Less is More: Trading a little Bandwidth for Ultra-Low Latency in the Data Center

Mohammad Alizadeh, Abdul Kabbani, Tom Edsall,
Balaji Prabhakar, Amin Vahdat, and Masato Yasuda

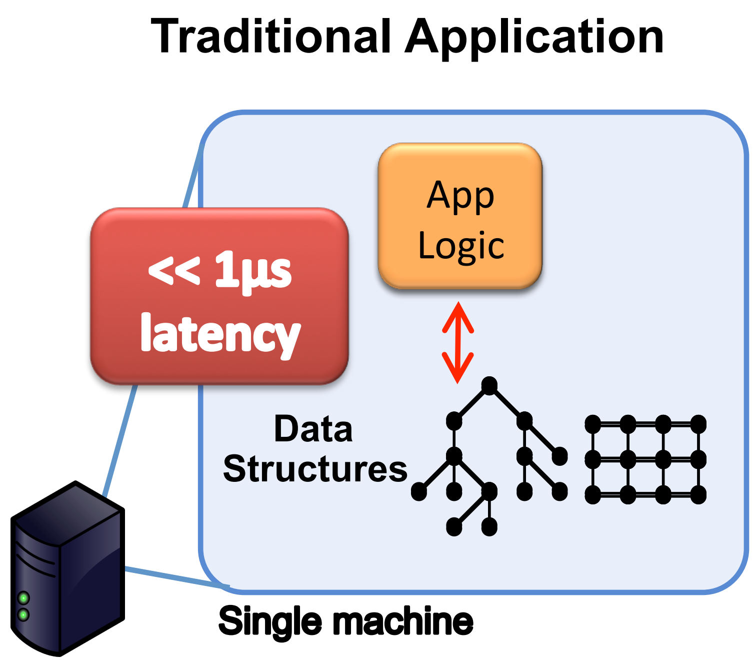
Latency in Data Centers

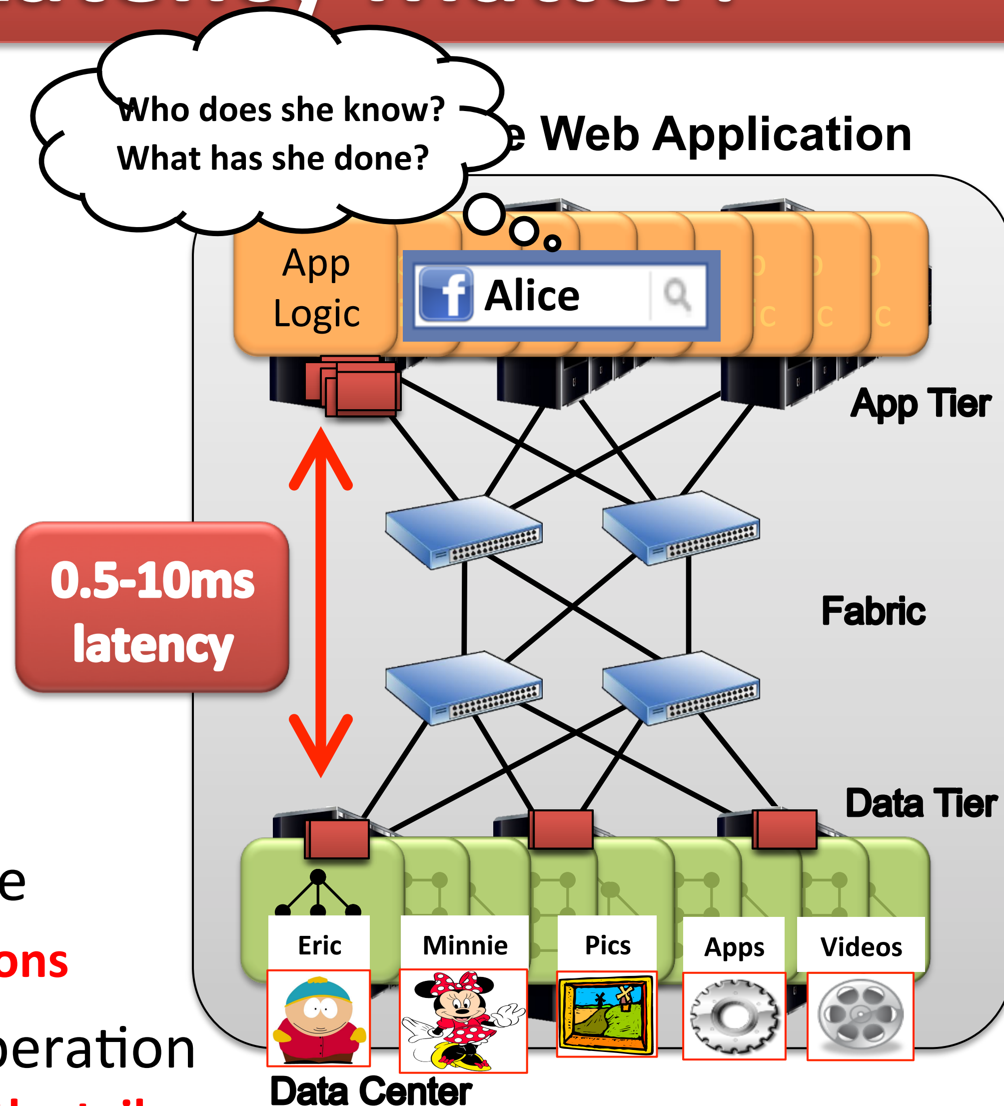
- Latency is becoming a primary performance metric in DC

- Low latency applications
  - High-frequency trading
  - High-performance computing
  - Large-scale web applications
  - RAMClouds (want < 10μs RPCs)

- Desire predictable low-latency delivery of individual packets

Why Does Latency Matter?





- Latency limits data access rate
  - ➢ **Fundamentally limits applications**
- Possibly 1000s of RPCs per operation
  - ➢ **Microseconds matter, even at the tail (e.g., 99.9th percentile)**

Reducing Latency

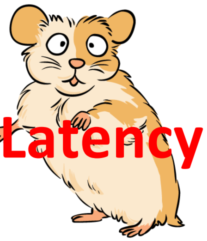
- Software and hardware are improving
  - Kernel bypass, RDMA; RAMCloud: software processing ~1μs
  - Low latency switches forward packets in a few 100ns
  - **Baseline fabric latency (propagation, switching) under 10μs is achievable.**
- Queuing delay: random and traffic dependent
  - Can easily reach 100s of microseconds or even milliseconds
    - One 1500B packet = 12μs @ 1Gbps

Goal: Reduce queuing delays to zero.

