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

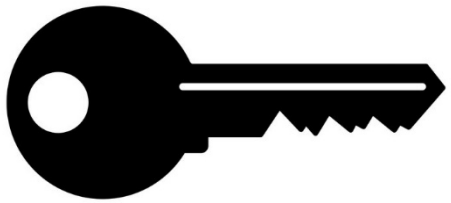
Standard with HTTP3

The QUIC Revolution

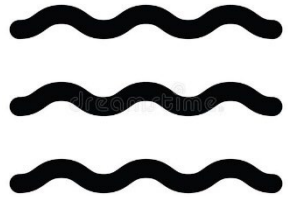
Reinvents many aspects of transport

New 0 RTT Handshake

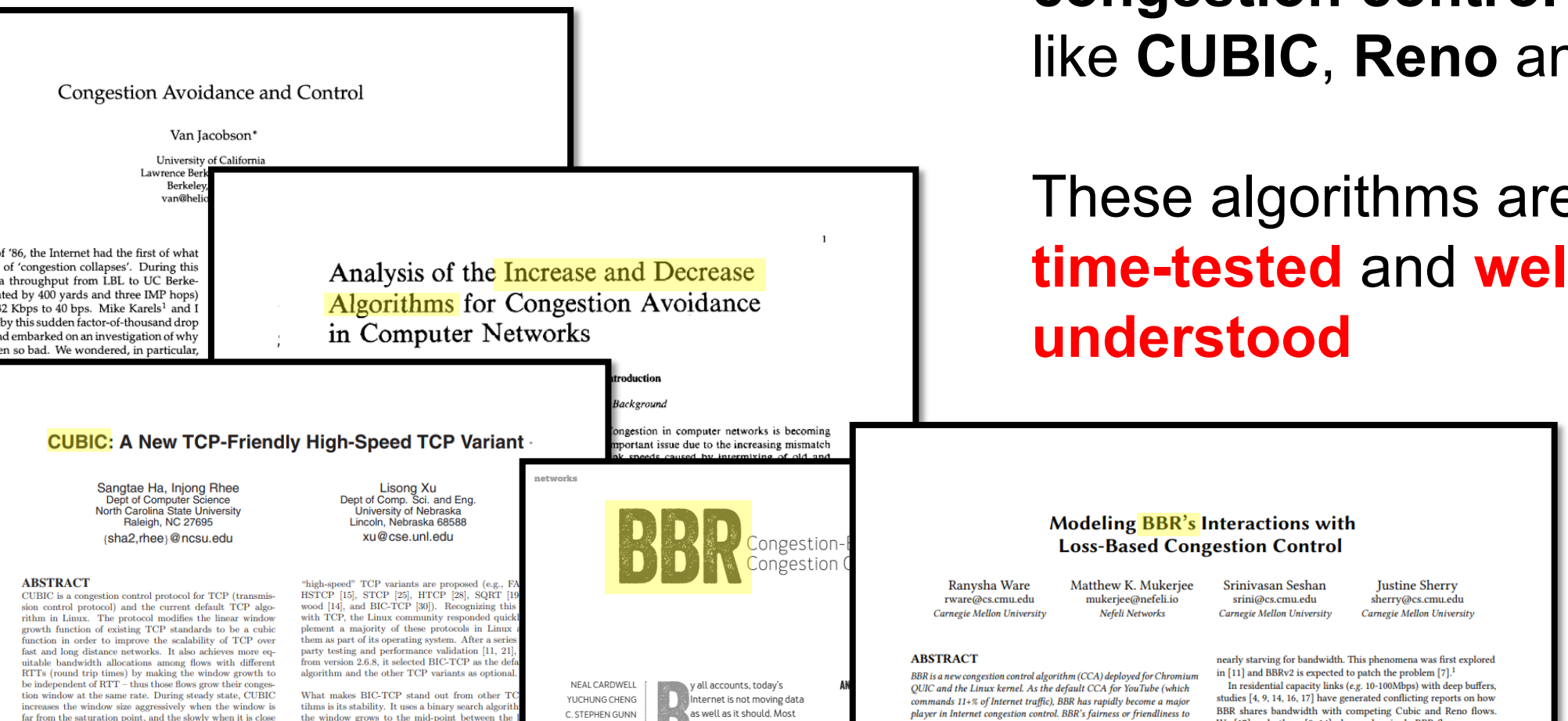
Baked-in encryption



Multistreaming support



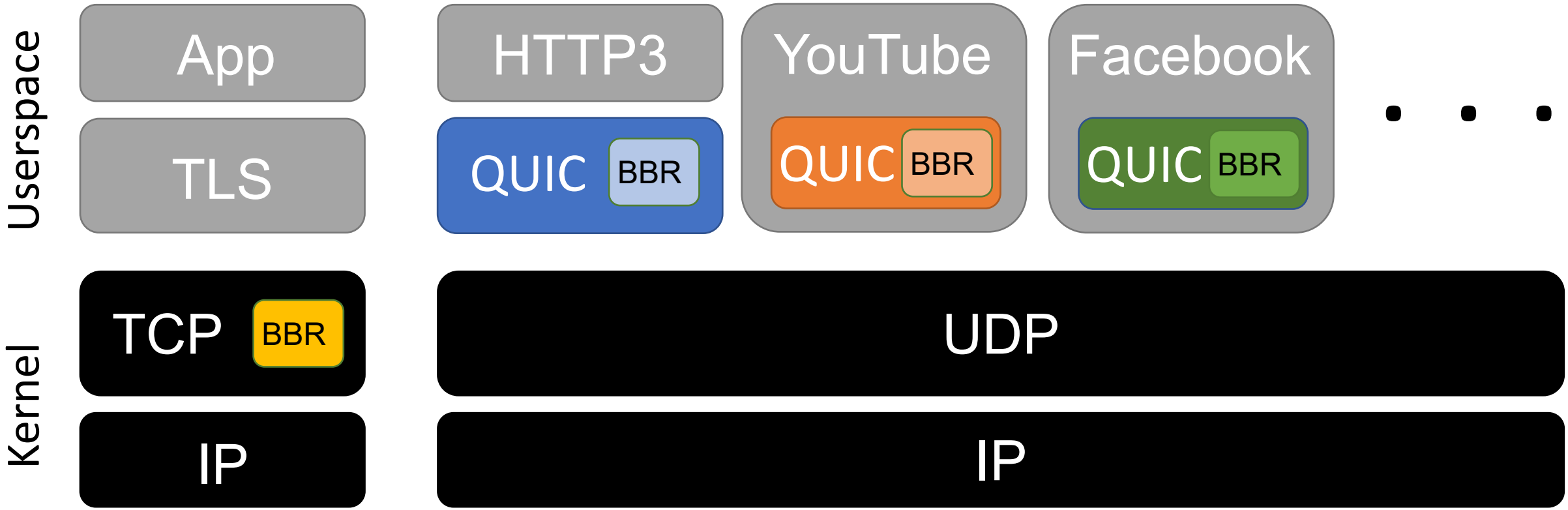
The QUIC Pragmatism



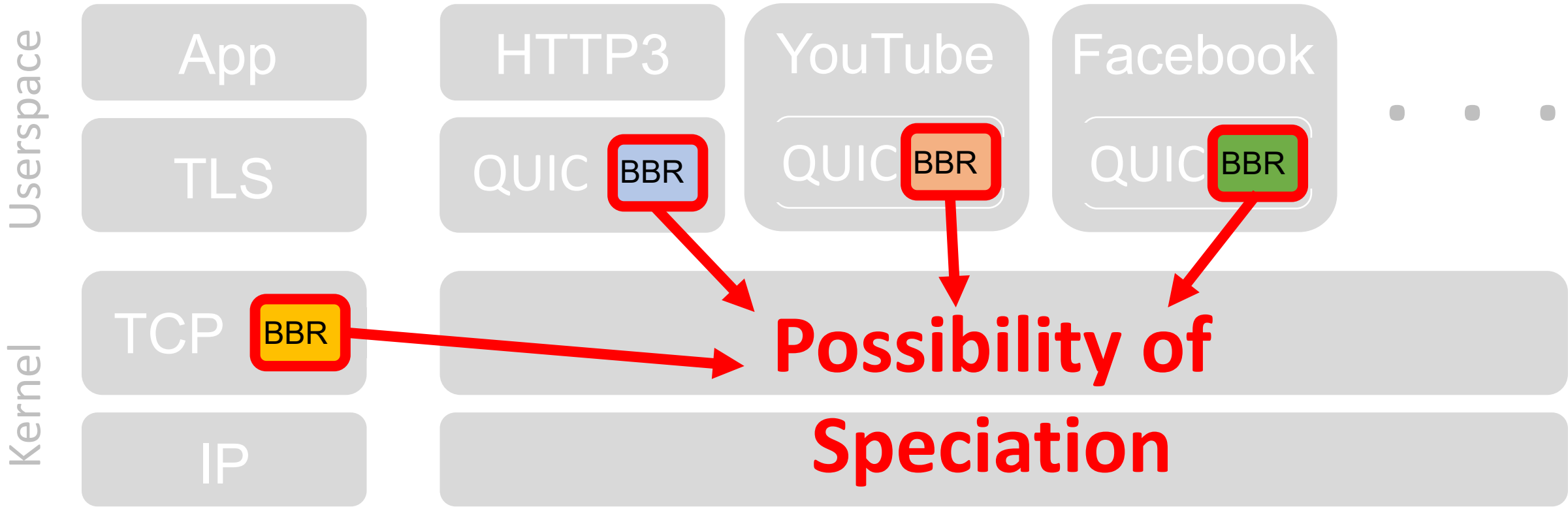
Most stacks stick to **standard congestion control algorithms** like **CUBIC**, **Reno** and **BBR**

