On-Ramp: Using Precise Packet Delays to Pause Flows at the Network’s Edge

Shiyu Liu, Ahmad Ghalayini, Mohammad Alizadeh*, Balaji Prabhakar, Mendel Rosenblum, Anirudh Sivaraman+
Stanford University  *MIT  +NYU
June 11, 2020
Congestion Management: Background

• In Wide Area Networks (WAN)
  • Key concerns: convergence time, stability, fairness, etc.

• In Data Center Networks (DCN)
  • DCNs are much better networks: Low RTTs, fat pipes, largely homogeneous
  • Higher expectations: Apps want extremely high bandwidth and very low latency simultaneously
Limitations of existing CC algorithms and transport protocols in DCN

• Recent CC algorithms and transport protocols show impressive performance in *on-premises data centers*:
  • Signals from switches: Explicit Congestion Notification (ECN), In-Band Network Telemetry (INT) (DCTCP, DCQCN, QCN, HPCC)
  • Network support for packet scheduling (pFabric, PIAS, QJump, TIMELY, pHost, Homa) or packet trimming (NDP)

• But they cannot be deployed by *cloud users* because
  • The current VM abstraction in public clouds
    • hides *in-network signals*
    • does not expose the *network controls inside and below hypervisors* to VMs

• Existing solutions available to cloud users (like CUBIC) incur significant performance penalties
  • Especially under incast-type loads
Our goal

• Develop a simple mechanism that *cloud users* can deploy on their own to improve performance, with *no in-network support*.

• Focus primarily on detecting and handling transient congestion.
  • Most CCs perform well in the long term: high throughput, fairness, etc.
  • Transient, like incast, is difficult to handle since senders must react very *quickly* and *forcefully* to prevent packet drops
    • which is in conflict with the stable convergence of CC
    • Existing solutions (reserve buffer/bandwidth headroom, PFC) require in-network supports
Why decoupling the handling of transience and equilibrium?

12 servers send TIMELY long flows to 1 server. 2 flows start at t=0. The other 10 flows start at t=200ms.

- It’s difficult to perform well in both transience and equilibrium if using a single set of parameters of CC.

β = 0.8

61% of line rate is utilized.

1.6ms

β = 0.3

2.1ms

93% of line rate is utilized.

β = 0.2

Link is fully utilized.

Why decoupling the handling of transience and equilibrium?

12 servers send **DCQCN** long flows to 1 server.  
2 flows start at t=0. The other 10 flows start at t=200ms.

- It’s difficult to perform well in both transience and equilibrium if using a single set of parameters of CC.

Yibo Zhu, et al. “Congestion Control for Large-Scale RDMA Deployments”. SIGCOMM ’15
Our proposal of On-Ramp

- On-Ramp: if the one-way delay (OWD) of the most-recently acked packet > threshold $T$, the sender temporarily holds back the packets from this flow.
  - A gate-keeper of packets at the edge of the network.
  - Decoupling transience from equilibrium congestion control

- Can be coupled with any CC, requires only end-host modifications.

- In addition to public cloud, On-Ramp can also improve network-assisted CC.
Outline

• Design
  • Strawman proposal
  • Final version

• Implementation

• Evaluation
  • Google Cloud
  • Cloudlab
  • ns-3

• Deep Dive
Strawman proposal for On-Ramp

• For a flow, if the measured $OWD > T$, the sender pauses this flow until $t_{Now} + OWD - T$.

• Hope: drain the queue down to $T$

• With feedback delay $\tau$: pause much longer than needed
  • Queue undershoots $T$
  • May cause under-utilization
Final version of On-Ramp

• Need to pause less. Two factors to consider:
  • **Feedback delay**: it is possible the sender also paused this flow when the green pkt was in flight, but the latest signal “OWD of the green pkt” hasn’t seen the effects of these pauses.
  • **Concurrency**: to account for the contributions to OWD from *other* senders.

• The rule of pausing needs to account for these.
Two long-lived CUBIC flows sharing a link

Strawman On-Ramp

Final version of On-Ramp
Outline

• Design
  • Strawman proposal
  • Final version
• Implementation
• Evaluation
  • Google Cloud
  • Cloudlab
  • ns-3
• Deep Dive
Implementation

• Linux kernel modules
  • End-host modifications only.
  • Easy to deploy. Hot-pluggable.
  • Incremental deployment is possible.

• ns-3
  • Emulate the NIC implementation
  • Built on top of the open-source HPCC simulator
Outline

• Design
  • Strawman proposal
  • Final version

• Implementation

• Evaluation
  • Google Cloud
  • Cloudlab
  • ns-3

• Deep Dive
Evaluation Setup

• **Environments:**
  - **VMs in Google Cloud:** 50 VMs, each has 4 vCPUs and 10G net.
  - **Bare-metal cloud in CloudLab:** 100 machines across 6 racks, 10G net.
  - **ns-3:** 320 servers in 20 racks, 100G net.

