# Data Center TCP (DCTCP)


Presented by Zeyuan Hu

# Paper Contributions

- Measure and analyze production traffic
- Data Center TCP (DCTCP)

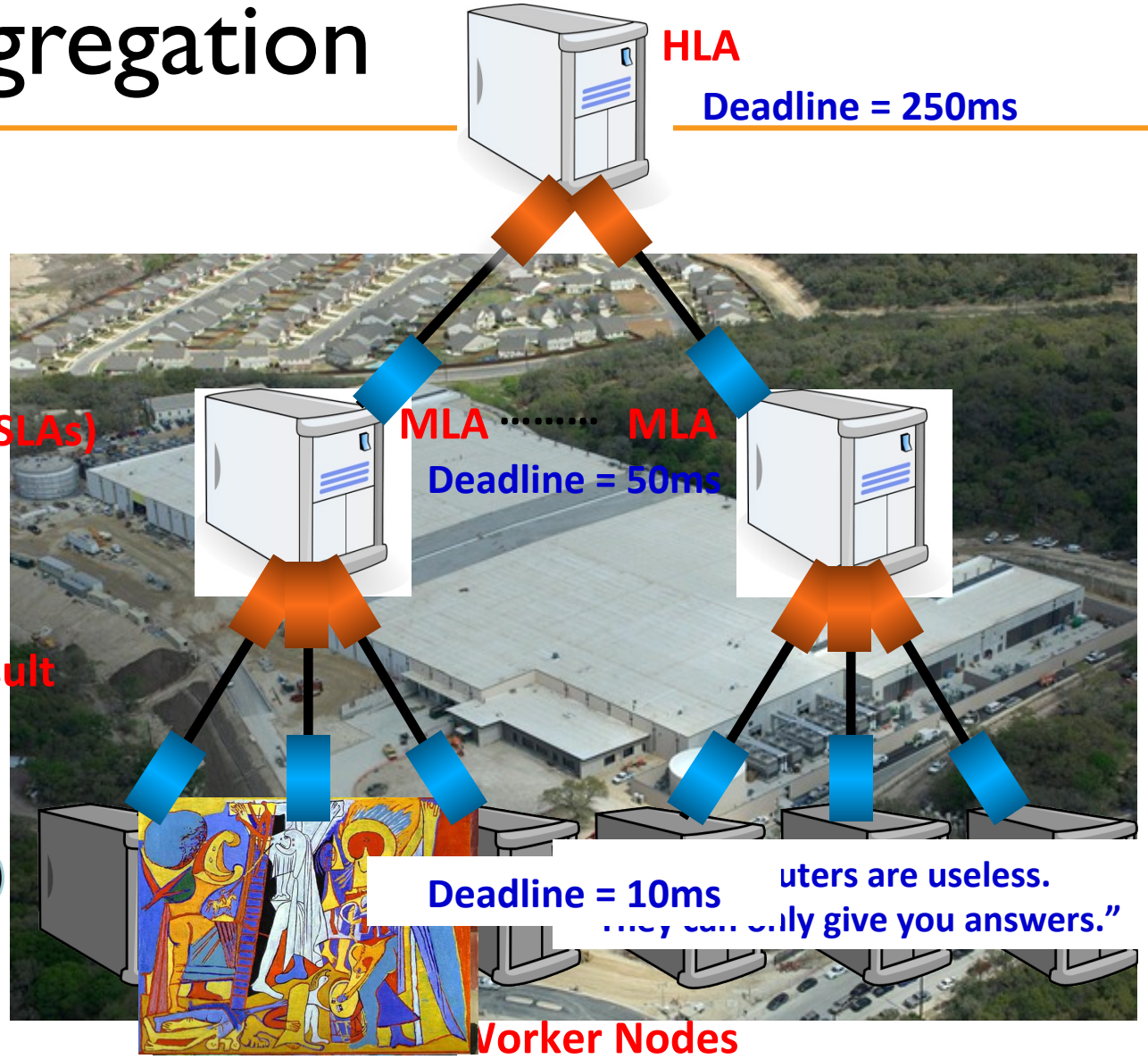
# Measure and analyze production traffic

# Measure and analyze production traffic

- **Case study: Microsoft Bing**
  - Measurements from 6000 server production cluster
  - passively collects socket level logs, selected packet-level logs, and app-level logs describing latencies
  - More than 150TB of compressed data over a month
- **99.91% of traffic in the Microsoft data center is TCP traffic**
- **Workloads**
  - Partition/Aggregate [2KB – 20KB] (Query) → delay-sensitive
  - Short messages [50KB – 1MB] (Coordination, Control state) → delay-sensitive
  - Large flows [1MB – 50MB] (Data update) → throughput-sensitive

# Partition/Aggregation

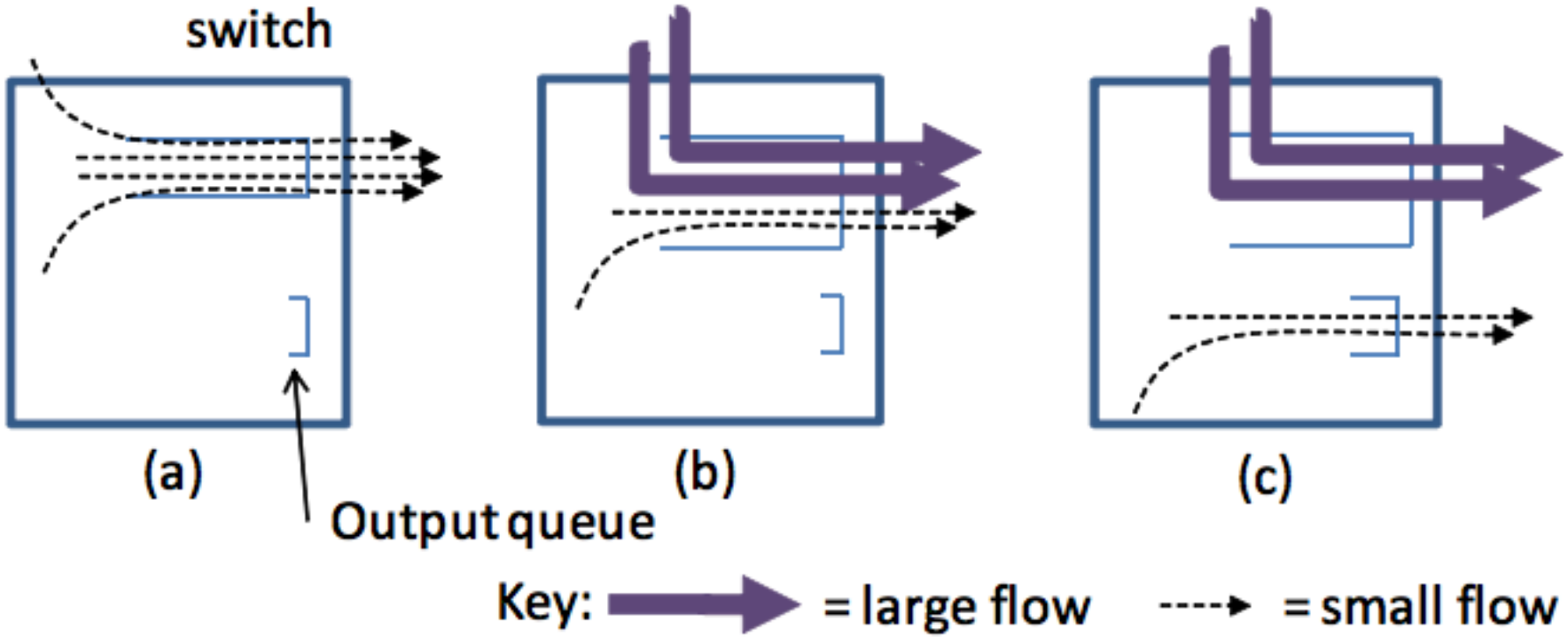
- Time is money
  - ➢ **Strict deadlines (SLAs)**
- Missed deadline
  - ➢ **Lower quality result**

