Host Networking (Part-II)

Nandita Dukkipati
Lecture at UC Berkeley, April 2024

Lecture Topics

(Last Lecture) Host Networking - Part I

Why Host Networking Matters.

The Role of Network Interface Cards (NICs).

Interfacing with Applications using Remote Direct Memory Access (RDMA).

(This Lecture) Host Networking - Part II
Techniques to reduce latency for applications

Congestion Control.

Load Balancing.

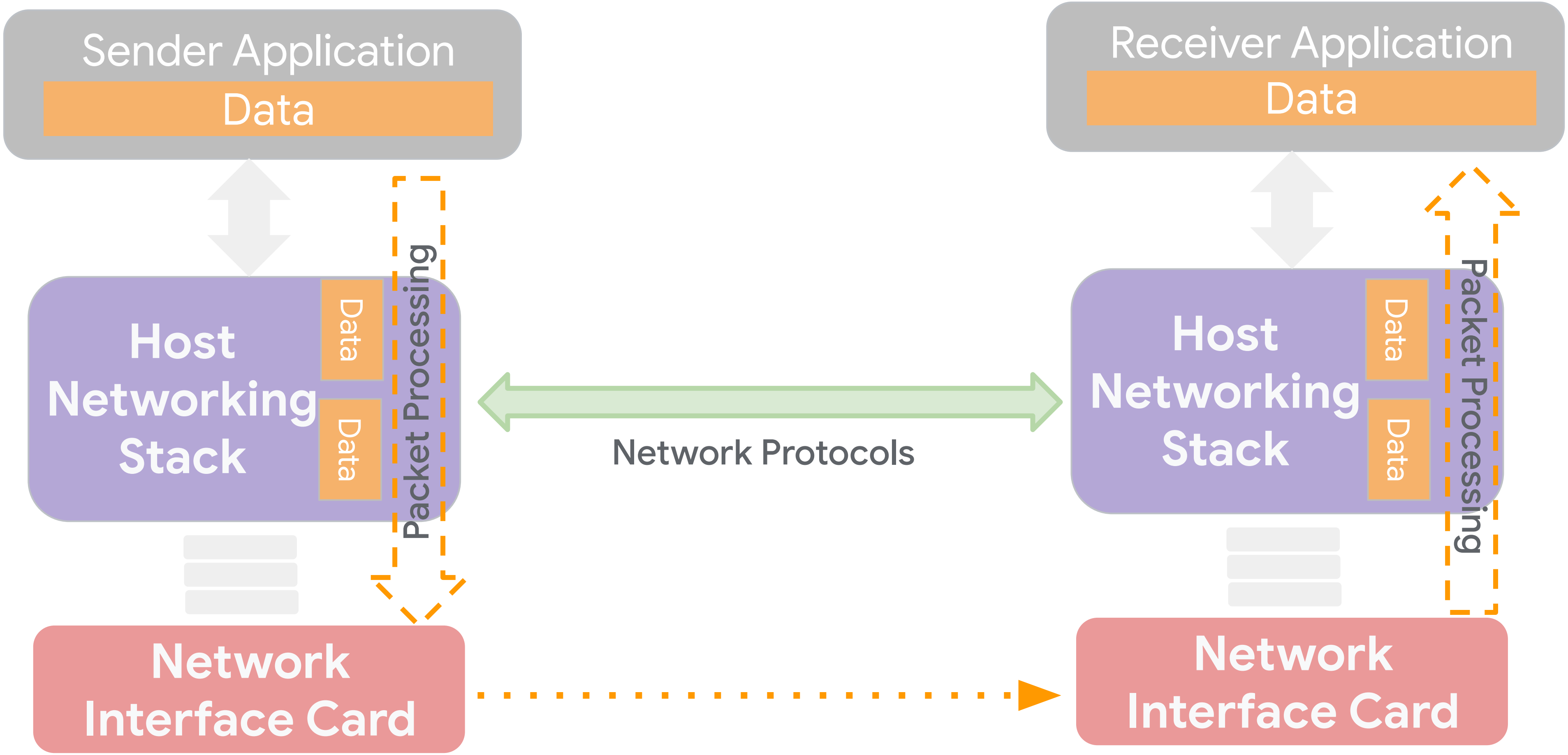
Shaping and Pacing traffic.

Quality-of-Service.

[If time permits] Reliable delivery of packets.

Recap: Protocols running in Hosts

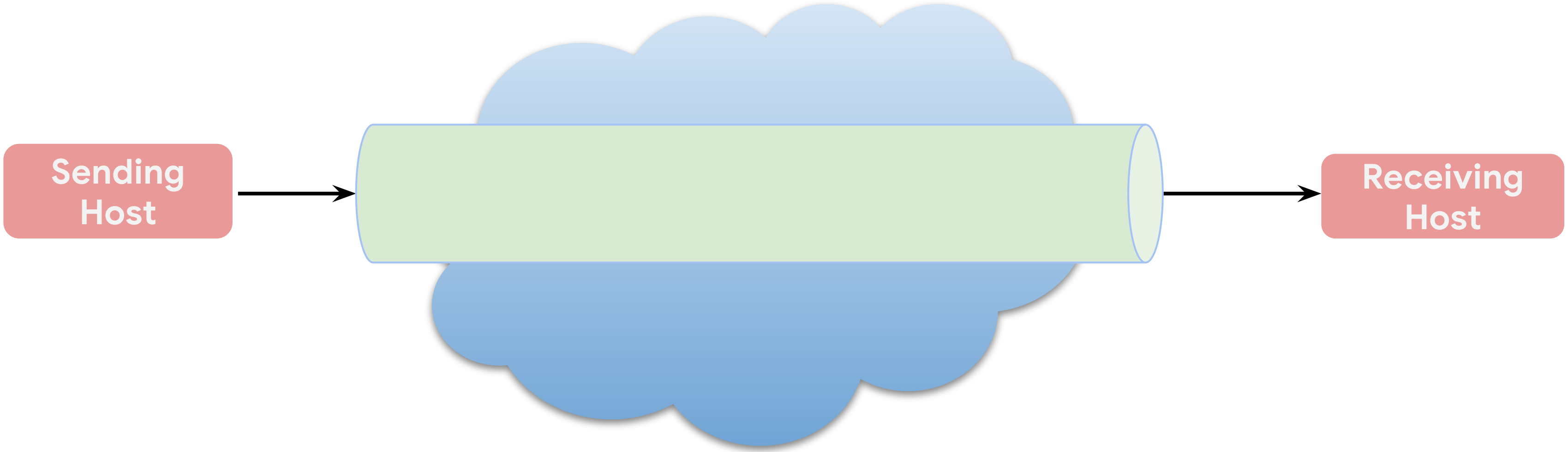
What constitutes Host Networking?



Protocols running in Hosts: the abstraction

Transport protocols, such as TCP, offer the abstraction of a fast, reliable, secure ordered byte stream.

Implemented on an insecure, unordered, lossy datagram network with varying speed and reliability.



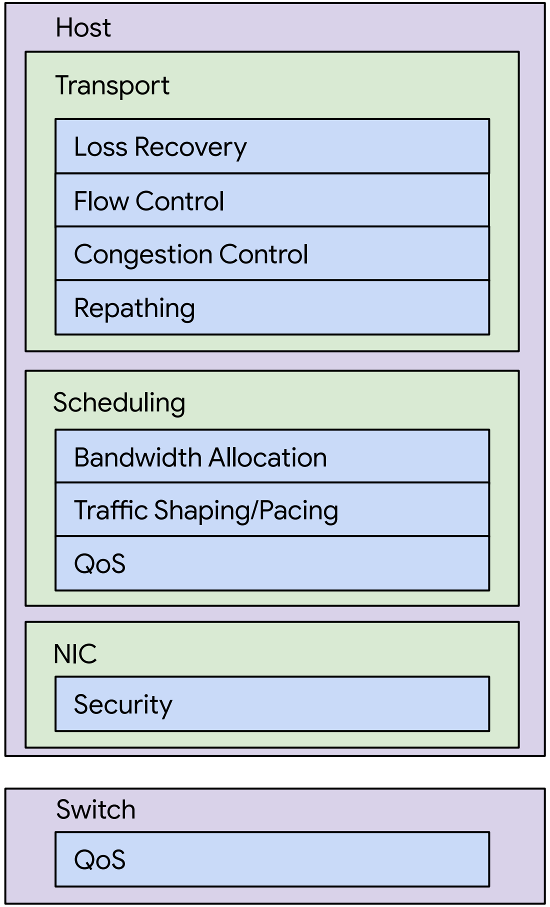
Protocols running in Hosts: Implementing the abstraction

Green: covered before
Purple: new topics
Orange: not covered

- **Congestion Control**: how fast to send, to avoid overloading the network?
  - Delay-based congestion control.
- **Loss Recovery**: what to send: how to infer which packets were lost, for retransmissions for reliability?
- **Flow Control**: how fast to send, to avoid overloading the receiver's memory and/or CPU?
- **Load Balancing**: which path/paths should the traffic travel to avoid congestion and black holes?