Low Latency AND High Throughput

Data Center Workloads:

- Short messages [100B-10KB] → **Low Latency**

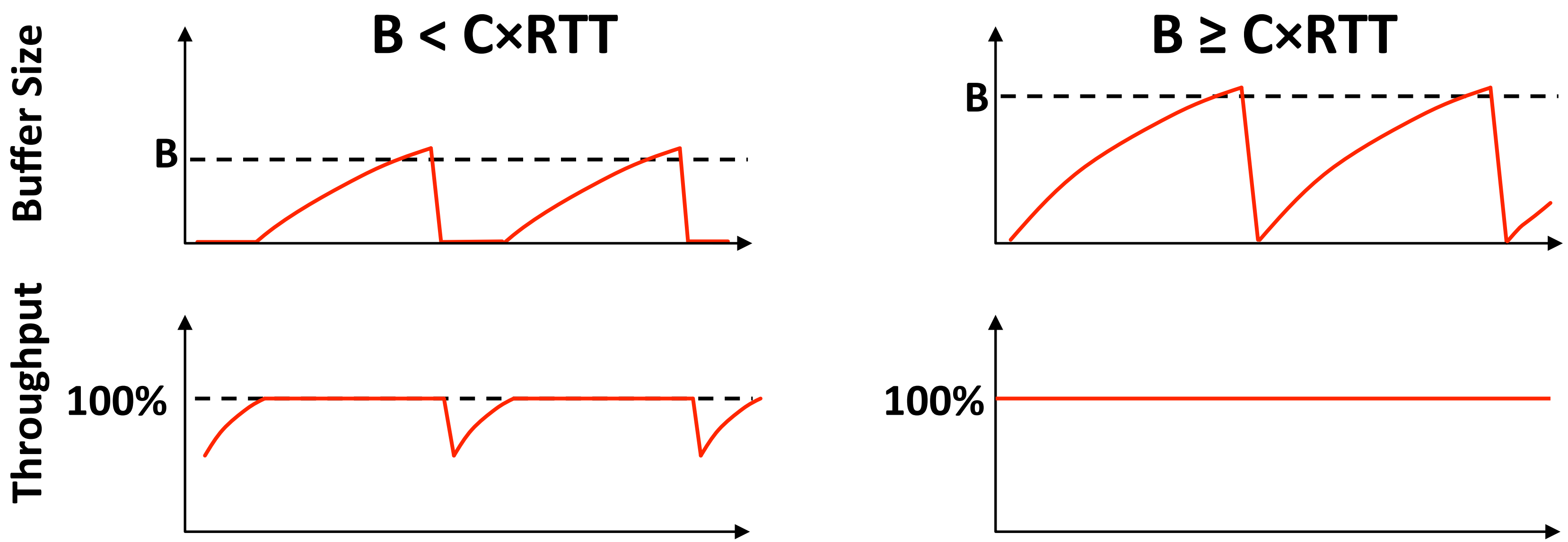


- Large flows [1MB-100MB] → **High Throughput**

We want baseline fabric latency AND high throughput.

Why do we need buffers?

- Main reason: to create "slack"
  - Handle temporary oversubscription
  - Absorb TCP's rate fluctuations as it discovers path bandwidth
- **Example: Bandwidth-delay product rule of thumb**
  - A single TCP flow needs **C×RTT** buffers for **100% Throughput.**



Overview of our Approach

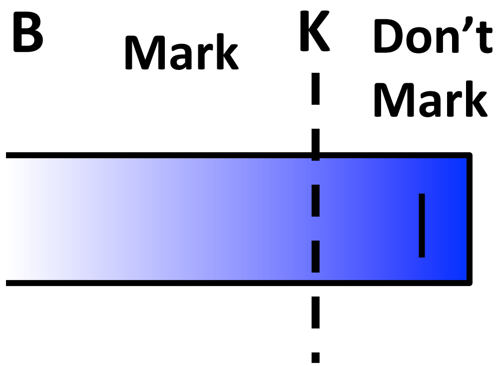
Main Idea

- Use "phantom queues"
  - Signal congestion **before** any queuing occurs
- Use DCTCP [SIGCOMM'10]
  - Mitigate throughput loss that can occur without buffers
- Use hardware pacers
  - Combat burstiness due to offload mechanisms like LSO and Interrupt coalescing

Review: DCTCP

Switch:

- Set ECN Mark when **Queue Length > K.**

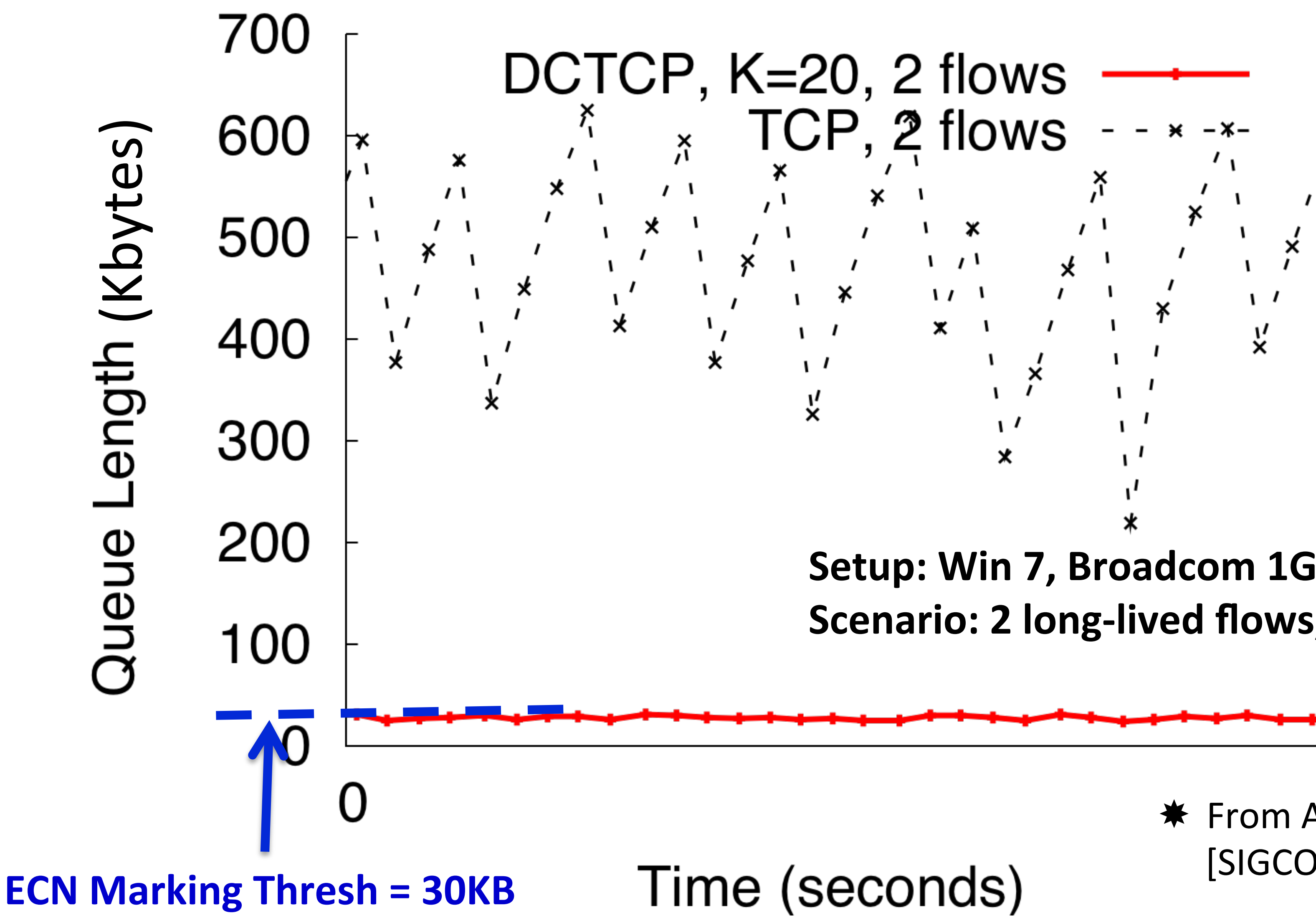


Source:

- React in proportion to the **extent** of congestion → less fluctuations
  - Reduce window size based on **fraction** of marked packets.

