Containing the Cambrian Explosion in QUIC Congestion Control

Ayush Mishra, Ben Leong

National University of Singapore

IETF 118, Prague

CCWG, 7th November 2023

On *Safety* and *Deployability* of a CCA

- So far, we've had discussions on determining if a CCA is *safe* and *deployable*.
- While this is an important step, these checks should go beyond the algorithm itself and apply to the implementations too.
- Our work shows that there is already significant speciation between implementations of standard congestion control algorithms like CUBIC, Reno, and BBR in QUIC.

On *Safety* and *Deployability* of a CCA

- Let's say a CCA is *safe* and *deployable*. How well can we expect these properties to propagate to all of its implementations?
- Case Study: **QUIC**

How well do the QUIC implementations of CUBIC, Reno, and BBR conform to their kernel counterparts?

- In the context of `5033bis`, this would mean determining the deployability of a CCA implementation by measuring how close it was to the *safe* and *deployable* version of that algorithm.

Measuring *Conformance*

- How do we measure if two implementations of a CCA are similar?
- The fine-grained approach: compare cwnd graphs

Problem: too restrictive and unrealistic

- The course-grained approach: compare relative-fairness

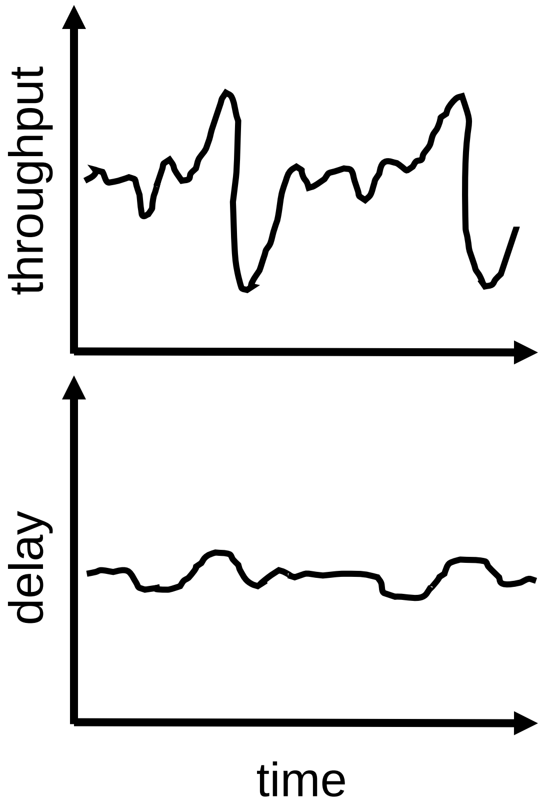
Problem: misses finer algorithmic differences

