Containing the **Cambrian Explosion** in QUIC Congestion Control

Ayush Mishra, Ben Leong

National University of Singapore

Internet Measurements Conference, 2023

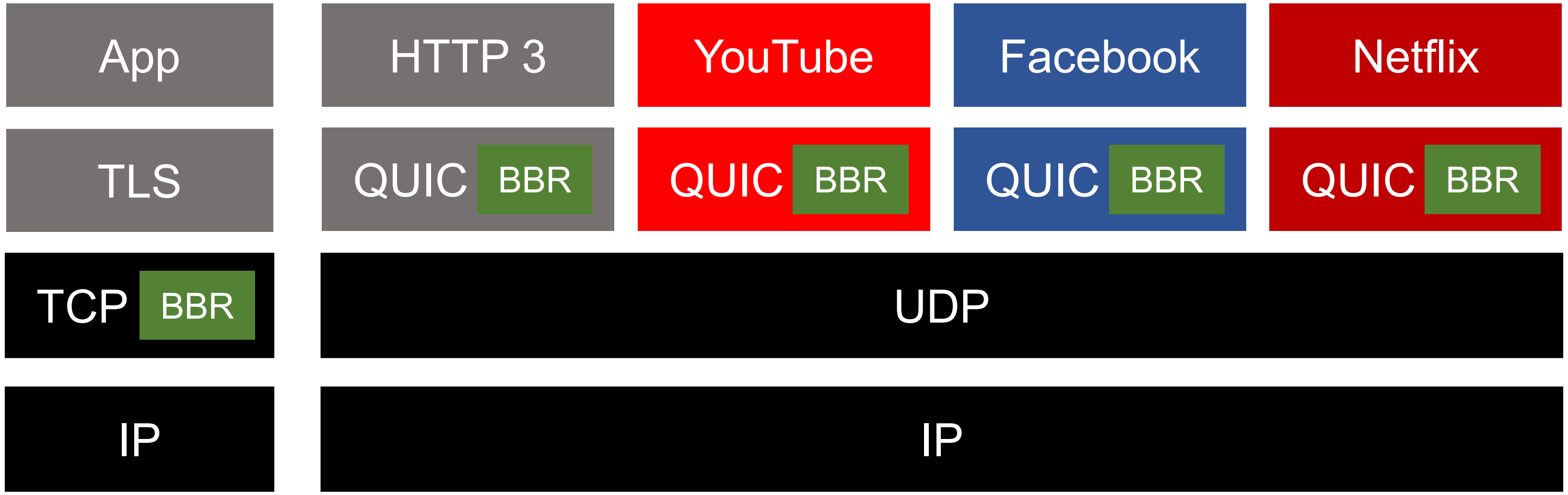
Montreal, Canada

The Internet Congestion Control Landscape is already very **heterogeneous**.

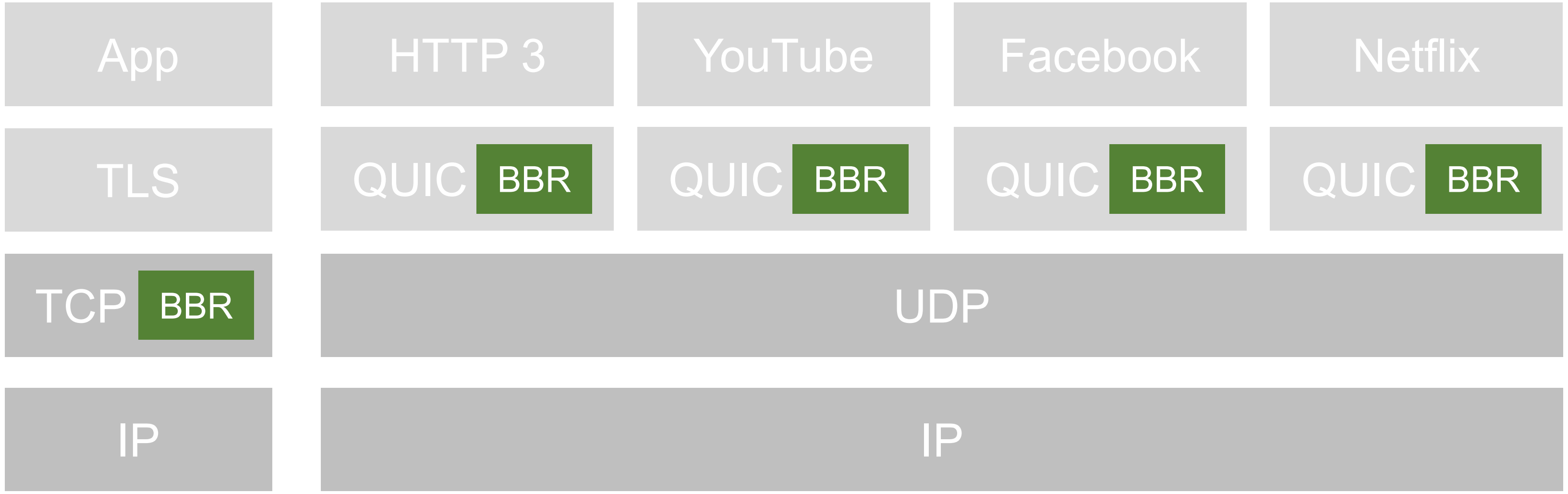
Managing this **heterogeneity** is an important problem.

The introduction of **just one new Congestion Control Algorithm (CCA)** back in 2016 forced us to **rethink** Buffer Sizing, Fairness, & Stability.

This **heterogeneity** risks being **increased** with the deployment of **QUIC**



This **heterogeneity** risks being **increased** with the deployment of **QUIC**



There is a **low barrier to the modification** of these re-implementations of standard CCAs.

(Meta)

Earlier investigations into **mvfst** and **Chromium** have already shown this is true

(Google)

Using a new metric called the **Performance Envelope**, we uncovered modified implementations of **CUBIC** in Chromium and **BBR** in mvfst

[IMC ’22]

Check for updates

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 implemen-

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

Our goals in this measurement study

1. Conduct a more **extensive measurement study** of all the actively deployed open source QUIC stacks.
2. Make improvements to the **Performance Envelope** metric
3. Provide **hints** on fixing implementations with low conformance.

1. A more **extensive** measurement study

Organization	Stack	CUBIC	BBR	Reno
Linux kernel	TCP	✓	✓	✓
Facebook	mvfst [6]	✓	✓	✓
Google	chromium [8]	✓	✓	✗
Microsoft	msquic [12]	✓	✗	✗
Cloudflare	quiche [5]	✓	✗	✓
LiteSpeed	lsquic [11]	✓	✓	✗
Go	quicgo [9]	✓	✗	✓
H2O	quicly [10]	✓	✗	✓
Rust	quinn [14]	✓	✗	✓
Amazon Web Services	s2n-quic [4]	✓	✗	✗
Alibaba	xquic [3]	✓	✓	✓
Mozilla	neqo [13]	✓	✗	✓

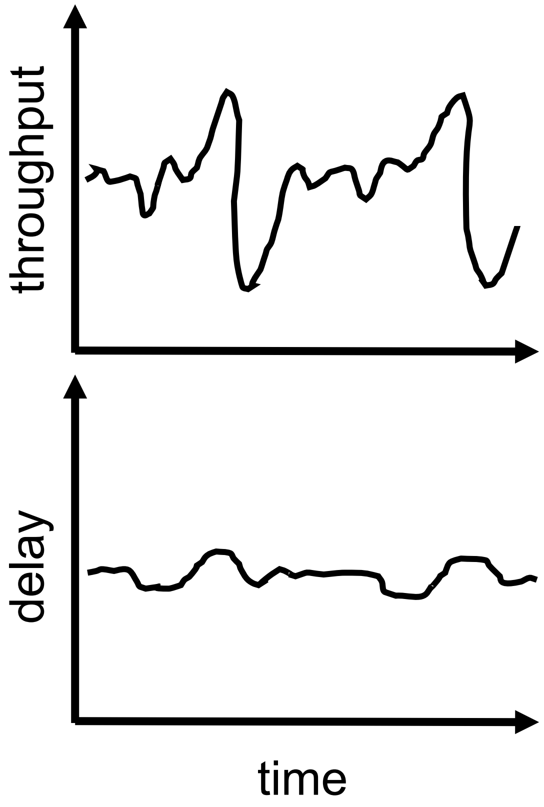
Benchmarked all QUIC stacks that were **deployed**, **open source**, and **implemented some congestion control algorithm**.

