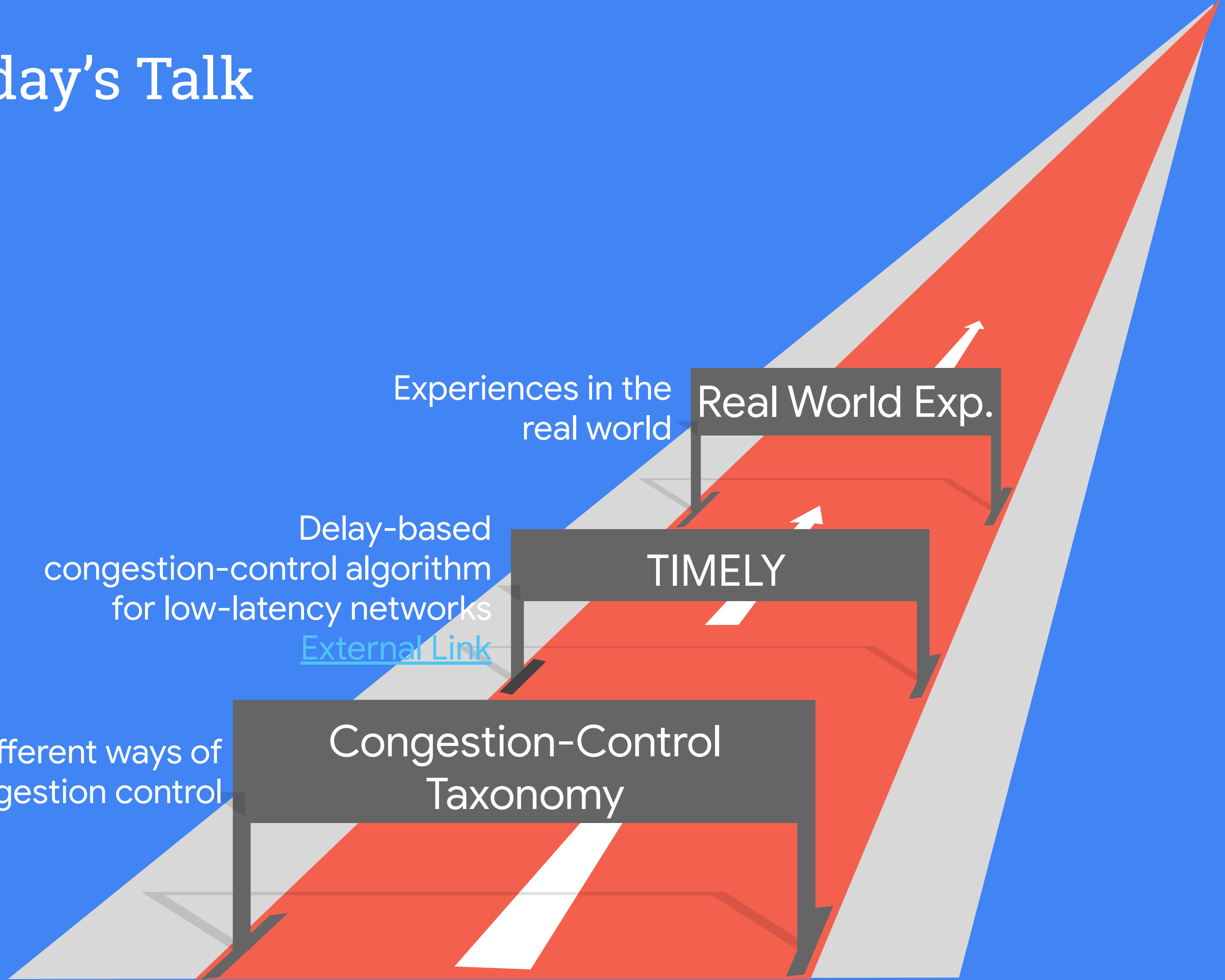
Congestion Control in the *Real* World

Nandita Dukkipati
nanditad@google.com

CS144 lecture
25 October, 2019

Today's Talk



Experiences in the real world

Delay-based congestion-control algorithm for low-latency networks
[External Link]

Many different ways of congestion control

Many different ways to signal Congestion

Google

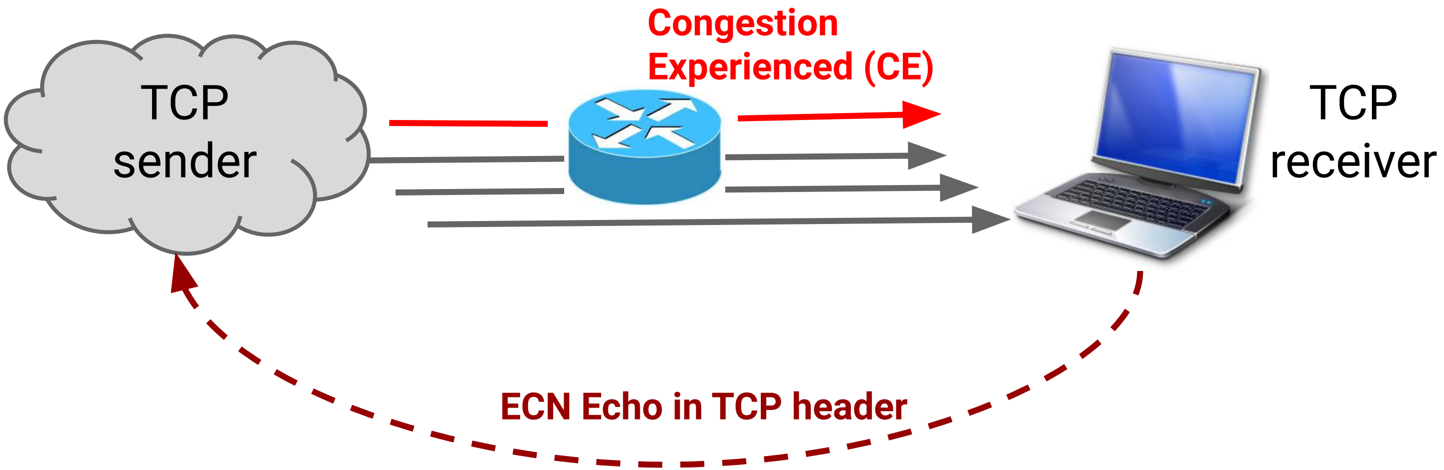
Congestion Control Taxonomy

Loss	New Reno, Cubic TCP
Delay	TCP Vegas, TIMELY, BBR
Explicit Feedback	DCTCP, XCP, RCP

[1] TIMELY: RTT-based Congestion Control for the Datacenter, SIGCOMM 2015
[2] DCTCP: Data Center TCP (DCTCP), SIGCOMM 2010
[3] BBR: Congestion-based Congestion Control, ACM Queue, 2016
[4] TCP Vegas: new techniques for congestion detection and avoidance, SIGCOMM 1994

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)