- The middle ground: **The Performance Envelope (PE)**

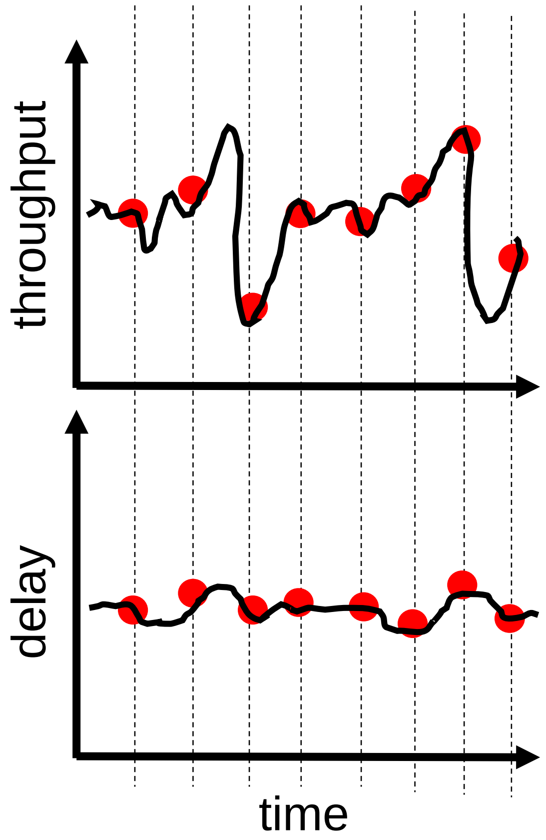
Measuring the *Performance Envelope*

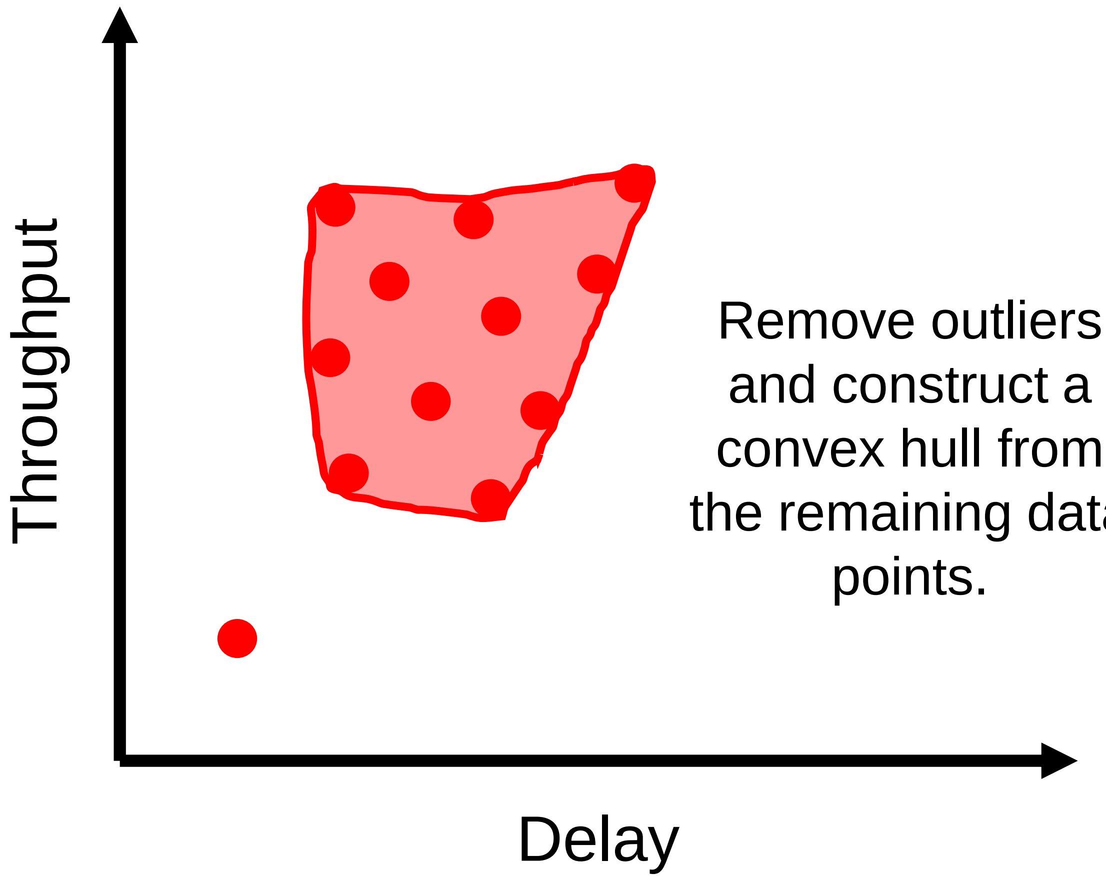
- The *Performance Envelope (PE)* metric is built on one key insight: **Different CCAs represent different trade-offs in the network.**
- We want to capture the trade off space in which an implementation operates.
- This trade off space can be multi-dimensional. The PEs discussed in this talk will be two-dimensional (Throughput vs Delay)

