Accurate Latency-based Congestion Feedback for Datacenters

Changhyun Lee

with Chunjong Park, Keon Jang*, Sue Moon, and Dongsu Han

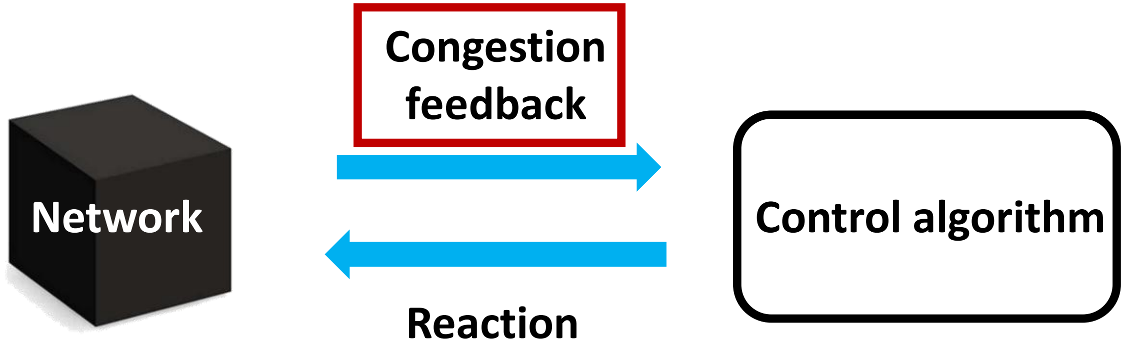
KAIST *Intel Labs

USENIX Annual Technical Conference (ATC)

July 10, 2015

Congestion control? Again???

- Numerous congestion control algorithms have been proposed since Jacobson's TCP



- Performance of congestion control fundamentally depends on congestion feedback
- New forms of congestion feedback have enabled innovative congestion control behavior
  - Packet loss, latency, bandwidth, ECN, in-network (RCP, XCP), etc.

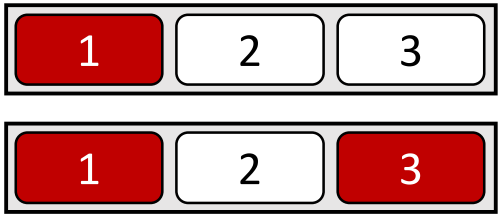
Congestion control challenges in DCN

- Datacenters' unique environment requires congestion control to be finer-grained than ever
  - Prevalence of latency sensitive flows (partition/aggregate workload)
  - Every 100ms slow down in Amazon = 1% drop in sales*
  - Dominance of queuing delay in end-to-end latency
- Accurate and fine-grained congestion feedback is **a must**!

*Cracking latency in cloud, http://www.datacenterdynamics.com/

The most popular choice so far: ECN

- ECN (Explicit Congestion Notification) detects congestion earlier than packet loss, but…
  - It still provides very coarse-grained feedback (binary)
- DCTCP puts in more effort to improve granularity
  - Other ECN-based work also employ the same technique



1 packet marked → congestion probability: 33%

2 packets marked → congestion probability: 66%

- Pursuit of better congestion feedback leads to customized in-network feedback ➔ hard to deploy

Our proposal: latency feedback

- Network latency is a good indicator of congestion

- Latency congestion feedback has a long history from CARD, DUAL, and TCP Vegas in wide-area networks
  - Used feedback: RTT measured in TCP stack

- We revisit latency feedback for use in datacenter networks

Can we reuse the same latency feedback from TCP Vegas?

Challenges in latency feedback in DC

- Network latency changes in μs time scale in datacenters

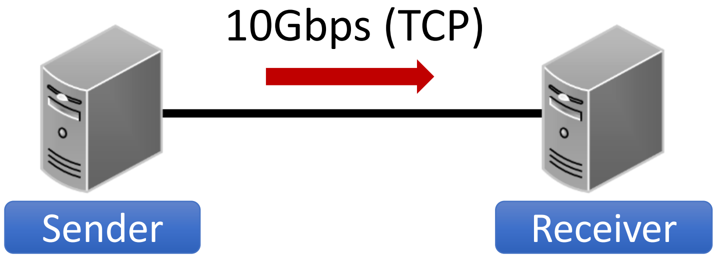
	Datacenter	Wide-area
Link speed	10 Gbps	100 Mbps
Transmission delay	1.2 μs	120 μs
Queueing delay (10 pkts)	12 μs	1.2 ms

- Differentiating network latency change from other noise becomes a challenging task

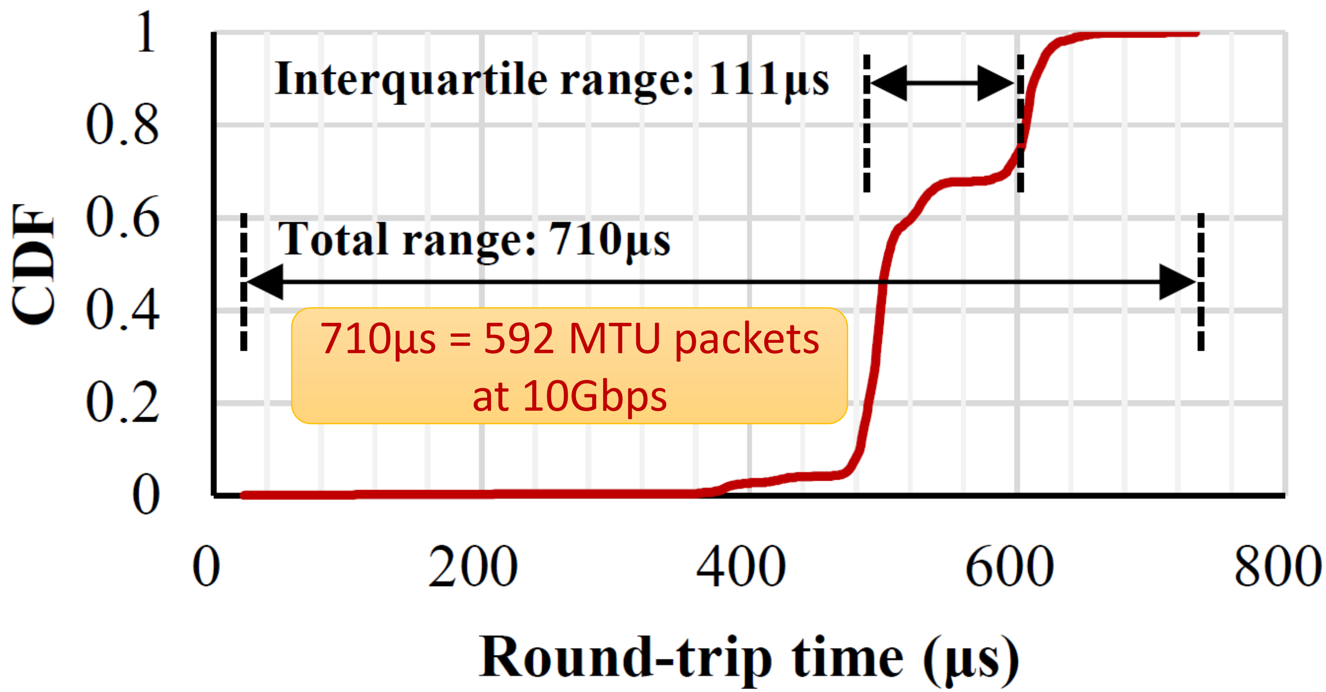
Measuring network latency accurately in microsecond scale is crucial

