Microsoft Research

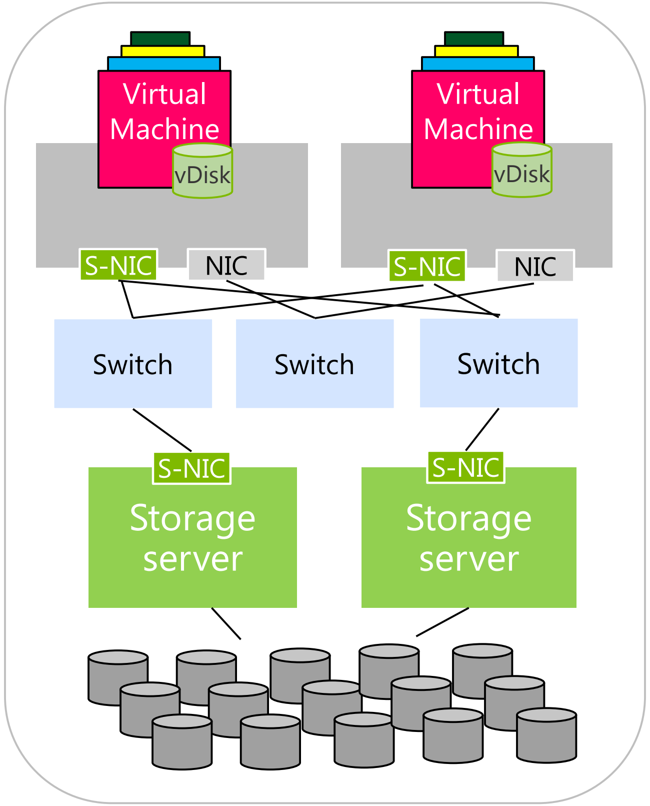
Microsoft

IOFlow: a Software-Defined Storage Architecture

Eno Thereska, Hitesh Ballani, Greg O'Shea, Thomas Karagiannis, Antony Rowstron, Tom Talpey, Richard Black, Timothy Zhu

Microsoft Research

Background: Enterprise data centers



- General purpose applications
- Application runs on several VMs
- Separate network for VM-to-VM traffic and <u>VM-to-Storage</u> traffic
- Storage is virtualized
- Resources are <u>shared</u>

Motivation

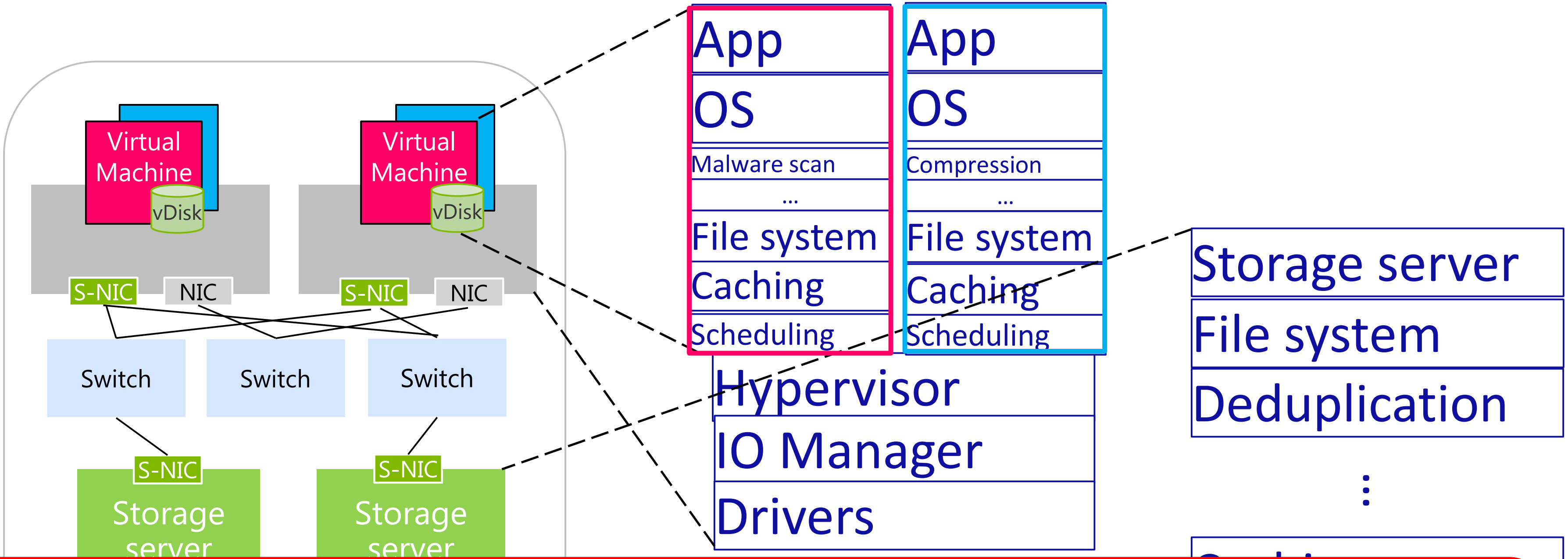
Want: predictable application behaviour and performance

Need system to provide end-to-end SLAs, e.g.,

- Guaranteed storage bandwidth B
- Guaranteed high IOPS and priority
- Per-application control over decisions along IOs' path