These algorithms are **time-tested** and **well understood**

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

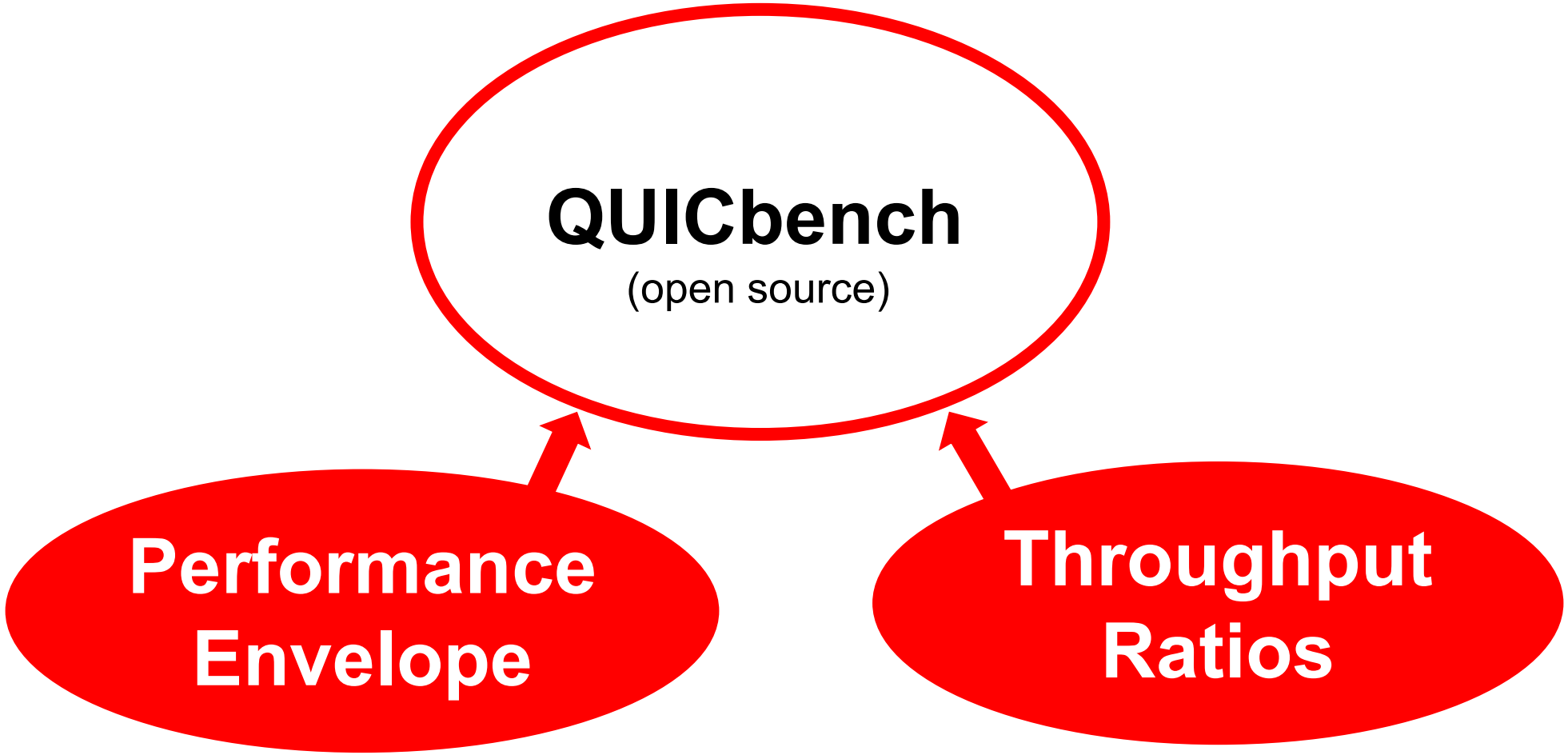
#2 Study interactions between different implementations

Performance Envelope

Throughput Ratios

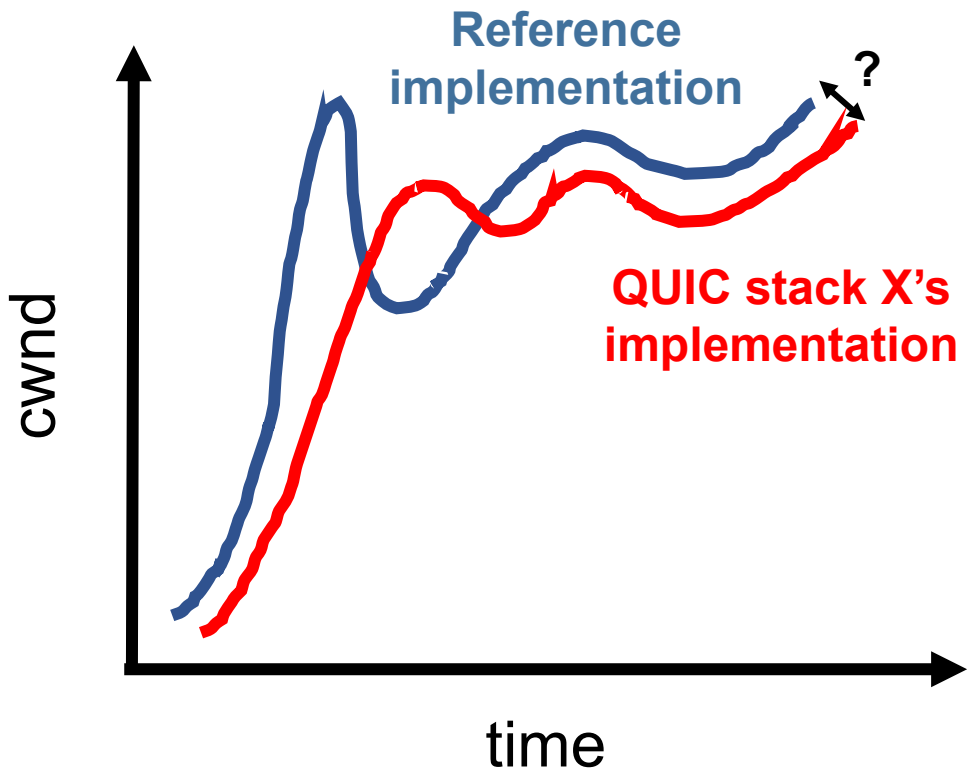
Benchmarking QUIC CC implementations

Goals:



Defining *Similarity*

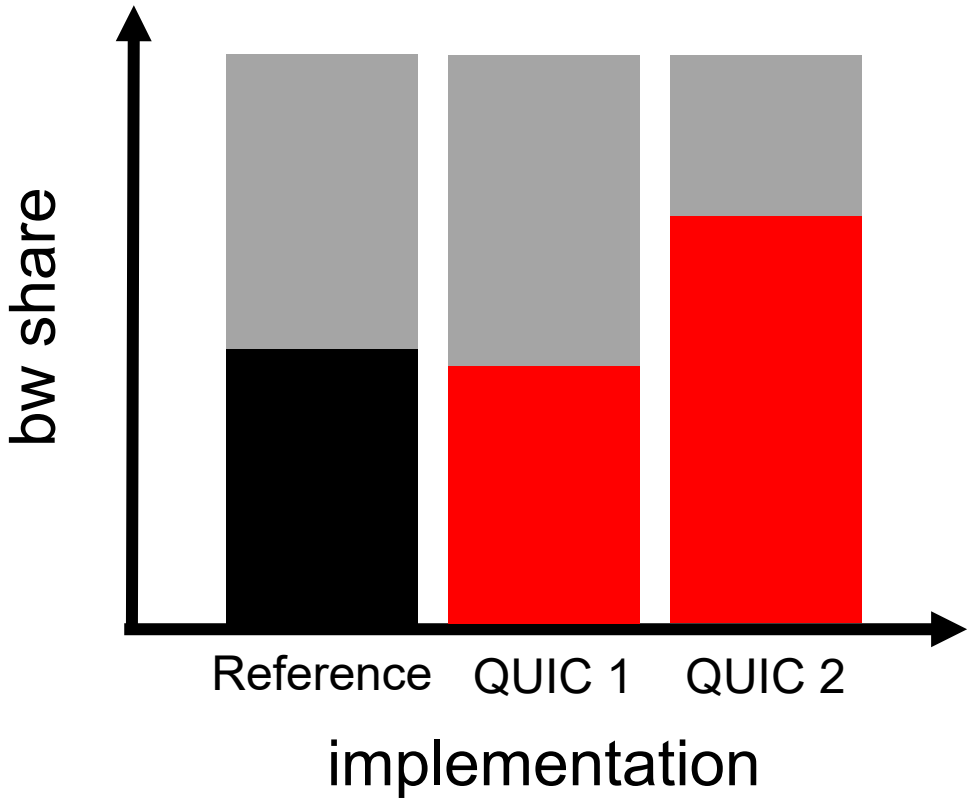
The fine-grained approach: **Compare cwnd graphs**