2. Improving the Performance Envelope metric

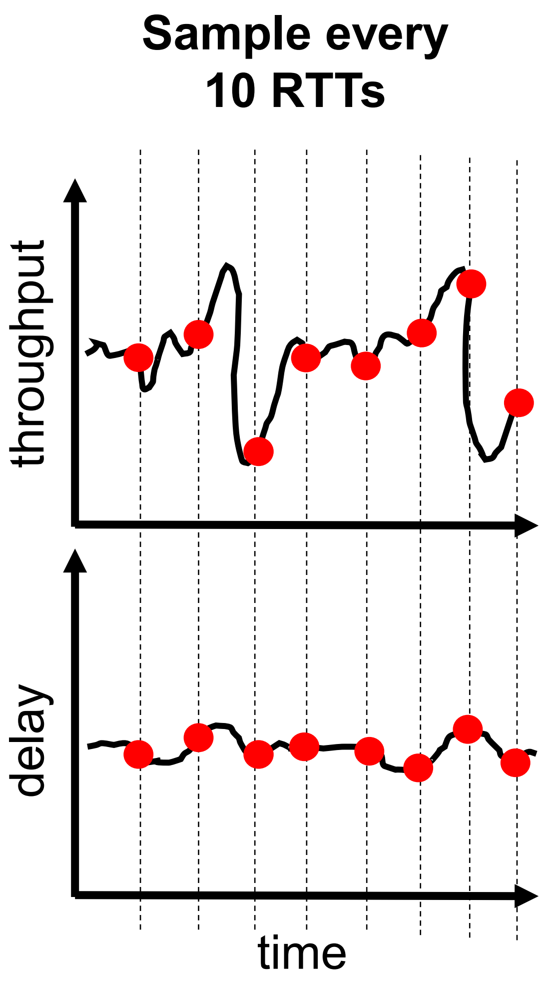
Recap:

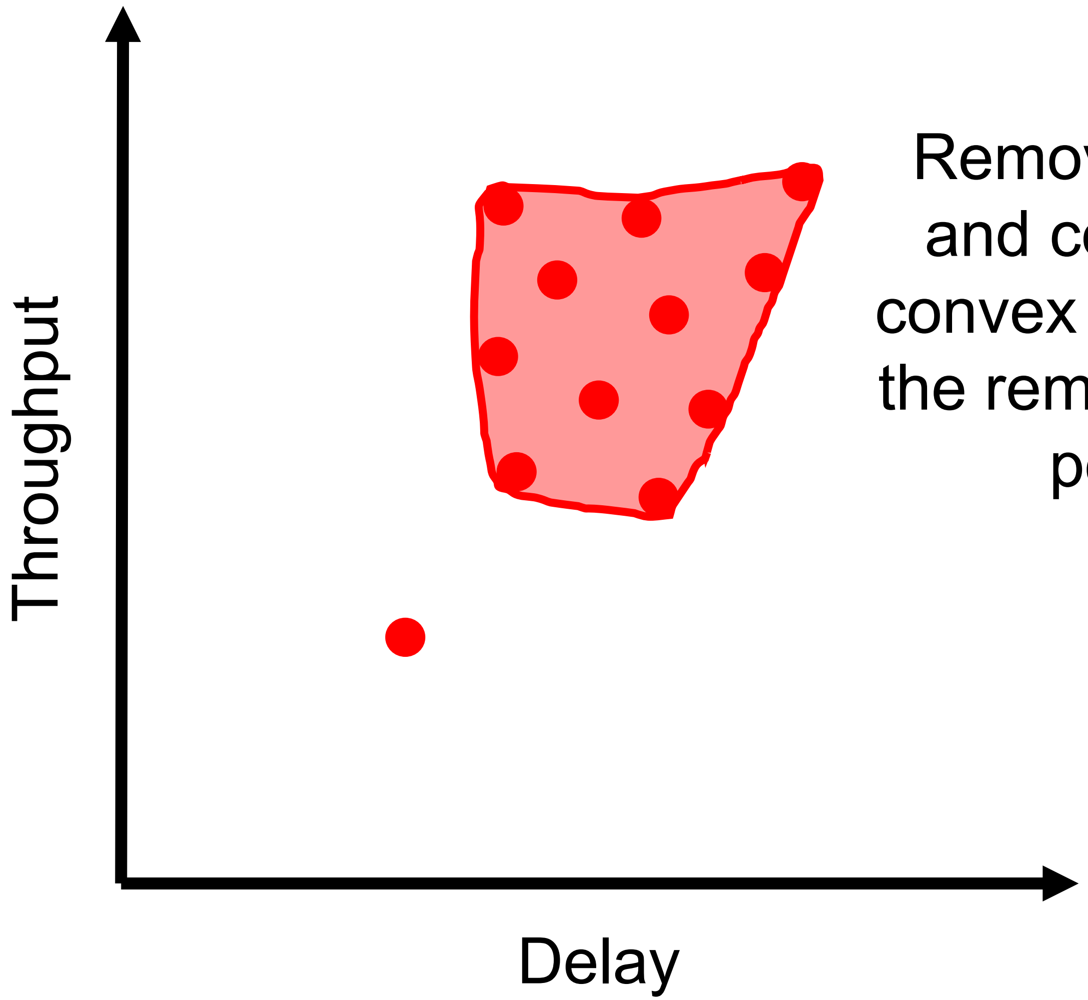
The Performance Envelope (PE)
was a way to capture the
throughput-delay trace-off space
for an implementation in a given
network environment.