It is hard to provide such SLAs today

Example: guarantee aggregate bandwidth B for Red tenant



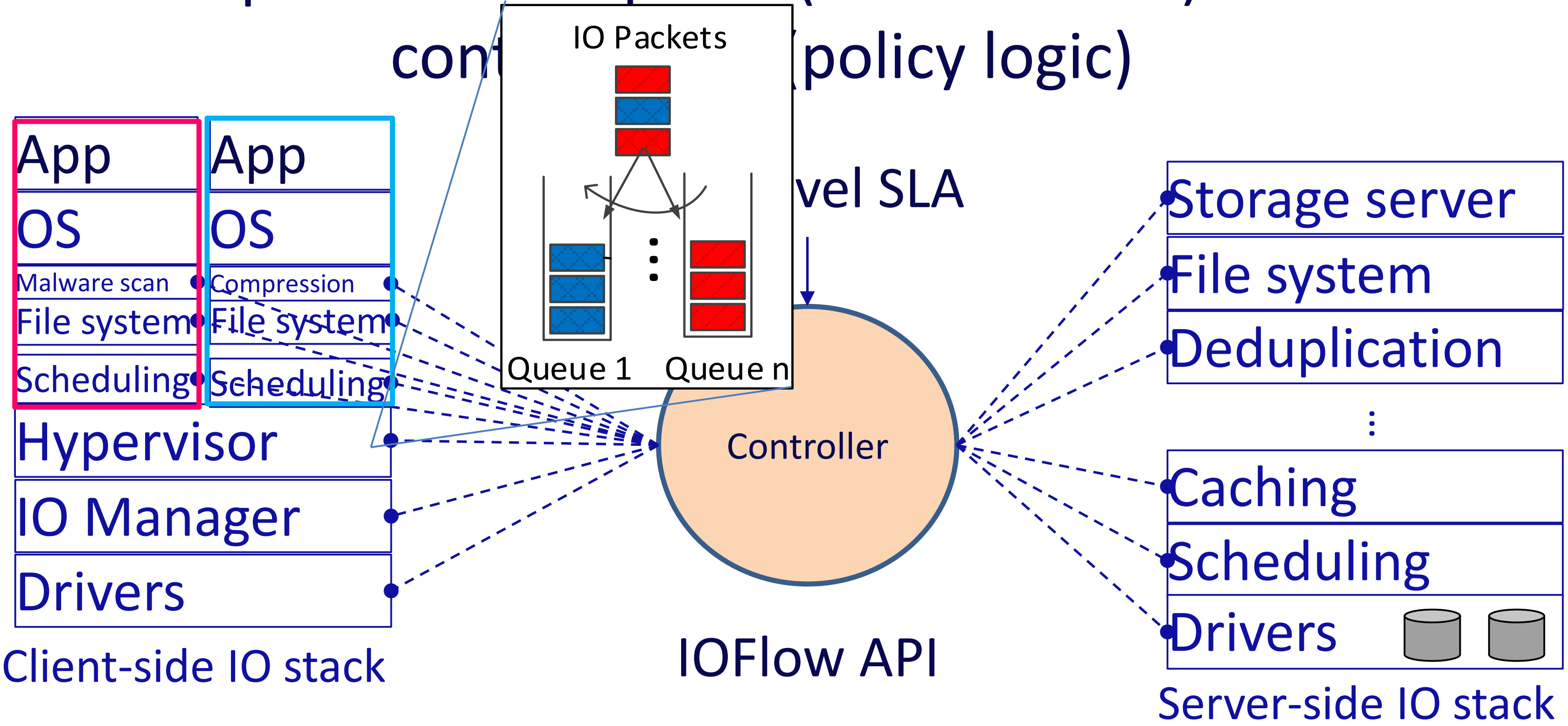
Deep IO path with 18+ different layers that are configured and operate independently and do not understand SLAs

Challenges in enforcing end-to-end SLAs

- No storage control plane
- No enforcing mechanism along storage data plane
- Aggregate performance SLAs
  - Across VMs, files and storage operations
- Want non-performance SLAs: control over IOs’ path
- Want to support unmodified applications and VMs

IOFlow architecture

Decouples the data plane (enforcement) from the cont… (policy logic)



Contributions

- Defined and built storage control plane
- Controllable queues in data plane
- Interface between control and data plane (IOFlow API)
- Built centralized control applications that demonstrate power of architecture

Storage flows

Storage “Flow” refers to all IO requests to which an SLA applies

<{VMs}, {File Operations}, {Files}, {Shares}> ---> SLA

source set

destination sets

- Aggregate, per-operation and per-file SLAs, e.g.,

  <{VM 1-100}, write, *, \\share\db-log}>---> high priority

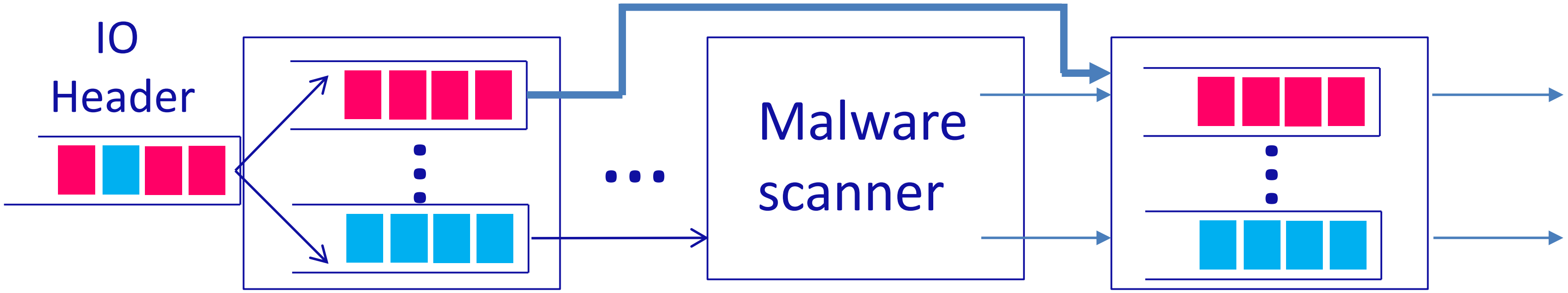
  <{VM 1-100}, *, *, \\share\db-data}> ---> min 100,000 IOPS