Problem:
Unrealistic goal for userspace implementations

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**

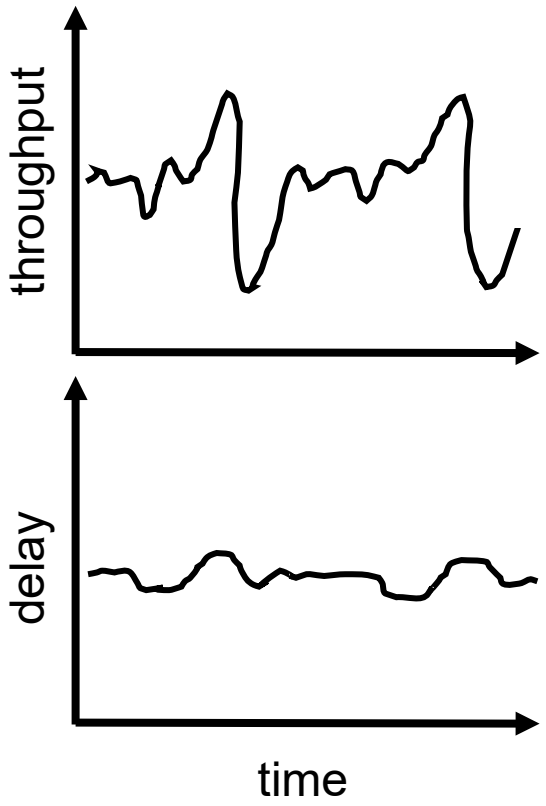
Performance Envelope is a

multi-dimensional metric

We chose **throughput** and **delay** as the two dimensions

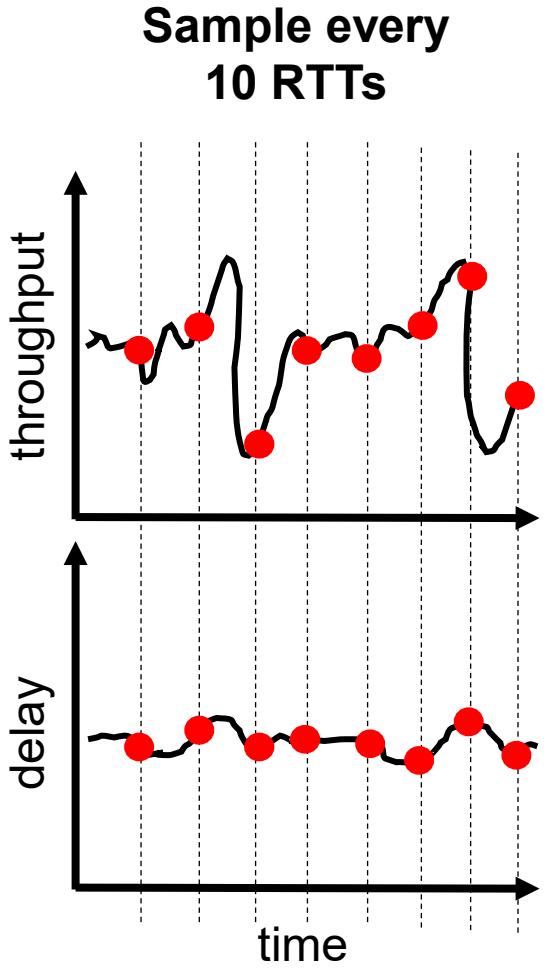
Measuring the

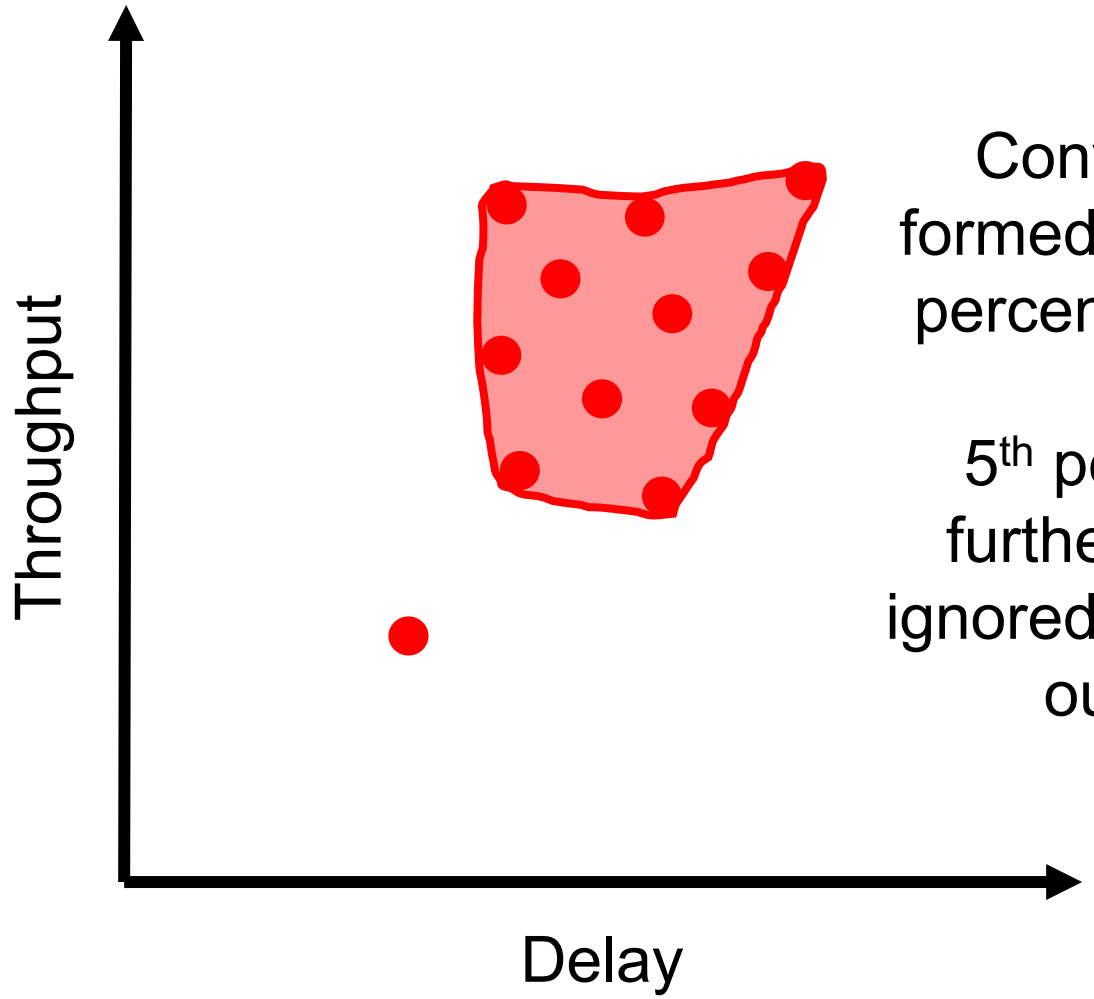
Performance Envelope (PE)



Measuring the

Performance Envelope (PE)



