a general-purpose transport protocol that is
     a) fully encrypted
     b) deployed on the user space over UDP
  2) development of modern congestion control (CC), i.e. BBR, aiming to optimize bandwidth utilization while minimizing network latency

- To this date, around 8.3% of websites are already using QUIC [1]. Therefore, we expect an increase of QUIC traffic over SATCOM.

1. **Problem statement**

**CHALLENGES FOR THE TRANSPORT LAYER INTRODUCED BY SATCOM LINKS**

mainly caused by

1. long propagation delay
2. propagation errors
3. bandwidth asymmetry

**HOW TO MITIGATE THE IMPACT OF THESE CHALLENGES?**

TCP traffic is usually optimized with **Performance-Enhancing Proxies (PEPs)**

However, **QUIC**’s full encryption disables PEP optimizations!

Several studies have shown that **TCP-PEP outperforms QUIC greatly** [2], even with QUIC’s fast handshake

Raised interest in boosting QUIC performance over SATCOM through protocol mechanisms

**We investigate the use of BBR congestion control**

---

Our contribution

In this work, we investigate the following items:

1. the **general performance** of QUIC with BBRv2 over GEO SATCOM
2. other important aspects of **CC performance** (e.g., intra- and inter-protocol fairness, latecomer fairness, etc.) over long-haul satellite links
3. the impact of **packet loss** and **bandwidth asymmetry**
4. the impact of **QUIC implementation choice**
2

Background
Challenges introduced by SATCOM links

1. **Long Round-Trip Time (RTT) ~ 600 ms**
   - Long protocol feedback
   - Slower CC convergence
   - Delayed loss detection and recovery
   - High Bandwidth-Delay Product (BDP) → Larger buffers needed

2. **Propagation errors** due to e.g. rain fading
   - Can be problematic for loss-based CC (e.g. NewReno or CUBIC)

3. **Bandwidth asymmetry**
   - ACK congestion in the return path can limit forward throughput
Transport Layer over SATCOM

Challenges introduced by SATCOM links

1. **Long Round-Trip Time (RTT) ~ 600 ms**
   - Long protocol feedback
   - Slower CC convergence
   - Delayed loss detection and recovery
   - High Bandwidth-Delay Product (BDP) → Larger buffers needed

2. **Propagation errors** due to e.g. rain fading
   - Can be problematic for loss-based CC (e.g. NewReno or CUBIC)

3. **Bandwidth asymmetry**
   - ACK congestion in the return path can limit forward throughput

Proposed Solutions

- Satellite-optimized CC (larger IW, faster slow start)
- BDP Frame extension [3]
- FEC for QUIC
- Model-based CC (e.g. BBR)
- ACK Frequency Extension [4]

---


Transport Layer over SATCOM

Challenges introduced by SATCOM links

1. **Long Round-Trip Time (RTT) ~ 600 ms**
   - Long protocol feedback
   - Slower CC convergence
   - Delayed loss detection and recovery
   - High Bandwidth-Delay Product (BDP) → Larger buffers needed

2. **Propagation errors** due to e.g. rain fading
   - Can be problematic for loss-based CC (e.g. NewReno or CUBIC)

3. **Bandwidth asymmetry**
   - ACK congestion in the return path can limit forward throughput

Proposed Solutions

- Satellite-optimized CC (larger IW, faster slow start)
- BDP Frame extension [3]
- FEC for QUIC
- Model-based CC (e.g. BBR)
- ACK Frequency Extension [4]

- The goal of BBR: find **optimal pacing rate** to maximize link utilization while keeping path RTT as low as possible
  - How? Measuring path **Bottleneck Bandwidth** and **RTT**.

- Previous studies show that BBR tends to beat CUBIC over lossy paths [5]

However, **three main issues** were found with the first version of BBR [6]

1. Unfairness between parallel BBR flows
2. Aggresiveness against parallel loss-based CC flows
3. RTT unfairness

Towards BBRv2

2019 - an **update to BBR is proposed**, named **BBRv2**, introducing a more complex bandwidth probing mechanism, aiming to solve the previously mentioned issues

