Lecture 4: Data Center Transport

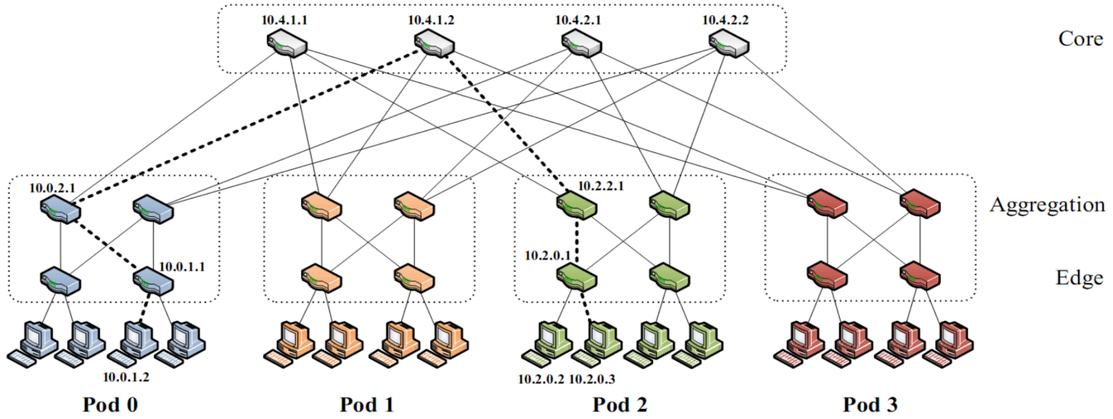
CS 234 / NetSys 210: Advanced Computer Networks

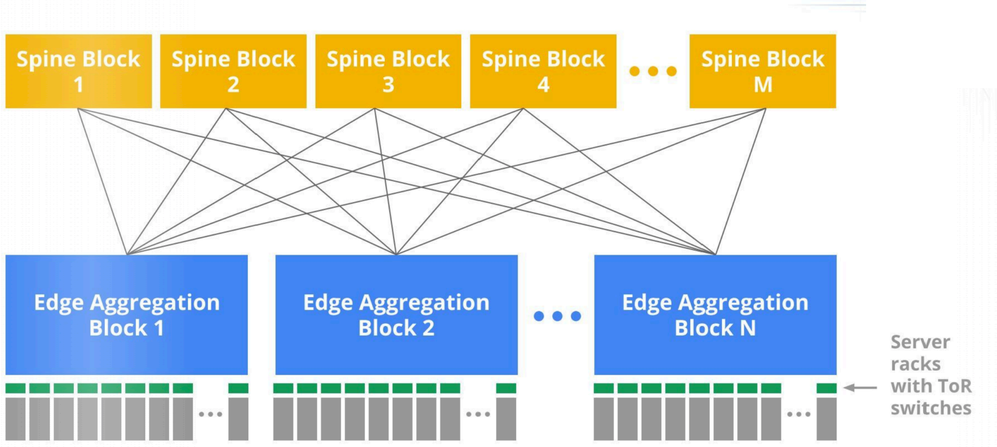
Sangeetha Abdu Jyothi

UCIrvine

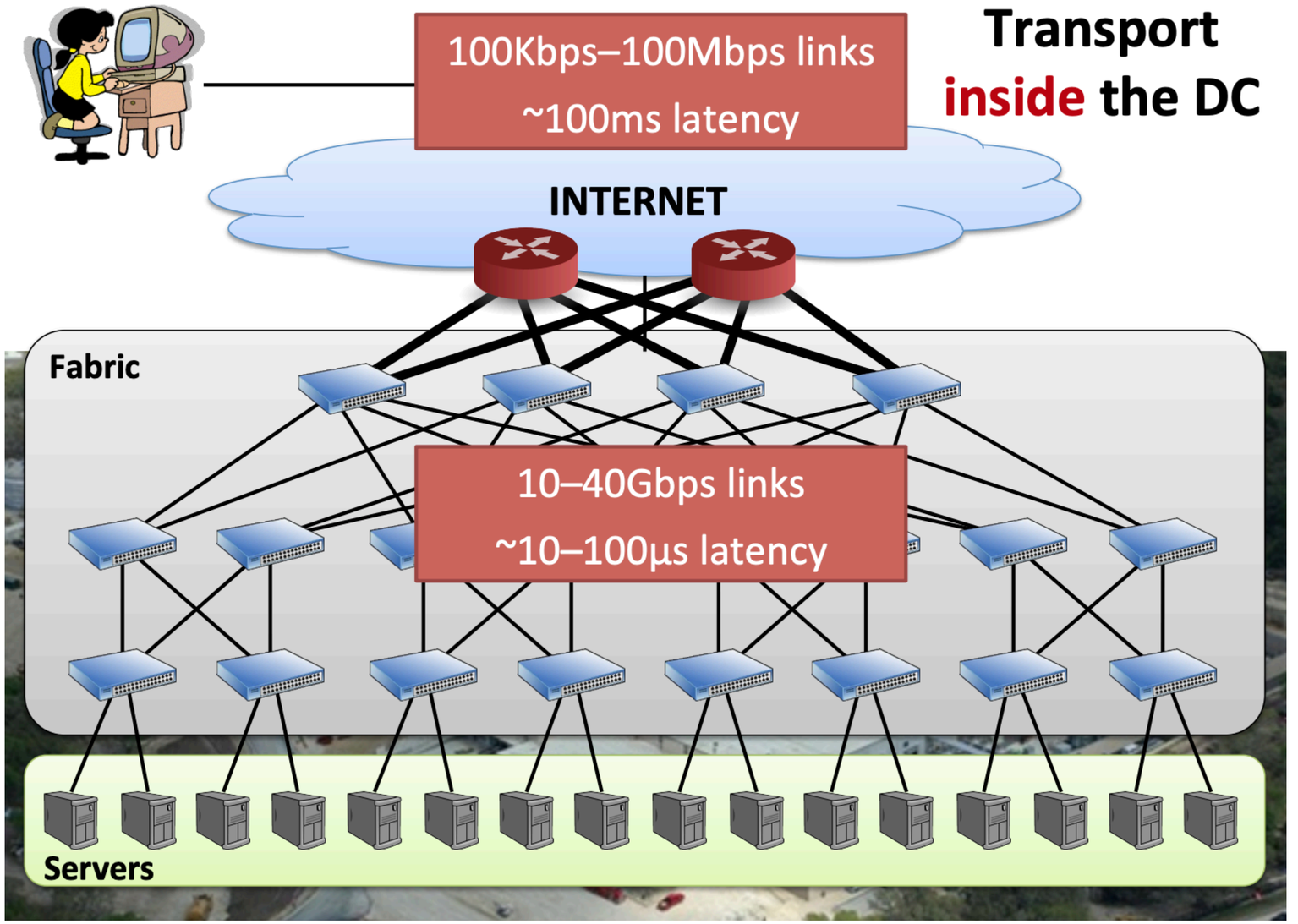
Parts of this lecture uses material from Mohammad Alizadeh and Radhika Mittal