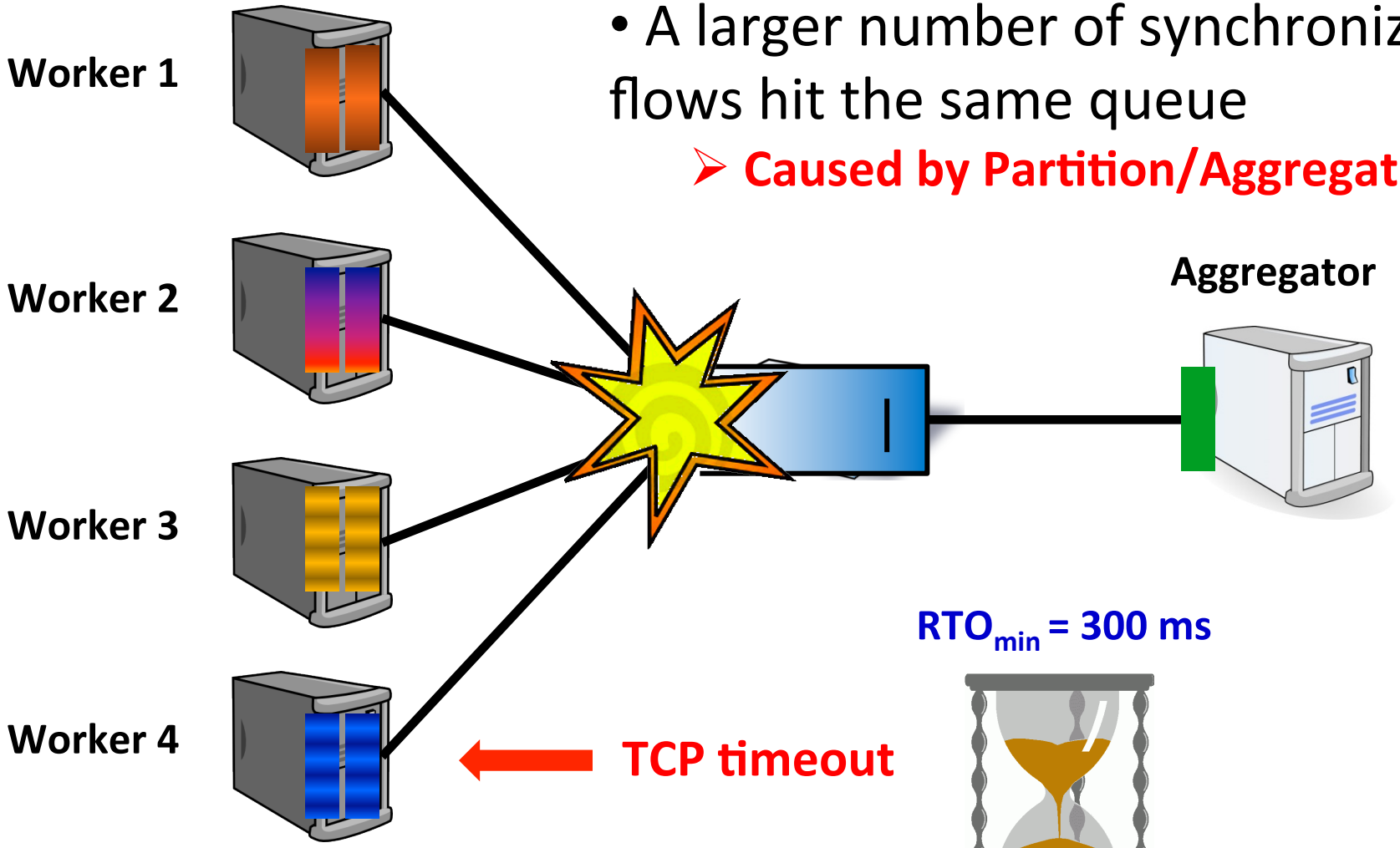

Adapted from Balaji Prabhakar, Stanford University

# Part of Findings - Three Impairments

Incast Queue buildup Buffer pressure



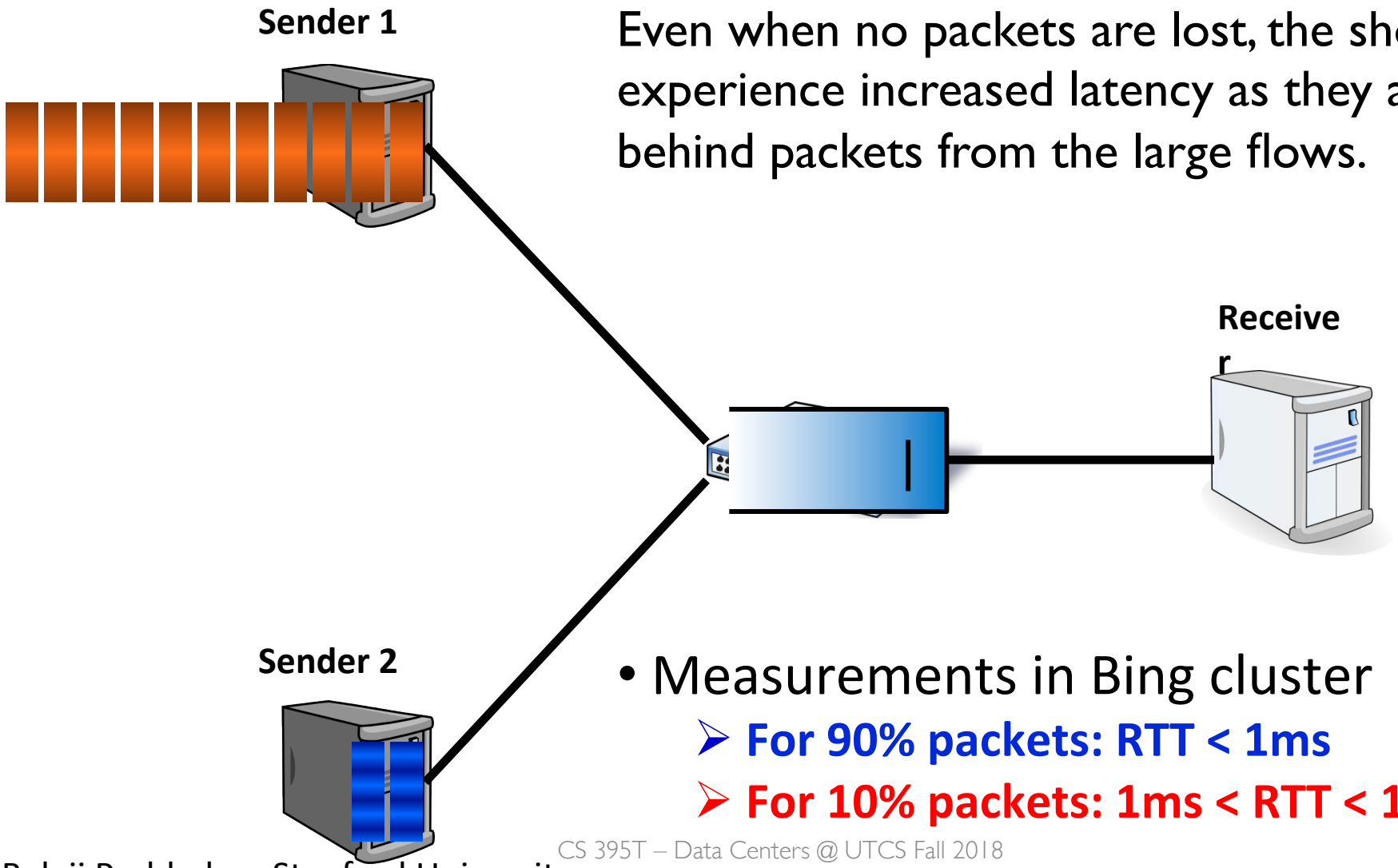
# Incast




- A larger number of synchronized small flows hit the same queue
  - ➢ **Caused by Partition/Aggregate.**

Adapted from Balaji Prabhakar, Stanford University

# Queue Buildup




Even when no packets are lost, the short flows experience increased latency as they are in queue behind packets from the large flows.

- Measurements in Bing cluster
  - **For 90% packets: RTT < 1ms**
  - **For 10% packets: 1ms < RTT < 15ms**

Adapted from Balaji Prabhakar, Stanford University

# Buffer Pressure

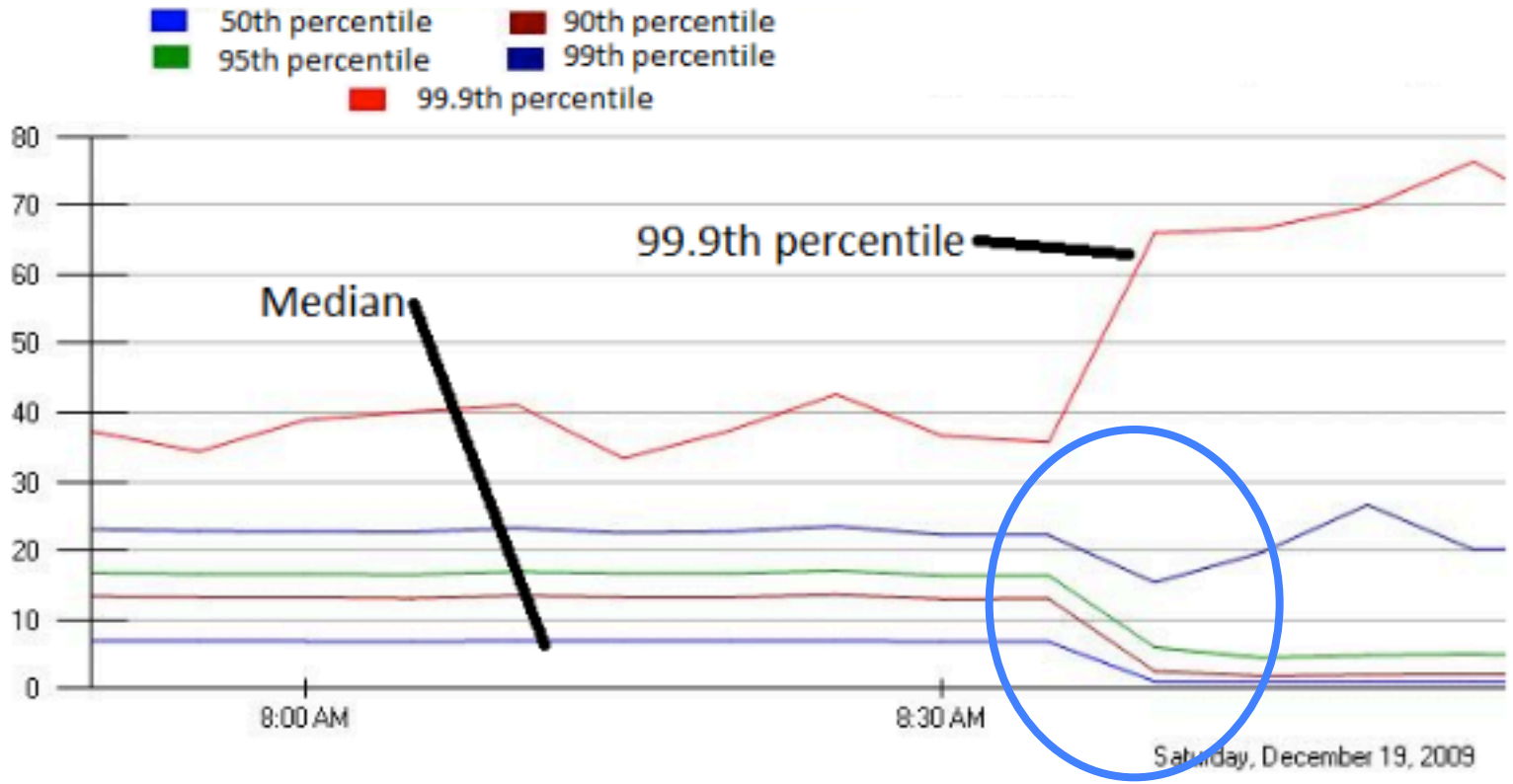
- The long, greedy TCP flows build up queues on their interfaces. Since buffer space is a shared resource (shallow buffered switches), the queue build up reduces the amount of buffer space available to absorb bursts of traffic from Partition/Aggregate traffic. → Packet loss & timeouts