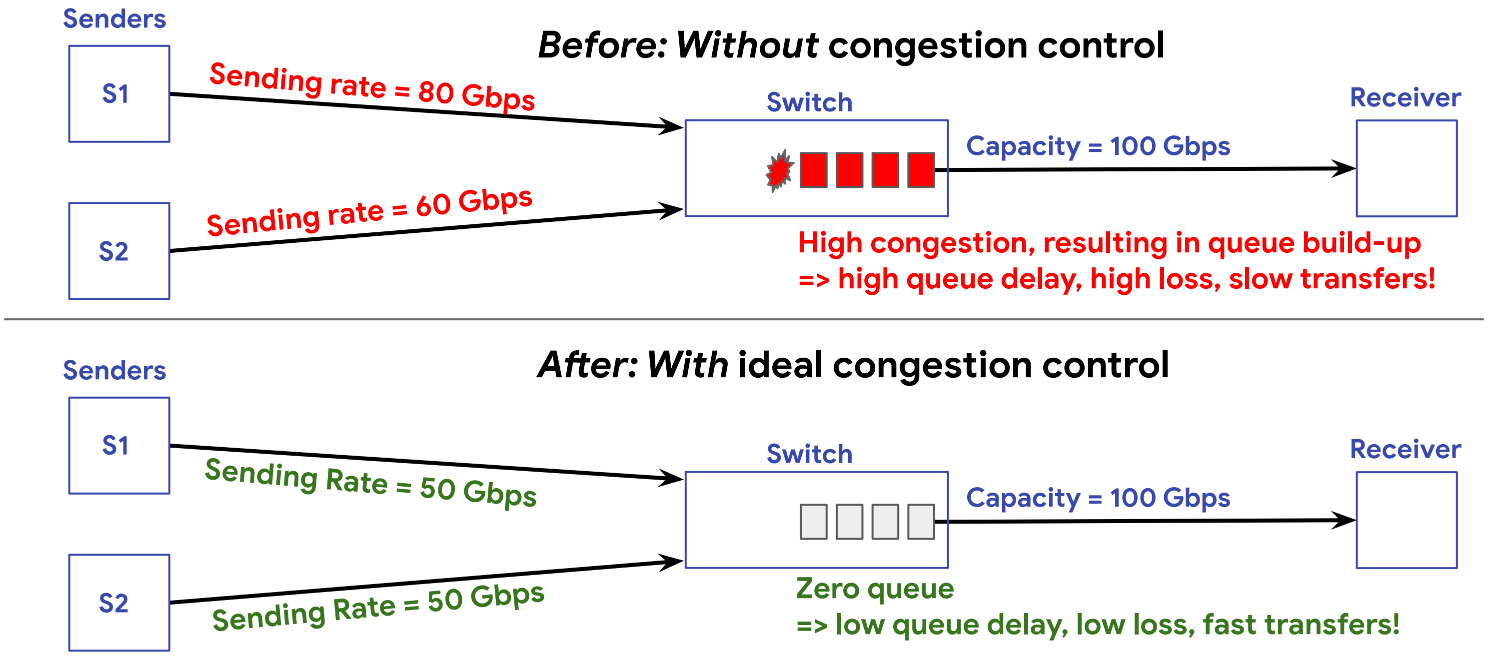
- **Traffic Shaping/Pacing**: when should each packet be sent, to maintain short network queues?
- **Quality of Service (QoS)**: what's the priority of this traffic for allocating buffers and bandwidth at each hop, at <RTT time scales?
- **Bandwidth Allocation**: how much bandwidth is this user allowed on this path, on long time scales?

- **Security**: how to ensure traffic has a known src/dst user (authentication) and content is correct (integrity / checksums) and secret (encryption)?



Congestion Control

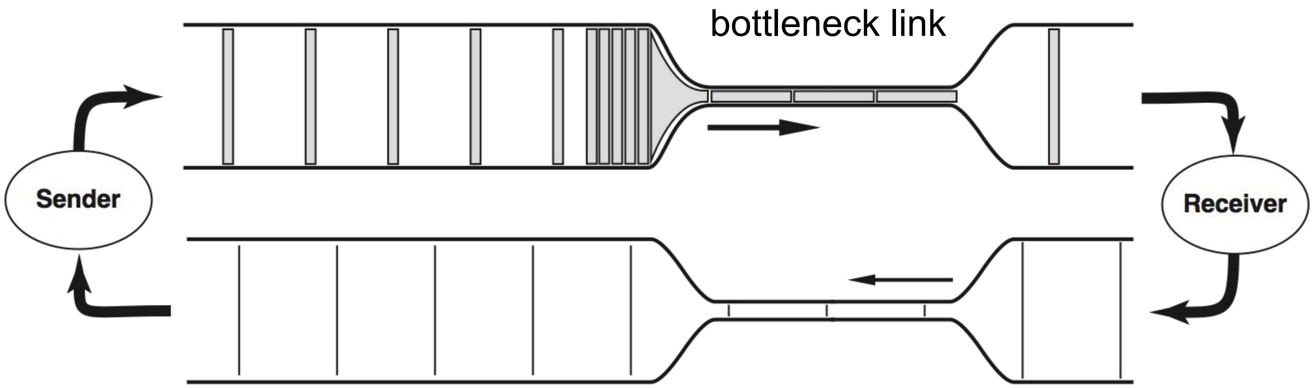
Why use congestion control?



Goals of congestion control:

1. **High throughput**: By fully utilizing bottleneck bandwidth
2. **Low latency, low loss**: By keeping queues short
3. **Approximate Fairness**: By sharing resources equally among flows

How to avoid congestion: matching the bottleneck



For full throughput with low queuing delay and low loss, congestion control must match sending process to network path delivery process, in 2 dimensions:

Data rate: **pace** data at the rate the network path delivers and acknowledges data

Data volume: maintain the minimum amount of data in flight (in the network) required for full rate:
the **BDP** = (**B**andwidth***D**elay **P**roduct) = Bottleneck_Bandwidth * Path_Round_Trip_Time

Congestion Control Challenges in Datacenter

Congestion control requirements

Transfers must complete quickly, low 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.

Operating System-bypassed transports.

Opportunities

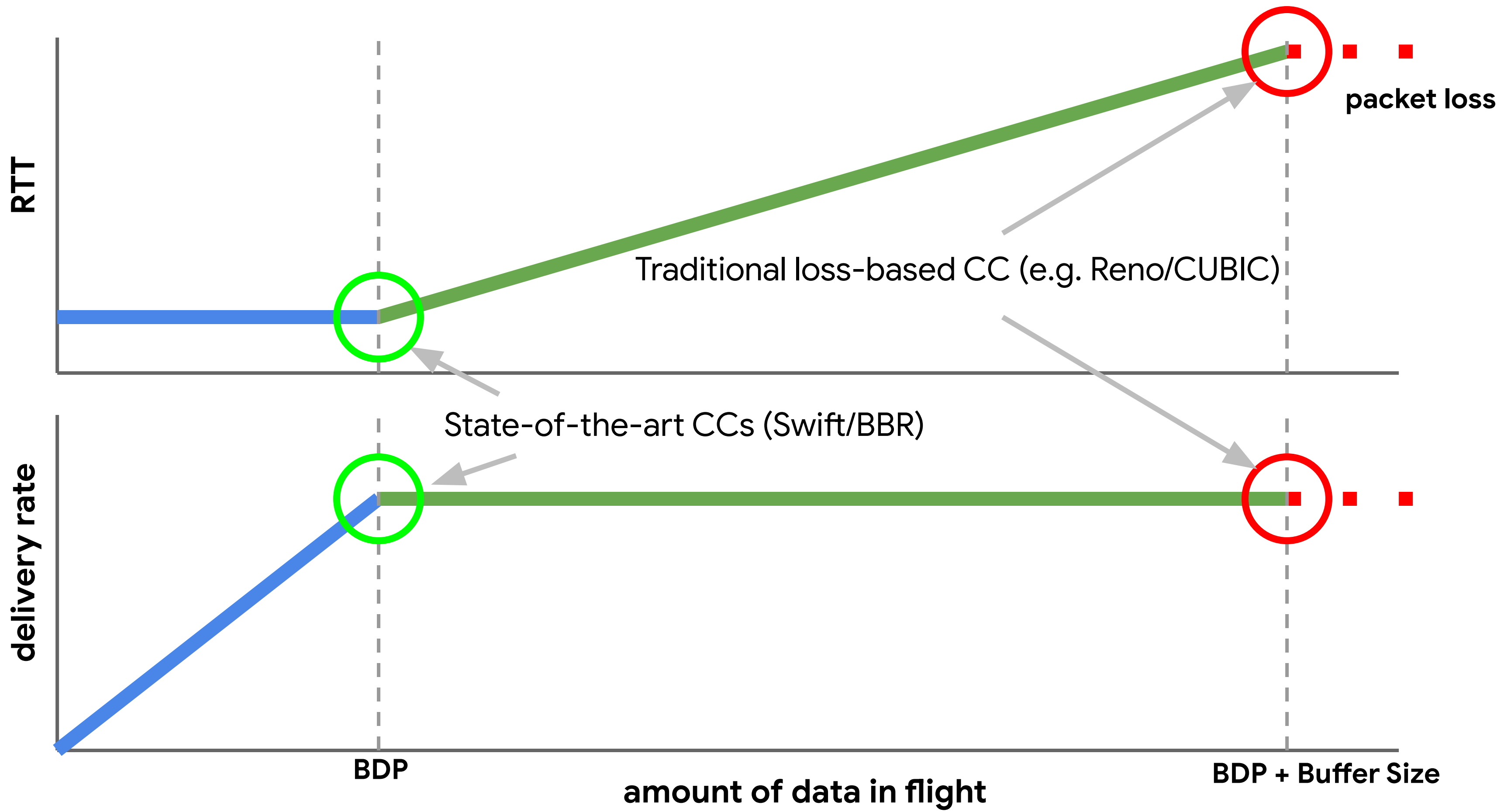
Hardware assistance.

Less worries of interoperability with legacy.

Explicit network feedback is easier to deploy.

Centralized control is possible.

Congestion control target operating points



Estimating the target operating point: congestion signals

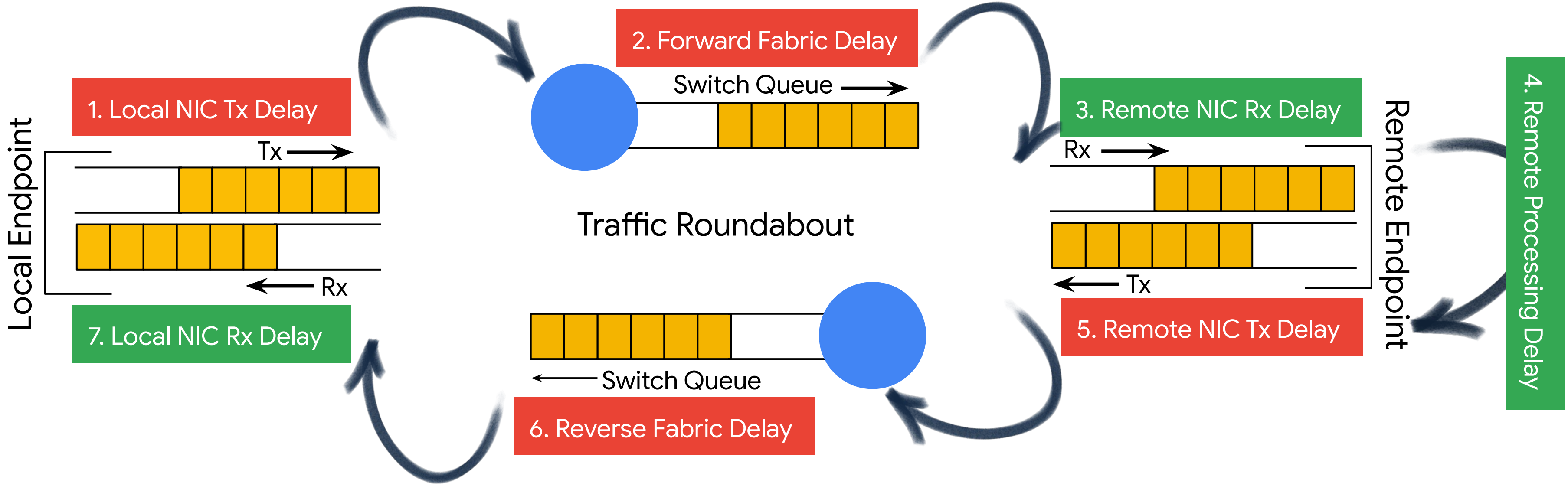
- Congestion control uses **congestion signals** to estimate whether a transport flow is sending too fast
- Congestion signals are diverse:
  - Different network environments offer different sets of signals
  - Signals offer differing levels of information/precision
- The commonly used congestion signals:

more information/precision ↓

Signal	How it is measured
Loss	When a queue exhausts buffer space, host/switch drops packets
Bandwidth	Sender measures the rate at which the receiver acknowledges packets
ECN	**E**xplicit **C**ongestion **N**otification marked if queue > byte or time threshold
Delay	Sender/receiver measure one-way or two-way delay to estimate queuing