Last Class: DC Topology

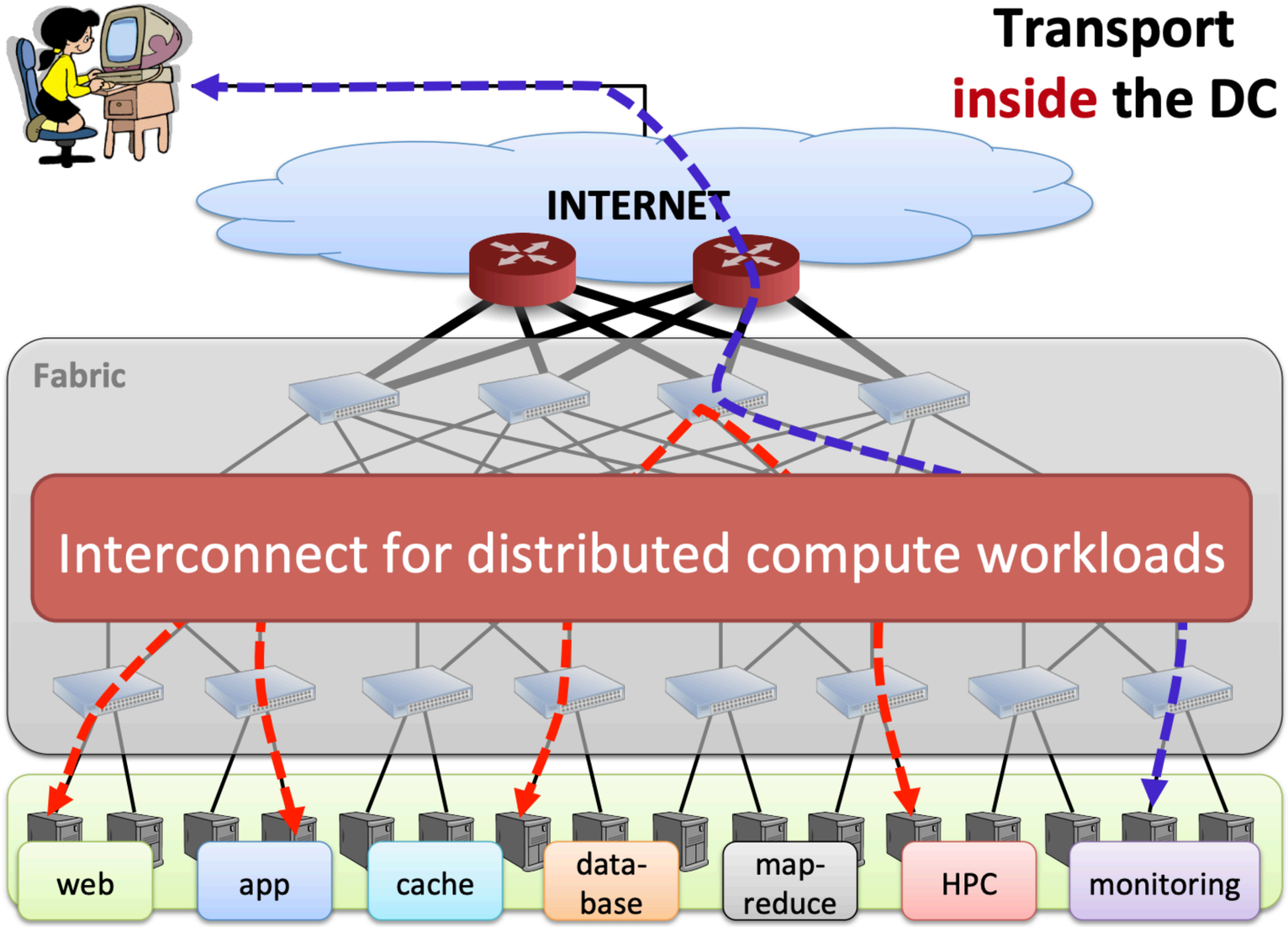




Today: Data Center Transport



Data Center Transport



What's Different About DC Transport?

- Network characteristics
  - Very high link speeds (Gb/s); very low latency (microseconds)
- Application characteristics
  - Large-scale distributed computation
- Cheap switches
  - Single-chip shared-memory devices; shallow buffers
- Challenging traffic patterns
  - Diverse mix of mice & elephants
  - Incast