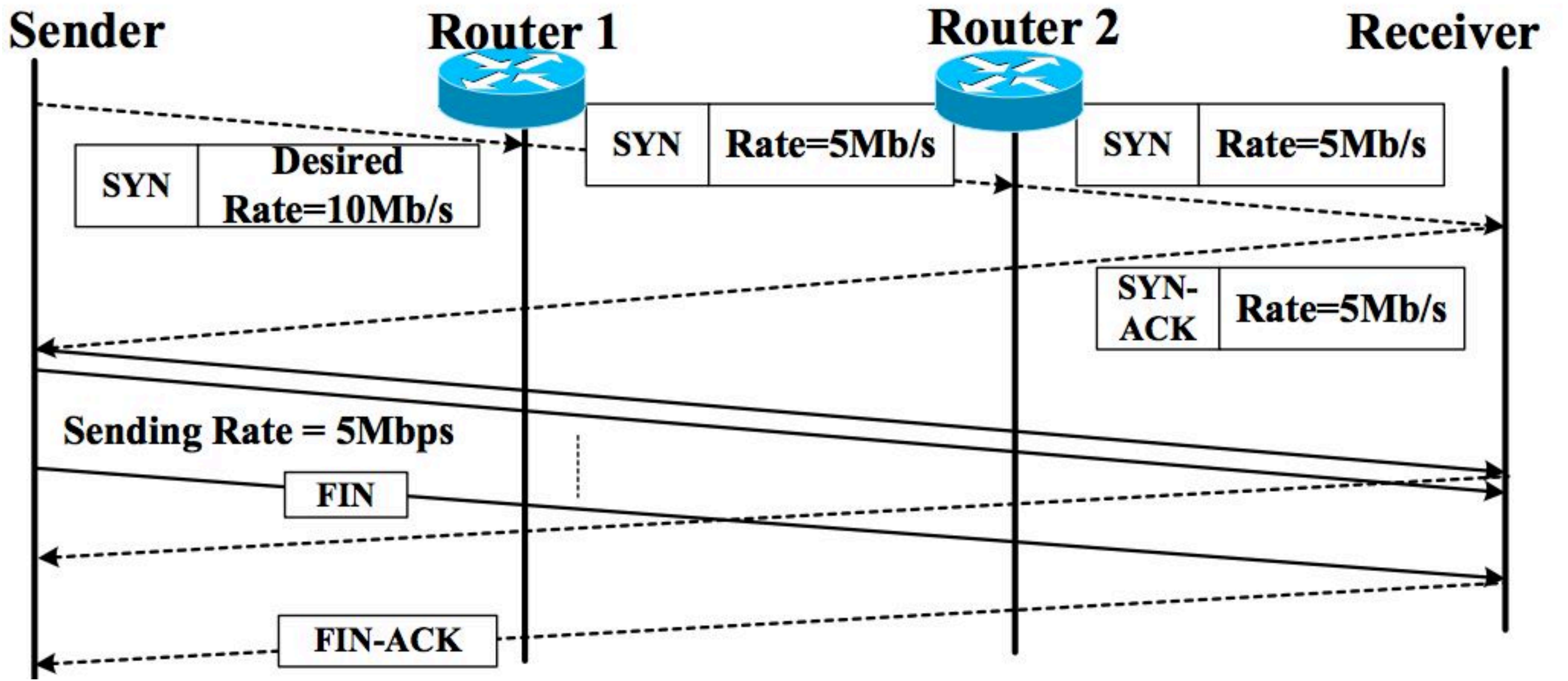
- Datacenter DCTCP (SIGCOMM 2010) uses ECN marks.
- Switches mark CE bit in IP header if queue > 65KB.
- Receivers reflect marks to senders (via TCP flags).
- Sender slows down according to proportion of marked packets each RTT.

Rate Control Protocol (RCP)

RCP motivation

- Uses flow completion time as the primary metric for congestion control.
- Should work well for short flows that don't have an opportunity for continual incremental rate updates.
- No complex per-packet computations in switches.

RCP uses min. supported rate along the path



RCP Algorithm

Link Capacity

Aggregate Traffic

queue

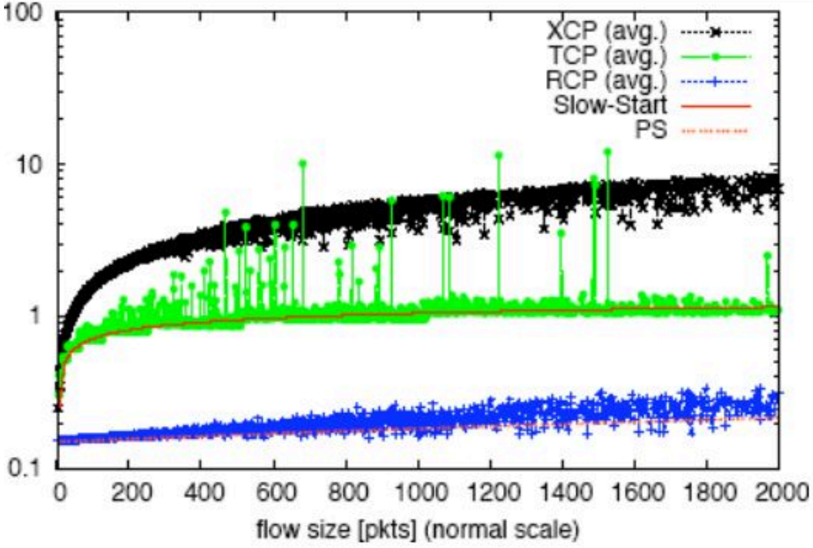
Average RTT

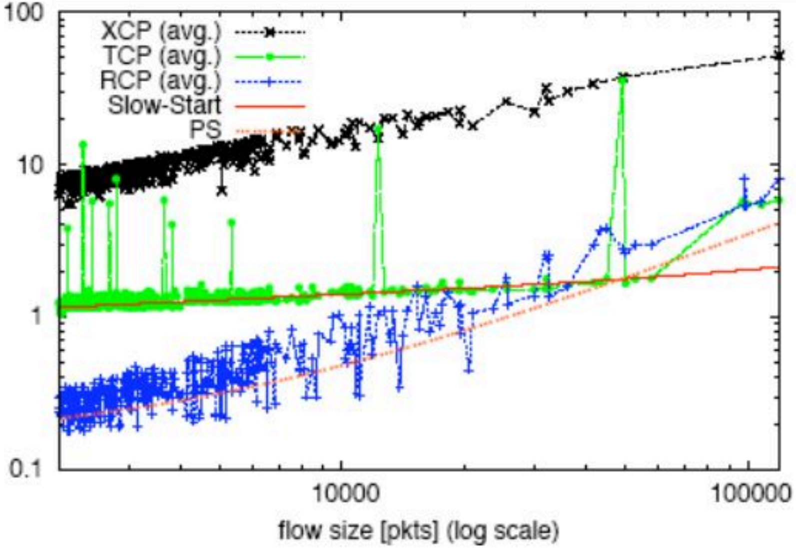
$$R(t) = R(t - d_0) + \frac{\alpha(C - y(t)) - \beta \frac{q(t)}{d_0}}{\widehat{N(t)}}$$

Estimate of # flows

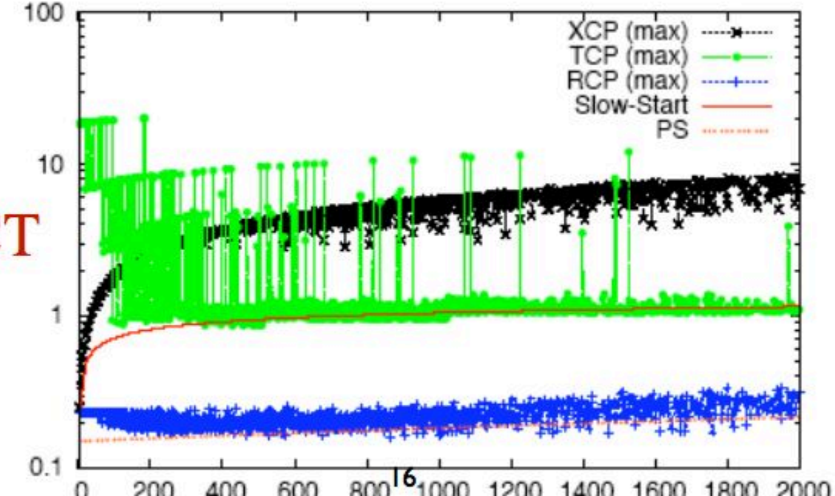
$$\widehat{N(t)} = \frac{C}{R(t - d_0)}$$

$$R(t) = R(t - T)[1 + \frac{\frac{T}{d_0}(\alpha(\gamma C - y(t)) - \beta \frac{(q(t) - q_0)}{d_0})}{\gamma C}]$$