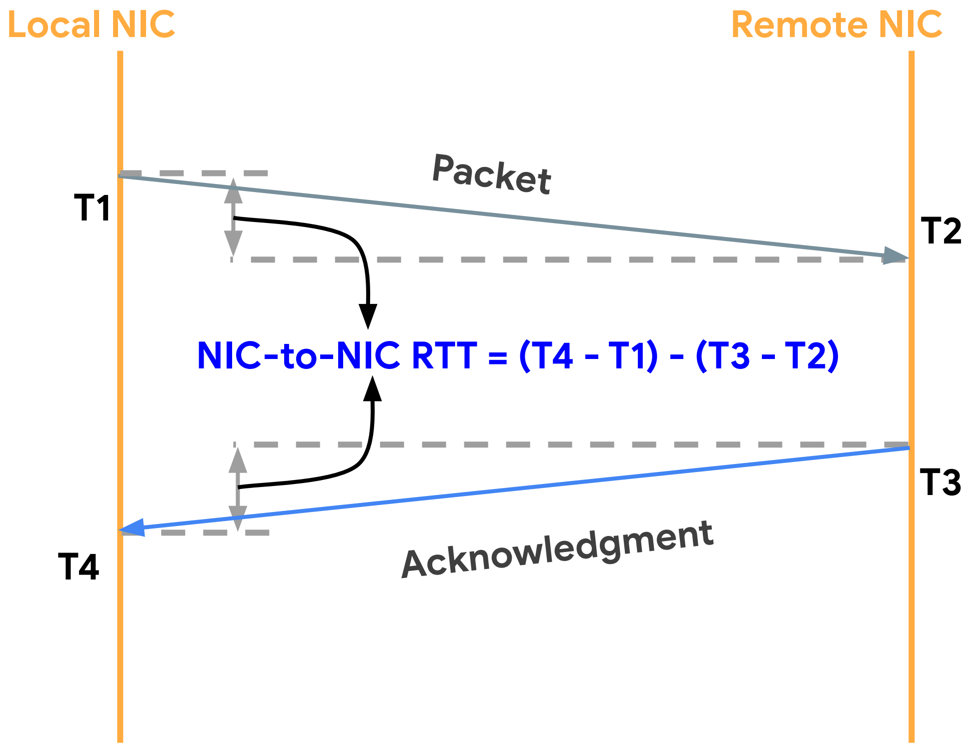
Delay Signals: End-to-end Delay Decomposition

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



There are multiple possible congestion points in the end-to-end path of a flow; endpoint congestion behaves differently than network congestion.

Computing Round-Trip Time and One Way Delay

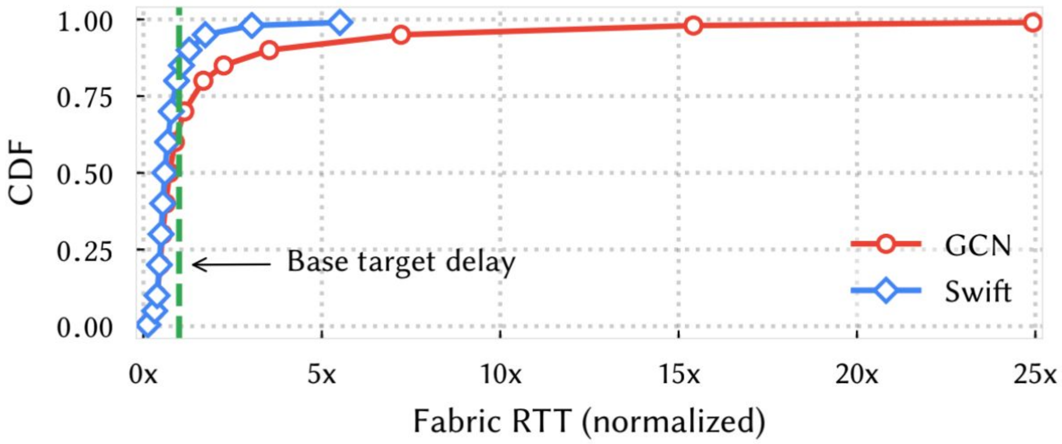


Swift: A delay-based Congestion Control

Simple Additive Increase Multiplicative Decrease based on a target-delay

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

Delay based CC keeps network RTT under control



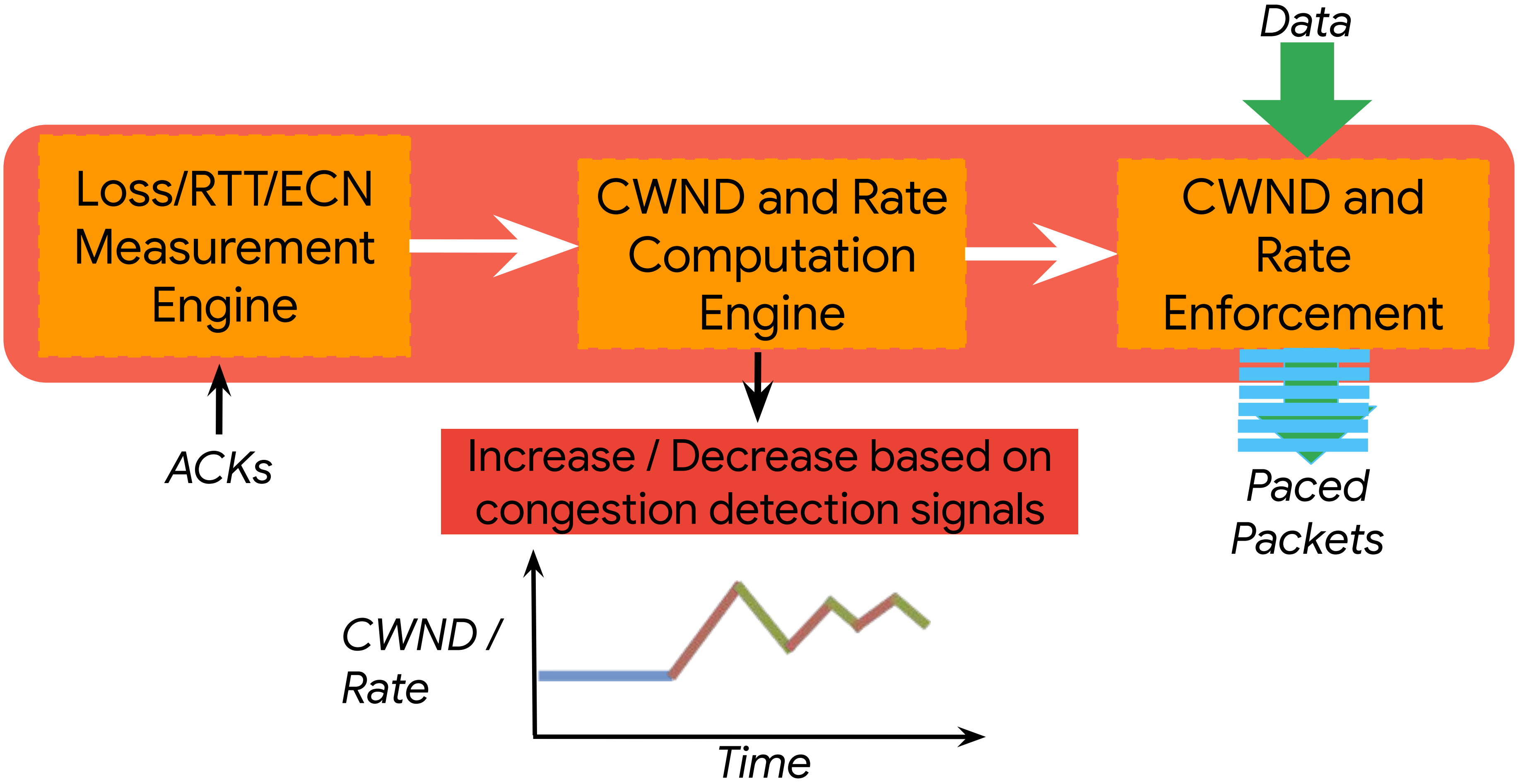
Comparison point:
GCN == Explicit Notification Based CC