Data Center Traffic: Mice and Elephant Flows

Mice & Elephants

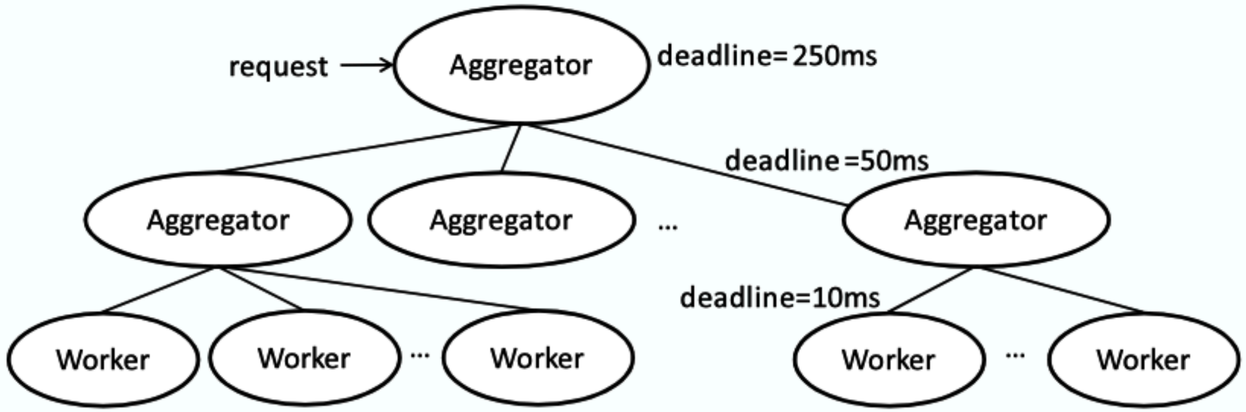
Short messages

(e.g., query, coordination) → **Low Latency**

Large flows

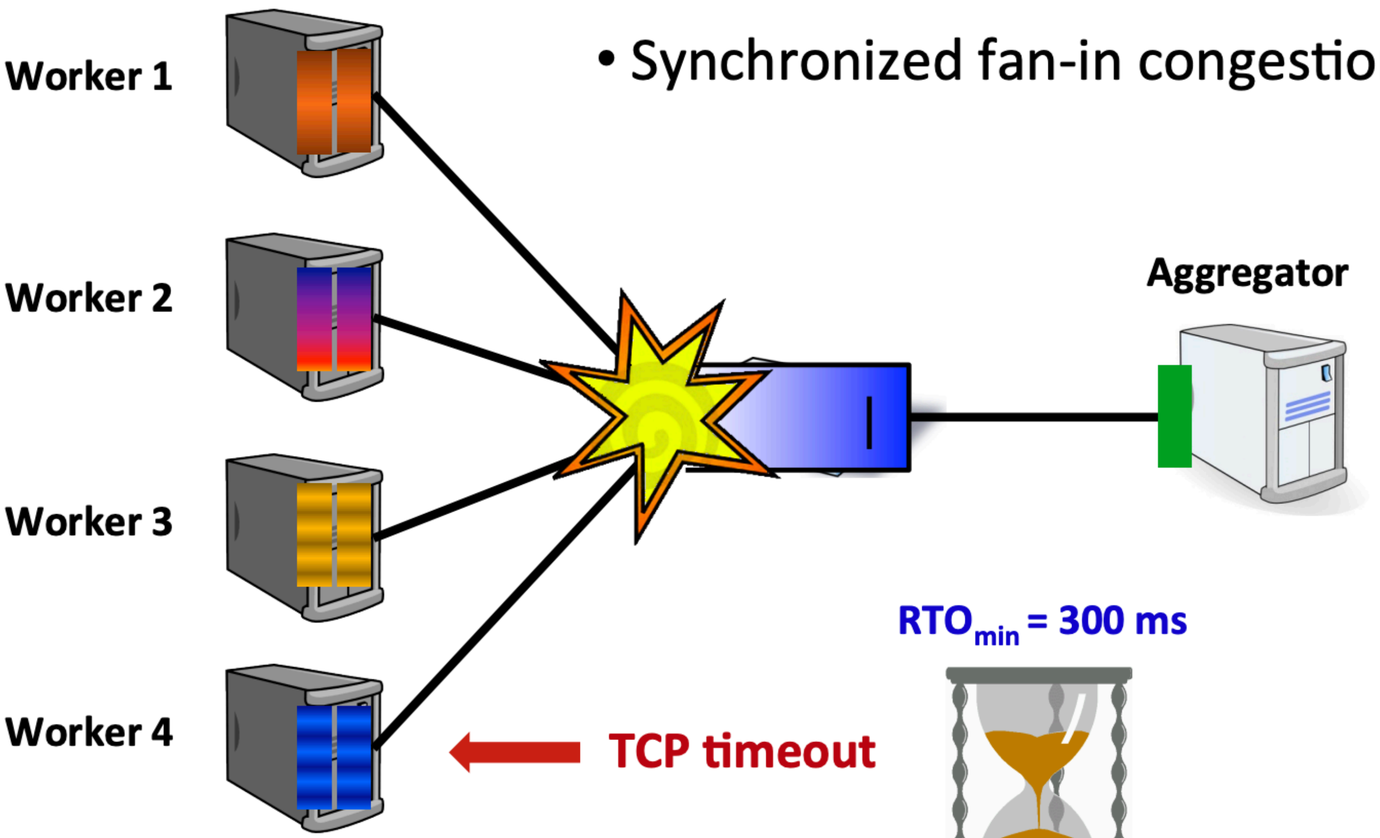
(e.g., data update, backup) → **High Throughput**

Data Center Traffic: Incast



The partition/aggregate design pattern

Data Center Traffic: Incast



✧ Vasudevan et al. (SIGCOMM'09)

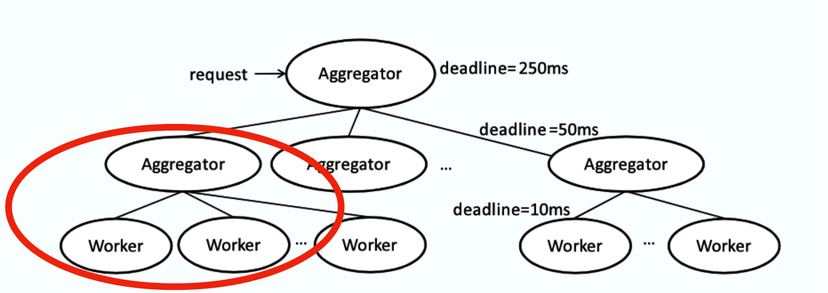
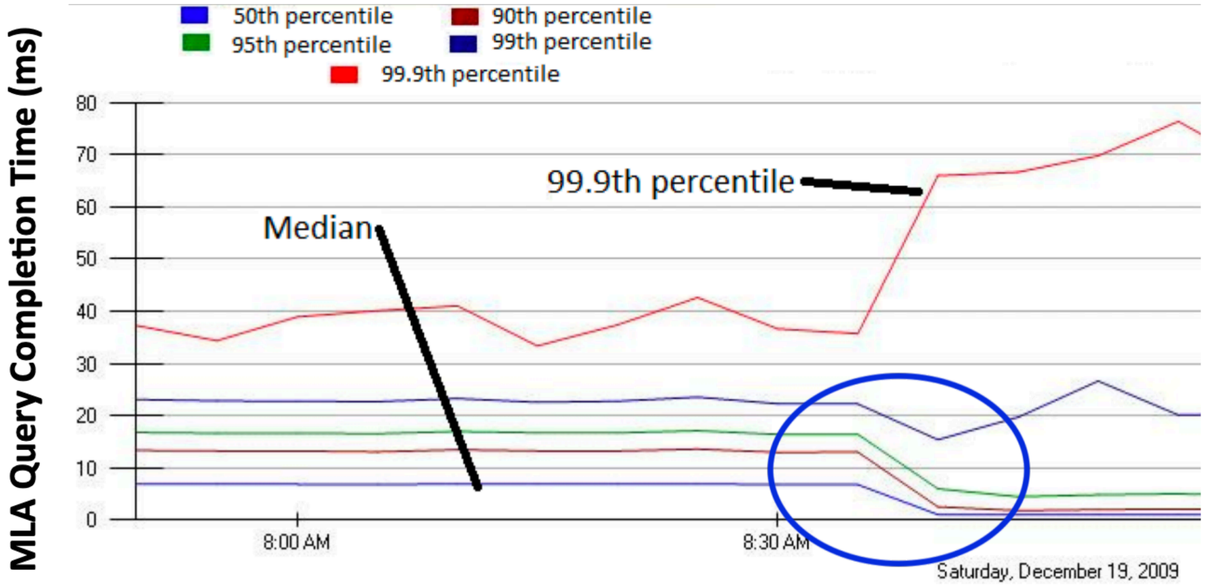


Figure 2: The partition/aggregate design pattern

Incast in Bing

- Solutions
  - Limit max response size to 2KB
  - Application-level jittering to desynchronize the responses



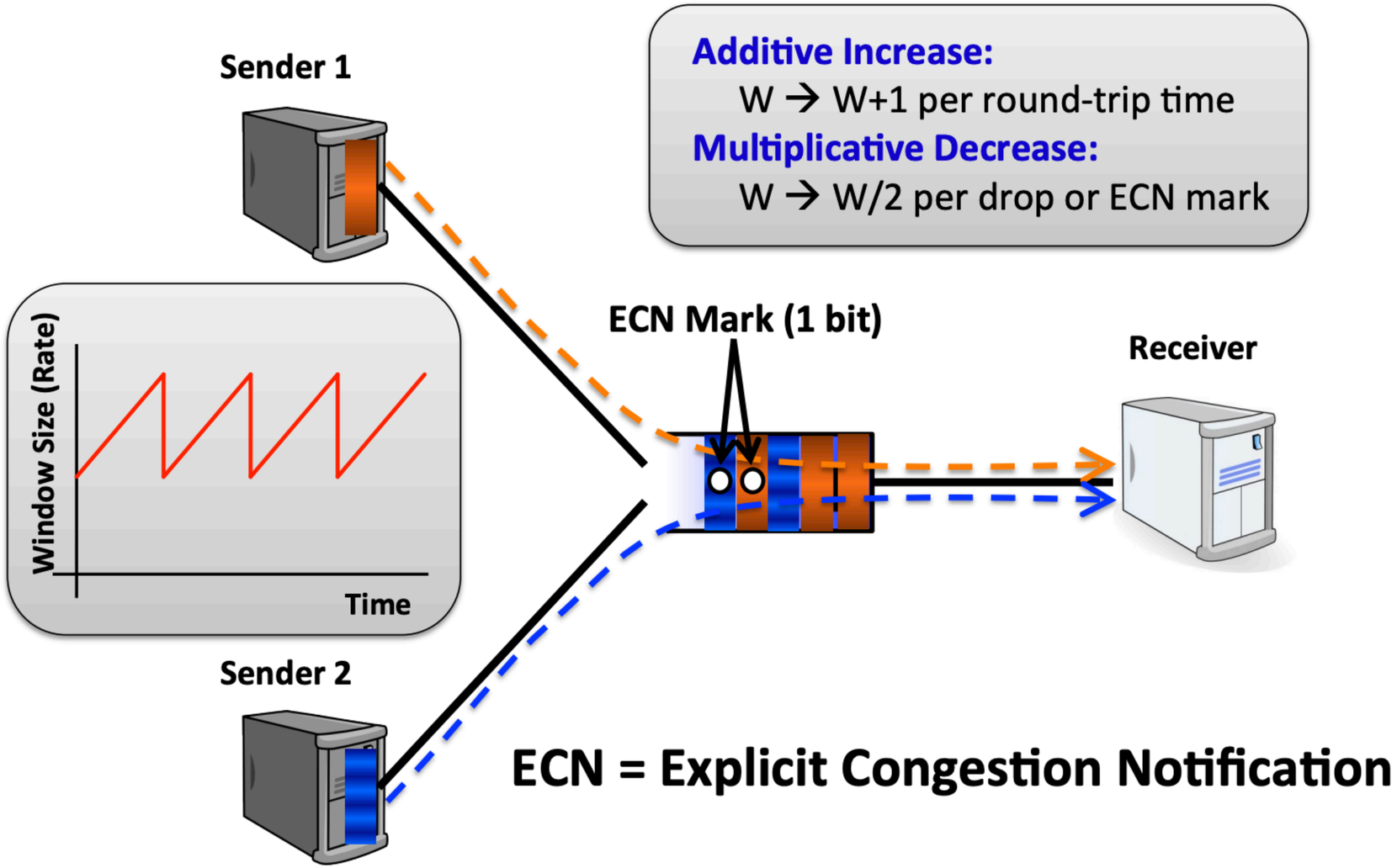
Jittering trades of median for high percentiles