XCP (avg.)
TCP (avg.)
RCP (avg.)
Slow-Start
PS
100
10
1
0.1
0 200 400 600 800 1000 1200 1400 1600 1800 2000
flow size [pkts] (normal scale)


XCP (avg.)
TCP (avg.)
RCP (avg.)
Slow-Start
PS
100
10
1
0.1
10000
100000
flow size [pkts] (log scale)

Max. FCT


XCP (max)
TCP (max)
RCP (max)
Slow-Start
PS
100
10
1
0.1
0 200 400 600 800 1000 1200 1400 1600 1800 2000

Explicit feedback is a game changer for CC

E2E congestion control has a hard time with:

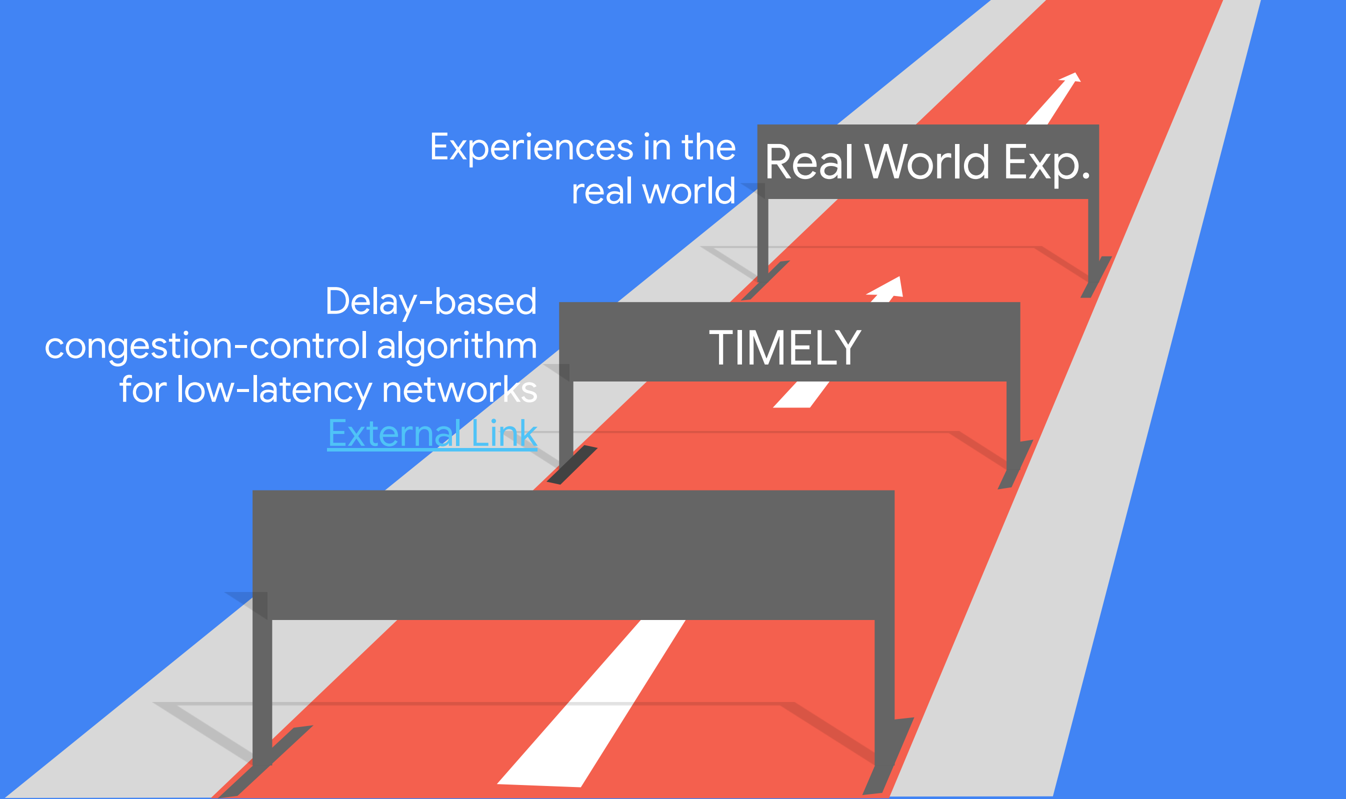
- Discerning between incidental losses/delay or persistent congestion.
- Disambiguating between forward and reverse path congestion.
- Figuring out which hop along the network path is contributing to delay.
- Applications starting up large number of connections, e.g., O(10K).
- Reconciling between LAN/WAN congestion.
- <and several others here>.

TIMELY: RTT-based Congestion Control for the Datacenter

Experiences in the real world

Delay-based congestion-control algorithm for low-latency networks

External Link



Datacenter Congestion Control
High Level Goals

Low-Latency for RPCs
Provides low RPC completion times for short and long RPCs.

End-to-end network performance isolation
Performance isolation at every congested point along the path.

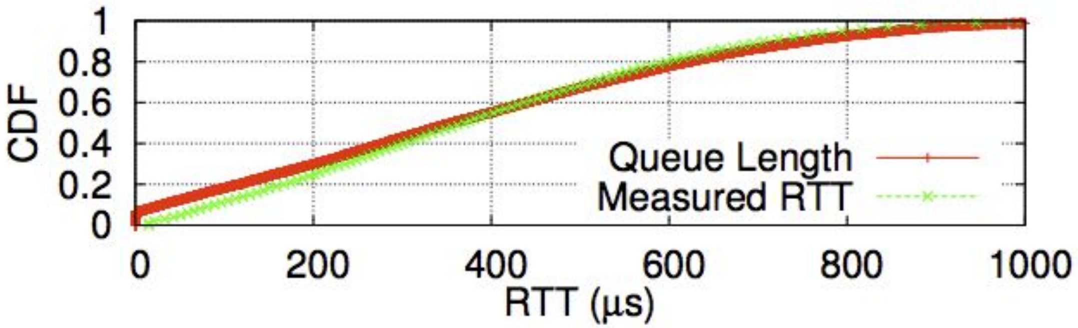
Bandwidth Efficiency
Provides high-throughput even under heavy-incasts.

CPU Efficiency
Uses constructs that use less CPU (e.g., doesn’t require too many locks).

