RESTRUCTURING ENDPOINT CONGESTION CONTROL

Akshay Narayan, Frank Cangialosi, Deepti Raghavan, Prateesh Goyal, Srinivas Narayana, Radhika Mittal, Mohammad Alizadeh, Hari Balakrishnan

CSAIL

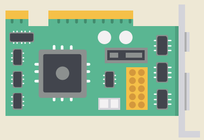
CONGESTION CONTROL

APPLICATION

Data

OS TCP

Data



NETWORK

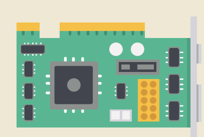
DATAPATHS

APPLICATION

Data

RDMA | SMARTNIC FPGA | MTCP DPDK | OS TCP | UDP | QUIC | DCCP

Data



NETWORK

NEW ALGORITHMS

Nimbus
Indigo
Vivace
XCP RCP DCTCP RC3 ABC
H-TCP FAST LEDBAT NV BBR
Vegas Veno Hybla Illinois Remy PCC Copa
Reno EBCC Westwood Compound Sprout TIMELY
Tahoe NewReno Binomial BIC Cubic PRR DCQCN

1987 1998 2001 2010 2018

ALGORITHM COMPLEXITY

Sprout (NSDI 2013): Bayesian forecasting
Remy (SIGCOMM 2013): Offline learning
PCC / PCC Vivace (NSDI 2015 / NSDI 2018): Online learning
Indigo (Usenix ATC 2018): Reinforcement learning

CROSS PRODUCT OF SADNESS

H-TCP Veno Hybla
TIMELY XCP Westwood
Compound Sprout EBCC BIC
Cubic PRR Binomial Nimbus
DCQCN Reno Vegas Indigo
Tahoe NewReno Vivace
RCP DCTCP RC3 ABC
FAST LEDBAT NV BBR
Illinois Remy PCC Copa

×

OS TCP
USERSPACE
RDMA
SMARTNIC
FPGA
MTCP
DPDK
QUIC
DCCP

CROSS PRODUCT OF SADNESS

A PCC-Vivace Kernel Module for Congestion Control

Nathan Jay*, Tomer Gilad , Nogah Frankel****

Tong Meng*, Brighten Godfrey*, Michael Schapira**

Jae Won Chung*, Vikram Siwach***, Jamal Hadi Salim*****
University of Illinois Urbana-Champaign*, Hebrew University of Jerusalem in Israel**, Verizon***

Abstract

The introduction of a high performance packet scheduler to the Linux kernel and modular congestion control system from BBR makes it possible to draw research congestion control algorithms into the Linux kernel. In this paper, we discuss the introduction of the PCC family of congestion control algorithms into the Linux kernel. We implement both loss- and latency-based congestion control using the rate-based PCC architecture and discuss possible interfaces for choosing congestion control parameters.

Keywords

Linux, networking, TCP, low latency, PCC

Introduction

Research on Internet congestion control has produced a variety of transport layer implementations in the past decades (*e.g.*, [6, 3, 4, 2, 10, 1, 8], *etc.*). Many research algorithms have stayed in the realm of research because of former challenges in implementing congestion control in modern operating systems. Thankfully, the recent introduction of rate-

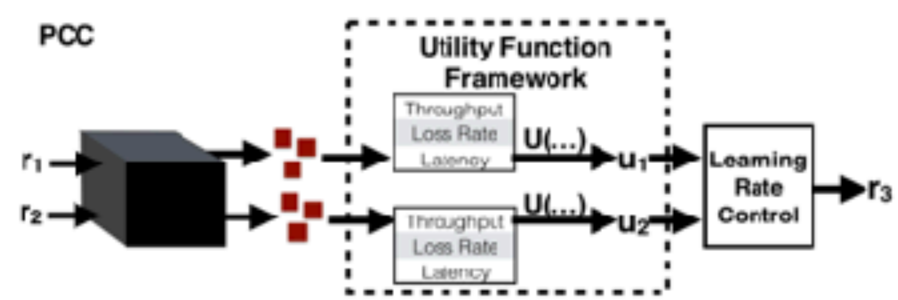
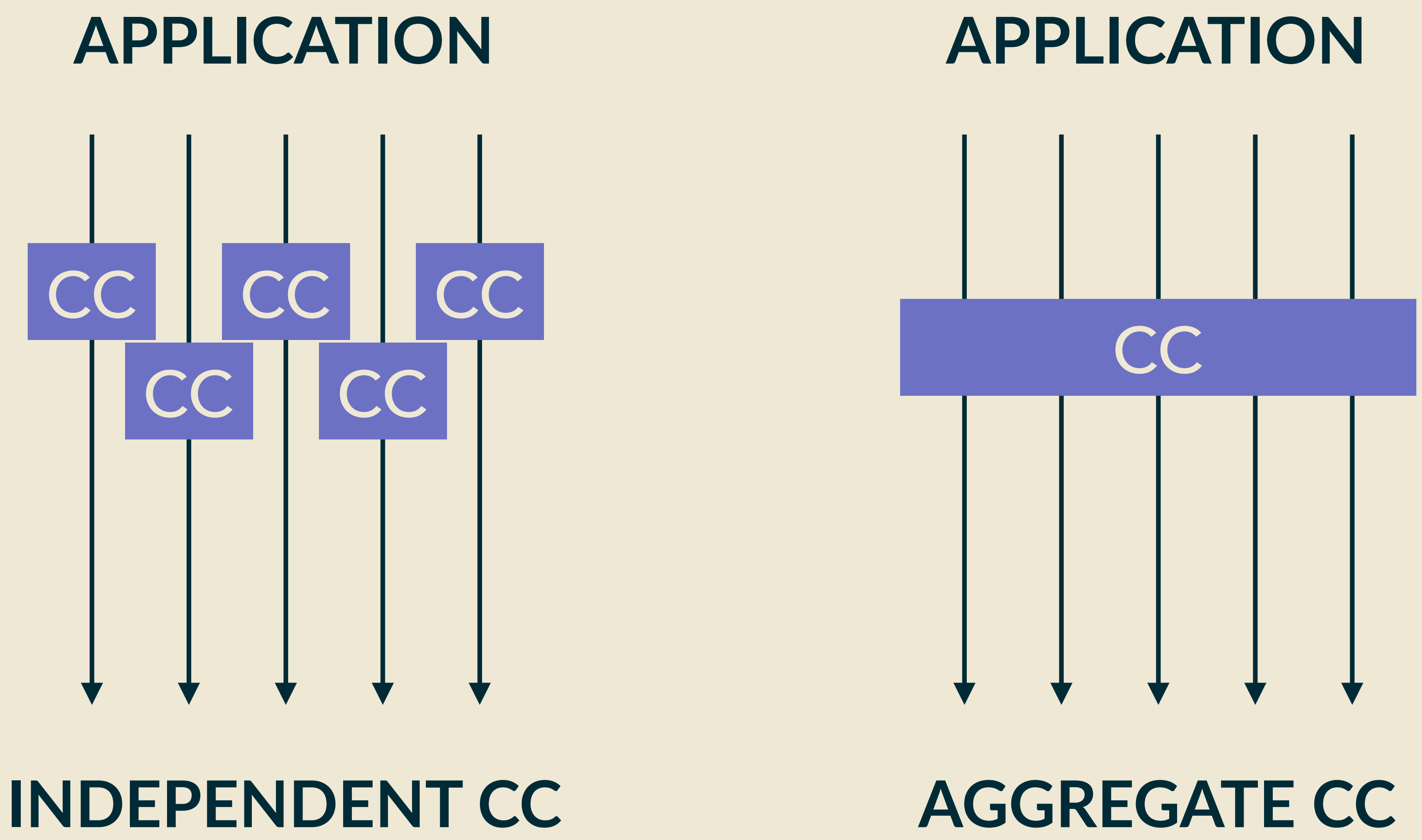