2. Improving the Performance Envelope metric



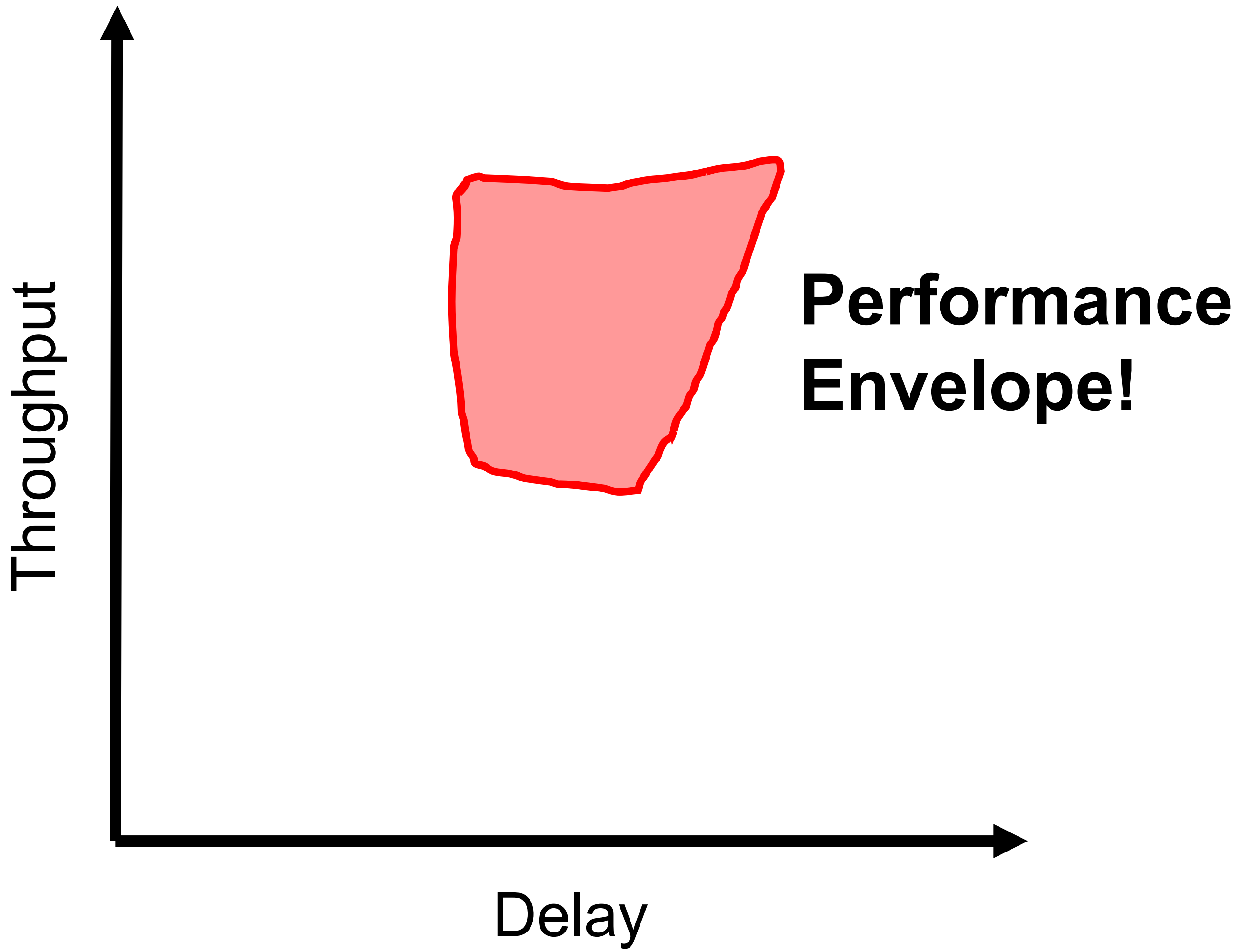
2. Improving the Performance Envelope metric





Remove outliers and construct a convex hull around the remaining data points.