Measuring the *Performance Envelope*



Measuring the *Performance Envelope*

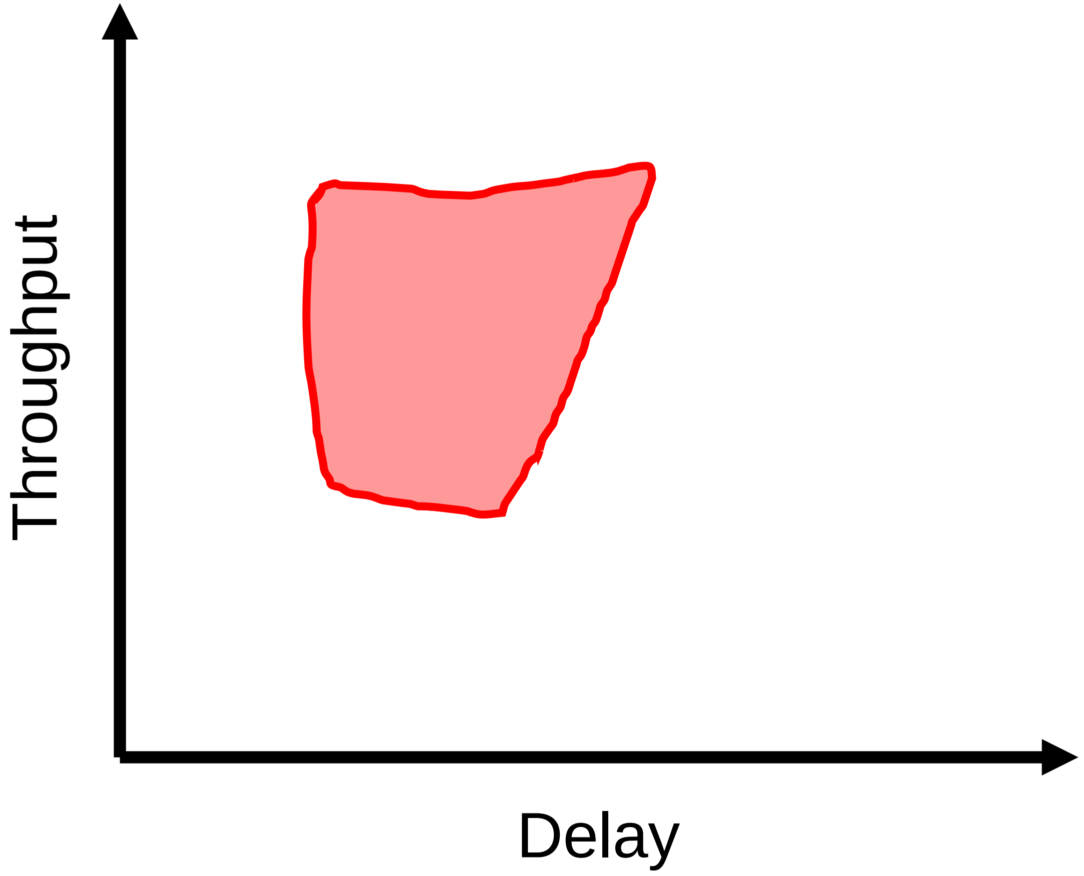




Remove outliers and construct a convex hull from the remaining data points.

Measuring the *Performance Envelope*

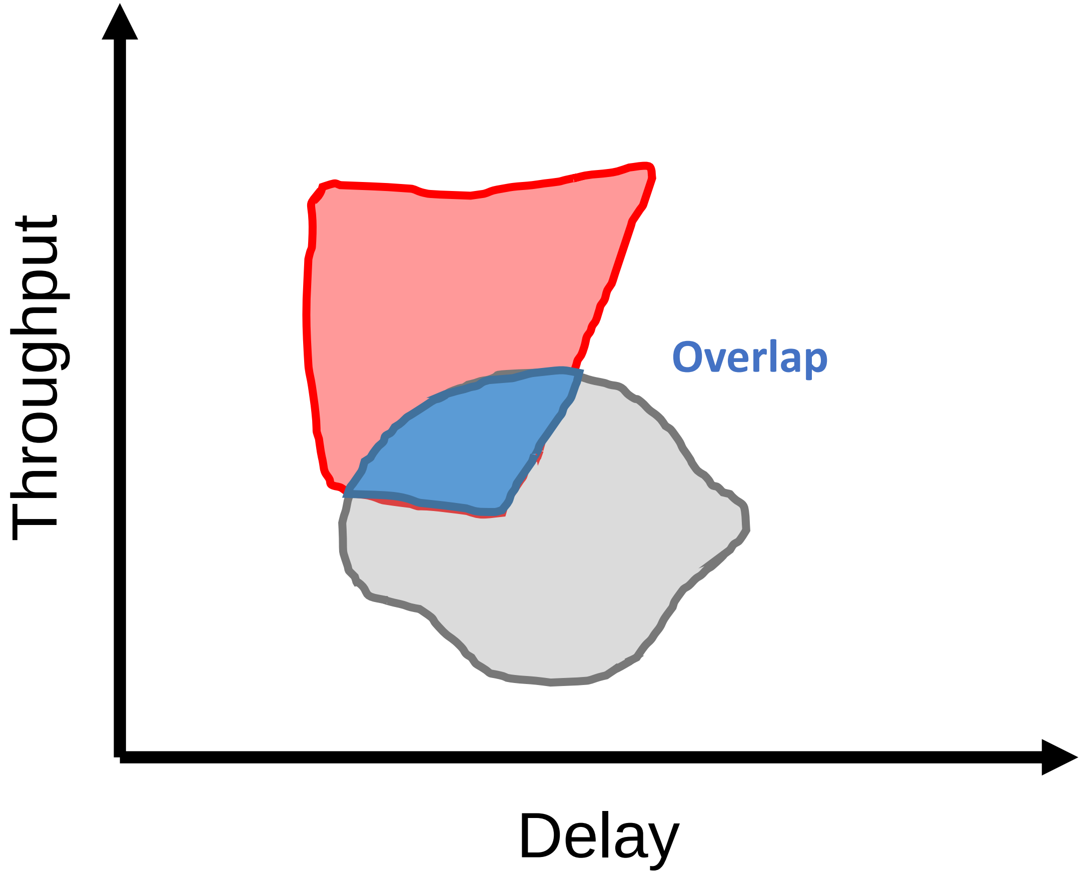
Performance Envelope!