Figure 1: PCC Architecture

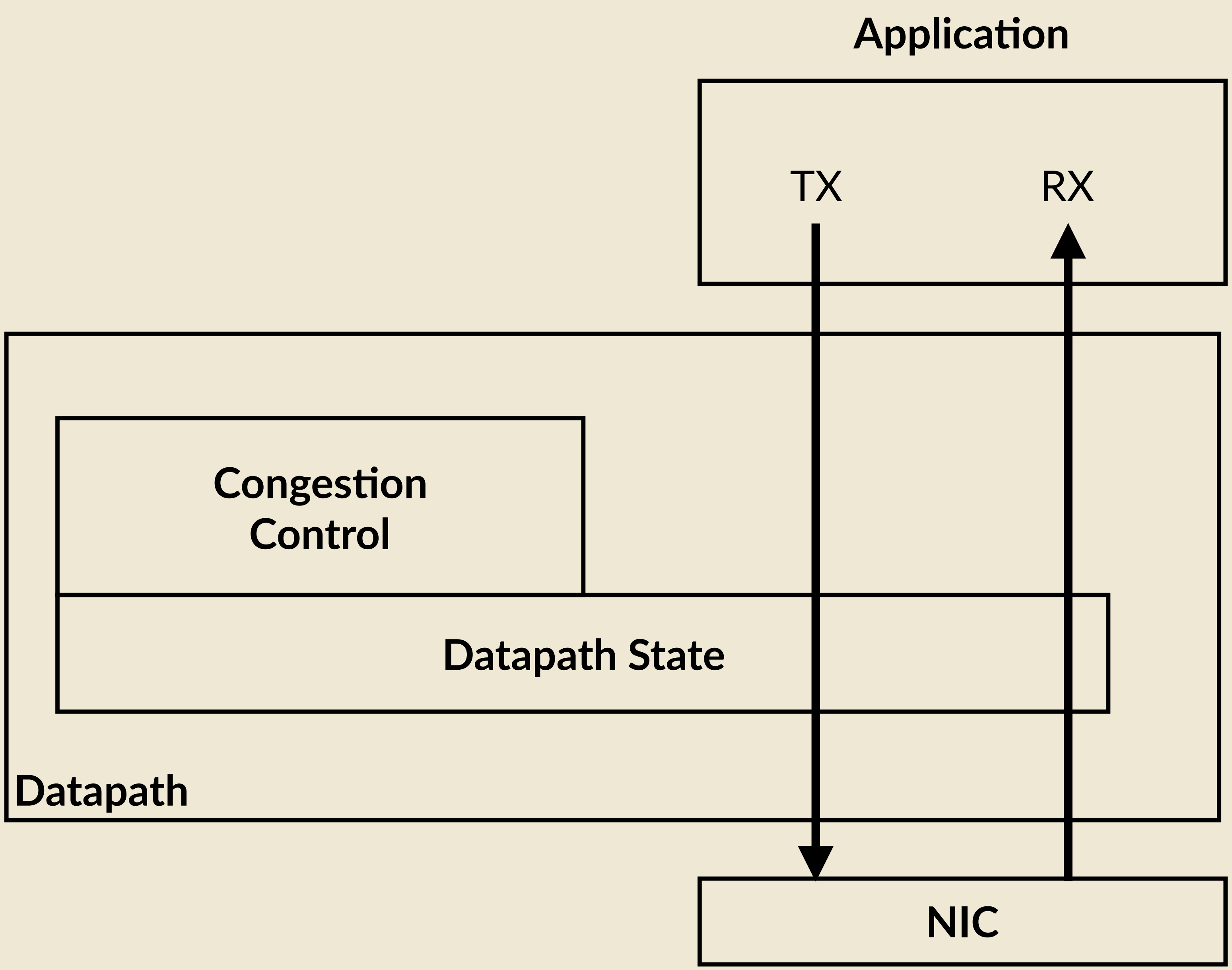
may have different optimal operating points for throughput and latency. Often, the only way for network operators or developers to choose different operating points is to choose a completely different congestion control algorithms. Unfortunately, the objective of each congestion control algorithm may not be clear, forcing network operators to test a variety of algorithms and develop in-house implementations to meet their needs.

Recognizing these challenges for congestion control, and the great opportunity afforded by the improved Linux networking code, we implement PCC-Vivace [4] with both loss- and latency-based utility functions in the Linux kernel and

