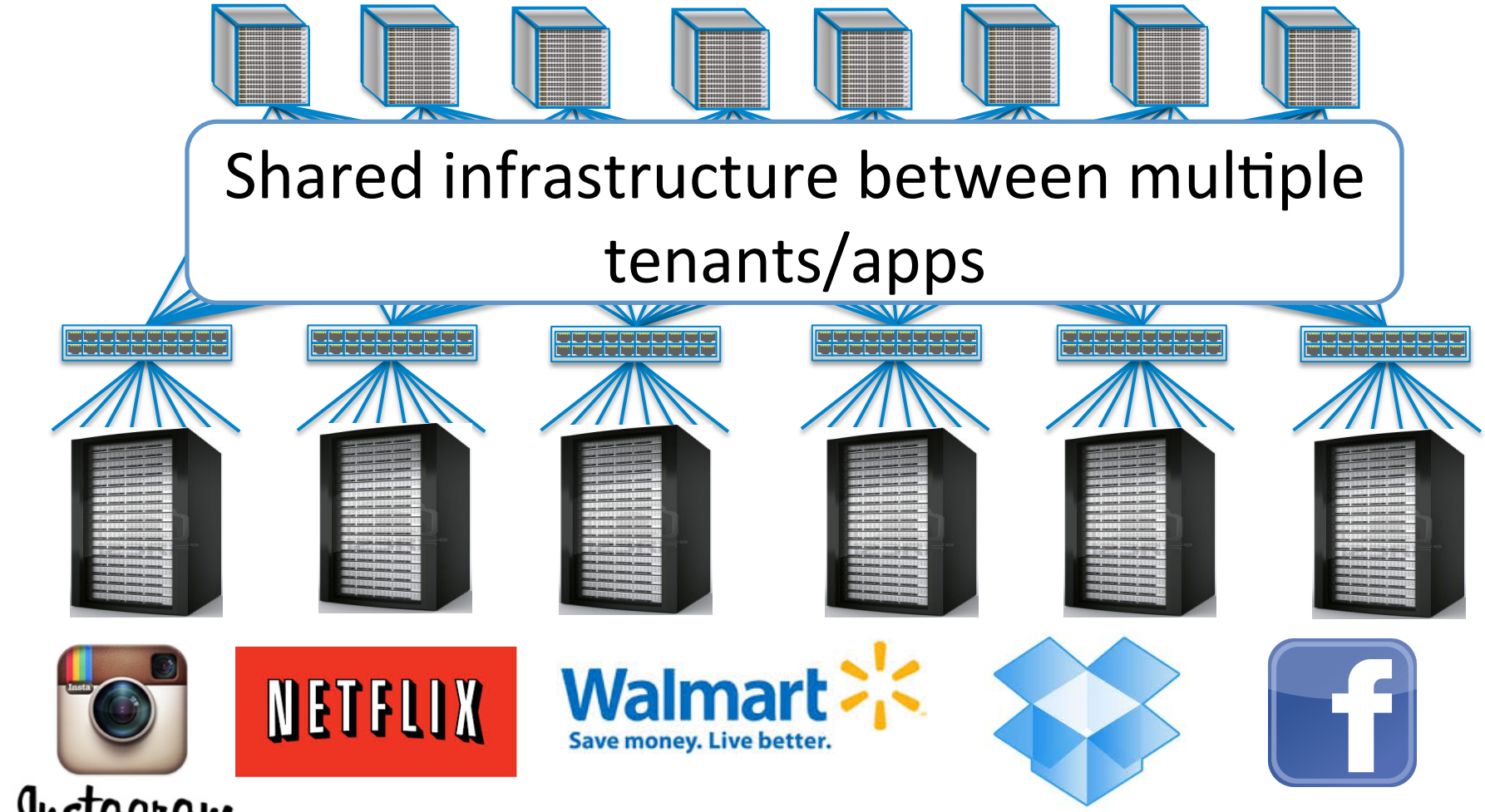
6.888
Lecture 6:
Network Performance Isolation

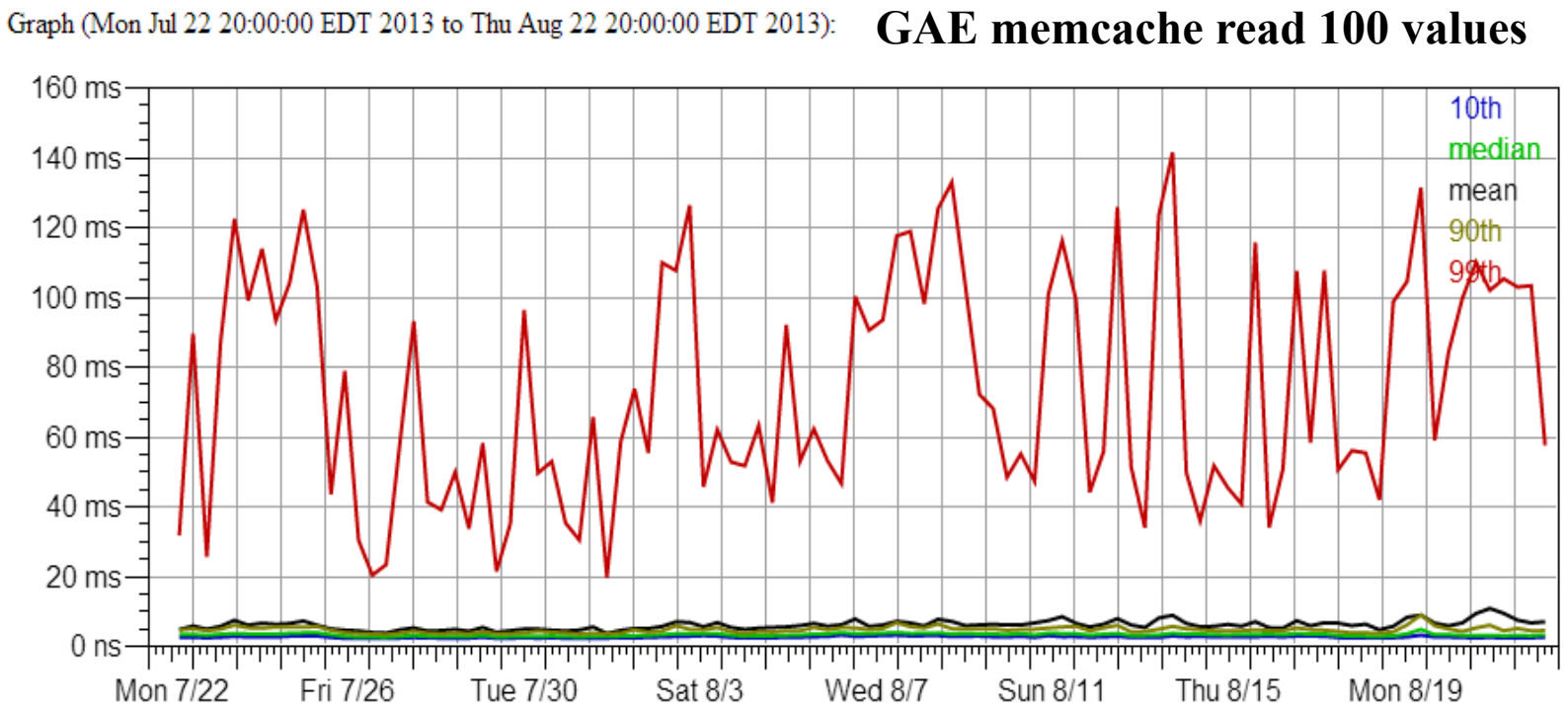
Mohammad Alizadeh

Spring 2016

Multi-tenant Cloud Data Centers



Lack of Performance Predictability



Unpredictable performance, esp. at the tail

Congestion Kills Predictability

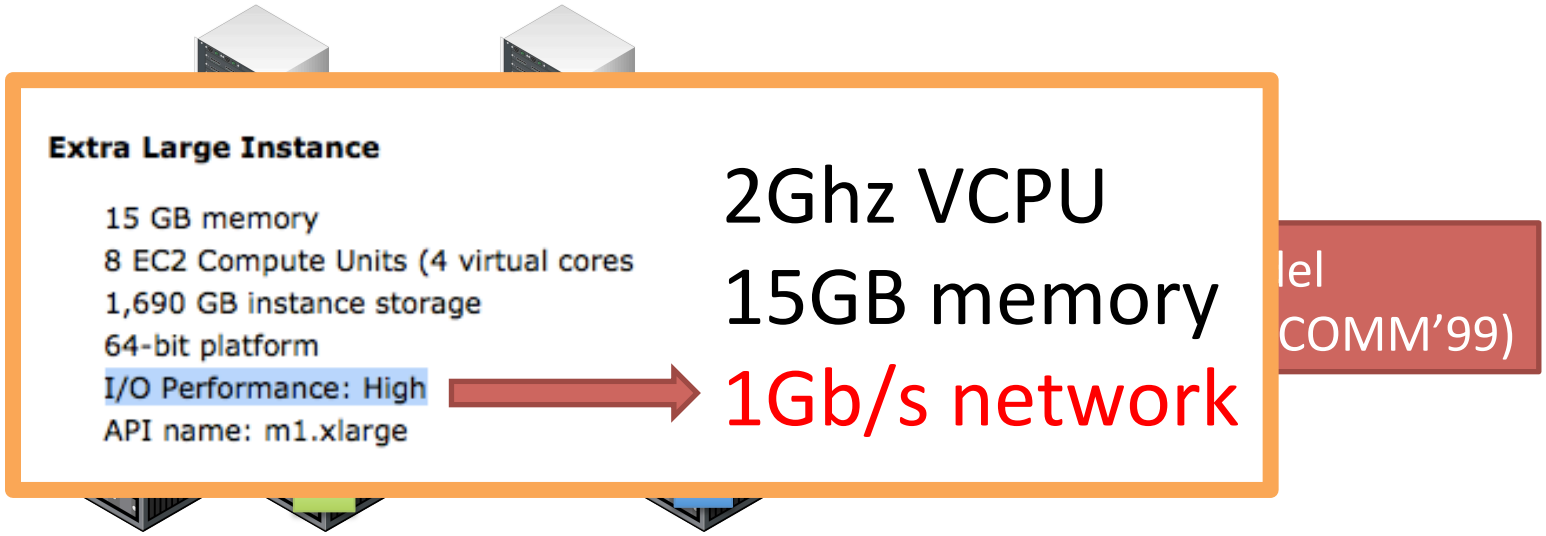
Amazon EC2
Product Details
Instances
Pricing
Purchasing Options
Developer Resources
FAQs
Getting Started
Amazon EC2 Run Command

