Congestion Control in the Real World

Lecture @Stanford, 4/11/22

Nandita Dukkipati
nanditad@google.com

Congestion Avoidance and Control - Discussion

- What are the key problems being solved in the paper?
- Key techniques in the paper.
- If instead of Additive increment, one did a multiplicative increase, would that work; why or why not?

Congestion Avoidance and Control - Discussion

- Why is it important to get the value of RTO right - in the ballpark?
- What was a core assumption for congestion avoidance to work?

Why care about Congestion Control in Practice

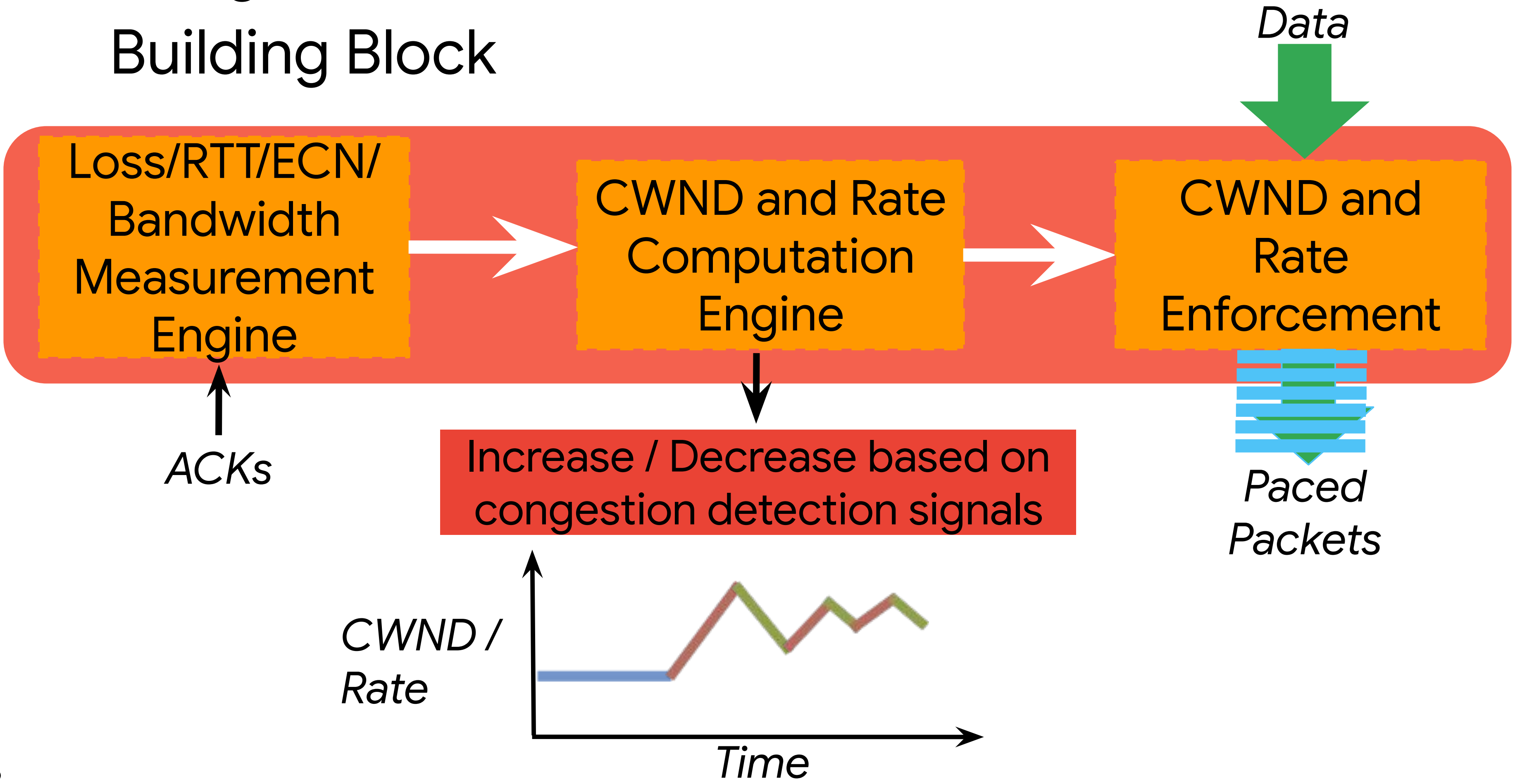
Congestion Control delivers excellent end-to-end network performance, isolation and efficiency through coupled host / NIC / switch capabilities for sharing network capacity.

