EyeQ: Practical Network Performance Isolation at the Edge

Vimalkumar Jeyakumar

Mohammad Alizadeh
Balaji Prabhakar
David Mazières

Changhoon Kim
Albert Greenberg
Once upon a time...
Once upon a time...
Once upon a time...
Once upon a time...
Performance Unpredictability

http://amistrongeryet.com/op_detail.jsp?op=gae_db_readCachedHandles_1&hoursAgo=24
Congestion Kills Predictability
Key Issue

Today’s transport (TCP/UDP) lacks predictability in sharing bandwidth
Status Quo is Insufficient
Status Quo is Insufficient

- **TCP**
  - Cannot force all to use TCP or agree on one TCP version!
  - Sharing is *per-flow*: not built for predictability

- **Performance Isolation with Per-tenant Queues**
  - State management complexity: >10k tenants, configuring queues on all links is an operational nightmare
  - WFQ/DRR does not ensure admissibility
Status Quo is Insufficient

- **TCP**
  - Cannot force all to use TCP or agree on one TCP version!
  - Sharing is per-flow: not built for predictability

- **Performance Isolation with Per-tenant Queues**
  - State management complexity: >10k tenants, configuring queues on all links is an operational nightmare
  - WFQ/DRR does not ensure admissibility
Where does Congestion Happen?

Ideal network fabric (one switch)

Shared 10Gb/s pipe

Server

VM

VM

VM

VM

VM

Tenant 1

Tenant 2
Where does Congestion Happen?

Ideal network fabric (one switch)

Shared 10Gb/s pipe

Server

VM

Tenant 1

Tenant 2

4 Apr 2013

NSDI 2013
Congestion Study on Windows Azure

Spine Layer

Leaf layer

< 3:1 oversub
Timescales: over 2 weeks, 99.9\textsuperscript{th} percentile = several minutes

Hottest storage cluster: 1000x more drops at the Edge, than Core.

16 of 17 clusters: 0 drops in the Core.
Customer specifies capacity of the virtual NIC. No traffic matrix. (Hose Model)

Provider: assures near dedicated performance.

EyeQ is deployable today at the Edge.
EyeQ’s Key Contribution: Simplicity

• Observation
  – Network Congestion predominantly occurs at the Edge (Hypervisor / Top of Rack)

• Consequences: Simplicity
  – Distributed, end-to-end bandwidth allocation
    • Amenable to NIC-based implementation
  – Network need not be tenant aware

• Implementation
  – High speed in software at 10Gb/s
Decentralized Scheduling

EyeQ Shim Layer
In the trusted Domain
(Hypervisor/NIC)
Decentralized Scheduling

- 2Gb/s VM
- 5Gb/s
- 2Gb/s
- 8Gb/s VM
- 5Gb/s
- 10Gb/s pipe
- (min) Rate Guarantees
- 2Gb/s VM
- 8Gb/s
- 8Gb/s VM
- 8Gb/s
Decentralized Scheduling
Decentralized Scheduling
Decentralized Scheduling
Work Conserving Allocations

2Gb/s VM

2Gb/s VM

8Gb/s VM

1Gb/s

1Gb/s

5Gb/s

5Gb/s

Spare capacity

RX Module

VM 2Gb/s

VM 8Gb/s

VM 8Gb/s

VM 8Gb/s
Work Conserving Allocations

2Gb/s VM  

2Gb/s VM  

8Gb/s VM  

2Gb/s VM  

2.5Gb/s  

2.5Gb/s VM  

5Gb/s VM  

5Gb/s VM  

2Gb/s VM  

8Gb/s VM  

8Gb/s VM  

8Gb/s VM
Transmit/Receive Modules

Per-destination rate limiters:
only if dest. is congested... bypass otherwise
Transmit/Receive Modules

RCP: Rate feedback (R) every 10kB (no per-source state needed)

Feedback pkt Rate: 1Gb/s

Per-destination rate limiters:
only if dest. is congested... bypass otherwise
Timescales Matter

• Fast convergence important
  – Switches only have few MB (milliseconds) worth of buffering before they drop packets

• RCP’s worst-case convergence time
  – N long lived flows competing for a single bottleneck: few milliseconds.
  – Usually few 100 microseconds.
But what if the Core gets congested?

How? → Transient failures or ECMP collisions

Case 1: Mild network congestion
- Use ECN for graceful fallback
  - Per receiver-VM max-min sharing
  - Congestion detector: multiplicative decrease on advertised rate on receiving ECN

Case 2: Severe network congestion (unlikely!)
- Multiplicative decrease (rate limiter timeout)
Software Prototype

Linux Kernel Module (qdisc)

Windows Filter Driver (in VMSwitch)

• Non-intrusive: no changes to applications or existing network stack. Works even with UDP.

• ~1700 lines of code

Linux Kernel Module is Open-Source

• Full system and documentation at http://jvimal.github.com/eyeq

• Fully functional version in Mininet to play with 😊
**High speed software rate limiters**

Single shared queue increases lock contention

- High CPU overhead
- High packet latency
- Controlled burst

Packets on the wire
Parallel transmit path

Split queue to per-cpu queues
• Lower CPU overhead
• Lower packet latency
• Fairness across CPU queues
• Higher, but bounded burst

CPU

Lazily
Grab tokens

Token filling
clocked by packets

Packets on the wire
Rate Limiter Efficiency

Throughput

Single rate limiter at 5Gb/s. HTB succumbs at 9Gb/s.
Rate Limiter Efficiency

Input rate to rate limiter limited by end-to-end latency.

Request (on multiple CPUs)  5Gb/s  Response

Latency

Median latency reduced by 2x

1B synchronous request response loop.
Macro Evaluation: Memcached Latency

Each server has 10Gb/s link

12 Client Pool

4 Server Pool
Macro Evaluation: Memcached Latency

Each server has 10Gb/s link

External Load: 144k SET req/sec

12 Client Pool

4 Server Pool

Set 6kB objects
Load: 2.3Gb/s/server
Macro Evaluation: Memcached Latency

External Load: 144k SET req/sec

12 Client Pool

4 Server Pool

Each server has 10Gb/s link

Set 6kB objects
Load: 2.3Gb/s/server

UDP bursty 5Gb/s
0.5s to 1 server, chosen round robin. 0.5s sleep between bursts.
## Macro Evaluation: Memcached Latency

Each server has 10Gb/s link

<table>
<thead>
<tr>
<th>Scenario</th>
<th>50\textsuperscript{th}</th>
<th>99.9\textsuperscript{th}</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline (Linux 3.4)</td>
<td>98us</td>
<td>666us</td>
<td>144kreq/s</td>
</tr>
<tr>
<td>Without Interference + EyeQ</td>
<td>100us</td>
<td>630us</td>
<td>144kreq/s</td>
</tr>
<tr>
<td>With Interference</td>
<td>4127us</td>
<td>&gt;10\textsuperscript{6}us</td>
<td>144kreq/s</td>
</tr>
<tr>
<td>With Interference + EyeQ</td>
<td>102us</td>
<td>750us</td>
<td>144kreq/s</td>
</tr>
</tbody>
</table>

Set 6kB objects
Load: 2.3Gb/s/server

**Scenario**: Set 6kB objects
Load: 2.3Gb/s/server

**12 Client Pool**

**4 Server Pool**
Thank you!

EyeQ: An edge-based flow scheduler for the data center...

to partition bandwidth in a simple and predictable way.

http://jvimal.github.com/eyeq
jvimal@stanford.edu