Evaluation of TCP stack measurement

- We test whether RTT measured in TCP stack can indicate network congestion level in datacenters
- We first evaluate the case of no congestion
- Ideally, all the RTT measurements should have the same value



Inaccuracy of TCP stack measurement



Latency feedback from stack cannot indicate network congestion level

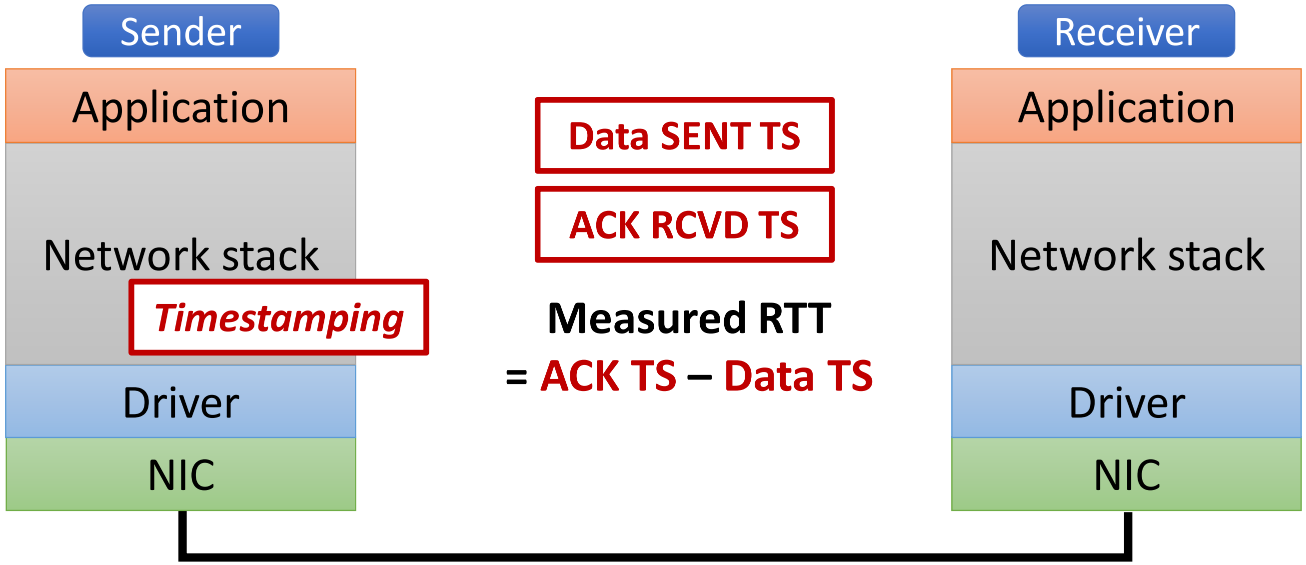
Why is TCP stack measurement unreliable?

- Sources of errors in RTT measurement
  - End-host stack delay
  - I/O batching
  - Reverse path delay
  - Clock drift

Refer to our paper

Identifying sources of errors (1)

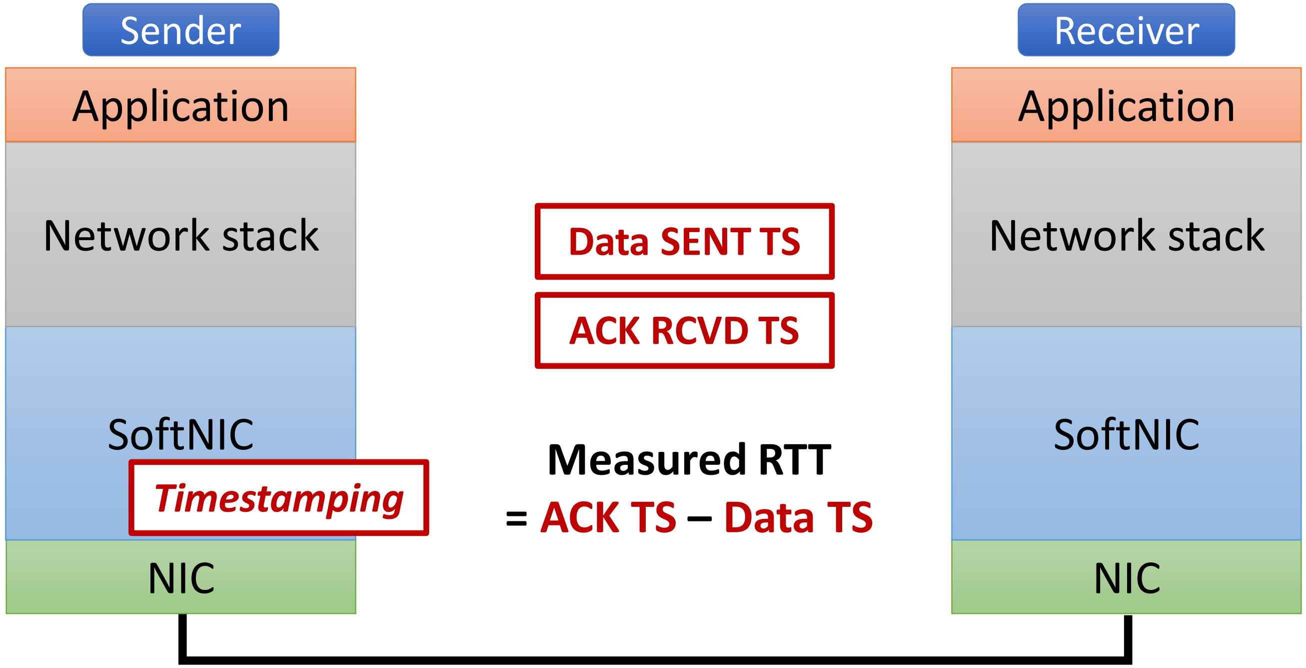
- End-host stack delay
  - Packet I/O, stack processing, interrupt handling, CPU scheduling, etc.