# Data Center Transport Requirements

- Low latency for short flows
- High burst tolerance **(handle incast problem due to Partition/ Aggregation)**
- High throughput for long flows
- Switch buffer occupancies need to be persistently low, while maintaining high throughput for the long flows

# TCP in the Data Center

- TCP does not meet demands of apps.
  - Incast
    - ➢ Suffers from bursty pacl
  - Builds up large queues:
    - ➢ Adds significant latency
    - ➢ Wastes precious buffer
- Operators work aroun
  - – Ad-hoc, inefficient, ofter



# DCTCP Algorithm

# Congestion Control

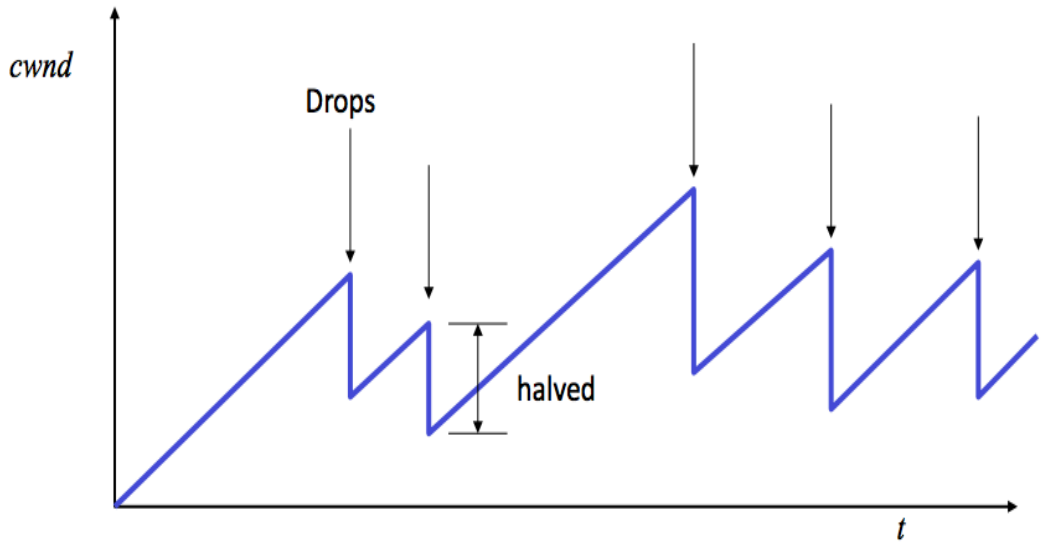
- **Sliding window: A flow control technique**
  - A generalization of stop-and-wait
  - Allow up to W unack packets in flight at any time (W = window size)
- **Congestion: Different sources compete for resources in the network**
  - Flows using up all link capacity $\Sigma r > C$
  - Short flows compete with large flows on buffers
- **Ways to perform congestion control:**
  - End-hosts (e.g. TCP congestion control)
  - Network-based (e.g. ECN)

# TCP Congestion Control

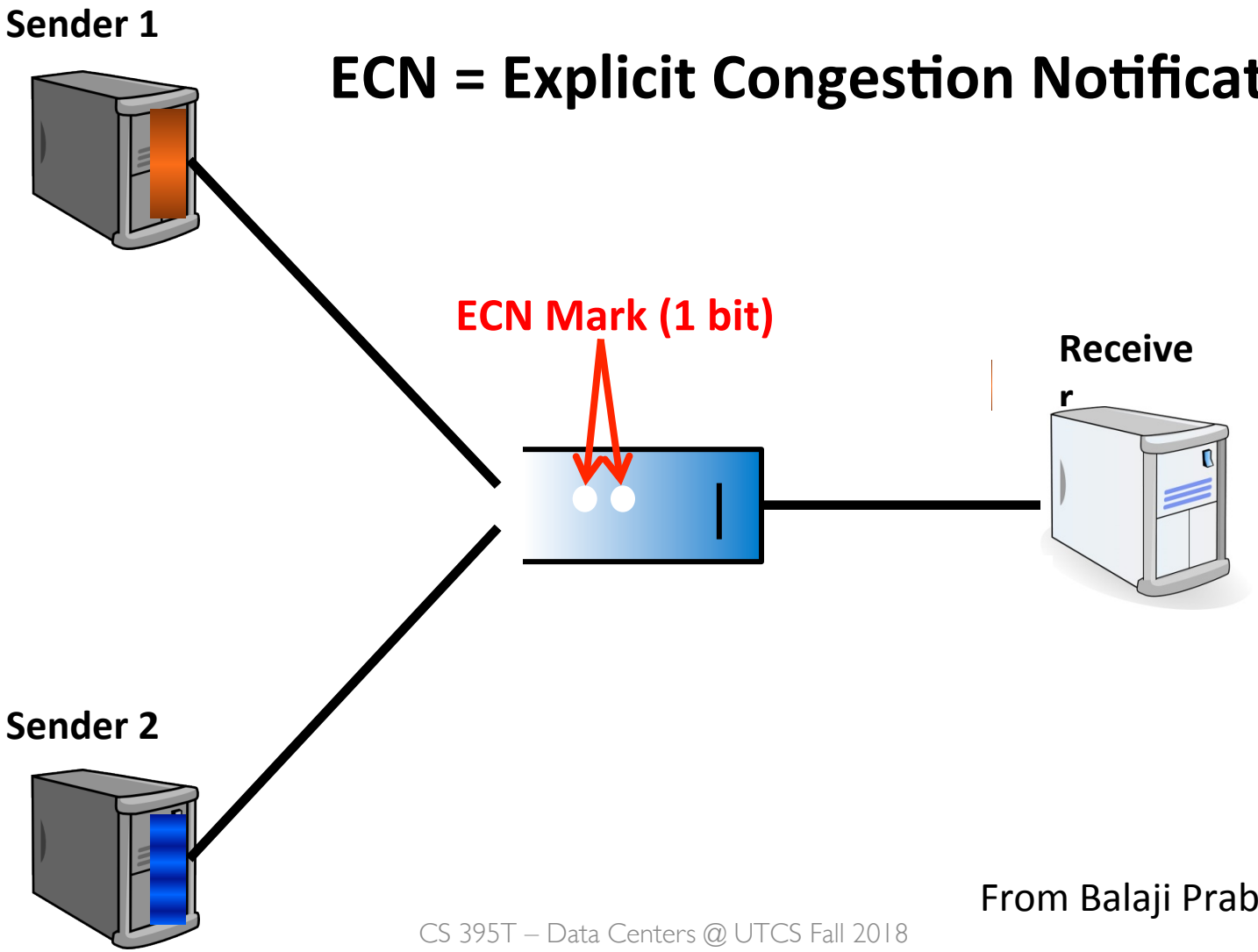
- **TCP varies the number of outstanding packets in the network by varying the window size**
  - Window size = min ( Advertised Window, Congestion Window )
  - Congestion Window is denoted as "cwnd"
  - Packet dropped ➡ congestion
- **How do we set cwnd? AIMD**
  - Additive Increase, Multiplicative Decrease

If packet received OK: $W \leftarrow W + \frac{1}{W}$

If a packet is dropped: $W \leftarrow \frac{W}{2}$



# Review: The TCP/ECN Control Loop




From Balaji Prabhakar, Stanford University

# Data Center Environment

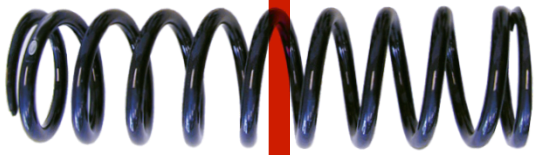
- Low round trip times (less than 250 microseconds)
- Little Statistical Multiplexing
- Network is homogeneous
- A single administrative control
- Separate from external traffic

# Two Categories of Congestion Control

- **Delay-based protocols use increases in RTT measurements as a sign of growing queuing delay, and hence of congestion**
  - Rely heavily on accurate RTT measurement → susceptible to noise
- **Active Queue Management (AQM) approaches use explicit feedback from congested switches**
  - DCTCP
  - RED (randomly early marking): RED monitors the average queue size marks packets based on statistical probabilities. If the buffer is almost empty, then all incoming packets are accepted. As the queue grows, the probability for dropping an incoming packet grows too. When the buffer is full, the probability has reached 1 and all incoming packets are dropped. → average queue size is too slow for bursty flow

# Balance Between Requirements

**High Throughput**
**High Burst Tolerance**



**Low Latency**

**Deep Buffers:**

- Qu
  Inc

**Shallow Buffers:**

&

**Objective:**
**Low Queue Occupancy & High Throughput**

**Reduced $RTO_{min}$ (SIGCOMM '09)**

- Doesn't Help Latency

**AQM – RED:**

- Avg Queue Not Fast Enough for Incast

From Balaji Prabhakar, Stanford University

# Two Key Ideas

1. React in proportion to the **extent** of congestion, not its **presence**.
   - ✓ Reduces variance in sending rates, lowering queuing requirements.

| 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% |

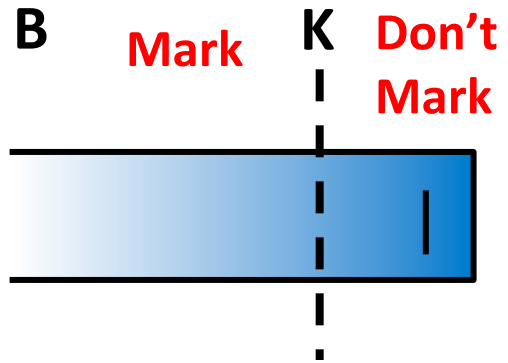
2. Mark based on **instantaneous** queue length.
   - ✓ Fast feedback to better deal with bursts.