DC Transport Requirements

- Low Latency
  - Short messages, queries
- High Throughput
  - Continuous data updates, backups
- High Burst Tolerance
  - Incast

The challenge is to achieve these *together*

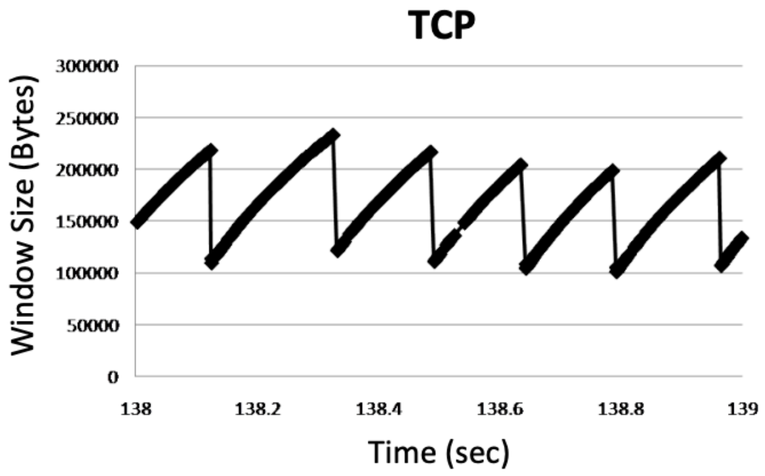
Review: TCP algorithm

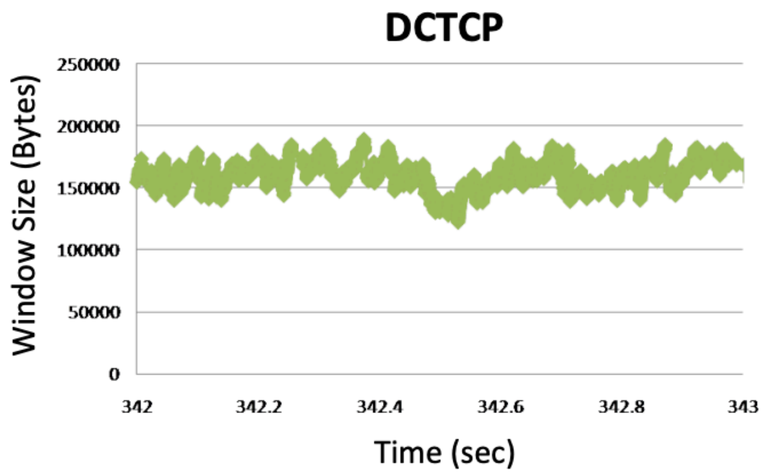


DCTCP Core Idea

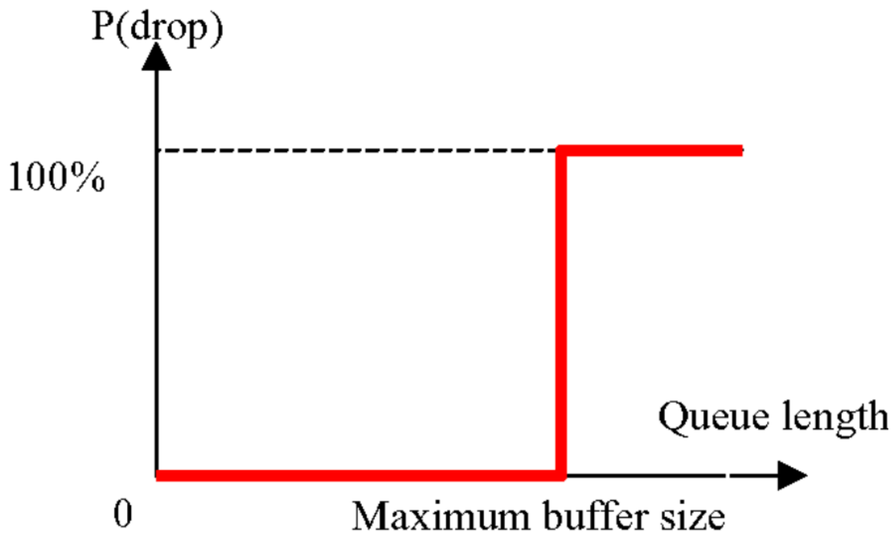
- Extract multi-bit feedback from single-bit stream of ECN marks
  - Reduce window size based on **fraction** of marked packets.

ECN Marks	TCP	DCTCP
1 0 1 1 1 1 0 1 1 1	Cut window by 50%	Cut window by 40%
0 0 0 0 0 0 0 0 0 1	Cut window by 50%	Cut window by 5%

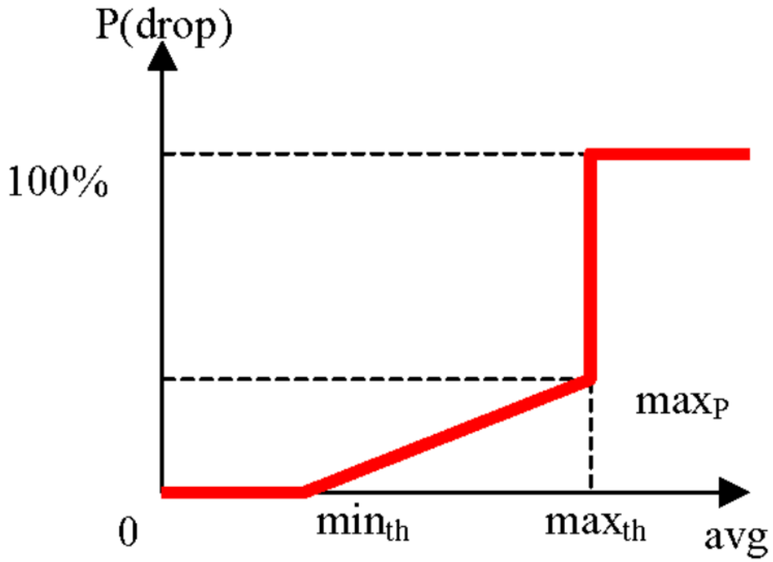




Random Early Detection (RED)