Network Bandwidth Sharing at Google

Swift[1], TCP-BBR.Swift and BBR[2]	Per-flow congestion control.
BwE [3], B4 TE [4]	Centralized Control of Flow Aggregates over WAN.
Static Limits	BW configuration based on CPU cores, storage etc.

[1] Swift: Delay is Simple and Effective for Congestion Control in the Datacenter, SIGCOMM 2020
[2] BBR: Congestion-based Congestion Control, ACM Queue, 2016
[3] BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing, SIGCOMM 2015.
[4] B4: Experience with a Globally-Deployed Software Defined WAN, SIGCOMM 2013.

Congestion Control: A Fundamental Network Building Block



Congestion detection signals

End-to-end

Packet loss

Round-trip time

Bandwidth

Loss	TCP New Reno, Cubic
Delay	Vegas, Fast, BBR*, Swift

Explicit Feedback from Network

Explicit Congestion Notification

Queue lengths and differentials

Sojourn time

Available bandwidth

Link utilization

DCTCP, XCP, RCP, DCQCN, HPCC

* Also uses ECN, and max. throughput

Algorithms and Heuristics

Starting behavior

Slow Start Exponential growth

Steady State Behavior

Additive Increase and Multiplicative Decrease (AIMD)

Adaptive increase and decrease

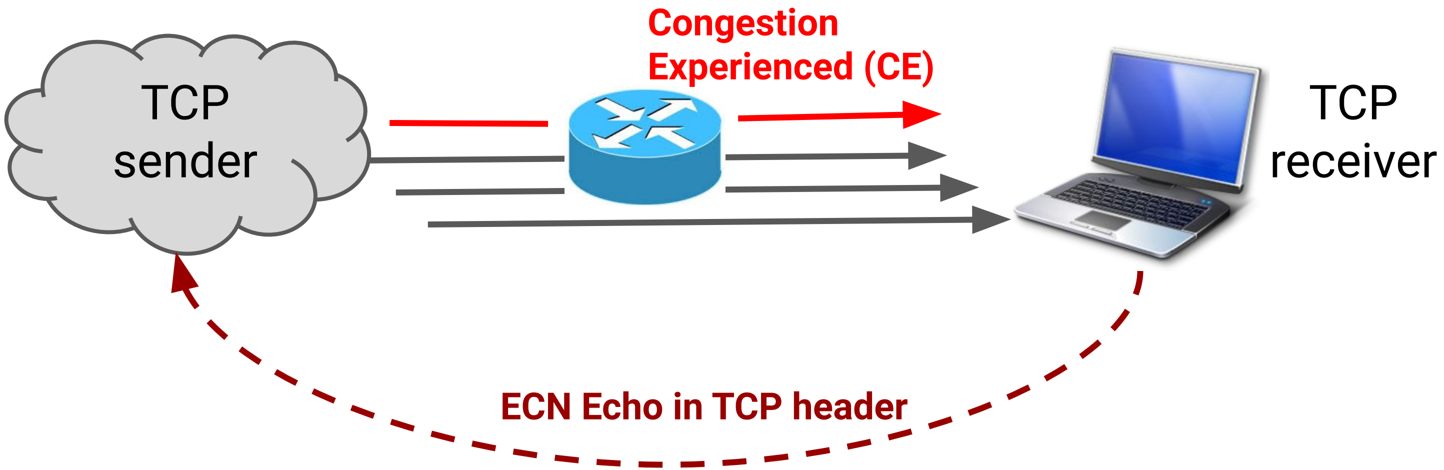
Faster Convergence

Hyper-active Increment

Cubic increase

Explicit Congestion Notification (ECN)

- Switches set "Congestion Experienced" bit on packets if the queue grows too large as per the IETF ECN standard.
- Switches inform receiver, which in turn can inform sender of congestion marks.