RELATED LINKS
Amazon EC2 Dedicated Hosts
Amazon EC2 Spot Instances
Amazon EC2 Reserved Instances
Amazon EC2 Dedicated Instances
Windows Instances
VM Import/Export
AWS Management Portal for vCenter
Management Console
Documentation
Release Notes
Discussion Forum
Amazon EC2 SLA
Run Command

Instance Types Matrix

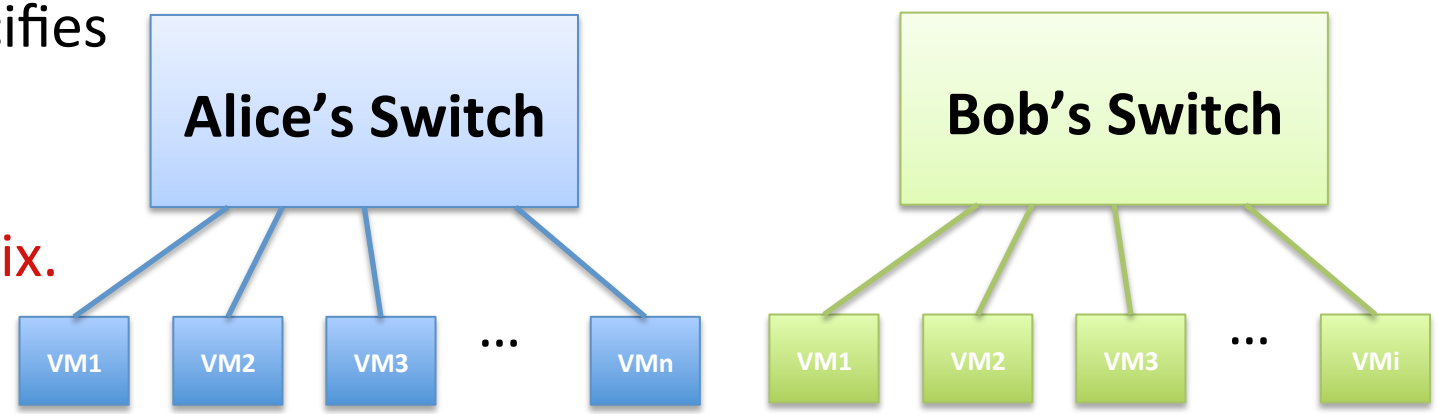
Instance Type	vCPU	Memory (GiB)	Storage (GB)	Networking Performance	Physical Processor	Clock Speed (GHz)	Intel AVX†	Intel AVX2†	Intel Turbo	EBS OPT	Enhanced Networking†
t2.nano	1	0.5	EBS Only	Low	Intel Xeon family	up to 3.3	Yes	-	Yes	-	-
t2.micro	1	1	EBS Only	Low to Moderate	Intel Xeon family	Up to 3.3	Yes	-	Yes	-	-
t2.small	1	2	EBS Only	Low to Moderate	Intel Xeon family	Up to 3.3	Yes	-	Yes	-	-
t2.medium	2	4	EBS Only	Low to Moderate	Intel Xeon family	Up to 3.3	Yes	-	Yes	-	-
t2.large	2	8	EBS Only	Low to Moderate	Intel Xeon family	Up to 3.0	Yes	-	Yes	-	-
m4.large	2	8	EBS Only	Moderate	Intel Xeon E5-2676 v3	2.4	Yes	Yes	Yes	Yes	Yes
m4.xlarge	4	16	EBS Only	High	Intel Xeon E5-2676 v3	2.4	Yes	Yes	Yes	Yes	Yes
m4.2xlarge	8	32	EBS Only	High	Intel Xeon E5-2676 v3	2.4	Yes	Yes	Yes	Yes	Yes
m4.4xlarge	16	64	EBS Only	High	Intel Xeon E5-2676 v3	2.4	Yes	Yes	Yes	Yes	Yes

Sharing the Network

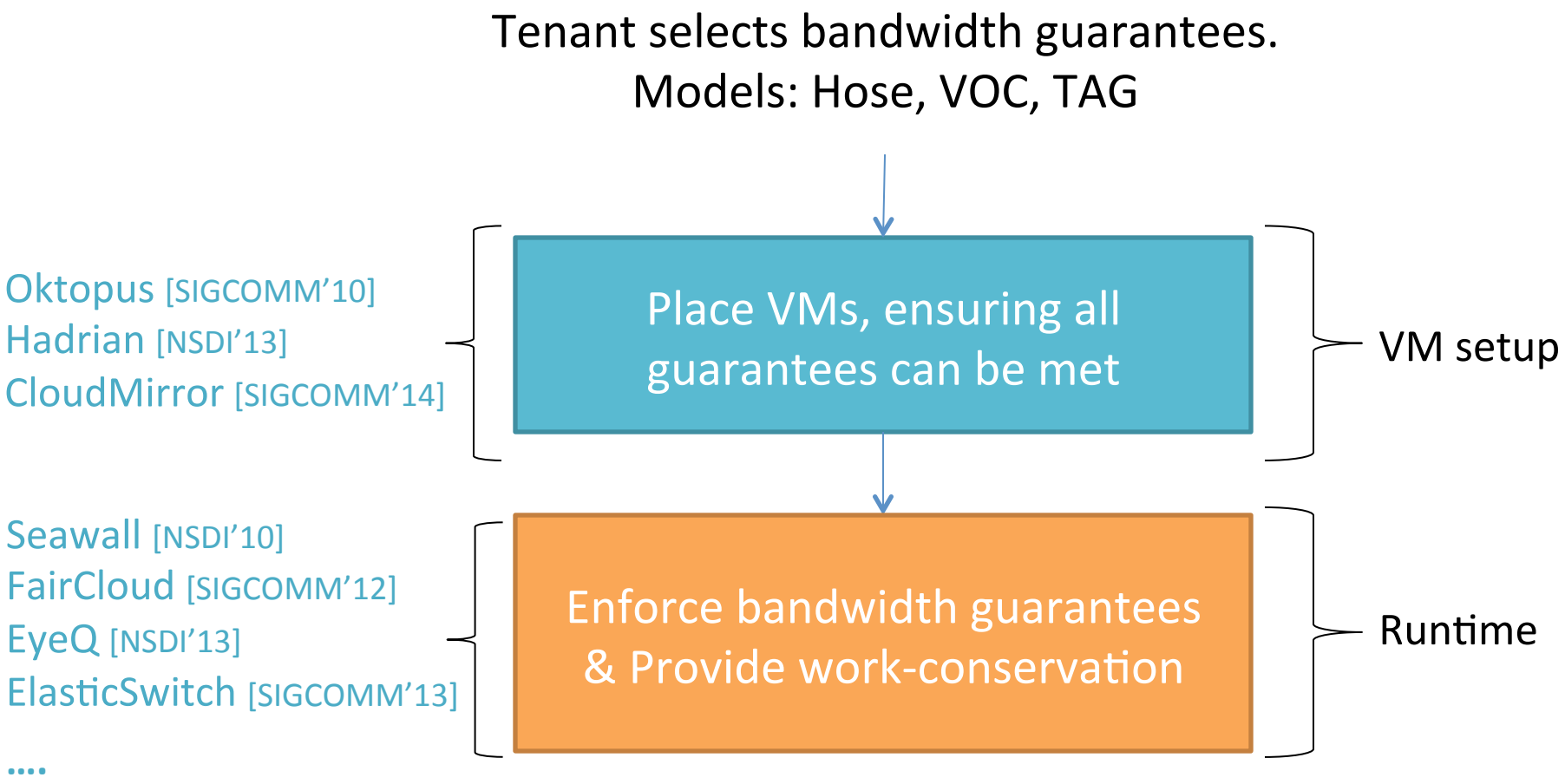


Customer specifies capacity of the virtual NIC.

No traffic matrix.