Measuring the *Performance Envelope*

Level of overlap with a reference implementation becomes a measure of **Conformance.**

Conformance lies between
1 (complete overlap) and
0 (no overlap)



Measurement Results

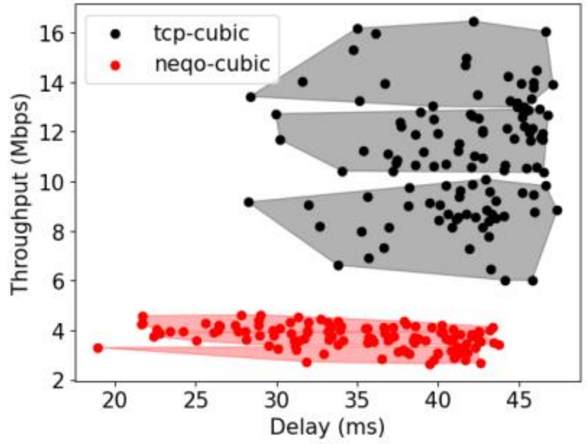
Organization	Stack	CUBIC	BBR	Reno
Linux kernel	TCP	✓	✓	✓
Facebook	mvfst [6]	✓	✓	✓
Google	chromium [8]	✓	✓	✗
Microsoft	msquic [12]	✓	✗	✗
Cloudflare	quiche [5]	✓	✗	✓
LiteSpeed	lsquic [11]	✓	✓	✗
Go	quicgo [9]	✓	✗	✓
H2O	quicly [10]	✓	✗	✓
Rust	quinn [14]	✓	✗	✓
Amazon Web Services	s2n-quic [4]	✓	✗	✗
Alibaba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

Benchmarked all QUIC stacks that were deployed, open source, and implemented some CCA.