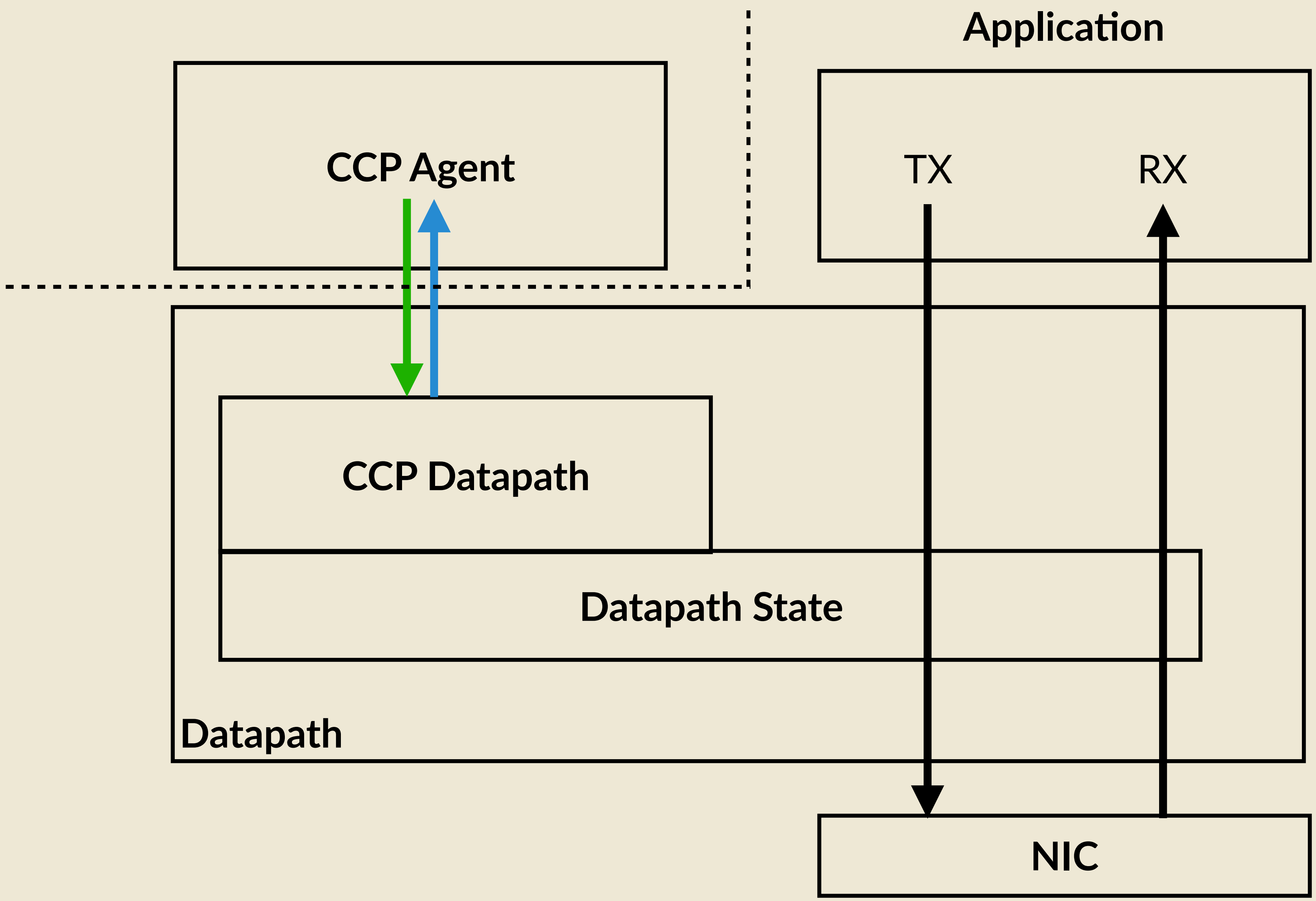
NEW CAPABILITIES



CURRENT DESIGN



CONGESTION CONTROL PLANE

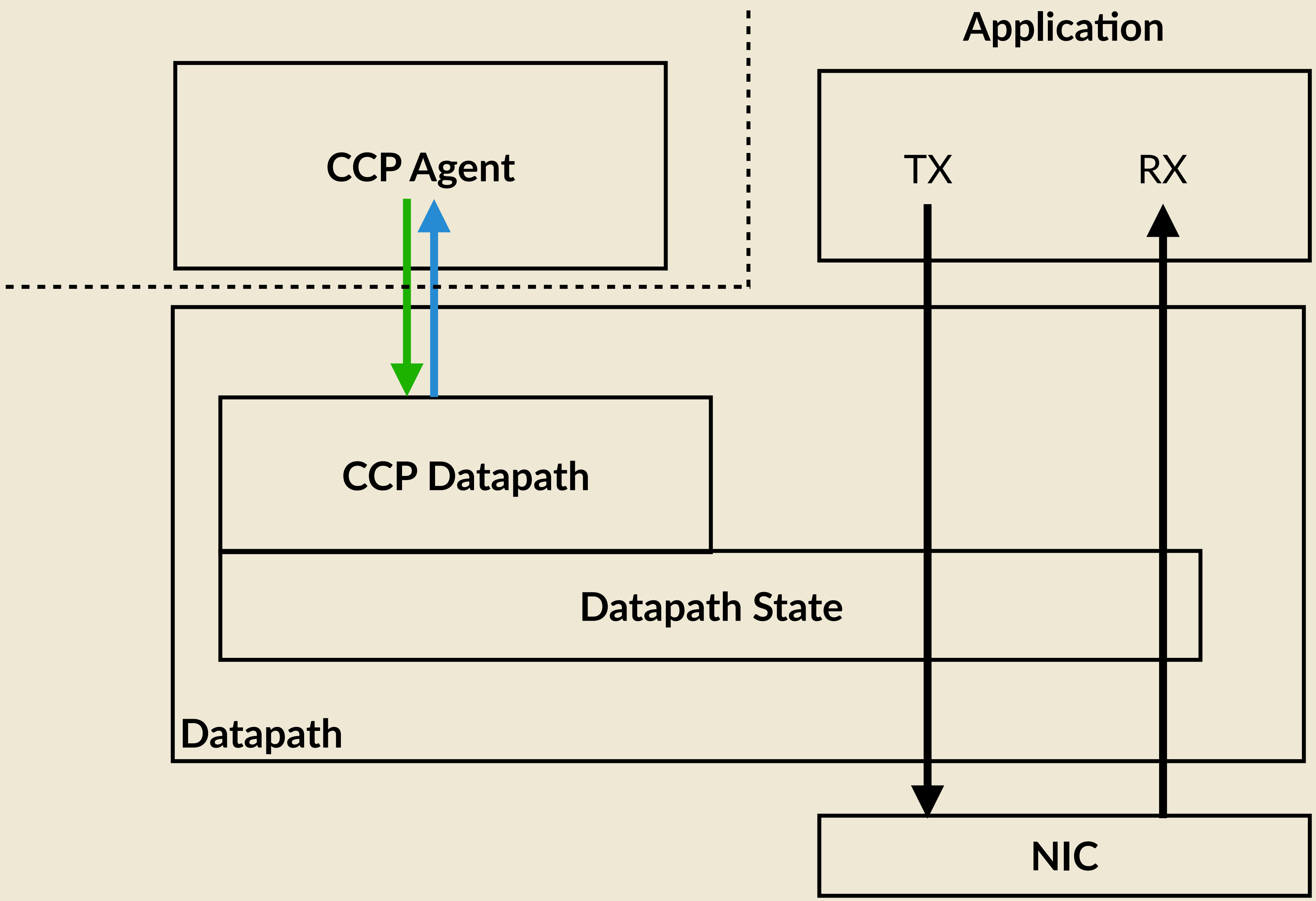


CONGESTION CONTROL PLANE

Write-once,
run-anywhere

Sophisticated
algorithms

New
capabilities