Typical Packet Drop Pattern



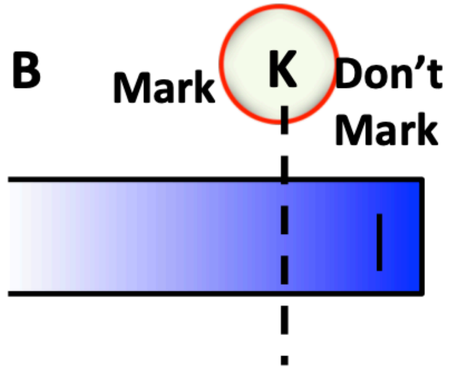
Random Early Detection (RED)
Packet Drop Pattern

RED can mark packets instead of dropping when used in conjunction with ECN

DCTCP Algorithm

Switch side:

– Mark packets when **Queue Length > K.**



Sender side:

– Maintain running average of ***fraction*** of packets marked **(α)**.

$$\text{each RTT}: \; F = \frac{\#\ \text{of marked ACKs}}{\text{Total}\ \#\ \text{of ACKs}} \quad \Rightarrow \quad \alpha \leftarrow (1-g)\alpha + gF$$

➢ **Adaptive window decreases:** $W \leftarrow (1-\frac{\alpha}{2})W$

– Note: decrease factor between 1 and 2.

DCTCP vs. TCP

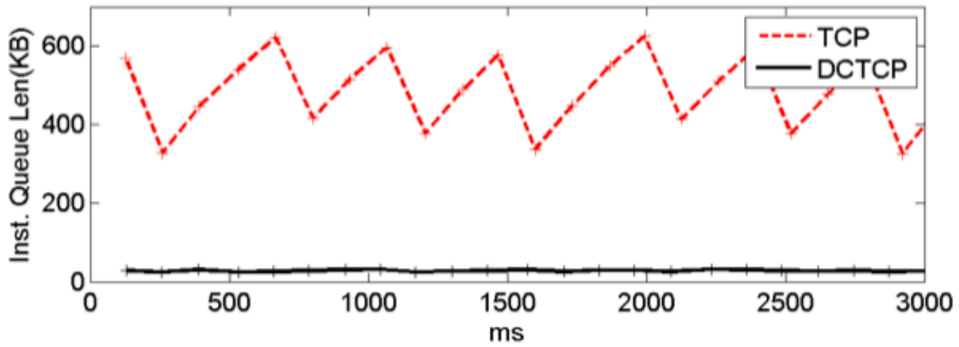
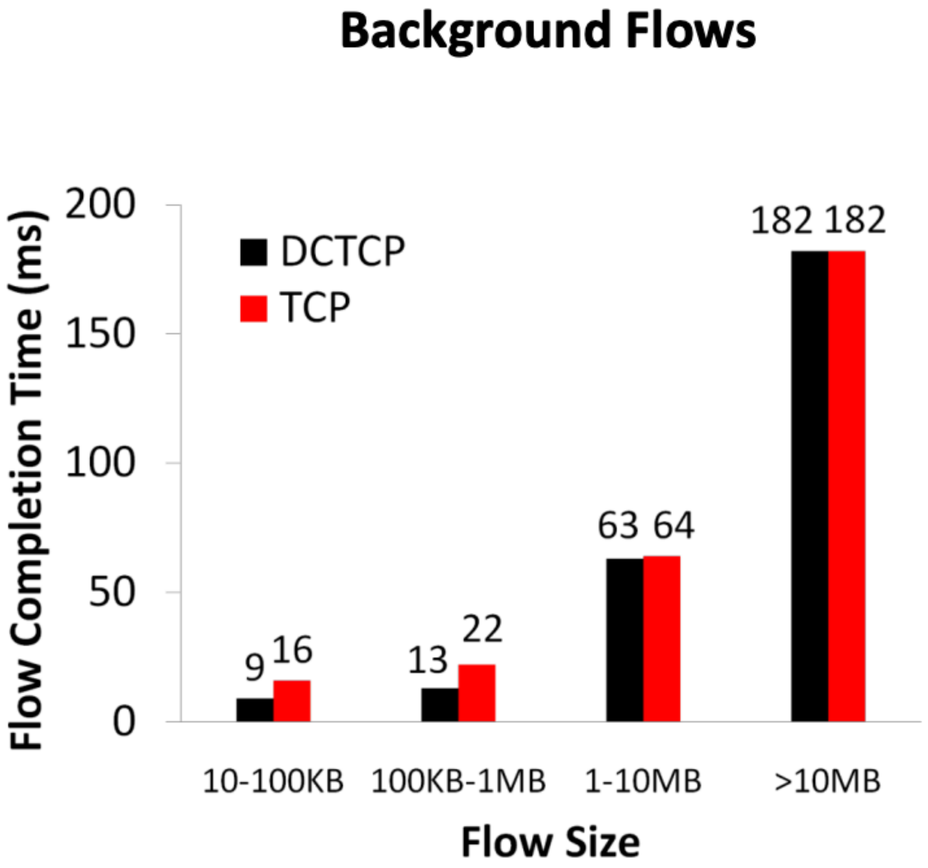
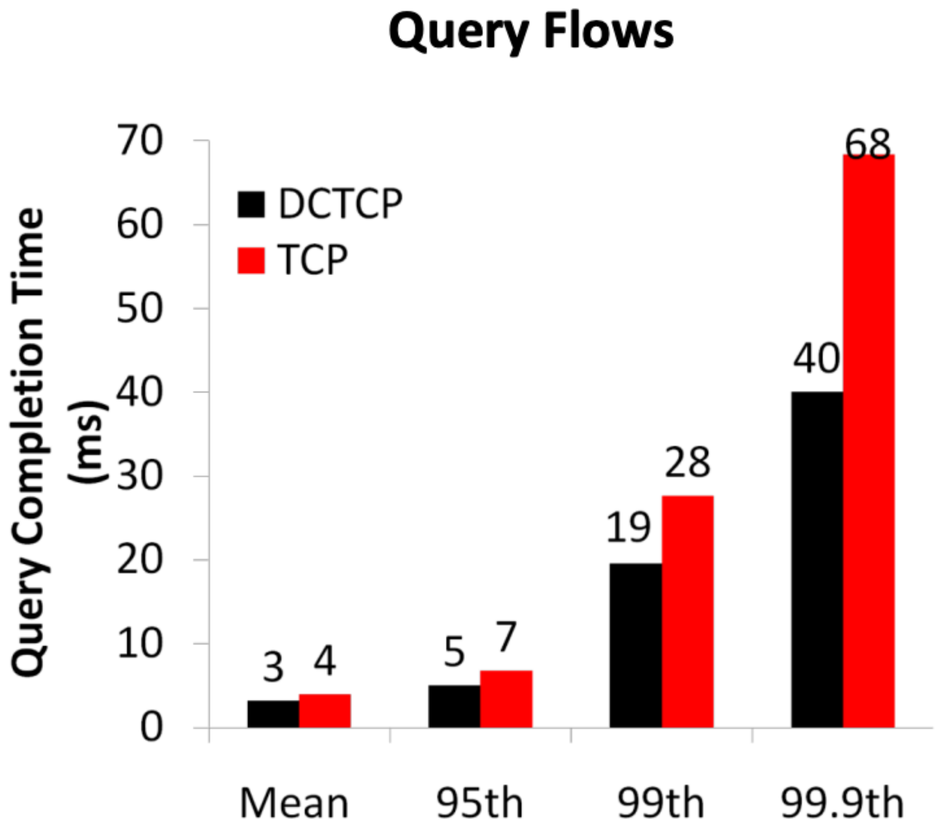


Figure 1: Queue length measured on a Broadcom Triumph switch. Two long flows are launched from distinct 1Gbps ports to a common 1Gbps port. Switch has dynamic memory management enabled, allowing flows to a common receiver to dynamically grab up to 700KB of buffer.