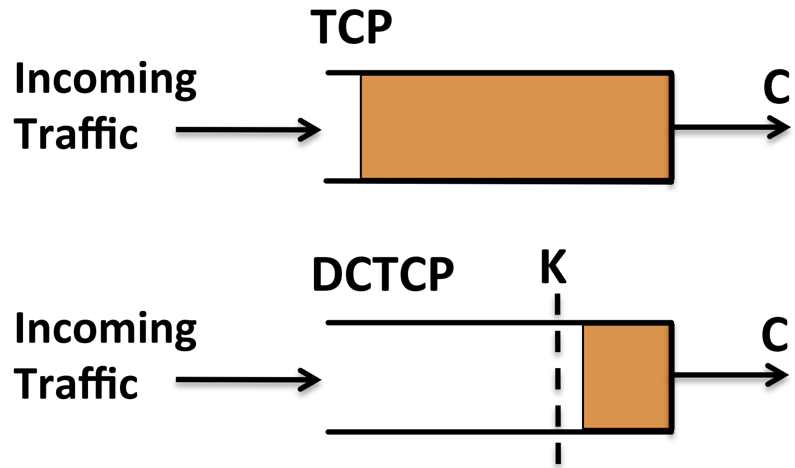
ECN Marks	TCP	DCTCP
1 0 1 1 1 1 0 1 1 1	**Cut window by 50%**	**Cut window by 40%**
0 0 0 0 0 0 0 0 0 1	**Cut window by 50%**	**Cut window by 5%**

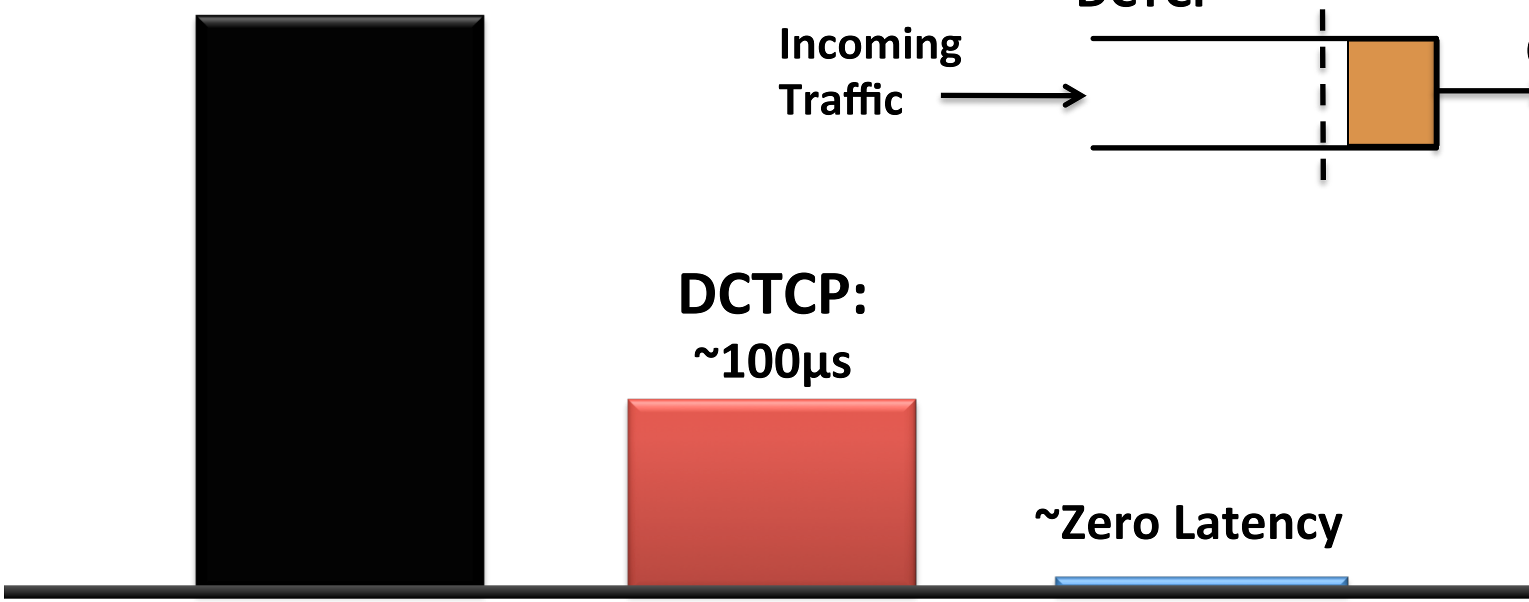
DCTCP vs TCP



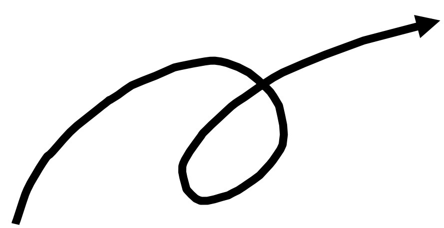
✸ From Alizadeh et al [SIGCOMM’10]