Google

Delay as a Congestion Signal

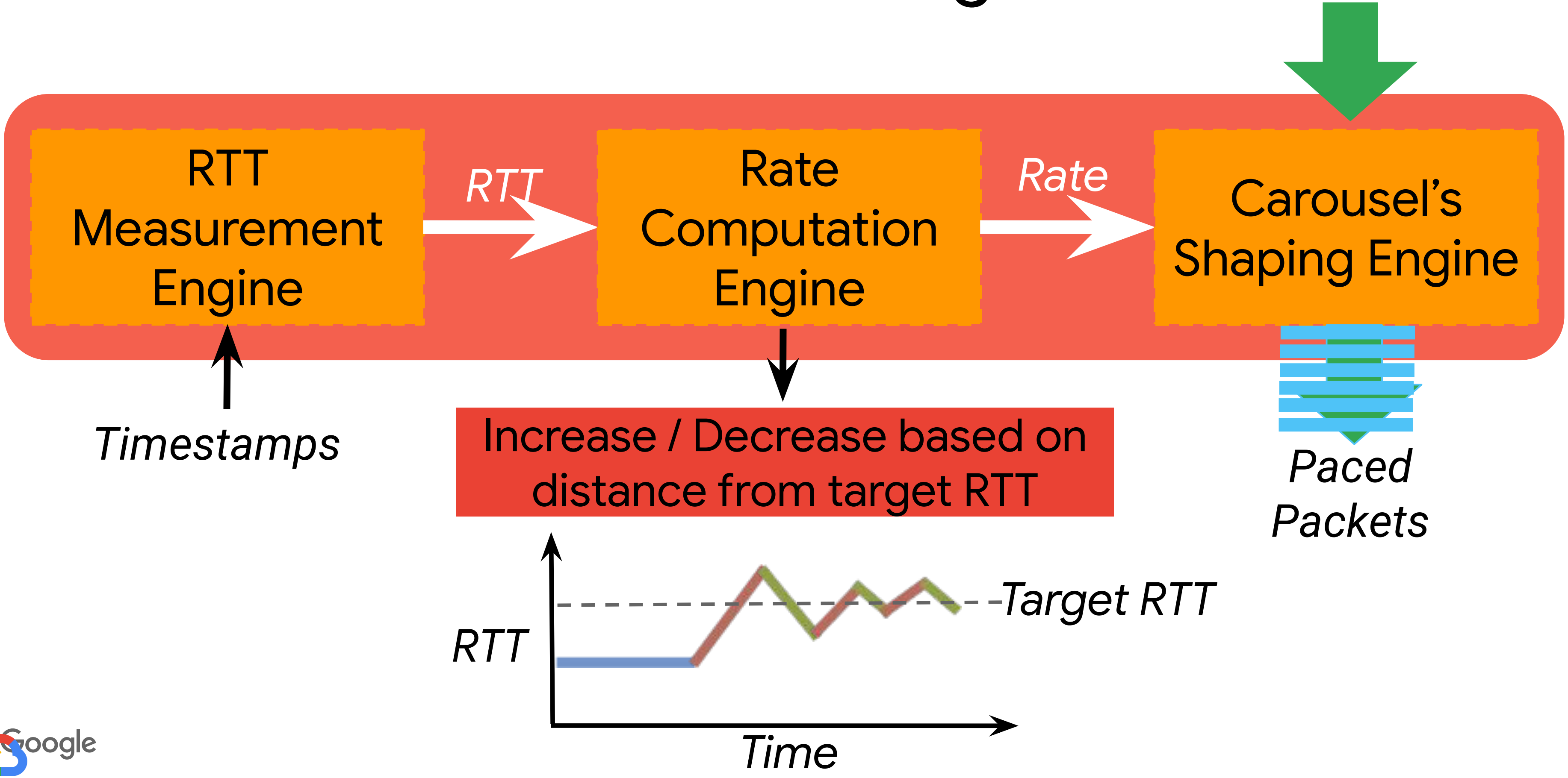
- Can be measured accurately using NIC hardware timestamps.
- Is a rapid, multibit signal that correlates extremely well with queuing.



- Is an end-to-end congestion signal and directly ties with latency-guarantees, congestion control is trying to provide.
- Is decomposable - can provide isolation at different congestion-points.

Google

TIMELY Framework and Algorithm



Google

Predictable Low-Latency

TIMELY adjusts sending rate based on measured delay

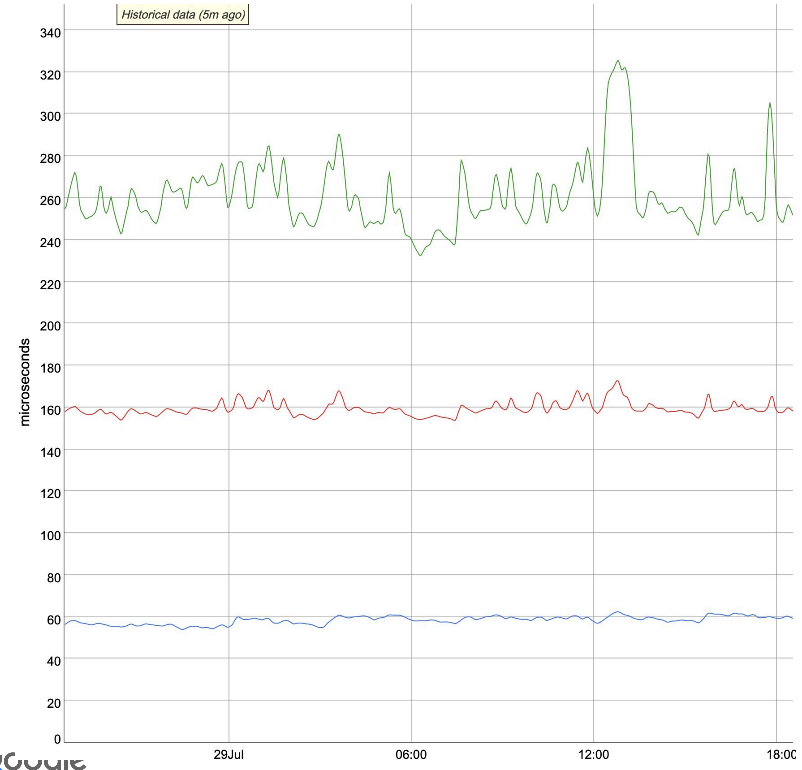
- Sender tracks a congestion-window (cwnd) to determine the rate to send at.
- If measured delay < target, increase the cwnd, otherwise decrease the cwnd.
- Additive-increase Multiplicative-decrease (AIMD) to ensure fairness b/w senders.

Target-delay allows fine-grain control over latency

- Experimental and production data shows that RTTs hover around the specified target.

Takeaway 1: TIMELY controls delay to be around the specified target

RTTs in a large setting



Median RTT ~59us

99th-p RTT ~ 158us

99.9th-p RTT ~ 251us

Even tail RTTs are much lower compared to traditional congestion-control algorithms

High Fan-In Traffic Patterns

TIMELY's cwnd can fall below 1 when a large number of flows send to the same host

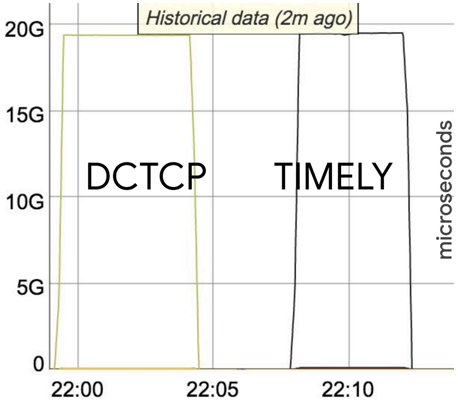
- This enables TIMELY to keep its promise of low-latency and zero loss, while still providing maximum throughput

cwnd < 1 is enabled via pacing packets using a Timing Wheel

- TIMELY moves seamlessly between a rate-based and window-based protocol

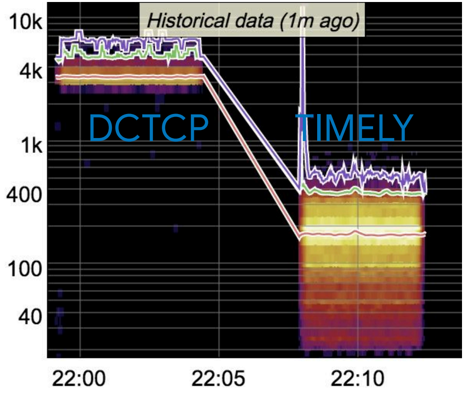
Takeaway 2: TIMELY supports extremely small cwnds to handle bursts efficiently, maintaining high-throughput, low-latency, and almost zero loss

Large Incast - 1000 flows sending to 1



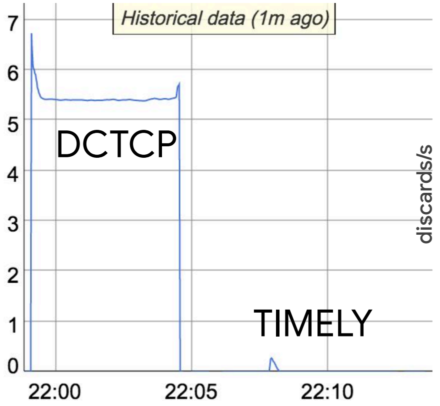
Throughput

TIMELY is able to provide line-rate throughput (NIC speed = 20G).