RTT measured from kernel gets affected by host delay jitter

Removing stack delay (sender-side)

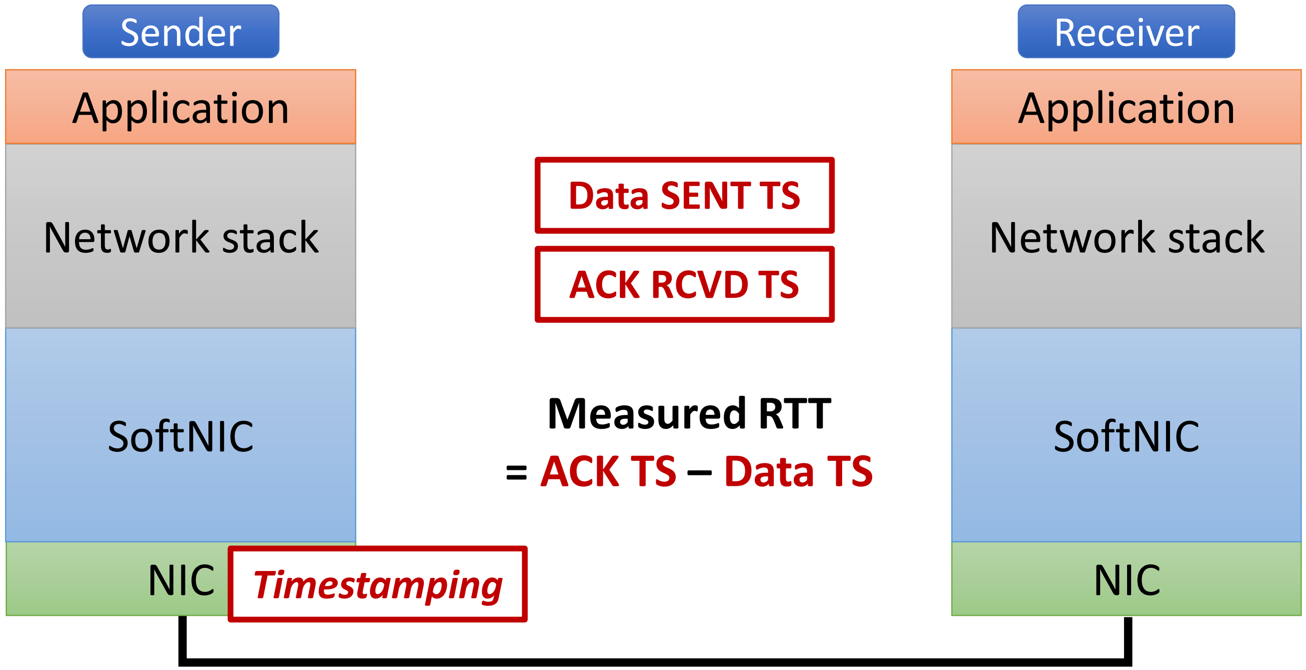
- Solution #1: Driver-level timestamping (software)
  - We use SoftNIC*, an Intel DPDK-based packet processing platform



* **SoftNIC: A Software NIC to Augment Hardware**, Sangjin Han, Keon Jang, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy (*Technical Report, UCB*)

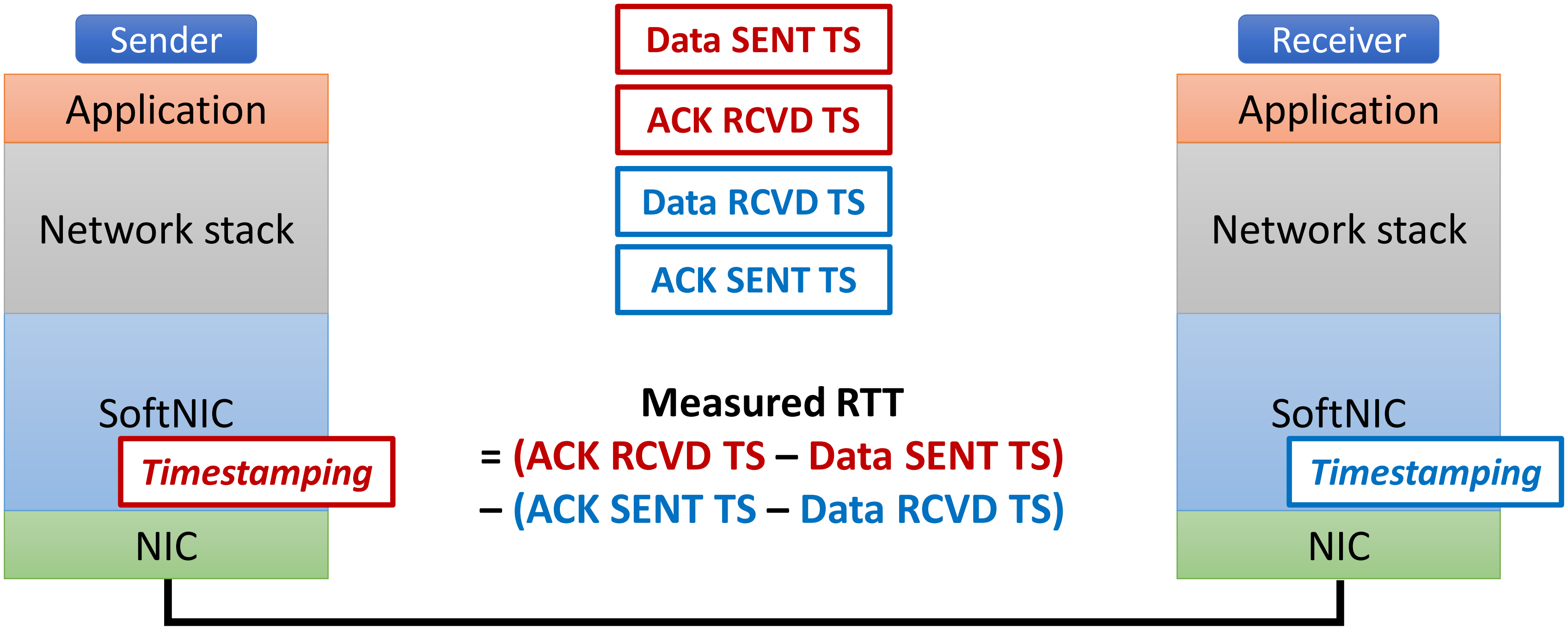
Removing stack delay (sender-side)

- Solution #2: NIC-level timestamping (hardware)
  - We use Mellanox ConnectX-3, a timestamp-capable NIC



Removing stack delay (receiver side)

- Solution #3: Timestamping also at the receiver host
  - We subtract receiver node's stack delay from RTT



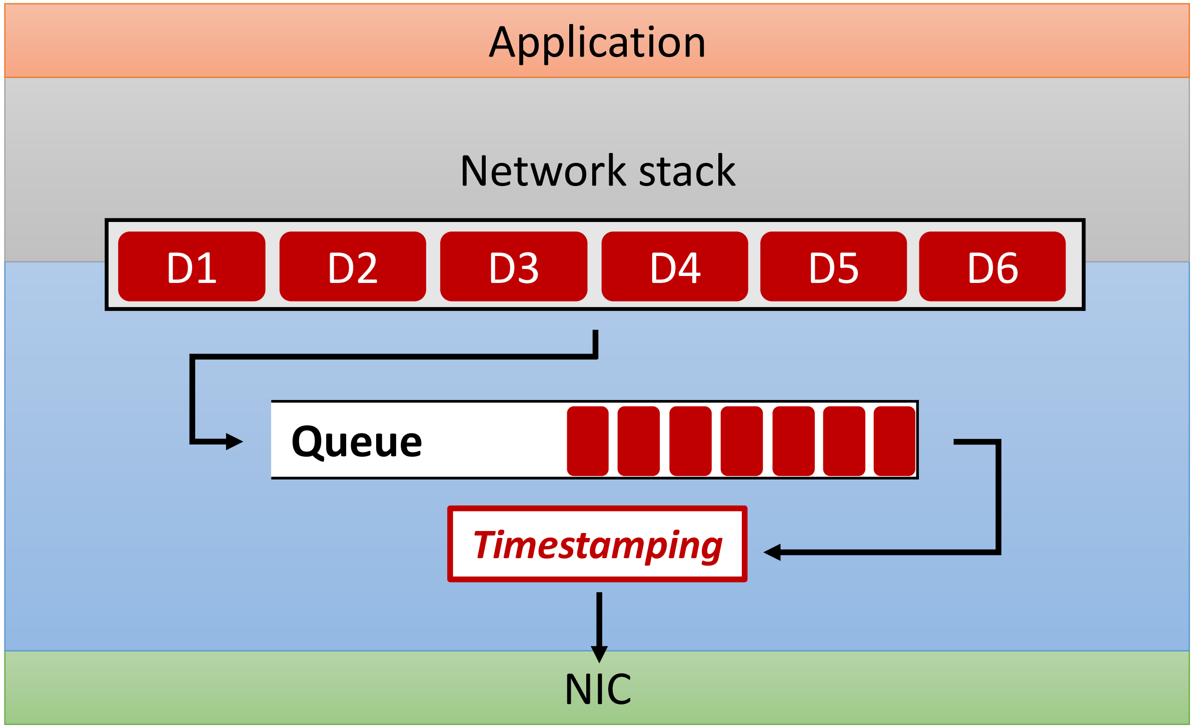
Identifying sources of errors (2)