Datacenter TCP (DCTCP)

- Datacenter DCTCP (SIGCOMM 2010) uses ECN marks.
- Switches mark CE bit in IP header if queue > 65KB.
- Receivers reflect marks to senders (via TCP flags).
- Sender slows down according to proportion of marked packets each RTT.

$$\alpha \leftarrow (1 - g) \times \alpha + g \times F$$

$$cwnd \leftarrow cwnd \times (1 - \alpha/2).$$

$\alpha \leftarrow$ Fraction of packets that are marked

$F \leftarrow$ Fraction of packets marked in the last window of data

Reactive and Proactive Schemes

Reactive Schemes

Act on feedback gathered from acknowledgements.

Proactive Schemes

Proactively schedules network transfers.

Centralized schemes arbitrate globally for network transfers.

Switch based schemes explicitly allocate resources.

Receivers explicitly schedule transfers.

Metrics in evaluating Congestion Control

Application / User centric

Response time of application's data unit (flow-completion time, RPC completion time)

Quality of experience for Video traffic

Round-trip delay

End-to-end goodput

Network Centric

Queue delay

Link throughput/utilization

Buffer overflows

Stability

Congestion Control Challenges in Datacenter

Congestion control requirements

Transfers must complete quickly, low tail latency.

Deliver high bandwidth (>> Gbps) and low latency (<< ms).

Efficient use of CPU.

Challenges

Bursty traffic because of applications and NIC offloading.

Small buffers.

Very small round-trip delays.

Incast traffic patterns with many (>1K) flows sharing very short paths.

Kernel-bypassed transports.

Opportunities

Hardware assistance.

Less worries of interoperability with legacy.

Explicit network feedback is easier to deploy.

Centralized control is possible.

Congestion Control Challenges in Wide Area Networks

High signal variability.

Small buffers and large round-trip times.

Mismatch in transport design and underlying link layer channel, e.g., channel bandwidth is time-varying and unpredictable, deep per-user buffers, burst scheduling algorithms

Deployed congestion control algorithms are heterogeneous and unknown to senders.

Coexistence with legacy algorithms that are sensitive to packet loss.

Explicit feedback from network is rare, and difficult to deploy widely.

Swift: Delay is Simple and Effective for Congestion Control in the Datacenter

What is Swift?

Swift is a delay based congestion-control for Datacenters that achieves low-latency, high-utilization, near-zero loss implemented completely at end-hosts supporting diverse workloads like large-scale incast across latency-sensitive, byte and IOPS-intensive applications working seamlessly in heterogeneous datacenters with minimal switch support

Swift achieves ~50μs tail latency for short-flows while maintaining near 100% utilization even at 100Gbps line-rate

Why we built a new Datacenter congestion-control at Google?

New applications w/ low-latency requirements

100µs access latency at 100k+ IOPS for Flash

NVM needs 10µs latency at 1M+ IOPS

Large-scale incast for partition-aggregate workloads

IOPS intensive applications, e.g., BigQuery shuffle operation

New stacks and new sources of congestion

New networking stacks such as PonyExpress[1] exhibit different congestion behavior which is no longer limited to the fabric

E.g., endpoint congestion becomes key for a non-interrupt based stack like PonyExpress