DCTCP Performance





Why Does DCTCP Work?

- Low Latency
  - Small buffer occupancies → low queuing delay
- High Throughput
  - ECN averaging → smooth rate adjustments, low variance
- High Burst Tolerance
  - Large buffer headroom → bursts fit
  - Aggressive marking → sources react before packets are dropped

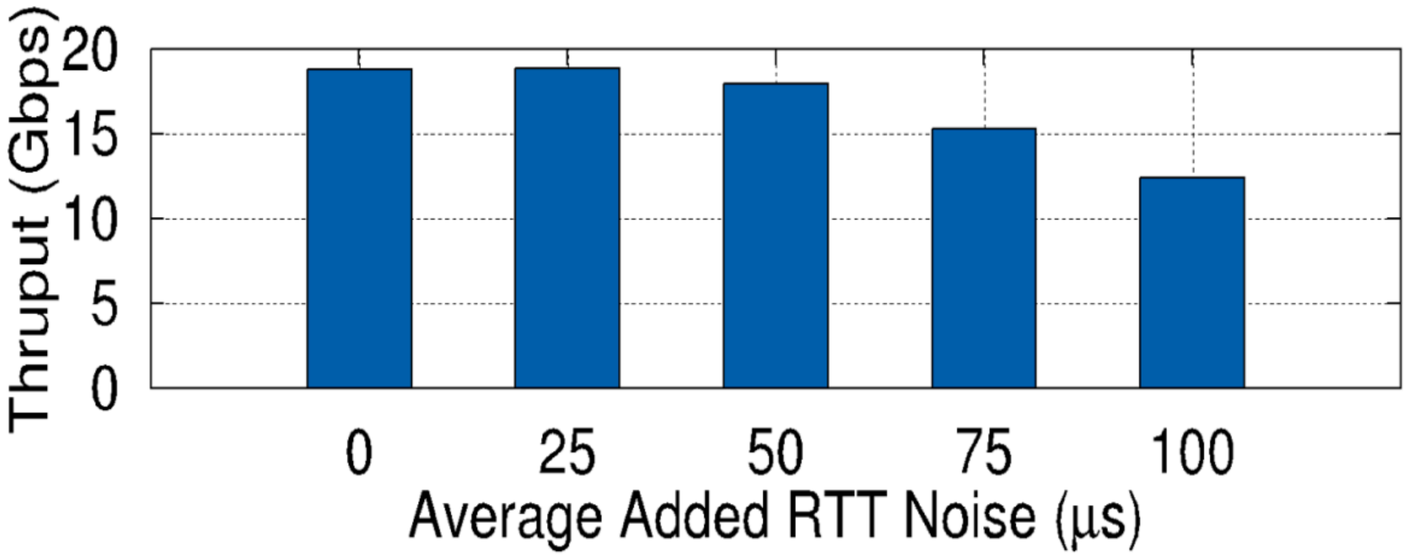
TIMELY [SIGCOMM'15]

- Core Idea
  - Packet delay, measured as round-trip times at hosts, is an effective congestion signal
- Qualities of RTT
  - Fine-grained and informative
  - Quick response time
  - No switch support needed
  - End-to-end metric

Key Challenge with RTT Measurement

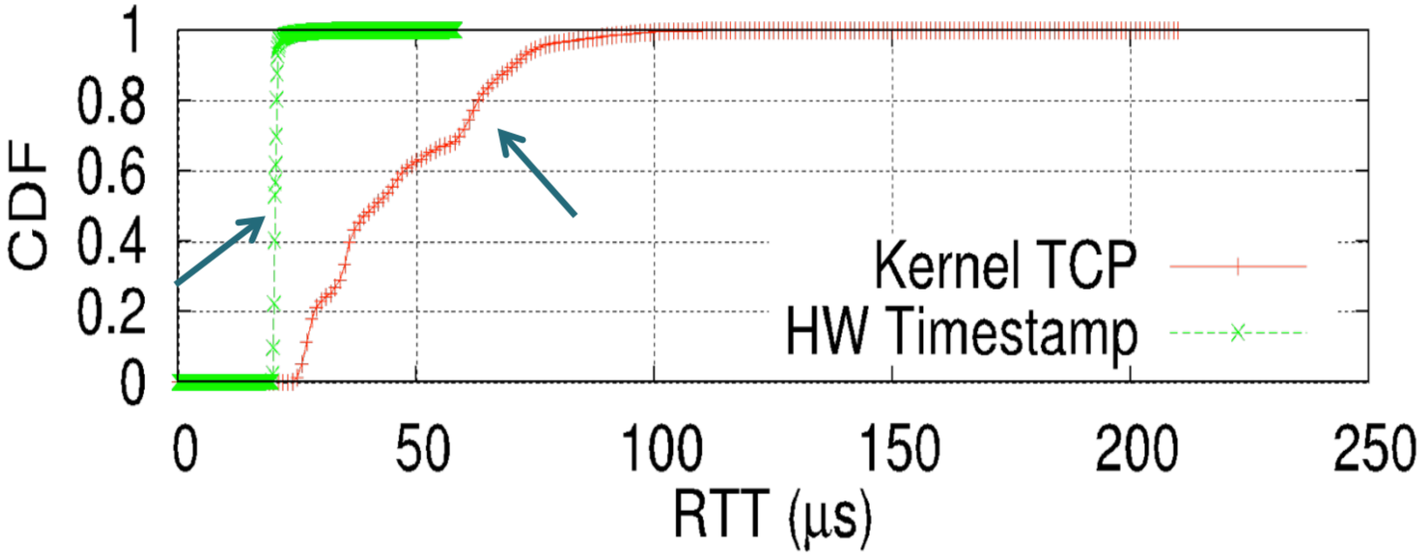
Time measurements can be noisy!

Impact of RTT noise



Throughput degrades with increasing noise in RTT.
Precise RTT measurement is crucial.