- Bursty timestamps from I/O batching
  - Multiple packets acquire the same timestamp in network stack

Timestamps do not reflect the actual sending/receiving time

Removing bursty timestamps (driver)

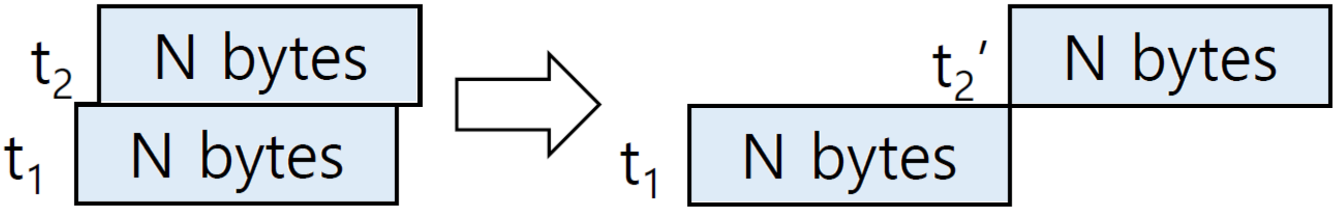
- SoftNIC stores bursty packets from upper-layer in a queue and pace before timestamping



Removing bursty timestamps (NIC)

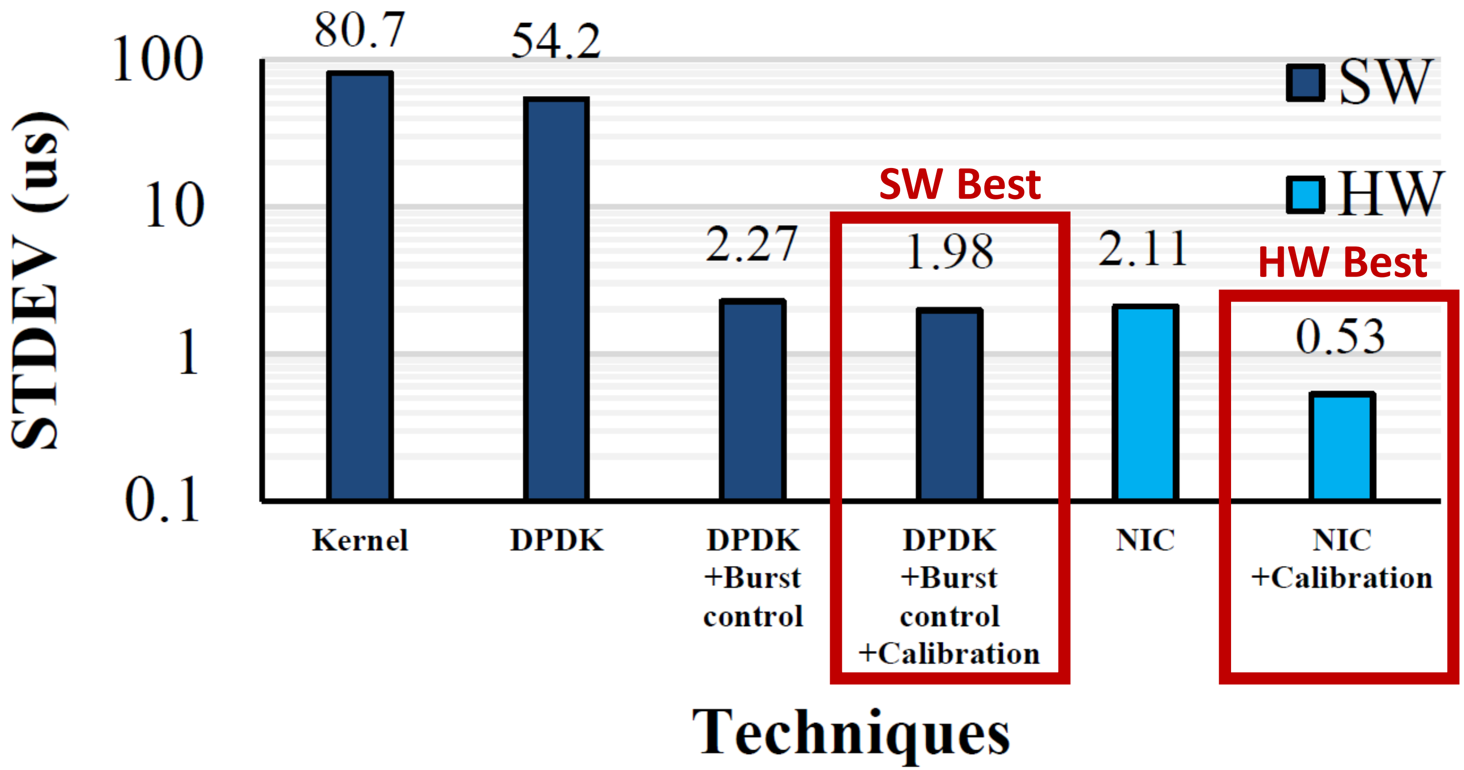
- Even NIC-level timestamping generates bursty timestamps
  - NIC timestamps packets after DMA completion, not when sending/receiving packets on the wire

- We calibrate timestamps based on link transmission delay



$$t_2' = \max(t_2, t_1 + N / C(\text{link capacity}))$$

Improved accuracy by our techniques

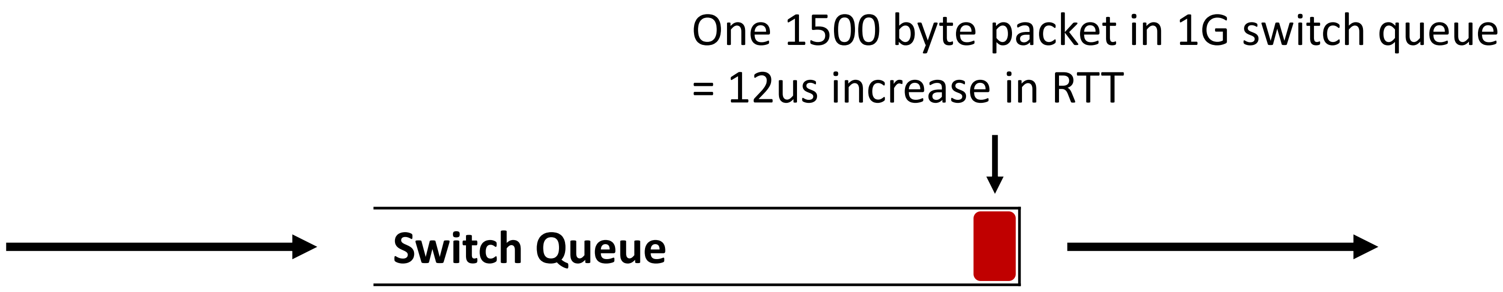


Accuracy of HW timestamping is sub-microsecond scale

Can we measure accurate queuing delay?