• **Traffic loads:**
  - **Background:** WebSearch, FB_Hadoop, GoogleSearchRPC, load = 40% ~ 80%.
  - **Incast:** Fanout=40, each flow=2KB or 500KB, load = 2% or 20%.

• **Clock sync:**
  - Huygens for Google Cloud and CloudLab
On-Ramp in Google Cloud

- CUBIC
- WebSearch @ 40% load + incast @ 2% load (fanout=40, each flow 2KB)

Incast RCT

FCT of WebSearch traffic
On-Ramp in Google Cloud

- **BBR**
- WebSearch @ 40% load + incast @ 2% load (fanout=40, each flow 2KB)

**Incast RCT**

- **No OR**
  - mean: 4.2 ms
  - 90p: 4.7 ms
  - 95p: 5.0 ms
  - 99p: 5.6 ms

- **OR**
  - mean: 4.2 ms
  - 90p: 4.7 ms
  - 95p: 5.0 ms
  - 99p: 5.6 ms

**FCT of WebSearch traffic**

- **No OR**
  - <10KB: 3.7 ms
  - [10KB, 1MB]: 4.5 ms
  - [1MB, 30MB]: 30.0 ms

- **OR**
  - <10KB: 3.7 ms
  - [10KB, 1MB]: 4.5 ms
  - [1MB, 30MB]: 30.0 ms
On-Ramp with Network-assisted CC (ns-3)

- WebSearch @ 60% load + incast @ 2% load (fanout=40, each flow 2KB)
- Bars: mean. Whiskers: 95th percentile
Outline

• Design
  • Strawman proposal
  • Final version

• Implementation

• Evaluation
  • Google Cloud
  • CloudLab
  • ns-3

• Deep Dive
  • Decoupling the handling of transience and equilibrium
  • The granularity of control
  • Co-existence
Deep-dive 1: Why decoupling the handling of transience and equilibrium?

12 servers send TIMELY long flows to 1 server. 2 flows start at $t=0$. The other 10 flows start at $t=200\text{ms}$.

- With On-Ramp, we can react very quickly and forcefully to transient congestion, while still keep the stable convergence during equilibrium.

\[ \beta = 0.8 \]

\[ \beta = 0.2, \text{ OR threshold } T = 100\mu s \]

\[ \beta = 0.2, \text{ OR threshold } T = 50\mu s \]
Deep-dive 1: Why decoupling the handling of transience and equilibrium?

12 servers send DCQCN long flows to 1 server. 2 flows start at \( t=0 \). The other 10 flows start at \( t=200\text{ms} \).

- Rate-increasing timers \( T_i = 55\mu s \)
- Rate-decreasing timers \( T_d = 50\mu s \)

With On-Ramp, we can react very quickly and forcefully to transient congestion, while still keep the stable convergence during equilibrium.
Deep-dive 2: The Granularity of Control

• On the sender side, Generic Segmentation Offloading (GSO) affects the granularity of control by On-Ramp
• Reducing max GSO size further improves performance but with higher CPU overhead

Google Cloud, CUBIC, WebSearch @ 40% load + incast @ 2% load (fanout=40, each flow 2KB)
Deep-dive 3: Co-existence

• The Google Cloud experiment shows: cloud users can achieve better performance by enabling On-Ramp in their own VM cluster even though there may be non-On-Ramp traffic on their paths.

• Re-visit this question in CloudLab.

• Experiment setup:
  • 100 servers randomly divided into 2 groups.
  • Inside each group, run: WebSearch @ 60% load + incast @ 2% load.
  • Don’t run cross-group traffic.
  • It models 2 users renting servers in a cloud environment but don’t know each other.
Deep-dive 3: Co-existence

Case A: Both groups not use On-Ramp vs.
Case B: Group 1 uses On-Ramp, Group 2 not

- Both groups do better in Case B than Case A.
  - On-Ramp enables Group 1 to transmit traffic at the moments when Group 2 traffic is at low instantaneous load.
  - Group 2’s is also improved because Group 1 reduces the overall congestion by using On-Ramp.
Deep-dive 3: Co-existence

Case B: Group 1 uses On-Ramp, Group 2 not vs.
Case C: Both groups use On-Ramp

• Both groups are further improved in Case C.
• Group 1’s is only slightly improved from Case B to C, compared to the large improvement from Case A to B.
  • Group 1 obtains almost the same benefit from using On-Ramp whether or not Group 2 uses it.
Conclusion

• On-Ramp allows *public cloud users* to take cloud network performance into their own hands
  • No need to change either the VM hypervisor or the network infrastructure
  • Can couple with existing congestion-control algorithms

• On-Ramp contains two ideas:
  • Using synced clocks to improve network performance
  • Decoupling the handling of transience & equilibrium