From Balaji Prabhakar, Stanford University

# Data Center TCP Algorithm

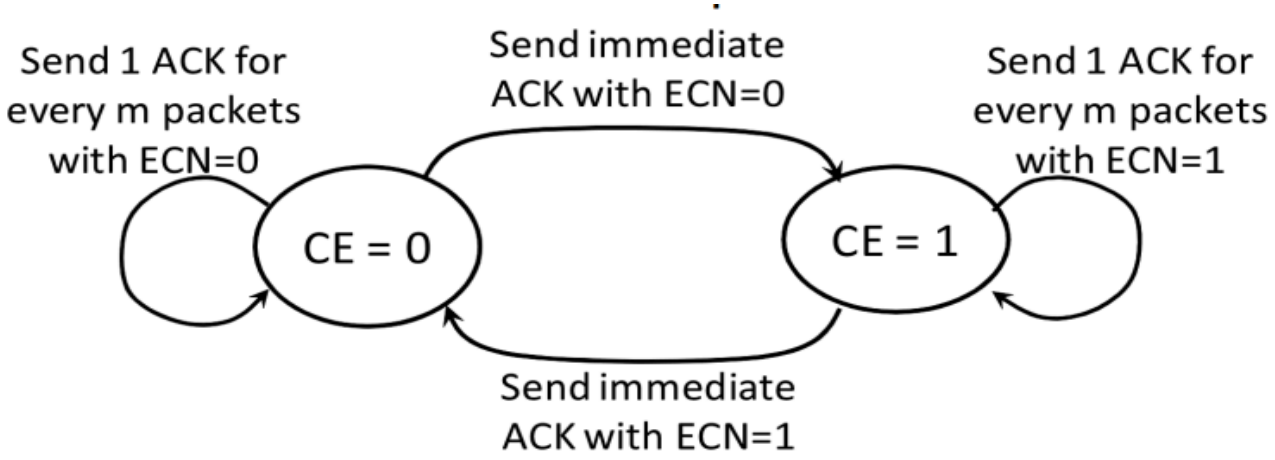
**Switch side:**

- Mark packets with Congestion Experienced (CE) code point when Queue Length > K.




**Receiver side:**

- Use state machine to decide whether to set ECN-Echo flag






Adapted from Balaji Prabhakar, Stanford University

# Data Center TCP Algorithm

**Sender side:**

- Maintain running average of ***fraction*** of packets marked **($\alpha$)**.

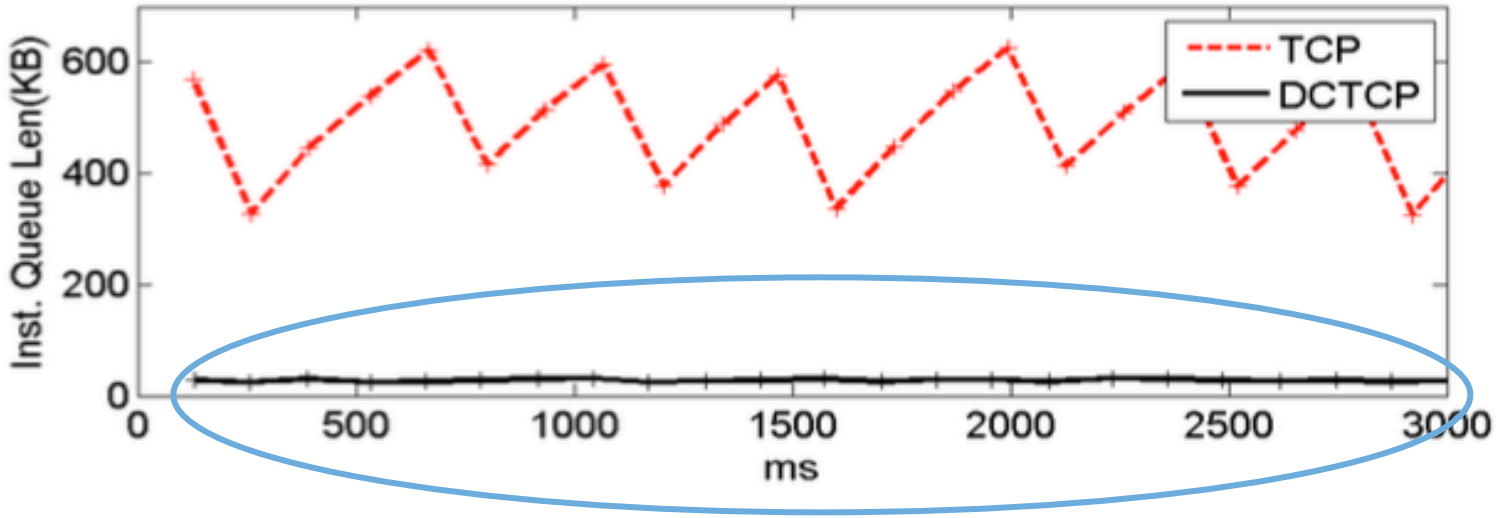
**In each RTT (i.e., once every W of data):**

$$F = \frac{\#\ of\ marked\ ACKs}{Total\ \#\ of\ ACKs} \qquad \alpha \leftarrow (1-g)\alpha + gF \qquad 0 < g < 1$$

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

Adapted from Balaji Prabhakar, Stanford University

# DCTCP in Action




Buffer occupancies are persistently low!

**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.**

# Why it Works

**1. High Burst Tolerance**

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

**2. Low Latency**

- ✓ Small buffer occupancies → low queuing delay.

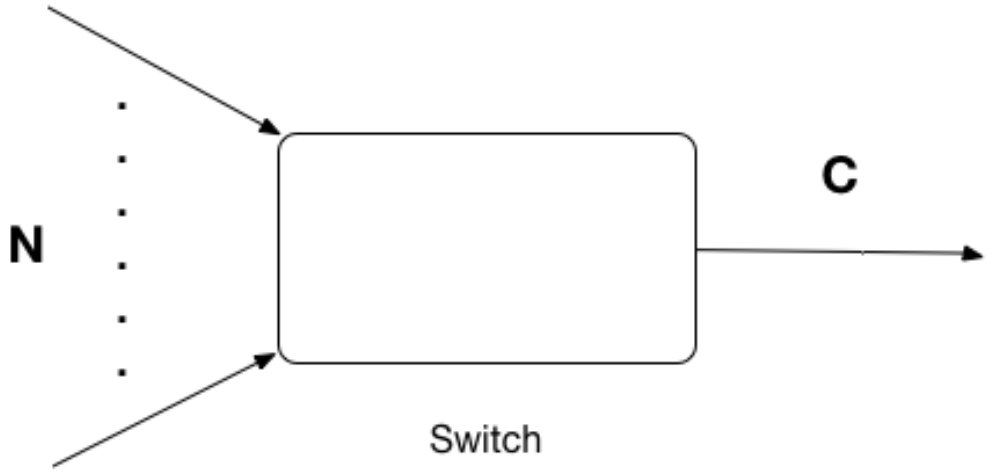
**3. High Throughput**

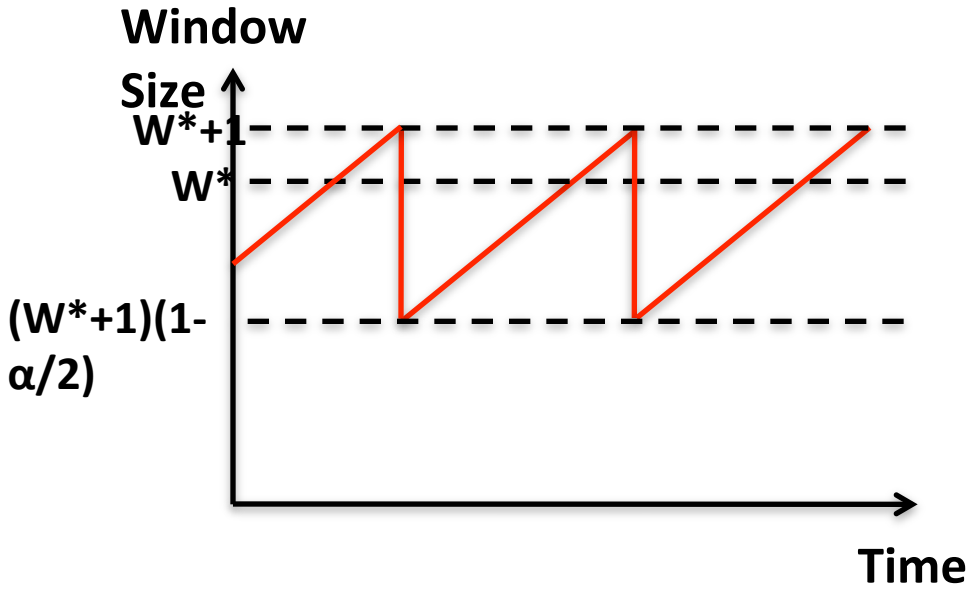
- ✓ ECN averaging → smooth rate adjustments, cwind low variance.