Achieving Zero Queuing Delay



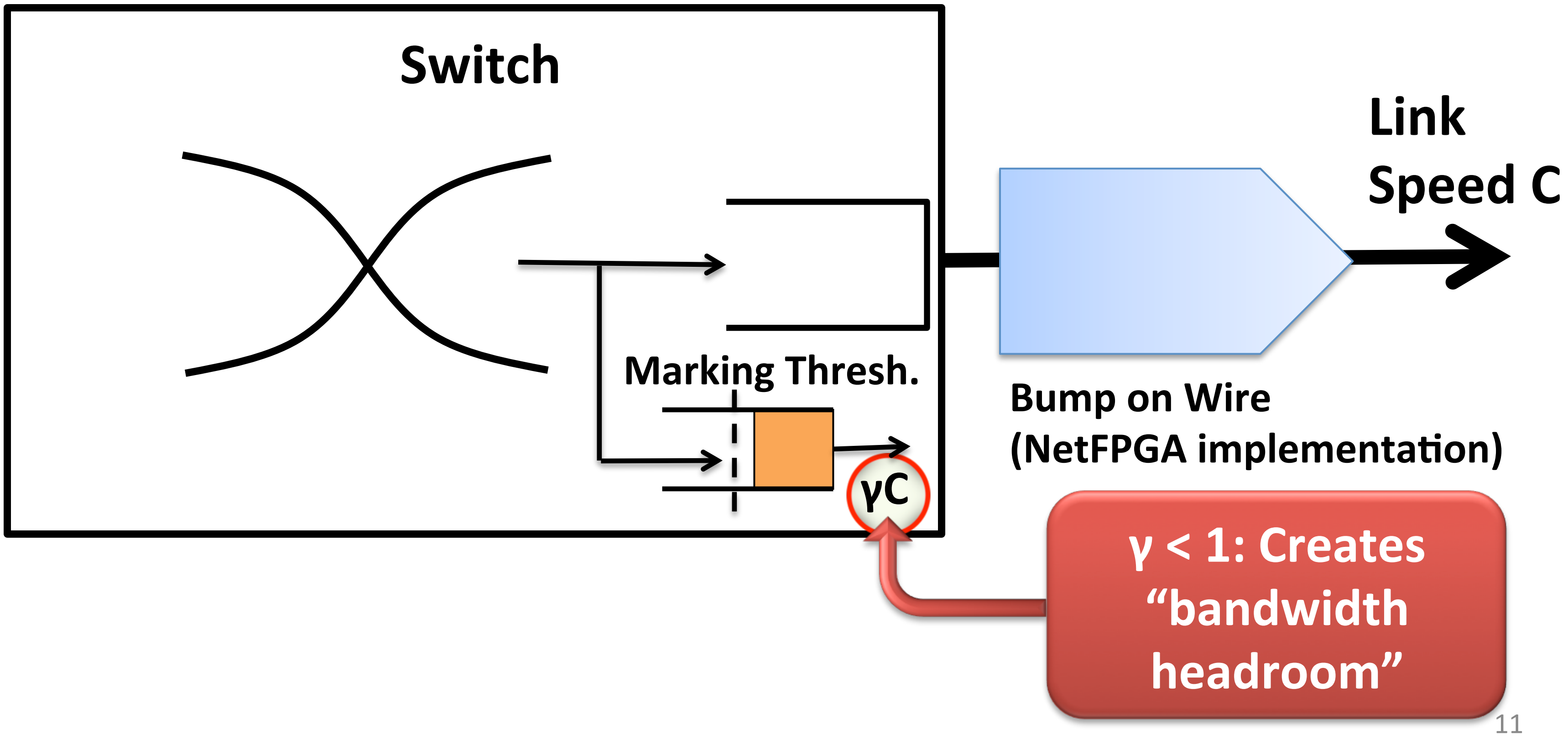


How do we get this?



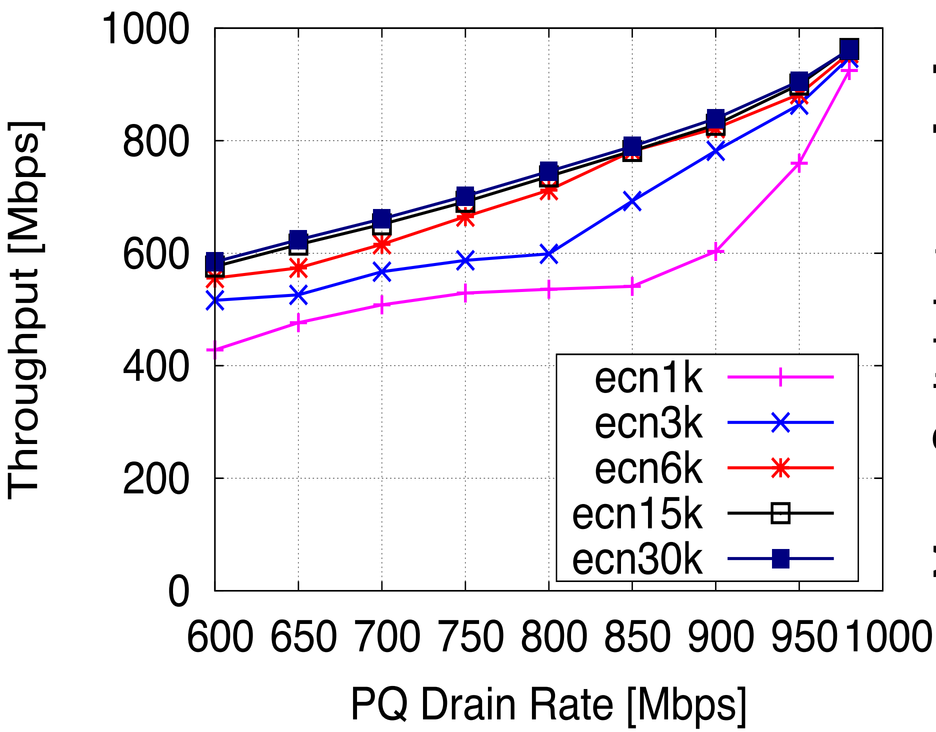
Phantom Queue

- Key idea:
  - Associate congestion with link utilization, not buffer occupancy
  - **Virtual Queue** (Gibbens & Kelly 1999, Kunniyur & Srikant 2001)

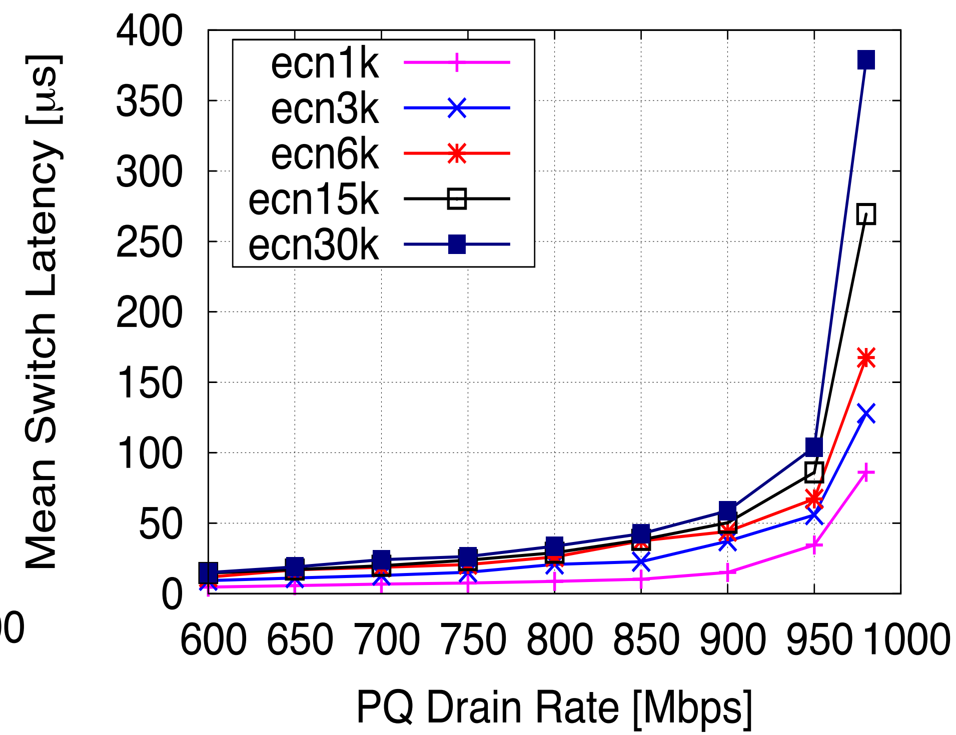


Throughput & Latency vs. PQ Drain Rate

Throughput



Switch latency (mean)

