Investigating Efficient Real-time Performance Guarantees on Storage Networks

Andrew Shewmaker
shewa@soe.ucsc.edu

Department of Computer Science
University of California Santa Cruz

October 19, 2010

Motivation

Goals of datacenters

- serve many users
- process petabytes of data

Design of datacenters

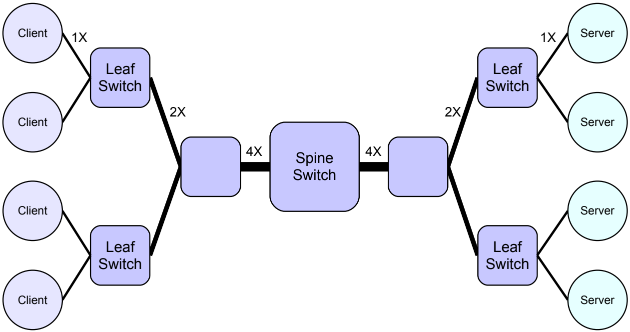
- use rules of thumb
- over-provision

An ad hoc approach creates marginal storage systems that cost more than necessary. A better system would be able to guarantee each user the performance they need from the CPUs, memory, disks, and network.

1. Introduction
2. RAD on Networks (Radon)
3. Conclusion

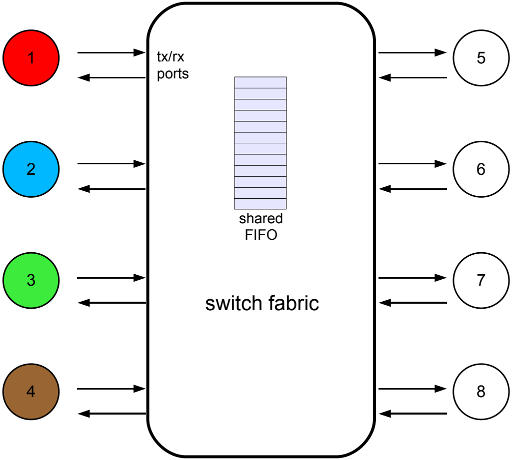
A Canonical Storage Network

Fat-tree with full bisection bandwidth trunk capacity matches the sum of the outer branches



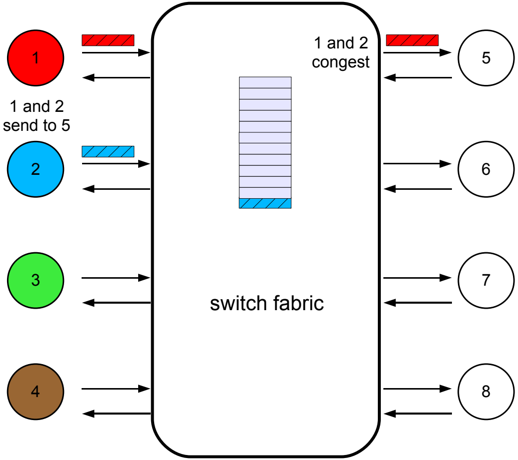
Congestion in a simple switch model

Each transmit port on the switch is a collision domain



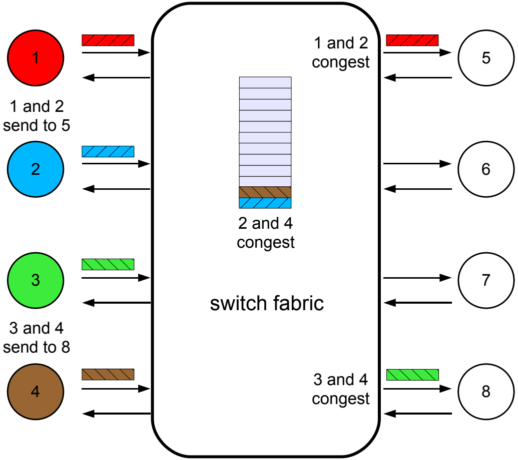
Congestion in a simple switch model

One of the packets destined for the same switch transmit port is delayed on the queue



Congestion in a simple switch model

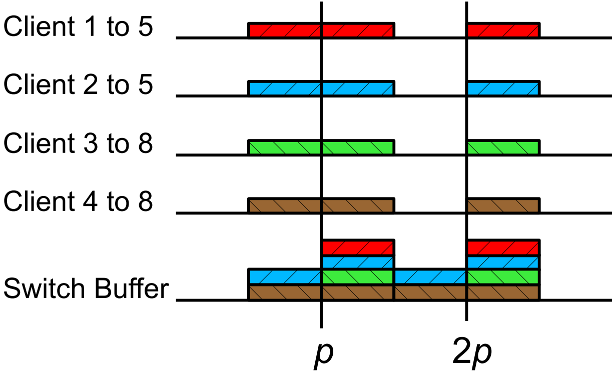
Delayed packets from unrelated streams affect each other on the queue



Worst Case Switch Buffer Size

Fixed rate, short period

$buffer = \sum_i rate_i \cdot period_i$



Le Boudec's and Thiran's Network Calculus

- Arrival and service curves
- Analyze using Min-plus algebra
- Switch's buffer requirement is $\sum_i burst_i$
- Bursts must be paid only once

In Practice, Simple Works ... sort of

- 1 Gbps, no problem
- 10 Gbps, do-able
- 40 Gbps, trouble
  - UDP achieves 21 Gbps
  - 50% CPU load
  - 60% system interrupt load
  - 16 MB host buffers

Does the 648 port QDR Mellanox switch possess 10 GB RAM?