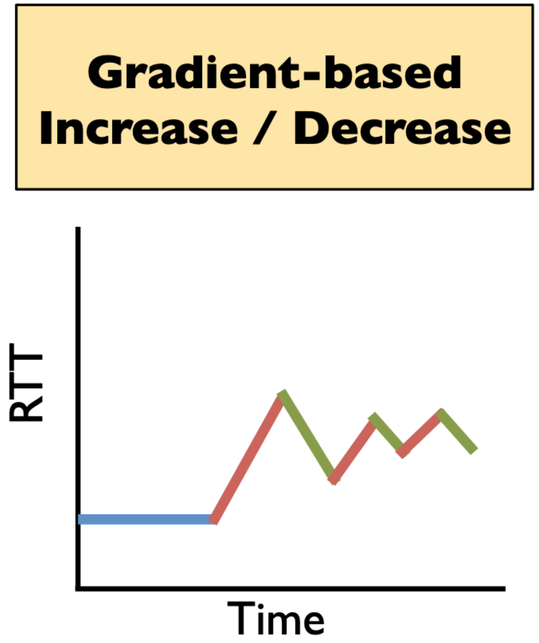
Hardware vs. Software Timestamps

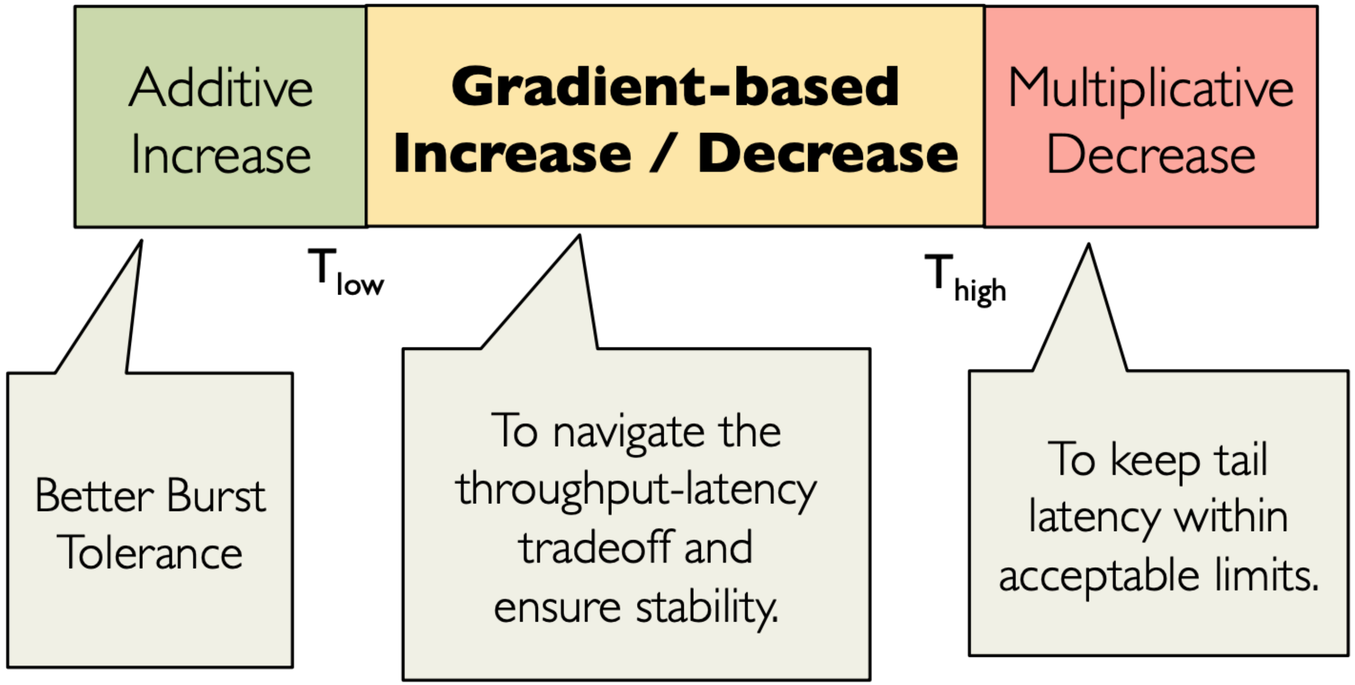


Kernel Timestamps introduce significant noise in RTT measurements compared to HW Timestamps.

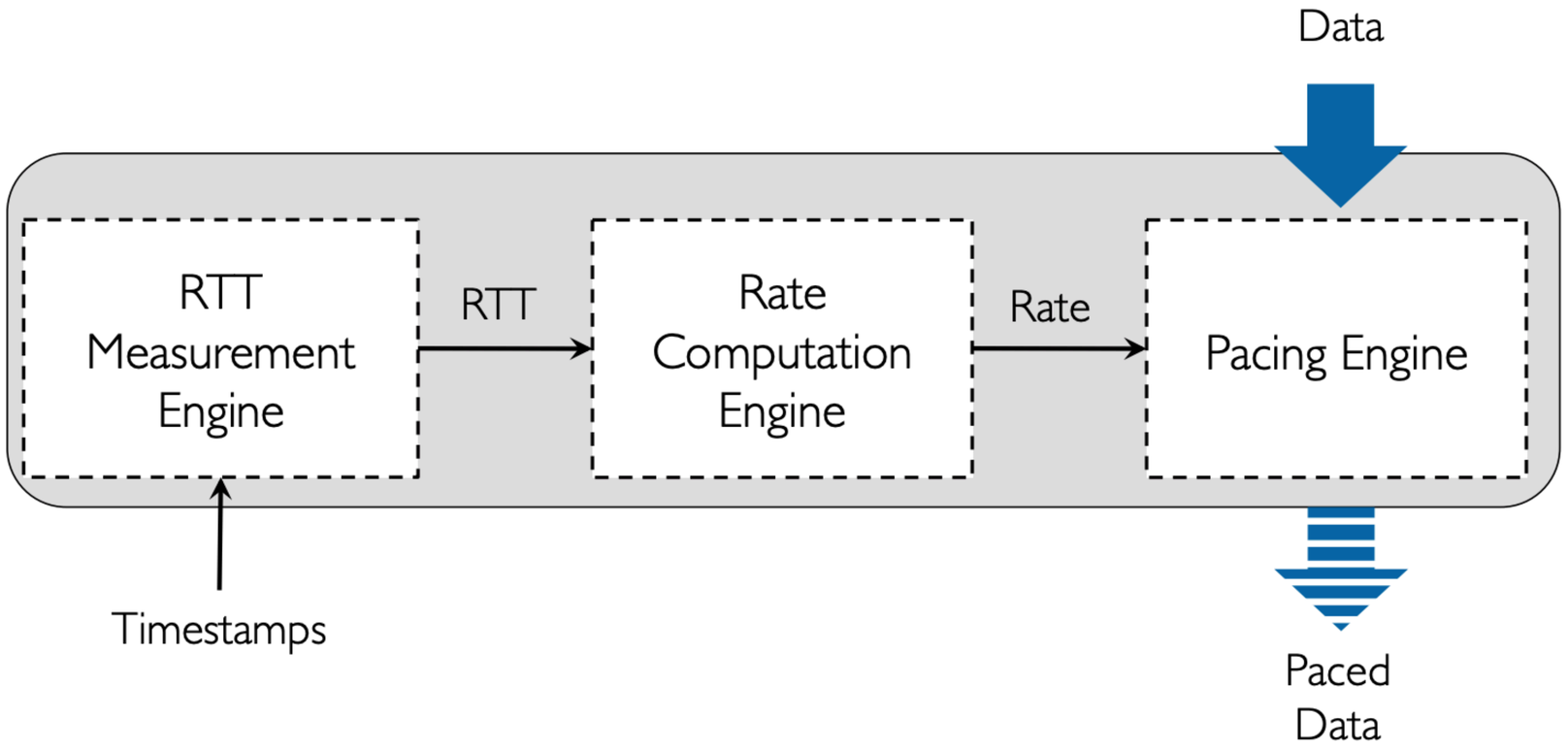
TIMELY Key Idea 1: Relies on NICs that provide hardware support for high-quality timestamping of packet events

