Congestion Control for Data Centers

Lecture 21, Computer Networks (198:552)

Fall 2019

Material adapted from slides by
Mohammad Alizadeh

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Review: TCP congestion control

- Keep some in-flight (un-ACK'ed) packets: congestion window

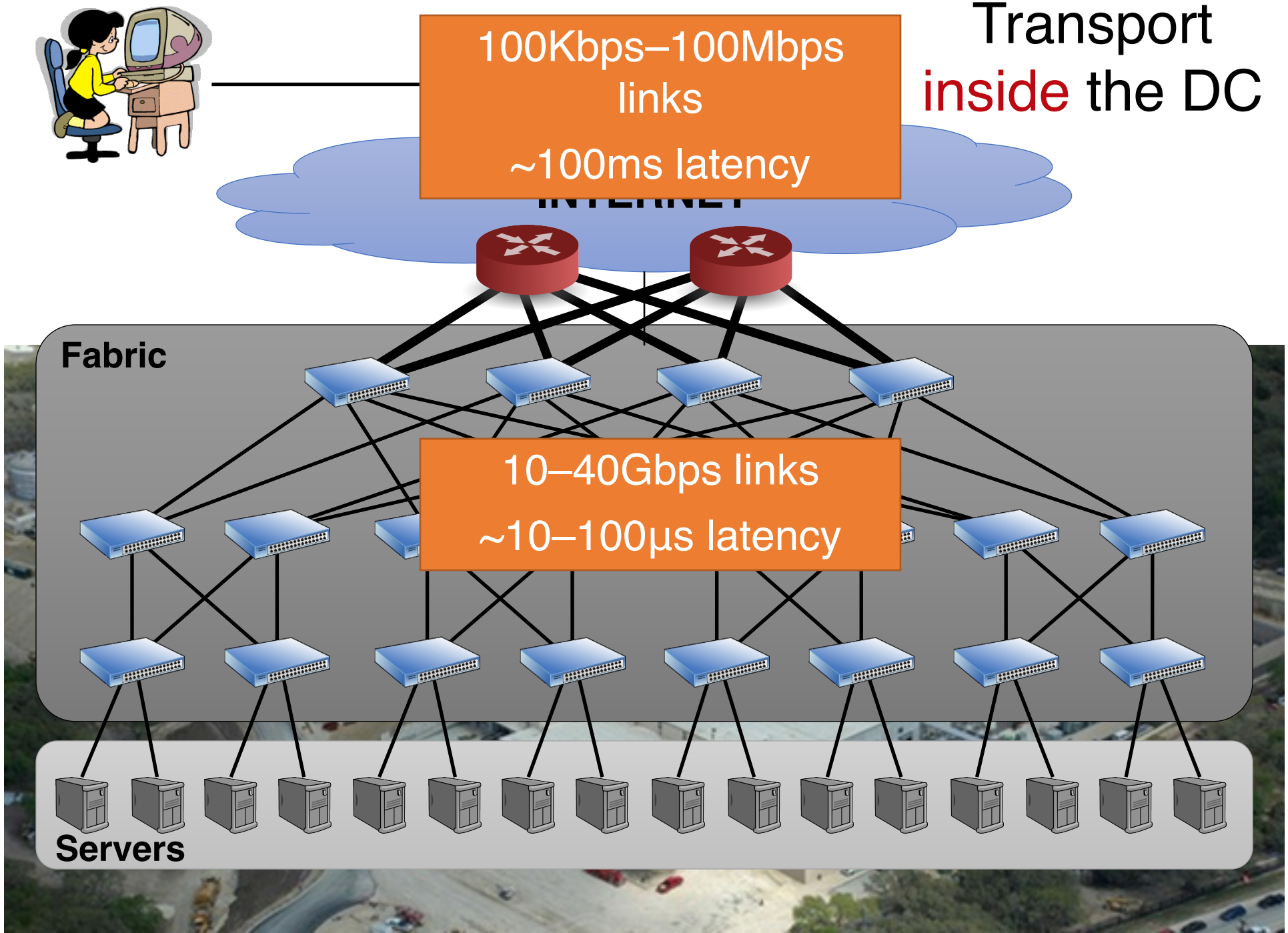
- Adjust window based on several algorithms:
  - Startup: slow start
  - Steady state: AIMD
  - Loss: fast retransmission, fast recovery

- Main question for this lecture:
  - (How) should this design change for data centers?