2. Improving the Performance Envelope metric

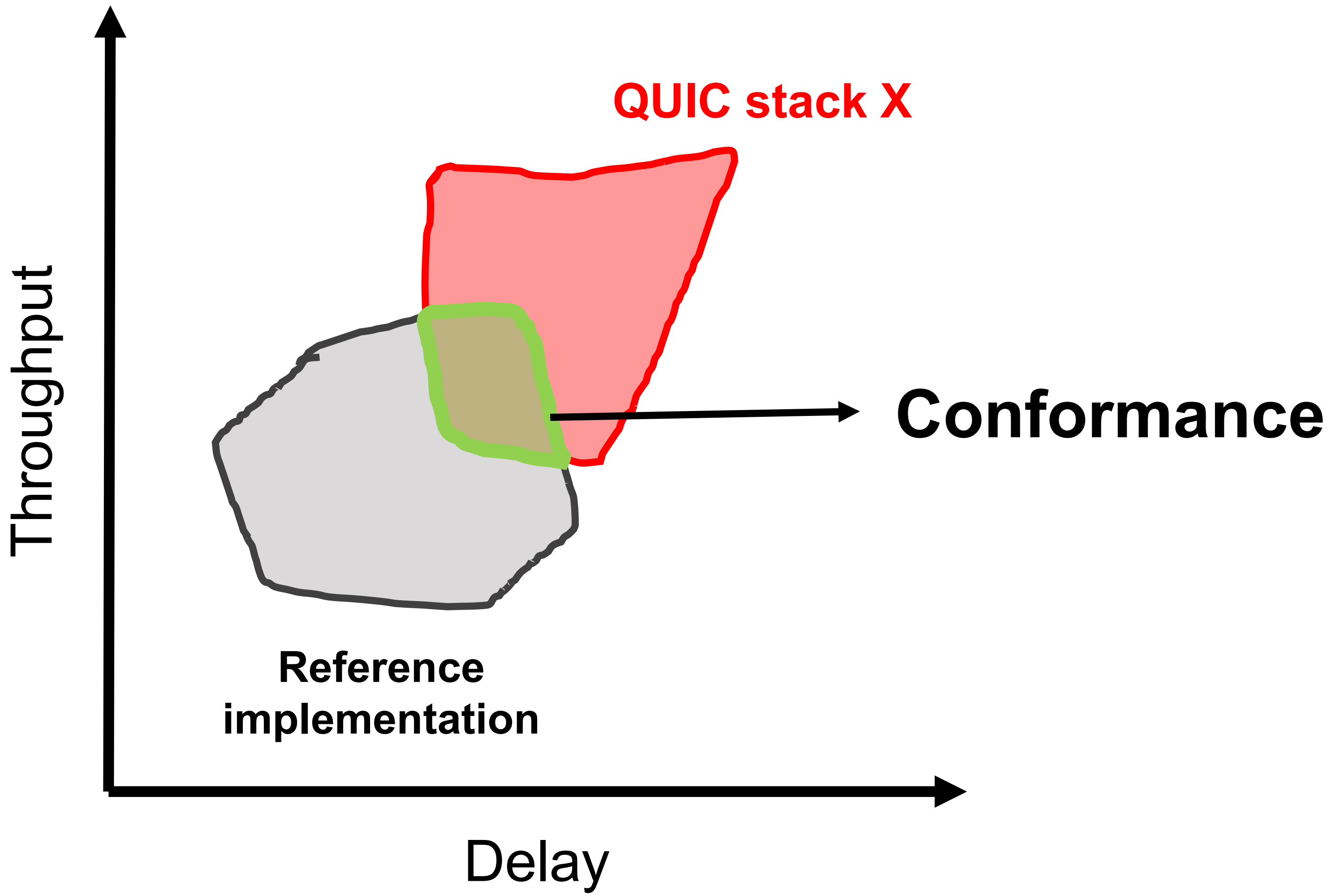


2. Improving the Performance Envelope metric

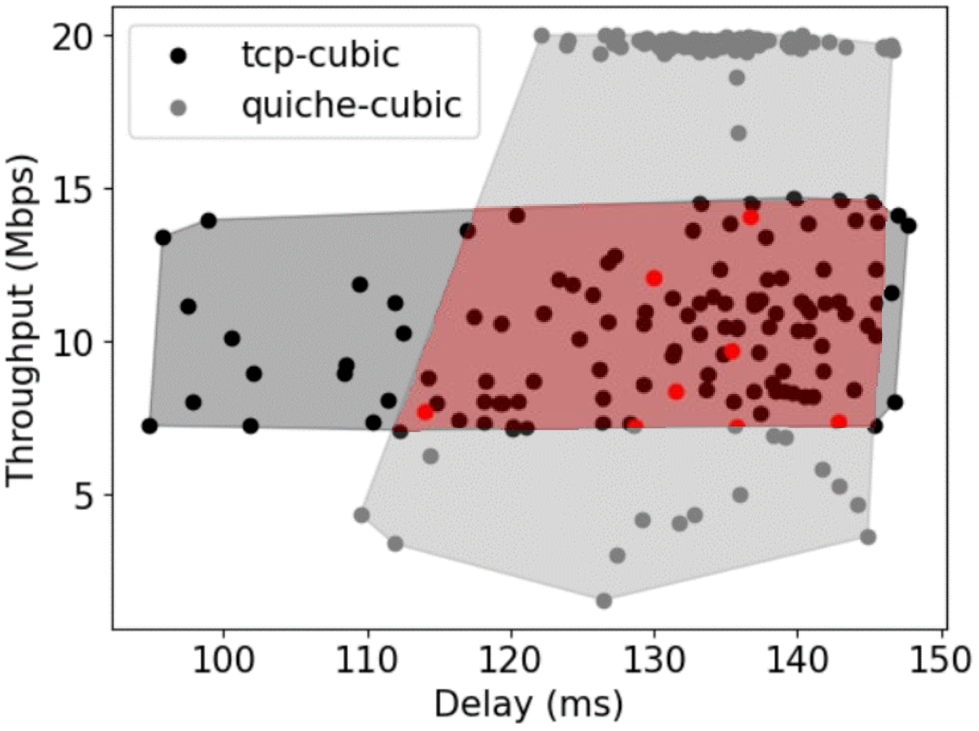
Conformance

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



2. Improving the Performance Envelope metric



Conf = 0.63

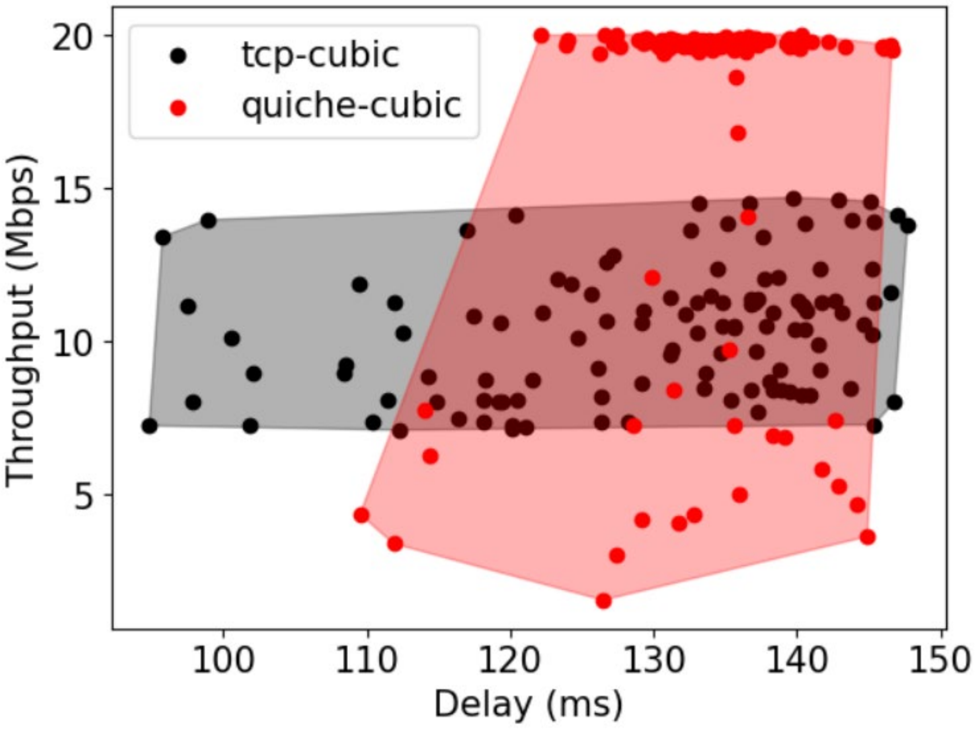
There is a clear problem with having only **one** convex hull per PE.

The overlap captured can often not be very meaningful.

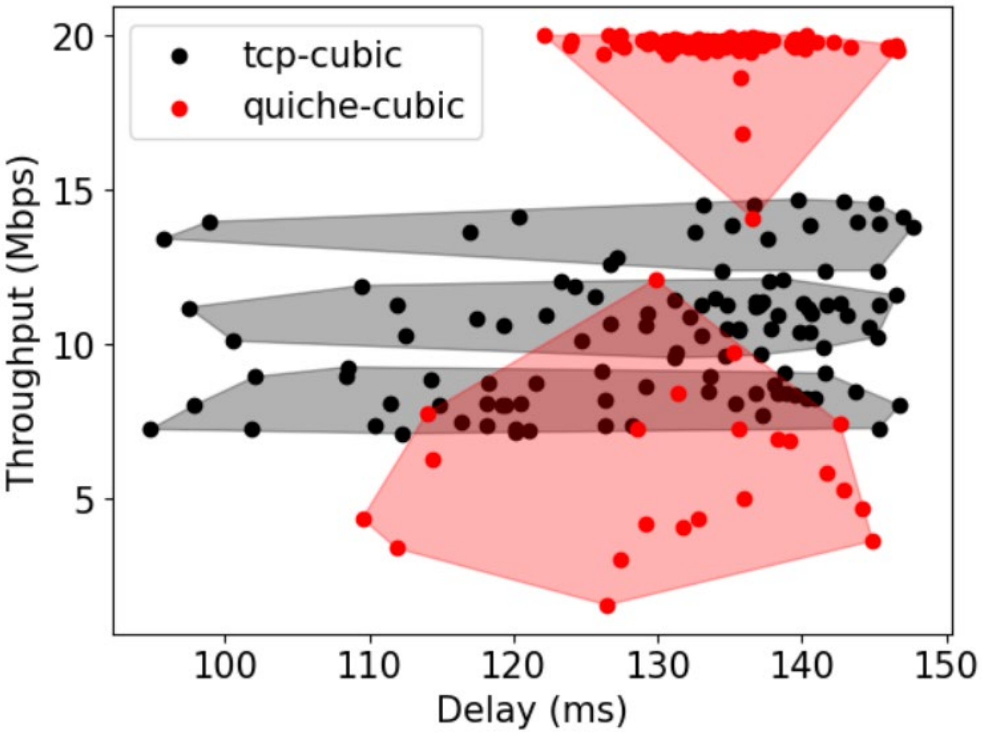
The old definition can often **overestimate** how conformant an implementation is.

2. Improving the Performance Envelope metric

Having multiple clusters solves this problem!