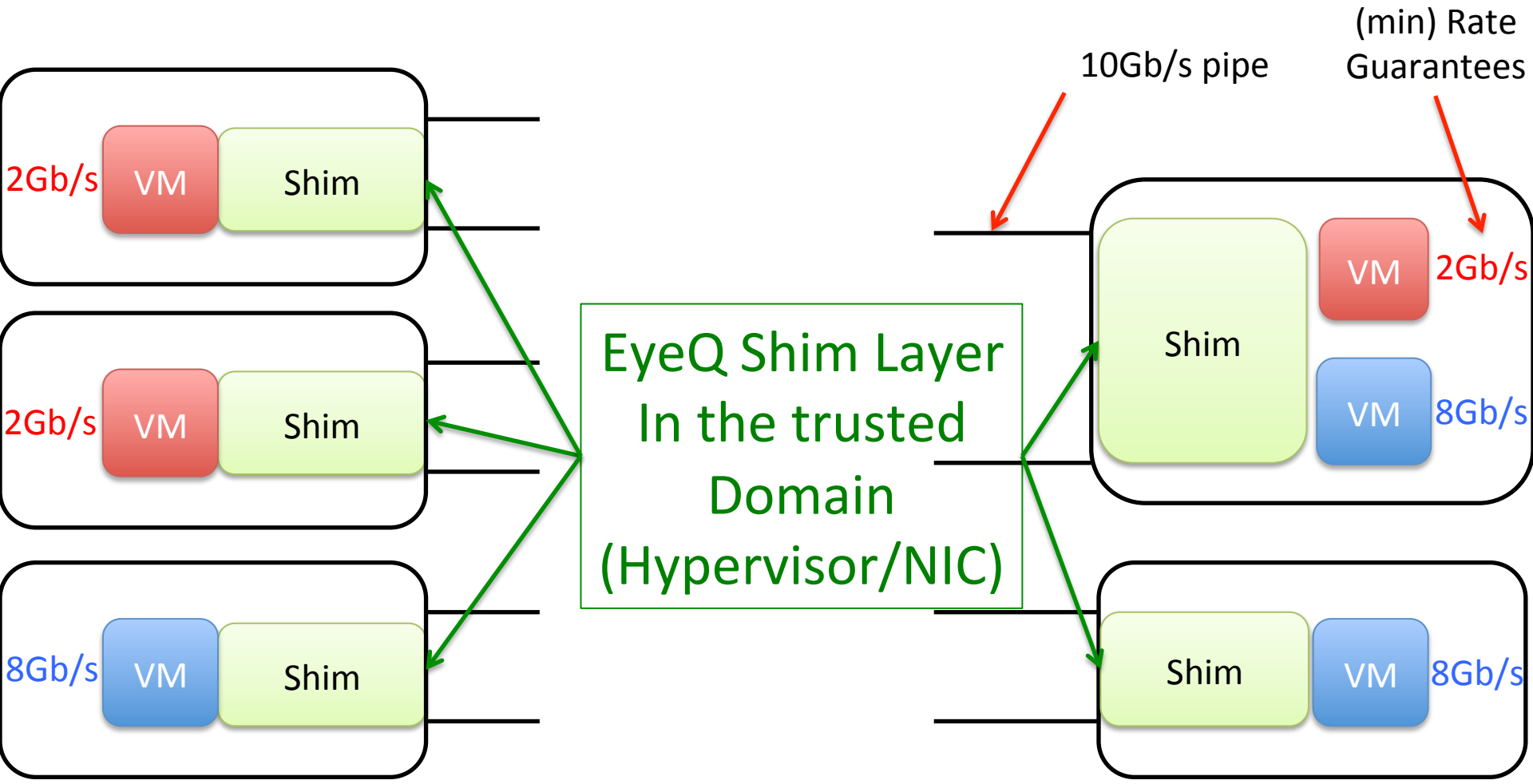
Sharing the Network



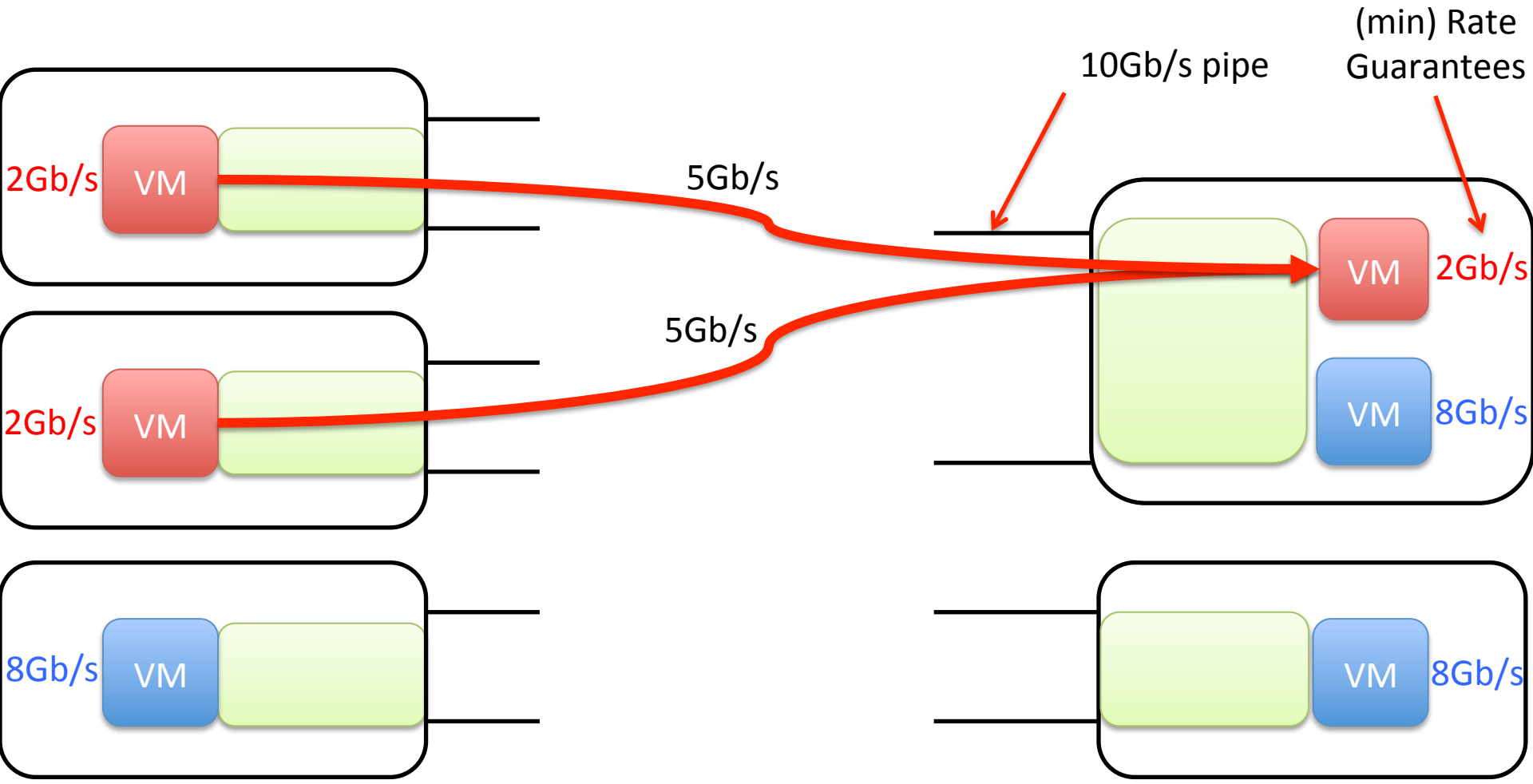
✧ Adapted from slide by Lucian Popa

Example Runtime System: EyeQ (NSDI'13)