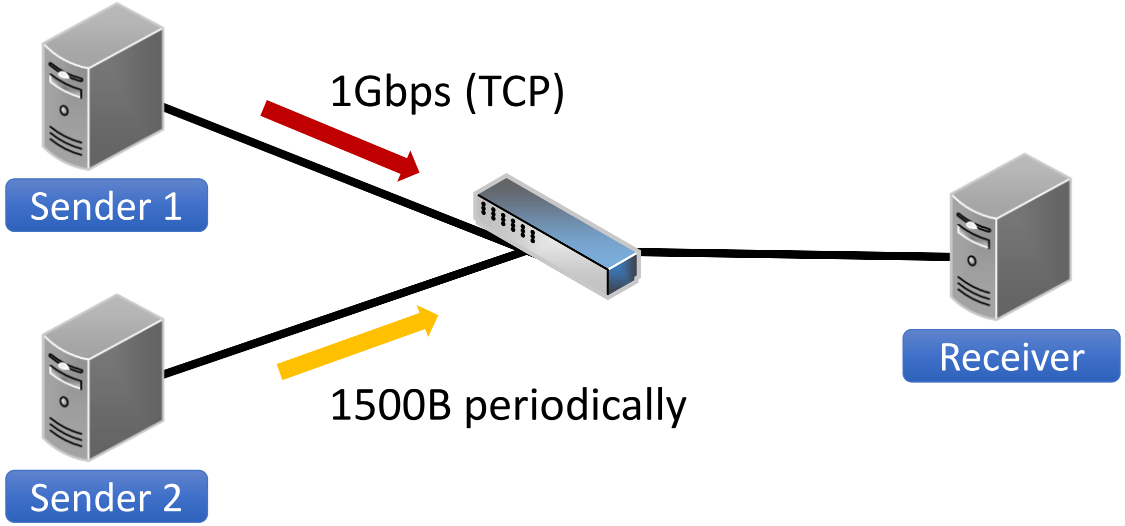
- Using our accurate RTT measurement,
  we infer queueing delay (queue length) at switch

- Queueing delay is calculated as (Current RTT – Base RTT)
  - Current RTT: RTT sample from current Data/ACK pair
  - Base RTT: RTT measured without congestion (minimum value)



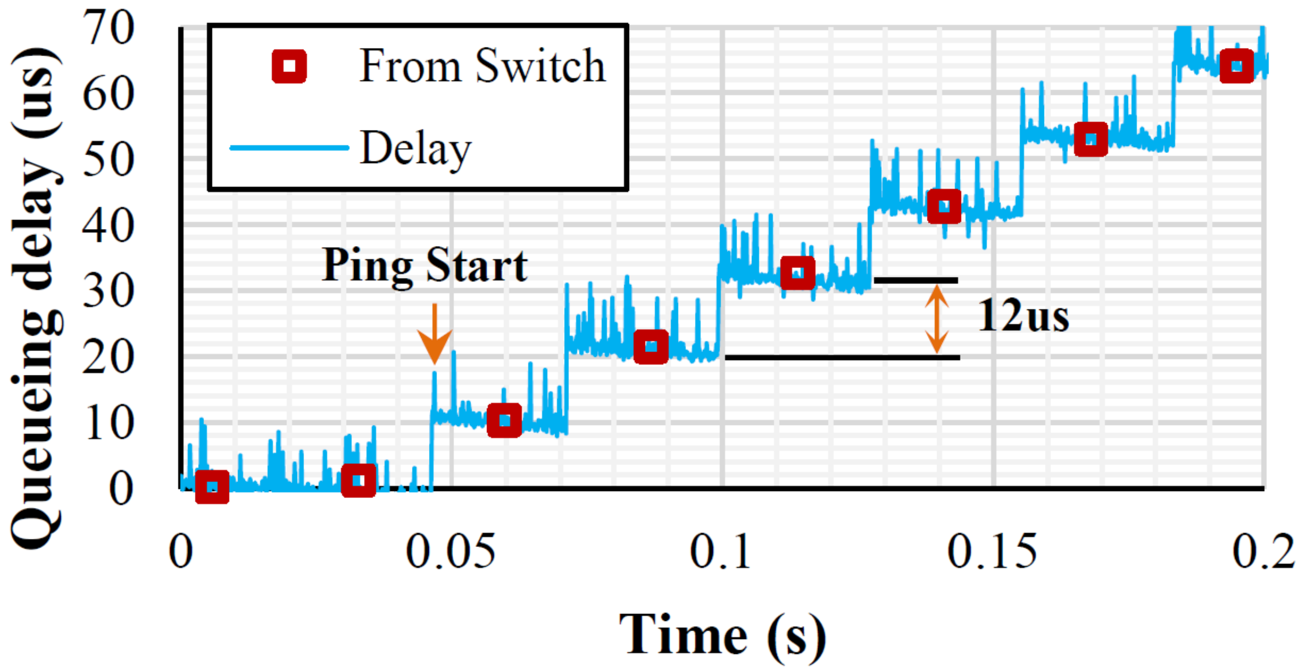
Evaluation of queuing delay measurement

- Traffic
  - Sender 1 generates 1Gbps full rate TCP traffic
  - Sender 2 generates an MTU (1500B) Ping packet every 25ms
- Measurement
  - Sender 1 measures queueing delay
  - Switch measures ground-truth queue length



Accuracy of queuing delay measurement

- We can measure queueing delay in single packet granularity
  - Ground truth from switch matches with delay measurement



DX: latency-based congestion control

- We propose DX, a new congestion control algorithm based on the accurate latency feedback
  - Goal: minimizing queueing delay while fully utilizing network links

- DX behavior is straightforward
  - When queuing delay is zero, DX increases window size
  - When queuing delay is positive, DX decreases window size

How much should we increase or decrease?