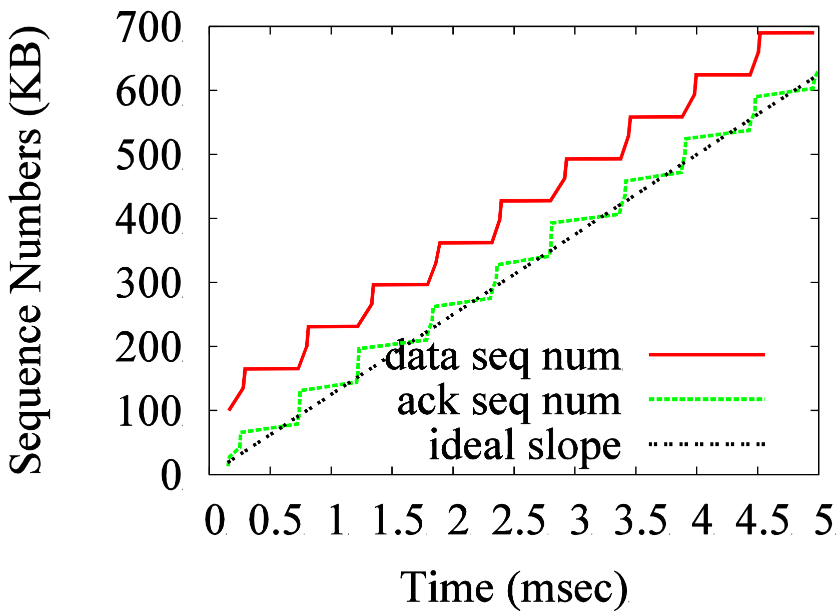


The Need for Pacing

- TCP traffic is very bursty
  - Made worse by CPU-offload optimizations like Large Send Offload and Interrupt Coalescing
  - Causes spikes in queuing, increasing latency

Example. 1Gbps flow on 10G NIC

65KB bursts every 0.5ms



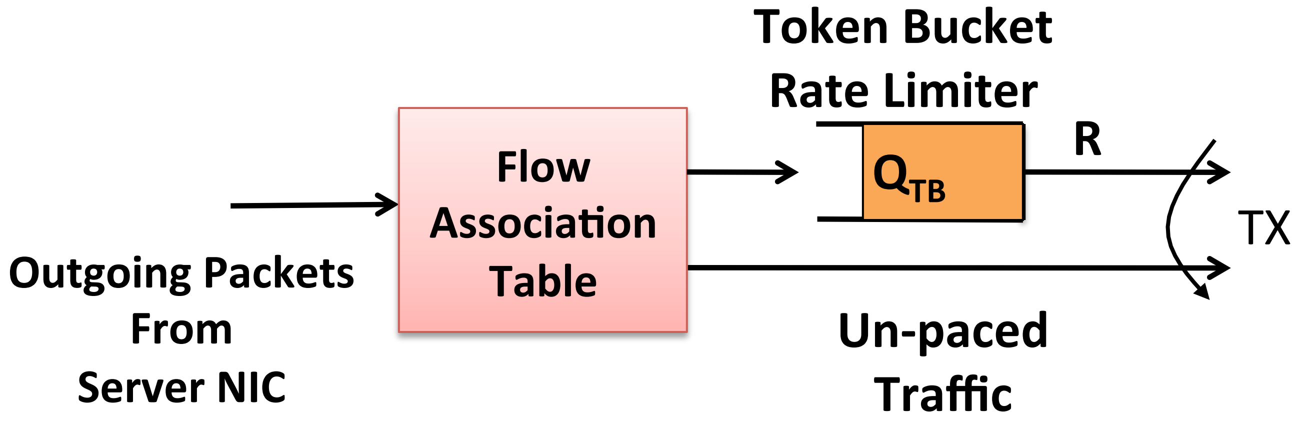
Impact of Interrupt Coalescing

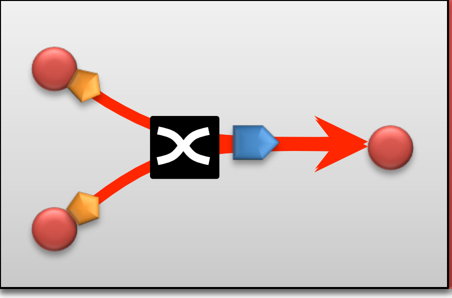
Interrupt Coalescing	Receiver CPU (%)	Throughput (Gbps)	Burst Size (KB)
disabled	99	7.7	67.4
rx-frames=2	98.7	9.3	11.4
rx-frames=8	75	9.5	12.2
rx-frames=32	53.2	9.5	16.5
rx-frames=128	30.7	9.5	64.0

More Interrupt Coalescing → **Lower CPU Utilization & Higher Throughput** → **More Burstiness**

Hardware Pacer Module

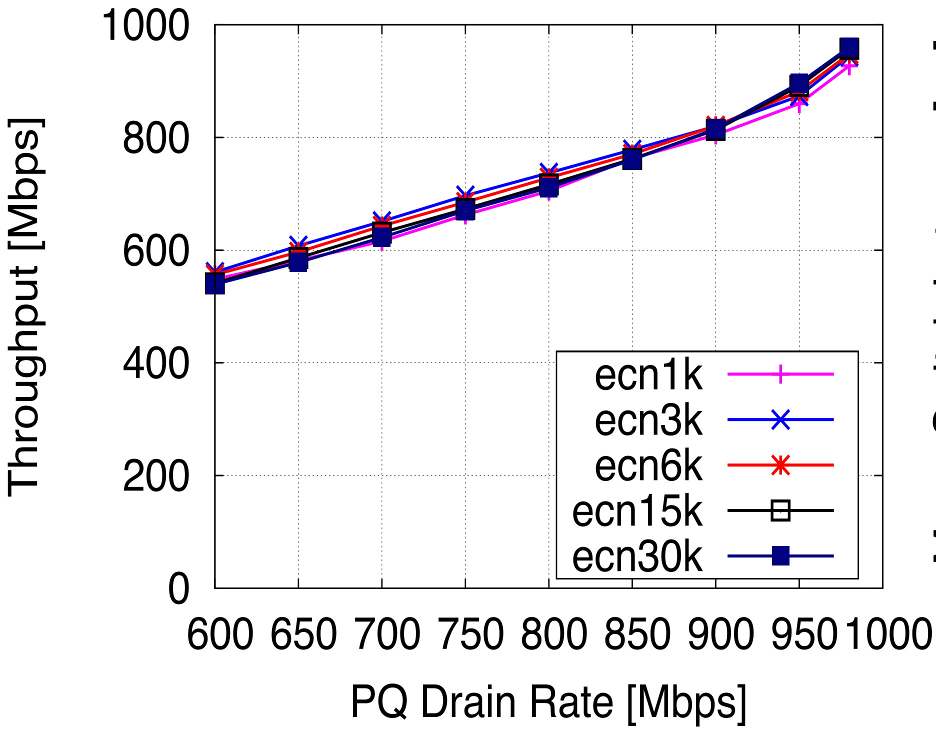
- Algorithmic challenges:
  - **At what rate to pace?**
    - **Found dynamically:** $R \leftarrow (1-\eta)R + \eta R_{measured} + \beta Q_{TB}$
  - **Which flows to pace?**
    - **Elephants:** On each ACK with ECN bit set, begin pacing the flow with some probability.