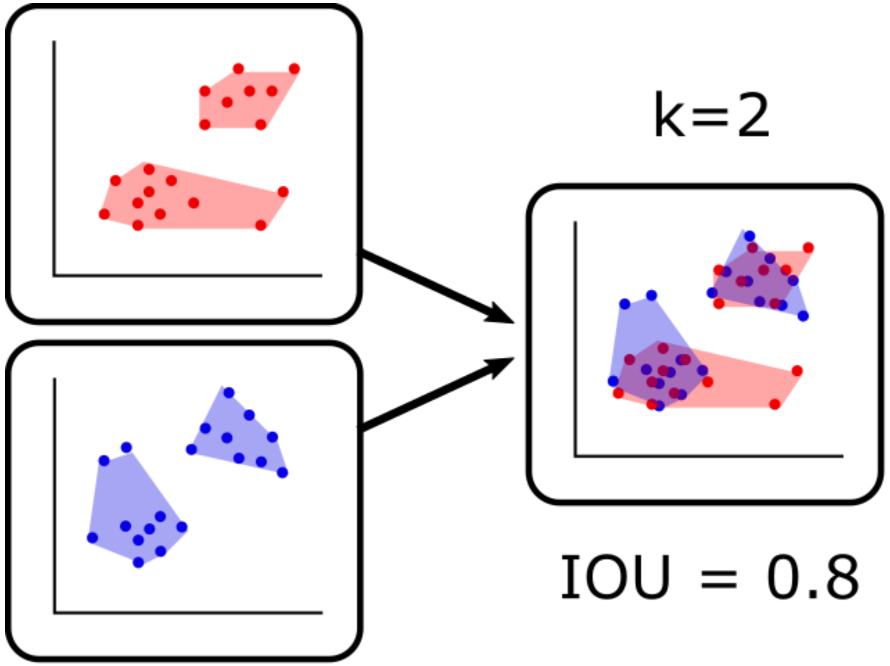


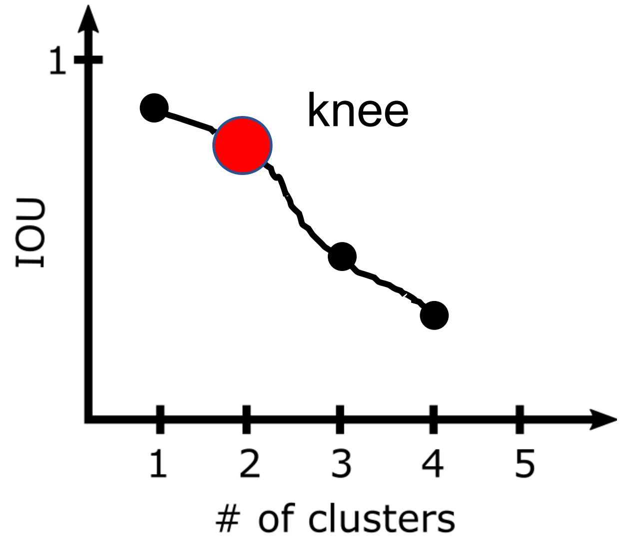
Conf = 0.63



Conf = 0.12

Determining a PE’s natural number of clusters





We found **5 new** QUIC CCA implementations showing **low conformance**.

Stack	Type	Conf
chromium[b]	CUBIC	0.6
neqo	CUBIC	0
quiche	CUBIC	0.08
xquic	CUBIC	0.55
mvfst[b]	BBR	0
xquic	BBR	0.15
xquic	Reno	0.38

a: old definition of Conformance
b: already shown to be non-conformant by a previous study.

The new definition of Conformance is a lot **better at highlighting** non-conformant implementations, especially for **quiche CUBIC** and **xquic BBR**.

We found **5 new** QUIC CCA implementations showing **low conformance**.

Stack	Type	Conf-old[a]	Conf
chromium[b]	CUBIC	0.65	0.6
neqo	CUBIC	0	0
quiche	CUBIC	0.48	0.08
xquic	CUBIC	0.6	0.55
mvfst[b]	BBR	0	0
xquic	BBR	0.37	0.15
xquic	Reno	0.43	0.38

a: old definition of Conformance
b: already shown to be non-conformant by a previous study.

The new definition of Conformance is a lot **better at highlighting** non-conformant implementations, especially for **quiche CUBIC** and **xquic BBR**.

All CCA implementations in the **xquic** stack show low conformance, pointing to a **possible stack-level issue**.

We found **5 new** QUIC CCA implementations showing **low conformance**.

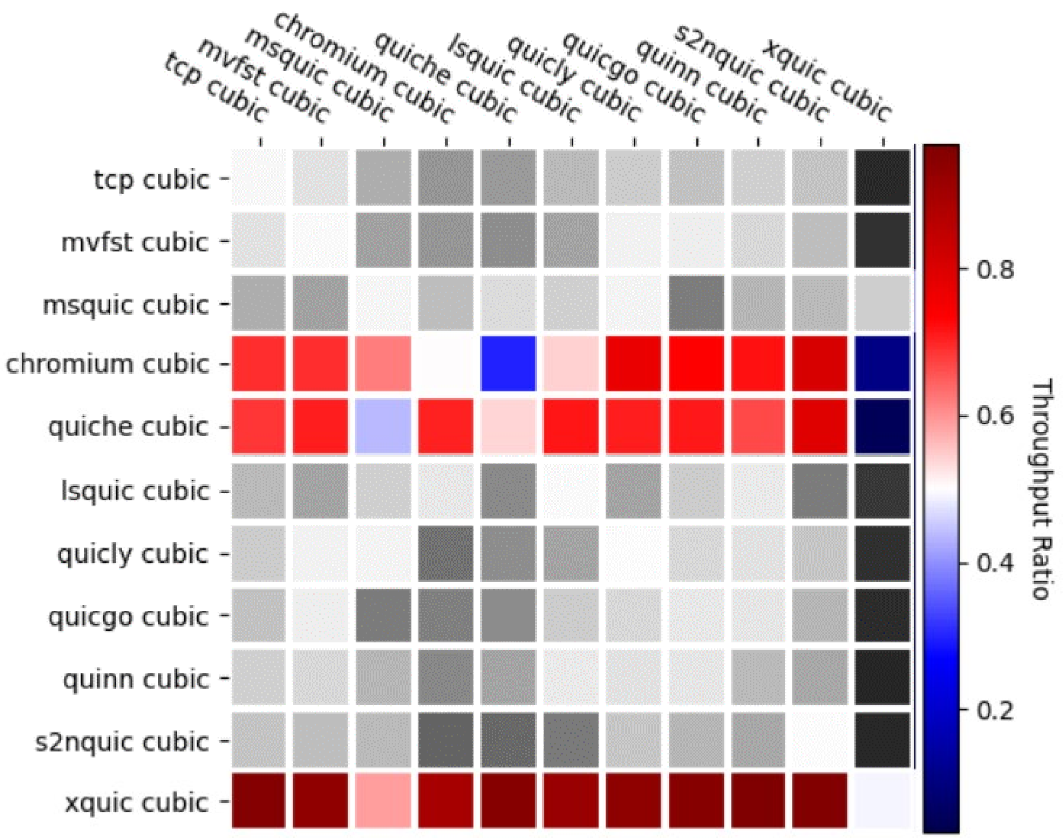
Stack	Type	Conf-old[a]	Conf
chromium[b]	CUBIC	0.65	0.6
neqo	CUBIC	0	0
quiche	CUBIC	0.48	0.08
xquic	CUBIC	0.6	0.55
mvfst[b]	BBR	0	0
xquic	BBR	0.37	0.15
xquic	Reno	0.43	0.38

a: old definition of Conformance
b: already shown to be non-conformant by a previous study.

But why does this non-conformance **matter**?