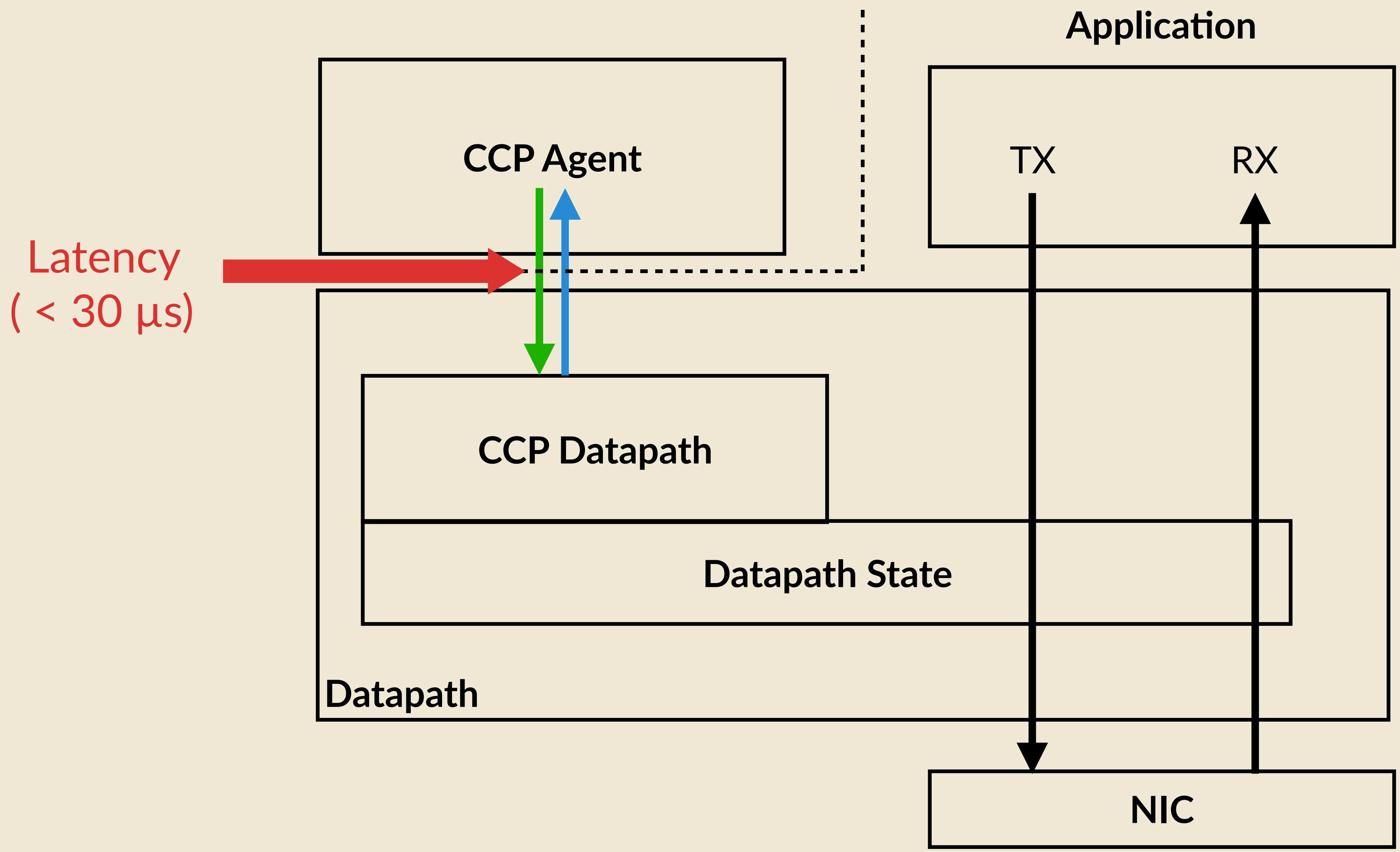


LATENCY TRADEOFF

Write-once, run-anywhere

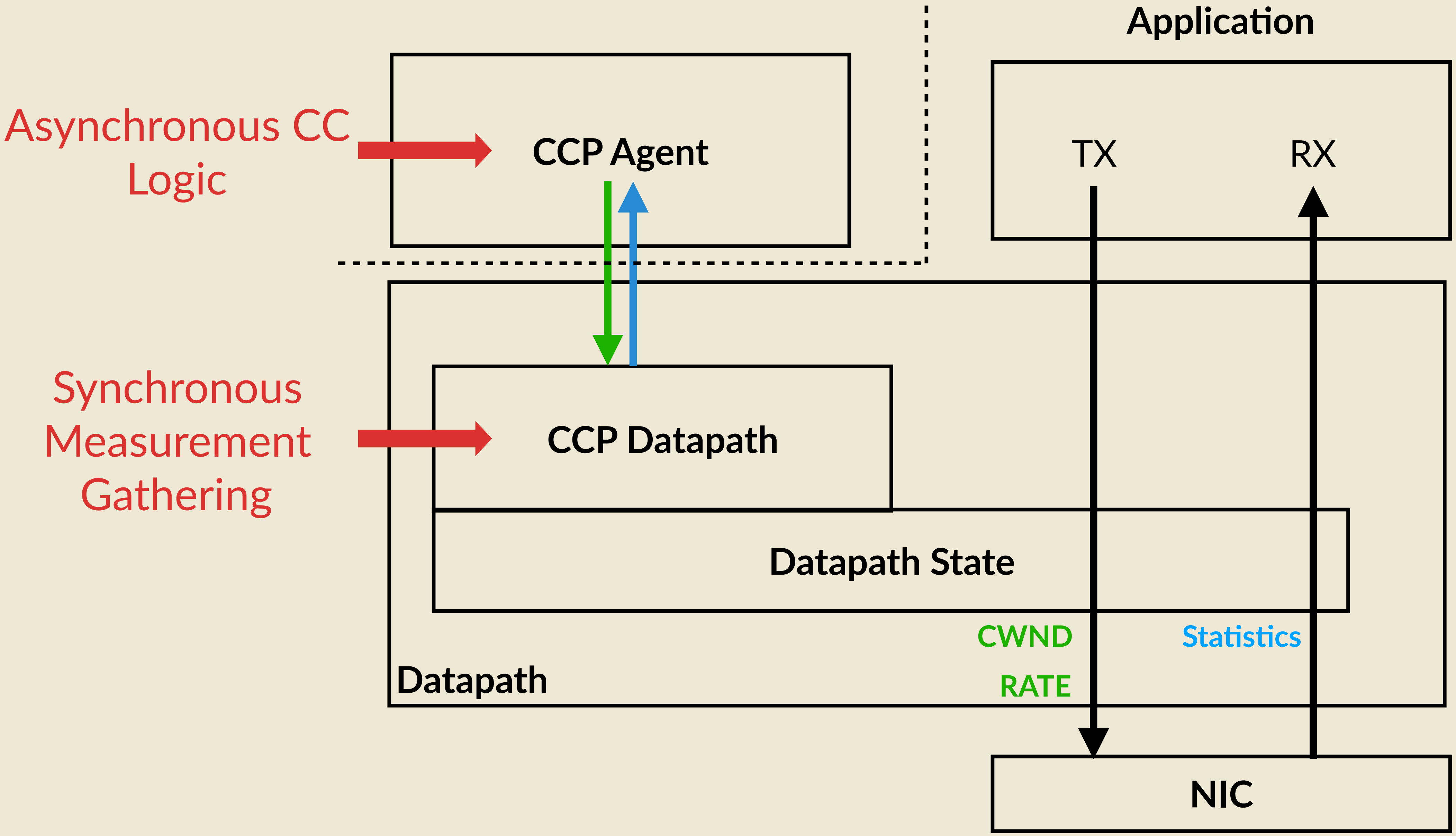
Sophisticated algorithms

New capabilities

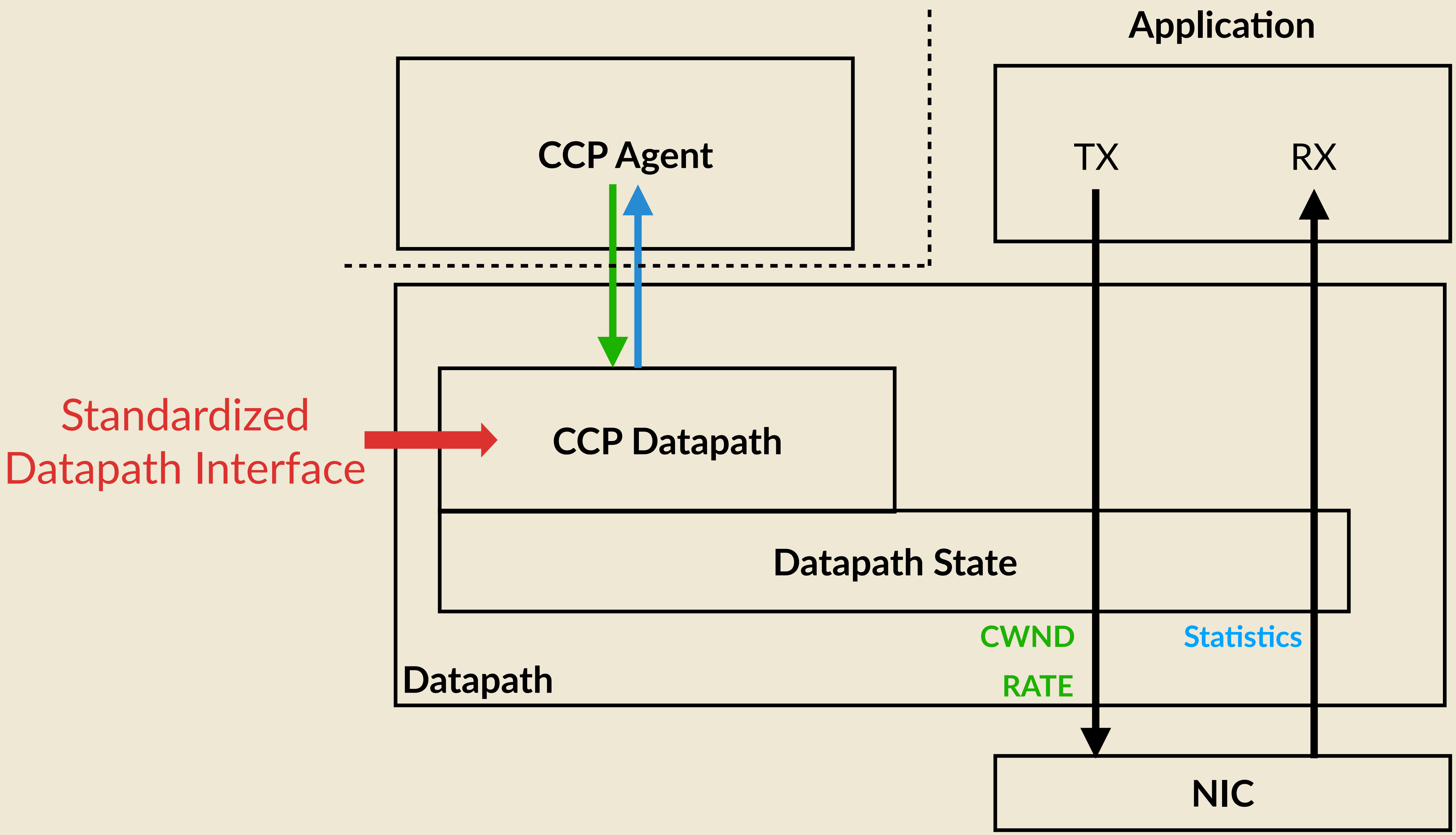


SPLIT IMPLEMENTATION

Split CC performs similarly to datapath-native