- Non-performance SLAs, e.g., path routing

  < VM 1, *, *, \\share\dataset>---> bypass malware scanner

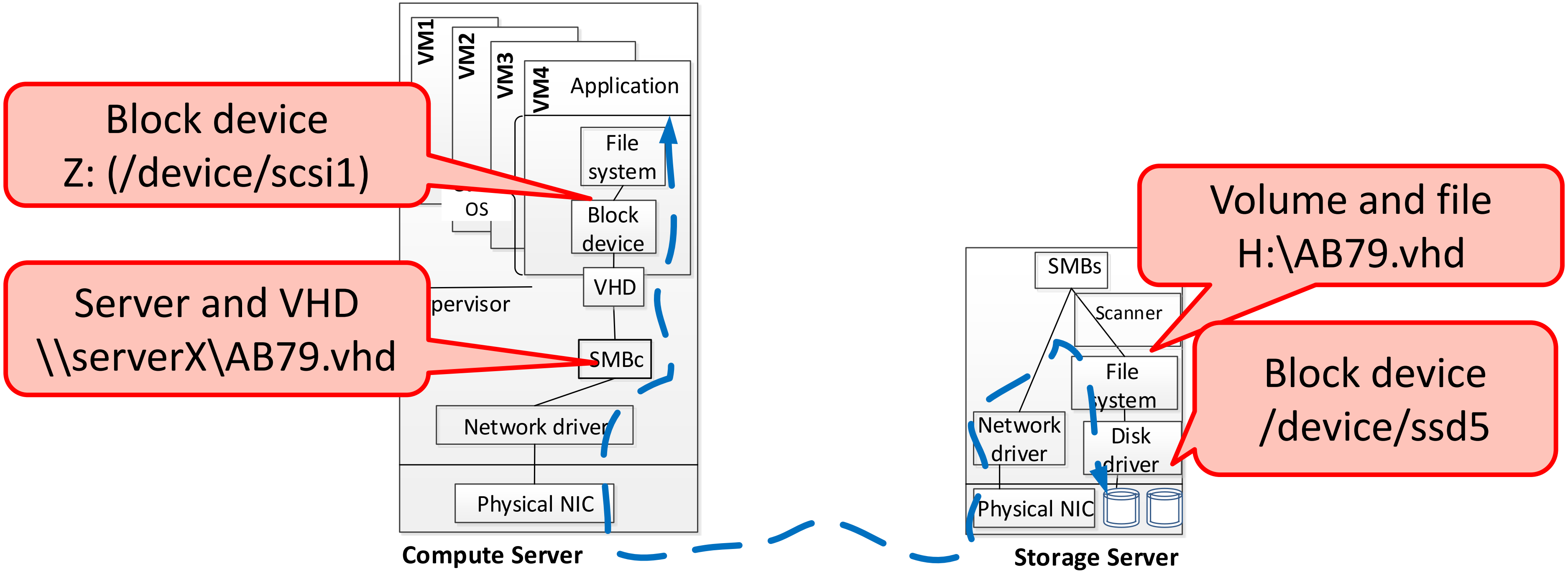
IOFlow API: programming data plane queues

1. Classification [IO Header -> *Queue*]
2. Queue servicing [Queue -> *<token rate, priority, queue size>*]
3. Routing [Queue -> *Next-hop*]



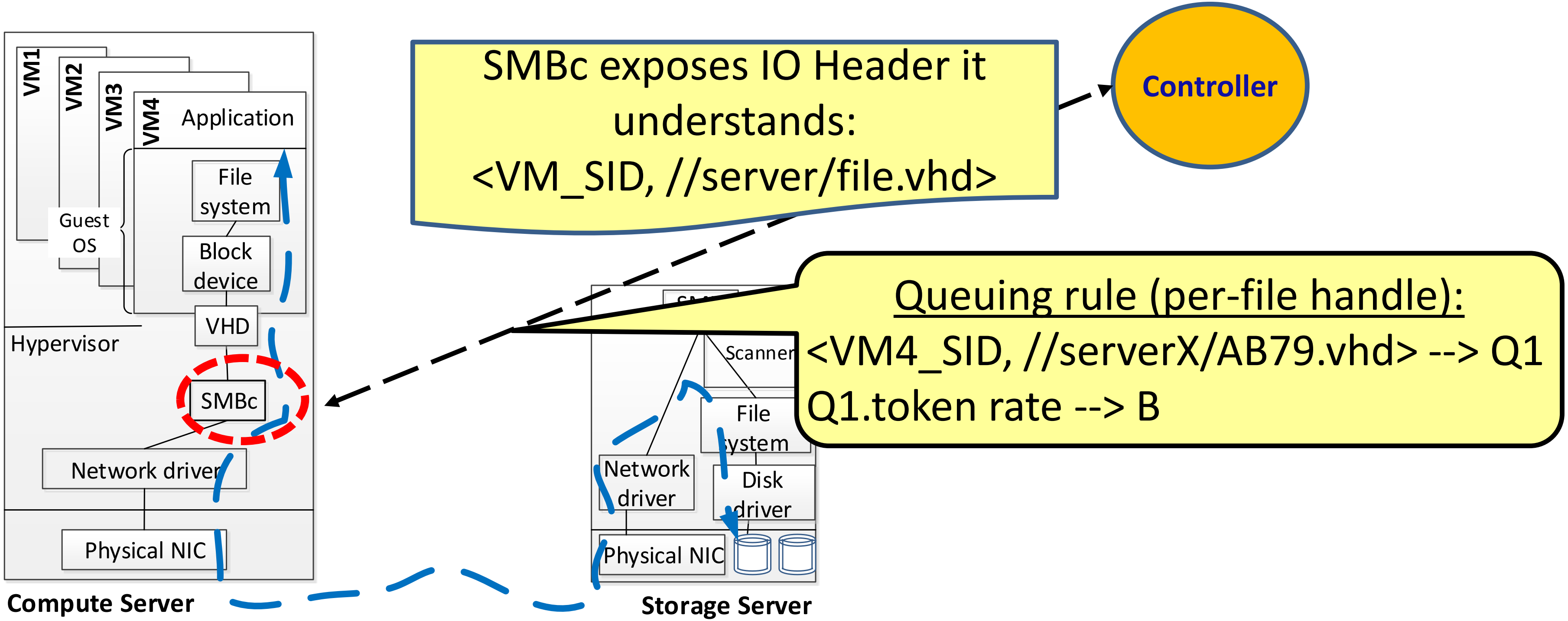
Lack of common IO Header for storage traffic

SLA: <VM 4, *, *, \\share\dataset> --> Bandwidth B



Flow name resolution through controller

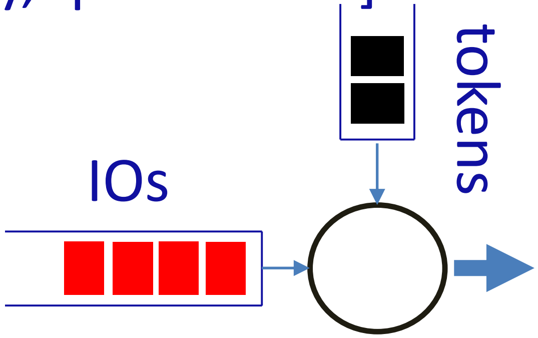
SLA: {VM 4, *, *, //share/dataset} --> Bandwidth B