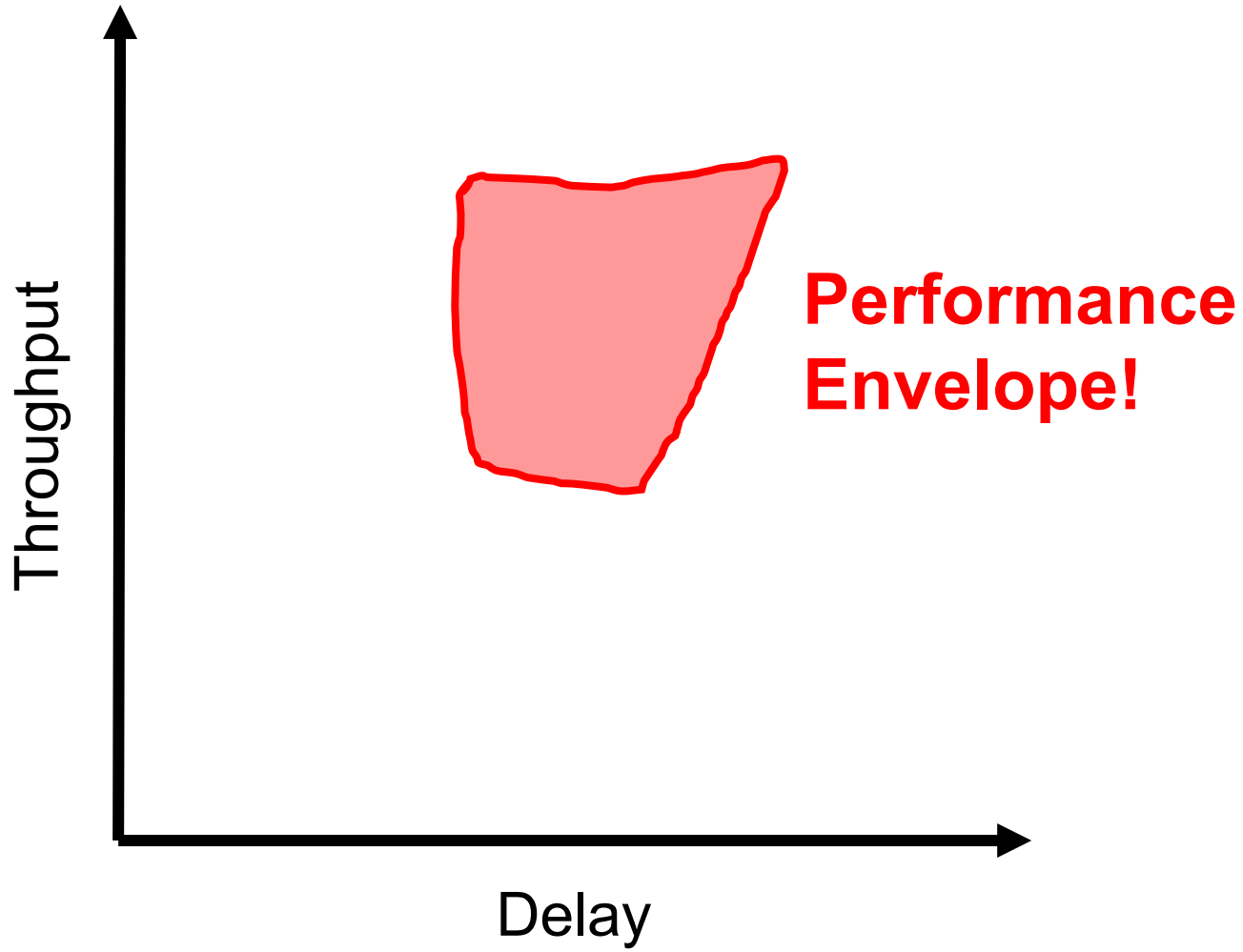


Convex Hull formed using 95th percentile points

5th percentile furthest points ignored to exclude outliers

Measuring the

Performance Envelope (PE)

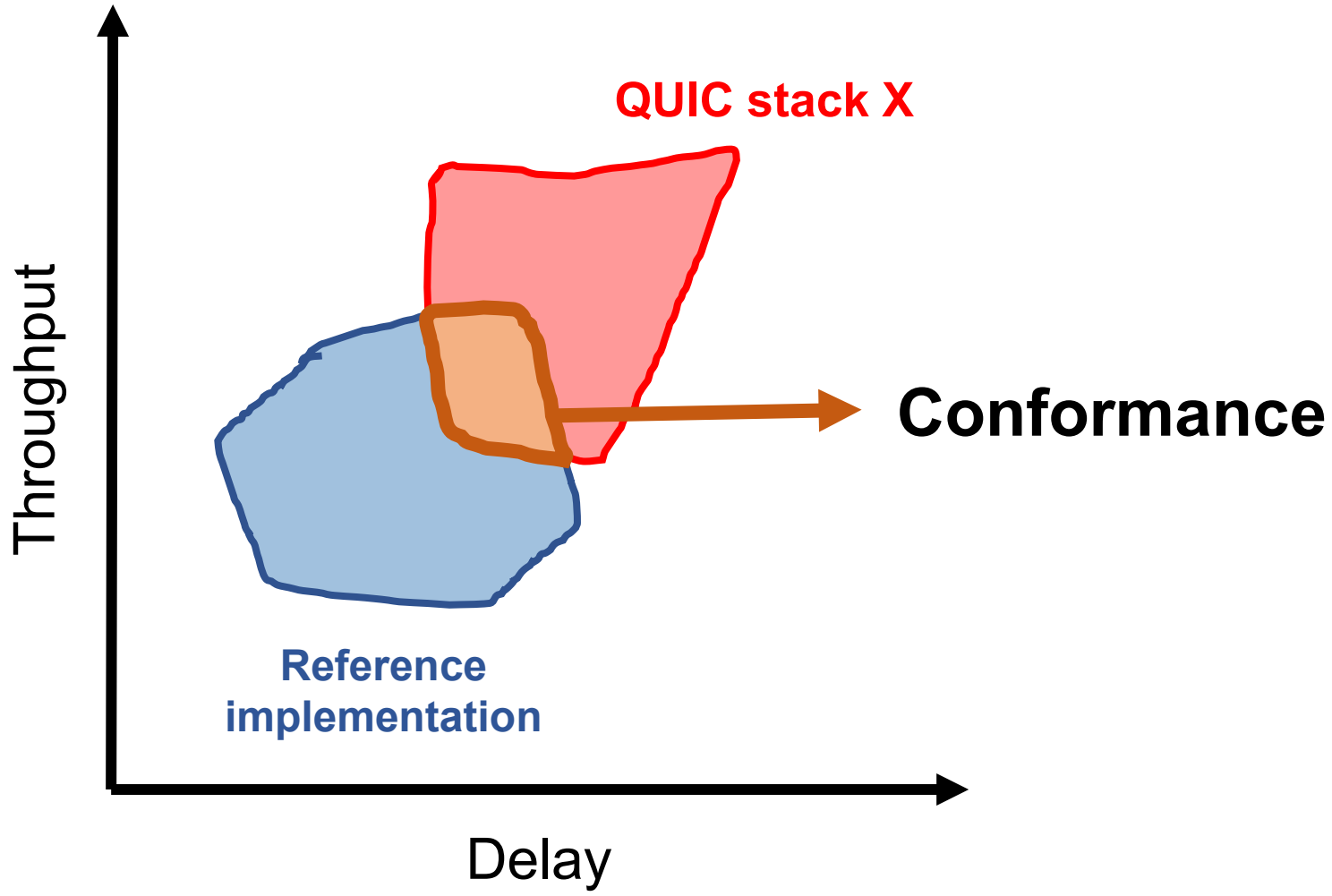


Performance Envelope (PE)