Takeaways

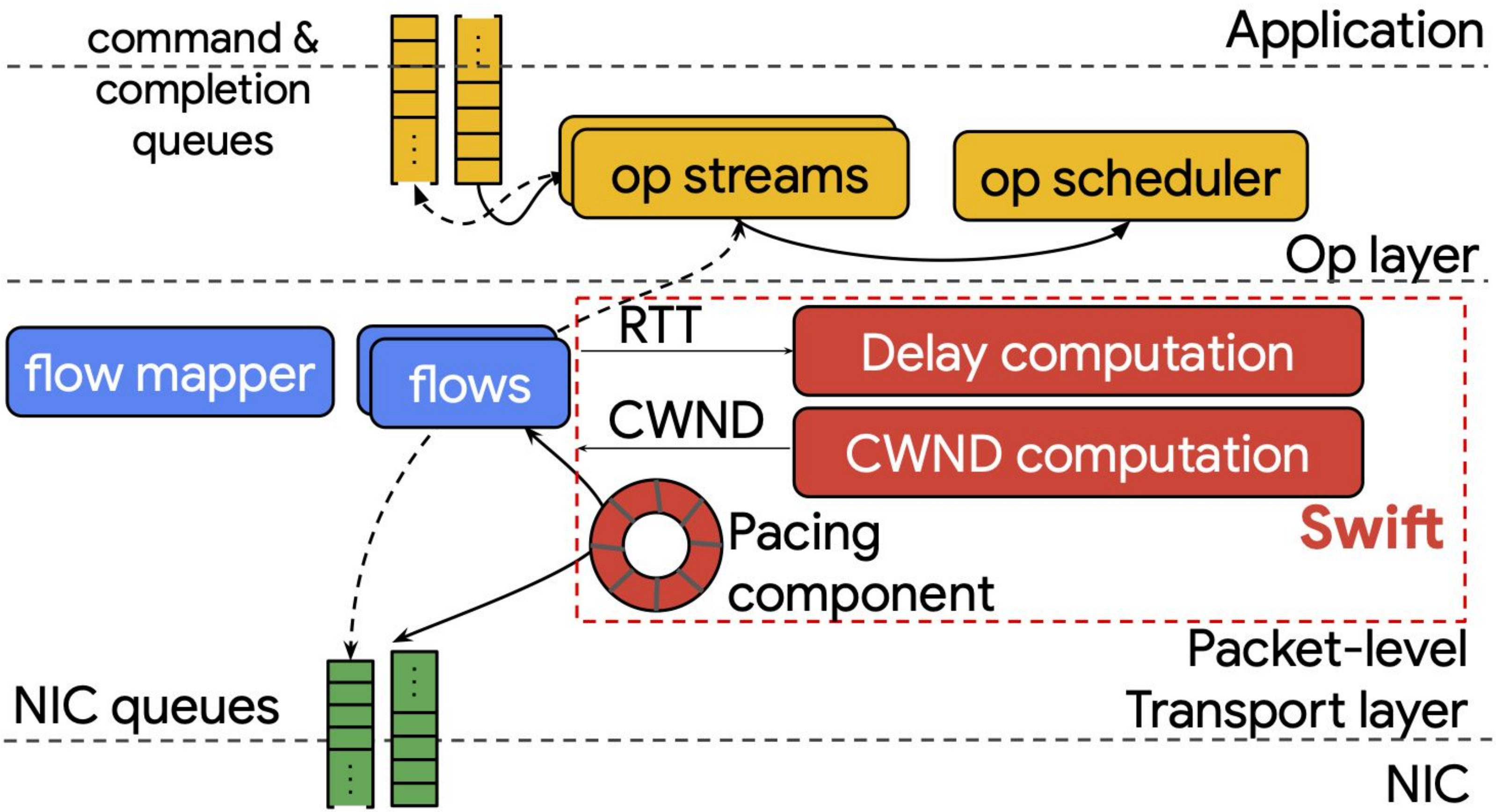
NIC-to-NIC (network) RTT with Swift is significantly smaller than GCN especially at the tail.

The target-delay as a tuning-parameter works well as the achieved RTT is close to the configured target (note that the figure only displays the base target delay modulo scaling)

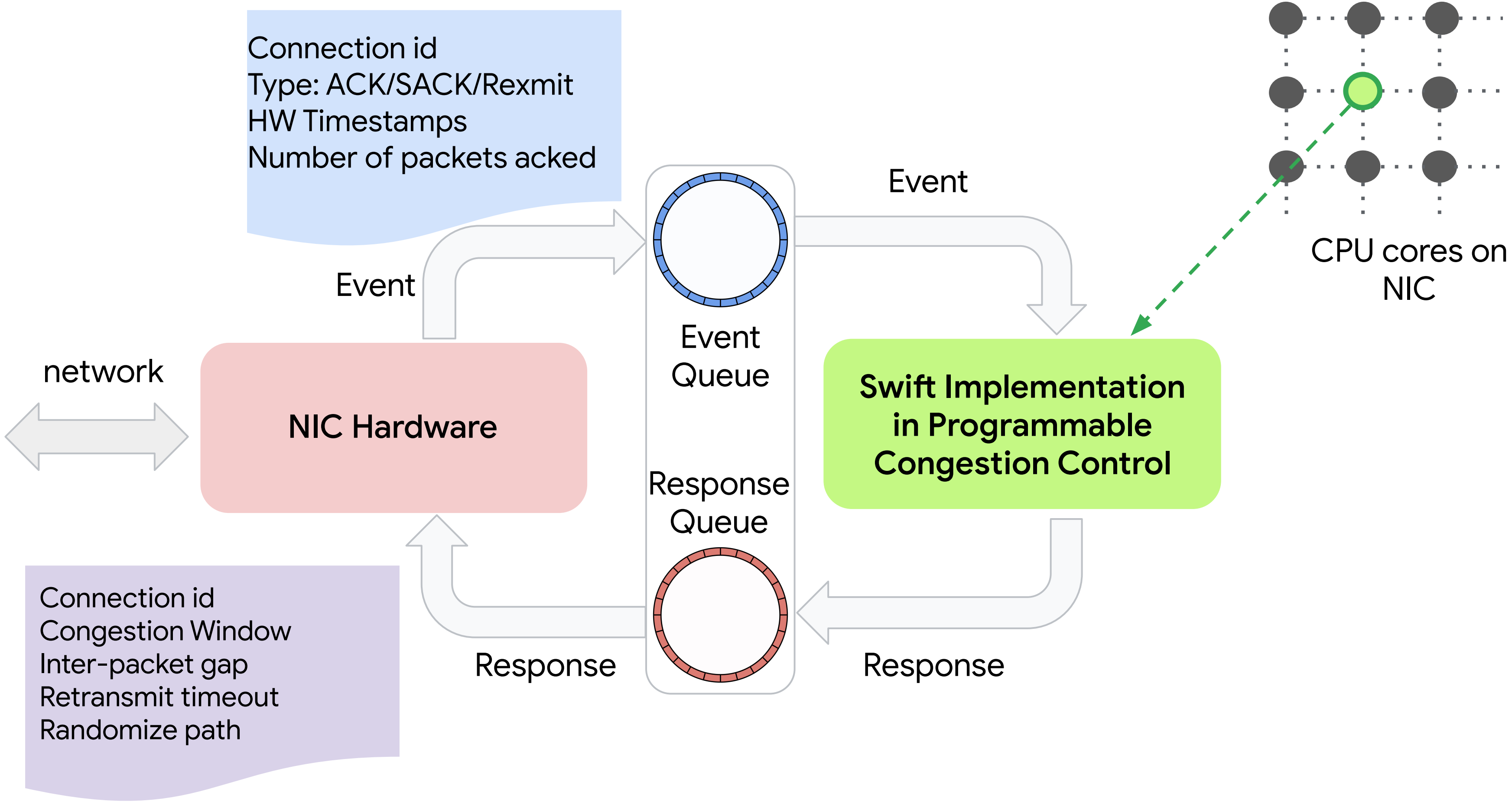
Congestion Control Framework implementing Swift



Swift in the context of PonyExpress



Swift Implementation in NICs



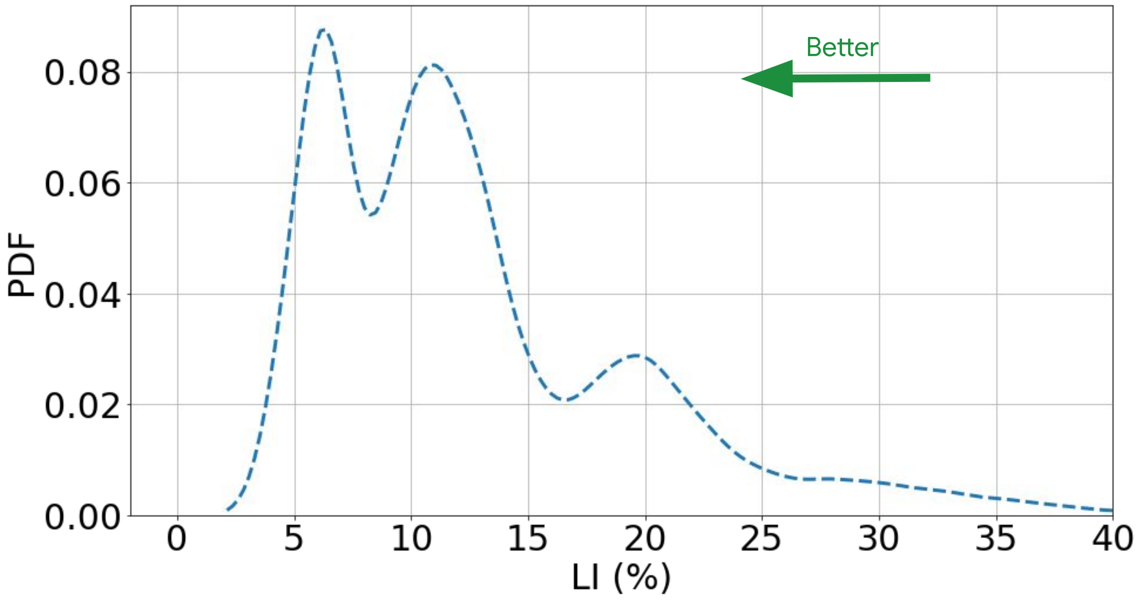
Load Balancing

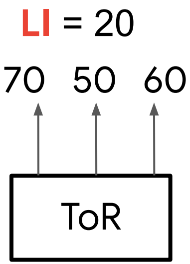
ECMP: Load Imbalance and Congestion Hotspots

Data Center Networks have path diversity which Equal/Weighted Cost Multi-path (E/WCMP) routing uses to balance load on network links.

- E/WCMP hashes a flow's src/dst ip/port (4-tuple) to determine the egress link.
- Dynamic bursts and flow sizes still cause links utilization imbalance.

ECMP: Load Imbalance and Congestion Hotspots





PDF of LI for ToR switches in a Google DCN where

- **Load Imbalance (LI)**=(max-min) utilization across switch uplinks over 30s
- ToRs have >70% average utilization across ports over +4 hours.

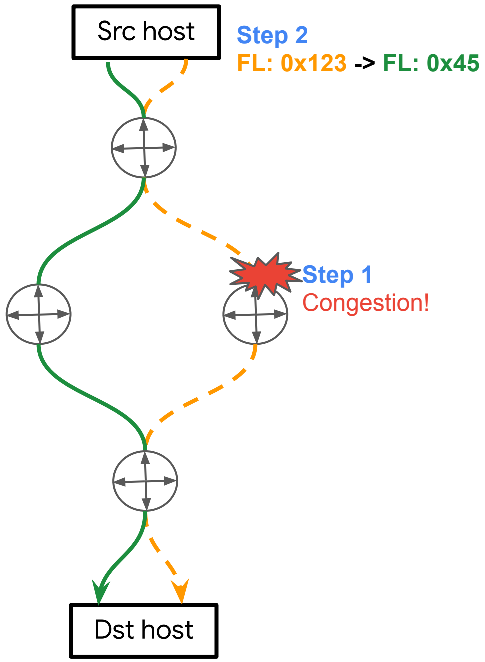
Congested flows could use other uncongested links with spare bandwidth!

Protective Load Balancing Intuition: Sense Congestion and Repath

Every connection already tracks end-to-end congestion state. Repath upon sustained congestion to find a congestion-free path.

How to repath from end-host?

Switches can be configured to use Flow Label field from IPv6 header plus 4-tuple to determine the egress port. The connection changes the Flow Label of outgoing packets to send on a random path



PLB - Protective Load Balancing

PLB algorithm

1. Mark a round-trip as congested if congestion (measured via round-trip time) above threshold.
2. After several consecutive congested rounds
   - Wait until connection goes idle and then repath to minimize packet reordering.
   - If connection does not go idle, force repath after specified congested rounds.
3. Repeat (goto 1.)