From Balaji Prabhakar, Stanford University

# Analysis

- **Assumptions:**
  - N infinitely long-lived flows with identical round-trip times RTT, sharing a single bottleneck link of capacity C.
  - N flows are synchronized (i.e. their "sawtooth" window dynamics are in-phase).



# Analysis

- Goals:
  - Mathematically characterize the "sawtooth" by computing:
    - The maximum queue size $Q_{max}$
    - The amplitude of queue oscillations A
    - The period of oscillations $T_C$

*A* quantifies how well DCTCP is able to maintain steady queues!

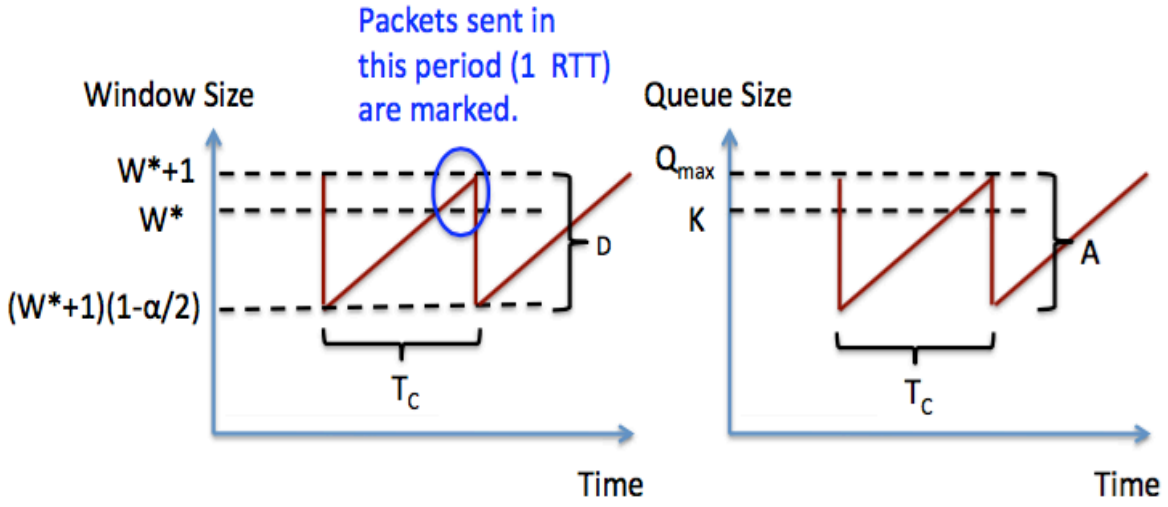



**Figure 11: Window size of a single DCTCP sender, and the queue size process.**

# Compute *A*

- By assumption, A = ND, where D is the amplitude of oscillation in window size of a single flow
- $$D = (W^* + 1) - (W^* + 1)(1 - \alpha/2). \qquad (7)$$
- W* is the window size at which the queue size reaches K
- $W^* = (C \times RTT + K)/N$
- Thus, to calculate A, we need to compute $\alpha$ :
  - Key observation: the queue size exceeds K for exactly one RTT in each period of the "sawtooth", before the sources receive ECN marks and reduce their window sizes accordingly.
  - We can compute $\alpha$ (the fraction of marked packets) by :
    - the number of packets sent during the last RTT of the period / the total number of packets sent during a full period of the sawtooth

# Compute *A*

$$S(W_1, W_2) = (W_2^2 - W_1^2)/2. \qquad (4)$$

$$\alpha = S(W^*, W^* + 1)/S((W^* + 1)(1 - \alpha/2), W^* + 1). \qquad (5)$$

$$\alpha^2(1 - \alpha/4) = (2W^* + 1)/(W^* + 1)^2 \approx 2/W^*, \qquad (6)$$

$$\alpha \approx \sqrt{2/W^*}.$$

# Compute *A*

$$A = ND = N(W^* + 1)\alpha/2 \approx \frac{N}{2}\sqrt{2W^*}$$

$$= \frac{1}{2}\sqrt{2N(C \times RTT + K)}, \qquad (8)$$

# Compute $T_C$

$$T_C = D = \frac{1}{2}\sqrt{2(C \times RTT + K)/N} \text{ (in RTTs).} \qquad (9)$$

# Compute $Q_{max}$

$$Q(t) = NW(t) - C \times RTT, \qquad (3)$$

- Queue size at time t = Arrival rate at t – Departure rate at t

$$Q_{max} = N(W^* + 1) - C \times RTT = K + N. \qquad (10)$$

# How good is the Analysis?

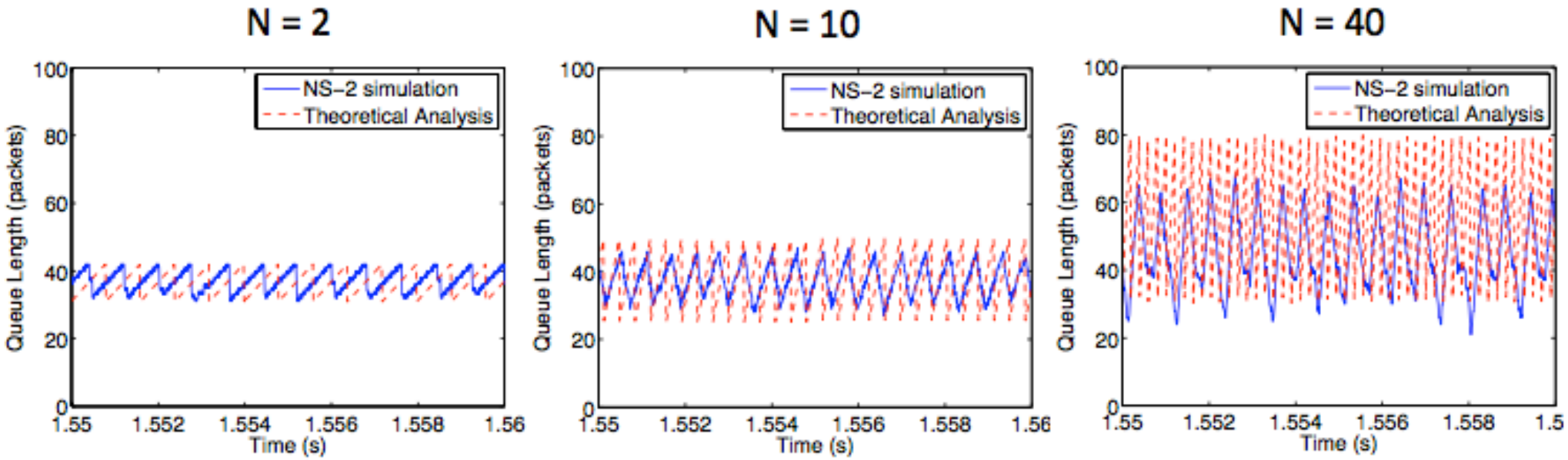


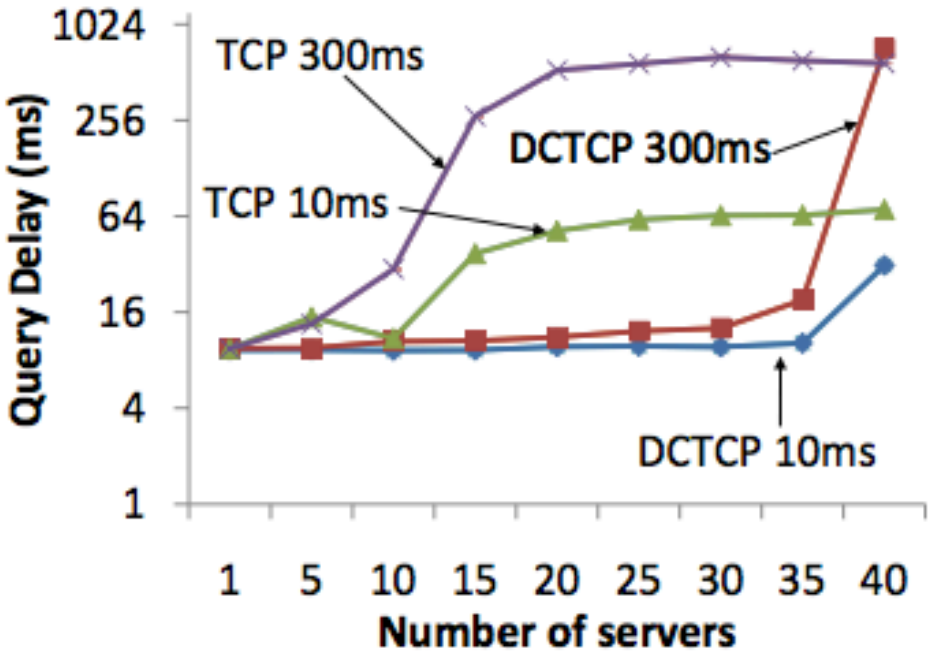

**Figure 12: Comparison between the queue size process predicted by the analysis with NS-2 simulations. The DCTCP parameters are set to $K = 40$ packets, and $g = 1/16$.**