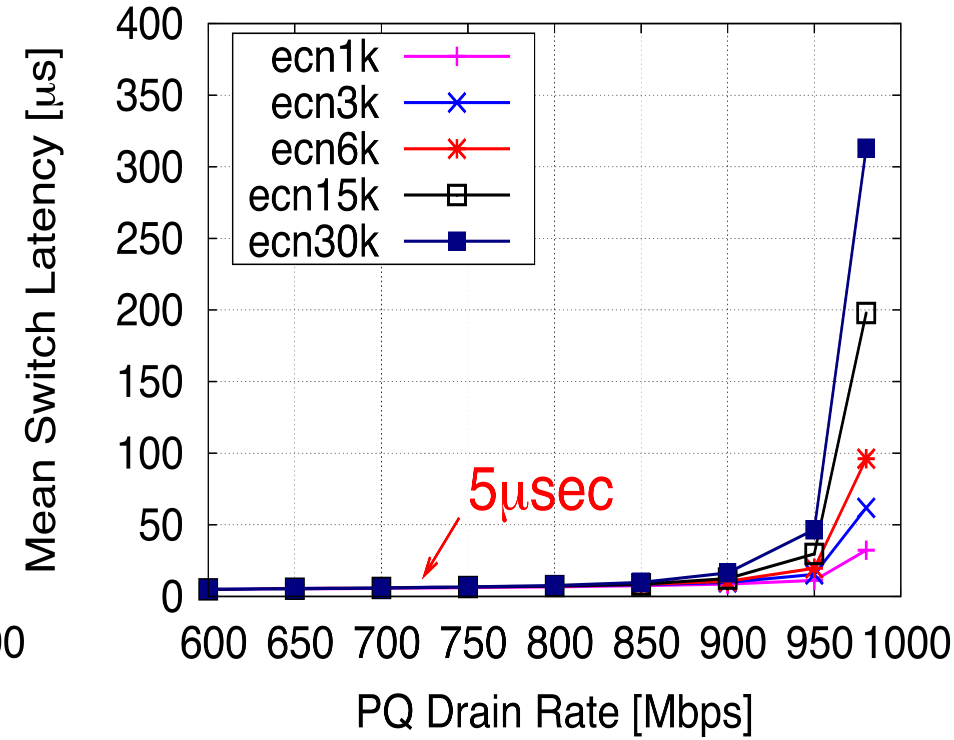


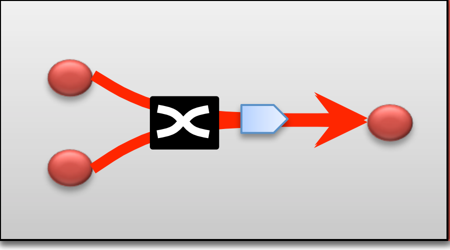
Throughput & Latency vs. PQ Drain Rate (with Pacing)