Rate limiting for congestion control

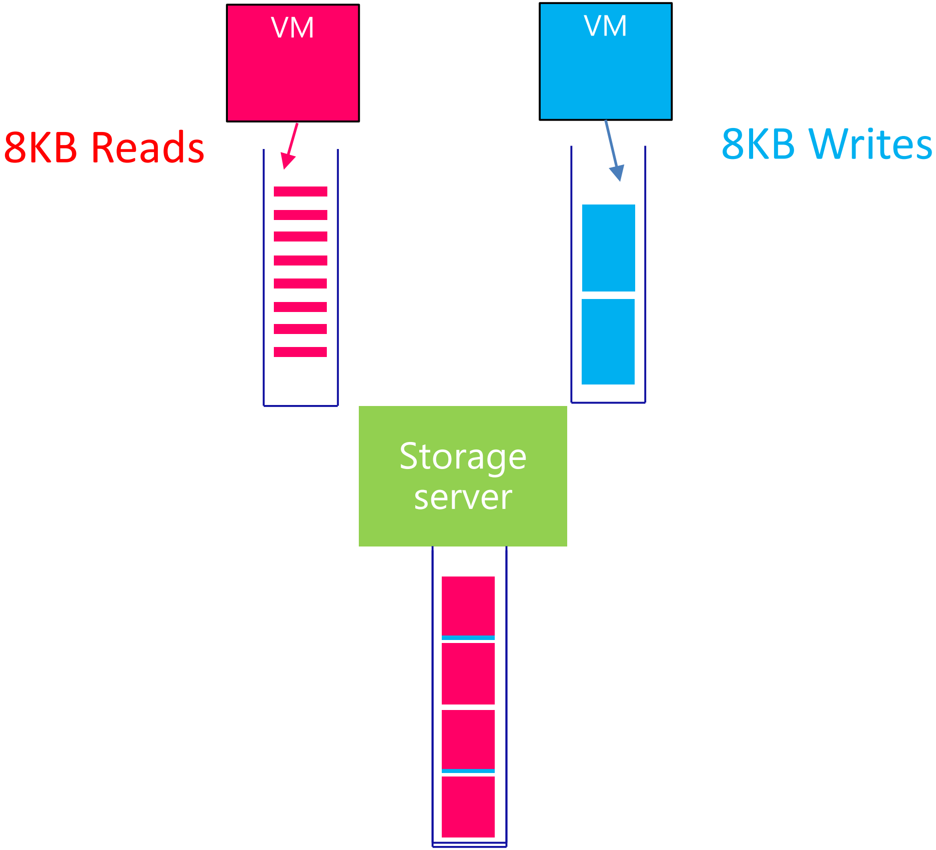
Queue servicing [Queue -> <token rate, priority, queue size>]

- Important for performance SLAs
- Today: no storage congestion control

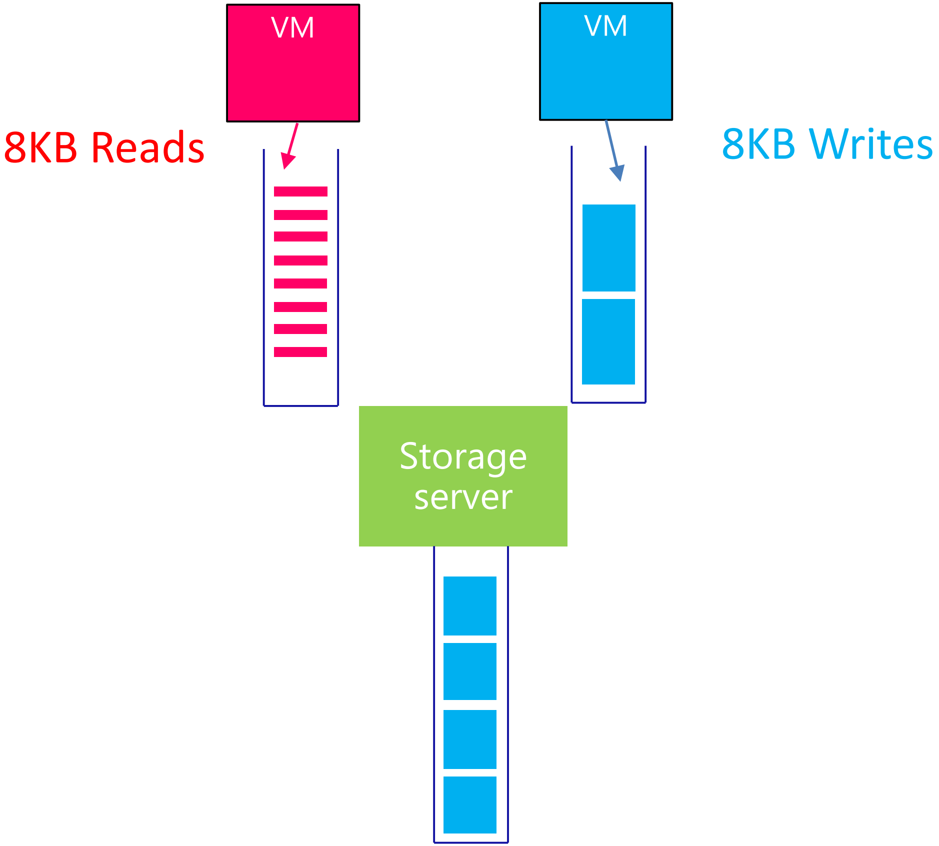


Challenging for storage: e.g., how to rate limit two VMs, one reading, one writing to get equal storage bandwidth?