# Evaluation

- Implemented in Windows stack.
- Real hardware, **1Gbps and 10Gbps** experiments
  - 90 server testbed
  - Broadcom Triumph 48 1G ports – 4MB shared memory
  - Cisco Cat4948 48 1G ports – 16MB shared memory
  - Broadcom Scorpion 24 10G ports – 4MB shared memory
- Numerous benchmarks
  - Throughput and Queue Length
  - Multi-hop
  - Queue Buildup
  - Buffer Pressure
  - **Fairness and Convergence**
  - **Incast**
  - **Static vs Dynamic Buffer Mgmt**

From Balaji Prabhakar, Stanford University

# Microbenchmarks: Incast



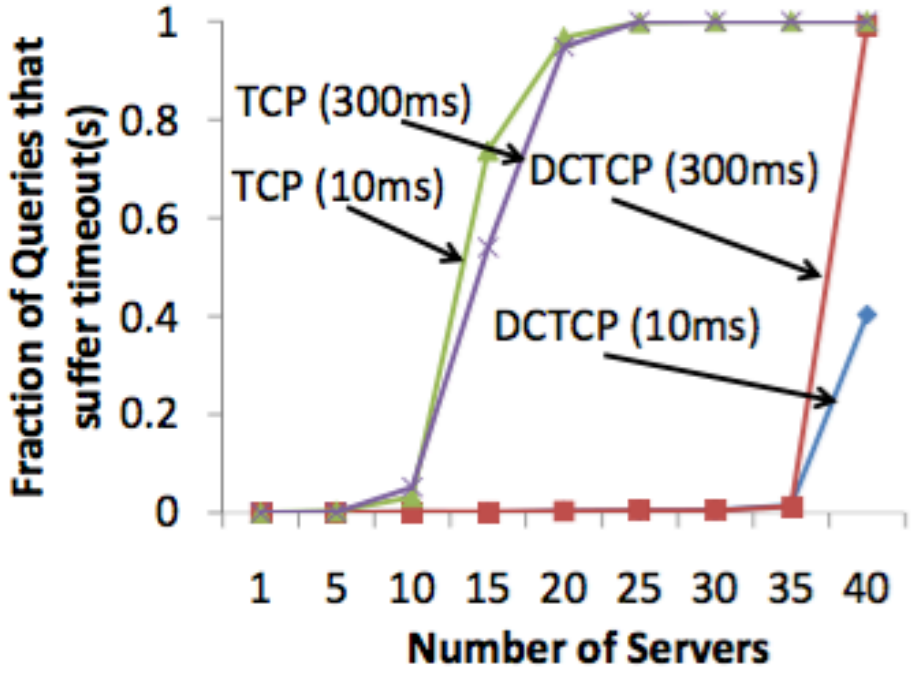
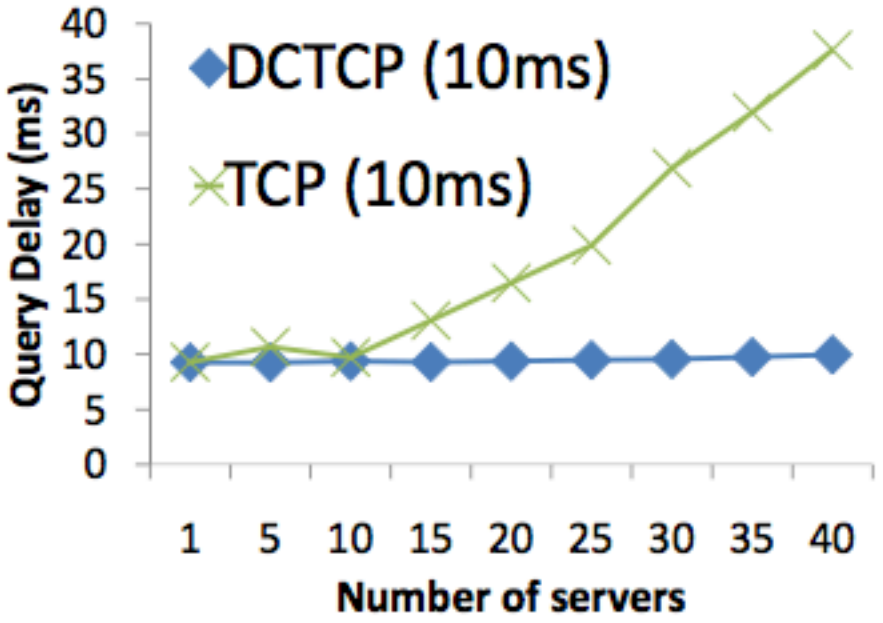




**Figure 18: DCTCP performs better than TCP and then converges at 35 senders (log scale on Y axis; 90% confidence intervals for the means are too small to be visible).**