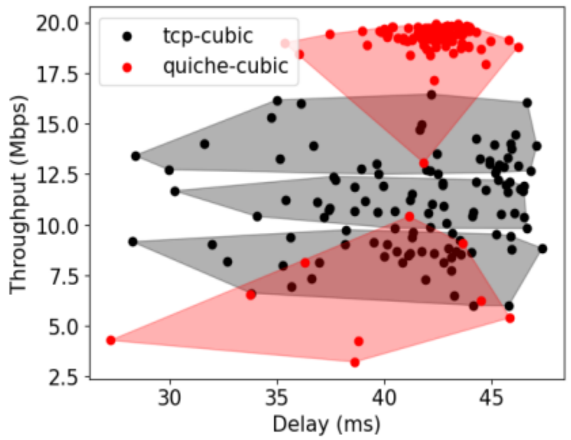
Measurement Results

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

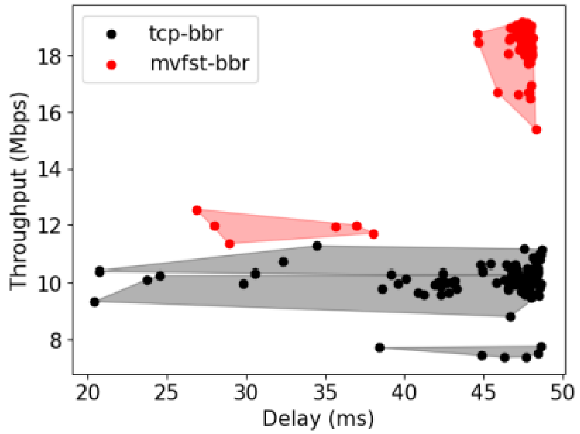
We found **7 implementations** of standard CCAs that showed poor conformance to their kernel counterparts