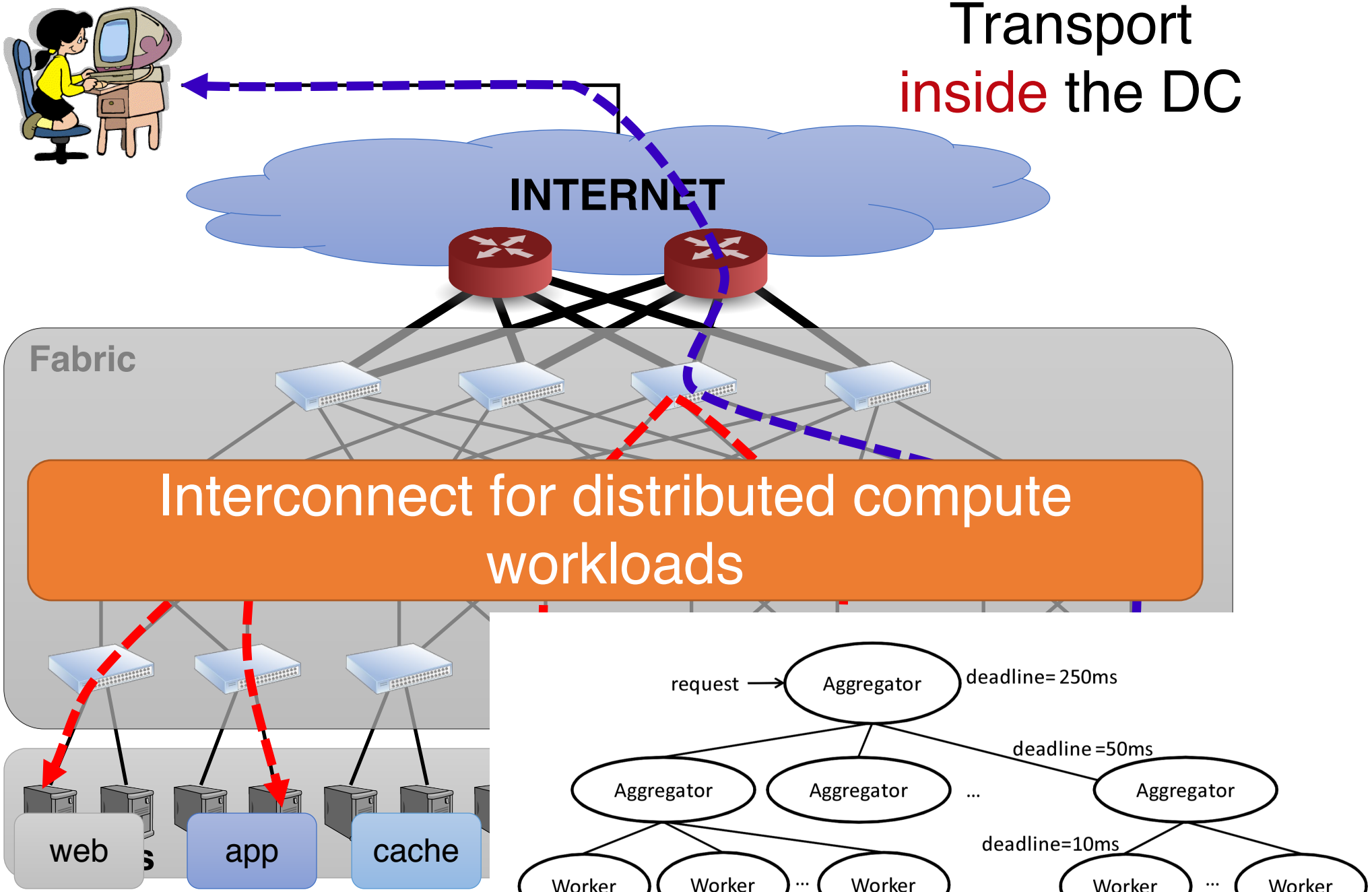
DC Transport Requirements

High throughput, low latency, burst tolerance

Transport inside the DC



Transport inside the DC

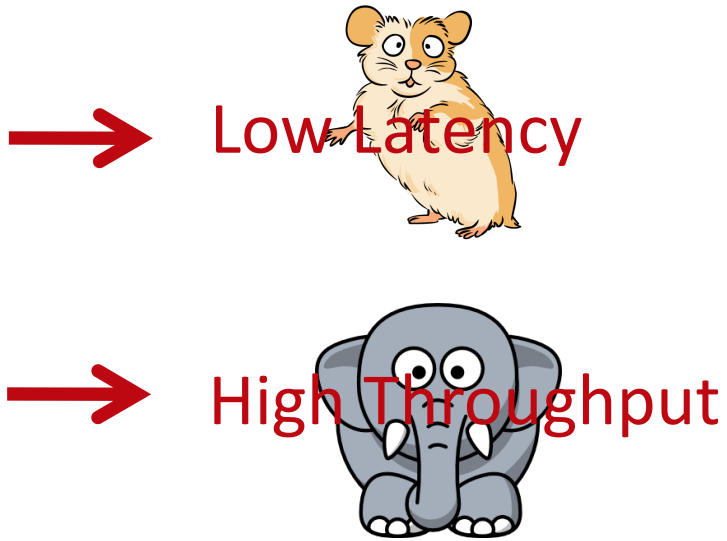


Data center workloads

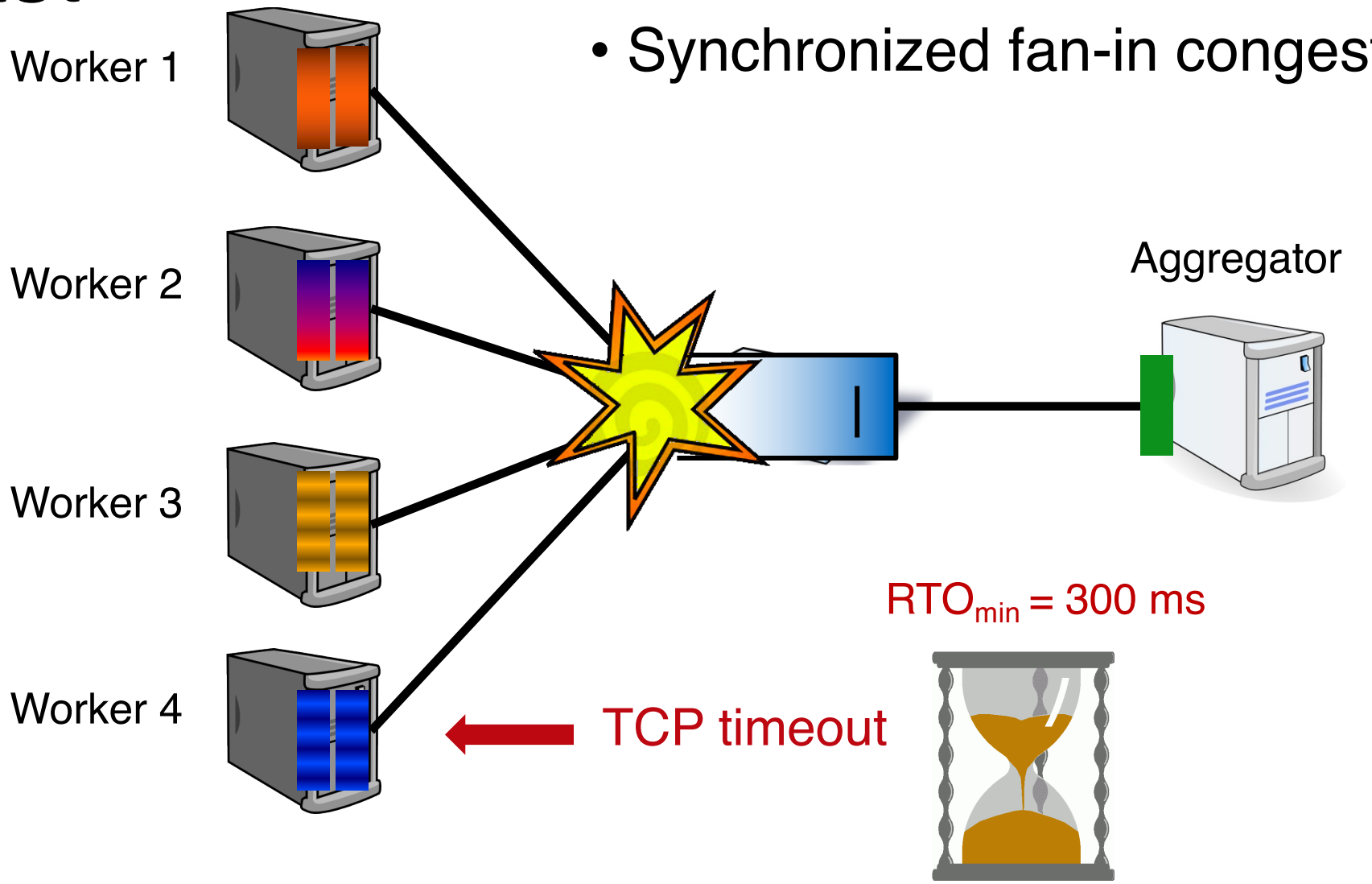
- Mice and Elephants

- Short messages
  (e.g., query, coordination)