Throughput

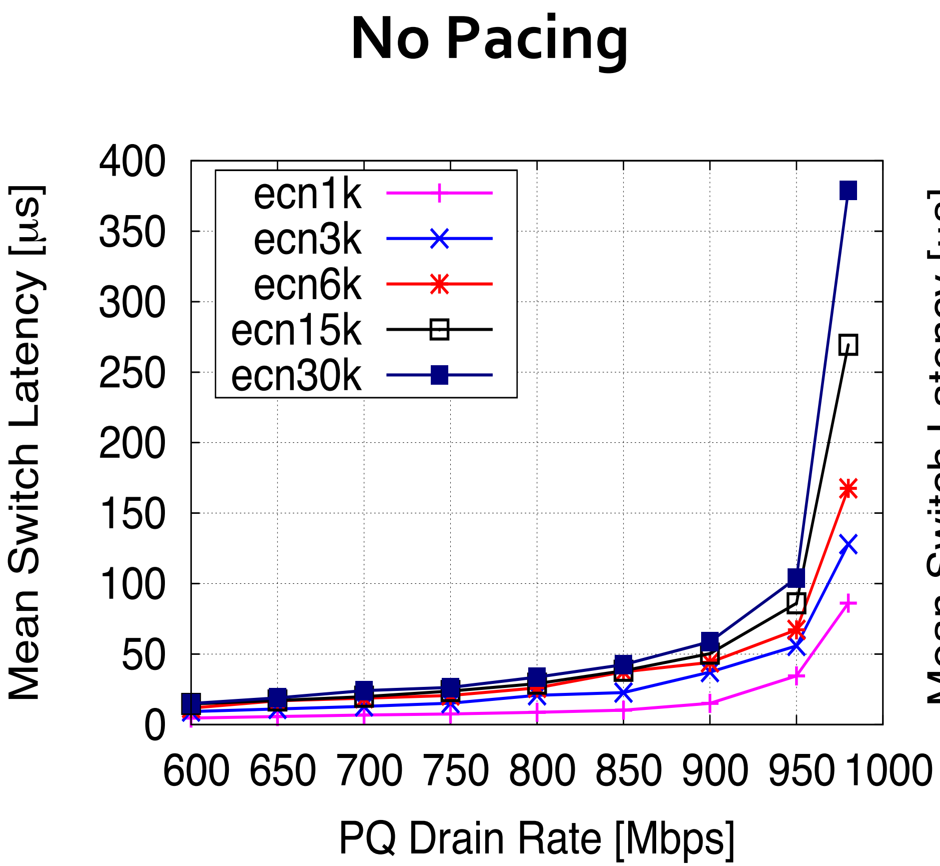


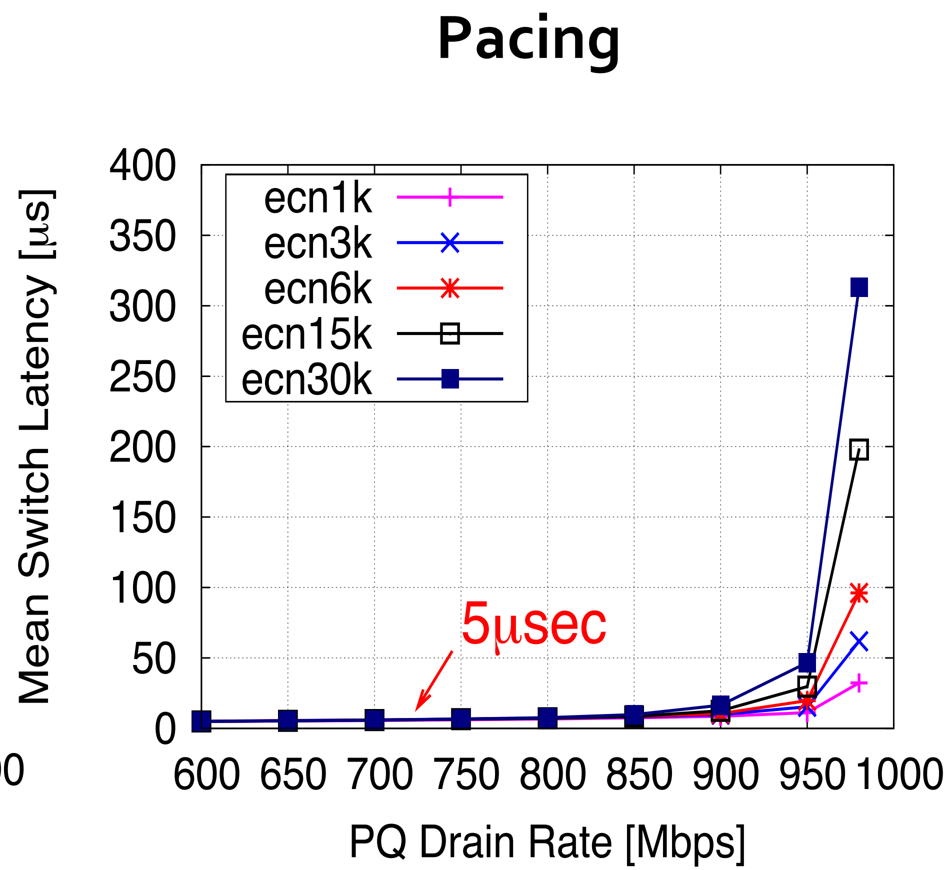
Switch latency (mean)



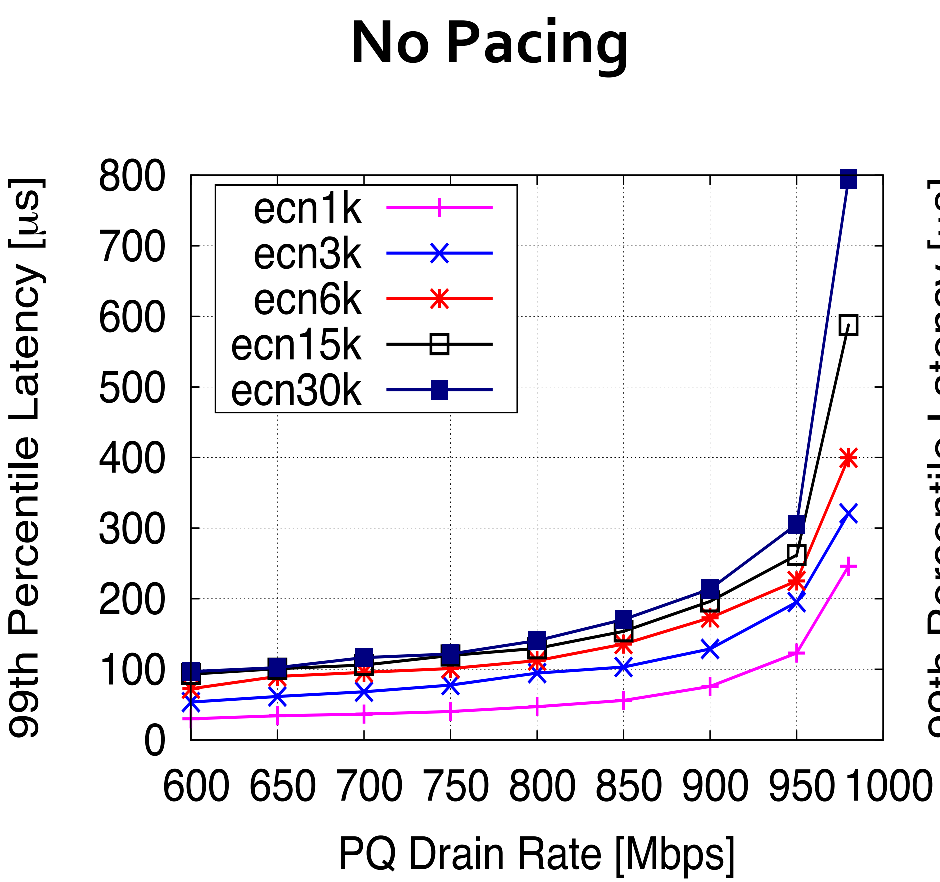


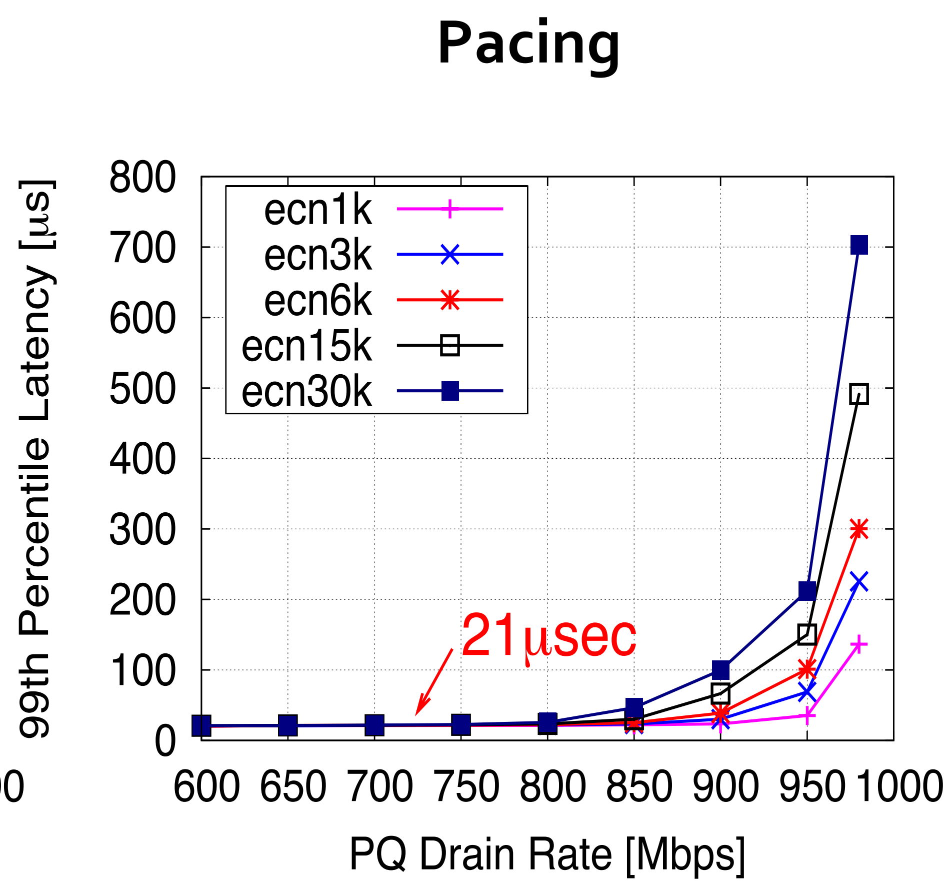
No Pacing vs Pacing
(Mean Latency)