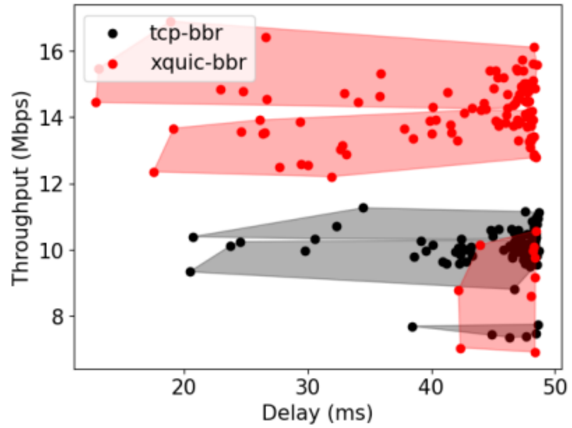
(b) neqo **CUBIC, Conf.**= 0



(c) quiche **CUBIC, Conf.**= 0.08



(e) mvfst **BBR, Conf.**= 0

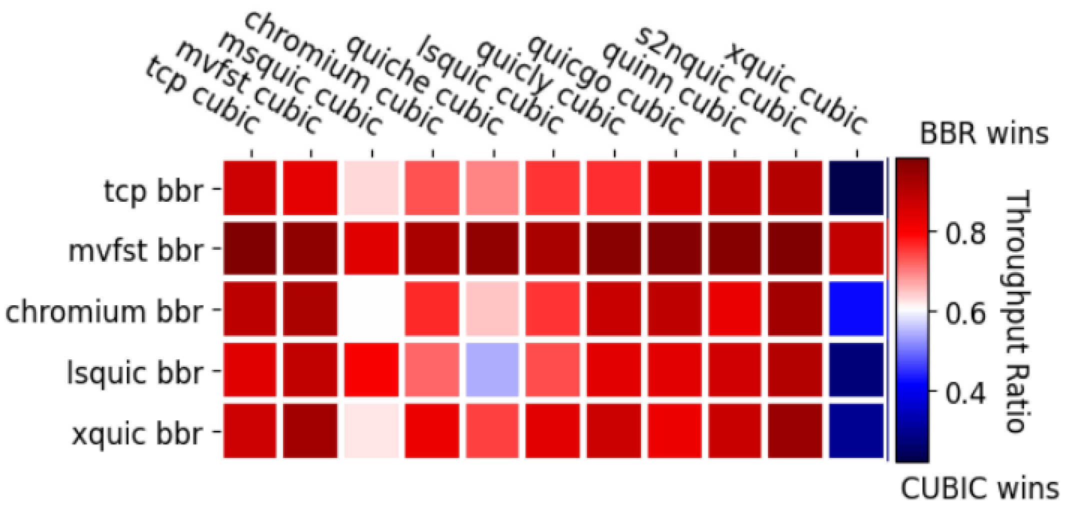


(f) xquic **BBR, Conf.**= 0.15

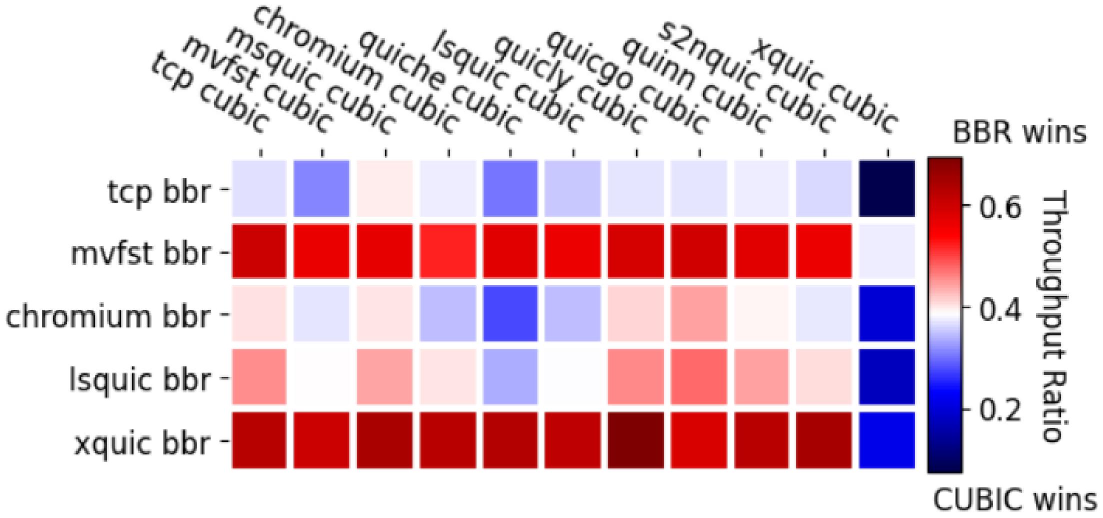
Impact: Subversion of Expectations

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 the QUIC implementation!



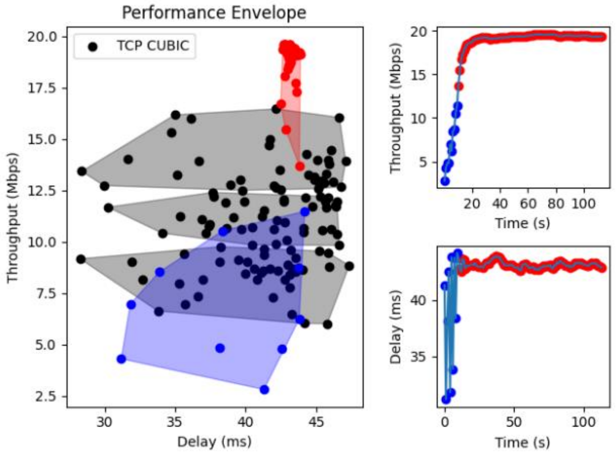
1 BDP buffer (expected to be red)



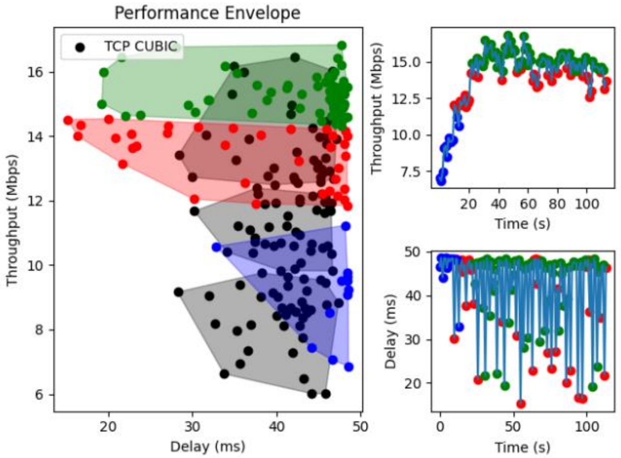
5 BDP buffer (expected to be blue)

Where does this *non-conformance* come from?

- With BBR, it's often improperly set parameters (mvfst, xquic)
- Other parts of the transport stack (Spurious loss detection in quiche)
- Often, even implementing the CCA correctly is not always enough (xquic Reno)



quiche CUBIC original
Conformance = 0.08



quiche CUBIC modified
Conformance = 0.55

Putting it all in context

- In its current scope, 5033bis recommends evaluating the deployability of a congestion control algorithm.
- There is a possible direction where we attach a “standard implementation” to the RFC of every deployable congestion control algorithm and then measure the conformance of every other implementation against this standard implementation.
- How do we deal with differently tuned CCAs?
- How do we police the deployment of *safe* CCAs?

Thank you for your time!