Increasing line-rates and robustness to heterogeneity

100Gbps networking and beyond

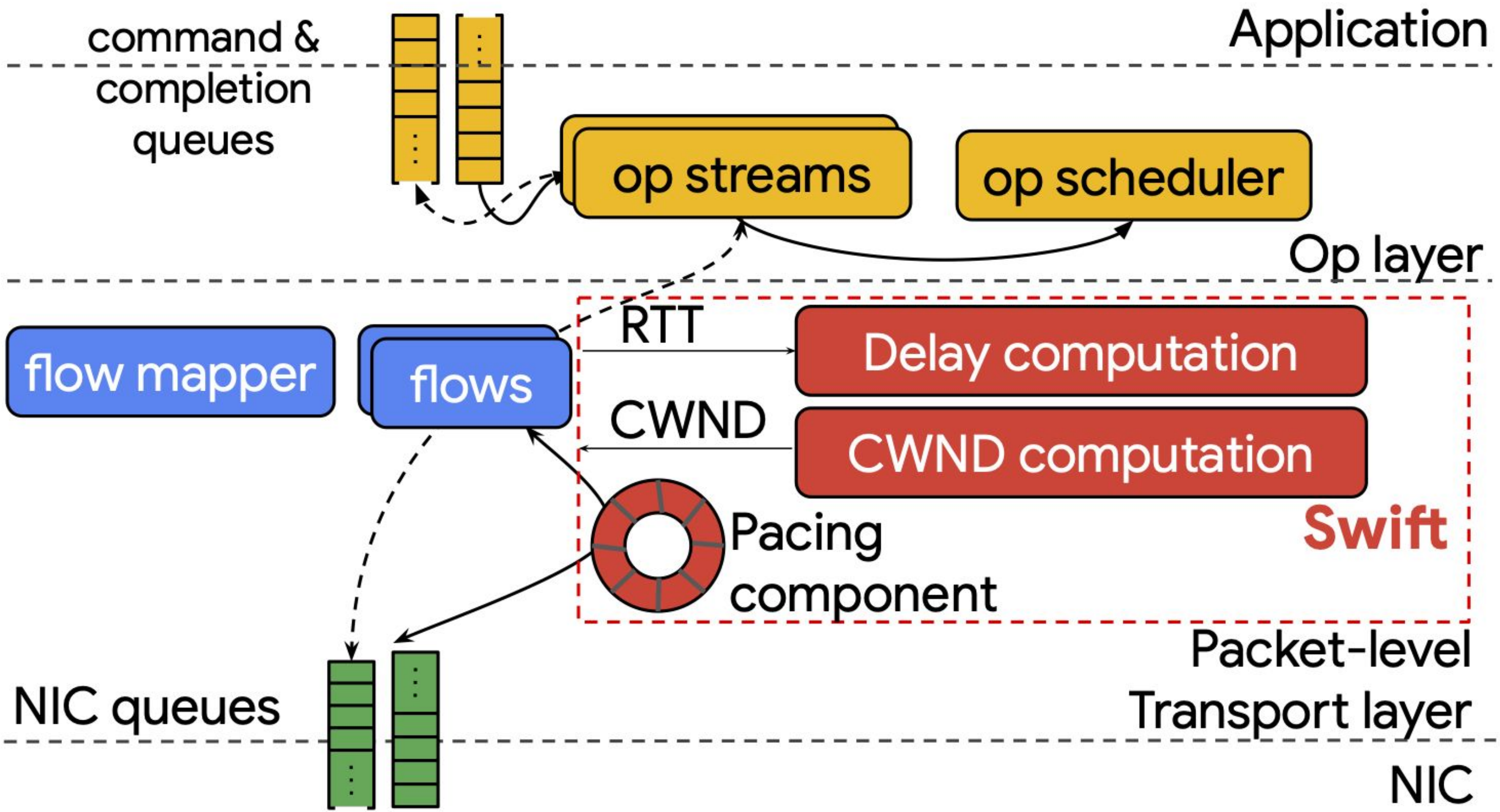
Fast reaction to congestion - queue build-up happens very quickly