# Microbenchmarks: Incast



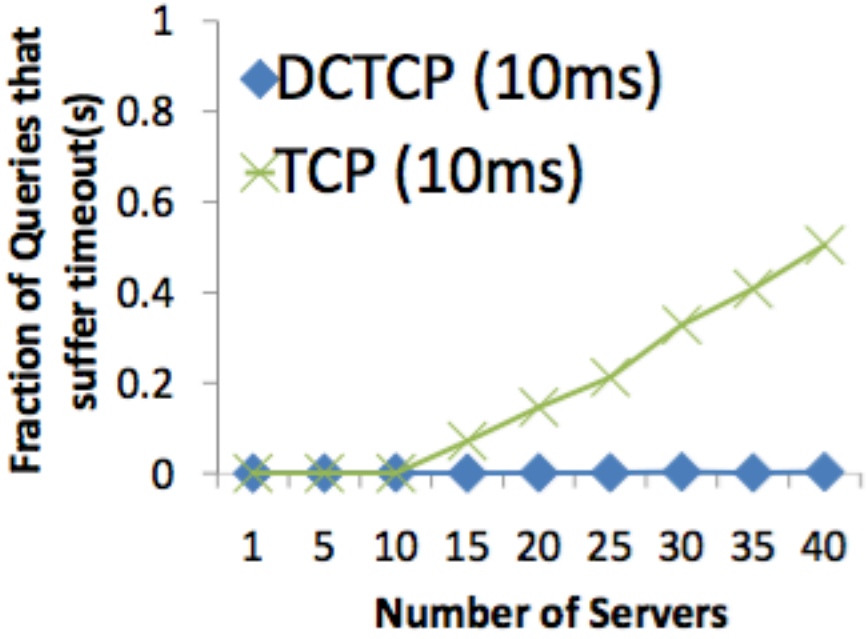




**Figure 19: Many-to-one: With dynamic buffering, DCTCP does not suffer problems at even high load.**

# Microbenchmarks: Queue buildup

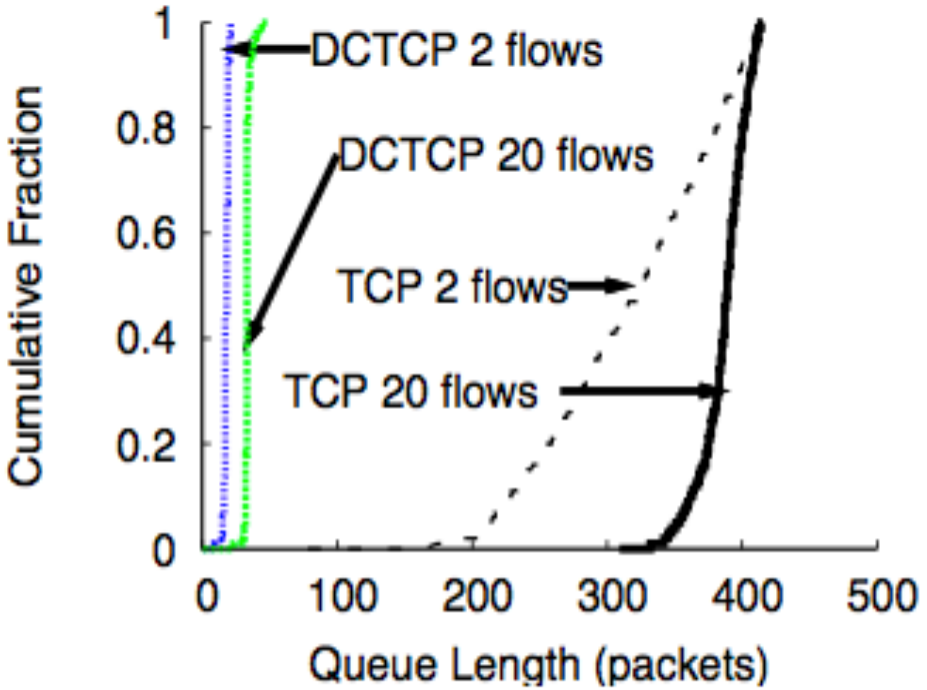



**Figure 13: Queue length CDF (1Gbps)**

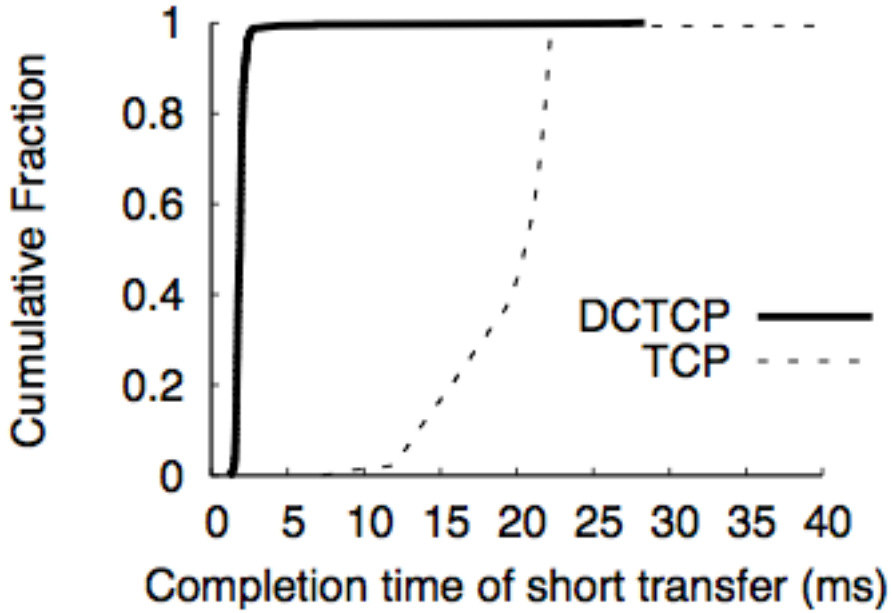


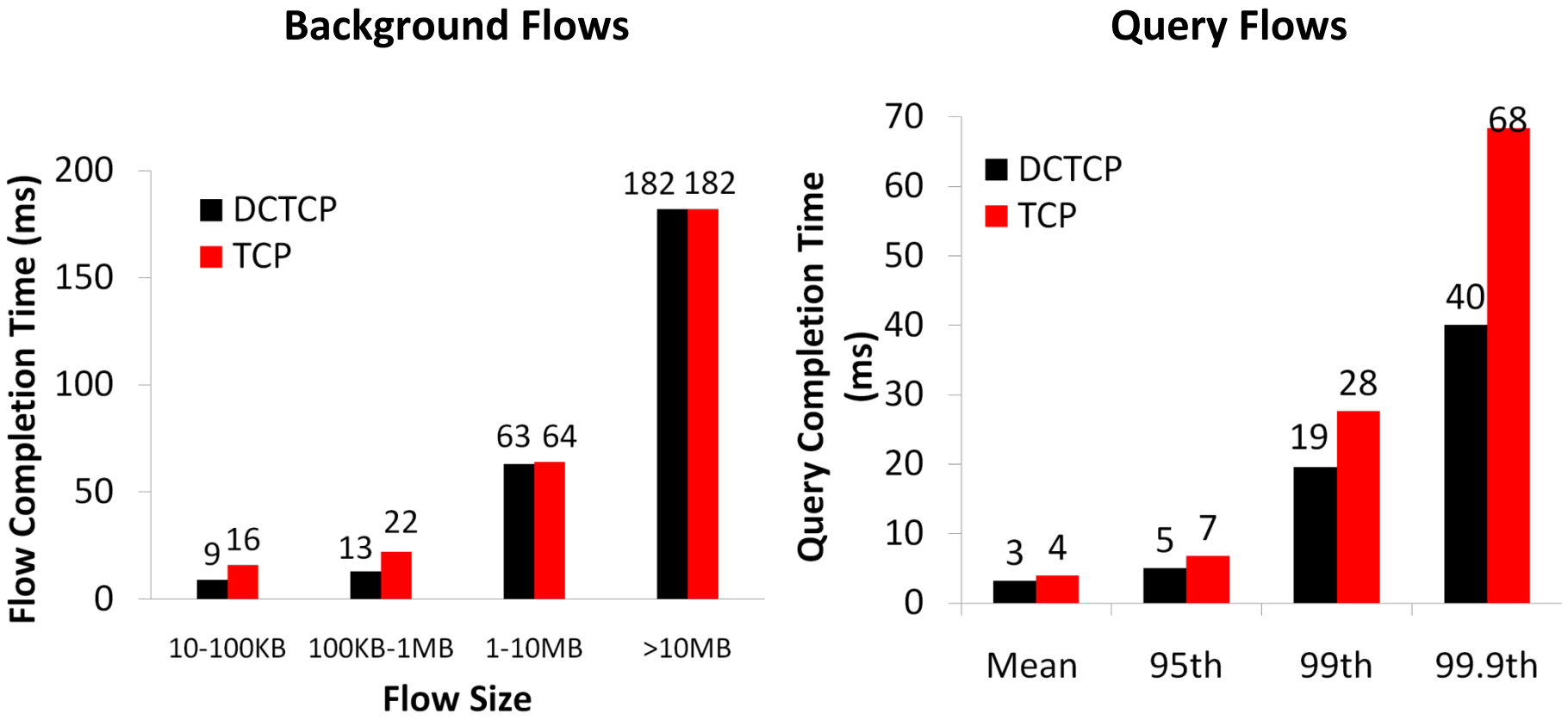

**Figure 21: Short transfers see low delay with DCTCP.**

# Microbenchmarks: Buffer Pressure

| | Without background traffic | With background traffic |
|---|---|---|
| TCP | 9.87ms | 46.94ms |
| DCTCP | 9.17ms | 9.09ms |

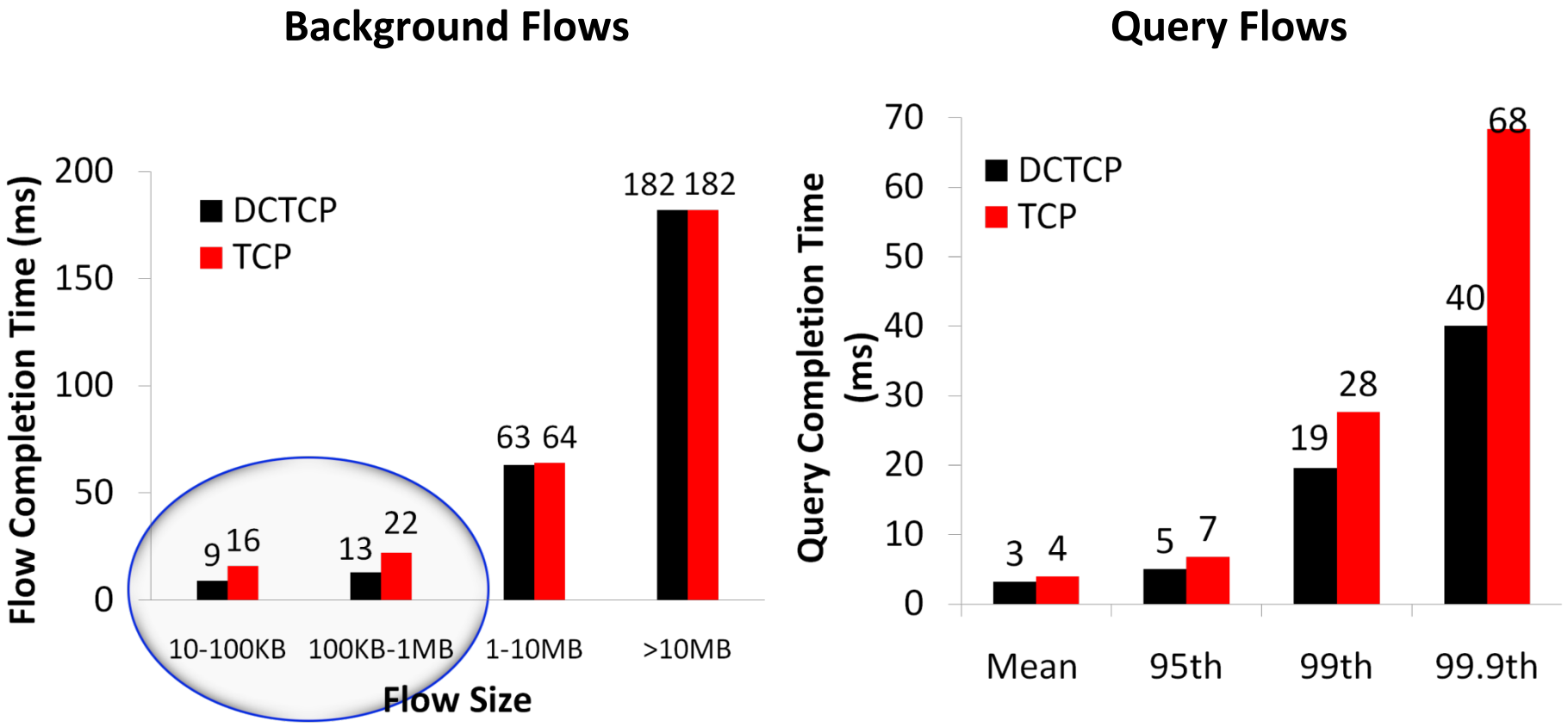
**Table 2: $95^{th}$ percentile of query completion time. DCTCP prevents background traffic from affecting performance of query traffic. $RTO_{min} = 10ms$, $K = 20$.**