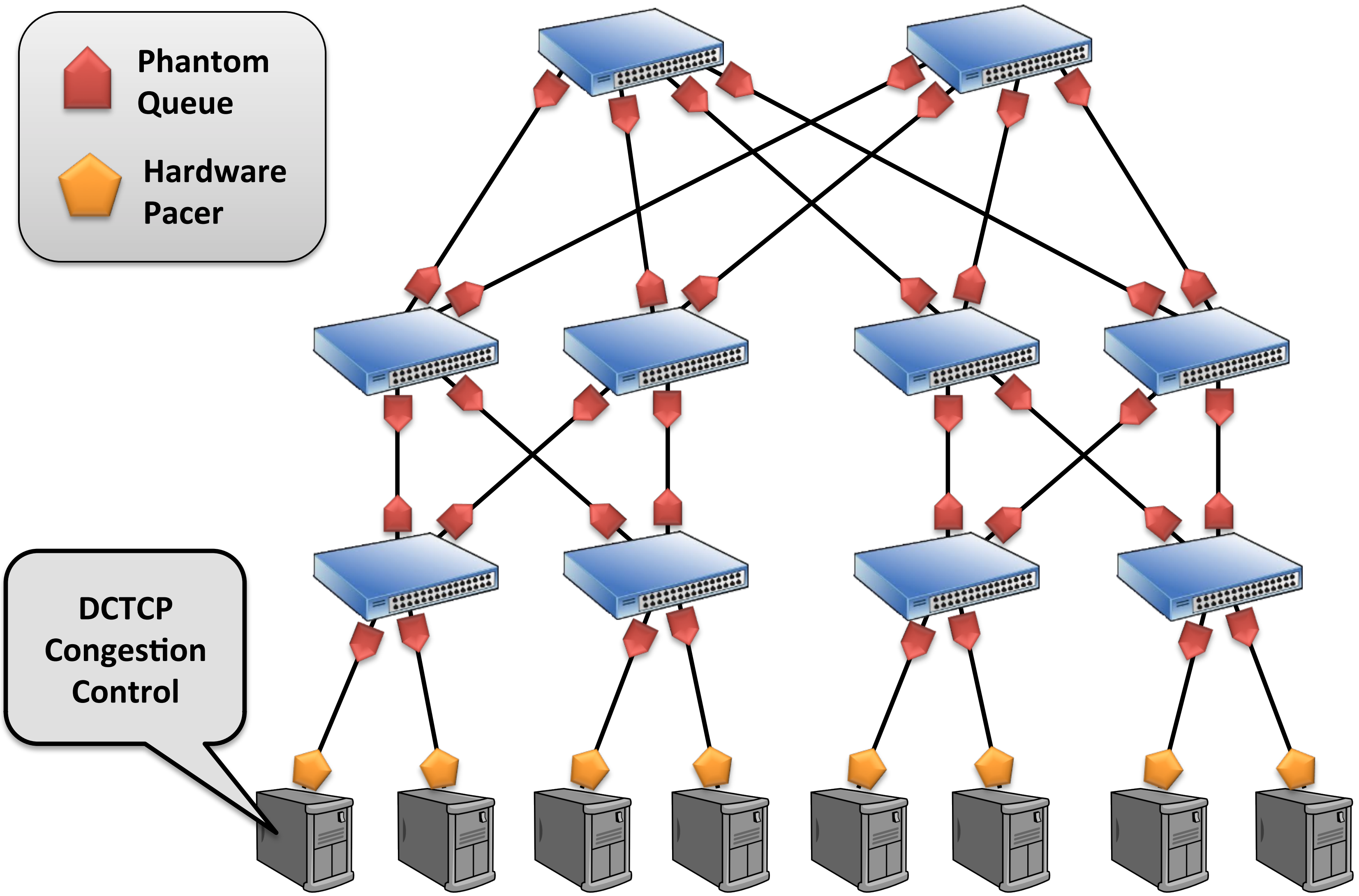


No Pacing vs Pacing (99th Percentile Latency)



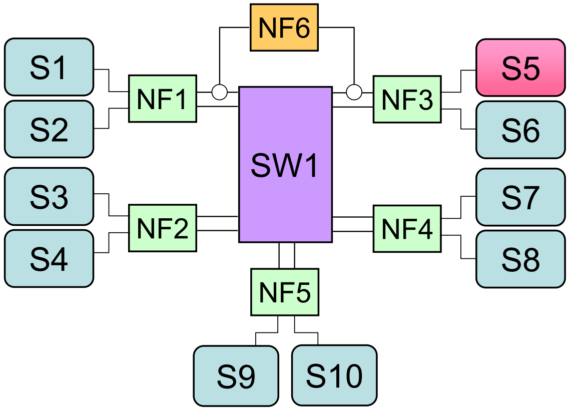


The HULL Architecture



Implementation and Evaluation

- Implementation
  - PQ, Pacer, and Latency Measurement modules implemented in NetFPGA
  - DCTCP in Linux (patch available online)



- Evaluation
  - 10 server testbed
  - Numerous micro-benchmarks
    - Static & dynamic workloads
    - Comparison with ‘ideal’ 2-priority QoS scheme
    - Different marking thresholds, switch buffer sizes
    - Effect of parameters
  - Large-scale ns-2 simulations

Dynamic Flow Experiment
20% load

- 9 senders → 1 receiver (80% 1KB flows, 20% 10MB flows).

Load: 20%	**Switch Latency (µs)**		**10MB FCT (ms)**	
	Avg	99th	Avg	99th
TCP	111.5	1,224.8	110.2	349.6
DCTCP-30K	38.4	295.2	**106.8**	**301.7**
DCTCP-6K-Pacer	6.6	59.7	111.8	320.0
DCTCP-PQ950-Pacer	**2.8**	**18.6**	125.4	359.9

Conclusion

- The HULL architecture combines
  - Phantom queues
  - DCTCP
  - Hardware pacing

- We trade some bandwidth (that is relatively plentiful) for significant latency reductions (often 10-40x compared to TCP and DCTCP).

Thank you!