[1] Snap: a Microkernel Approach to Host Networking, SOSP 2019

Design

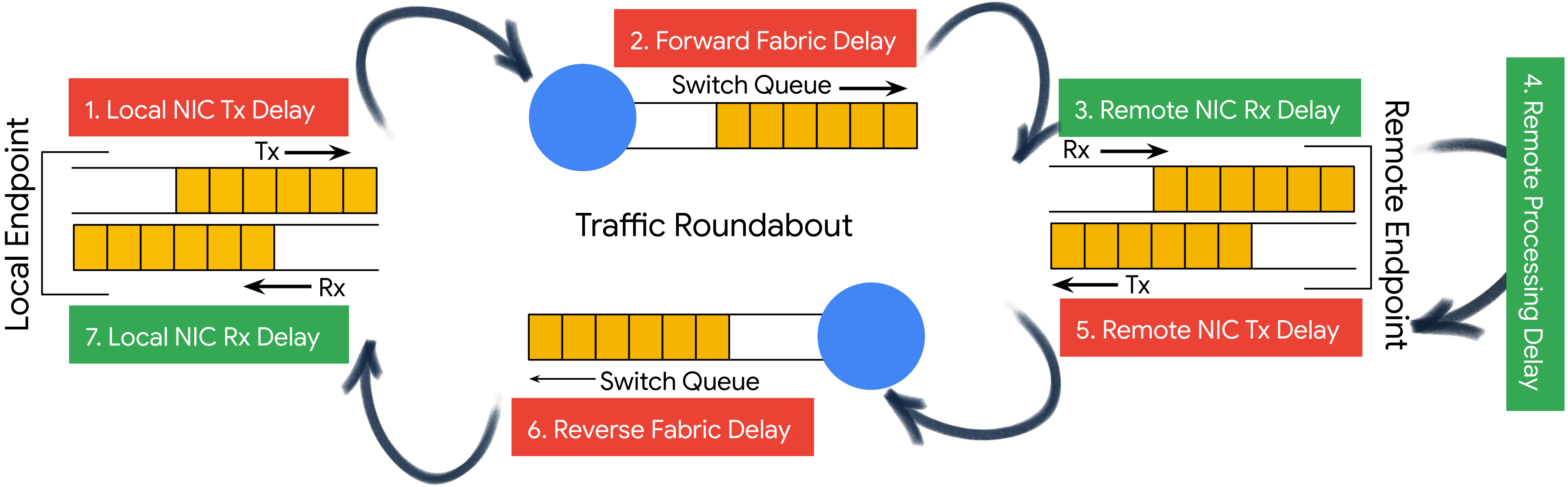
Key aspects of Swift's design

Swift in the context of PonyExpress



Swift Design

End-to-end delay decomposition of a Packet and its ACK



Swift maintains two congestion-windows (in #packets) - one based on fabric-delay and one based on endpoint-delay with their respective cwnd

Effective cwnd is the **minimum** of the two

Swift Design contd.

Simple AIMD based on a target-delay

```
if delay < Target
     increase cwnd
   (Additively)
else
     decrease cwnd
   (Multiplicatively)
```

Use of HW and SW timestamps

To provide accurate delay measurements and separate them into fabric and host components

Seamless transition b/w cwnd and rate

Swift allows cwnd to fall below 1 to handle large-scale incast

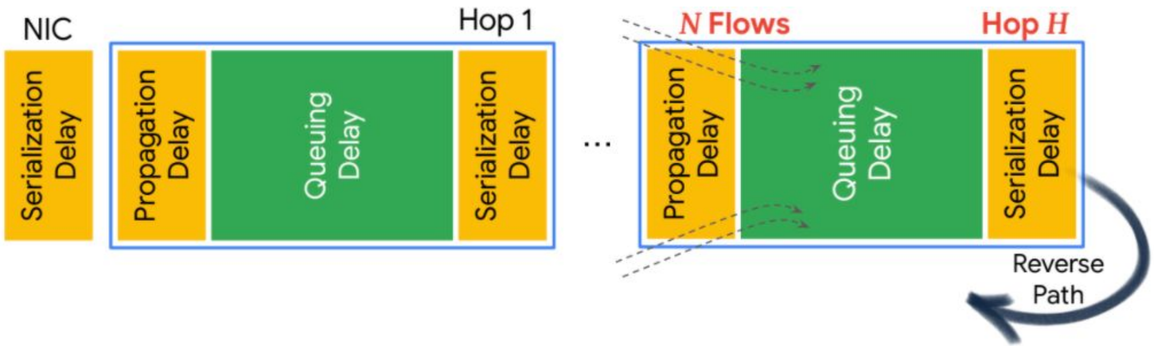
cwnd < 1 implemented via pacing using Timing Wheel, pacing off when cwnd > 1

Swift Design contd.