- Large flows
  (e.g., data update, backup)

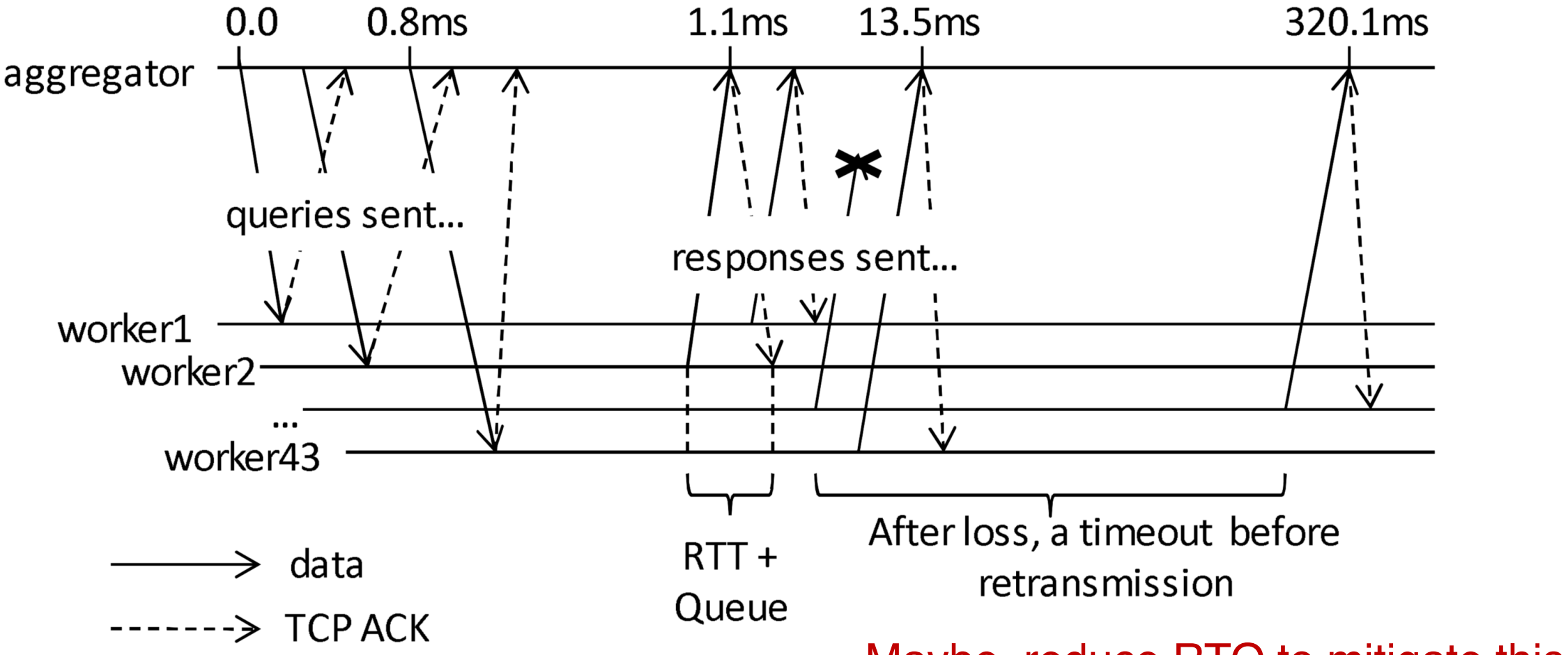


Incast



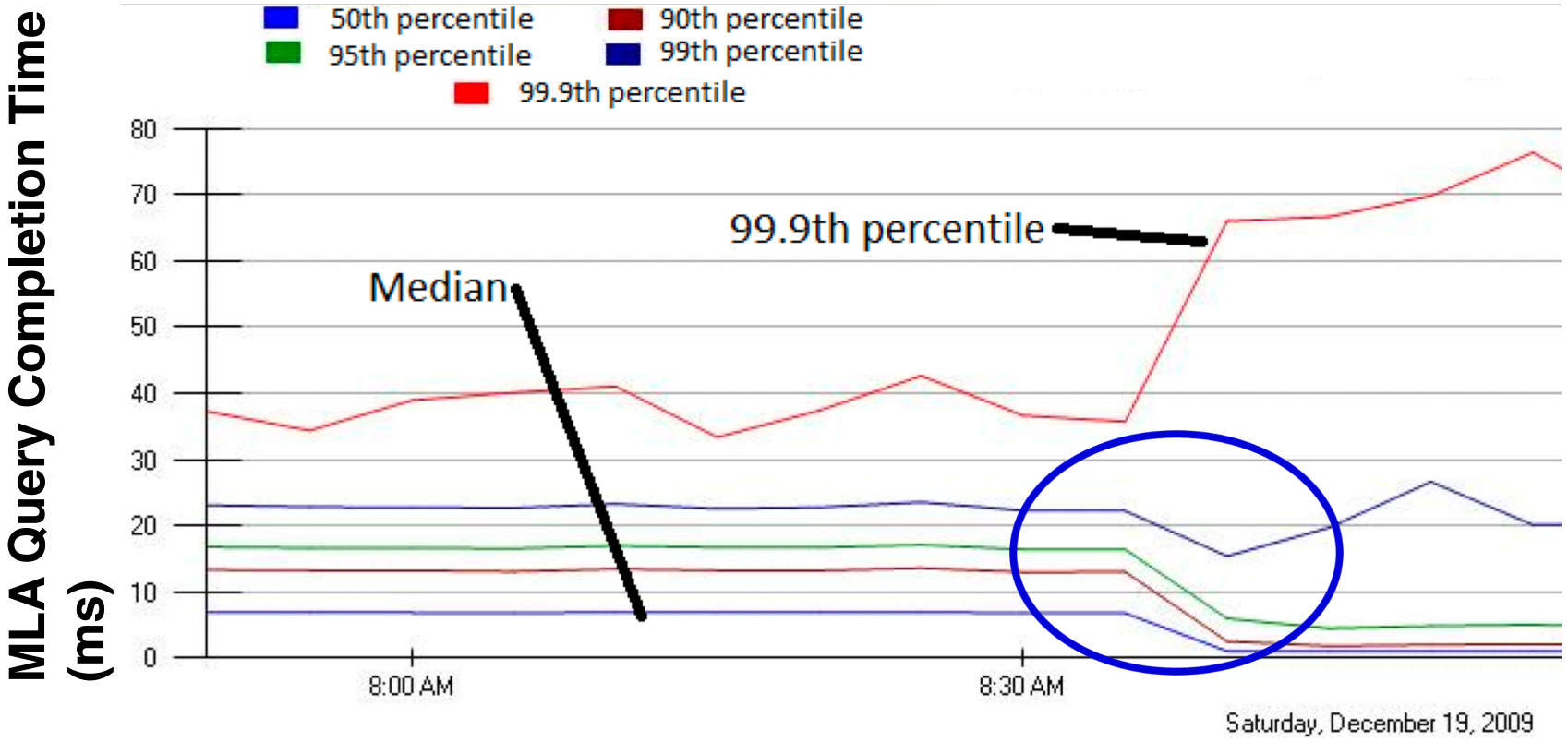
Vasudevan et al. (SIGCOMM'09)