# Evaluation




From Balaji Prabhakar, Stanford University

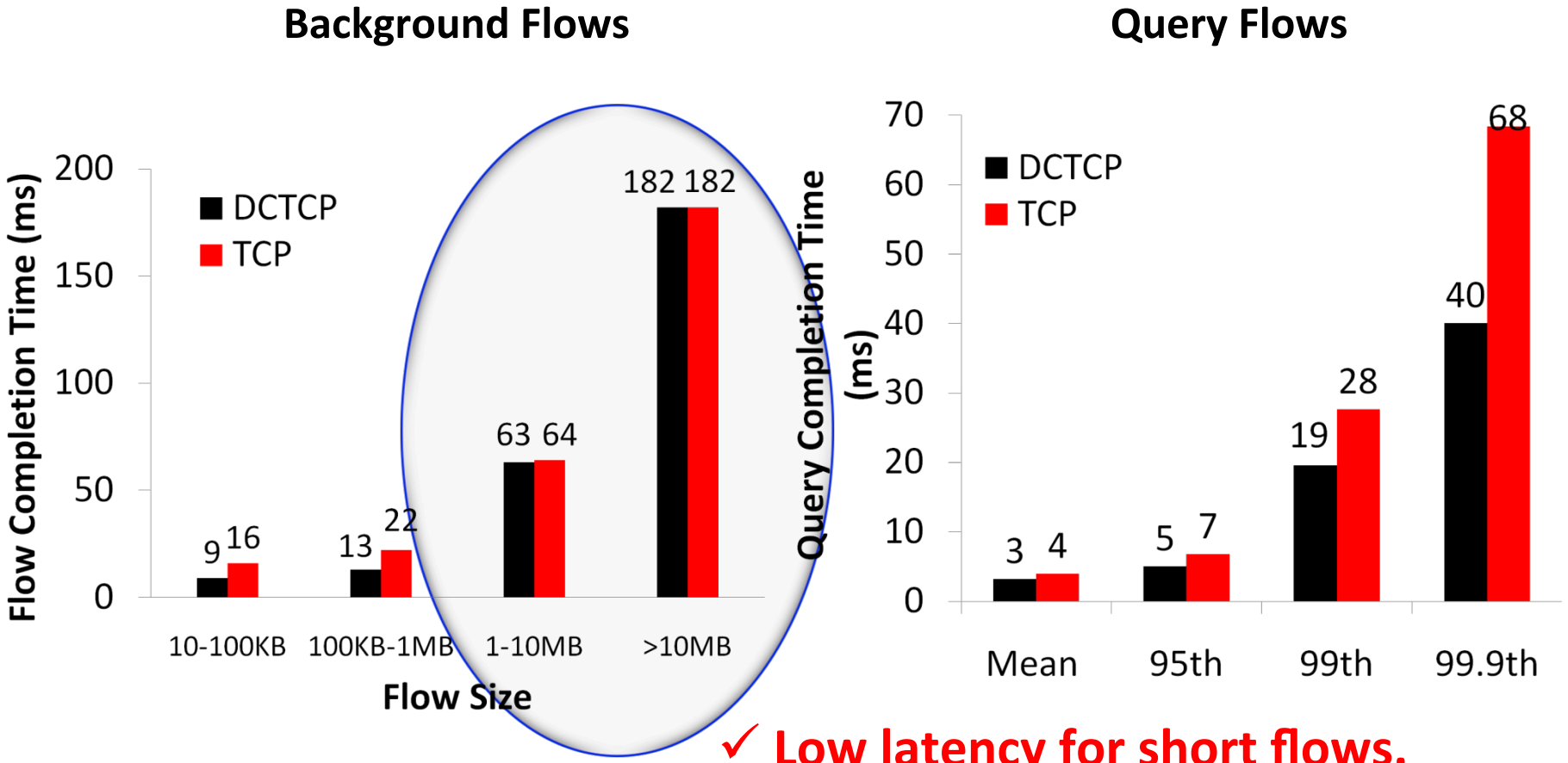
# Evaluation




✓ **Low latency for short flows.**

From Balaji Prabhakar, Stanford University

# Evaluation

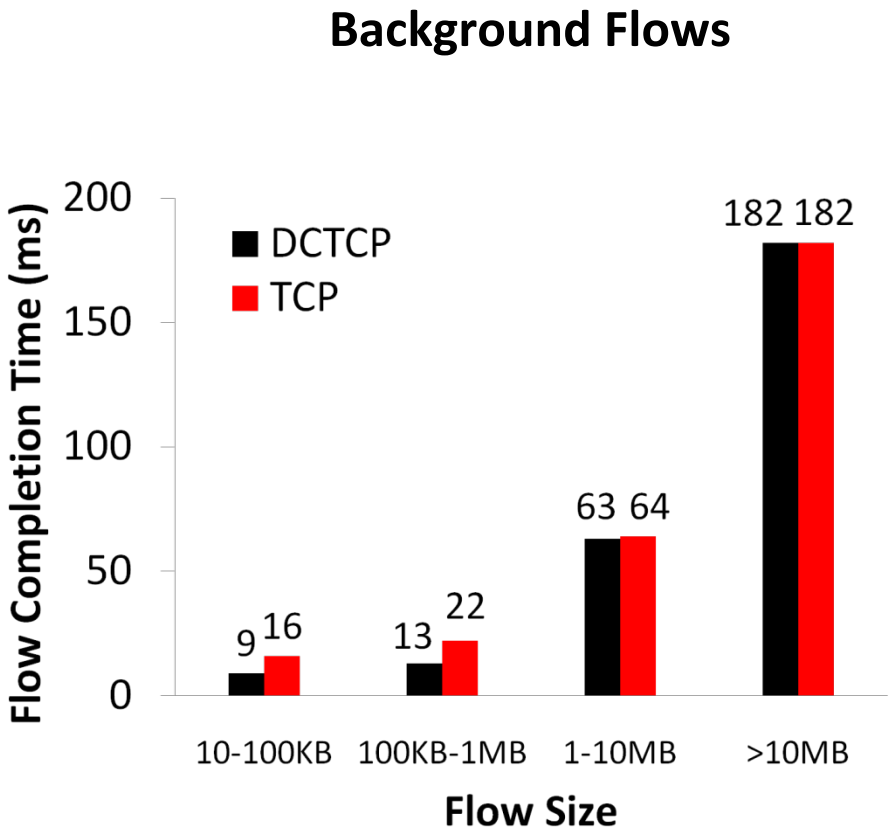


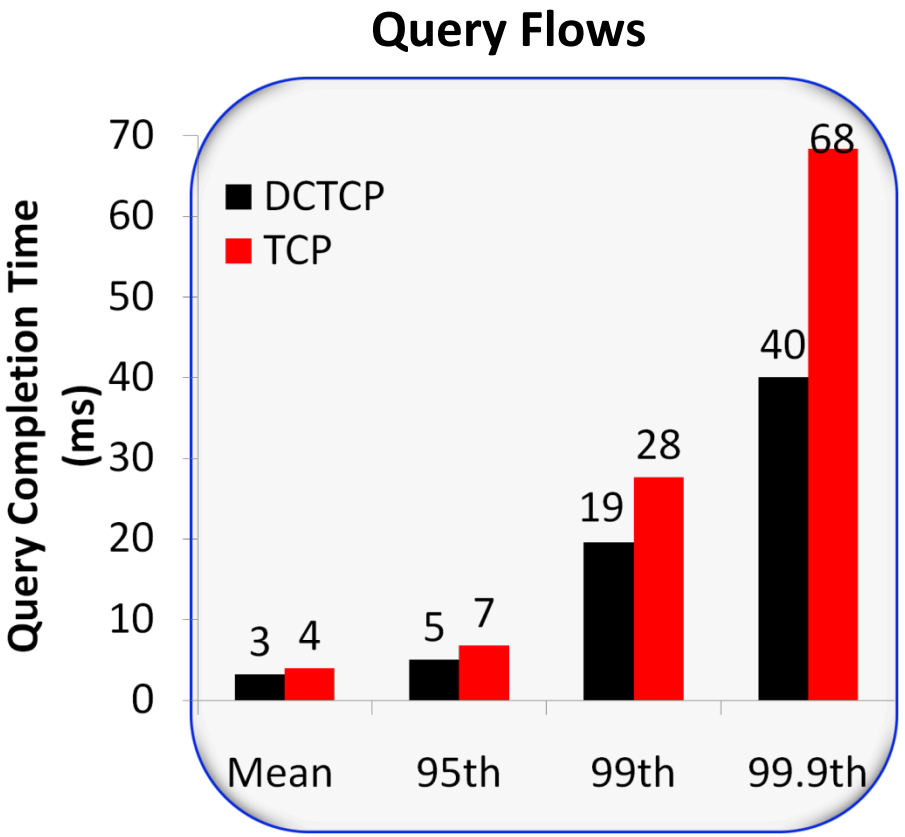

✓ **Low latency for short flows.**

✓ **High throughput for long flows.**

From Balaji Prabhakar, Stanford University

# Evaluation




- ✓ **Low latency for short flows.**
- ✓ **High throughput for long flows.**
- ✓ **High burst tolerance for query flows.**

From Balaji Prabhakar, Stanford University

# Conclusions: Measure and analyze production traffic

- 99.91% of data center traffic is TCP
- Throughput-sensitive large flows, delay sensitive short flows and bursty query traffic co-exists in a data center network
- Switch buffer occupancies need to be persistently low, while maintaining high throughput for the long flows
- Data Center Transport Requirements
  - Low latency for short flows
  - High burst tolerance
  - High utilization for long flows

# Conclusions: DCTCP

- DCTCP satisfies all our requirements for Data Center packet transport.
  - ✓ Handles bursts well
  - ✓ Keeps queuing delays low
  - ✓ Achieves high throughput
- Features:
  - ✓ Very simple change to TCP and a single switch parameter K.
  - ✓ Based on ECN mechanisms already available in commodity switch.

# Terms appeared in the paper

- MMU (Memory Management Unit): perform translation of virtual memory address to physical address; effectively perform virtual memory management
- RTO (Recovery Time Objective): duration of time before packet retransmission (i.e. TCP minimum retransmission timeout)
- Statistical multiplexing: the idea of taking a single resource and sharing it across multiple users in a probabilistic or statistical
  - It's statistical in that each user receives a statistical share of the resource based on how much others are using it
  - For example, if your friend is reading, you can use all of the link. If both of you are loading a page, you receive half of the link capacity
  - Packet switching