IMC 2022 Nice, France

Understanding Speciation in QUIC Congestion Control

Ayush Mishra, Sherman Lim, Ben Leong

National University of Singapore



The quick rise of QUIC

Userspace transport stack built over UDP

According to Sandvine, it contributes to 30% downstream traffic in EMEA 16% downstream traffic in North America 75% of Meta's traffic

Standard with HTTP3

The QUIC Revolution

Reinvents many aspects of transport

New 0 RTT Handshake Baked-in encryption Multistreaming support







The QUIC Pragmatism

Ranysha Ware

rware@cs.cmu.edu

Carnegie Mellon University

BBR is a new congestion control algorithm (CCA) deployed for Chromium

OUIC and the Linux kernel. As the default CCA for YouTube (which

commands 11+% of Internet traffic), BBR has rapidly become a major

player in Internet congestion control. BBR's fairness or friendliness to

Most stacks stick to standard congestion control algorithms like CUBIC, Reno and BBR

These algorithms are time-tested and well understood



Matthew K. Mukerjee Srinivasan Seshan mukerjee@nefeli.io srini@cs.cmu.edu Nefeli Networks Carnegie Mellon University

Justine Sherry sherry@cs.cmu.edu Carnegie Mellon University

nearly starving for bandwidth. This phenomena was first explored in [11] and BBRv2 is expected to patch the problem [7].1 In residential capacity links (e.g. 10-100Mbps) with deep buffers, studies [4, 9, 14, 16, 17] have generated conflicting reports on how BBR shares bandwidth with competing Cubic and Reno flows.

Congestion Avoidance and Control Van Jacobson* University of California Lawrence Be Berkel van@hel '86, the Internet had the first of what of 'congestion collapses'. During this Analysis of the Increase and Decrease a throughput from LBL to UC Berketed by 400 yards and three IMP hops) Algorithms for Congestion Avoidance 2 Kbps to 40 bps. Mike Karels¹ and I by this sudden factor-of-thousand drop in Computer Networks d embarked on an investigation of why en so bad. We wondered, in particular ackground gestion in computer networks is becoming CUBIC: A New TCP-Friendly High-Speed TCP Variant portant issue due to the increasing mismatch networks Sangtae Ha, Injong Rhee Lisong Xu Dept of Comp. Sci. and Eng. Dept of Computer Science North Carolina State University University of Nebraska Raleigh, NC 27695 Lincoln, Nebraska 68588 {sha2.rhee}@ncsu.edu xu@cse.unl.edu ABSTRACT "high-speed" TCP variants are proposed (e.g., HSTCP [15], STCP [25], HTCP [28], SQRT CUBIC is a congestion control protocol for TCP (transmis sion control protocol) and the current default TCP algo wood [14], and BIC-TCP [30]). Recognizing th rithm in Linux. The protocol modifies the linear window with TCP, the Linux community responded quick growth function of existing TCP standards to be a cubic plement a majority of these protocols in Linux them as part of its operating system. After a serie function in order to improve the scalability of TCP over party testing and performance validation [11, 2 ABSTRACT from version 2.6.8, it selected BIC-TCP as the de

algorithm and the other TCP variants as options

What makes BIC-TCP stand out from other T

tihms is its stability. It uses a binary search algori

the window grows to the mid-point between t

NEAL CARDWELL

YUCHUNG CHENG

C STEPHEN GUNN

v all accounts. todav's

as well as it should. Most

Internet is not moving data

fast and long distance networks. It also achieves more eq uitable bandwidth allocations among flows with different RTTs (round trip times) by making the window growth to be independent of RTT - thus those flows grow their conges tion window at the same rate. During steady state, CUBIC increases the window size aggressively when the window is

far from the saturation point, and the slowly when it is close



Speciation in CCA implementations in QUIC



Speciation in CCA implementations in QUIC



Speciation in CCA implementations in QUIC

30+ QUIC stacks

Currently there is **no systematic way** to **test** and **validate** QUIC implementations of standard congestion control algorithms

Benchmarking QUIC CC implementations

Goals:

#1 Define and measure similarity between QUIC implementations and their reference kernel implementations

#2 Study interactions between different implementations



Benchmarking QUIC CC implementations

Goals:



Defining Similarity

The fine-grained approach: Compare cwnd graphs



Defining Similarity

The coarse-grained approach: Relative fairness



Problem: Does not capture finer algorithmic differences

Middle ground: Performance Envelope

Key insight: CCAs represent trade offs

Performance Envelope is a multi-dimensional metric

We chose throughput and delay as the two dimensions

Measuring the **Performance Envelope (PE)**



Measuring the **Performance Envelope (PE)**





Performance Envelope (PE)