SPLIT IMPLEMENTATION

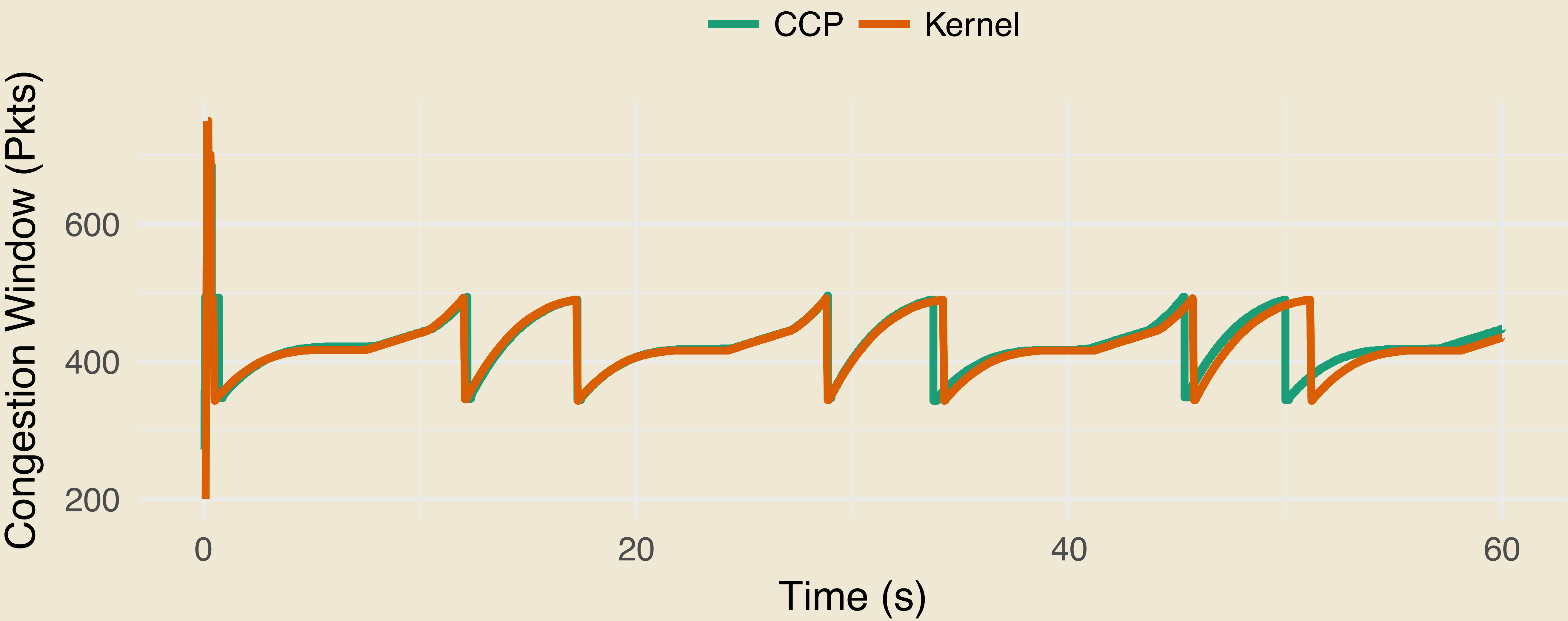


MEASUREMENT PRIMITIVES

Measurement timestamp
In-order acked bytes
Out-of-order acked bytes
ECN-marked bytes
Lost bytes
Timeout occurred
RTT sample
Bytes in flight
Outgoing rate
Incoming rate

