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 $C \times RTT$ buffers for **100% Throughput**.
Overview of our Approach

• 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
**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.

<table>
<thead>
<tr>
<th>ECN Marks</th>
<th>TCP</th>
<th>DCTTCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 1 1 1 0 1 1 1</td>
<td>Cut window by 50%</td>
<td>Cut window by 40%</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 1</td>
<td>Cut window by 50%</td>
<td>Cut window by 5%</td>
</tr>
</tbody>
</table>
Setup:
Win 7, Broadcom 1Gbps Switch

Scenario:
2 long-lived flows,

ECN Marking Thresh = 30KB

DCTCP vs TCP

DCTCP, K=20, 2 flows
TCP, 2 flows

Setup: Win 7, Broadcom 1Gbps Switch
Scenario: 2 long-lived flows,

From Alizadeh et al [SIGCOMM’10]
Achieving Zero Queuing Delay

**TCP:**
~1–10ms

**DCTCP:**
~100μs

How do we get this?
• Key idea:
  – Associate congestion with link utilization, not buffer occupancy
  – **Virtual Queue** (Gibbens & Kelly 1999, Kunniyur & Srikant 2001)

\[ \gamma < 1 : \text{Creates “bandwidth headroom”} \]
Throughput & Latency vs. PQ Drain Rate

Throughput

Switch latency (mean)

Throughput [Mbps]

Mean Switch Latency [µs]

600 650 700 750 800 850 900 950 1000

PQ Drain Rate [Mbps]

PQ Drain Rate [Mbps]

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k
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

![Graph showing sequence numbers over time](image)
# Impact of Interrupt Coalescing

<table>
<thead>
<tr>
<th>Interrupt Coalescing</th>
<th>Receiver CPU (%)</th>
<th>Throughput (Gbps)</th>
<th>Burst Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>disabled</td>
<td>99</td>
<td>7.7</td>
<td>67.4</td>
</tr>
<tr>
<td>rx-frames=2</td>
<td>98.7</td>
<td>9.3</td>
<td>11.4</td>
</tr>
<tr>
<td>rx-frames=8</td>
<td>75</td>
<td>9.5</td>
<td>12.2</td>
</tr>
<tr>
<td>rx-frames=32</td>
<td>53.2</td>
<td>9.5</td>
<td>16.5</td>
</tr>
<tr>
<td>rx-frames=128</td>
<td>30.7</td>
<td>9.5</td>
<td>64.0</td>
</tr>
</tbody>
</table>

More Interrupt Coalescing → Lower CPU Utilization & Higher Throughput → More Burstiness
• Algorithmic challenges:
  – At what rate to pace?
    • Found dynamically: \[ R \leftarrow (1 - \eta)R + \eta R_{\text{measured}} + \beta Q_{TB} \]
  – 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**

- Throughput vs. PQ Drain Rate (with Pacing)

**Switch latency (mean)**

- Mean Switch Latency vs. PQ Drain Rate

Legend:
- ecn1k
- ecn3k
- ecn6k
- ecn15k
- ecn30k

5μsec
No Pacing vs Pacing (Mean Latency)

No Pacing

Pacing

Mean Switch Latency [µs]

PQ Drain Rate [Mbps]

ecn1k  ecn3k  ecn6k  ecn15k  ecn30k

No Pacing vs Pacing (Mean Latency)

Mean Switch Latency [µs]

PQ Drain Rate [Mbps]

ecn1k  ecn3k  ecn6k  ecn15k  ecn30k

5µsec
No Pacing vs Pacing (99th Percentile Latency)

No Pacing

PQ Drain Rate [Mbps]

99th Percentile Latency [µs]

Pacing

PQ Drain Rate [Mbps]

99th Percentile Latency [µs]

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

21 µsec

No Pacing

Pacing

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

No Pacing

Pacing

ecn1k
ecn3k
ecn6k
ecn15k
ecn30k

21 µsec
The HULL Architecture

- Phantom Queue
- Hardware Pacer
- DCTCP Congestion Control
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).

<table>
<thead>
<tr>
<th>Load: 20%</th>
<th>Switch Latency (μs)</th>
<th>10MB FCT (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg</td>
<td>99&lt;sup&gt;th&lt;/sup&gt;</td>
</tr>
<tr>
<td>TCP</td>
<td>111.5</td>
<td>1,224.8</td>
</tr>
<tr>
<td>DCTCP-30K</td>
<td>38.4</td>
<td>295.2</td>
</tr>
<tr>
<td>DCTCP-6K-Pacer</td>
<td>6.6</td>
<td>59.7</td>
</tr>
<tr>
<td>DCTCP-PQ950-Pacer</td>
<td>2.8</td>
<td>18.6</td>
</tr>
</tbody>
</table>
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!