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

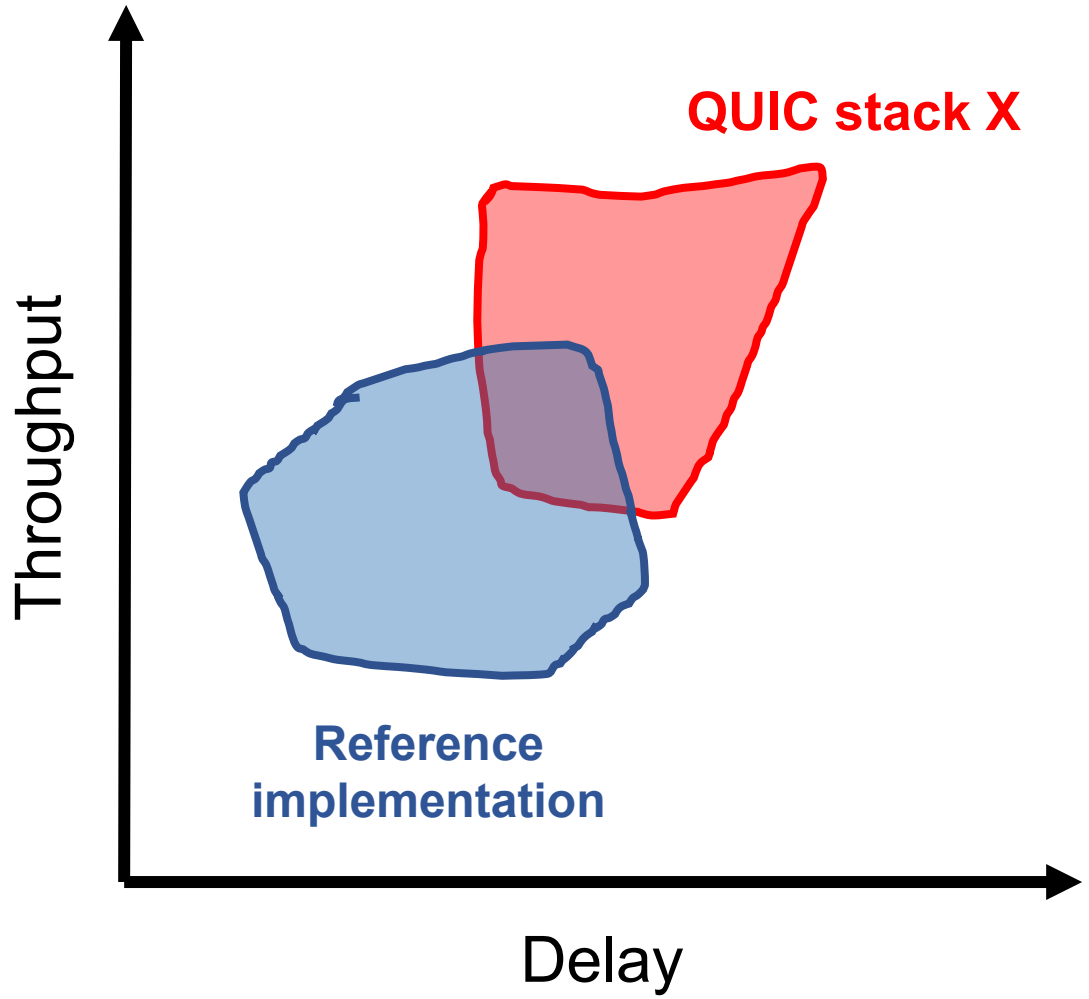


Performance Envelope (PE)

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



Deviation

The measure of dissimilarity

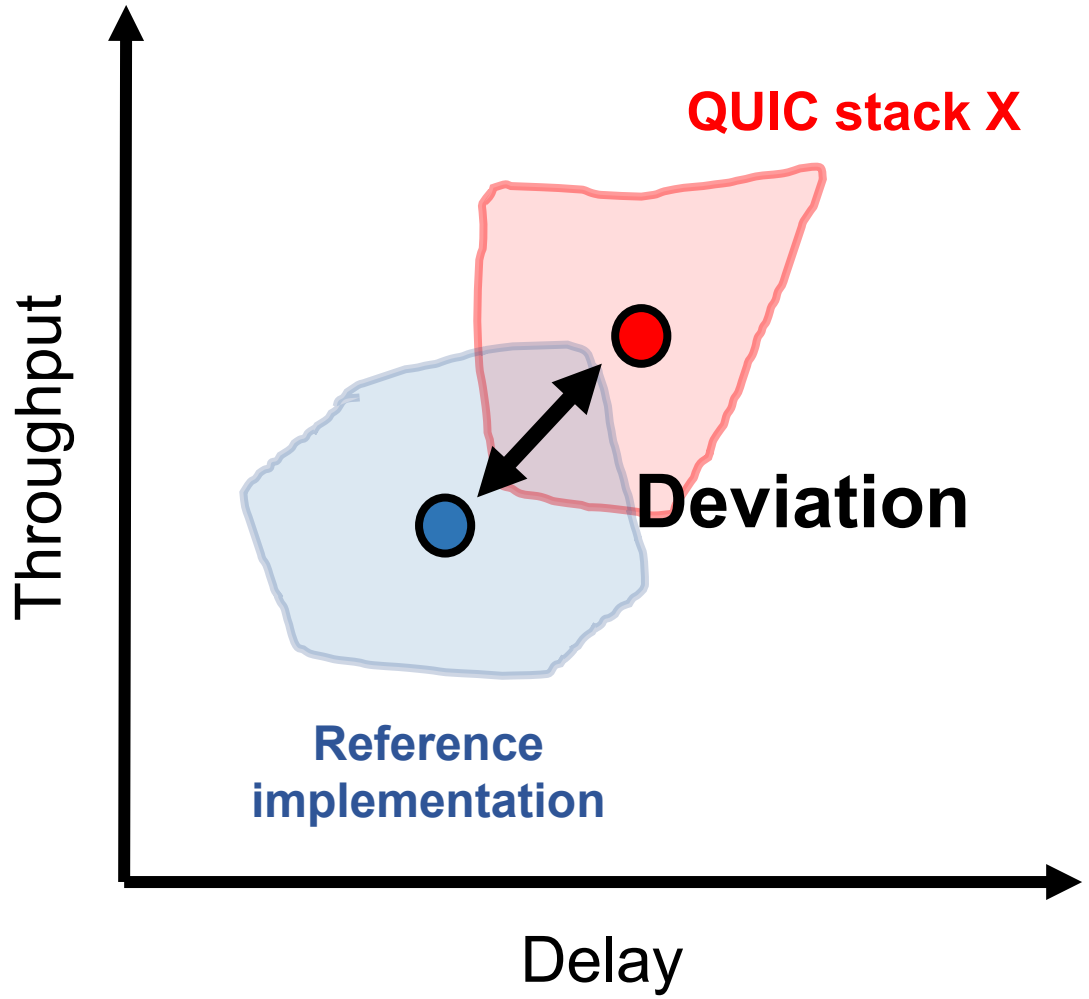
defined as normalized distance between the centroids of the two PEs