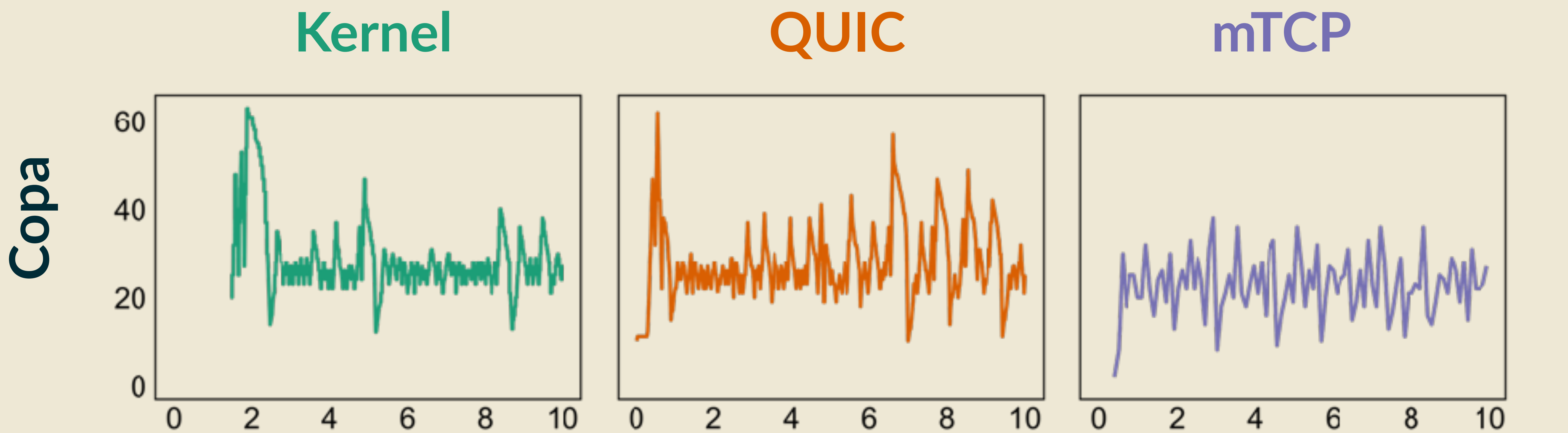
Demo

CUBIC WINDOW DYNAMICS

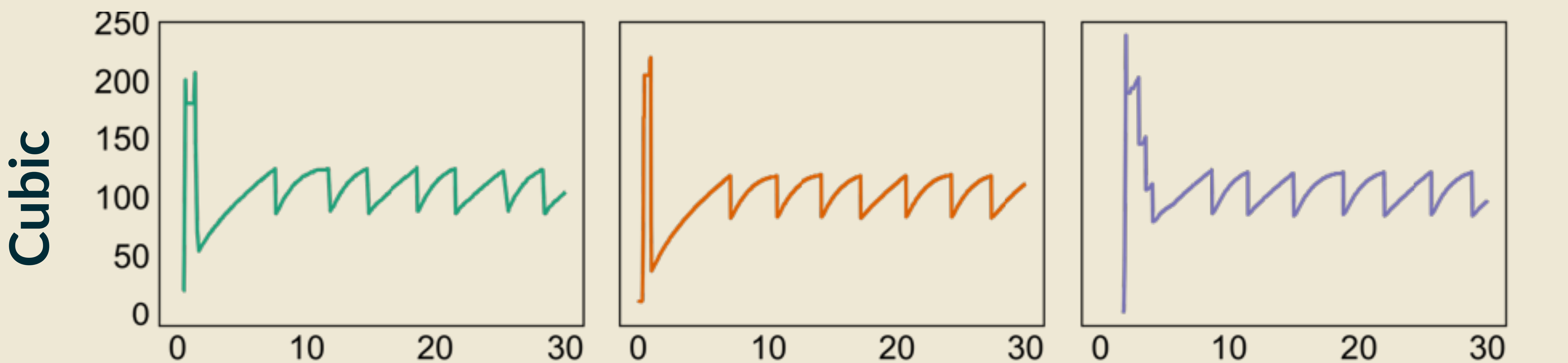


96 Mbit/s, 20ms link RTT

WRITE-ONCE RUN-ANYWHERE

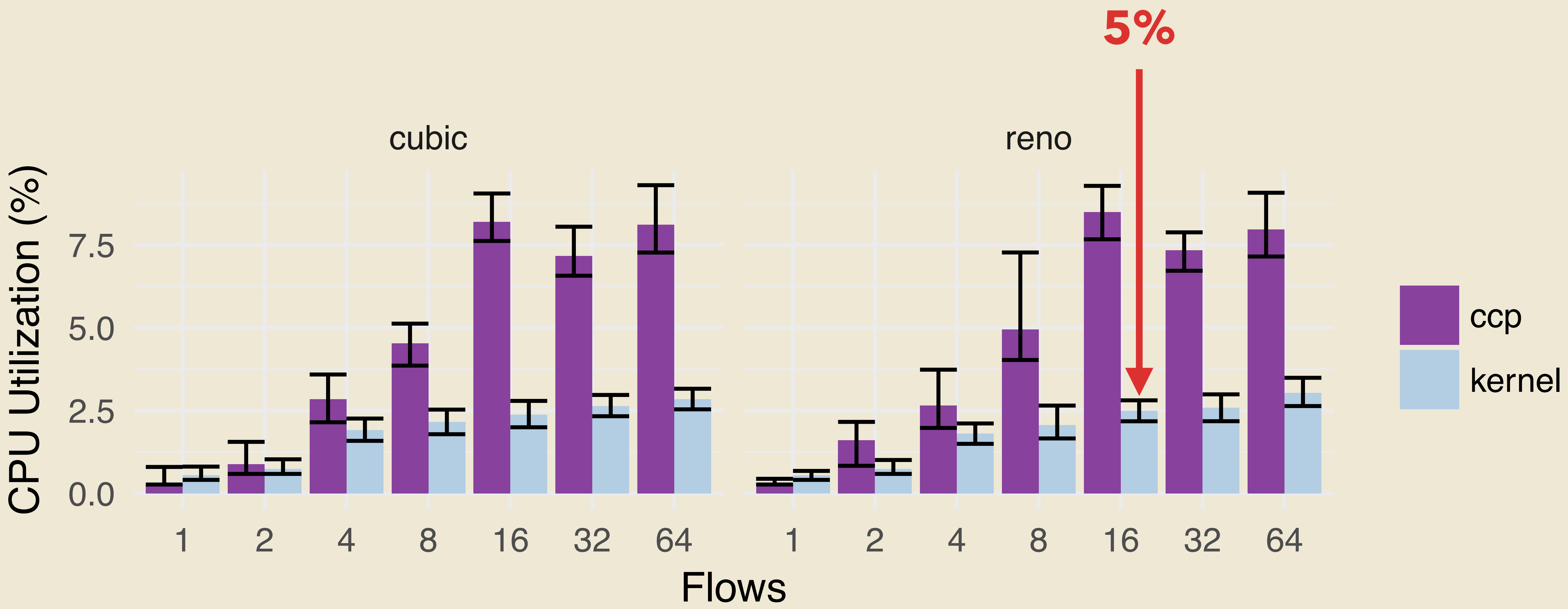


Link: 12 Mbit/s, 20ms RTT



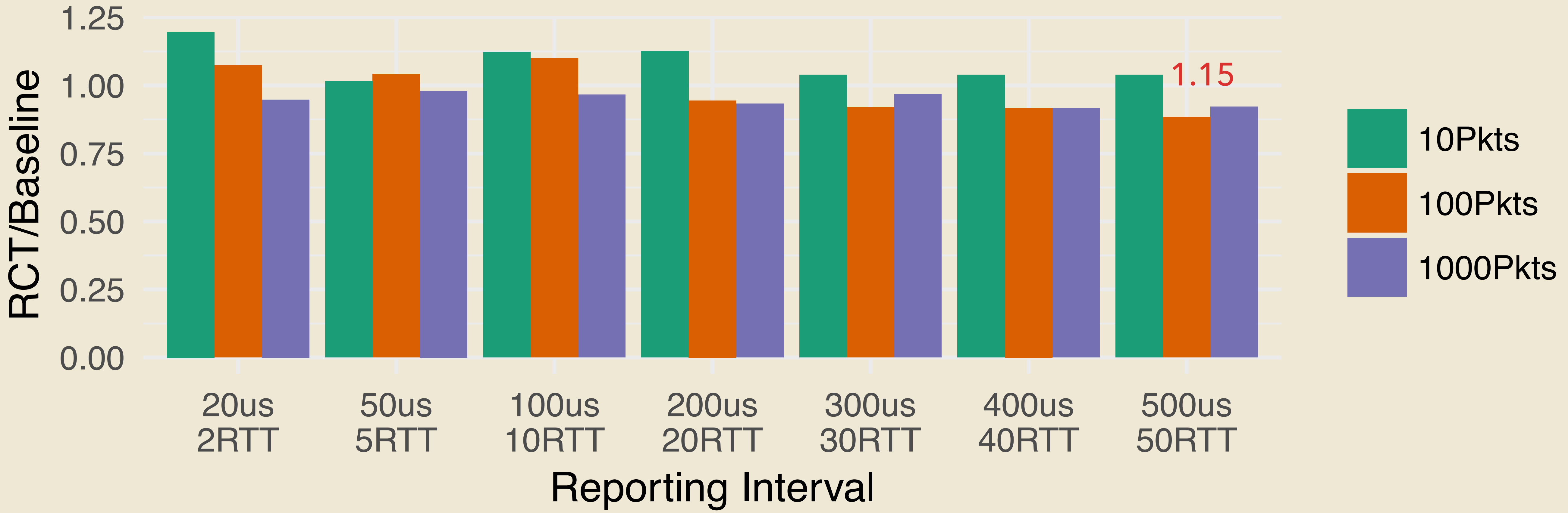
Link: 24 Mbit/s, 20ms RTT

STRESS TEST