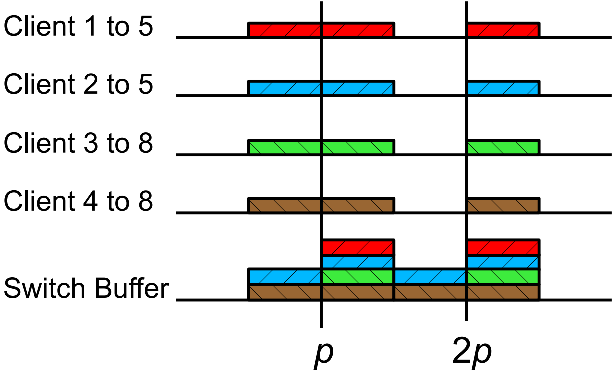
Comparison of Simple Solutions

Approach	Processor Time	Buffer Size
short periods	$\propto \mathit{period}_i$	$\sum_i \mathit{rate}_i \cdot \mathit{period}_i$
coarse-grained shaping	$\propto \frac{1}{\mathit{burst}_i}$	$\sum_i \mathit{burst}_i$
fine-grained shaping	100%	minimal

Worst Case Switch Buffer Size

Fixed rate, short period

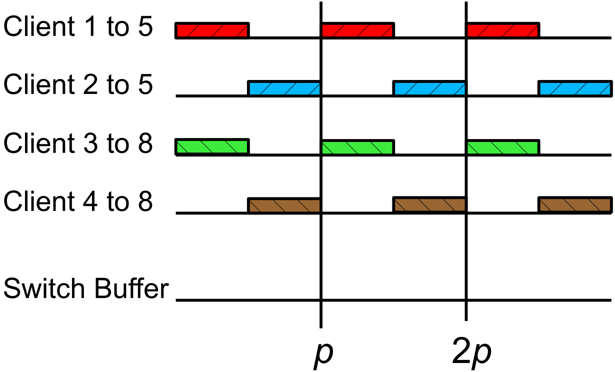
$buffer = \sum_i rate_i \cdot period_i$



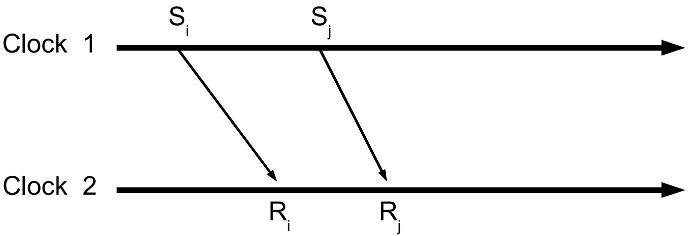
Best Case Switch Buffer Size

Fixed rate, short period

$buffer = \sum_i rate_i \cdot period_i$



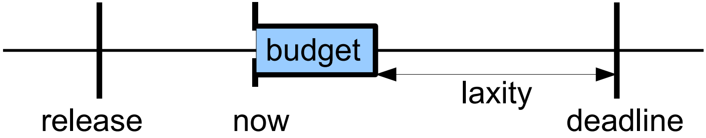
Network Resource Measurements



Relative Forward Delay $RFD_{i,j} = (R_j - R_i) - (S_j - S_i)$

While the clocks requires no synchronization, they should be stable and not reset between timestamps

Real-time Information



- Deadline is absolute
- Laxity is relative
- Budget gives global information

Rate-based Percent Budget scheduling

Flow Control Budget (in packets) $m_i = e_i / pktS$, where $pktS$ (s/packet) is the worst case packet service time

Congestion Control Adjust wait time between packets

- **Percent Budget** $\%budget = (1 - \%laxity) = \frac{e_i}{d-t}$
- **Packet Wait Time Target** $w_{op} = \frac{w_{min}}{\%budget}$
- **New Wait Time** $w_{k+1} = \min\left(w_{max},\ \max\left(w_{min},\ w_k - \frac{w_k - w_{op}}{2}\right)\right)$

Jump to window-based Radon

Evaluation of Radon

- Difficult to implement as kernel qdisc
- Difficult to implement in discrete event simulator
- Some success with userspace app on older hardware
- Nearly identical behavior to simple traffic shaping on newer hardware

Conclusion

- Traffic shaping guarantees at expense of CPU or switch memory
- Radon should be able to do better

Future Work

- Combine with other RAD-based resource schedulers
- Kernel level TCP or DCCP plugin
- Custom network simulator
- Continue evaluation using 10 Gigabit Ethernet and Infiniband
- Analyze interaction with TCP

Window-based Percent Budget scheduling

Flow Control Budget (in packets) $m_i = e_i / pktS$, where $pktS$ (s/packet) is the worst case packet service time

Congestion Control Adjust window size and offset

- **Percent Budget** $\%budget = (1 - \%laxity) = \frac{e_i}{d-t}$
- **Window Target** $w_{op} = (1 - \%laxity) \cdot w_{max}$
- **Size Change** $w_\Delta = \frac{-|w_k - w_{op}|}{2}$
- **Dispatch Offset** $w_{offset} = \frac{N_{obs}}{pktS} \cdot \text{rand}$

Where w_k is the current window size and N_{obs} is the depth of the bottleneck switch's queue modeled using observations of relative forward delay.

Jump to rate-based Radon