RTT

TIMELY’s median RTT is 172us whereas DCTCP”s median RTT is 3310us.
At 99th-p, TIMELY’s RTT is 370us while DCTCP’s is 4900us.



Retransmissions

DCTCP has 5.5% retransmissions whereas TIMELY has ~0%.

Google

Real World Experience

Google

Special Switch Queues for TIMELY

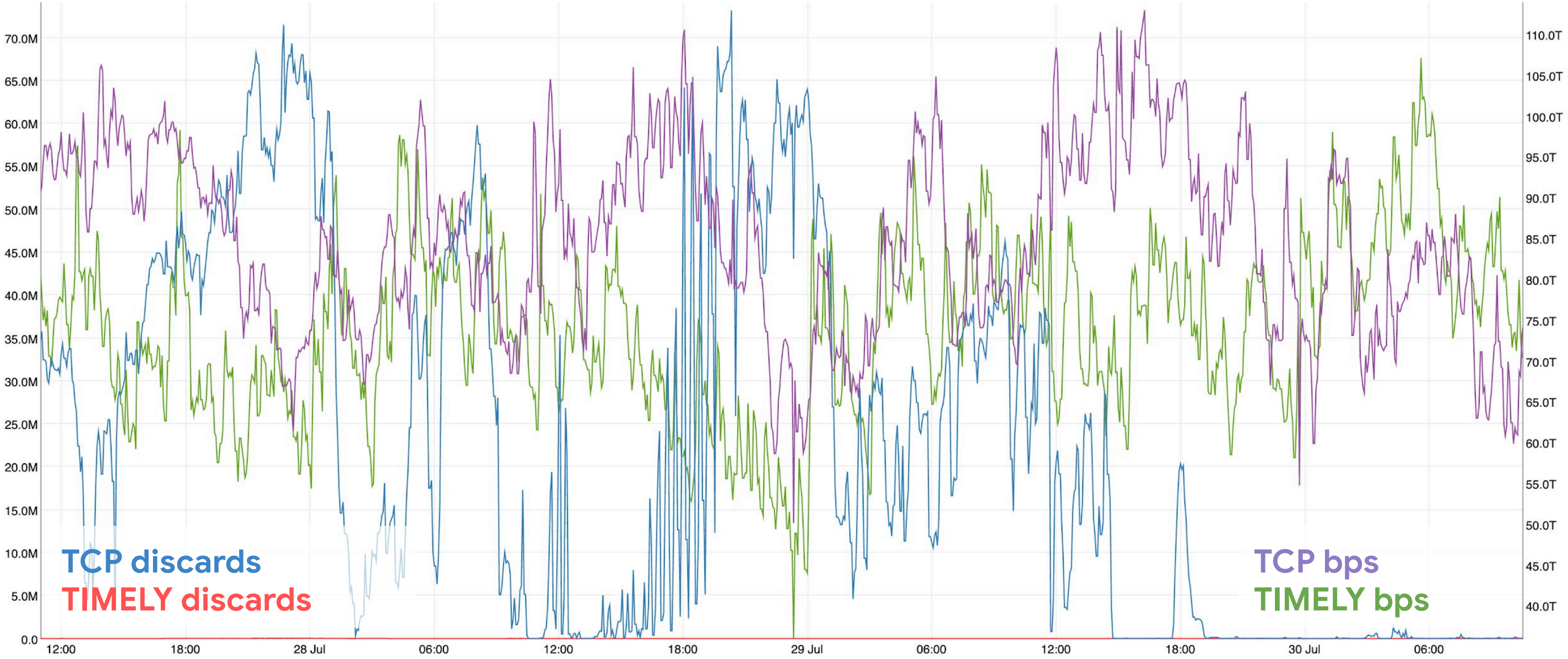
TIMELY is non-TCP-compatible and requires its own network-queue

Separate queues provide isolation vs TCP traffic

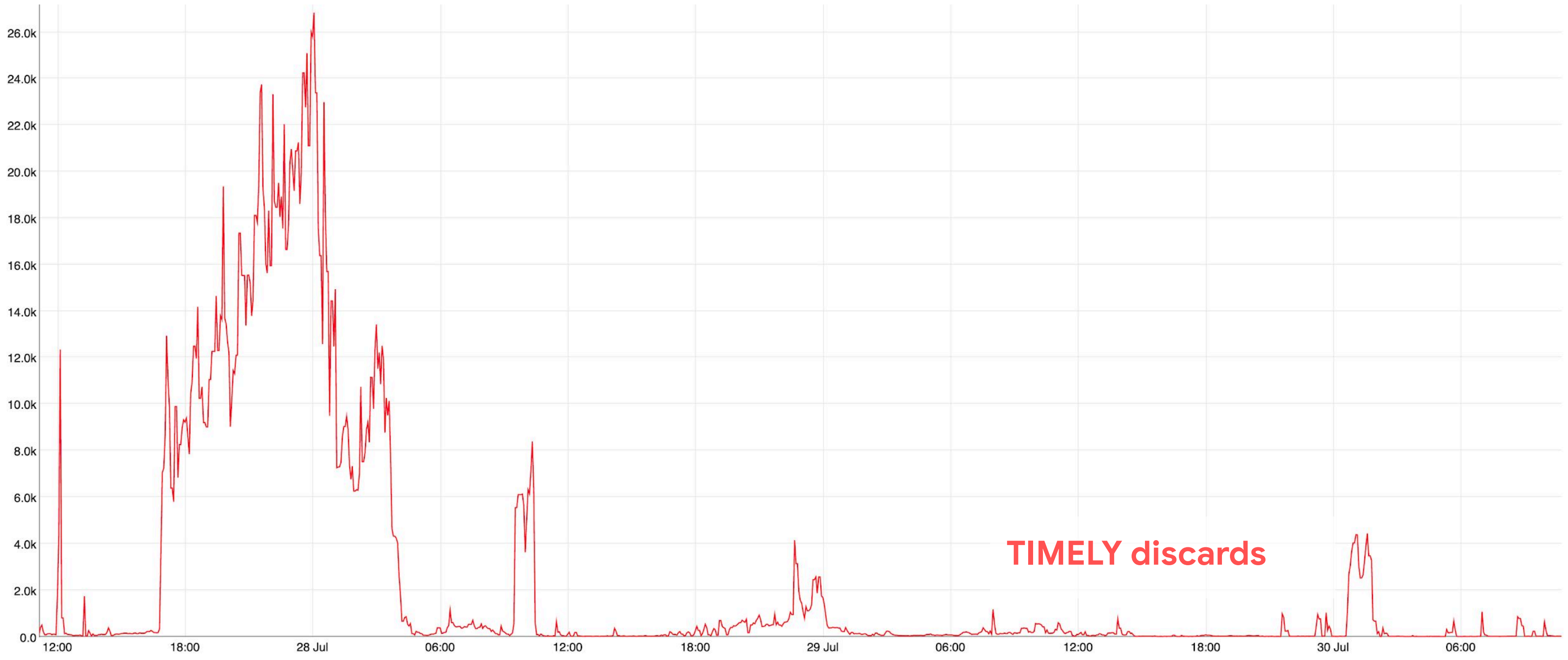
- If TIMELY shared queues, latency, loss, throughput will be affected by TCP, since TCP typically requires more buffers.

Takeaway 3: TIMELY runs on it's own network queues which enables it's promise of low-latency and almost-zero loss.

Substantially lower discards at similar throughputs



XXXbps on TIMELY and almost 0 loss



CPU constrained Networks

Congestion is not limited to the fabric, can also occur at endpoints.