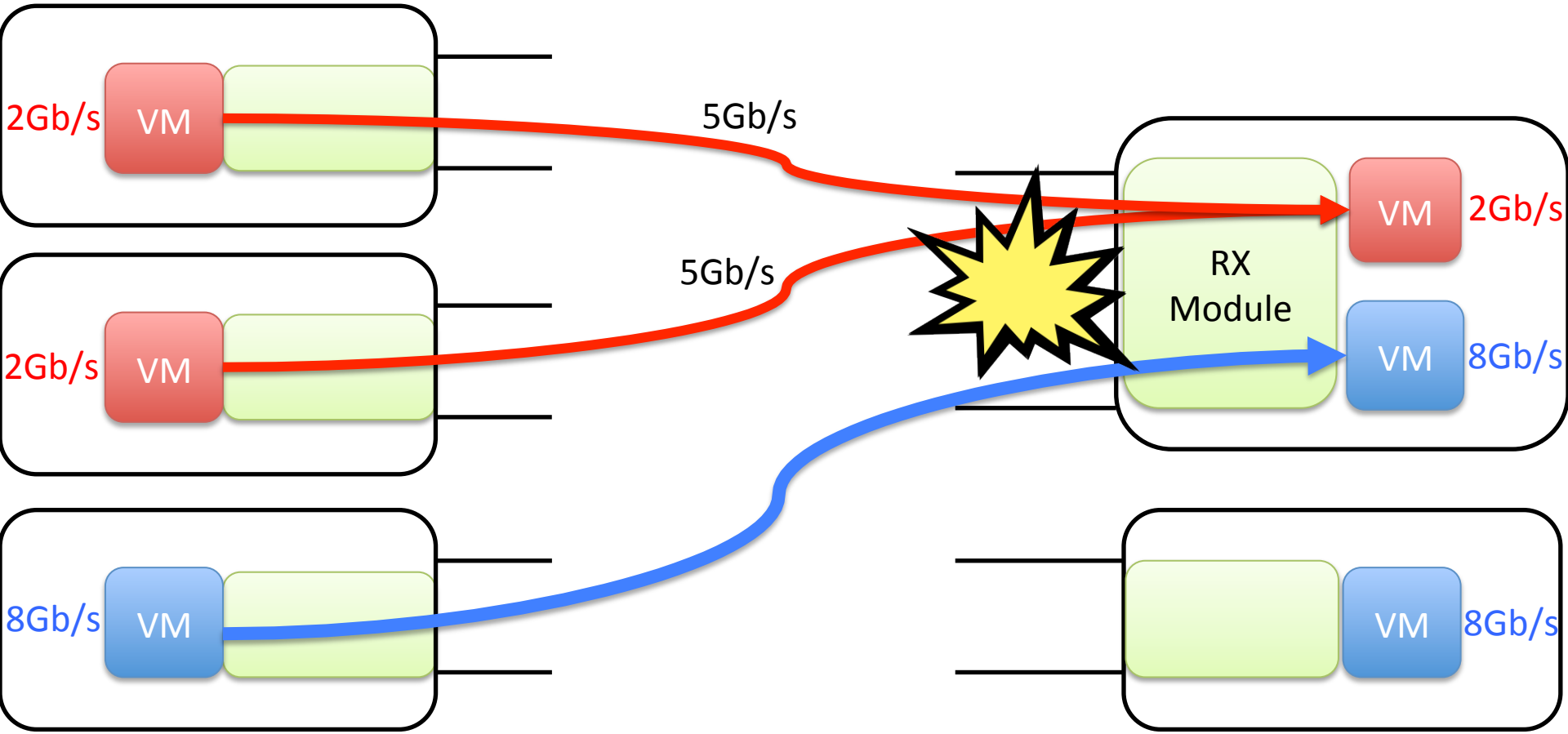
Distributed Rate Allocation



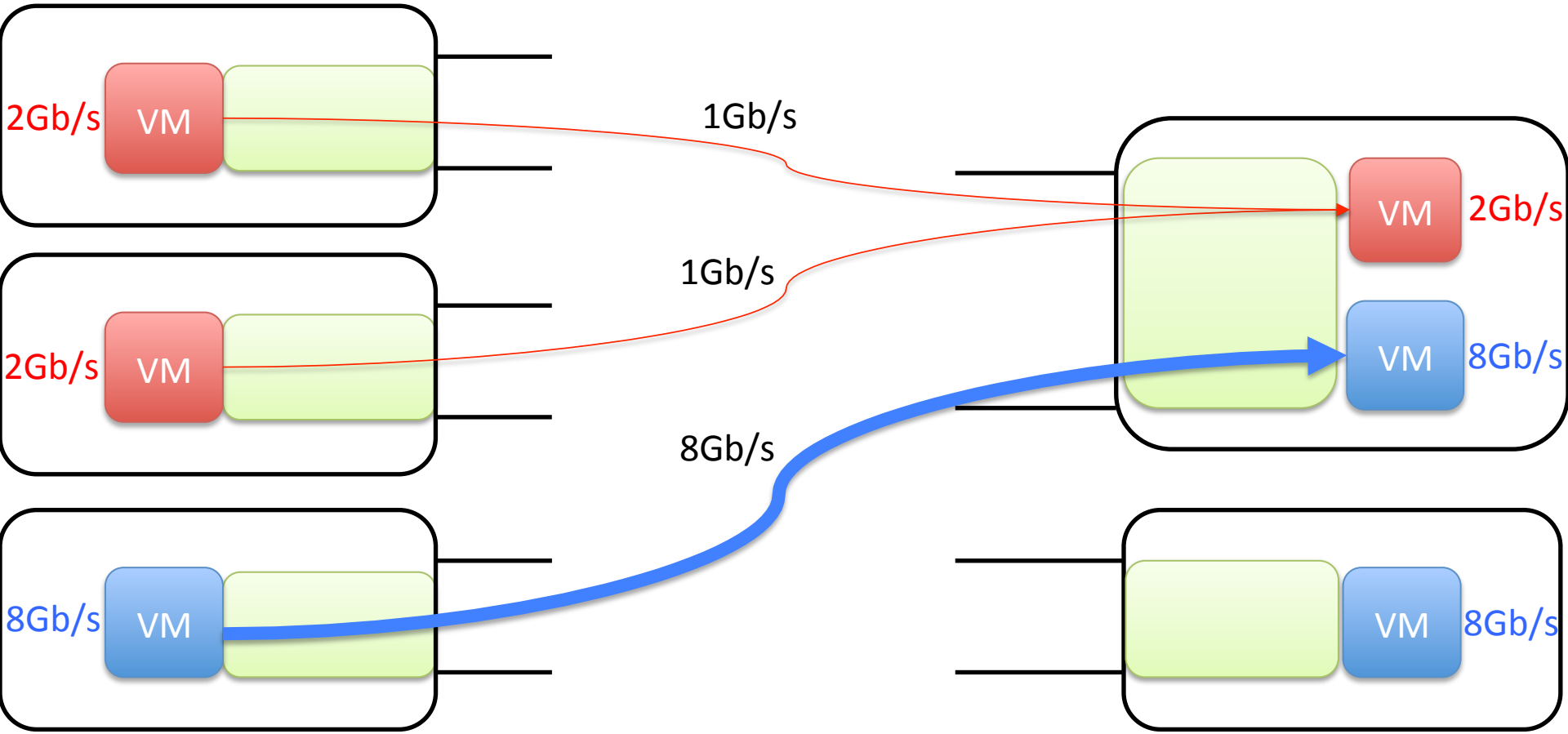
Distributed Rate Allocation



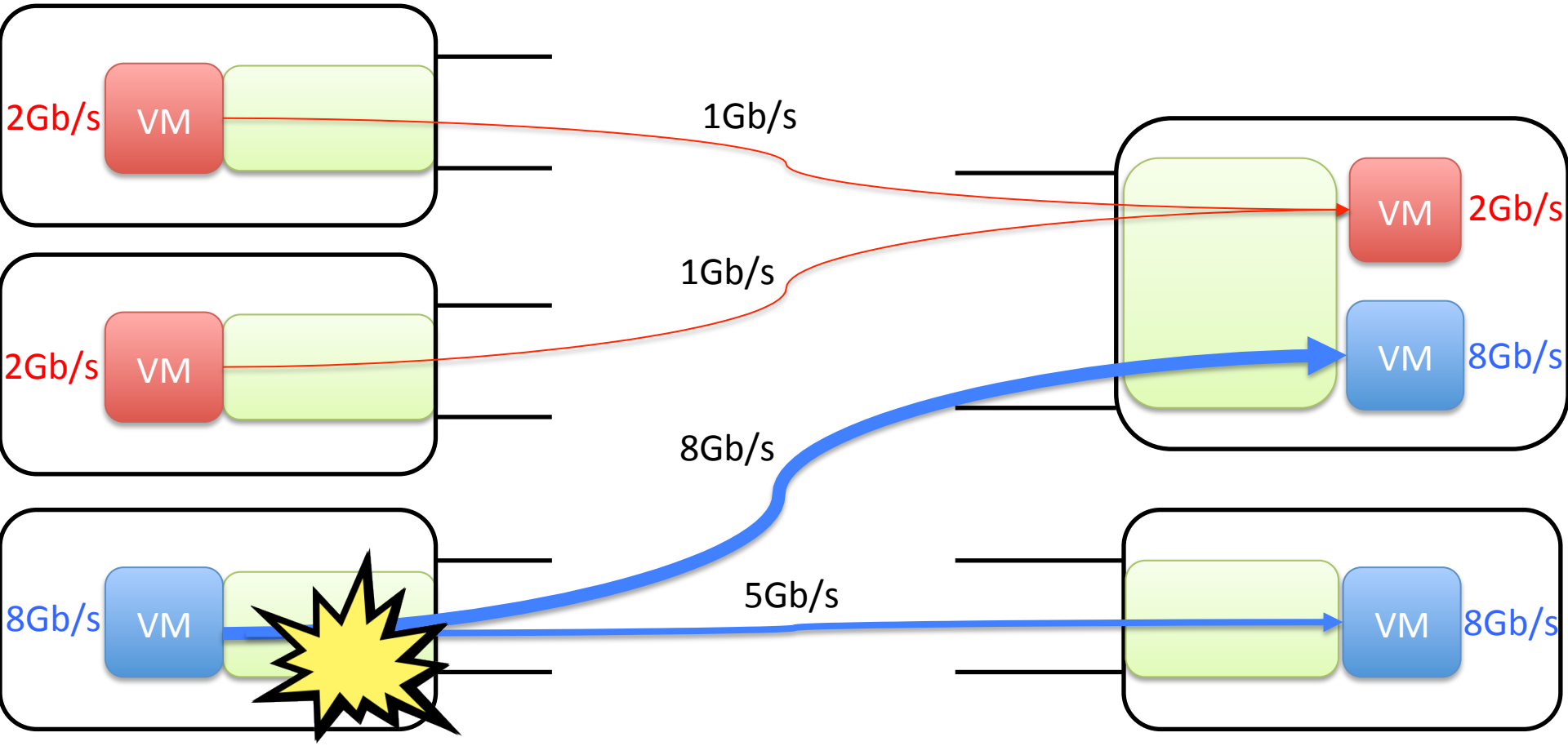
Distributed Rate Allocation



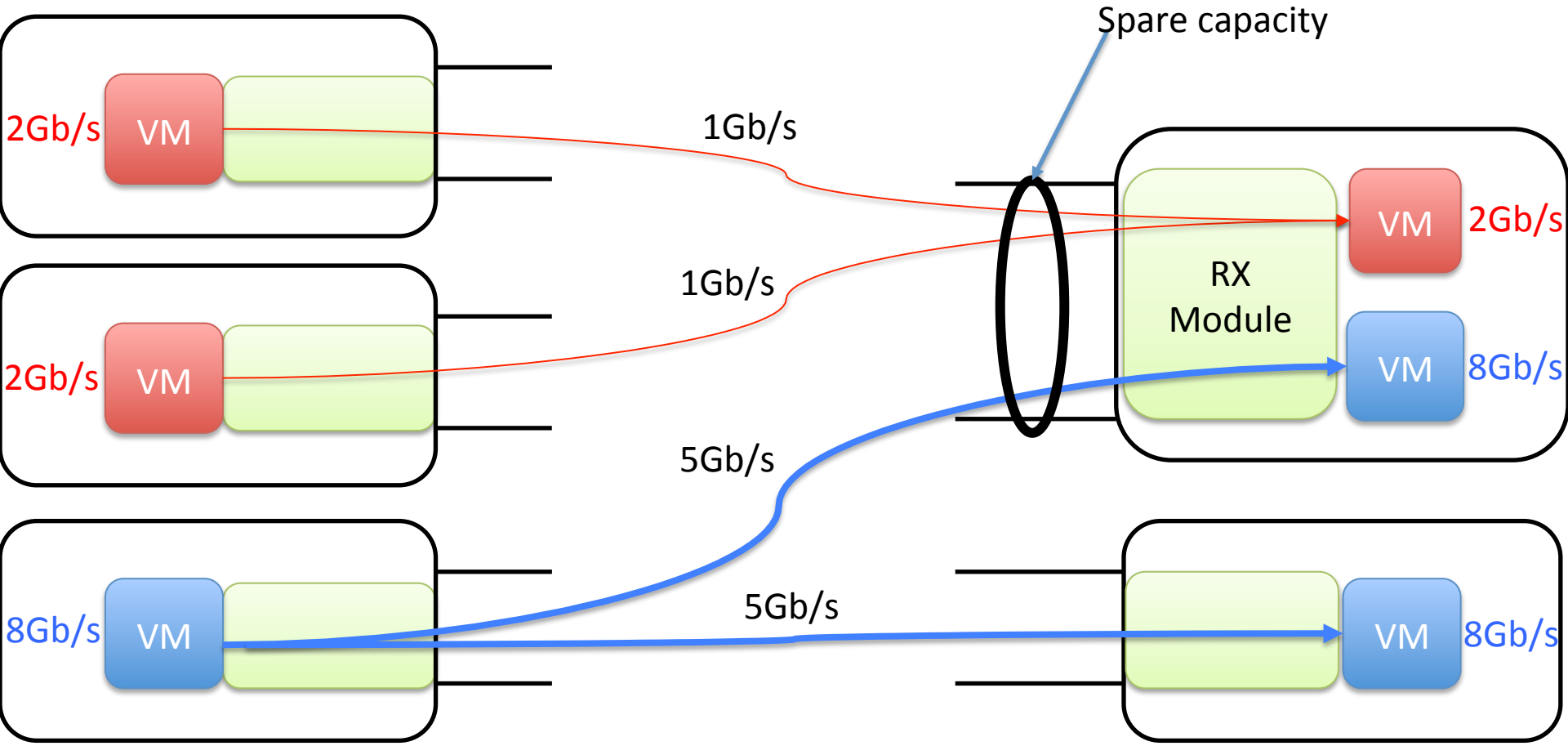