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

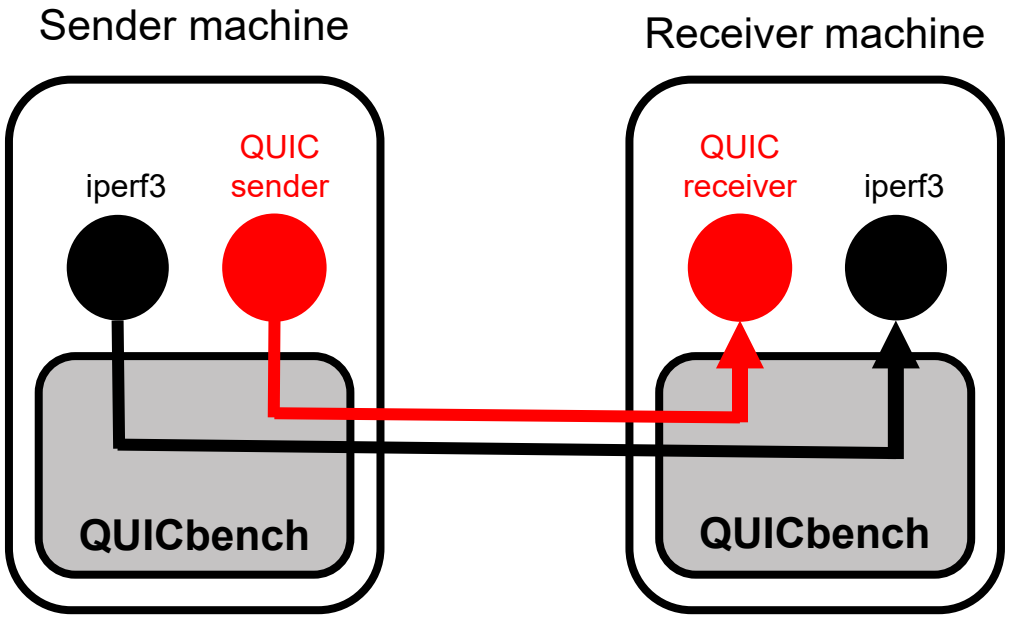


Deviation

The measure of dissimilarity

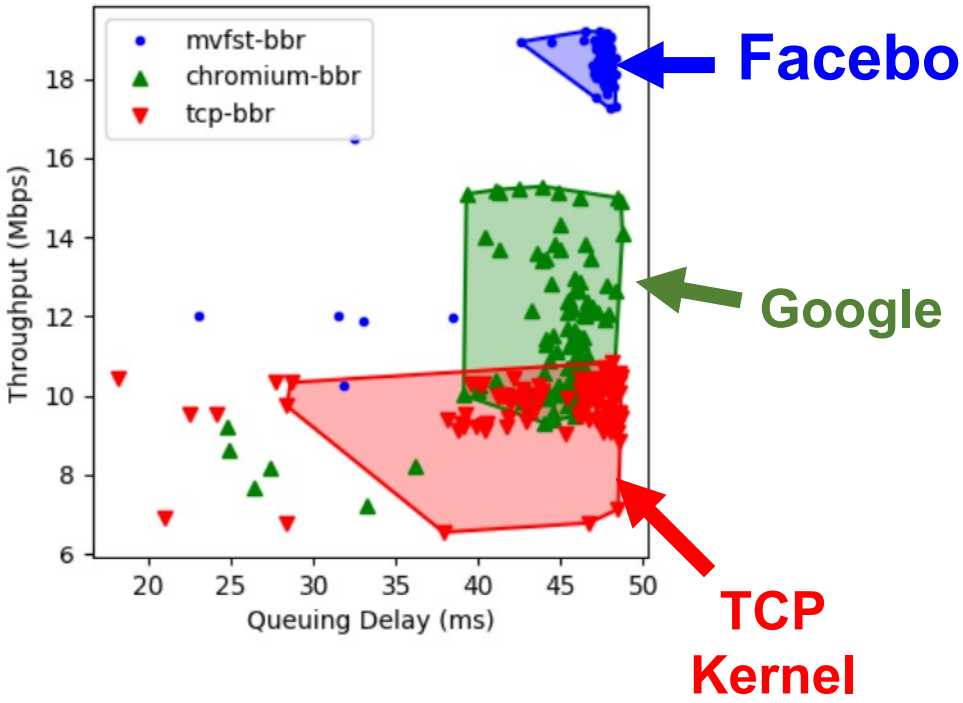
defined as normalized distance between the centroids of the two PEs

Evaluation details



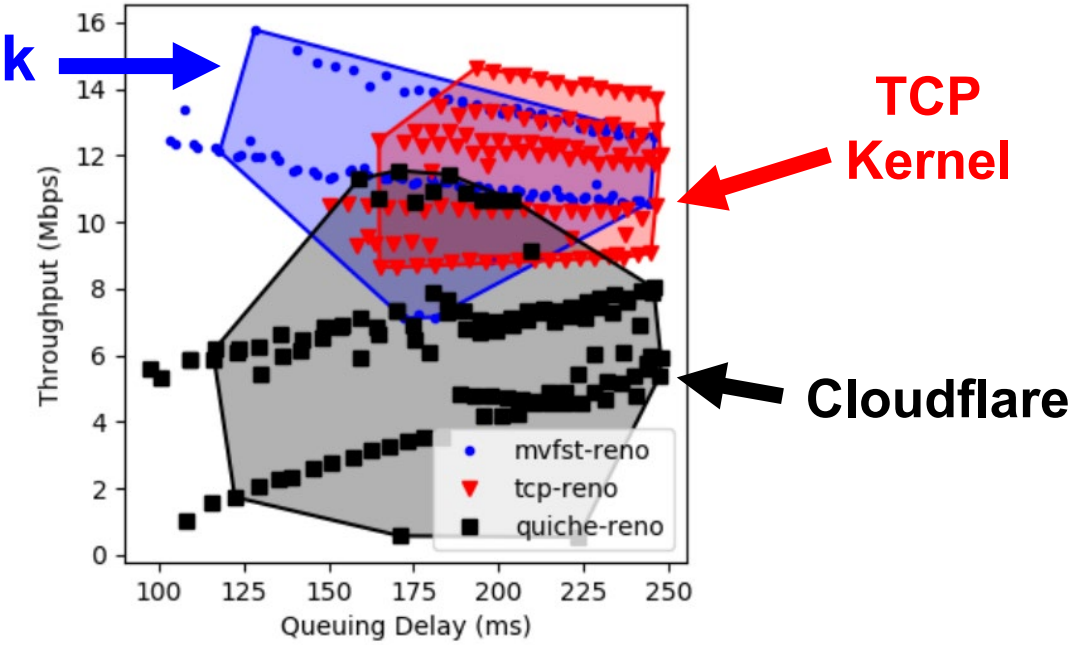
Organization	Stack	CUBIC	BBR	Reno
-	Linux kernel	✓	✓	✓
Facebook	`mvfst` [3]	✓	✓	✓
Google	`chromium` [4]	✓	✓	✗
Microsoft	`msquic` [9]	✓	✗	✗
Cloudflare	`quiche` [2]	✓	✗	✓

Significant Speciation!



Example of low Conformance in BBR

20 Mbps, 50 ms RTT, 1 BDP Buffer



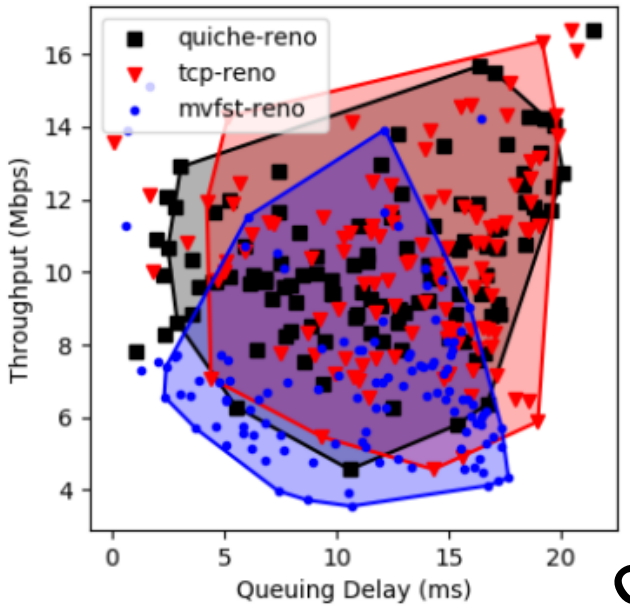
Example of low Conformance in Reno

20 Mbps, 50 ms RTT, 5 BDP Buffer

Rate-based vs. Window-based

Conformance deteriorates for window-based congestion control algorithms like **CUBIC** and **Reno** in **deeper buffers**

20 Mbps, 50 ms RTT, Reno

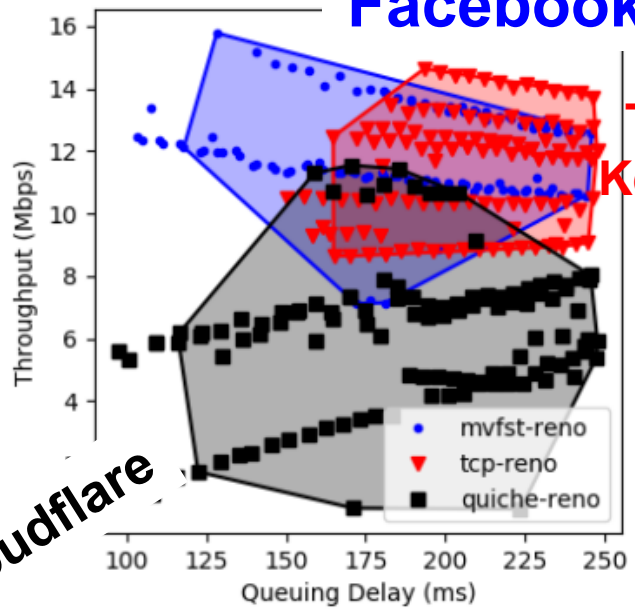


20 Mbps, 50 ms RTT, Reno

Facebook

TCP Kernel

Cloudflare

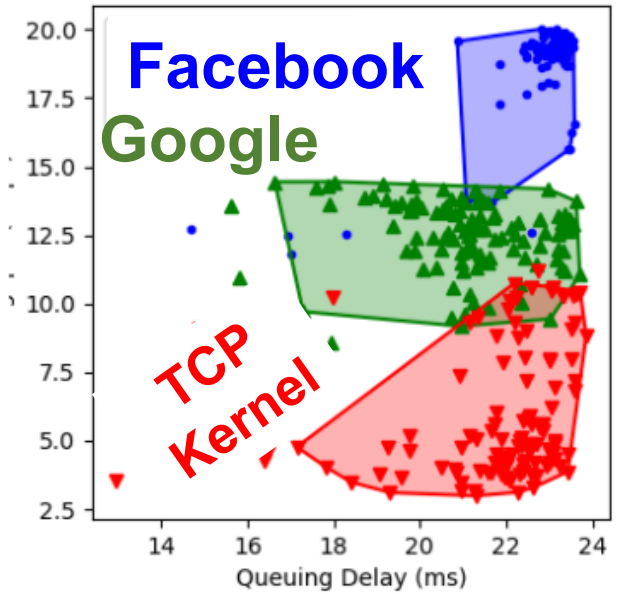


Large overlaps in 0.5 BDP Buffers → **Smaller overlaps in 5 BDP Buffers**

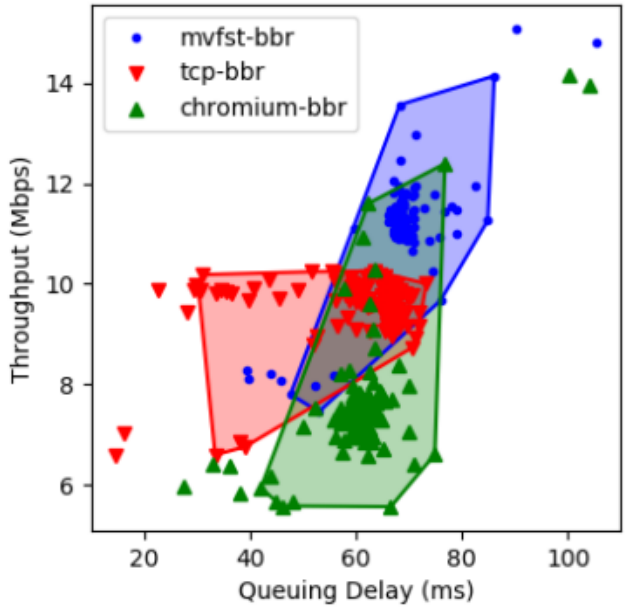
Rate-based vs. Window-based

Conformance improves for rate-based congestion control algorithms like **BBR** in **deeper buffers**