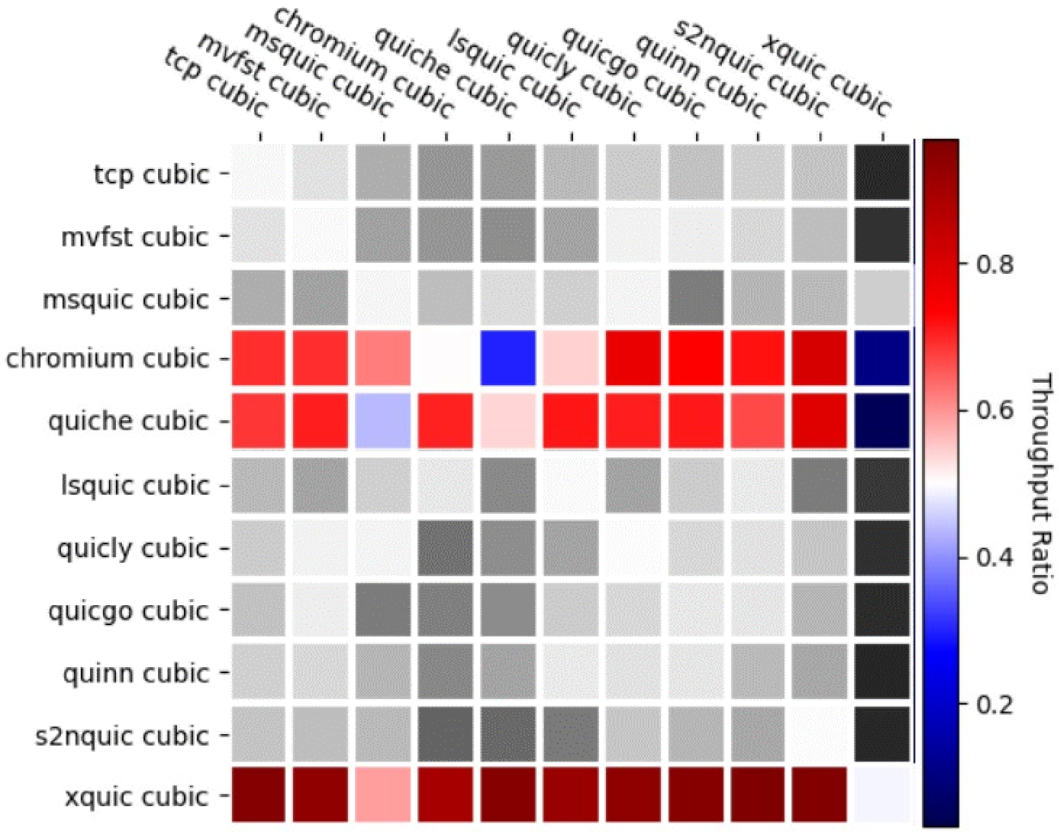
Is there a case for **containing it**?

General unfairness

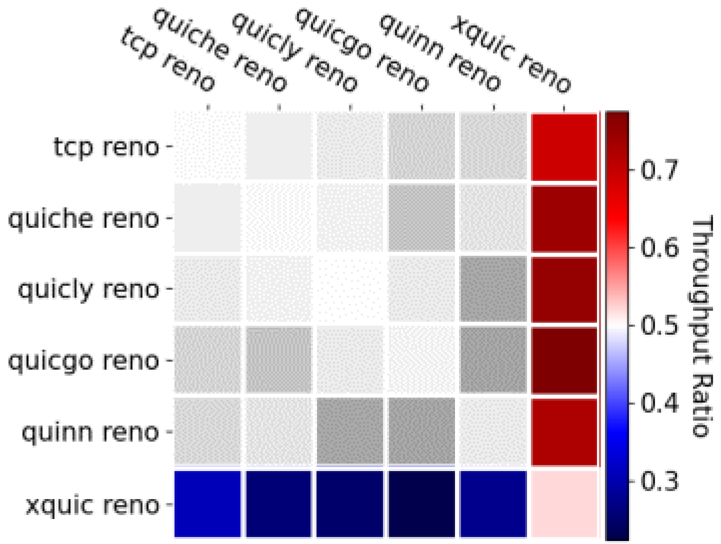


Non-Conformant implementations of CUBIC **introduce unfairness**

General unfairness



Non-Conformant implementations of CUBIC **introduce unfairness**



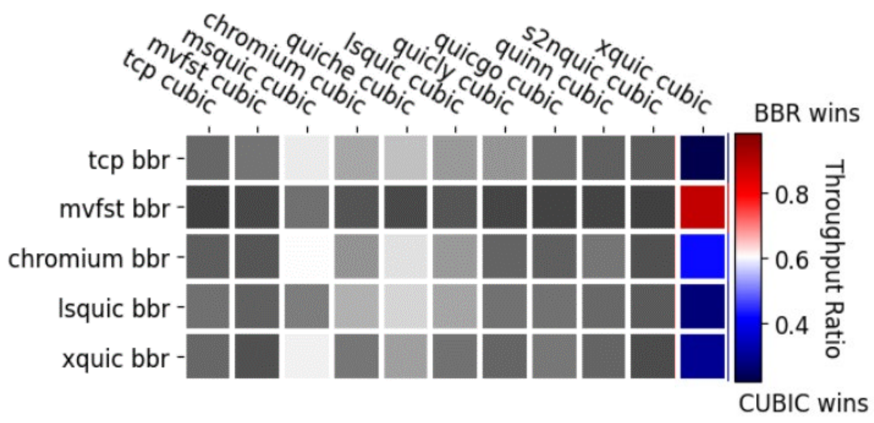
The same is true for Reno

But in this case, the non-conformant implementation of **xquic Reno starves**.

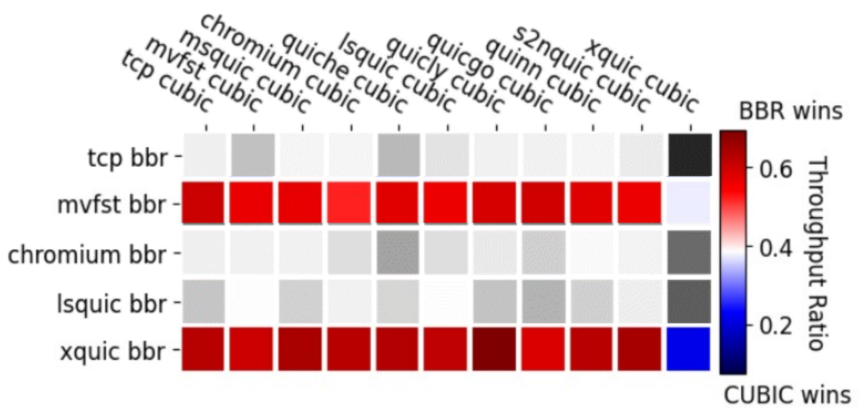
Subversion of Expectations

Well-known trend when CUBIC competes with BBR:
CUBIC gets more bandwidth in **deep buffers**,
BBR gets more bandwidth in **shallow buffers**

But this trend can change depending on which QUIC implementation you use!

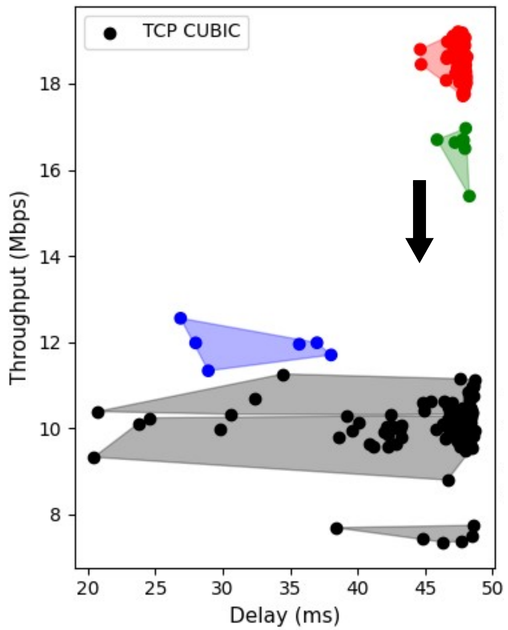


1 BDP buffer (expected to be red)



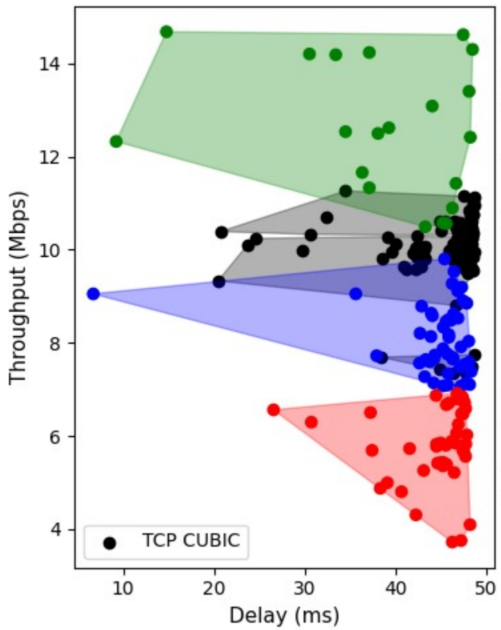
5 BDP buffer (expected to be **blue**)