Distributed Rate Allocation



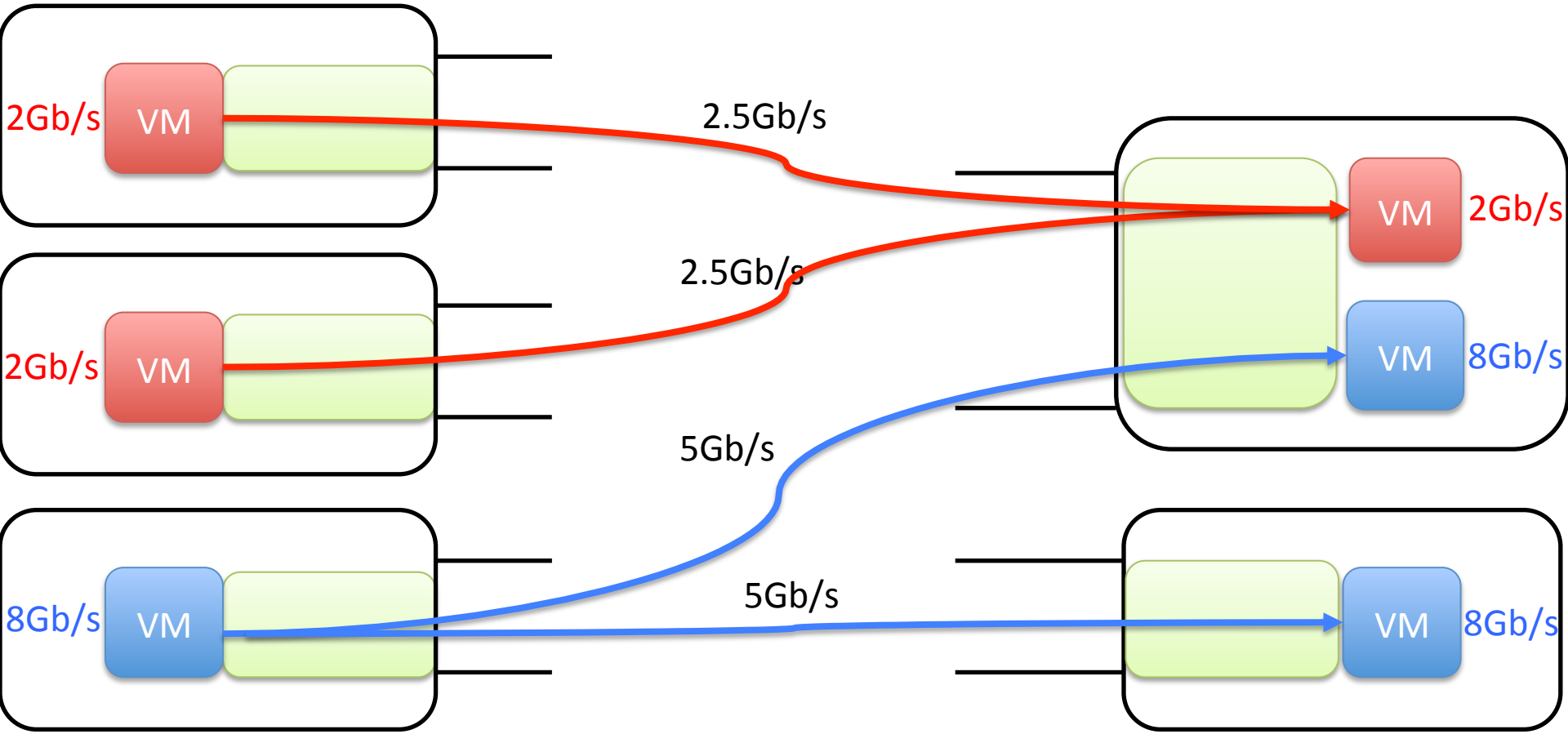
Distributed Rate Allocation



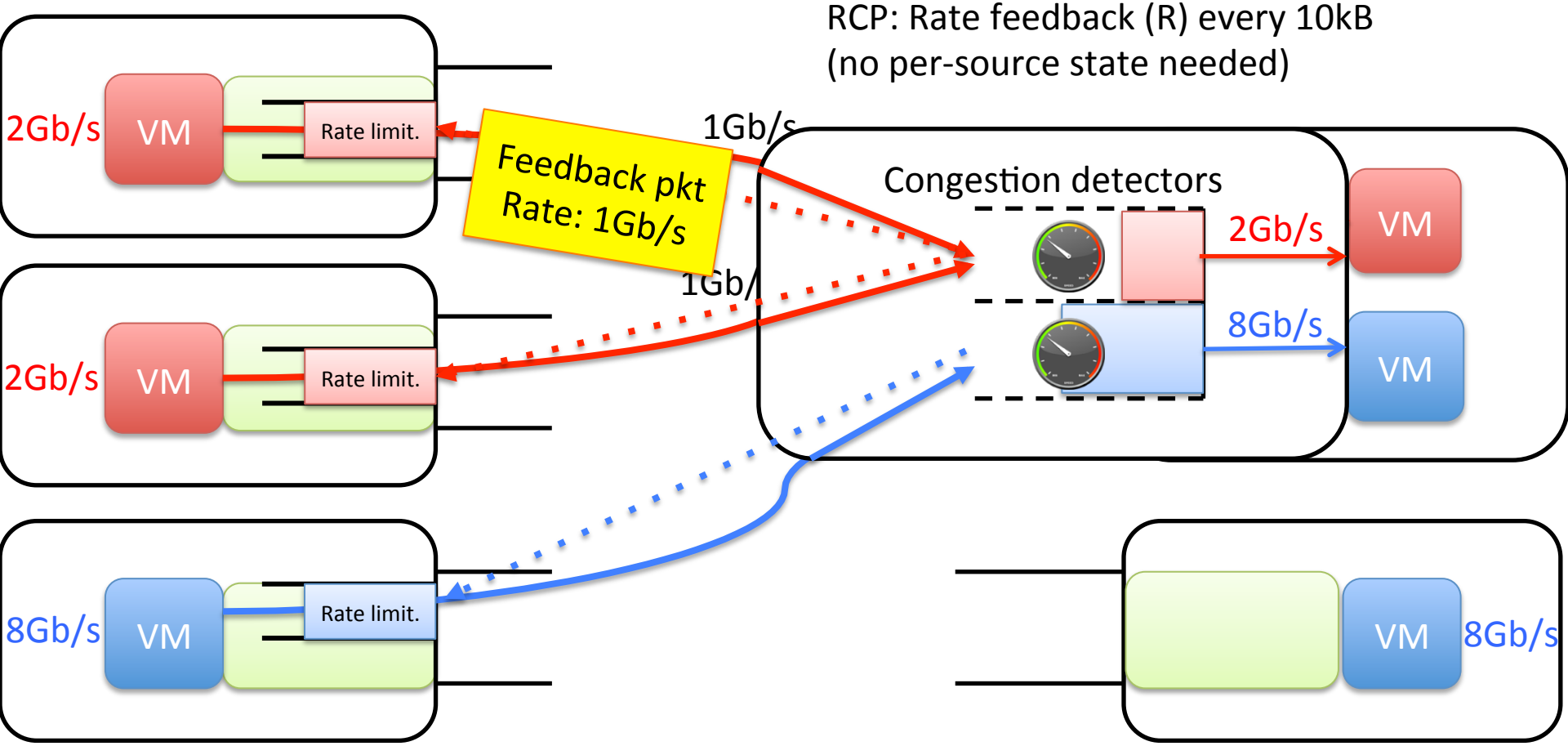
Distributed Rate Allocation



Distributed Rate Allocation

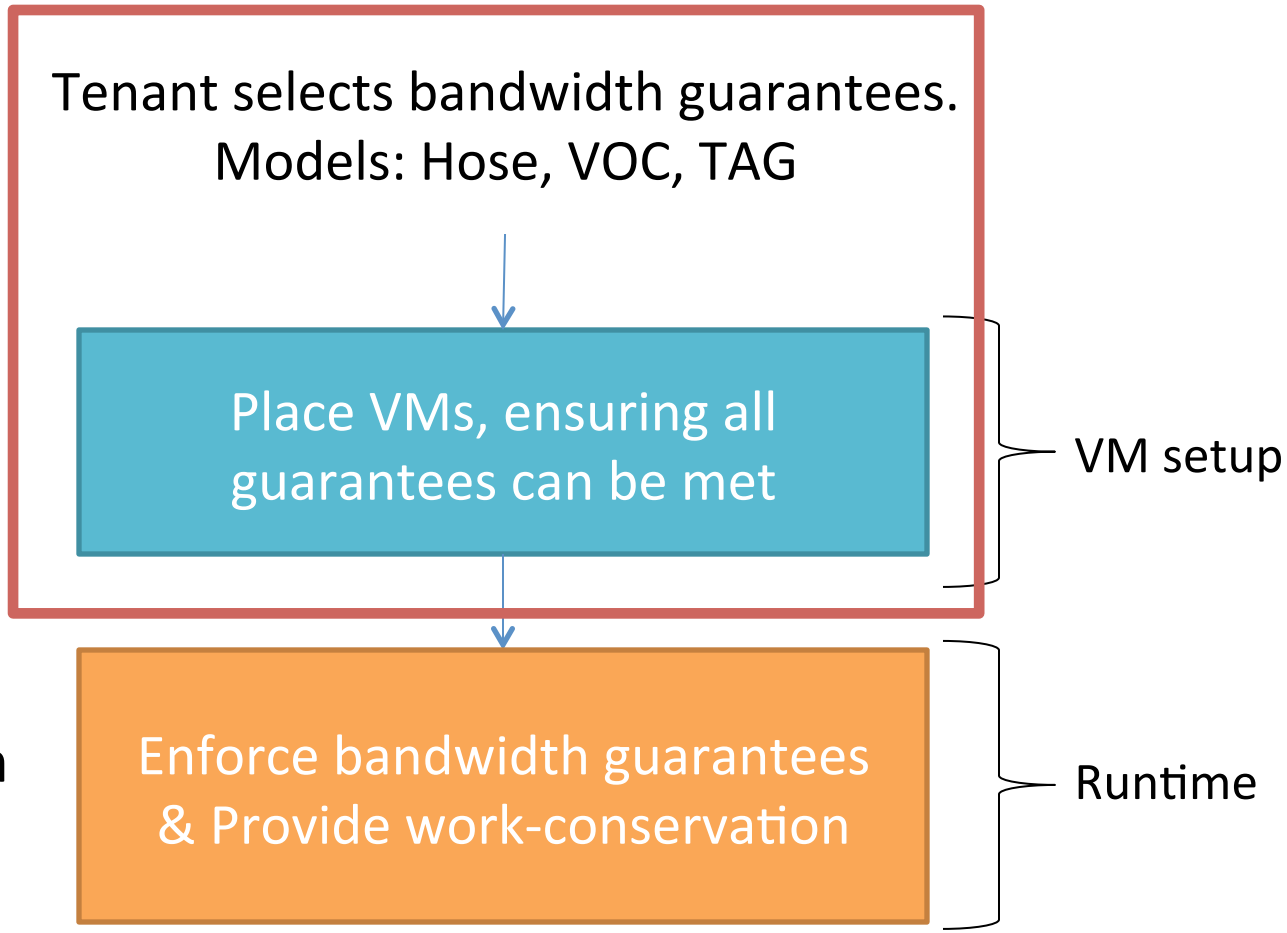


Transmit/Receive Modules