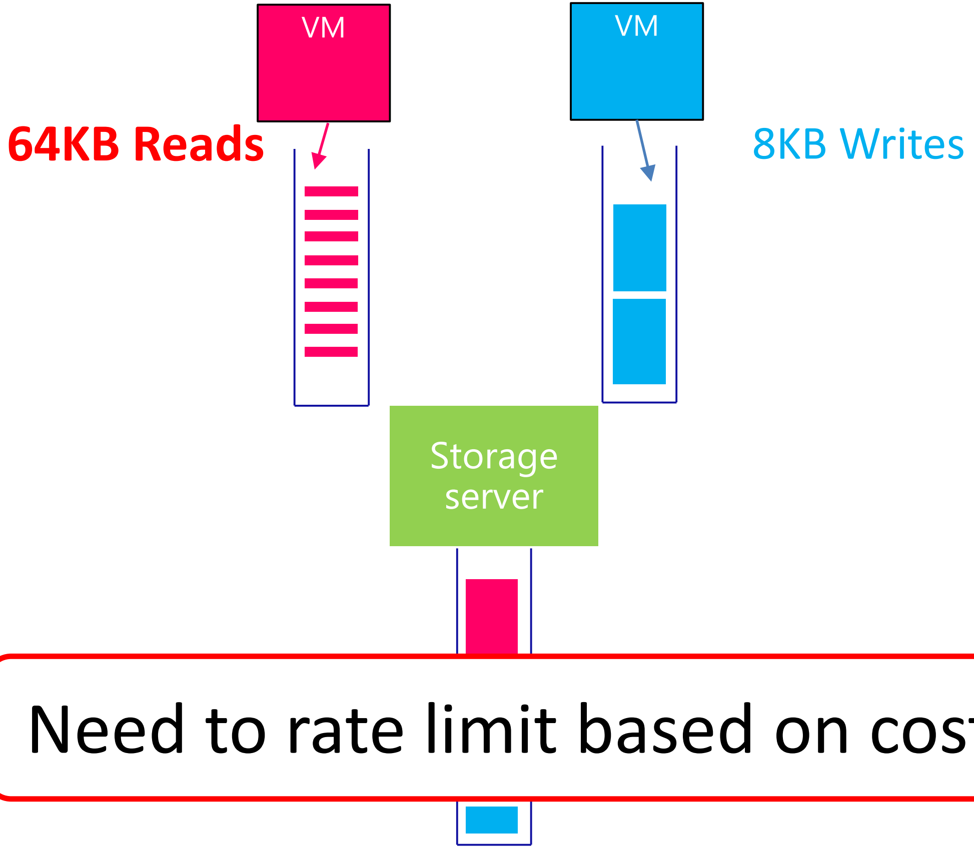
Rate limiting on payload bytes does not work



Rate limiting on bytes does not work



Rate limiting on IOPS does not work



Need to rate limit based on cost

Rate limiting based on cost

- Controller constructs empirical cost models based on device type and workload characteristics
  - RAM, SSDs, disks: read/write ratio, request size

- Cost models assigned to each queue
  - *ConfigureTokenBucket [Queue -> cost model]*

- Large request sizes split for pre-emption

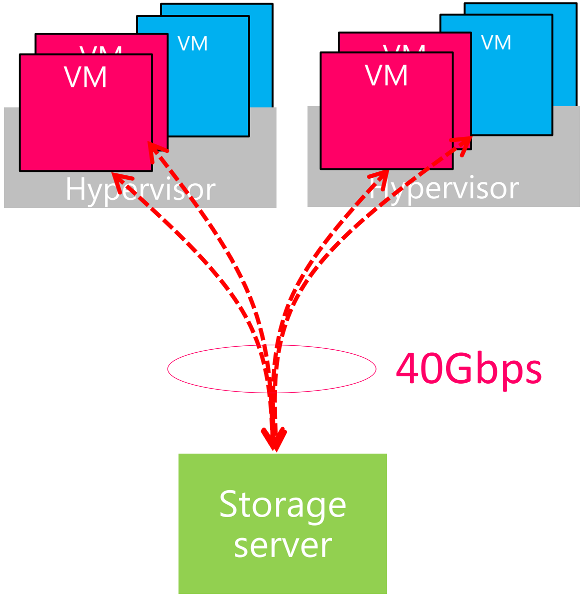
Recap: Programmable queues on data plane

- Classification [IO Header -> *Queue*]
  - Per-layer metadata exposed to controller
  - Controller out of critical path
- Queue servicing [Queue -> *<token rate, priority, queue size>*]
  - Congestion control based on operation cost
- Routing [Queue -> *Next-hop*]

How does controller enforce SLA?

Distributed, dynamic enforcement

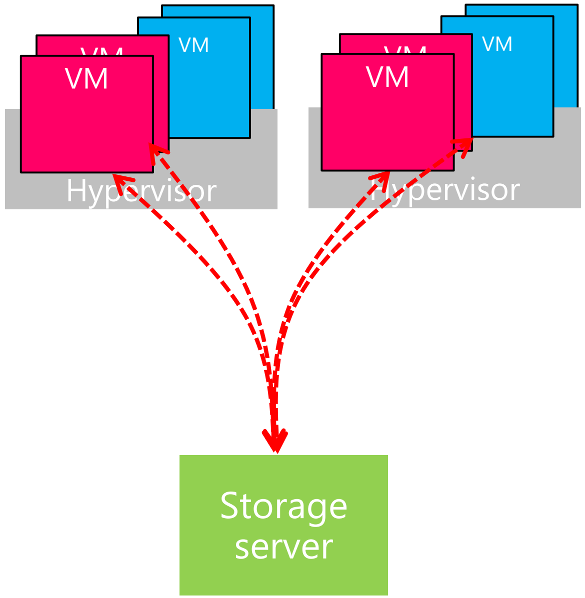
<{Red VMs 1-4}, *, * //share/dataset> --> Bandwidth 40 Gbps



- SLA needs per-VM enforcement
- Need to control the aggregate rate of VMs 1-4 that reside on different physical machines

- Static partitioning of bandwidth is sub-optimal

Work-conserving solution



- VMs with traffic demand should be able to send it as long as the aggregate rate does not exceed 40 Gbps
- **Solution:** *Max-min fair sharing*

Max-min fair sharing

Well studied problem in networks

- Existing solutions are distributed
  - Each VM varies its rate based on congestion
  - Converge to max-min sharing
- *Drawbacks*: complex and requires congestion signal