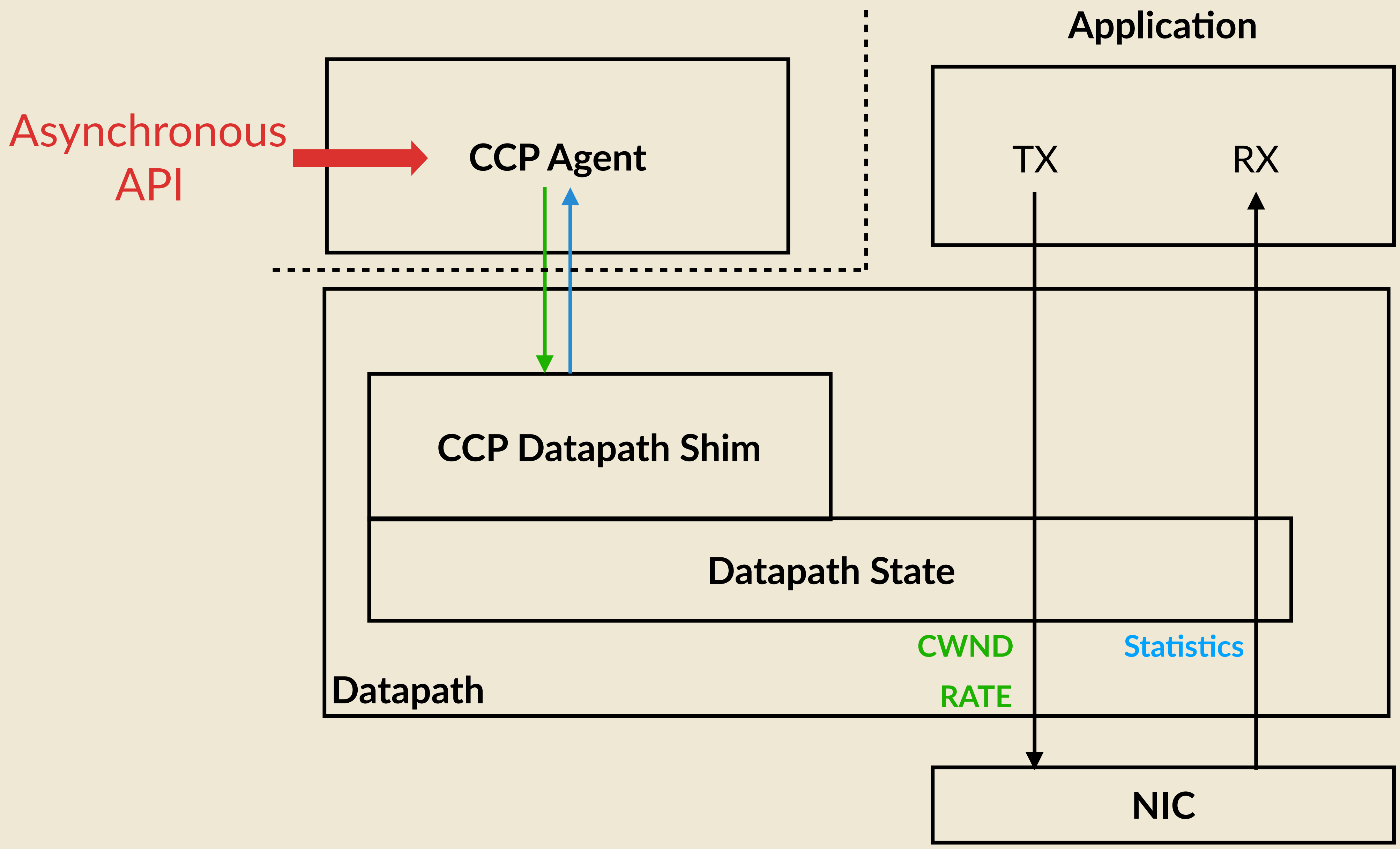
Link: 10Gbit/s, 100μs RTT

LOW-RTT SCENARIOS

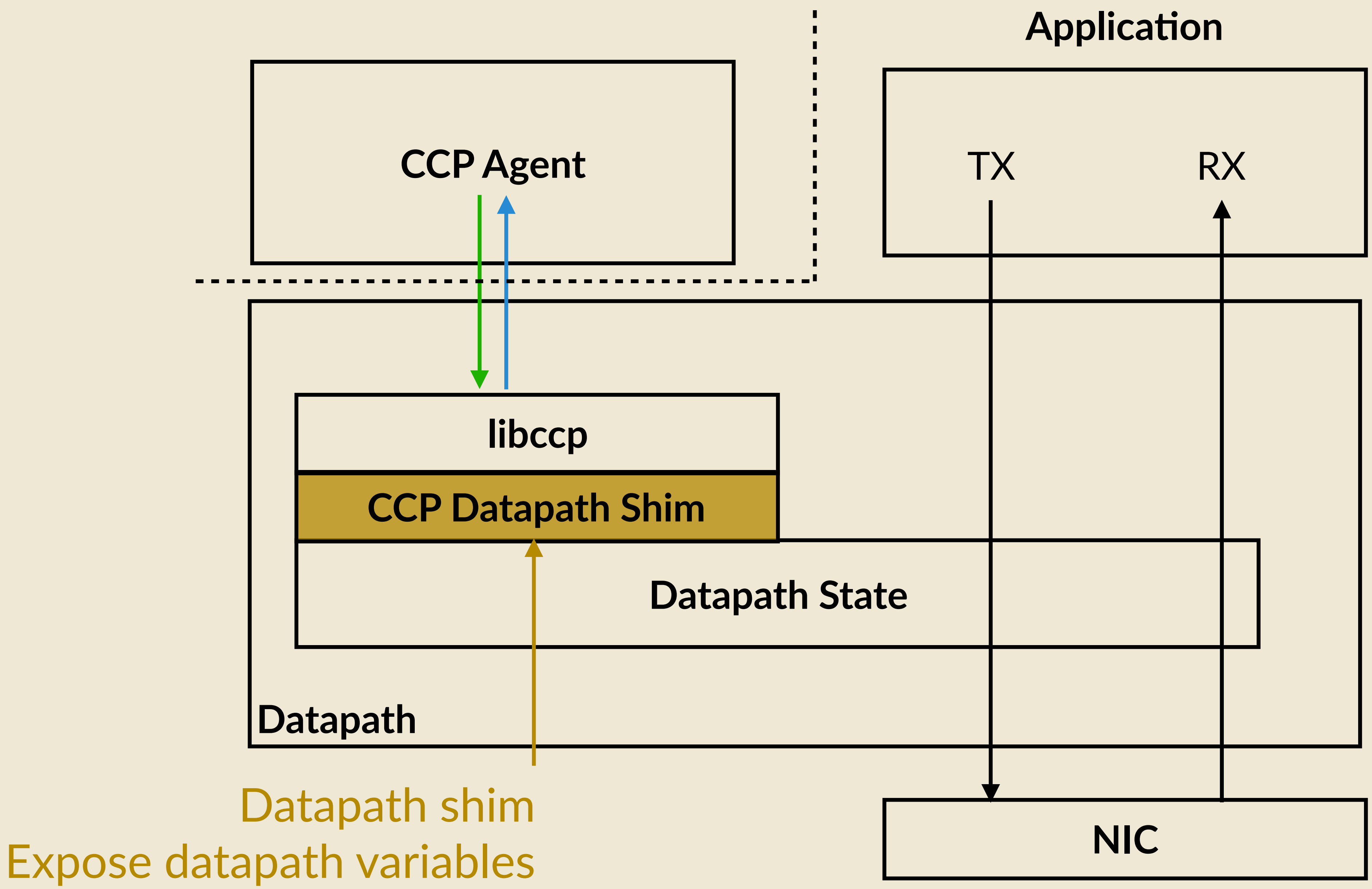


Link: 10Gbit/s, 10 μs

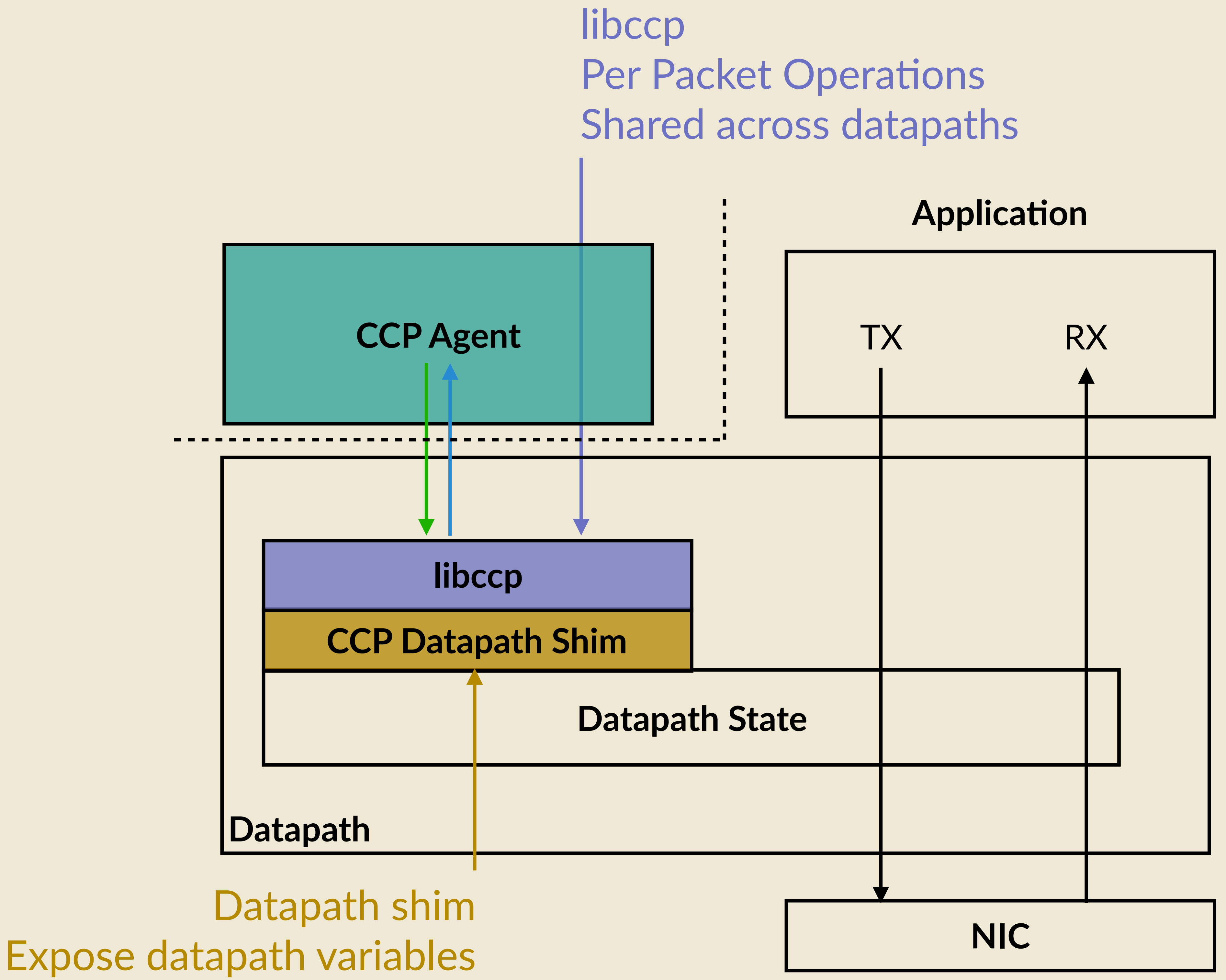
DESIGN: FAST AND SLOW PATH



SPLIT IMPLEMENTATION



SPLIT IMPLEMENTATION



BBR SPLIT IMPLEMENTATION

Asynchronous:

- Every report
  - Calculate new rate based on measurements
  - Handle switching between modes

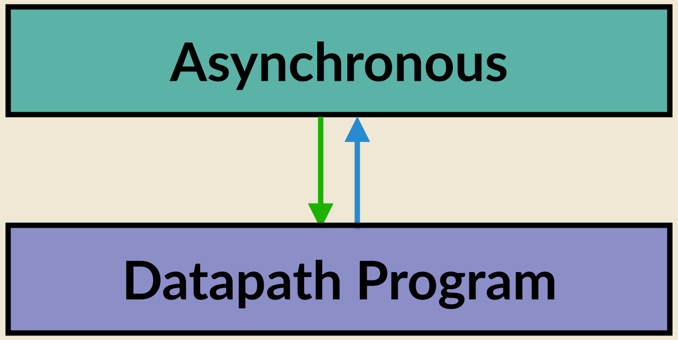
Datapath Program:

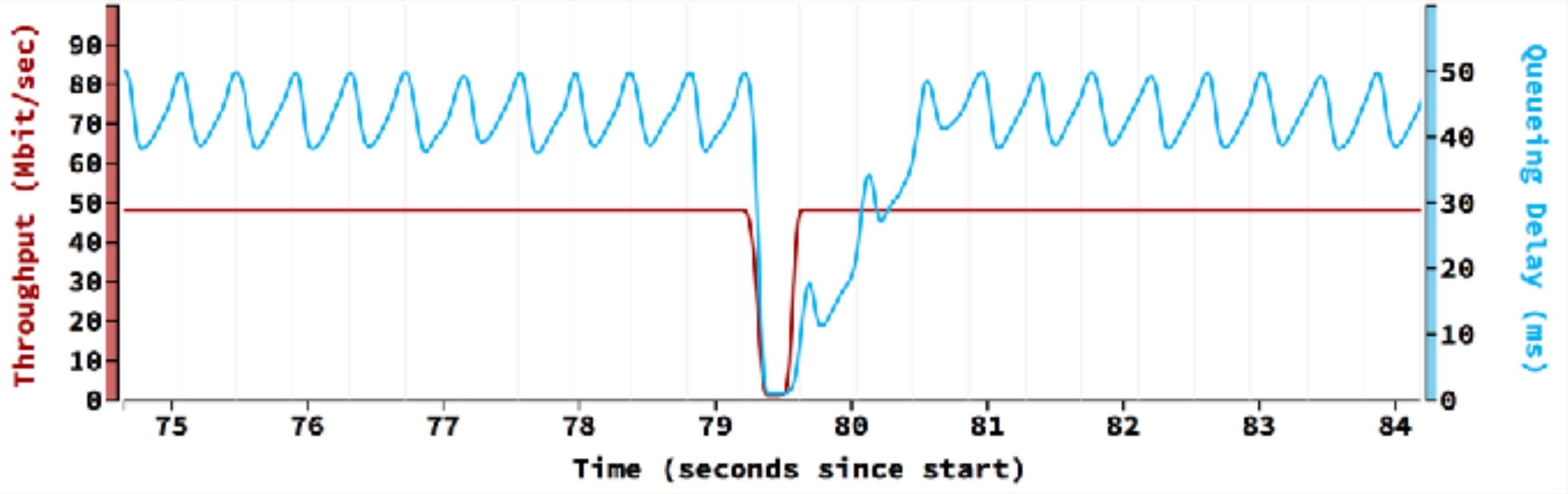
- Per ACK measurements
- Pulse:

  Rate = 1.25 x bottle rate
- After 1 RTT:

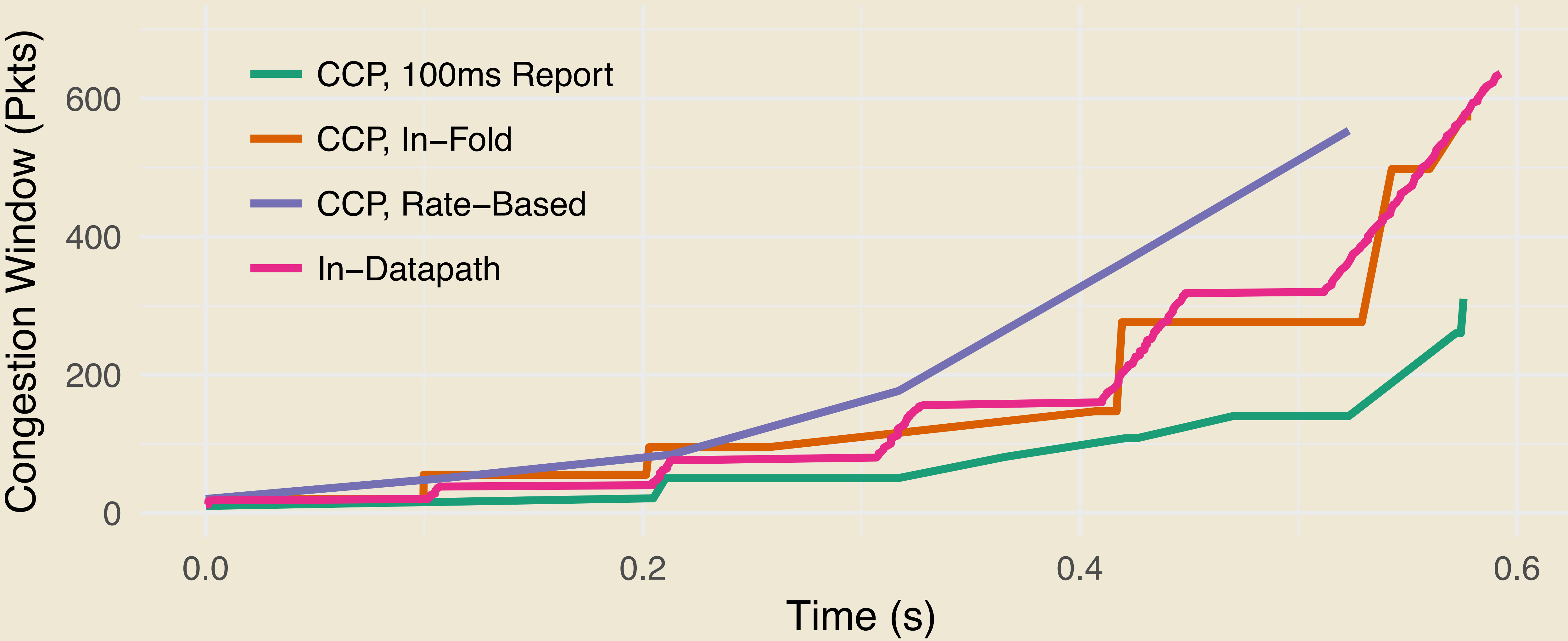
  Rate = 0.75 x bottle rate
- After 2 RTT:

  Rate = bottle rate
- After 8 RTT: repeat





SLOW START



48Mbit/s, 100ms link RTT

NEW CAPABILITIES

Sophisticated algorithms

Rapid prototyping

CC for flow aggregates

Application-integrated CC

Dynamic, path-specific CC

NEXT STEPS

- More algorithms!
- Hardware datapaths
- Impact of new API on congestion control algorithms
- New capabilities using CCP platform

CURRENT STATUS

- Datapaths (libccp):
  - Linux TCP
  - QUIC
  - mTCP/DPDK
- CCP Agent (portus)

Reproduce our results and build your own congestion control at

github.com/ccp-project

Extra Slides

EBPF

Front-End
(Language)

- Event-driven semantics
- Explicit reporting model

Back-End
(Datapath)

- Congestion control enforcement
- Direct access to socket state

```
(def (Report (acked 0)))
(when true
  (:= Report.acked (+ Report.acked Ack.bytes_acked))
  (:= Cwnd (+ Cwnd Report.acked))
  (fallthrough))
(when (> Flow.lost_pkts_sample 0)
  (report))
```