Networking doesn't have infinite CPU: OS-bypass/User-Space packet-processing gives a CPU-efficient alternative, but packet-processing engines can get overloaded and queues can build up.

Endpoint congestion behaves differently than Fabric congestion

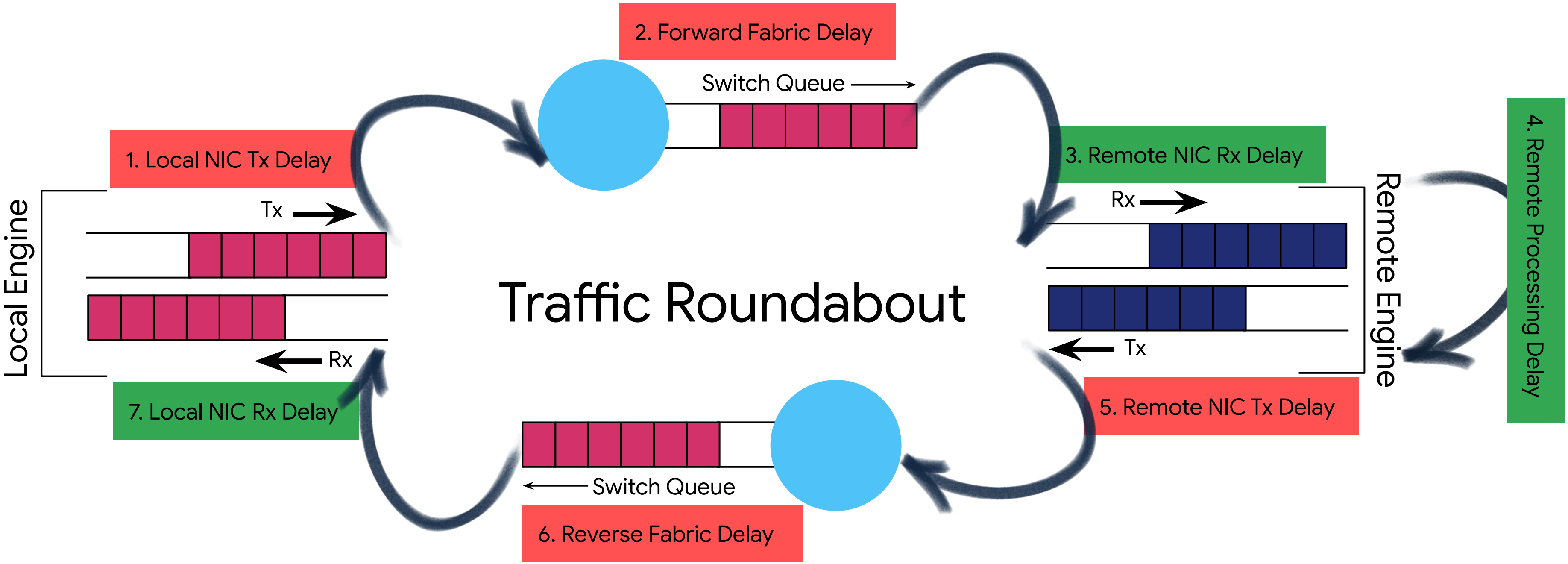
Think more volatile, transient queue build-ups - not just depend on incoming pps but also host-specific attributes, e.g., cache locality.

Takeaway 4: Two congestion-windows instead of one

One to track available capacity in the fabric, and one to track available capacity at the endpoints.

$cwnd = min\ (fcwnd, ecwnd)$

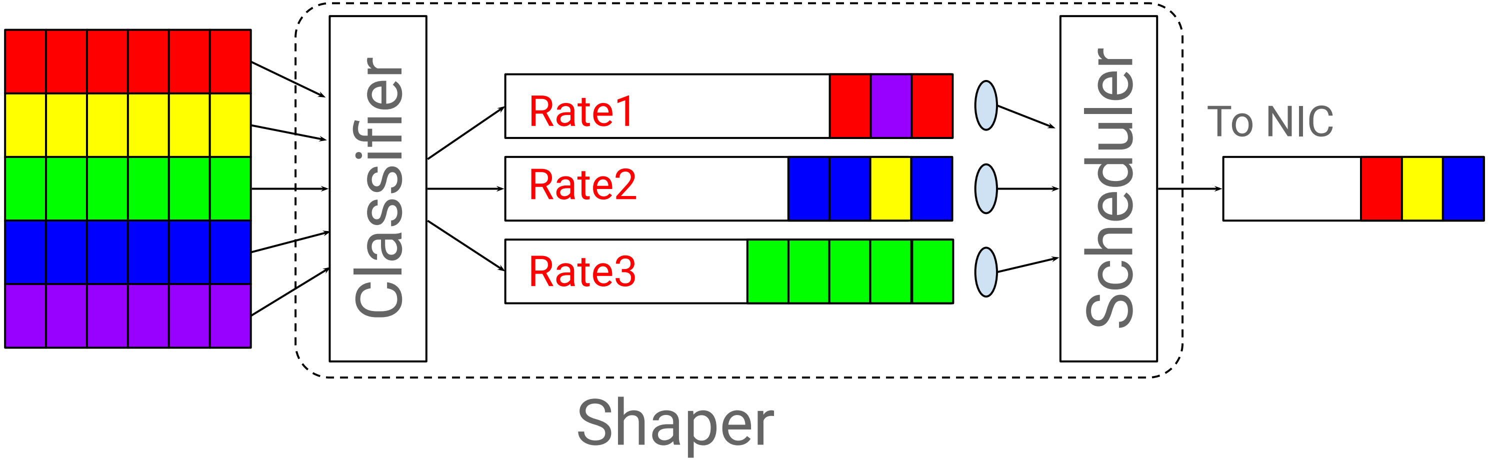
Delay sources in SNAP[1]



