RESTRUCTURING ENDPOINT CONGESTION CONTROL

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

APPLICATION

Data

OS TCP

Data

NETWORK
NEW ALGORITHMS

1987 - Vegas, Reno, Tahoe, NewReno
1998 - EBCC, Westwood, Binomial, BIC
2001 - XCP, H-TCP, Veno, Hybla
2010 - FAST, Illinois, Compound, Cubic
2018 - DCTCP, LEDBAT, Remy, Sprout, PRR, DCQCN

1987 - XCP, H-TCP, FAST, RCP, DCTCP
2001 - XCP, H-TCP, FAST, RCP, DCTCP
2010 - XCP, H-TCP, FAST, RCP, DCTCP
2018 - XCP, H-TCP, FAST, RCP, DCTCP

1987 - Nevada, Illinois
1998 - Nevada, Illinois
2001 - Nevada, Illinois
2010 - Nevada, Illinois
2018 - Nevada, Illinois

1987 - Tahoe, NewReno
1998 - Tahoe, NewReno
2001 - Tahoe, NewReno
2010 - Tahoe, NewReno
2018 - Tahoe, NewReno

1987 - Reno, NewReno
1998 - Reno, NewReno
2001 - Reno, NewReno
2010 - Reno, NewReno
2018 - Reno, NewReno
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
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-based congestion control, 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

Figure 1: PCC Architecture
NEW CAPABILITIES

APPLICATION

INDEPENDENT CC

APPLICATION

AGGREGATE CC
CONGESTION CONTROL PLANE

- Write-once, run-anywhere
- Sophisticated algorithms
- New capabilities

Diagram:
- Application
  - TX
  - RX
- CCP Agent
- CCP Datapath
- Datapath State
- Datapath
- NIC
LATENCY TRADEOFF

- Write-once, run-anywhere
- Sophisticated algorithms
- New capabilities

Latency (< 30 µs)

Datapath

CCP Agent

CCP Datapath

Datapath State

Application

TX
RX

NIC
SPLIT IMPLEMENTATION

Split CC performs similarly to datapath-native
# MEASUREMENT PRIMITIVES

<table>
<thead>
<tr>
<th>Measurement primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measurement timestamp</td>
</tr>
<tr>
<td>In-order acked bytes</td>
</tr>
<tr>
<td>Out-of-order acked bytes</td>
</tr>
<tr>
<td>ECN-marked bytes</td>
</tr>
<tr>
<td>Lost bytes</td>
</tr>
<tr>
<td>Timeout occurred</td>
</tr>
<tr>
<td>RTT sample</td>
</tr>
<tr>
<td>Bytes in flight</td>
</tr>
<tr>
<td>Outgoing rate</td>
</tr>
<tr>
<td>Incoming rate</td>
</tr>
</tbody>
</table>
Demo
CUBIC WINDOW DYNAMICS

Congestion Window (Pkts)

Time (s)

CCP  Kernel

96 Mbit/s, 20ms link RTT
WRITE-ONCE RUN-ANYWHERE

Kernel

QUIC

mTCP

Copa

Link: 24 Mbit/s, 20ms RTT

Link: 12 Mbit/s, 20ms RTT

Cubic

Link: 24 Mbit/s, 20ms RTT

Link: 12 Mbit/s, 20ms RTT
STRESS TEST

Flows

CPU Utilization (%)

0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0

1 2 4 8 16 32 64

Link: 10Gbit/s, 100μs RTT
LOW-RTT SCENARIOS

Reporting Interval

Link: 10Gbit/s, 10 µs
DESIGN: FAST AND SLOW PATH

Asynchronous API

CCP Agent

CCP Datapath Shim

Datapath State

Datapath

CWND

Statistics

NIC

TX

RX

Application

Rate
SPLIT IMPLEMENTATION

Datapath

CCP Agent

libccp

CCP Datapath Shim

Datapath State

TX

RX

Application

Datapath shim

Exposé datapath variables

NIC
SPLIT IMPLEMENTATION

libccp
Per Packet Operations
Shared across datapaths

Datapath shim
Expose datapath variables
### 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

- CCP, 100ms Report
- CCP, In-Fold
- CCP, Rate-Based
- In-Datapath

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

```lisp
(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))
```