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

Sharing the Network



Cloud Mirror

Uses ElasticSwitch [SIGCOMM'13]

✧ Adapted from slide by Lucian Popa

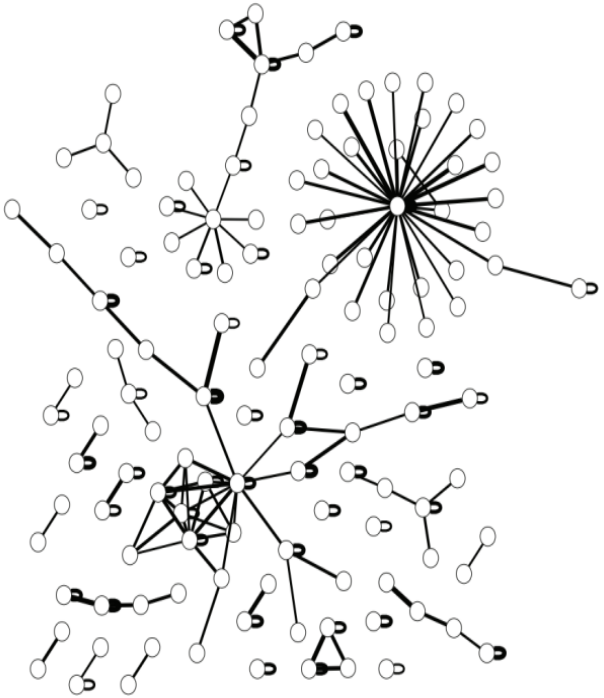
Cloud Mirror

✧ Slides based on presentation by JK Lee (HP)