Containing the Cambrian Explosion



Original
mvfst BBR

gain = 1.25



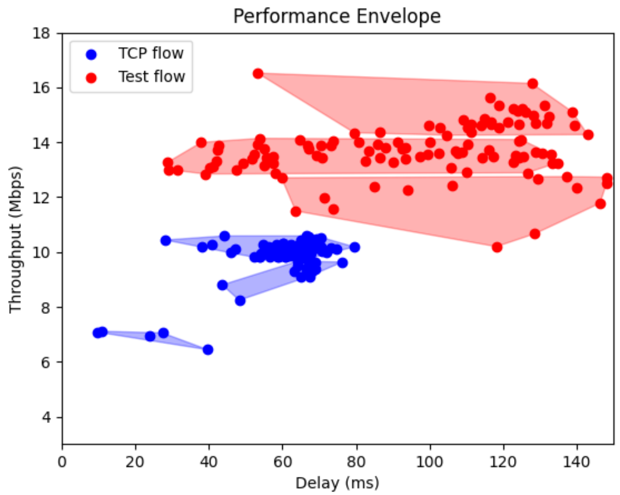
Modified
mvfst BBR

gain = 1

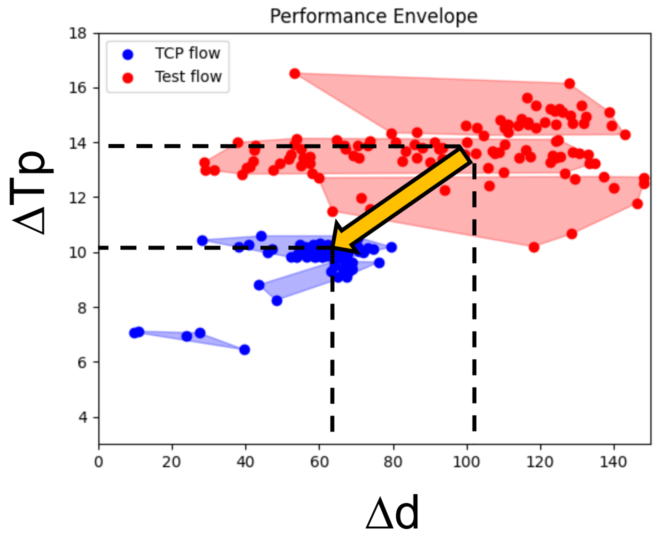
Correction ➡ Translation

Can we compute the translation needed and then use it to inform the correction?

Conformance post-translation

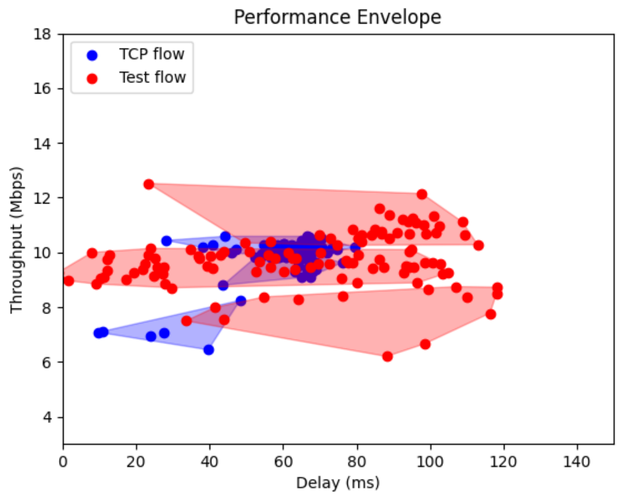


PE of some implementation with low conformance.



Compute the translation needed to maximize the overlap between the two PEs.

(Δd, ΔTp) is the **Translation Vector**



Compute the new conformance as **Conformance-T** or Conformance post-translation.

Conformance post-translation

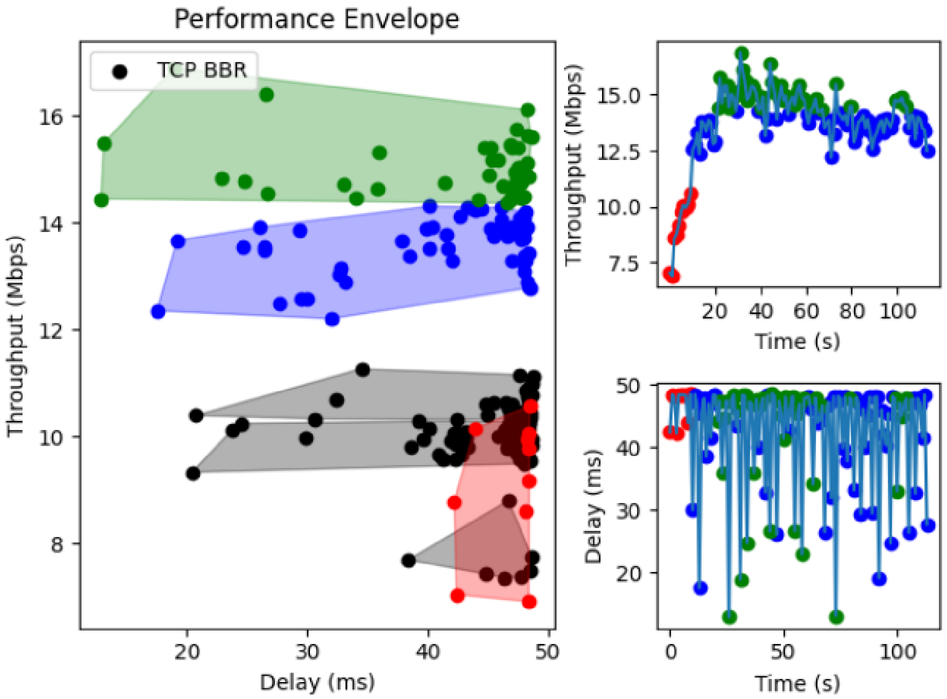
A large **improvement in Conformance-T** would indicate that it is **possible to improve the Conformance** of that implementation via simple **parameter tuning**.