Trace of a real incast event



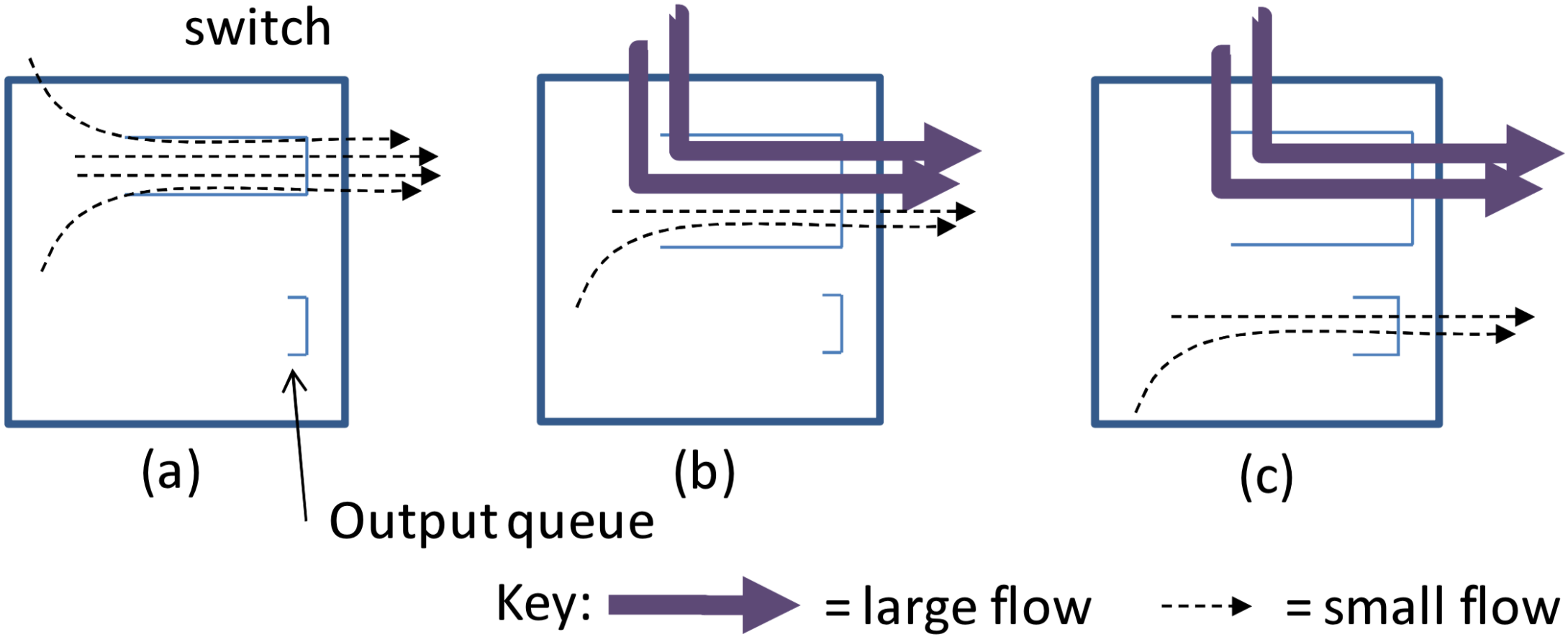
Maybe, reduce RTO to mitigate this

Jittering to mitigate incast



Jittering trades of median for high percentiles

HOL Blocking and Buffer Pressure



Queue buildup increases latency for everyone
(Reducing RTO doesn't help latency)

Another possibility: Delay-based CC

- Keep just a few packets in queues by observing delays

$$\texttt{queue\_use} = \texttt{cwnd} - \text{BWE} \times \text{RTT}_{\text{noLoad}} = \texttt{cwnd} \times (1 - \text{RTT}_{\text{noLoad}}/\text{RTT}_{\text{actual}})$$

- Adjust window such that only a few packets are in queue

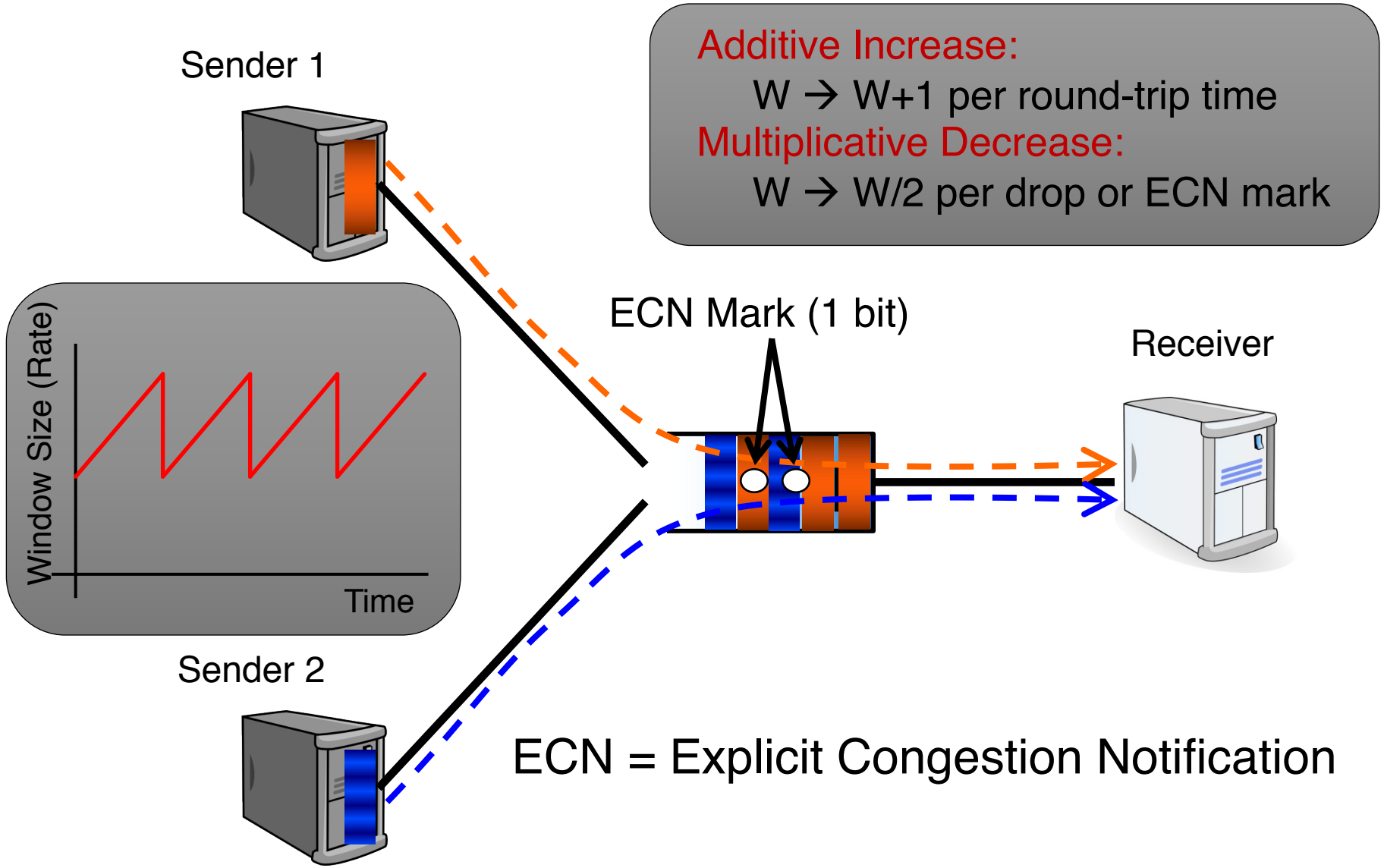
$$\alpha \leq \texttt{queue\_use} \leq \beta$$