DX window calculation rule

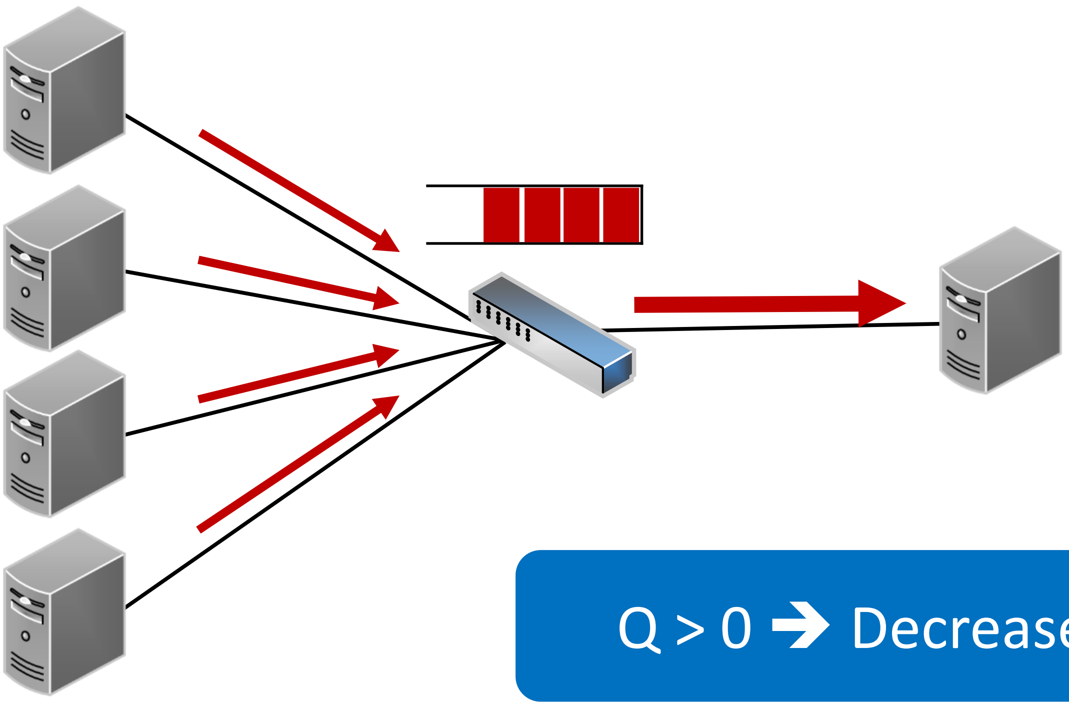
- Additive Increase: one packet per RTT
- Multiplicative Decrease: proportional to the queuing delay

- Challenge: How can we keep 100% utilization after decrement?

Q: queueing delay
V: normalizer

$$new\,CWND = \begin{cases} CWND + 1, & \text{if } Q = 0 \\ CWND \times (1 - \frac{Q}{\boxed{V}}), & \text{if } Q > 0 \end{cases}$$

DX example scenario

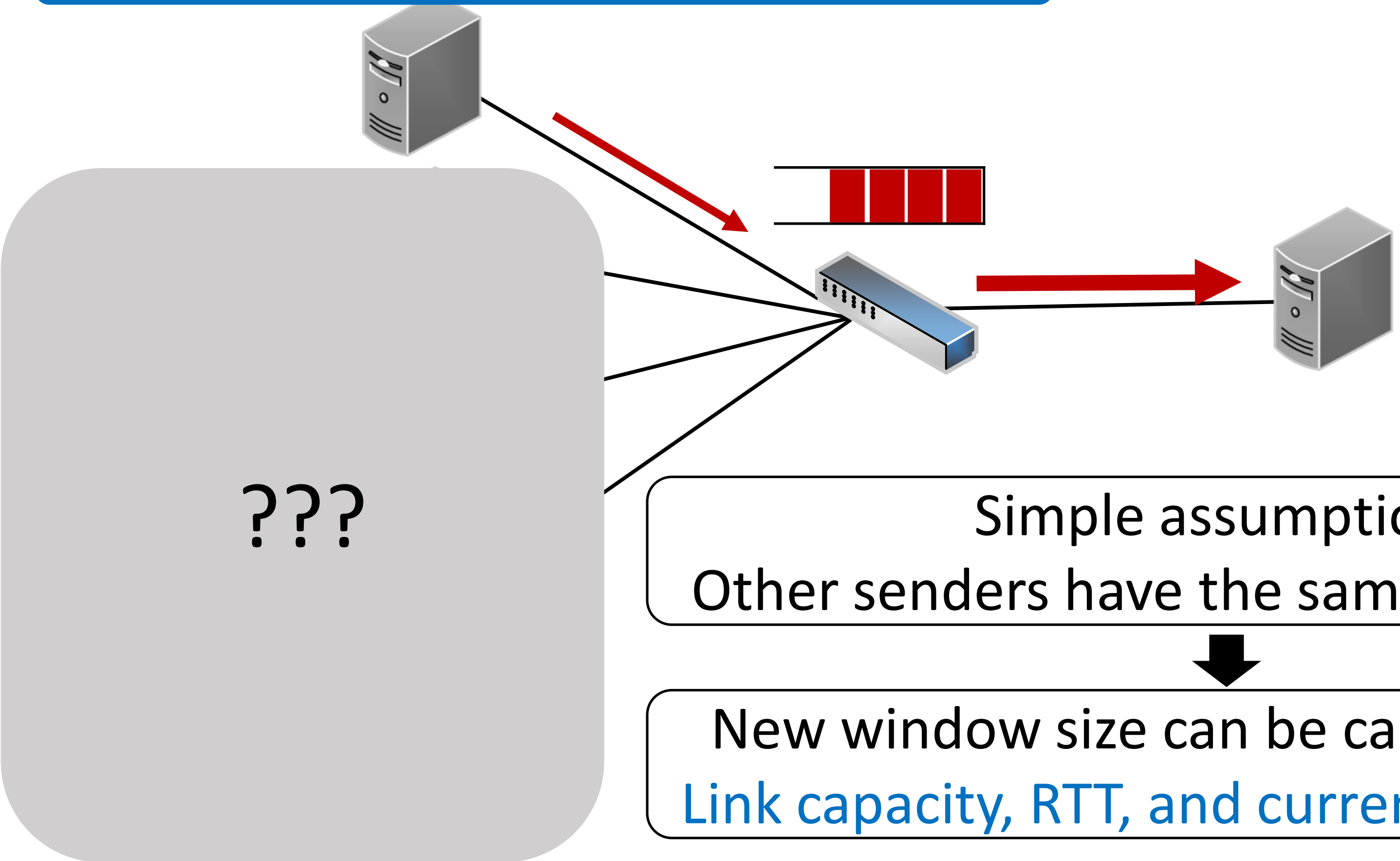


Q > 0 ➔ Decrease window

Challenge: sender #1's view

How much should I decrease?

How much congestion am "I" responsible for?