In other instances, it would help narrowing down the possible issues with an implementation (incorrectly set gains, larger cwnd than usual)

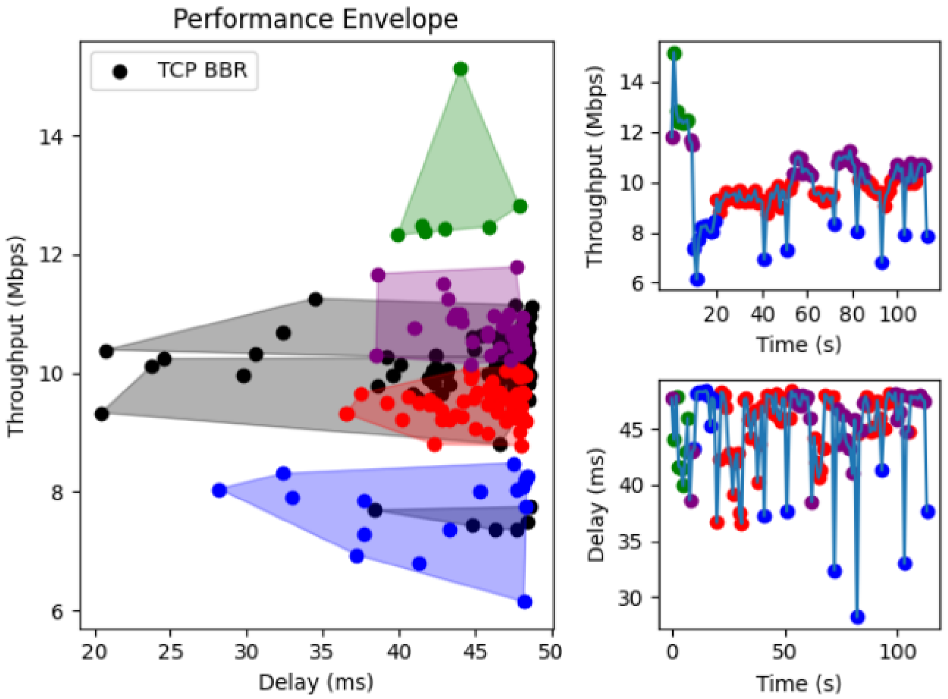
		Original implementation			
Stack	Type	Conf	Conf-T	Δ-tput	Δ-delay
chromium+	CUBIC	0.6	0.74	+3 Mbps	0 ms
mvfst+	BBR	0	0.7	+9 Mbps	0 ms
xquic	BBR	0.15	0.42	+4 Mbps	0 ms
quiche	CUBIC	0.08	0.55	+5.5 Mbps	0 ms
xquic	CUBIC	0.55	0.64	0 Mbps	-5 ms
		0.72	0.81	-2 Mbps	0 ms
xquic	Reno	0.38	0.81	-4 Mbps	-3 ms
nego	CUBIC	0	0.62	-6 Mbps	-5 ms

Fixing **xquic BBR**

Before and after changing xquic BBR’s pacing gain from **1.5** to **1**



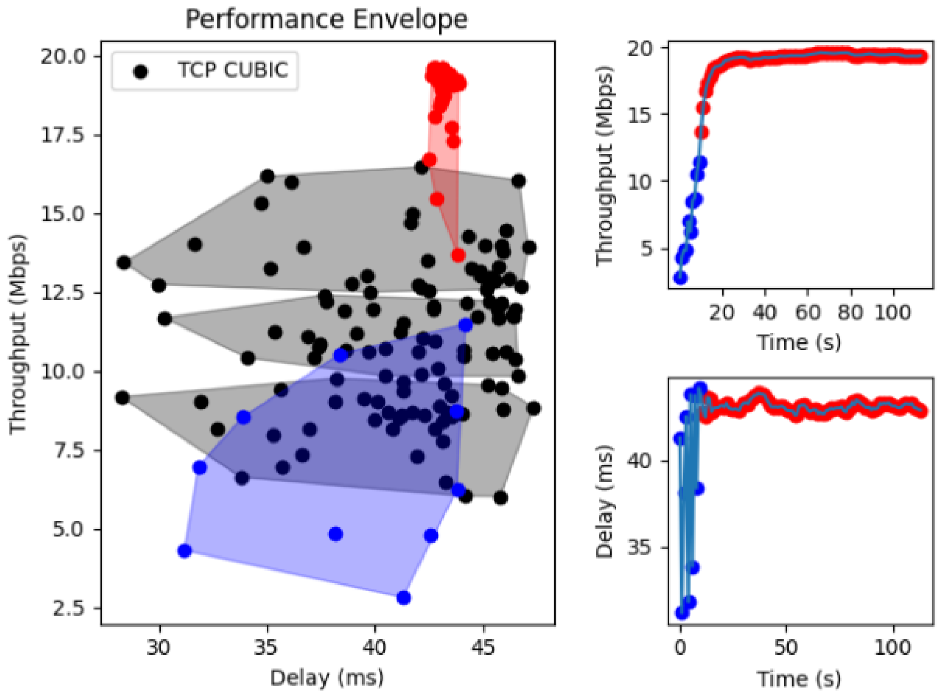
xquic BBR original
Conformance = 0.15



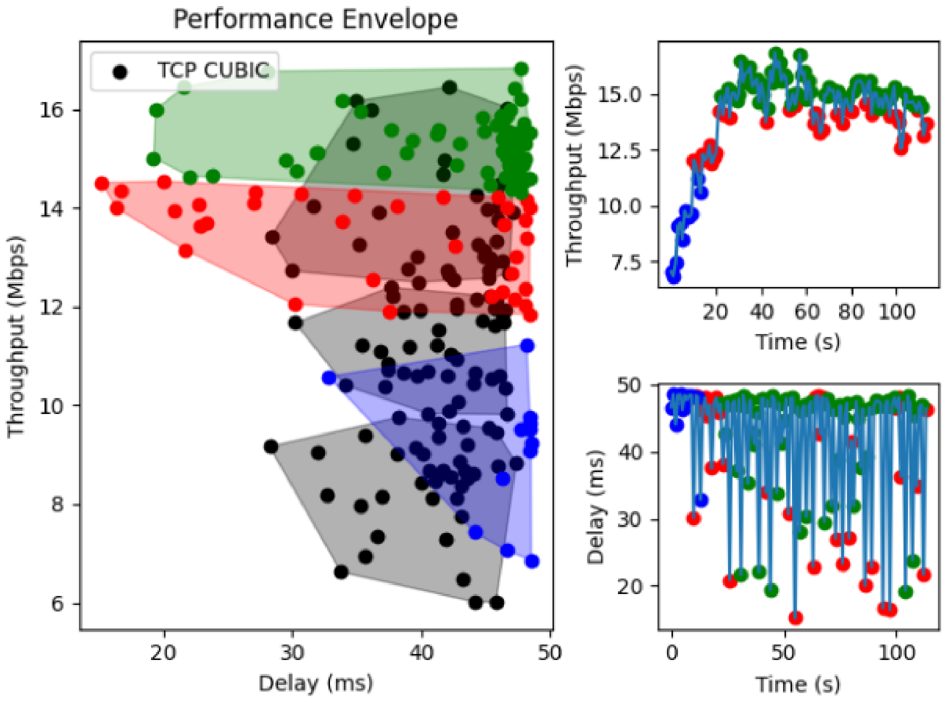
xquic BBR modified
Conformance = 0.38

Fixing **quiche CUBIC**

Before and After disabling quiche CUBIC's **spurious loss detection**



quiche CUBIC original
Conformance = 0.08



quiche CUBIC modified
Conformance = 0.55

Summary

We **benchmarked** the CCA implementations of **11 actively deployed QUIC stacks**

We **improved** and **Performance Envelope** metric and used it to identify **5 new CCA implementations** showing low Conformance

We introduced a new metric called **Conformance-T** that can **help developers** improve the conformance of a CCA implementation

We **identified implementation-level differences** and **improved the Conformance** of 2 of the 5 newly identified low-conformance CCA implementations.

Run the code

Read the paper

Thank you!