- RTT estimates need to be very accurate and precise
  - Can be challenging in low-RTT data centers

Data Center TCP (DCTCP)

Design of the congestion control algorithm

Review: TCP algorithm

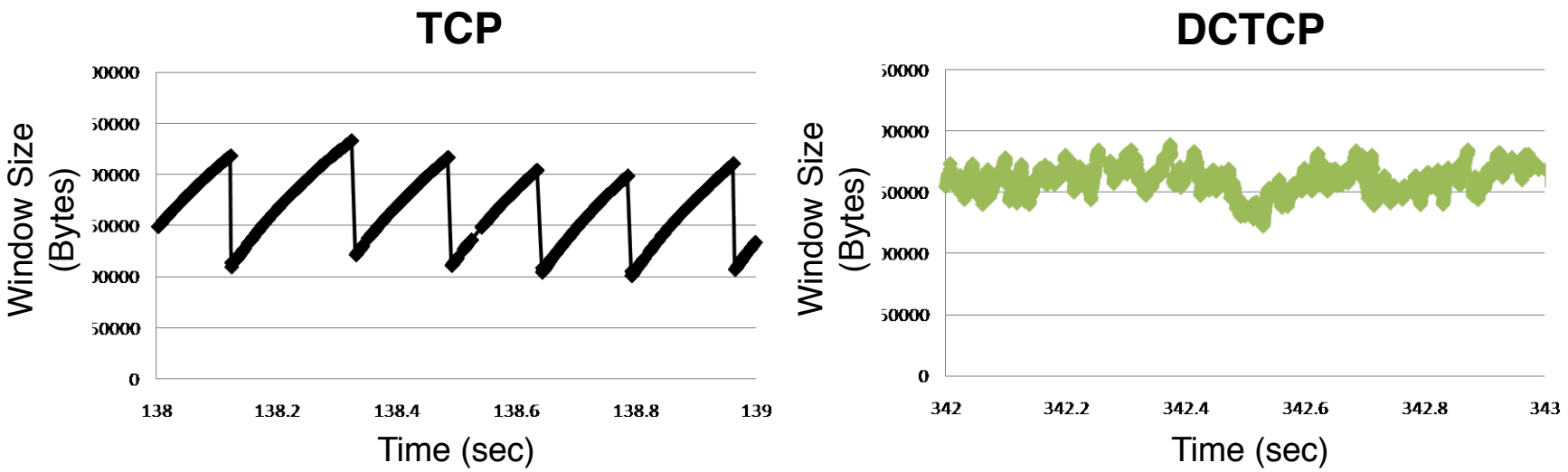


DCTCP: Main idea

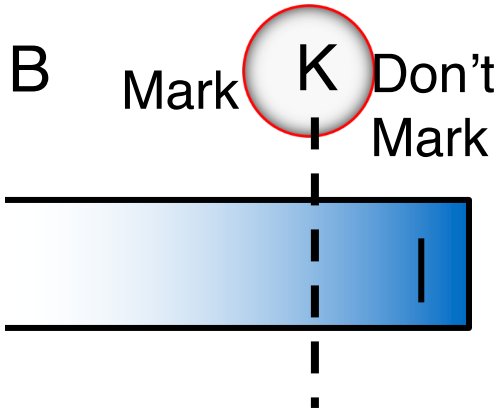
- Extract multi-bit feedback from single-bit stream of ECN marks
  - Reduce window size based on **fraction** of marked packets

DCTCP: Main idea

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%



DCTCP algorithm



Switch side:

- Mark packets when Queue Length > K.