Scaling of target-delay

Topology-based scaling (TBS) for RTT-fairness

Flow-based scaling (FBS for fairness)



Loss recovery and ACKing policy

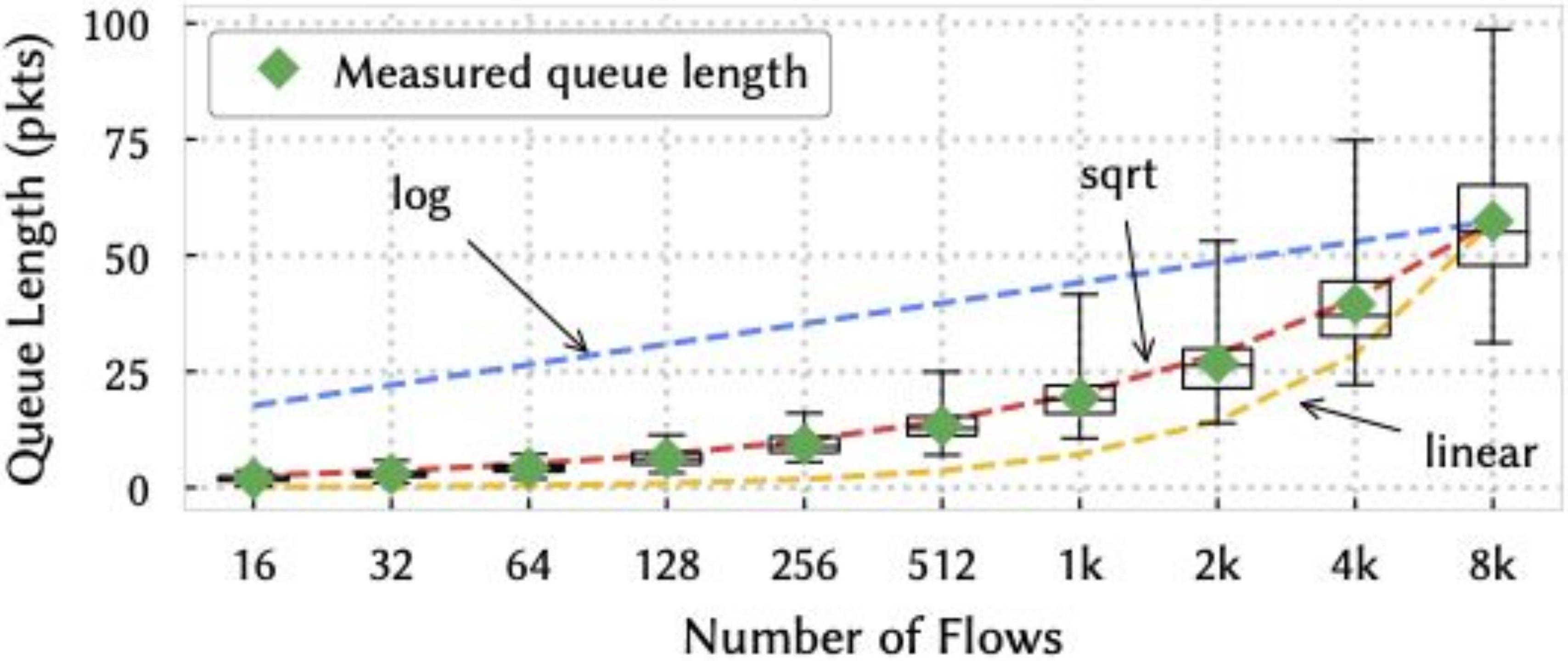
Minimal investment in loss-recovery - losses are rare: SACK and RTO.

Coexistence via QoS

Multiple CC in shared deployments, e.g., WAN traffic, Cloud traffic etc.