<u>But we have a centralized controller</u>

- Converts to simple algorithm at controller

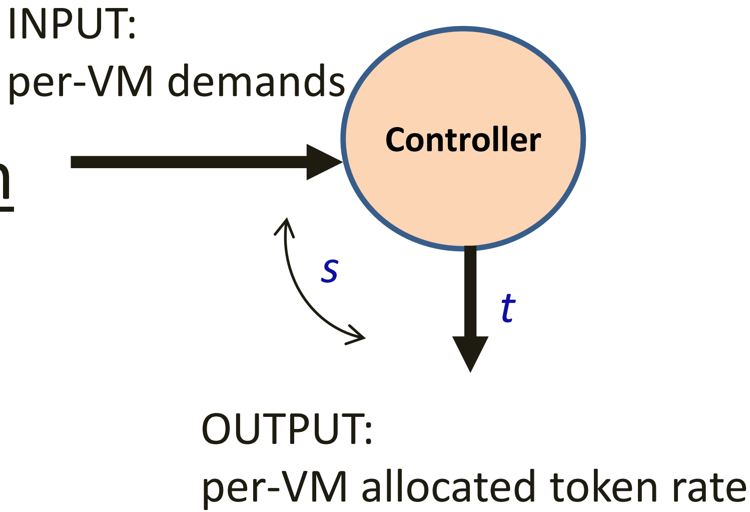
Controller-based max-min fair sharing

t = control interval

s = stats sampling interval

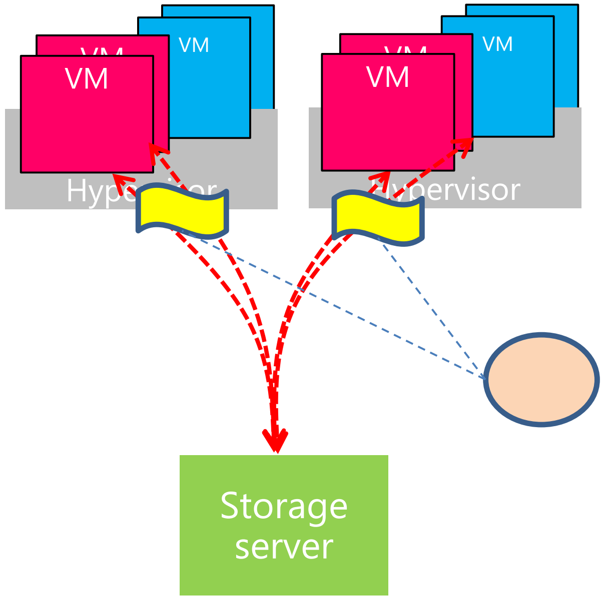
What does controller do?

- Infers VM demands
- Uses centralized max-min <u>within</u> a tenant and <u>across</u> tenants
- Sets VM token rates
- Chooses best place to enforce



Controller decides *where* to enforce

Minimize # times IO is queued and distribute rate limiting load



SLA constraints

- Queues where resources shared
- Bandwidth enforced close to source
- Priority enforced end-to-end

Efficiency considerations

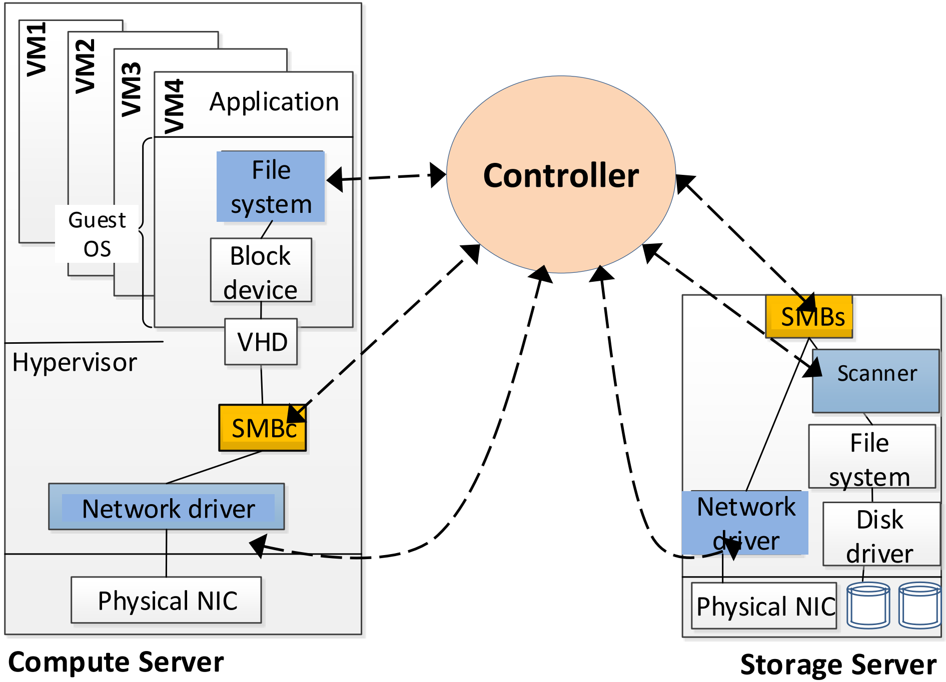
- Overhead in data plane ~ # queues
- Important at 40+ Gbps

Centralized vs. decentralized control

Centralized controller in SDS allows for simple algorithms that focus on SLA enforcement and ***not*** on distributed system challenges

Analogous to benefits of centralized control in software-defined networking (SDN)

IOFlow implementation



2 key layers for VM-to-Storage performance SLAs

4 other layers
. Scanner driver (routing)
. User-level (routing)