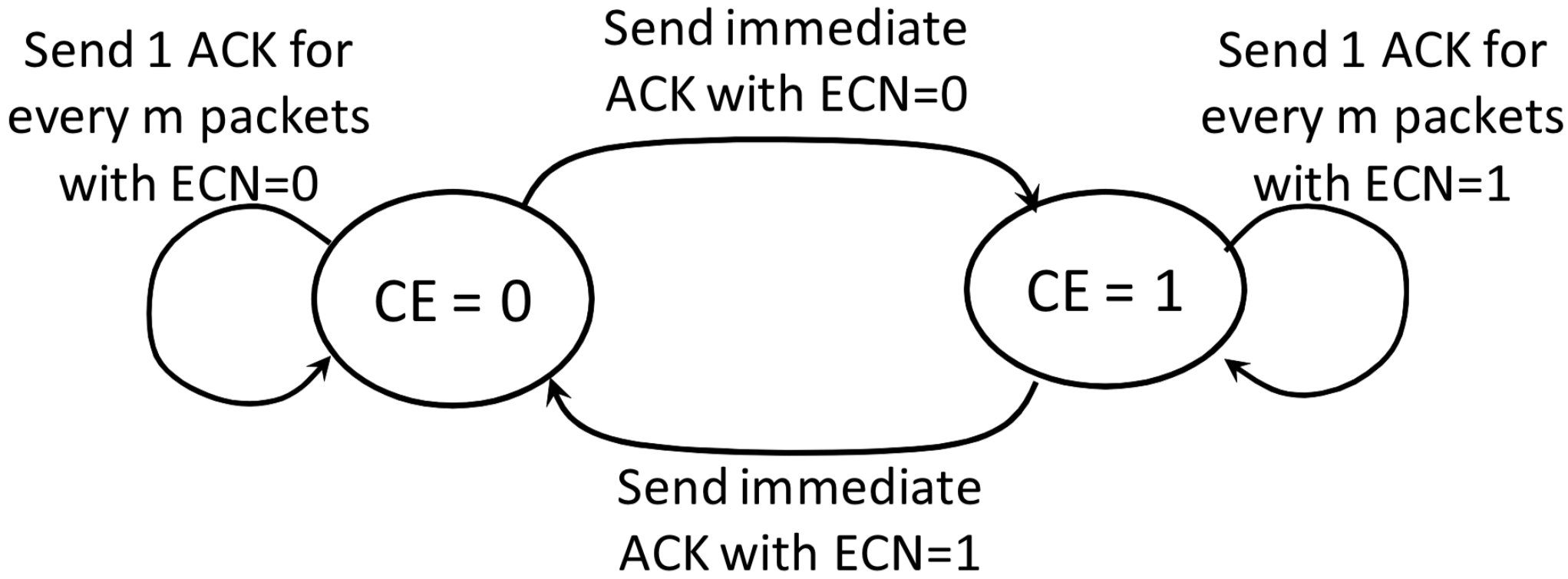
Sender side:

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

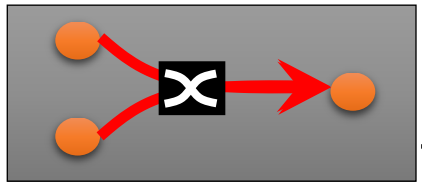
$$\text{each RTT}: \; F = \frac{\#\text{ of marked ACKs}}{\text{Total }\#\text{ of ACKs}} \quad \Rightarrow \quad \alpha \leftarrow (1-g)\alpha + gF$$

- Adaptive window decreases: $W \leftarrow (1-\frac{\alpha}{2})W$
  - Note: decrease factor between 1 and 2.

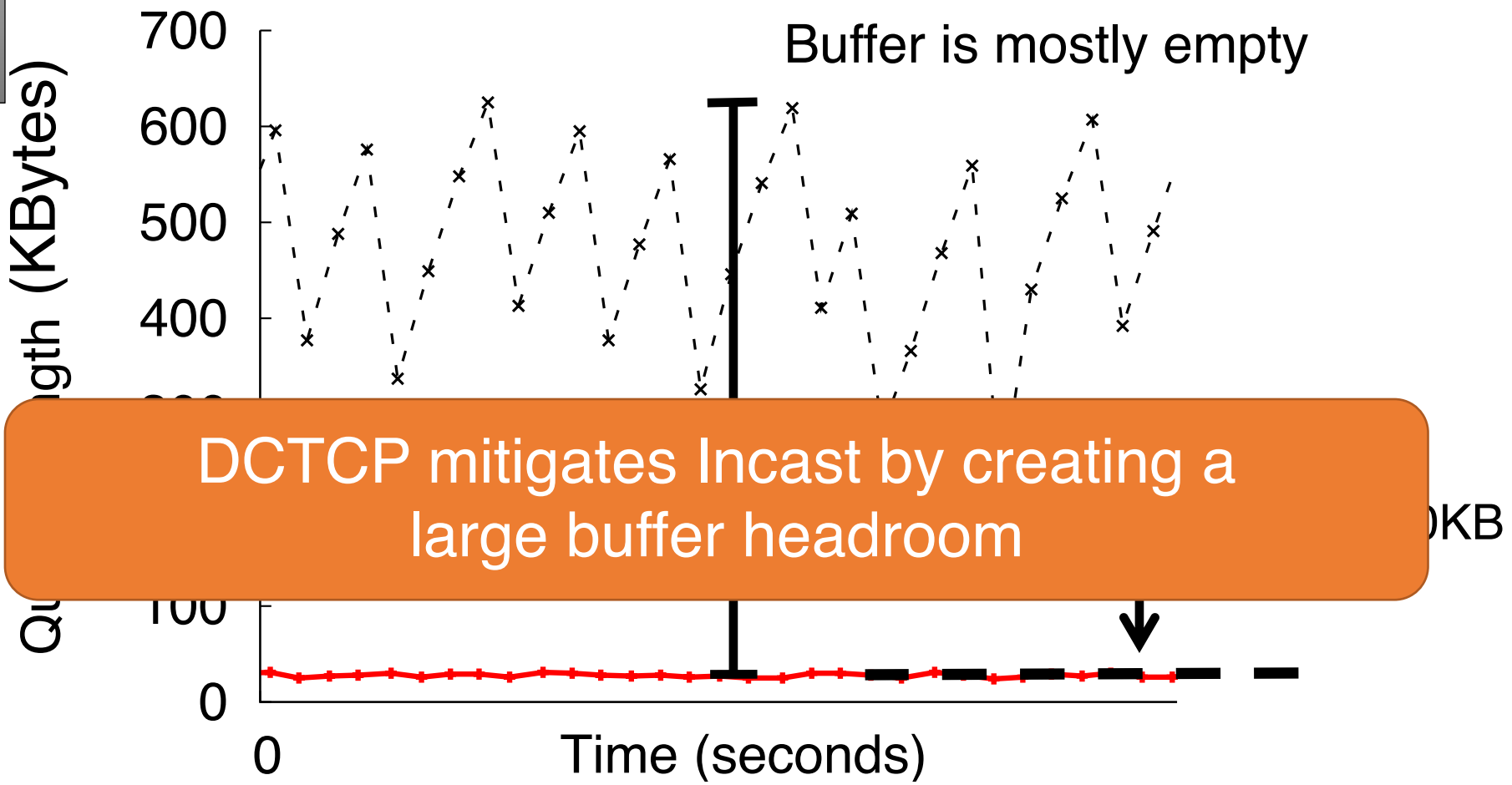
Efficient and “lossless” ACK generation



DCTCP vs TCP



Experiment: 2 flows (Win 7 stack), Broadcom 1Gbps Switch



DCTCP mitigates Incast by creating a large buffer headroom

Why it works

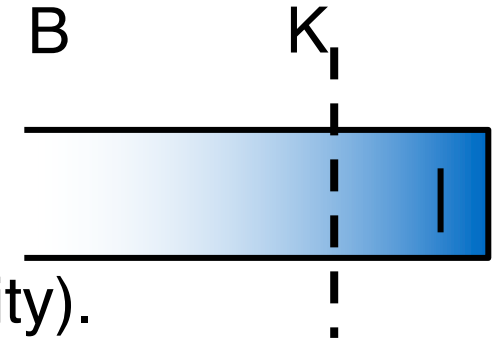
1. Low Latency
   - ✓ Small buffer occupancies → low queuing delay