- BBRv2 **also reacts to packet loss and ECN**

Source: BBR IETF112 (ICCRG)
## Related studies

<table>
<thead>
<tr>
<th>Year</th>
<th>Study</th>
<th>Transport</th>
<th>Scenario</th>
<th>Congestion Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016</td>
<td>Cardwell et al.</td>
<td>TCP</td>
<td>Terrestrial</td>
<td>BBRv1, CUBIC</td>
</tr>
<tr>
<td>2017</td>
<td>Hock et al.</td>
<td>TCP</td>
<td>Terrestrial</td>
<td>BBRv2, BBRv1, CUBIC</td>
</tr>
<tr>
<td>2018</td>
<td>Scholz et al.</td>
<td>TCP</td>
<td>Terrestrial</td>
<td>BBRv2, BBRv1, CUBIC</td>
</tr>
<tr>
<td>2019</td>
<td>Jaeger et al.</td>
<td>TCP</td>
<td>Terrestrial</td>
<td>BBRv2, BBRv1, CUBIC</td>
</tr>
<tr>
<td>2020</td>
<td>Gomez et al.</td>
<td>TCP</td>
<td>Terrestrial</td>
<td>BBRv2, BBRv1, CUBIC</td>
</tr>
<tr>
<td>2020</td>
<td>Song et al.</td>
<td>TCP</td>
<td>Terrestrial</td>
<td>BBRv2, BBRv1, CUBIC</td>
</tr>
<tr>
<td>2021</td>
<td>Song et al.</td>
<td>TCP</td>
<td>SATCOM</td>
<td>BBRv1, CUBIC</td>
</tr>
<tr>
<td>2022</td>
<td>Yang et al.</td>
<td>TCP</td>
<td>SATCOM</td>
<td>BBRv1, CUBIC, PCC, Hybla</td>
</tr>
<tr>
<td>2021</td>
<td>Claypool et al.</td>
<td>TCP</td>
<td>SATCOM</td>
<td>BBRv1, CUBIC</td>
</tr>
<tr>
<td>2022</td>
<td>Zhao et al.</td>
<td>TCP</td>
<td>SATCOM</td>
<td>BBRv1, CUBIC</td>
</tr>
<tr>
<td>2018</td>
<td>Wang et al.</td>
<td>QUIC</td>
<td>SATCOM</td>
<td>BBRv1, CUBIC</td>
</tr>
<tr>
<td>2022</td>
<td><strong>Our paper</strong></td>
<td>QUIC</td>
<td>SATCOM</td>
<td><strong>BBRv2</strong>, BBRv1, CUBIC</td>
</tr>
</tbody>
</table>
A **high amount of QUIC implementations** out there, developed by different agents:

- Web and CDN service providers, e.g. LiteSpeed, Akamai, Cloudflare
- Big technological companies, e.g. Google, Facebook, Apple, Microsoft
- IETF/IRTF contributors, e.g., ngtcp2, picoquic

- *Robin Marx et al. (2020)* [7], *Sebastian Endres et al. (2022)* [8] have reported high heterogeneity among implementations
- Even though RFC9002 specifies a CC mechanism similar to NewReno for QUIC, **many stacks implement CUBIC and BBR**

Experimental Testbed Setup
Experimental Testbed Setup

UiS Testbed based on TEACUP

Flexible experiment design and test automation

For this work, TEACUP was extended for QUIC and QLOG support

Experimental Testbed Setup

- Link emulation with netem/tc

<table>
<thead>
<tr>
<th></th>
<th>SAT</th>
<th>TERR</th>
</tr>
</thead>
<tbody>
<tr>
<td>One Way Delay (OWD)</td>
<td>300 ms</td>
<td>50 ms</td>
</tr>
<tr>
<td>Downlink Bandwidth</td>
<td>20 Mbps</td>
<td>20 Mbps</td>
</tr>
<tr>
<td>Uplink Bandwidth</td>
<td>20/2 Mbps</td>
<td>20 Mbps</td>
</tr>
<tr>
<td>Bottleneck Buffer Size</td>
<td>0.25</td>
<td>0.5</td>
</tr>
<tr>
<td>Packet Loss Ratio (PLR)</td>
<td>0%, 0.1%, 1%</td>
<td></td>
</tr>
</tbody>
</table>

- Two QUIC implementations:
  - ngtcp2 – allows to experiment with BBRv2
  - picoquic – reported good performance over SATCOM [8]

Results
Results

We set up four different experimental scenarios:

1. Single-Flow Bulk Download
2. Multi-Flow Fairness
3. Latecomer Issue
4. Mice versus Elephant flows

All the experiments are run 10 times
We set up four different experimental scenarios:

1. **Single-Flow Bulk Download**
2. **Multi-Flow Fairness**
3. **Latecomer Issue**
4. **Mice versus Elephant flows**

*Run duration: 120 seconds*

*Implementations: ngtcp2 and picoquic*
Results: Bulk Download

Symmetric link (20/20), no packet loss

<table>
<thead>
<tr>
<th>Goodput (Mbps)</th>
<th>Bottleneck Buffer Size (BDP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>picoquic</td>
<td>0.25 0.5 1 2</td>
</tr>
<tr>
<td>ngtcp2</td>
<td>0.25 0.5 1 2</td>
</tr>
</tbody>
</table>