Key Properties of PLB

- Nimbly move mice flows away from elephant-clogged bottlenecks
- Waits for congestion control to react before it repaths

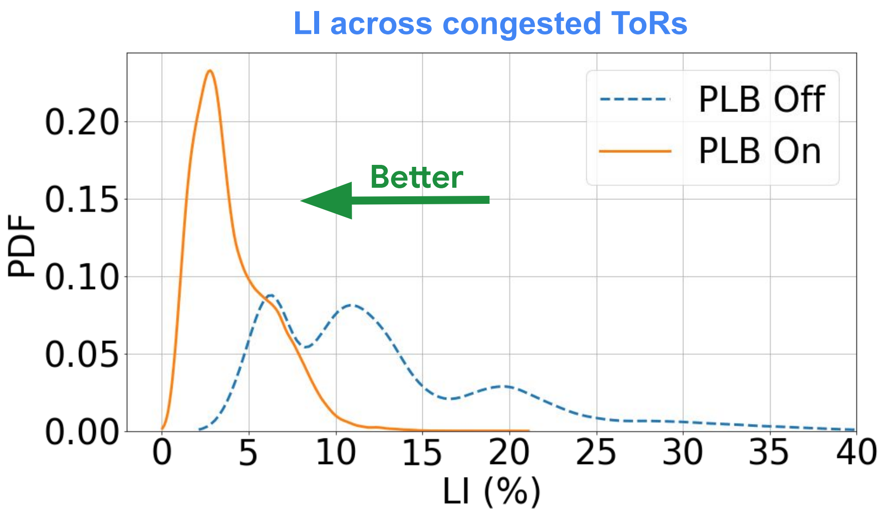
PLB Deployment in Google

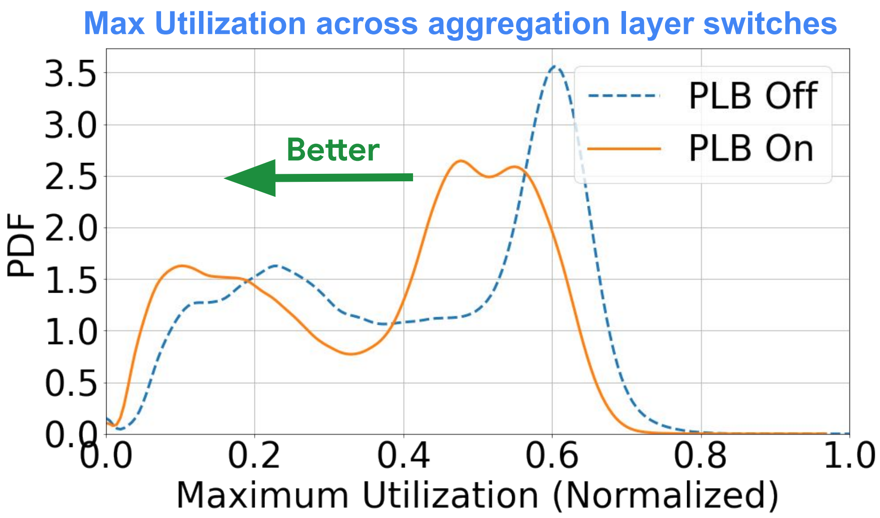
Deployment Features

- Simple changes (~50 LoCs) in each host networking stack (TCP BBRv2, Pony Express Swift[1,2]).
- Google DCs already fully use IPv6.
- Requires only switch config change and sender code change.

[1] Snap: a Microkernel Approach to Host Networking, SOSP 2019
[2] Swift: Delay is Simple and Effective for Congestion Control in the Datacenter, SIGCOMM 2020

PLB reduces congestion hotspots across switches in datacenter



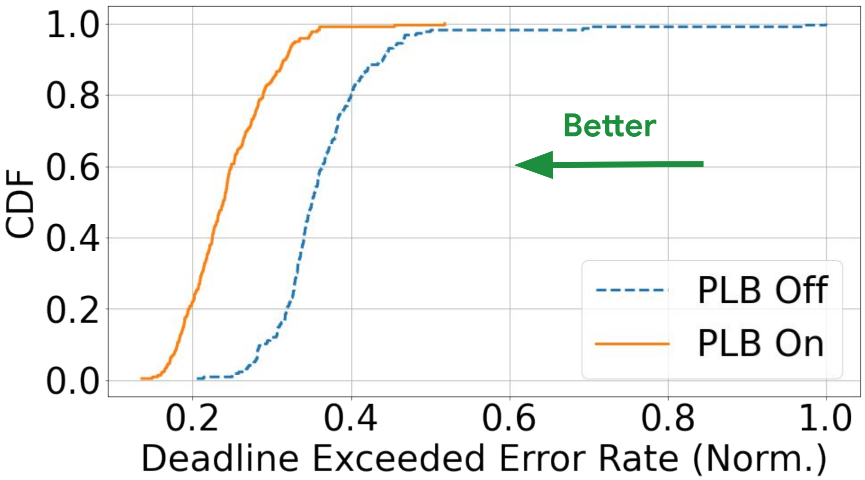


50% lower packet drops

33% lower packet drops

LI=(max-min) utilization across switch uplinks over 30s

Better load balancing across datacenter translates into application gains



Median deadline exceeded error rate drop by 66% for a low-latency filesystem

Summary

- Link load Imbalance exists at different levels of data center hierarchy
  - Top of Rack Switches, core switches, gateways to WAN

- PLB leverages existing DCN host and network features
  - Incremental PLB deployment receives immediate benefit

- Congestion is a powerful metric to *align* traffic to true carrying capacity
  - Traffic Engineering can create hotspots due to workload fluctuation or demand misprediction but can recover in O(sec) to O(min). PLB helps meanwhile.

Traffic Shaping and Pacing

Spreading traffic with shaping and pacing

Shaping: spreading traffic over time to constrain bandwidth use by users, based on policy.

Problem: without shaping, bandwidth is allocated per transport connection, via congestion control

Solution: shaping allocates bandwidth per-user based on the business priority of each user

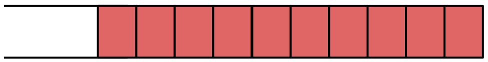
Pacing: spreading traffic over time, to reduce bursts, queues, queuing delay, packet loss (Swift, BBR CCs)

Problem: unpaced traffic creates NIC-line-rate bursts, increasing queuing delays and packet loss

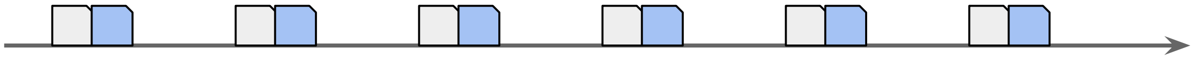
Solution: pacing inserts delays between packets, reducing queuing delays and packet loss

Unpaced traffic can create large line-rate bursts:

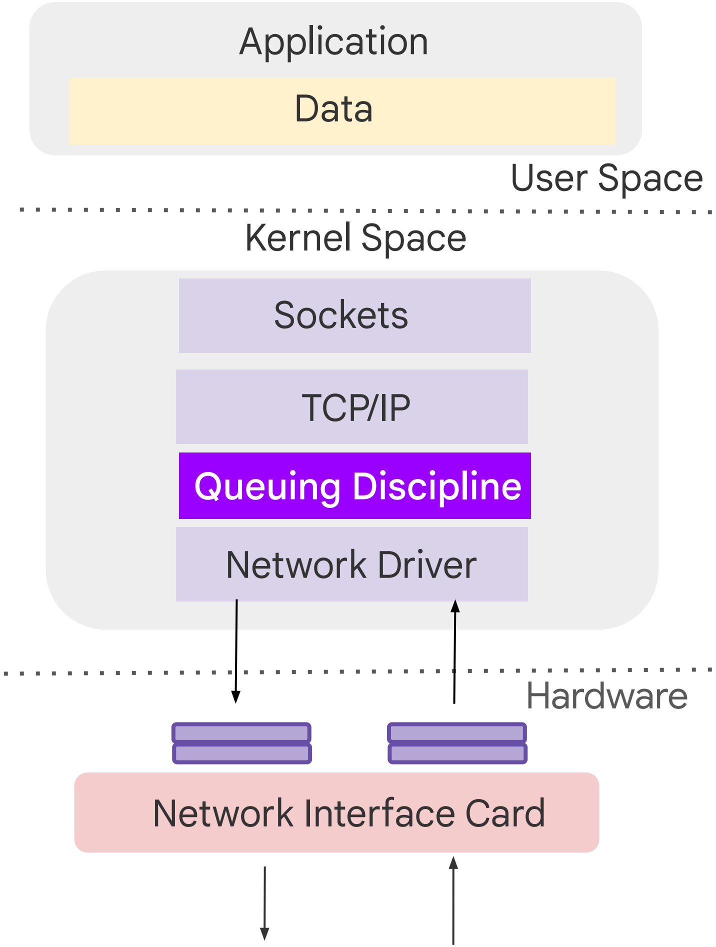
Line-rate bursts => queues at bottleneck:



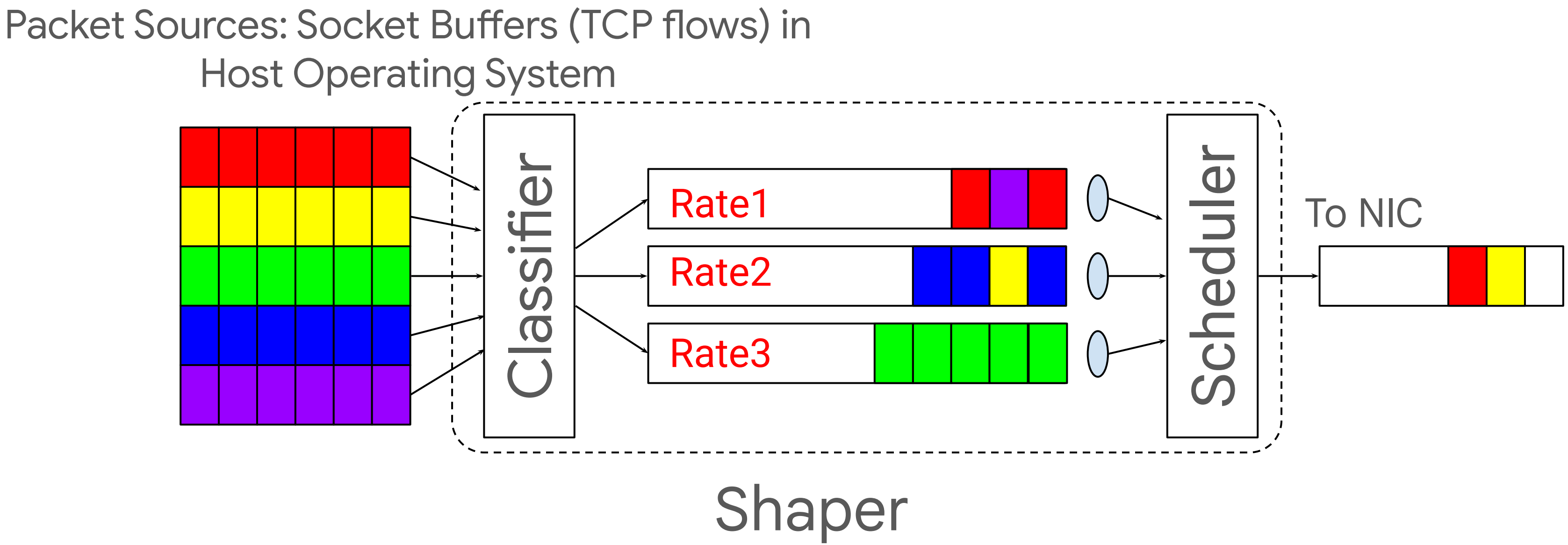
Paced traffic has smaller burst and better mixing of flows:



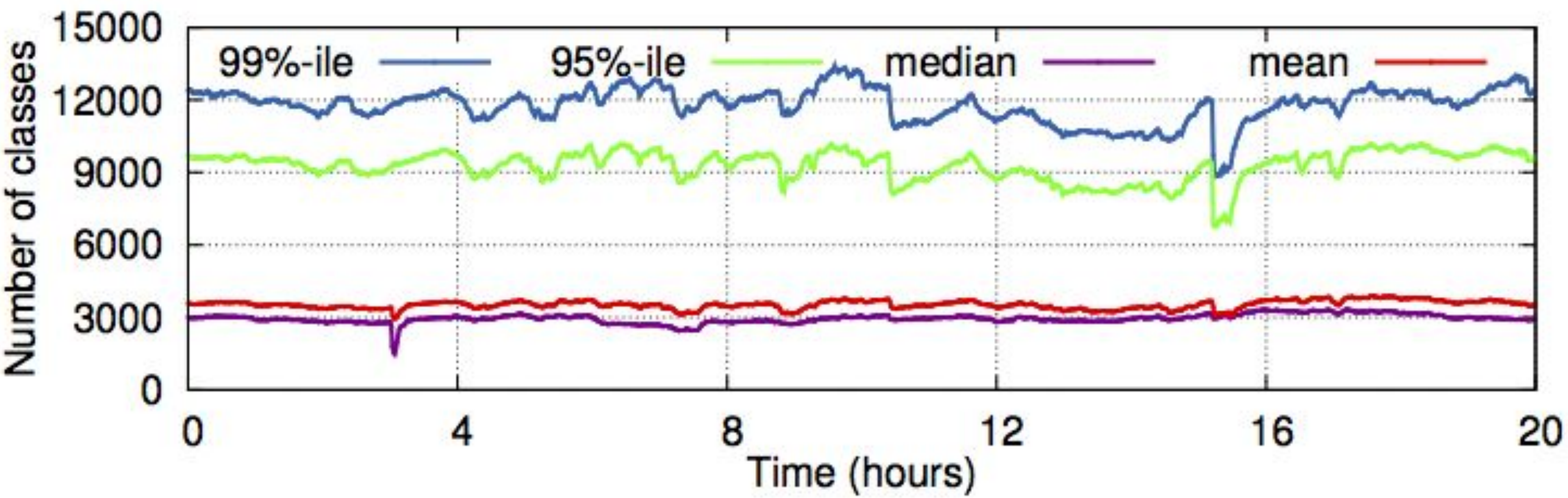
Pacing vastly reduces queuing:


Application
Data
User Space
Kernel Space
Sockets
TCP/IP
Queuing Discipline
Network Driver
Hardware
Network Interface Card

Traffic Shapers in Practice



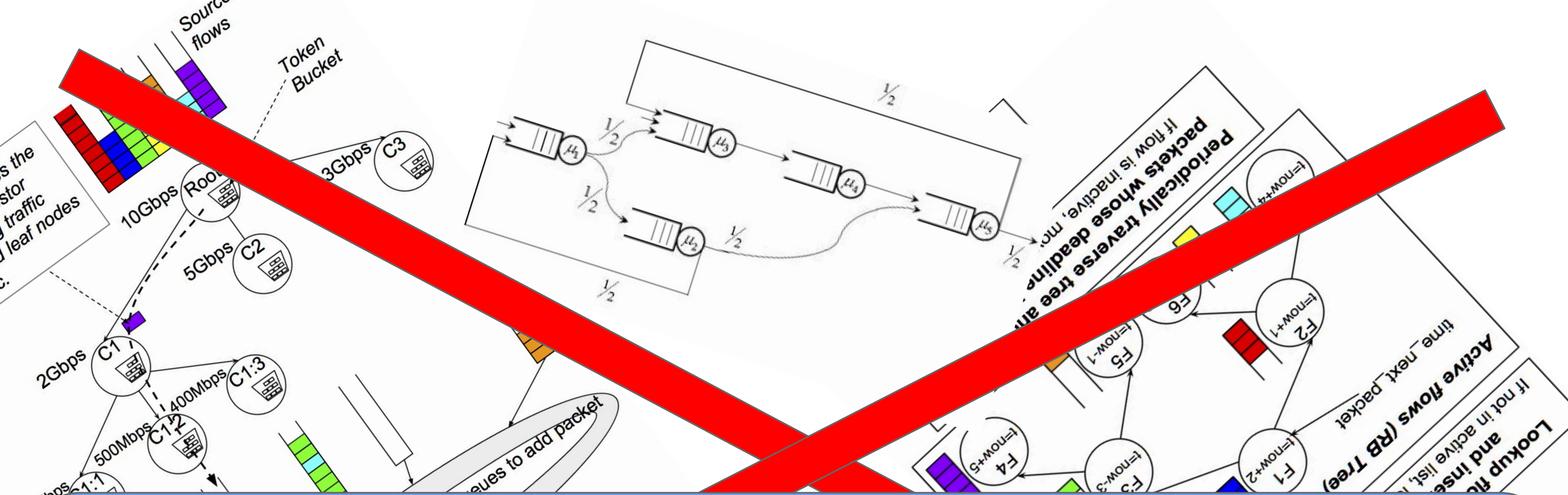
Example: #Queues instantiated on a single machine



Google

Queues are high maintenance

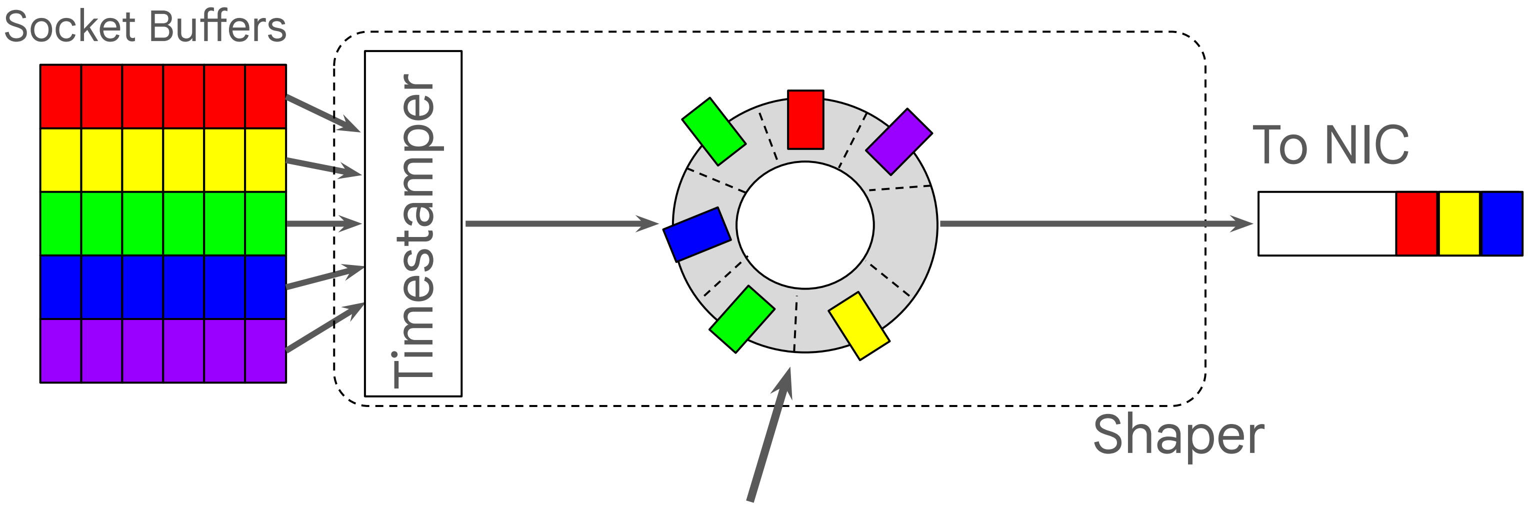
- CPU cost of maintaining queues grows super-linearly with #queues.
  - ➔ Polling queues for packets to process.
  - ➔ Complex algorithms for tracking active queues.
  - ➔ High overhead data structures.
  - ➔ Garbage collection.
- Synchronization cost on multi-CPU systems is dominated by locking and contention overhead when sharing queues amongst CPUs.
- Complexity of memory consumption and management grows with #queues.



We do not need these queues and their associated cost.

Using *Time* as a basic construct to shape flows, we can get all of the benefits of the queues, at a low cost.