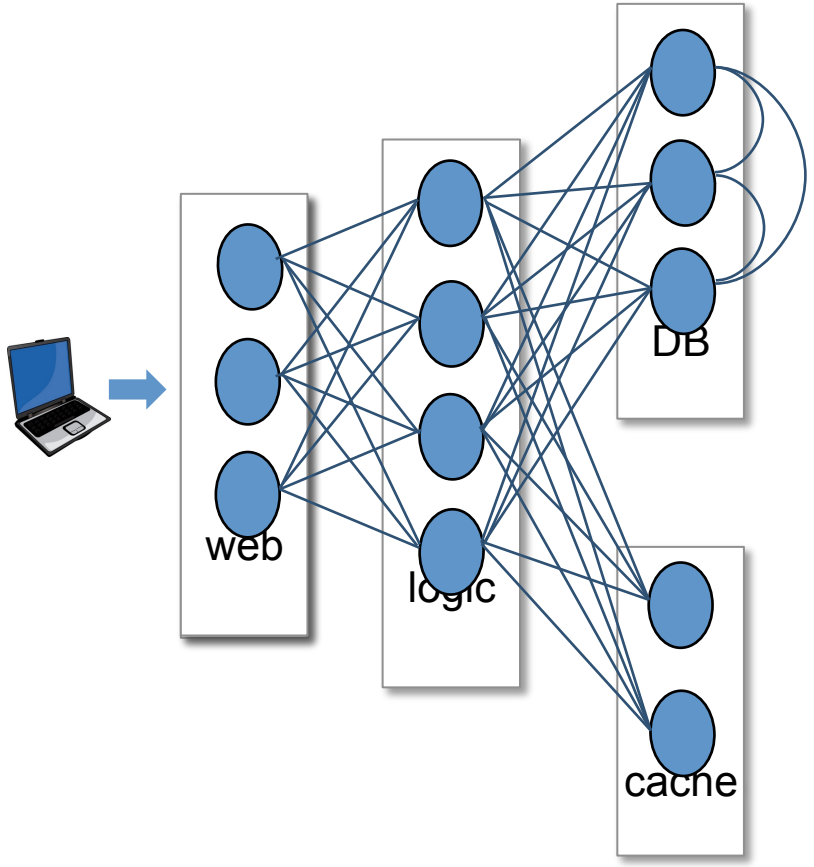
Motivation

Cloud applications are diverse & complex

Bandwidth models like pipe and hose not a good fit



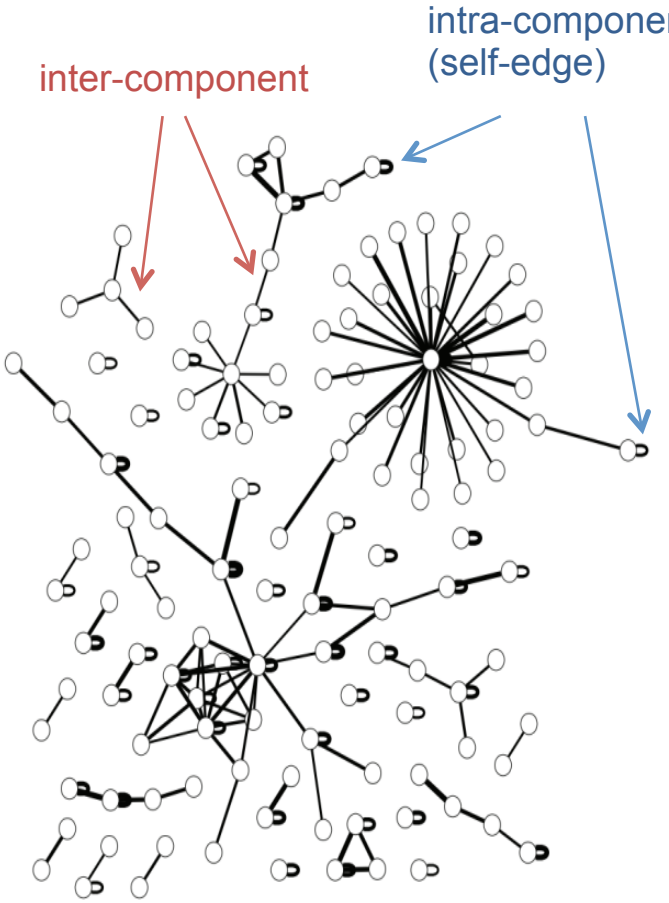
[Bing.com traffic pattern, Sigcomm'12]



Hose model is unfit

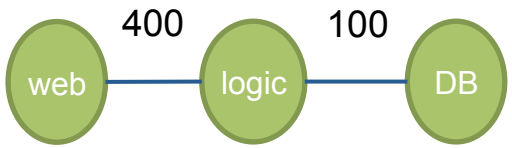
Hose aggregates BW towards different components

- Too coarse-grained
- Prevents accurate and efficient guarantees on infrastructure

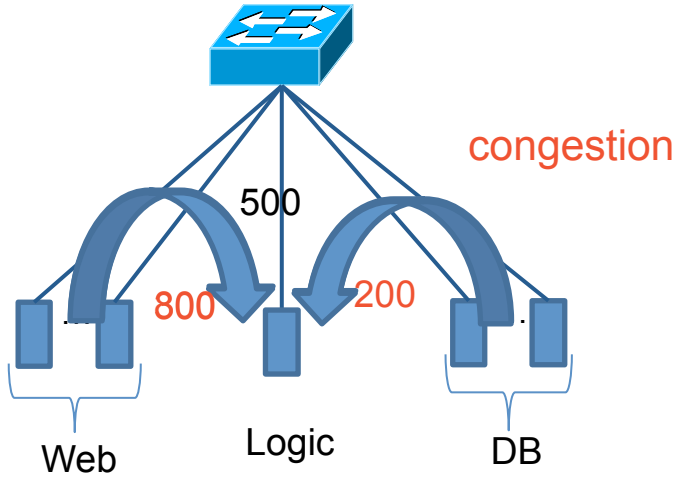


Hose is too coarse-grained

3-tier web example

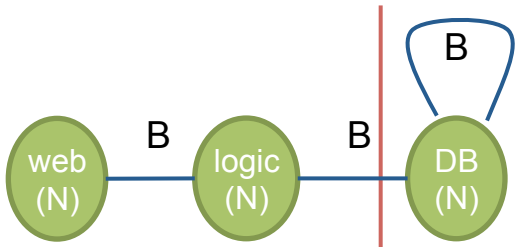


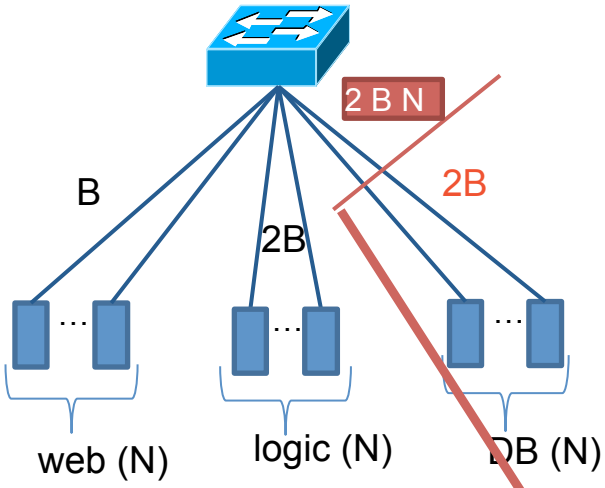
Hose model

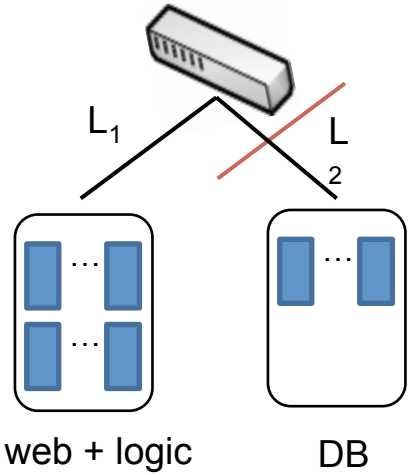


TCP-like fair allocation would yield 300:200

Hose over-provisions physical link bandwidth







N: # VMs in each tier

B: per-VM per-edge bandwidth

Physical deployment example

Hose model reservation at L_2 : $2B \cdot N$

logic - DB demand = $B \cdot N$

2X overprovision by Hose Model

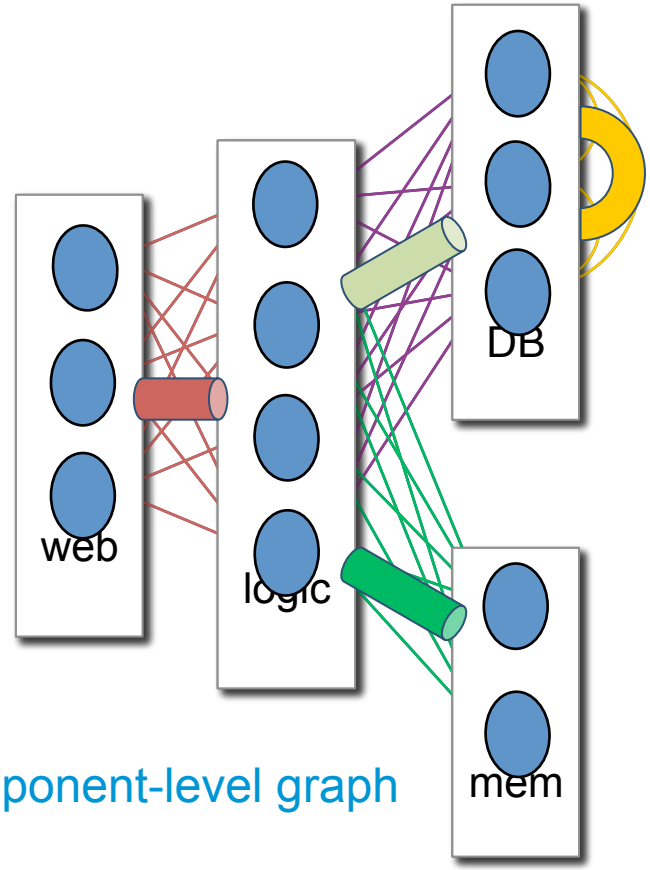
Contributions

1. Tenant Application Graph (TAG)
   - Accurate for complex apps
   - Flexible to elastic scaling
   - Intuitive

2. VM Placement Algorithm
   - Guarantee bandwidth and high availability
   - Efficient for network and compute resources

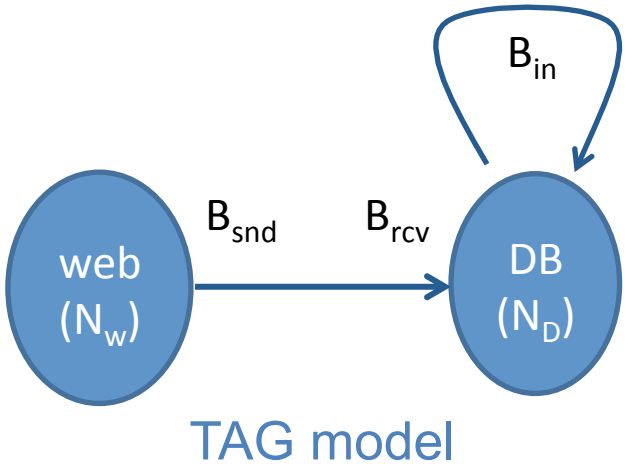
Tenant Application Graph (TAG)

1. Aggregate pipes (like Hose)
   - Model simplicity
   - Multiplexing gain
2. Preserve inter-component structure (like Pipe)
   - Accurately capture application demands
   - Efficiently utilize network resources

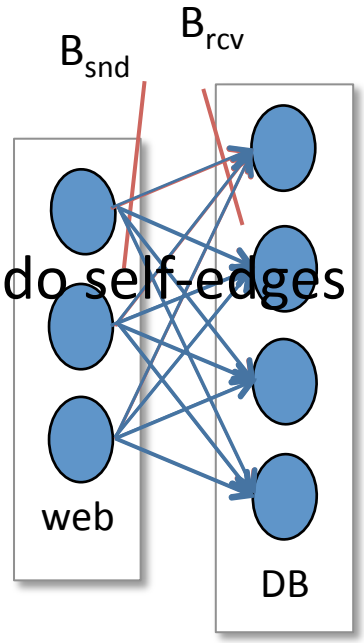


Component-level graph

Tenant Application Graph (TAG)



TAG model



What do self-edges mean?