Simple assumption:
Other senders have the same window size

New window size can be calculated from
Link capacity, RTT, and current window size

*Refer to our paper for detailed derivation

Implementation

- We implement timestamping module in SoftNIC
  - Timestamp collection
  - Data and ACK packet match
  - RTT and queueing delay calculation
  - Bursty timestamp calibration
- We implement DX control algorithm in Linux 3.13 kernel
  - 200+ lines of code addition (mainly in tcp_ack())
  - Use of TCP option header for storing timestamps

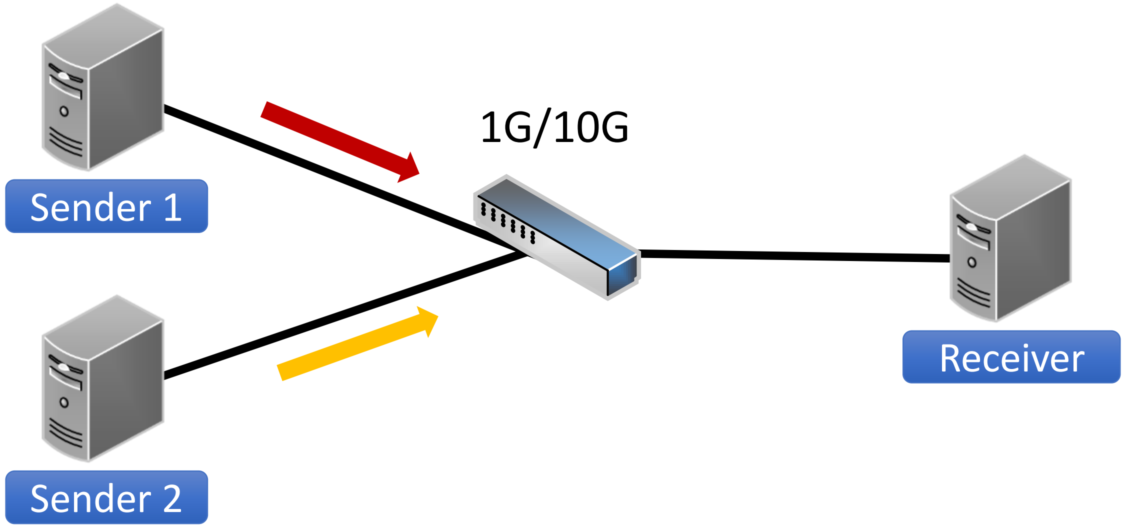
Evaluation methodology

- Testbed experiment (small-scale)
  - Bottleneck queue length in 2-to-1 topology
- Ns-2 simulation (large-scale)
  - Flow completion time of datacenter workload in a toy datacenter

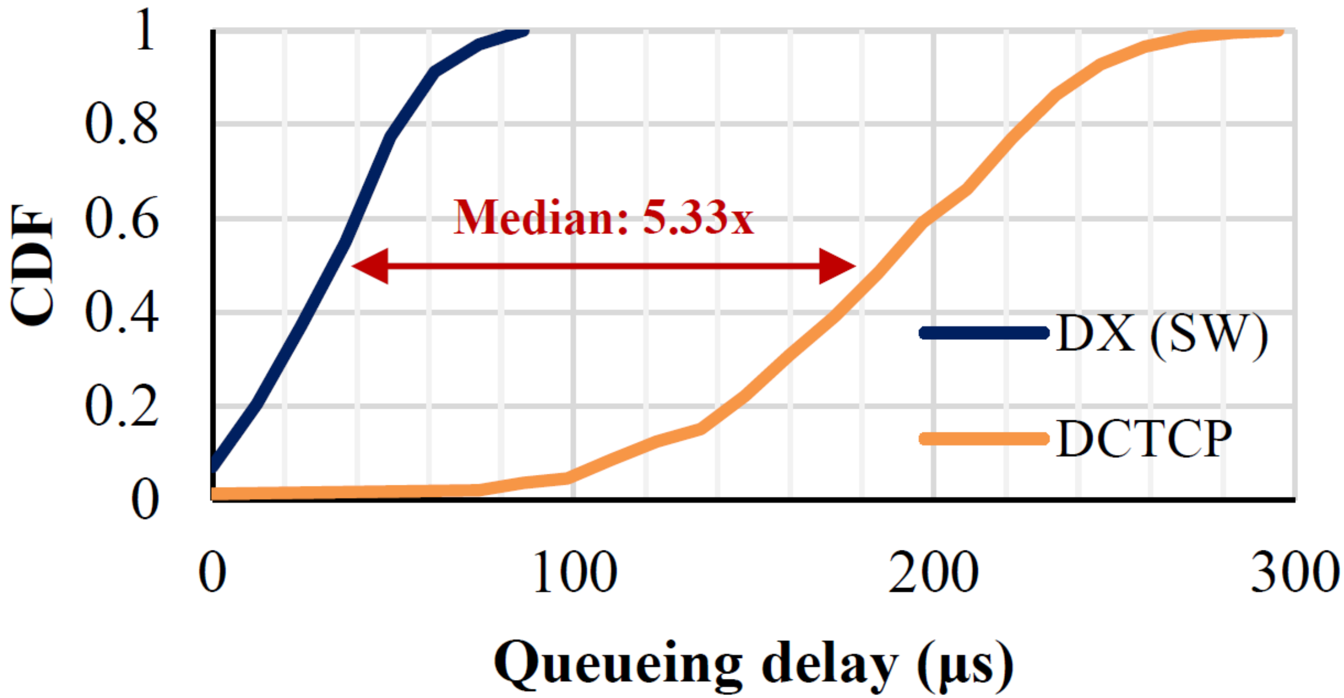
- More in our paper
  - Queueing delay and utilization with 10/20/30 senders
  - Flow throughput convergence
  - Impact of measurement noise to headroom
  - Fairness and throughput stability

Testbed experiment setup

- Two senders share a bottleneck link (1Gbps/10Gbps)
- Senders generate DX/DCTCP traffic to fully utilize the link
- We measure and compare the queue length of DX/DCTCP