. Network driver
. Guest OS file system

Implemented as filter drivers on top of layers

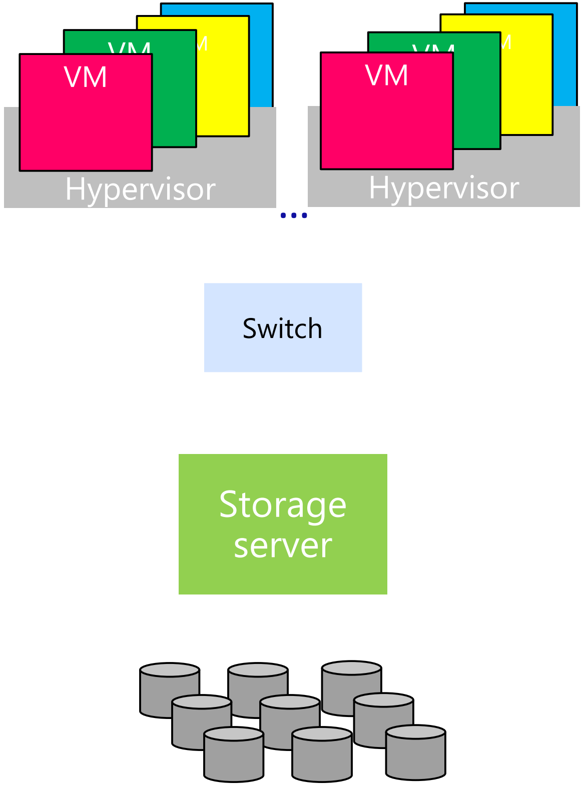
Evaluation map

IOFlow's ability to enforce end-to-end SLAs

- Aggregate bandwidth SLAs
- Priority SLAs and routing application in paper

Performance of data and control planes

Evaluation setup



Clients:10 hypervisor servers, 12 VMs each

4 tenants (Red, Green, Yellow, Blue)

30 VMs/tenant, 3 VMs/tenant/server

Storage network:

Mellanox 40Gbps RDMA RoCE full-duplex

1 storage server:

16 CPUs, 2.4GHz (Dell R720)

SMB 3.0 file server protocol

3 types of backend: RAM, SSDs, Disks

Controller: 1 separate server

1 sec control interval (configurable)

Workloads

- 4 Hotmail tenants {Index, Data, Message, Log}

  Used for trace replay on SSDs (see paper)

- IoMeter is parametrized with Hotmail tenant characteristics (read/write ratio, request size)

	Index	Data	Message	Log
Read %	75%	61%	56%	1%
IO Sizes	4/64 KB	8 KB	4/64 KB	0.5/64 KB
Seq/rand	Mixed	Rand	Rand	Seq
# IOs	32M	158M	36M	54M

Enforcing bandwidth SLAs

4 tenants with different storage bandwidth SLAs

Tenant	SLA
Red	{VM1 – 30} -> Min 800 MB/s
Green	{VM31 – 60} -> Min 800 MB/s
Yellow	{VM61 – 90} -> Min 2500 MB/s
Blue	{VM91 – 120} -> Min 1500 MB/s