1. High-performance Host-Networking stack supporting RDMA (**Snap: a Microkernel Approach to Host Networking**)

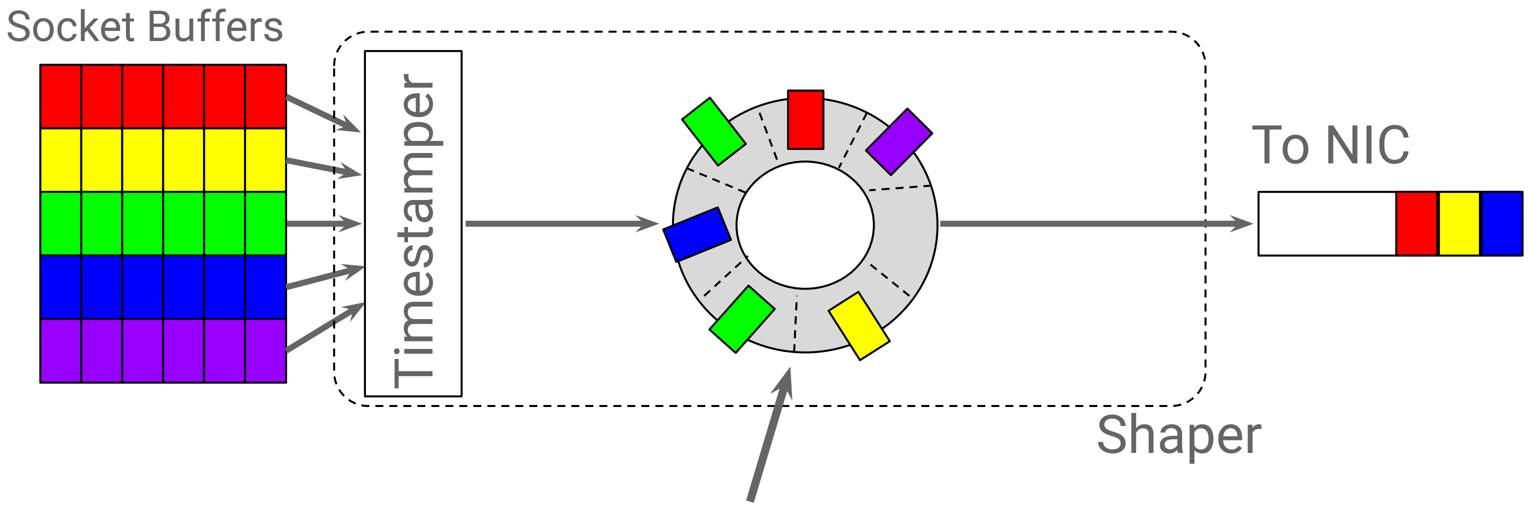
Traffic Shapers in Practice

Packet Sources: Socket Buffers
in Host OS or Guest VM

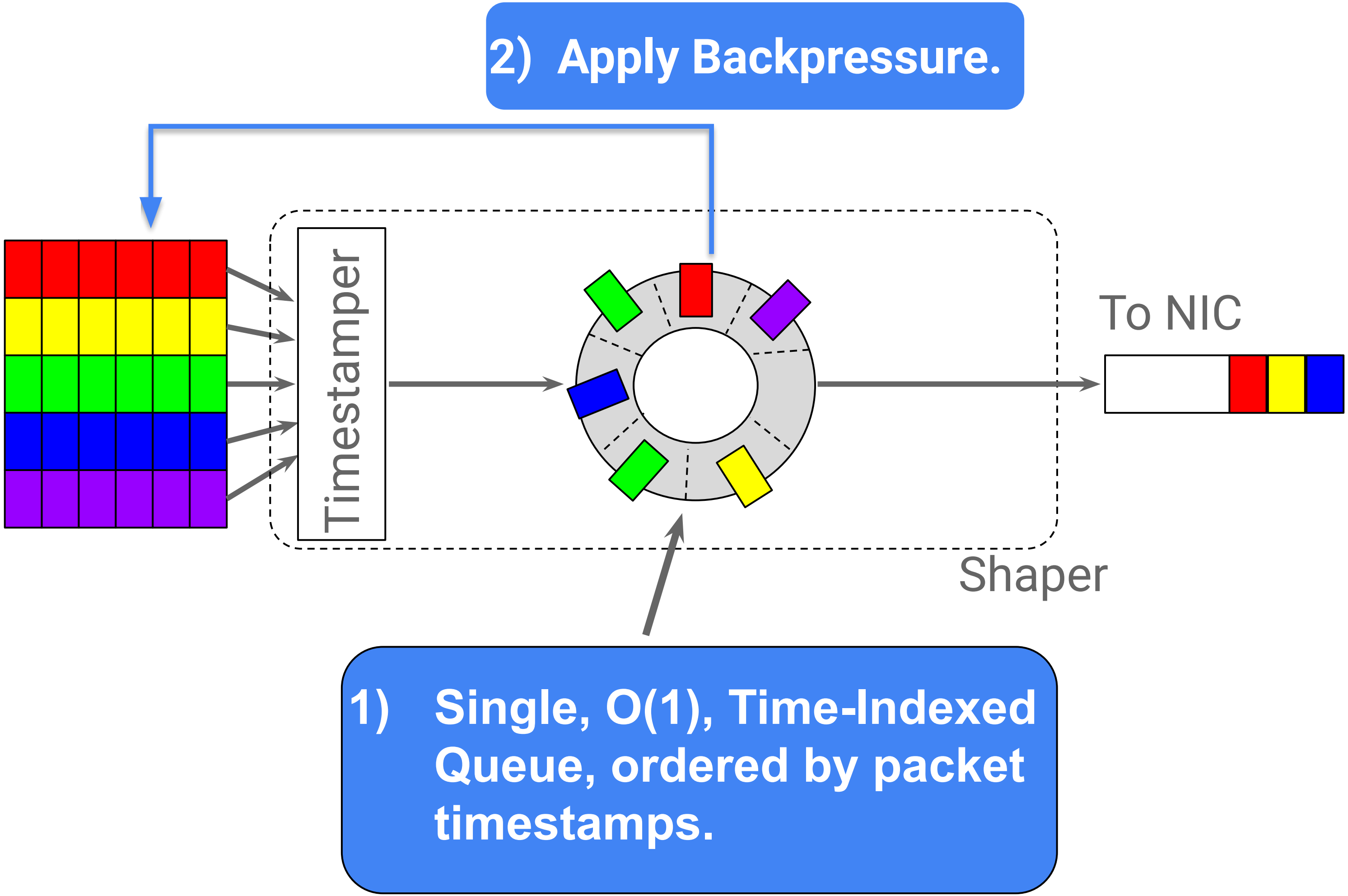


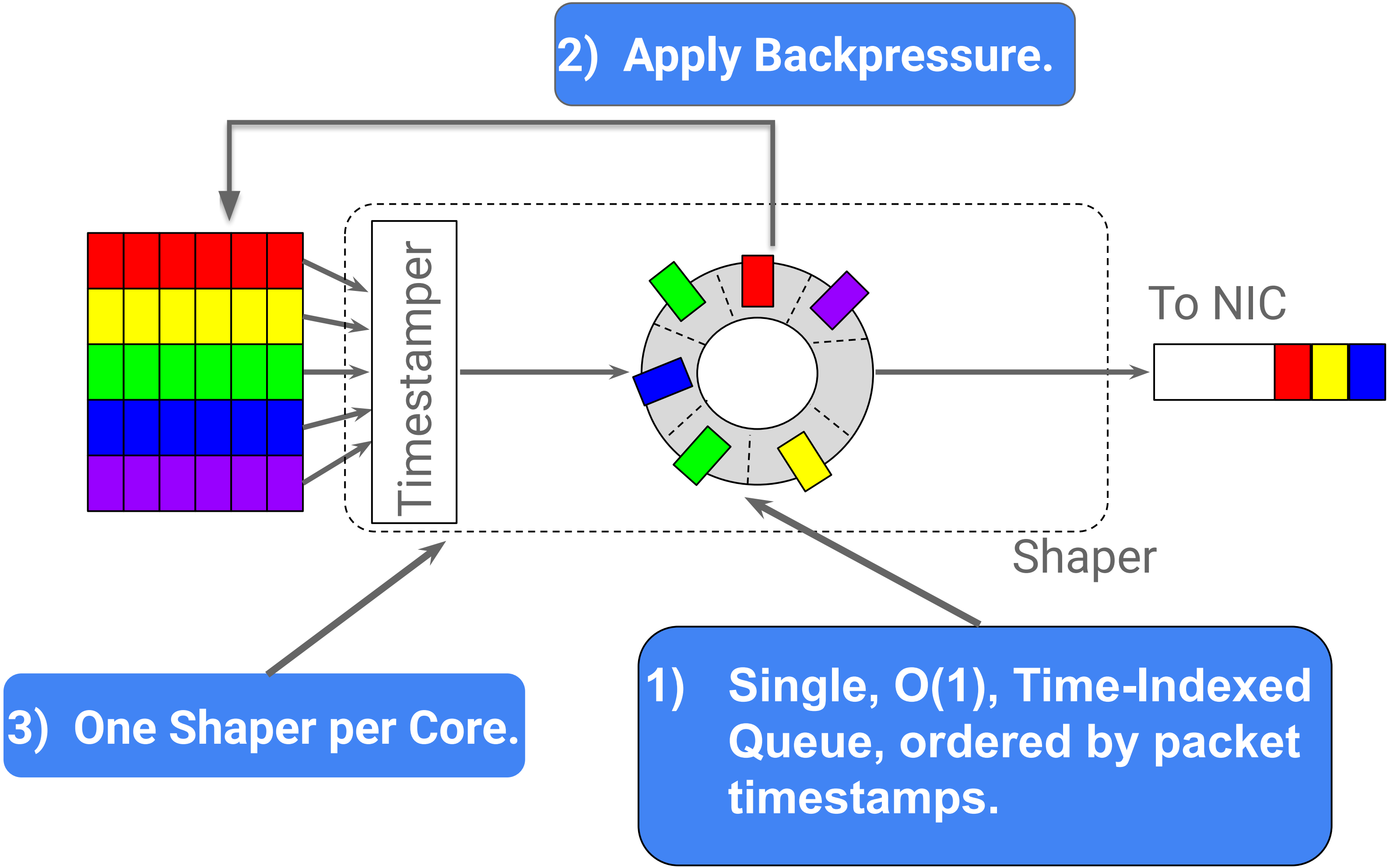
Pre-filtering with Multiple Token Bucket Queues.