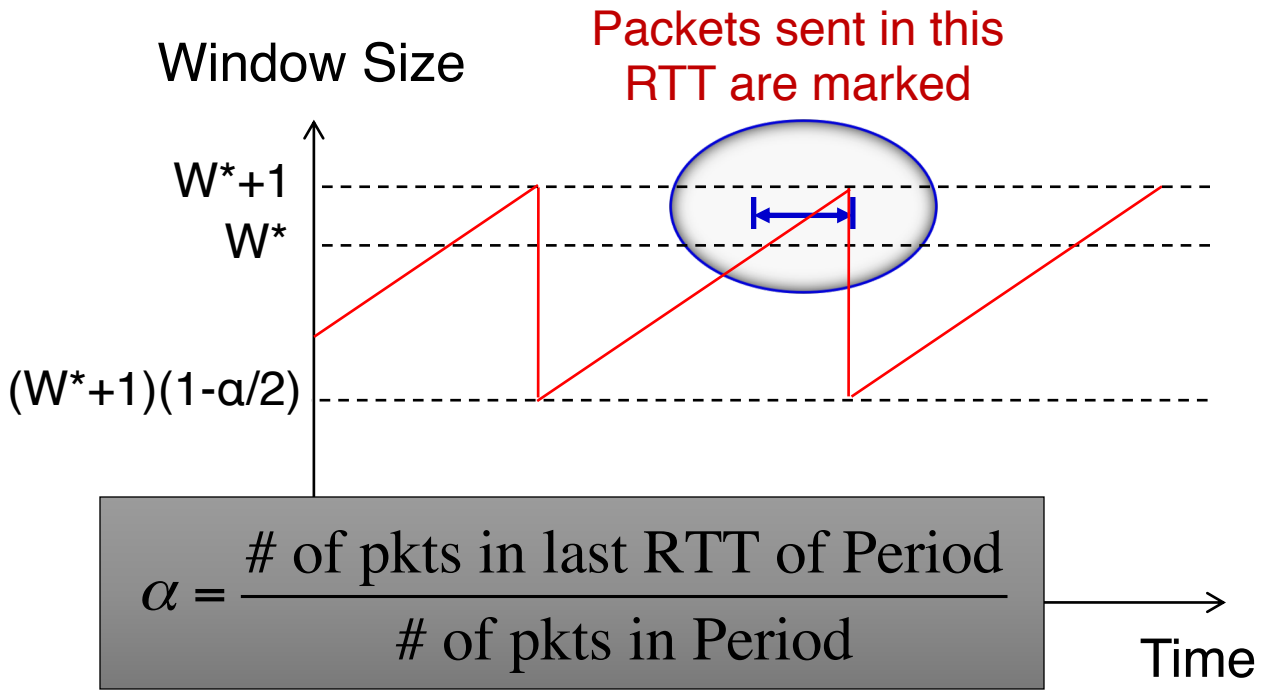
2. High Throughput
   - ✓ ECN averaging → smooth rate adjustments, low variance

3. High Burst Tolerance
   - ✓ Large buffer headroom → bursts fit
   - ✓ Aggressive marking → sources react before packets are dropped

Setting parameters: A bit of analysis

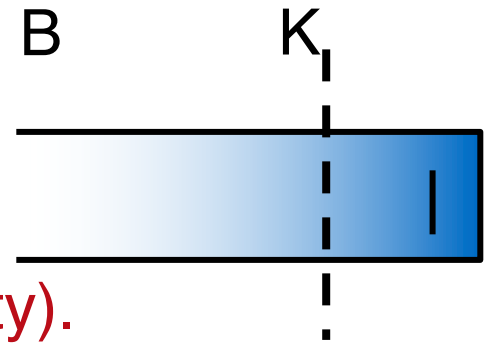
- How much buffering does DCTCP need for 100% throughput?
  - Need to quantify queue size oscillations (stability).