TIMELY Key Idea 2: RTT Gradient

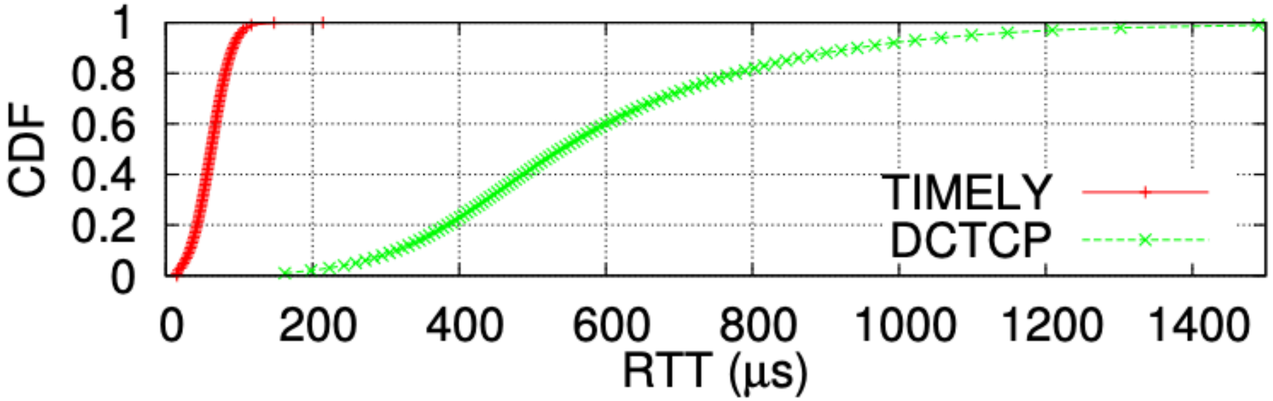




TIMELY Overview



TIMELY vs. DCTCP



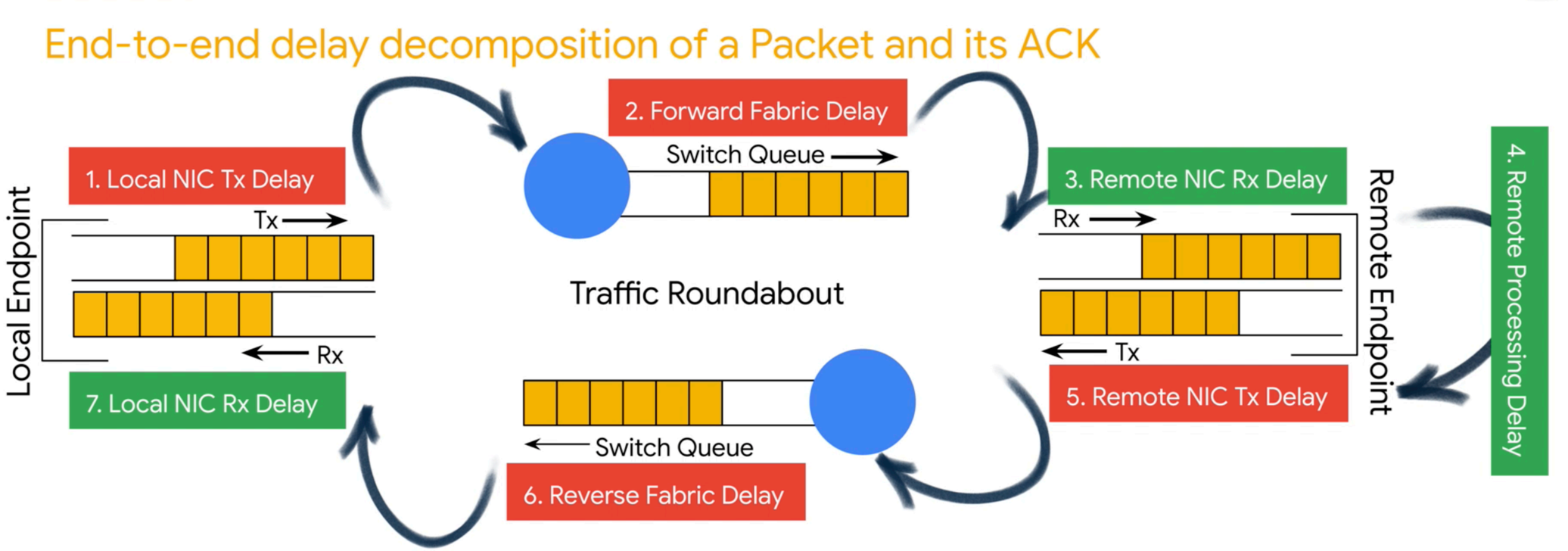
CDF of RTT distribution

! Tested on different stacks

Swift [SIGCOMM'20]

- Swift: Delay is Simple and Effective for Congestion Control in the Datacenter [SIGCOMM'20]
  - Low latency, high utilization, near-zero loss
  - 50 microseconds tail latency for short flows
  - Near 100% utilization at 100 Gbps line rate

Swift Key Idea



Which delays to respond to depends on the details of the stack

Swift maintains two congestion-windows - one based on fabric-delay and one based on endpoint-delay with their respective cwnd. Effective cwnd is the minimum of the two

Google

Fabric vs. Endpoint Delays

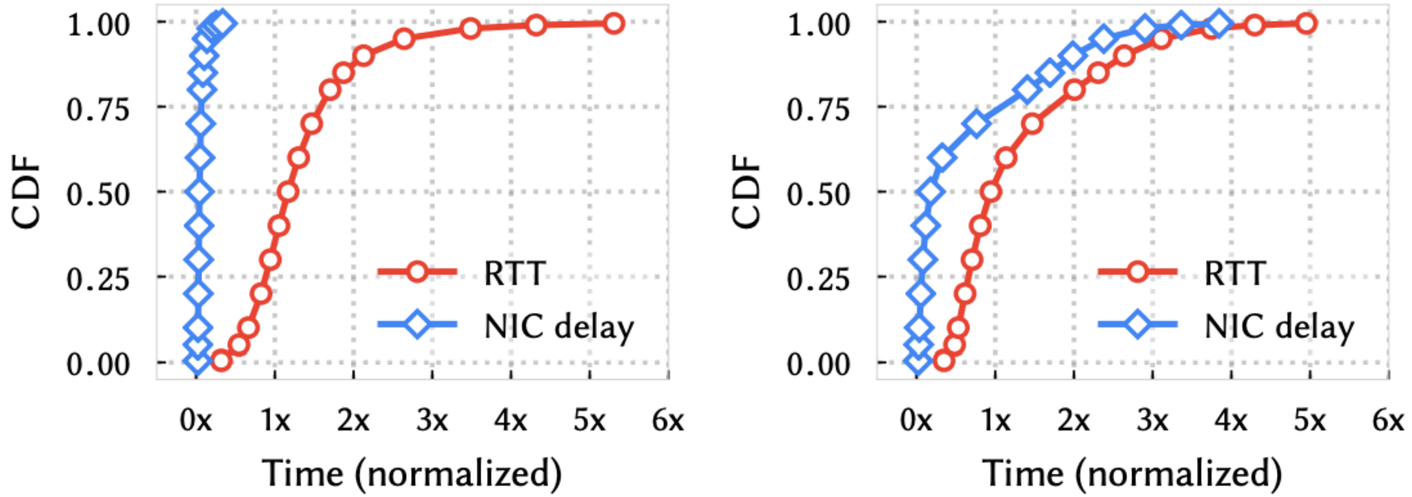


Figure 14: CDF of end-to-end packet RTT and NIC-Rx-queuing delay for the throughput-intensive cluster (left) and IOPS-intensive cluster (right).

Reminder

Title and Plan due TODAY!!

Thanks!