Design Principles of Carousel

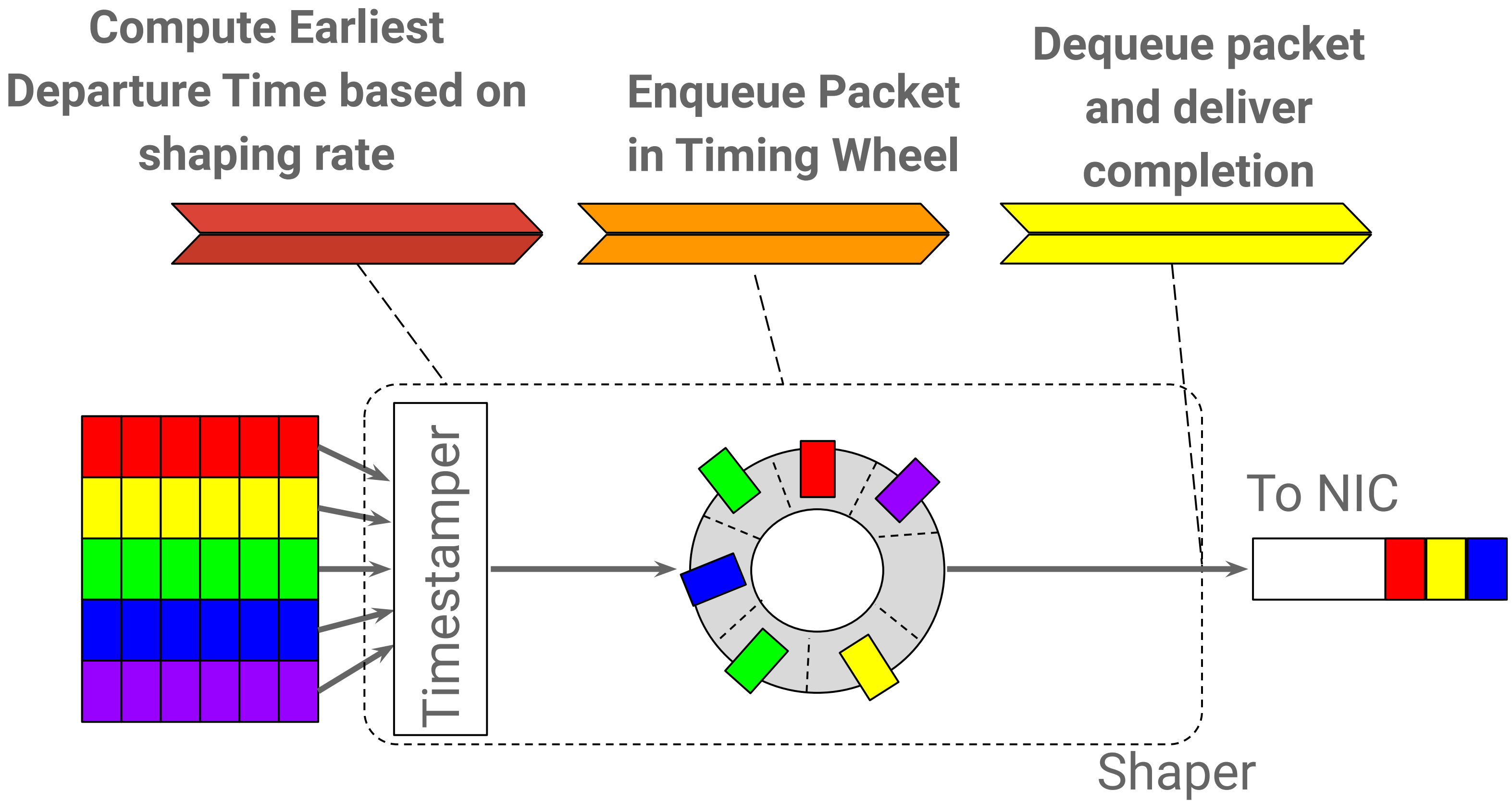


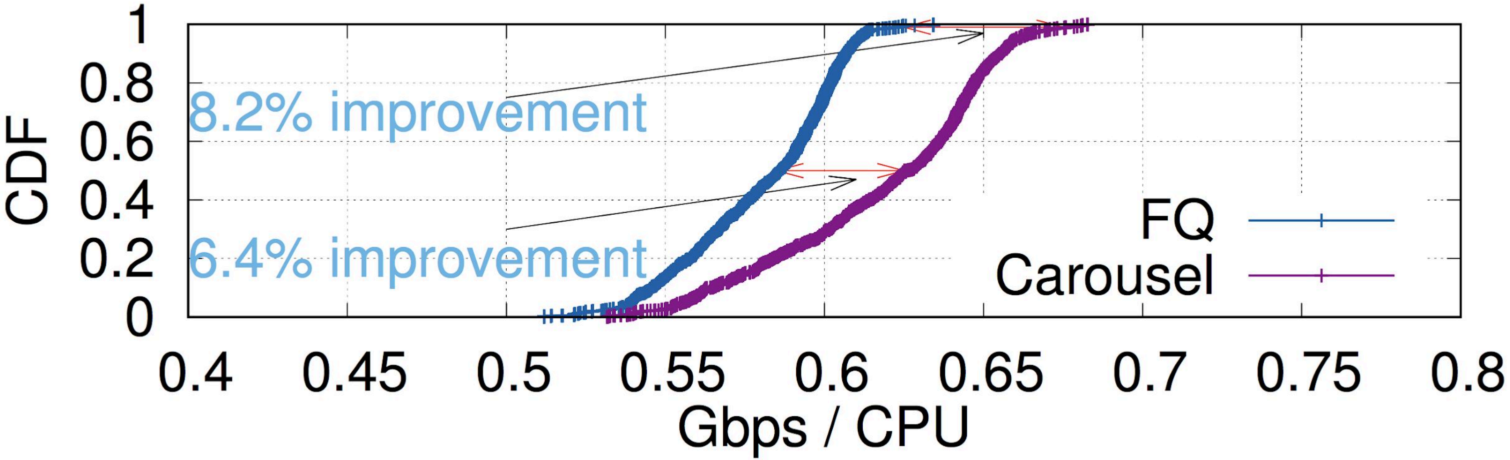
1) **Single, O(1), Time-Indexed Queue, ordered by packet timestamps.**


2) Apply Backpressure.
Timestamper
To NIC
Shaper
1) Single, O(1), Time-Indexed Queue, ordered by packet timestamps.


2) Apply Backpressure.
Timestamper
To NIC
Shaper
3) One Shaper per Core.
1) Single, O(1), Time-Indexed Queue, ordered by packet timestamps.

Life of a Packet in Carousel





Carousel saves up to 8.2% of overall CPU utilization.
(5.9 cores on a 72 core machine)

Concluding Remarks

Congestion control is a systems problem with broad applications:

- Resource allocation.
- Efficiency.
- Isolation.
- Directly impacts the performance of applications.

Acknowledgments

Thanks to several colleagues at Google for collaboration on the work and slides content, in particular Gautam Kumar.