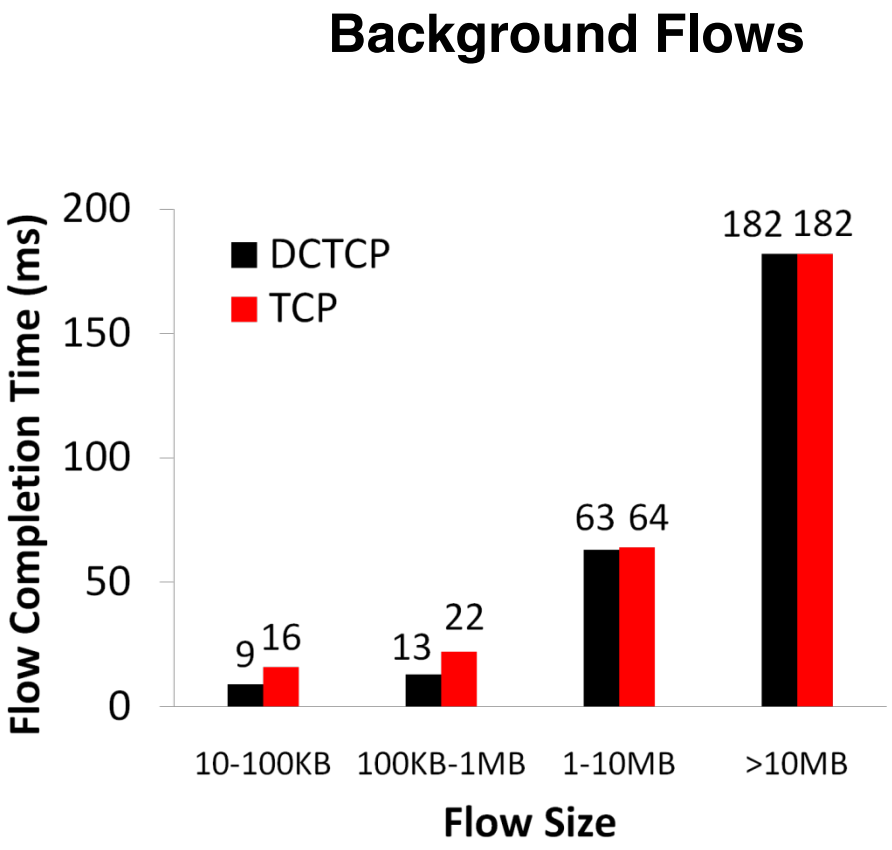
Setting parameters: A bit of analysis

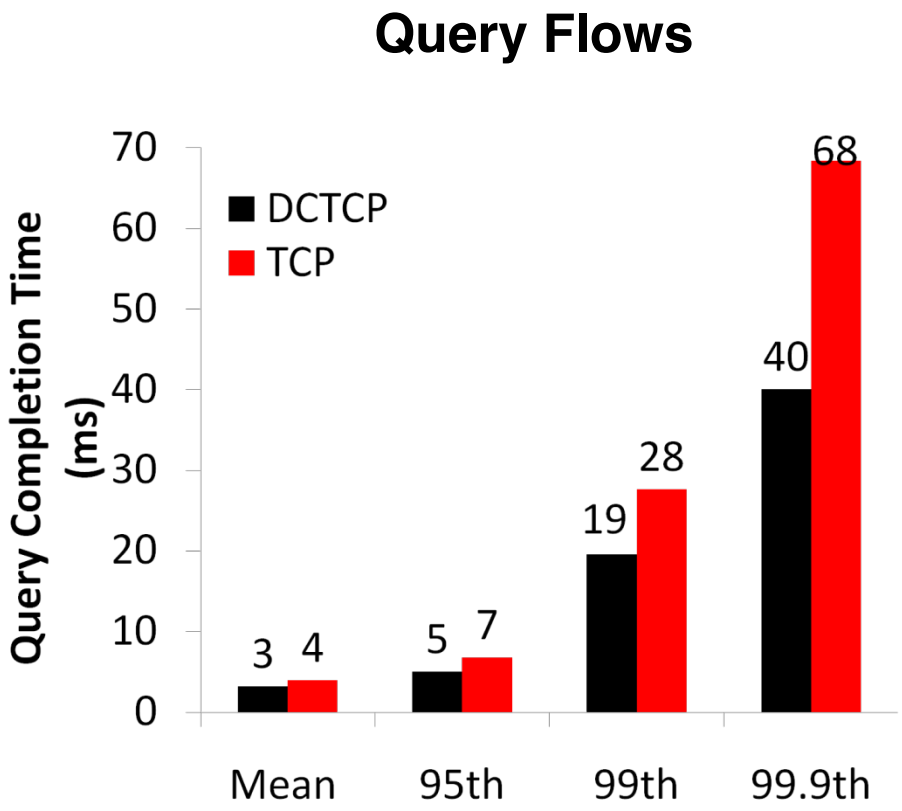
- How small can queues be without loss of throughput?
  - ➢ Need to quantify queue size oscillations (Stability).



K > (1/7) C x RTT ⟷ for TCP: K > C x RTT

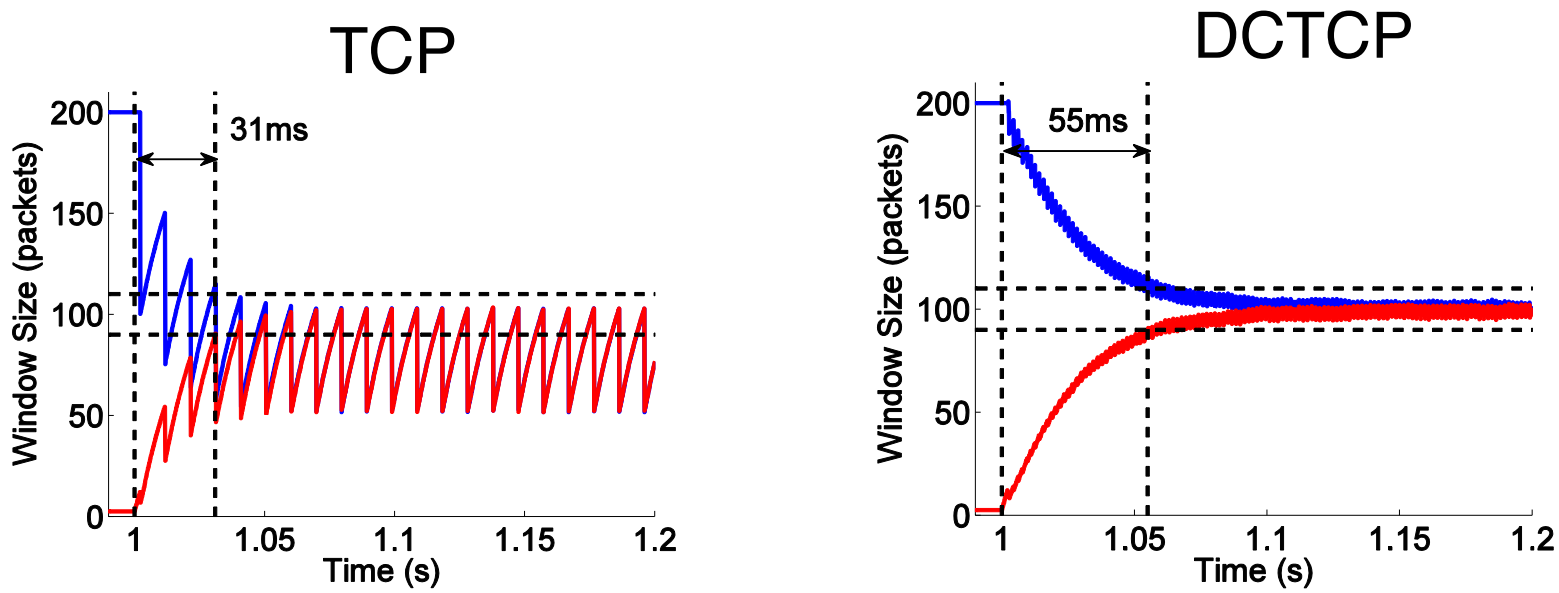
Bing benchmark (baseline)





Convergence time

- DCTCP takes at most ~40% more RTTs than TCP
  - "Analysis of DCTCP", SIGMETRICS 2011
- Intuition: DCTCP makes smaller adjustments than TCP, but makes them much more frequently



CC evaluation: several aspects!

- Throughput, delays, flow completion times
- Fairness, convergence times
- Specific impairments:
  - incast (many to one, all to all)
  - collateral damage from incast
  - buffer pressure
- Impact on background traffic
- Multi-hop versus single-hop bottlenecks

CC Deployment Concerns

Life ain’t easy in the fast lane

Practical deployment concerns in DCs

- Coexistence with legacy protocols like TCP Cubic
  - Application code can’t be upgraded in one shot
- Minimum window size matters during heavy incast events
  - e.g., 2 packets versus 1 packet!
- Setting pkt flags appropriately at senders, receivers, and routers
  - Non “ECN-capable” flagged packets will be dropped when Q > K
  - … including the SYN packets of any connection
- Receive-buffer tuning
  - Receive buffer must be at least BDP, but what is the BDP?