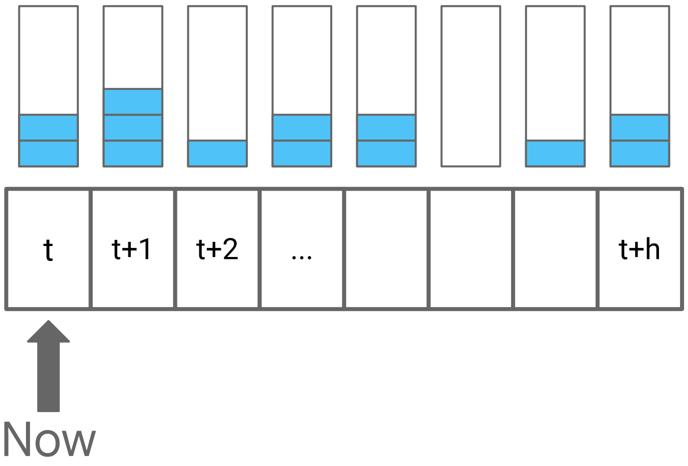
Design Principles of Timing Wheel based Traffic Shaper



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

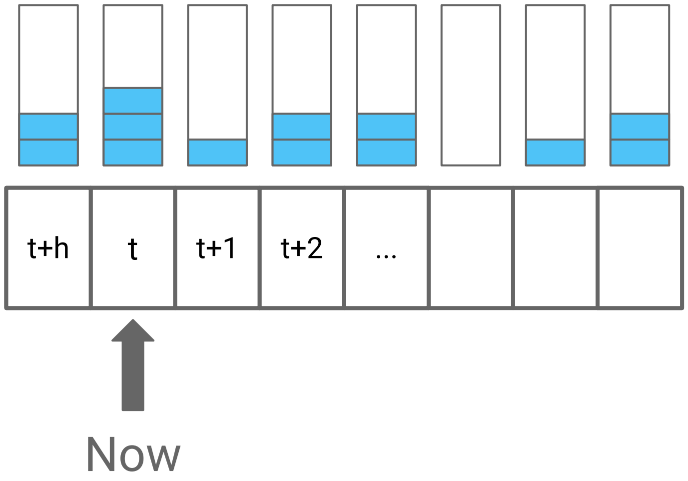
Single Time-Indexed Queue

- Single queue to handle tens of thousands of flows and wide ranging rates.
- O(1) Enqueue/Dequeue at line rate is key.
- Timing Wheel [Varghese et. al. SOSP ‘87].
    - Circular array of buckets.
    - Each bucket represents a time slot of fixed size.
    - Array represents time span from **now** to **horizon**.



Single Time-Indexed Queue

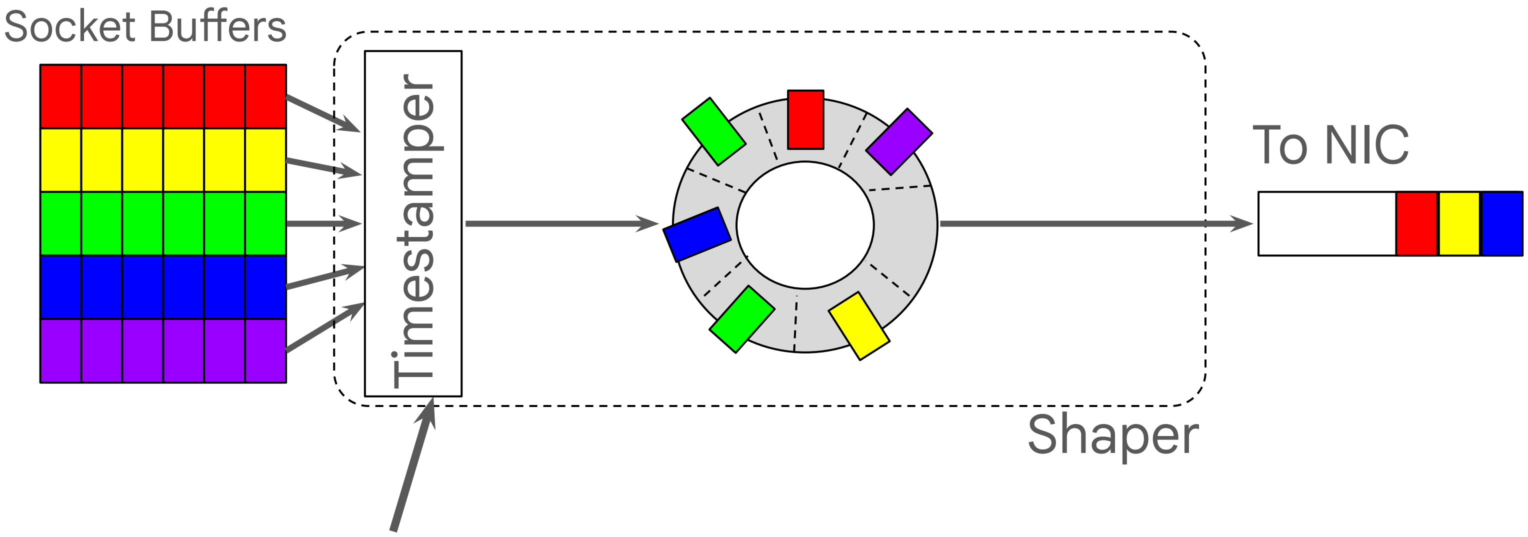
- Single queue to handle tens of thousands of flows and wide ranging rates.
- O(1) Enqueue/Dequeue at line rate is key.
- Timing Wheel [Varghese et. al. SOSP ‘87].
    - Circular array of buckets.
    - Each bucket represents a time slot of fixed size.
    - Array represents time span from **now** to **horizon**.



O(1) Enqueue / Dequeue regardless of #packets

Number of Packets in Timing Wheel	1000	4000	32000	256000
Overhead per Packet (ns)	22	21	21	21

Design Principles of Timing Wheel based Traffic Shaper



Determine Departure Time based on Pacing/Shaping Rate.

Timestampers

Compute **Earliest Departure Time** based on policy.

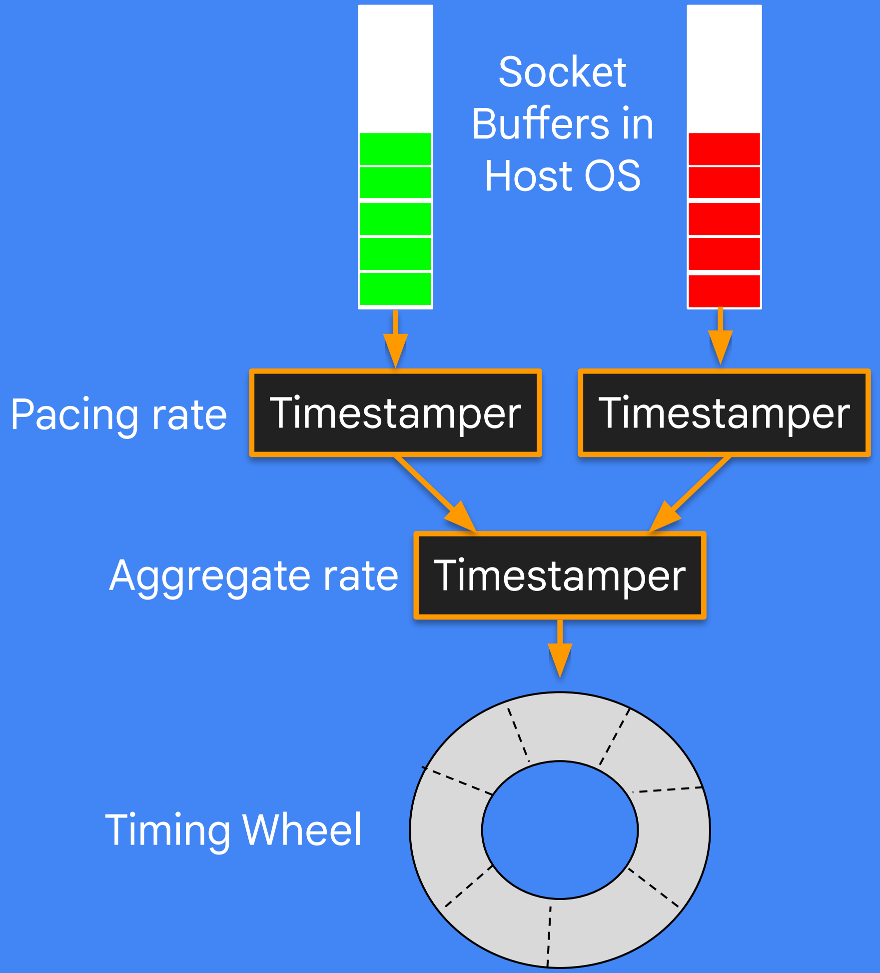
Example:

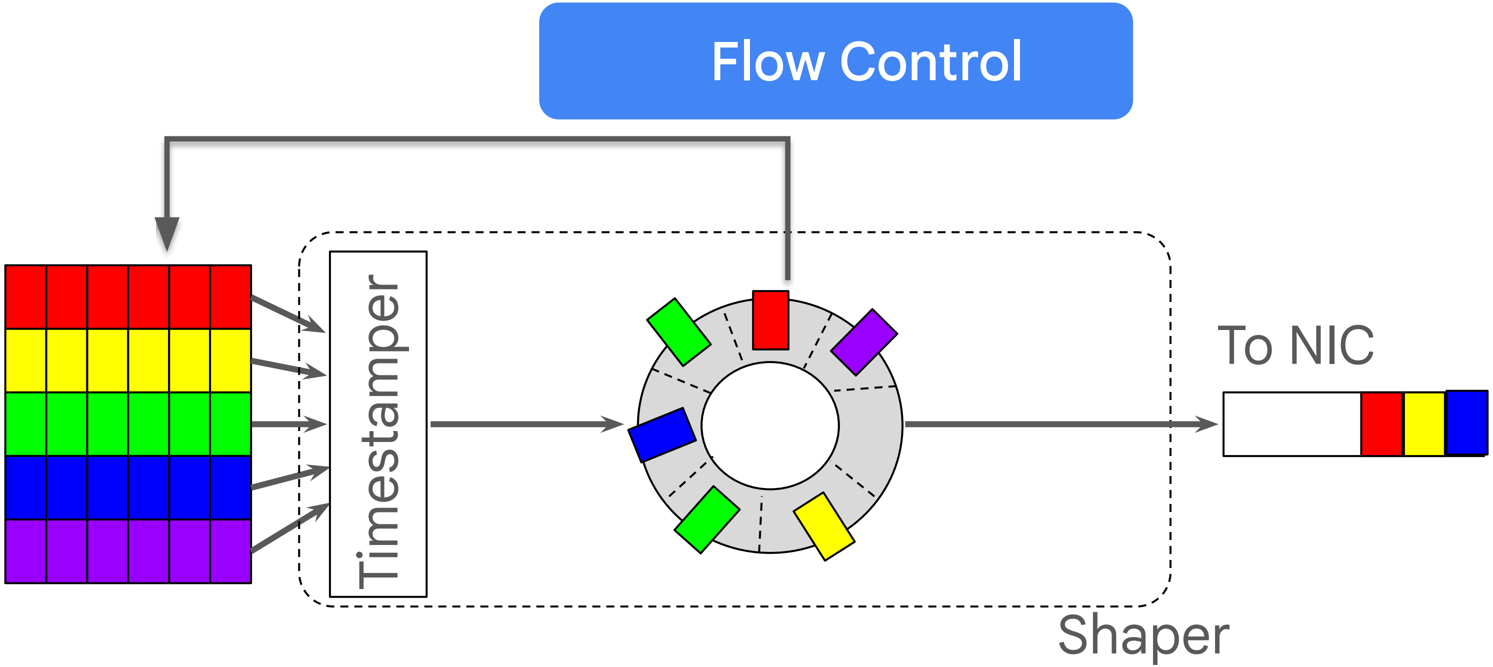
- TCP timestamps a packet based on its pacing rate.
- Bandwidth Enforcer timestamps a packet based on flow aggregate rate.