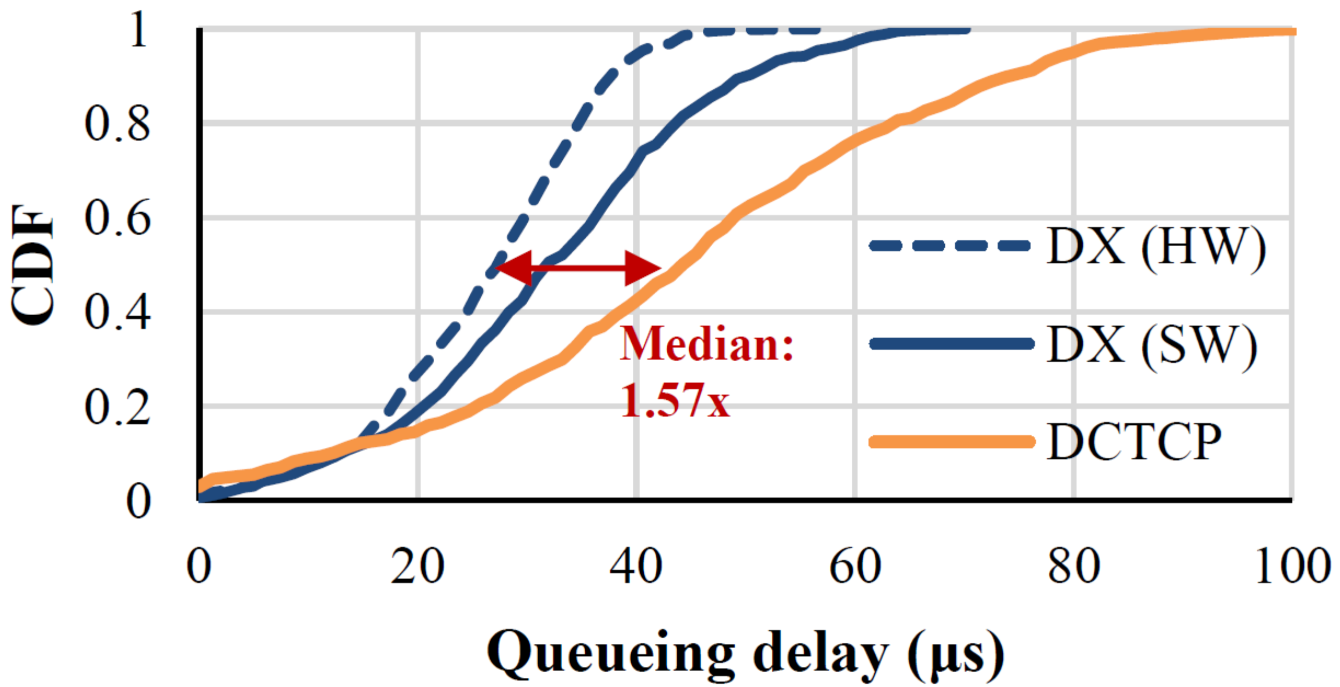


Testbed experiment result at 1Gbps



DX reduces median queuing delay by 5.33 times from DCTCP

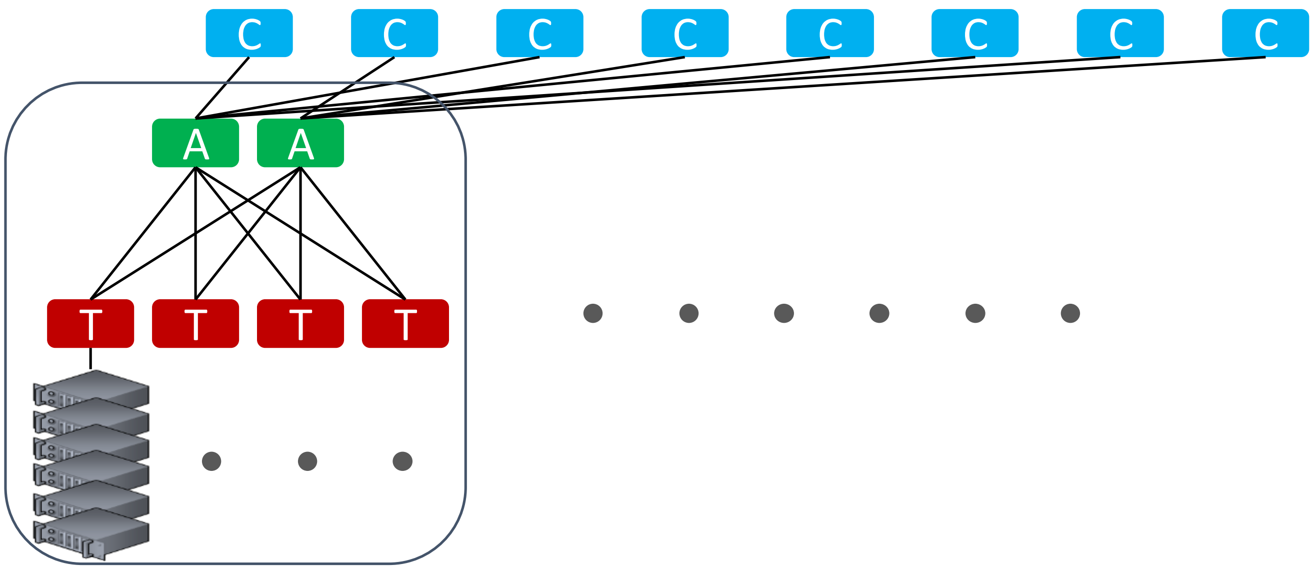
Testbed experiment result at 10Gbps



Hardware timestamping achieves further queueing delay reduction

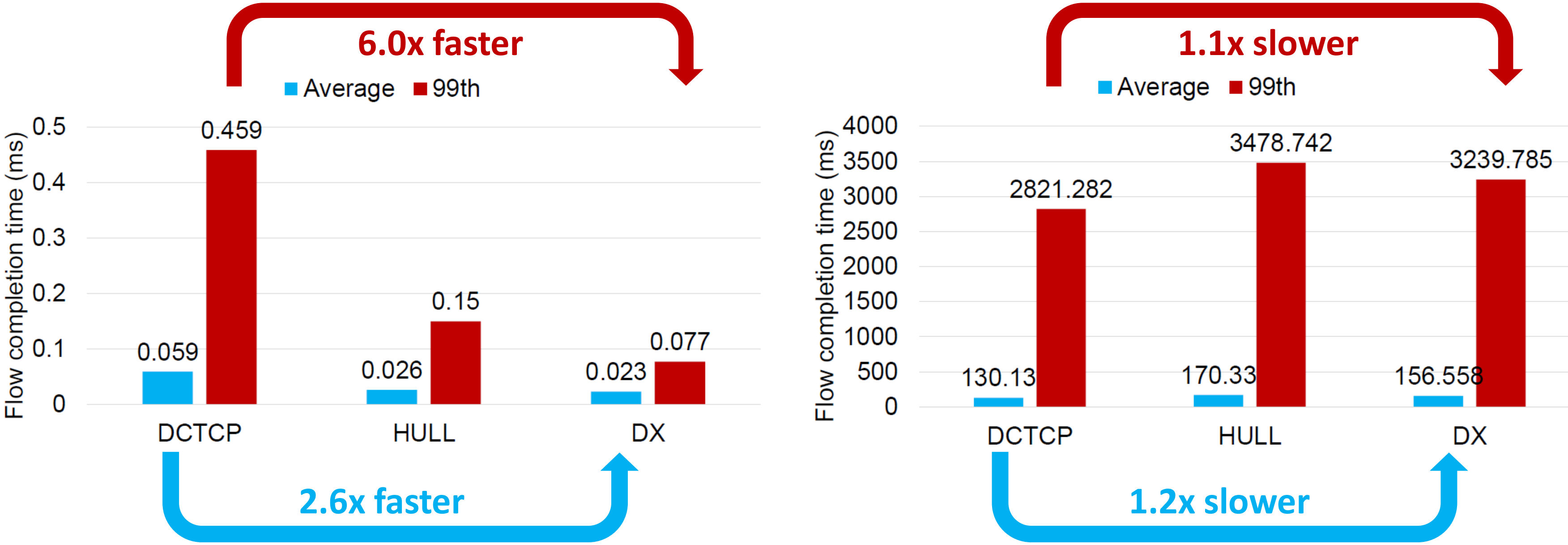
Simulation with datacenter workload

- Topology
  - A 3-tier fat tree with 192 nodes and 56 switches



- Workload
  - Empirical web search workload from production datacenter

FCT of search workload simulation



0KB - 10KB

10MB -

DX effectively reduces the completion time of small flows

Conclusion

- The quality of congestion feedback fundamentally governs the performance of congestion control
- We propose to use latency feedback in datacenters with support from our SW/HW timestamping techniques
- We develop DX, a new latency-based congestion control, which achieves 5.3 times (1Gbps) and 1.6 times (10Gbps) queueing delay reduction than ECN-based DCTCP