Conformance

The measure of similarity

Throughput

defined as the ratio of points inside the overlapping region of the two PEs and the total number of sampled points



Performance Envelope (PE)

Conformance

The measure of similarity

Throughput

defined as the ratio of points inside the overlapping region of the two PEs and the total number of sampled points

Reference implementation

QUIC stack X

Deviation The measure of dissimilarity

defined as normalized distance between the centroids of the two PEs

Performance Envelope (PE)

Conformance

The measure of similarity

defined as the ratio of points inside the overlapping region of the two PEs and the total number of sampled points Throughput

QUIC stack X Deviation Reference implementation

Deviation

The measure of dissimilarity

defined as normalized distance between the centroids of the two PEs

Evaluation details



Organization	Stack	CUBIC	BBR	Reno
-	Linux kernel	1	1	✓
Facebook	mvfst[3]	\checkmark	✓	1
Google	chromium[4]	\checkmark	✓	×
Microsoft	msquic [9]	\checkmark	×	×
Cloudflare	quiche [2]	\checkmark	×	\checkmark

Significant Speciation!



Rate-based vs. Window-based



Rate-based vs. Window-based

20 Mbps, 50 ms RTT, Reno Conformance 20.0 mvfst-bbr **Facebook** tcp-bbr improves for rate-14 17.5 chromium-bbr Google based congestion 15.0 Throughput (Mbps) 12 12.5 control algorithms like 10 י 10.0 **BBR** in deeper 7.5 8 **buffers** 5.0 6 2.5 16 18 14 20 22 24 20 40 60 80 100 Queuing Delay (ms) Queuing Delay (ms) Small overlaps in Larger overlaps in 0.5 BDP Buffers **5 BDP Buffers**

20 Mbps, 50 ms RTT, Reno

Ran all possible pairs of implementations and plotted their throughput ratio on a heat map.

If stack X and stack Y compete, then stack X's throughput ratio is

$$\frac{T_x}{T_x + T_y}$$

If the ratio is greater than 0.5, Stack X gets more throughput



¹ BDP Buffer, 20 Mbps, 50 ms RTT



QUIC stacks in general outperform TCP implementations of standard congestion control algorithms

¹ BDP Buffer, 20 Mbps, 50 ms RTT



Facebook's QUIC stack MVFST BBR massively outperforms all other QUIC implementations

1 BDP Buffer, 20 Mbps, 50 ms RTT

MVFST BBR

We found that MVFST BBR's aggression was down to **implementation level differences.**

It applies an additional gain of **120%**

Changing this to **100%** improves conformance



(conformance = **0.8**)

Summary

We introduce the *Performance Envelope*, a new metric for measuring the similarity between implementations of standard congestion control algorithms

We show that there is already significant speciation in QUIC implementations of CUBIC, BBR, and Reno

We demonstrate that QUICbench can identify differences in implementations and help improve their conformance

Future Work

Enhance the *Performance Envelope* metric and evaluate stacks over a variety of network conditions

Evaluate more QUIC stacks

Exporting and verifying key CCA parameters

Read the paper:



Thank you!

Get in touch: ayush@comp.nus.edu.sg

Understanding Speciation in QUIC Congestion Control

Ayush Mishra, Sherman Lim, and Ben Leong National University of Singapore

ABSTRACT

The QUIC standard is expected to replace TCP in HTTP 3.0. While QUIC implements a number of the standard features of TCP differently, most QUIC stacks re-implement standard congestion control algorithms. This is because these algorithms are well-understood and time-tested. However, there is currently no systematic way to ensure that these QUIC congestion control protocols are implemented correctly and predict how these different QUIC implementations will interact with other congestion control algorithms on the Internet.

To address this gap, we present QUICbench, which, to the best

easily modify and push updates to their QUIC stacks. While this flexibility could potentially allow QUIC to become a more secure alternative to TCP, the converse is also true: it also makes it easier to make mistakes.

The QUIC standard, as described by its many prescriptive IETF RFCs and drafts today [5], implements a protocol that is different from TCP. However, existing QUIC stacks [1] still implement the classic congestion control algorithms (CCA) used by TCP instead of inventing new ones. There is a good reason for this. Classic congestion control algorithms are well understood, predictable, and already have good track records of convergence and stability