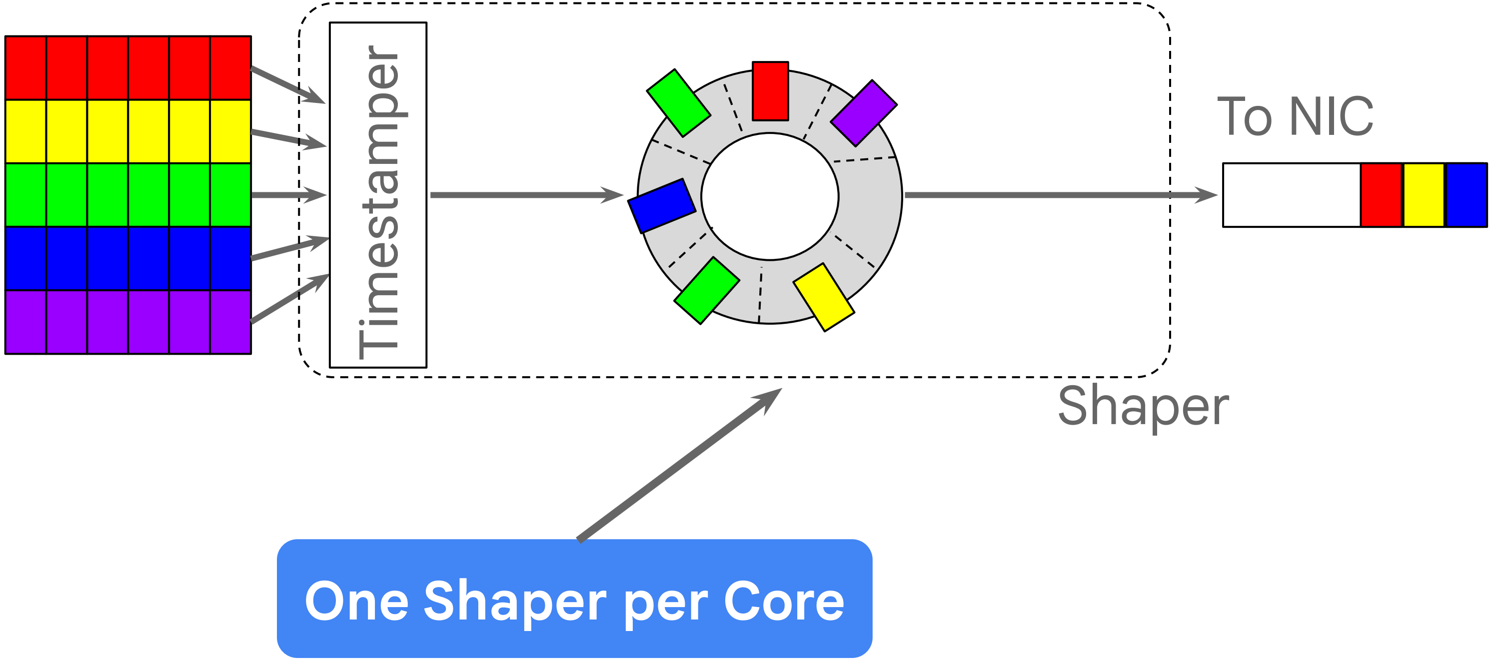
Consolidate Timestamps by choosing the largest one (== smallest rate).

$$\text{NextTS} = \text{LastTS} + \frac{\text{SizeOfPacket}}{\text{ConfiguredRate}}$$

Google




Flow Control
Timestamper
To NIC
Shaper

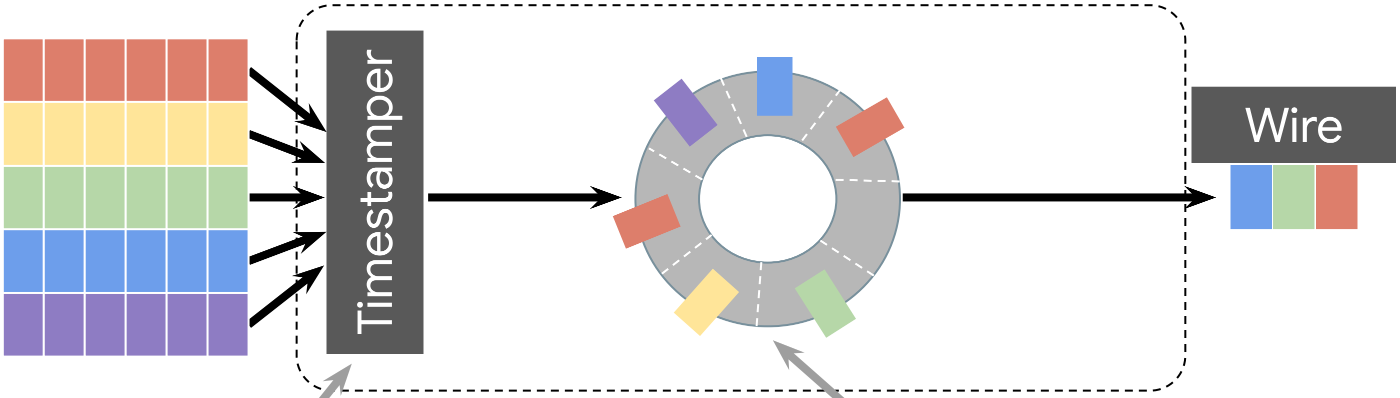

Timestamper
To NIC
Shaper
One Shaper per Core

Timing Wheel: efficient pacing/shaping of transmissions

Compute Earliest Departure Time based on pacing/shaping rate

Enqueue packet in Timing Wheel

Dequeue packet and send it out



Packets in Socket Buffers

Schedule departure based on prev departure time, packet size, rate

Timing Wheel: a single, O(1), time-indexed queue, ordered by packet Earliest Departure Timestamp

Quality-of-Service

QoS (Quality of Service): allocating resources at links

Problem: When there is congestion at a link, how should the host/switch/router allocate resources (buffer space, bandwidth)?

QoS (Quality of Service): allocating resources at links

Problem: When there is congestion at a link, how should the host/switch/router allocate resources (buffer space, bandwidth)?

Solution: Sender application => priority class => QoS in packet => queue => 1: buffer, 2: bandwidth

QoS (Quality of Service): allocating resources at links

Problem: When there is congestion at a link, how should the host/switch/router allocate resources (buffer space, bandwidth)?

Solution: Sender application => priority class => QoS in packet => queue => 1: buffer, 2: bandwidth

E.g., Storage, MapReduce, distributed in-memory file system, web search indexing, query serving, and caching services.

E.g., Latency-sensitive, throughput-intensive, best effort.

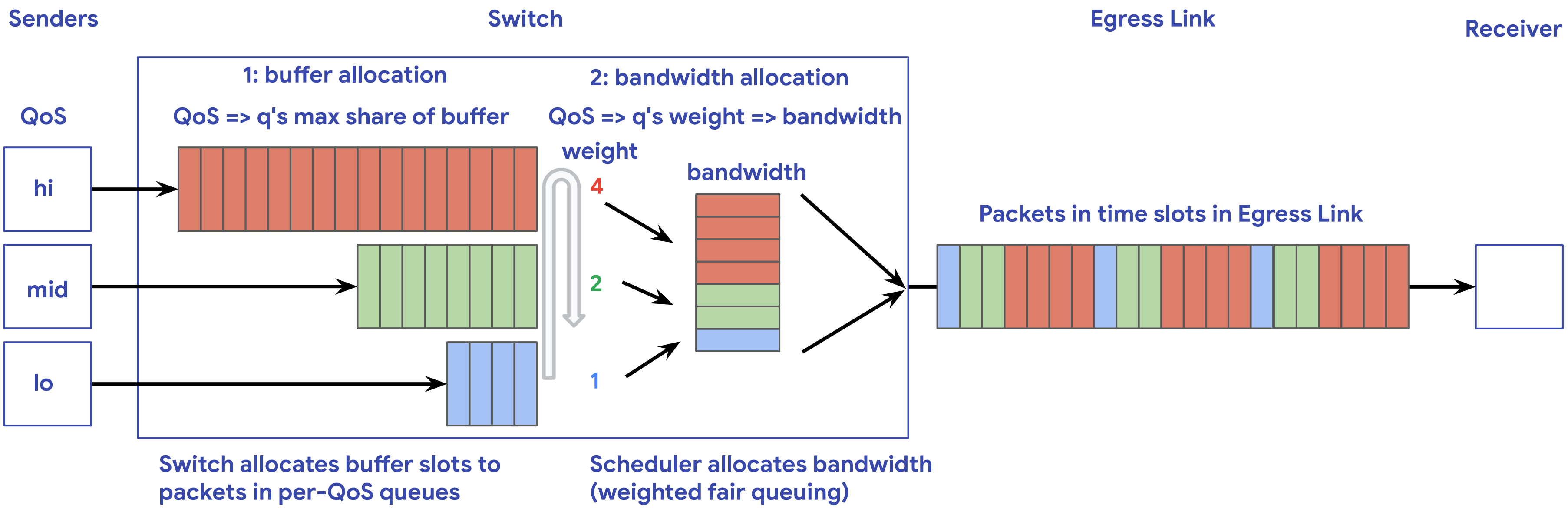
Encoded in IP header (DSCP)

E.g., low, mid, high.

QoS (Quality of Service): allocating resources at links

Problem: When there is congestion at a link, how should the host/switch/router allocate resources (buffer space, bandwidth)?

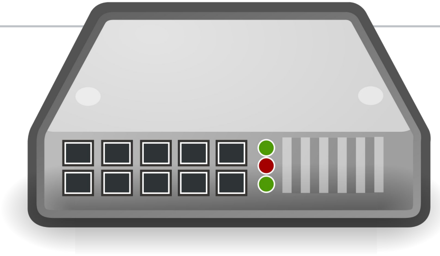
Solution: Sender app priority => service class => QoS in packet => queue => 1: buffer, 2: bandwidth



Where does QoS Matter?

Sender: Prioritization in host networking stack and NICs

Receiver: Prioritization in host networking stack and NICs



Packet buffering & scheduling in switches

QoS prioritization in the network

- QoS only matters when a switch port is 100% utilized when packets arrive.
  - Under <100% utilization, every packet is sent at line-rate hence QoS does not matter
- QoS prioritization policy under congestion.
  - Send higher priority packets (than the lower ones) at small timescales (<ms)
  - On buffer overrun, drop lower priority packets first (if available)
- Need to provision buffers carefully.