Subset of QoS queues reserved for Swift

Average Queue Buildup with Randomized flow arrival and perfect rate control



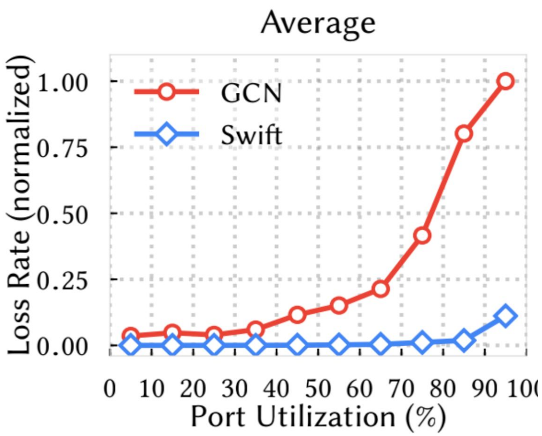
Key Takeaways

From experiences with deployment at Google

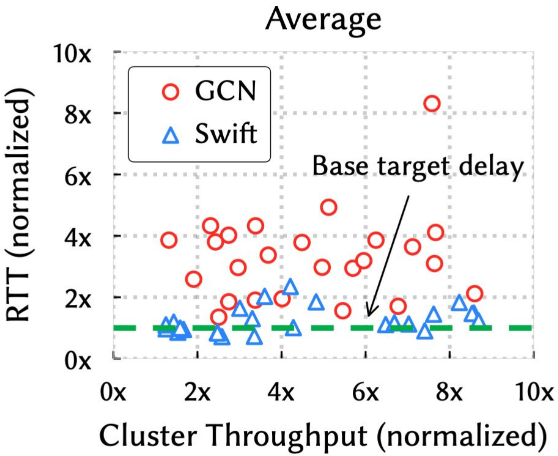
Production Results - Loss and Latency

GCN is a DCTCP[2]-style congestion-control deployed at Google and serves as the comparison point for the production results presented here.

Loss rate vs. Port Utilization at Edge



Latency vs. Cluster Throughput



Takeaways

Swift keeps loss-rates very small even at the 99.9th-p and at near line-rate utilization

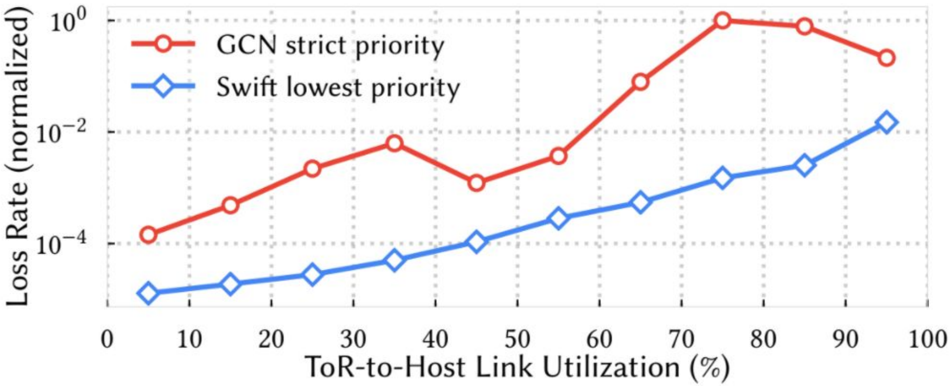
Loss-rate improvement doesn't come at the cost of throughput; Swift sustains same cluster throughput as GCN

We find that Swift is able to maintain the average fabric round-trip around the configured target delay

[2] Datacenter TCP (DCTCP), SIGCOMM 2010

Production Results - Isolation in shared deployments

Isolation via QoS



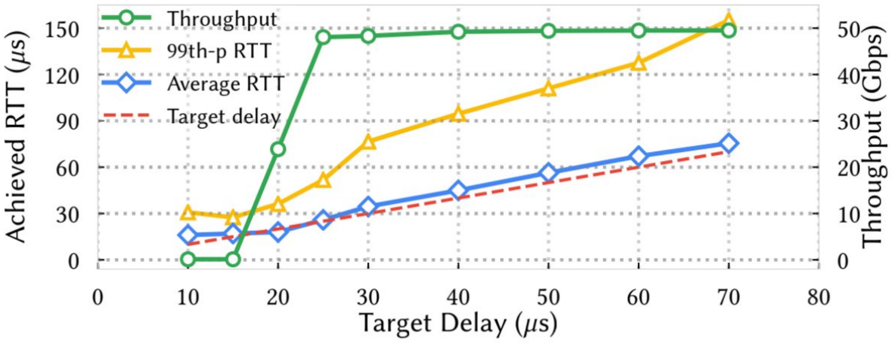
Takeaways

Use of QoS works extremely well in providing isolation from non-Swift traffic in shared infrastructure