20 Mbps, 50 ms RTT, Reno



20 Mbps, 50 ms RTT, Reno



Small overlaps in 0.5 BDP Buffers → **Larger overlaps in 5 BDP Buffers**

Inter-stack fairness

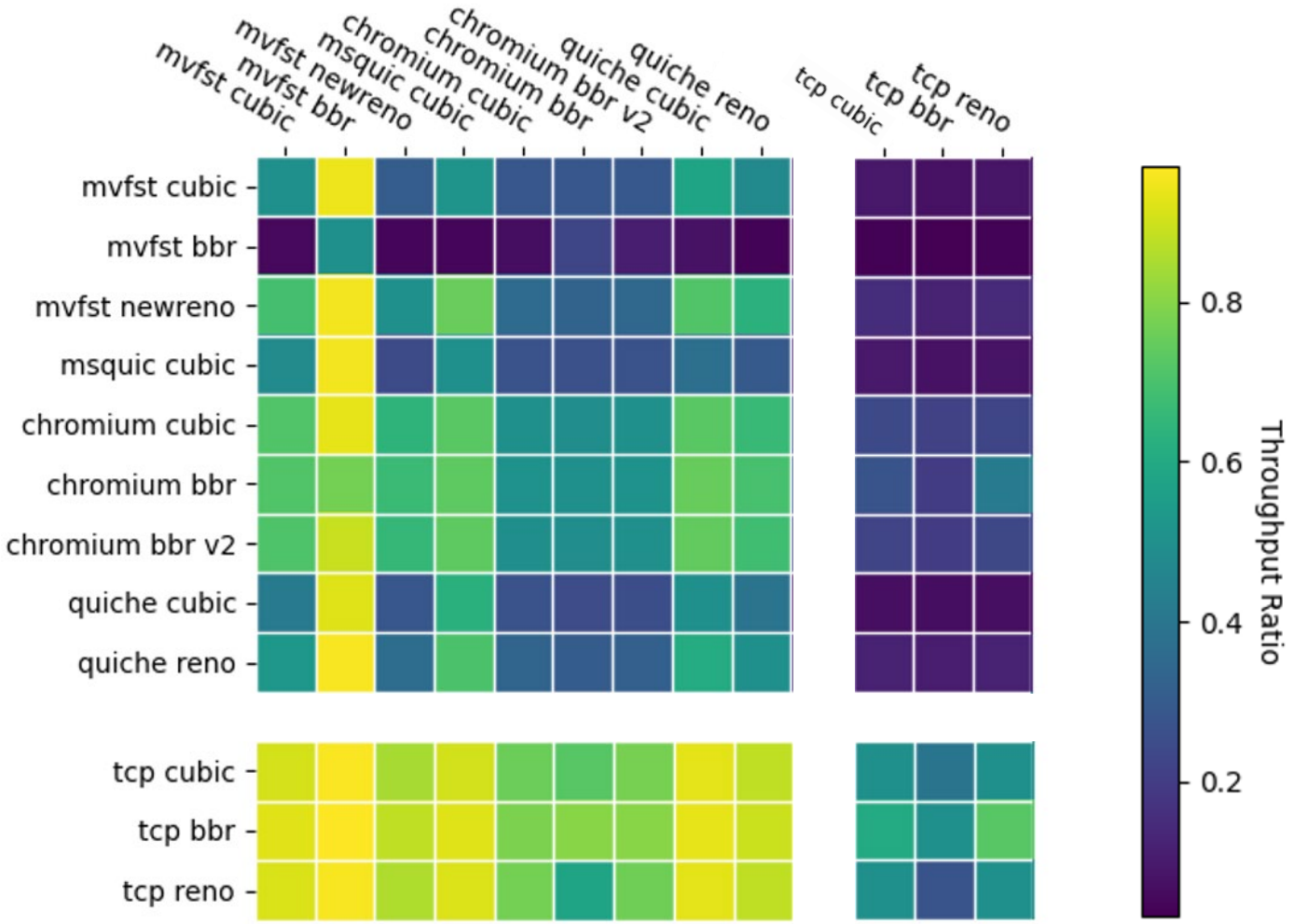
Ran all possible pairs of implementations and plotted their **throughput ratio** on a heat map.

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

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

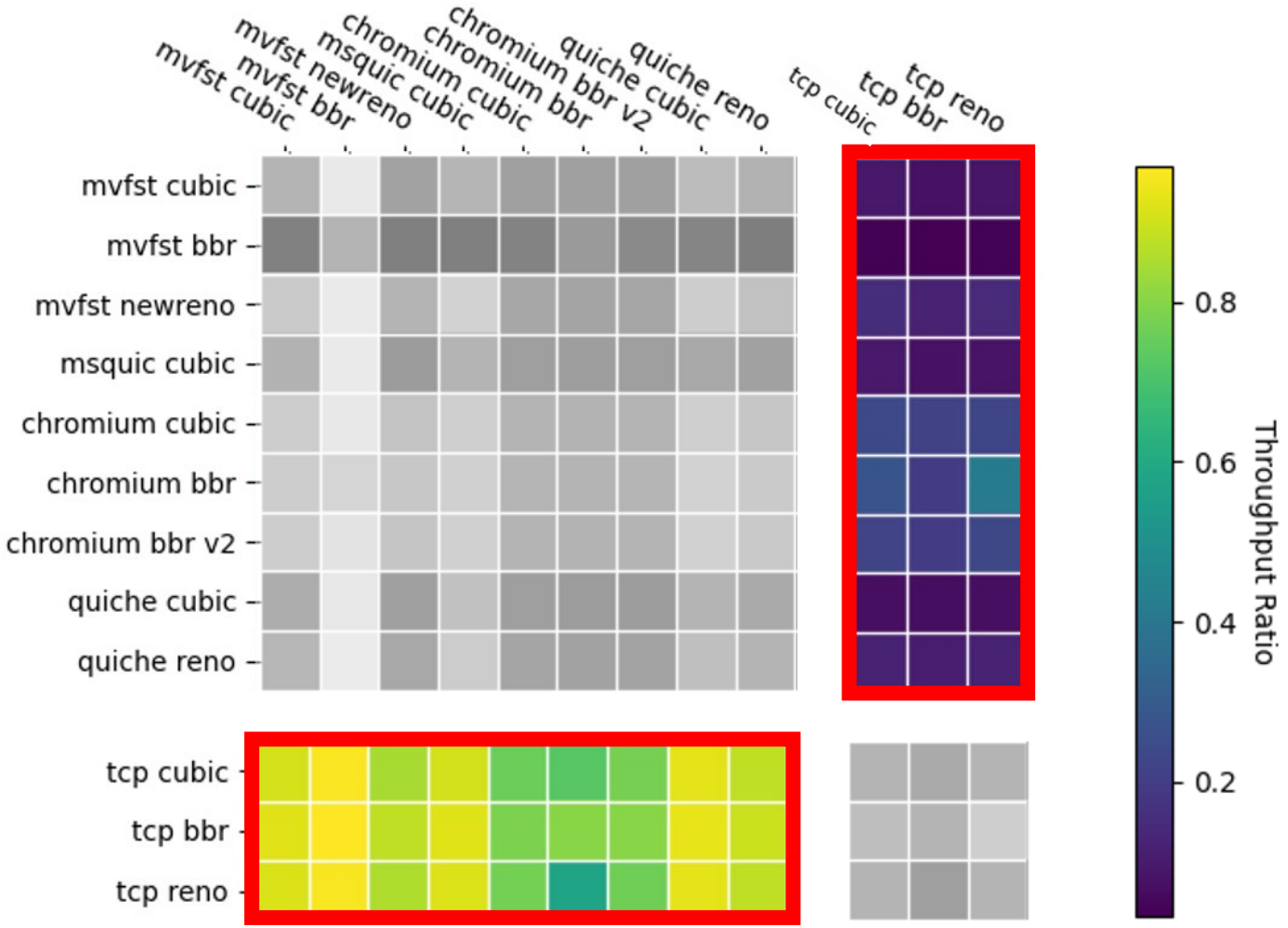
If the ratio is greater than 0.5, Stack X gets more throughput