SAT scenario

TERR scenario
Results: Bulk Download

Symmetric link (20/20), no packet loss
Results: Bulk Download with packet loss

Symmetric link (20/20), PLR = 0.1%

Symmetric link (20/20), PLR = 1%
Results: Bulk Download with packet loss

Symmetric link (20/20), PLR = 0.1%

Symmetric link (20/20), PLR = 1%

ngtcp2 – cubic performance drops dramatically!
Results: Bulk Download with packet loss

Symmetric link (20/20), PLR = 1%

Symmetric link (20/20), PLR = 0.1%

Long RTT + high packet loss reduce BBRv2 performance!
Results: Bulk Download with uplink traffic

Now, we introduce **cross-traffic in the uplink**, and measure forward goodput over symmetric and asymmetric bandwidth SATCOM setups

**Average Forward Goodput (Mbps)**

<table>
<thead>
<tr>
<th>ngtcp2</th>
<th>Symmetric 20/20</th>
<th>Asymmetric 20/2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BBRv2</td>
<td>BBRv1</td>
</tr>
<tr>
<td>Down</td>
<td>Up</td>
<td></td>
</tr>
<tr>
<td>BBRv2</td>
<td>14.73</td>
<td>14.94</td>
</tr>
<tr>
<td>BBRv1</td>
<td>15.10</td>
<td>14.97</td>
</tr>
<tr>
<td>CUBIC</td>
<td>14.90</td>
<td>13.64</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>picoquic</th>
<th>Symmetric 20/20</th>
<th>Asymmetric 20/2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BBRv2</td>
<td>BBRv1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Down</td>
<td>Up</td>
<td></td>
</tr>
<tr>
<td>BBRv2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BBRv1</td>
<td>-</td>
<td>18.36</td>
</tr>
<tr>
<td>CUBIC</td>
<td>-</td>
<td>18.38</td>
</tr>
</tbody>
</table>
Results: Bulk Download with uplink traffic

Now, we introduce cross-traffic in the uplink, and measure forward goodput over symmetric and asymmetric bandwidth setups.

**Average Forward Goodput (Mbps)**

<table>
<thead>
<tr>
<th>ngtcp2</th>
<th>Symmetric 20/20</th>
<th>Asymmetric 20/2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Down</td>
<td>Up</td>
<td>BBRv2</td>
</tr>
<tr>
<td>BBRv2</td>
<td>14.73</td>
<td>14.94</td>
</tr>
<tr>
<td>BBRv1</td>
<td>15.10</td>
<td>14.97</td>
</tr>
<tr>
<td>CUBIC</td>
<td>14.90</td>
<td>13.64</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>picoquic</th>
<th>Symmetric 20/20</th>
<th>Asymmetric 20/2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Down</td>
<td>Up</td>
<td>BBRv2</td>
</tr>
<tr>
<td>BBRv2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BBRv1</td>
<td>-</td>
<td>18.36</td>
</tr>
<tr>
<td>CUBIC</td>
<td>-</td>
<td>18.38</td>
</tr>
</tbody>
</table>

ngtcp2 performance drops on asymmetric links
Now, we introduce **cross-traffic in the uplink**, and measure forward goodput over symmetric and asymmetric bandwidth setups.

**Average Forward Goodput (Mbps)**

<table>
<thead>
<tr>
<th>ngtcp2</th>
<th>Symmetric 20/20</th>
<th>Asymmetric 20/2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BBRv2</td>
<td>BBRv1</td>
</tr>
<tr>
<td>Down</td>
<td>Up</td>
<td></td>
</tr>
<tr>
<td>BBRv2</td>
<td>14.73</td>
<td>14.94</td>
</tr>
<tr>
<td>BBRv1</td>
<td>15.10</td>
<td>14.97</td>
</tr>
<tr>
<td>CUBIC</td>
<td>14.90</td>
<td>13.64</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>picoquic</th>
<th>Symmetric 20/20</th>
<th>Asymmetric 20/2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BBRv2</td>
<td>BBRv1</td>
</tr>
<tr>
<td>Down</td>
<td>Up</td>
<td></td>
</tr>
<tr>
<td>BBRv2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BBRv1</td>
<td>-</td>
<td>18.36</td>
</tr>
<tr>
<td>CUBIC</td>
<td>-</td>
<td>18.38</td>
</tr>
</tbody>
</table>

*picoquic* remains stable on asymmetric links
We set up four different experimental scenarios:

1. Single-Flow Bulk Download
2. Multi-Flow Fairness
3. Latecomer Issue
4. Mice versus Elephant flows

We use Jain’s Fairness Index (JFI) to measure fairness between parallel flows.