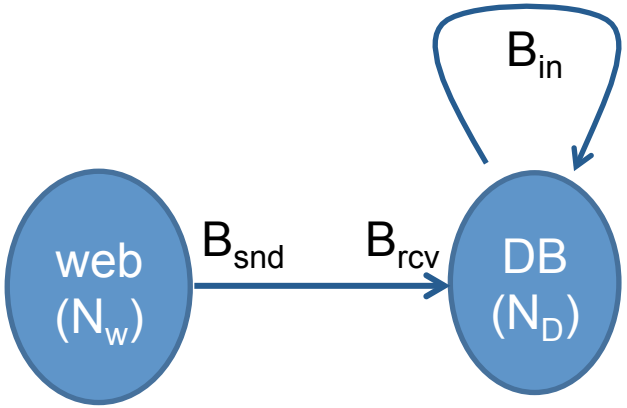
B_{snd} = per-VM sending bandwidth

(B_{rcv} = per-VM receiving bandwidth)

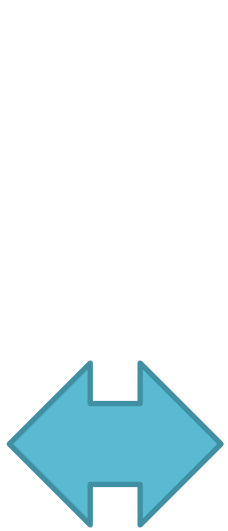
Abstract models in TAG

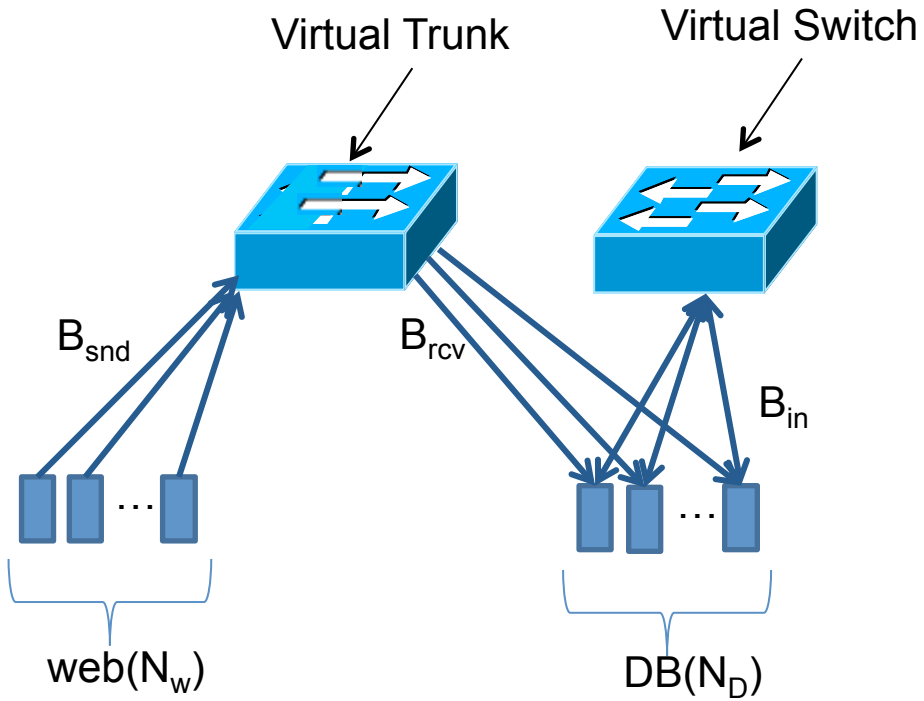
Self-edge ↔ Hose

Directional edge ↔ directional Hose, Virtual Trunk



TAG model





Total guarantee of virtual trunk

= $\min(B_{snd} \cdot N_w, B_{rcv} \cdot N_D)$

Questions

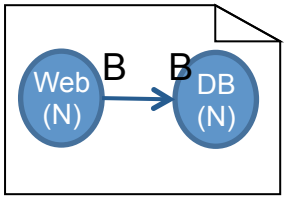
How are TAGs constructed?

How to predict bandwidth demands?

What is missing for the TAG model?

CloudMirror operation

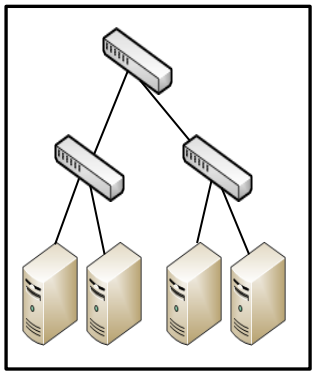
TAG spec

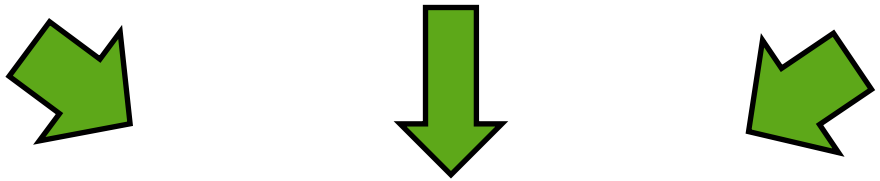


Available VM slots

host1	10
host2	50
host3	25

Network topology &
BW reservation state





VM placement
BW reservation
Admission control

Discussion

Next Time: Centralized Arbitration

Fastpass: A Centralized "Zero-Queue" Datacenter Network

Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, Hans Fugal
M.I.T. Computer Science & Artificial Intelligence Lab Facebook
http://fastpass.mit.edu/

ABSTRACT

An ideal datacenter network should provide several properties, including low median and tail latency, high utilization (throughput), fair allocation of network resources between users or applications, deadline-aware scheduling, and congestion (loss) avoidance. Current datacenter networks inherit the principles that went into the design of the Internet, where packet transmission and path selection decisions are distributed among the endpoints and routers. Instead, we propose that each sender should delegate control—to a centralized arbiter—of when each packet should be transmitted and what path it should follow.

This paper describes Fastpass, a datacenter network architecture built using this principle. Fastpass incorporates two fast algorithms: the first determines the time at which each packet should be transmitted, while the second determines the path to use for that packet. In addition, Fastpass uses an efficient protocol between the endpoints and the arbiter and an arbiter replication strategy for fault-tolerant failover. We deployed and evaluated Fastpass in a portion of Face-

Current network architectures distribute packet transmission decisions among the endpoints ("congestion control") and path selection among the network's switches ("routing"). The result is strong fault-tolerance and scalability, but at the cost of a loss of control over packet delays and paths taken. Achieving high throughput requires the network to accommodate bursts of packets, which entails the use of queues to absorb these bursts, leading to delays that rise and fall. Mean delays may be low when the load is modest, but tail (e.g., 99th percentile) delays are rarely low.

Instead, we advocate what may seem like a rather extreme approach to exercise (very) tight control over when endpoints can send packets and what paths packets take. We propose that *each* packet's timing be controlled by a logically centralized *arbiter*, which also determines the packet's path (Fig. 1). If this idea works, then flow rates can match available network capacity over the time-scale of individual packet transmission times, unlike over multiple round-trip times (RTTs) with distributed congestion control. Not only will persistent congestion be eliminated, but packet latencies will not rise

Flowtune: Flowlet Control for Datacenter Networks

Jonathan Perry, Hari Balakrishnan and Devavrat Shah
M.I.T. Computer Science and Artificial Intelligence Laboratory
{yonch,hari,devavrat}@mit.edu

January 2016, submitted for review

ABSTRACT

Rapid convergence to a desired allocation of network resources to endpoint traffic has been a long-standing challenge for packet-switched networks. The reason for this is that congestion control decisions are distributed across the endpoints, which vary their offered load in response to changes in application demand and network feedback on a packet-by-packet basis. We propose a different approach for datacenter networks, *flowlet control*, in which congestion control decisions are made at the granularity of a flowlet, not a packet. With flowlet control, allocations have to change only when flowlets arrive or leave.

We have implemented this idea in a system called Flowtune using a centralized allocator that receives flowlet start and end notifications from endpoints. The allocator computes optimal rates using a new, fast method for network utility maximization, and updates endpoint congestion-control parameters. Experiments show that Flowtune outperforms DCTCP, pFabric, sfqCoDel, and XCP on tail packet delays in various settings, converging to optimal rates within a few packets rather than over several RTTs. Our implementation of Flowtune handles 10.4× more throughput per core and scales

For this reason, in this paper, we adopt the position that a *flowlet*, and not a packet, is a better granularity for congestion control. By "flowlet", we mean a batch of packets that are backlogged at a sender; a flowlet ends when there is a threshold amount of time where a sender's queue is empty. Our idea is to compute optimal rates for a set of active flowlets and to update those rates dynamically as flowlets enter and leave the network.

We have developed these ideas in a system called *Flowtune*. It is targeted at datacenter environments, although it may also be used in enterprise and carrier networks, but is not intended for use in the wide-area Internet.

In datacenters, fast convergence of allocations is critical, as flowlets tend to be short (one study shows that the majority of flows are under 10 packets [11]) and link capacities are large (40 Gbits/s and increasing); if it takes more than, say, 40 μs to converge to the right rate, then most flowlets will have already finished. Most current approaches use distributed, end-to-end congestion control, and generally take multiple RTTs to converge. By contrast, Flowtune uses a centralized rate allocator, aiming for fast convergence to optimal rates for all flows in the network.