Inter-stack fairness



1 BDP Buffer, 20 Mbps, 50 ms RTT

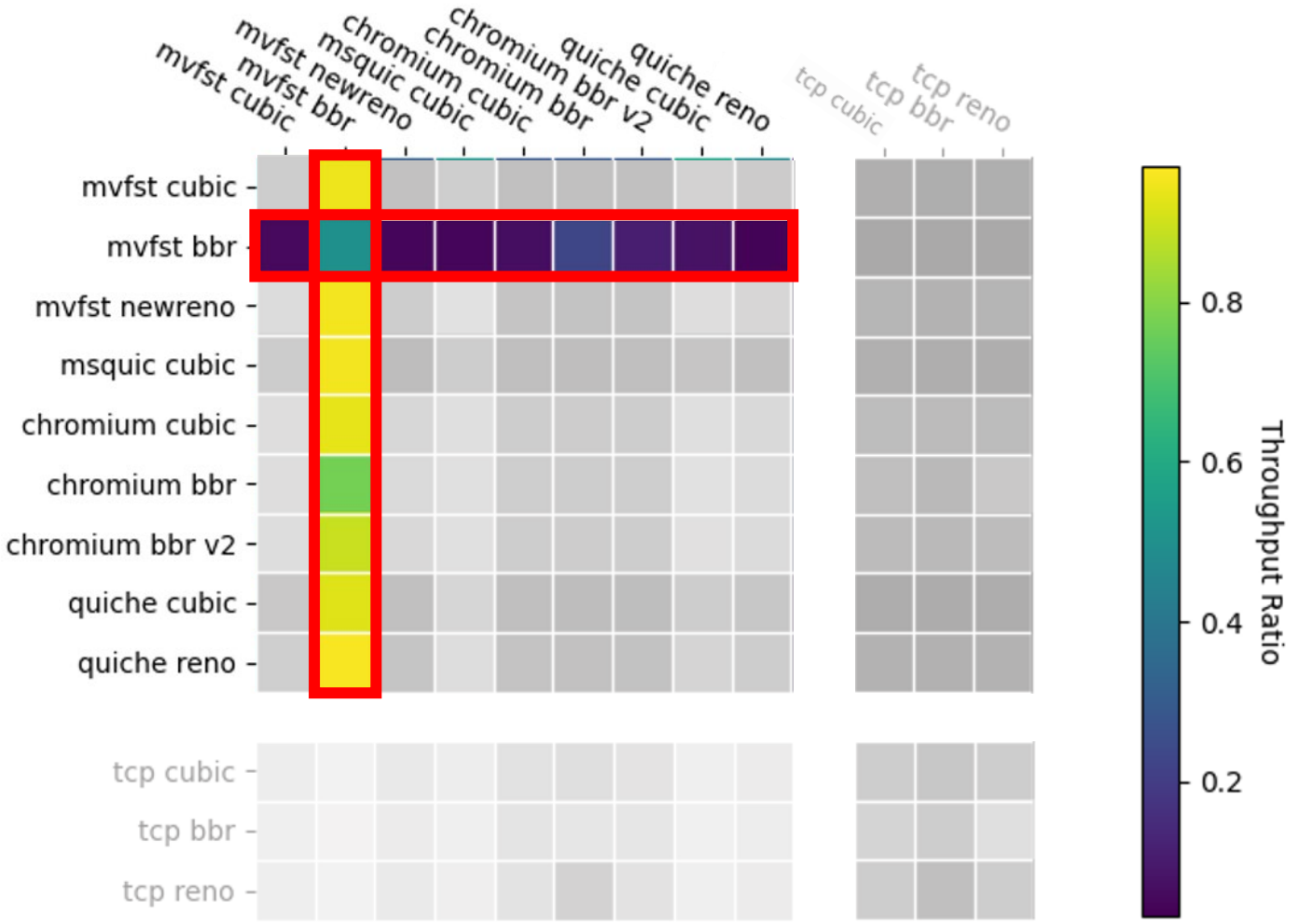
Inter-stack fairness



1 BDP Buffer, 20 Mbps, 50 ms RTT

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

Inter-stack fairness



1 BDP Buffer, 20 Mbps, 50 ms RTT

Facebook's QUIC stack

MVFST BBR massively outperforms all other QUIC implementations

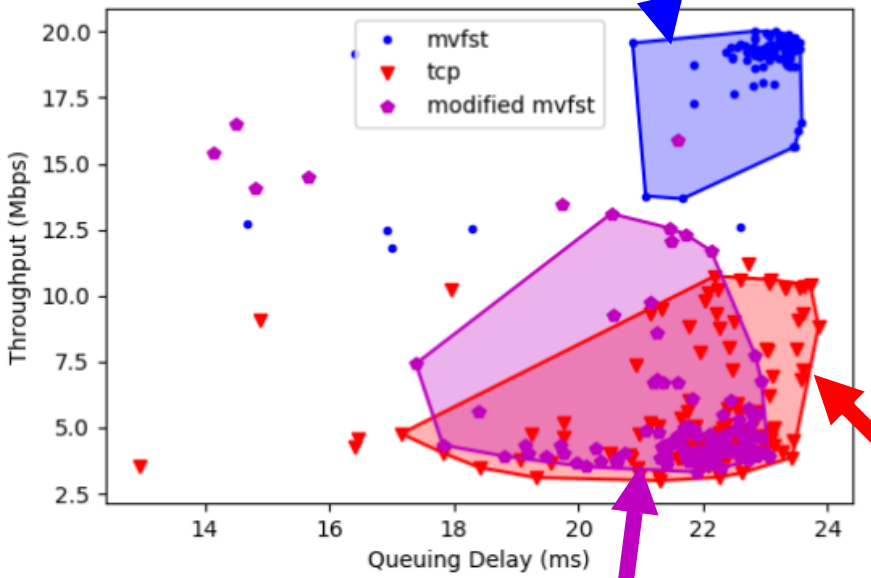
MVFST BBR

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

Original Mvfst
(conformance = **0.0**)



TCP Kernel

Modified Mvfst
(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

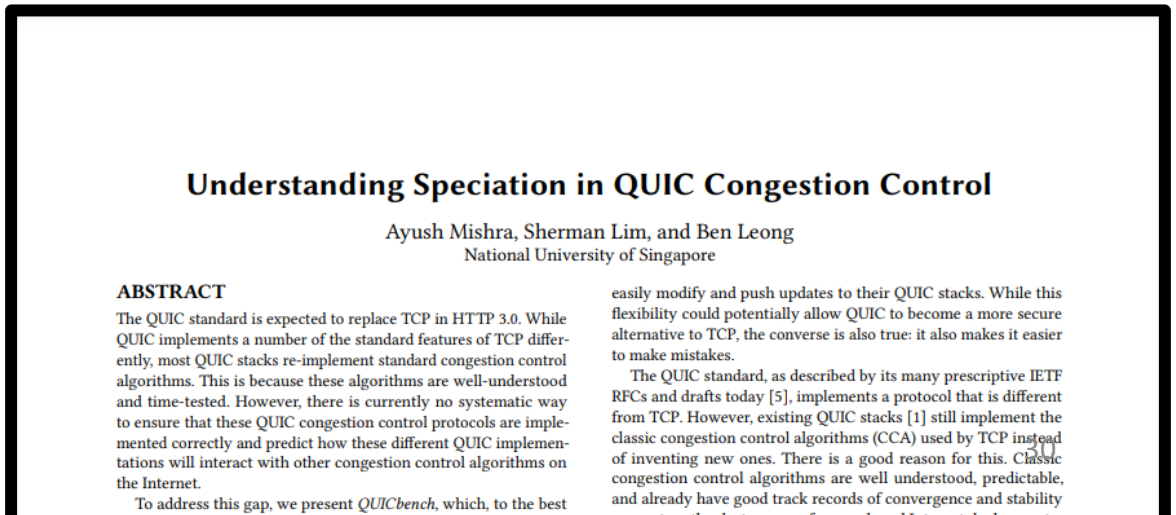
Evaluate more QUIC stacks

Exporting and verifying key CCA parameters

Thank you!

Read the paper:

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