Run duration: 300 seconds
Implementation: ngtcp2
Results: Intra-protocol fairness

The diagram shows the Jain's Fairness Index (JFI) for different protocols and the number of flows. The Y-axis represents the JFI, ranging from 0 to 1. The X-axis indicates the number of flows, ranging from 2 to 64. The protocols compared are BBRv2, BBRv1, and Cubic. The box plots indicate the distribution of JFI for each protocol and number of flows, with the median, quartiles, and outliers visible.
Results: Inter-protocol fairness

A: BBRv2 vs BBRv1
B: BBRv2 vs CUBIC
C: BBRv1 vs CUBIC
Results: Inter-protocol fairness

A: BBRv2 vs BBRv1
B: BBRv2 vs CUBIC
C: BBRv1 vs CUBIC

BBRv2 provides better fairness towards CUBIC!
Results: Inter-protocol fairness

Average goodput ratio achieved by each flow

BBRv2 vs BBRv1

- BBRv1 -1: 40.7%
- BBRv1 -2: 43.1%
- BBRv2 -1: 10.7%
- BBRv2 -2: 5.5%

BBRv2 vs CUBIC

- CUBIC -1: 35.6%
- CUBIC -2: 27.8%
- BBRv2 -1: 17.3%
- BBRv2 -2: 19.3%

BBRv1 vs CUBIC

- BBRv1 -1: 54.1%
- BBRv1 -2: 33.8%
- CUBIC -1: 5.24%
- CUBIC -2: 6.8%
We set up four different experimental scenarios:

1. Single-Flow Bulk Download
2. Multi-Flow Fairness
3. Latecomer Issue
4. Mice versus Elephant flows

Run duration: 300 seconds
Implementation: ngtcp2
Results: Latecomer fairness with CUBIC

CUBIC latecomers converge very slowly on long RTT paths
Results: Latecomer fairness with BBR

SATCOM (RTT = 600 ms)

BBR latecomers converge faster!
Results: Latecomer fairness with BBR

SATCOM (RTT = 600 ms)

BBRv2 latecomers join the link less aggressively
Results: Latecomer fairness with BBR

SATCOM (RTT = 600 ms)

BBRv2 achieves better long-term fairness
Results: Latecomer fairness with BBR

SATCOM (RTT = 600 ms)

But BBRv2 fails to recover available bandwidth after the rest of the flows end

We have seen that this is a consequence of the long RTT
We set up four different experimental scenarios:

1. Single-Flow Bulk Download
2. Multi-Flow Fairness
3. Latecomer Issue
4. Mice versus Elephant flows

Implementation: ngtcp2
Results: Mice vs Elephant Flows

Background Traffic
BBRv1
Discussion and Conclusion
Discussion

Impact of Congestion Control choice

- **BBR provides better performance overall under lossy links**
  - BBRv1 provides the best performance
  - BBRv2 provides better fairness towards itself and towards CUBIC
  - BBRv2 latecomers are less aggressive and still converge fast

- BBRv2 seems to be the on the right path for fairer coexistence with other flows
  - But **BBRv2 performance suffers from the long RTT + packet loss** present in SATCOM links
  - Further BBR iterations could contemplate these long RTT scenarios
Discussion

Impact of bandwidth asymmetry

- A 1:10 asymmetry has proven to be a great challenge for ngtcp2, with great performance drops

- But results have shown that an ACK policy such as picoquic’s can maintain performance
  - picoquic sends around 10 times less ACK frames in our experiment results
  - This stresses further research into optimized ACK strategies for SATCOM networks
Impact of QUIC implementation

- picoquic outperforms ngtcp2 across CC algorithms
  - Better resilience to packet loss
  - Better performance with bandwidth asymmetry
  - Possible reasons
    • Flow control window mechanism
    • ACK policies
5 Conclusion

To summarize:

- **BBR** seems to be a good candidate to yield better performance SATCOM networks
  - BBRv2 adds great improvements to fairness
  - **But** BBRv2 fairness and performance could be further improved for SATCOM-like scenarios (i.e. high BDP and packet loss)

- **Bandwidth asymmetry** is a problem in the absence of satellite-optimized ACK policies

- Picoquic’s satellite-optimizations seem to be key for QUIC
Future directions:

- Improve the experimental setup:
  - Introduce a more realistic satellite model (L1-L2 mechanisms, packet loss models)
  - Use a wider set of QUIC implementations

- Propose and study different ACK strategies under various asymmetric setups
  - Could this be better implemented using MASQUE?
  - How do different CC coexist with these ACK policies?
Thank you! Questions?

Aitor Martin
Naeem Khademi
Department of Electrical Engineering and Computer Science
University of Stavanger, Norway