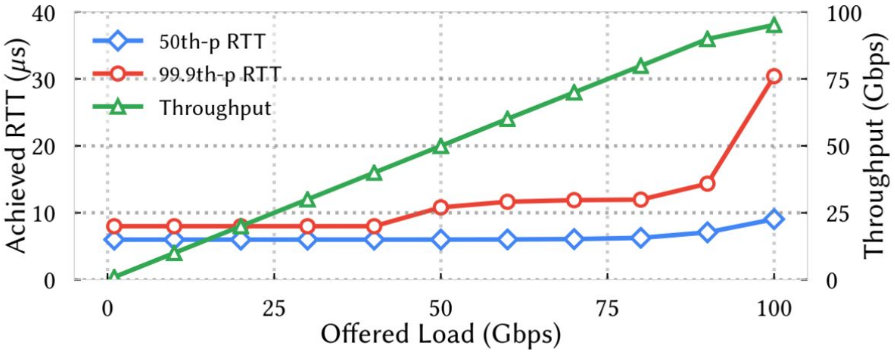
Swift loss rate on the lowest priority QoS is lower than GCN loss rate on strict priority QoS

Swift controls the delay as per the target-delay

Achieved RTT vs. Target Delay (line-rate: 50Gbps)



Load-Latency Curve (line-rate: 100Gbps)



Takeaways

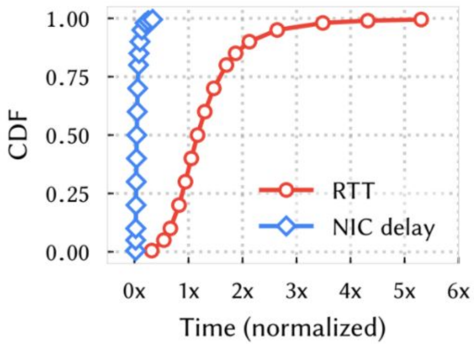
Swift is able to precisely control the achieved average RTT to be around the target

A very small target delay value hurts utilization

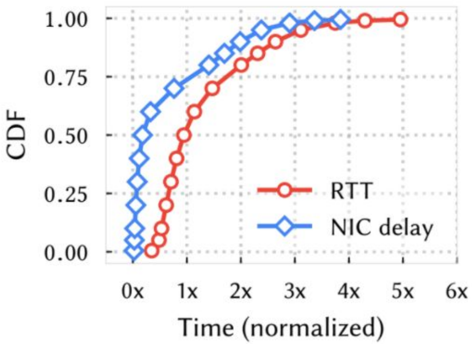
The load-latency curve exhibits the variation in achieved RTT as the per- machine offered load is increased

Production Results - Endpoint congestion

Separation of Fabric vs. Endpoint Congestion



Throughput-intensive cluster



IOPS-intensive cluster

Takeaways

Endpoint congestion (measured by endpoint delays such as in the NIC Queue) is also important to address

NIC delays can account for a significant portion of RTT, especially for IOPS intensive applications

Key conclusions from our experiences with Swift deployment

Delay works really well

Use of delay as a multi-bit congestion signal has proven effective for excellent performance

Use of absolute target delay is performant and robust

And simplicity that has helped greatly with operational issues.

Fabric and Host congestion are both important to respond to

Both forms matter across a range of workloads.

Delay is decomposable to separate concerns

Important for end-to-end performance for applications

Wide range of workloads

Including large scale incast

Pace packets when there are more flows than the bandwidth-delay product (BDP)

Use a congestion window at higher flow rates for CPU efficiency

Future Directions for Research

Optimal CC that works straight out of the box from NICs

What is the optimal increase function for e2e Congestion Control?

Decrease is easier as it’s performed based on an explicit signal such as RTT or ECN.

How can we tell if Congestion Control is work conserving

at Scale?

A systematic way to handling bottlenecks and congestion at hosts

Congestion Control that can run in Hypervisors w/o direct access to Guest transports

Achieving ultra low latencies (<10us) for short transfers that's close to propagation delay in the presence of bandwidth intensive transfers

Is Congestion Control at the packet layer fundamentally better than one at higher level entities such as messages (RMAs, RPCs)?

A robust well-performing and simple congestion control for the WAN that's tolerant of noisy signals and works for small or large buffers

Questions and Discussion

nanditad@google.com