Tenants have different workloads

- **Red** tenant is aggressive: generates more requests/second

Things to look for

Distributed enforcement across 4 competing tenants

- Aggressive tenant(s) under control

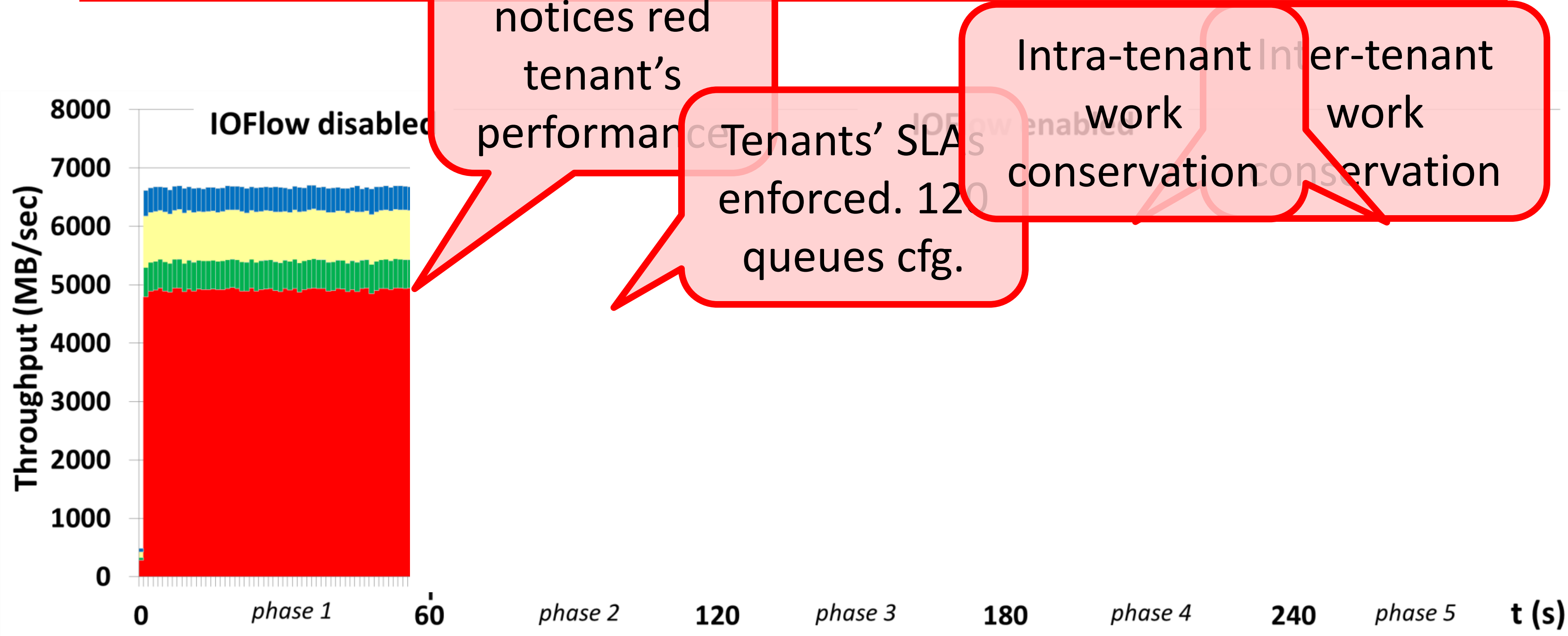
Dynamic inter-tenant work conservation

- Bandwidth released by idle tenant given to active tenants

Dynamic intra-tenant work conservation

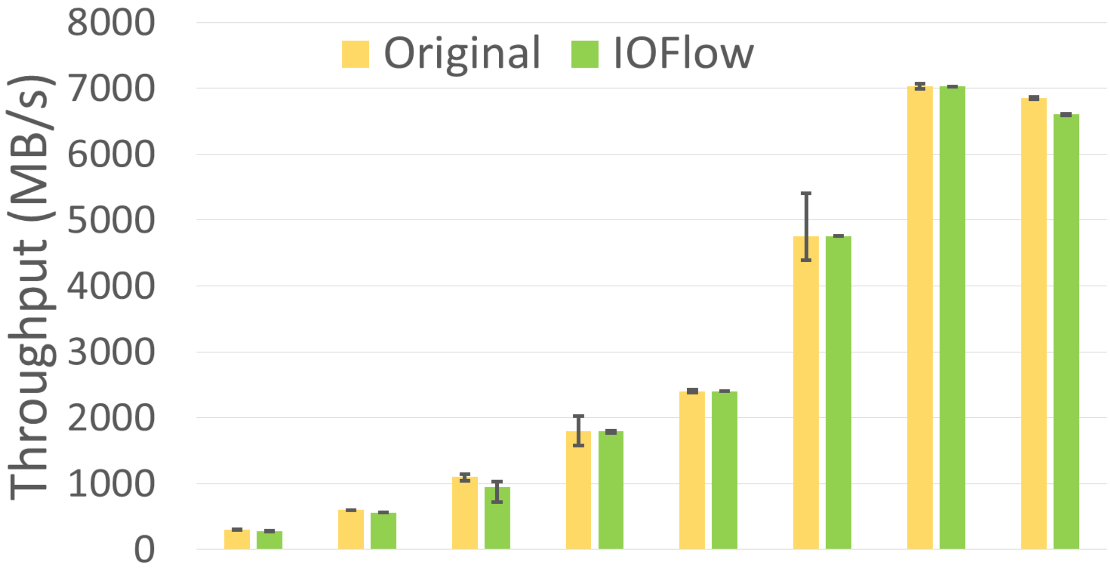
- Bandwidth of tenant’s idle VMs given to its active VMs

Results


Controller notices red tenant's performance
Tenants' SLAs enforced. 120 queues cfg.
Intra-tenant work conservation
Inter-tenant work conservation
IOFlow disabled
IOFlow enabled
Throughput (MB/sec)
8000
7000
6000
5000
4000
3000
2000
1000
0
0
phase 1
60
phase 2
120
phase 3
180
phase 4
240
phase 5
t (s)

Data plane overheads at 40Gbps RDMA

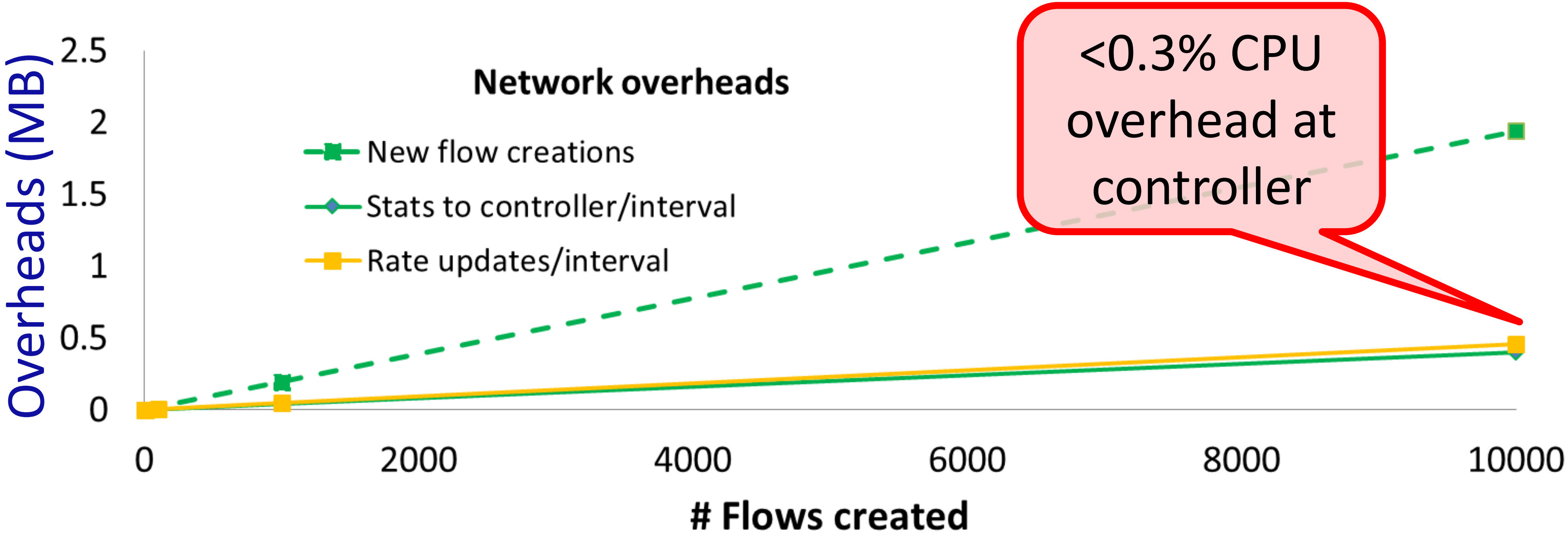
Negligible in previous experiment. To bring out worst case varied IO sizes from 512Bytes to 64KB



Reasonable overheads for enforcing SLAs

Control plane overheads: network and CPU

Controller configures queue rules, receives statistics and updates token rates every interval



Summary of contributions

- Defined and built storage control plane
- Controllable queues in data plane
- Interface between control and data plane (IOFlow API)
- Built centralized control applications that demonstrate power of architecture
- *Ongoing work: applying to public cloud scenarios*

Backup slides

Related work (1)

- Software-defined Networking (SDN)
  - [Casado et al. SIGCOMM’07], [Yan et al. NSDI’07], [Koponen et al. OSDI’10], [Qazi et al. SIGCOMM’13], and more in associated workshops.
  - OpenFlow [McKeown et al. SIGCOMM Comp. Comm.Review’08]
  - Languages and compilers [Ferguson et al. SIGCOMM’13], [Monsanto et al. NSDI’13]
- SEDA [Welsh et al. SOSP’01] and Click [Kohler et al. ACM ToCS’00]

Related work (2)

- Flow name resolution
  - Label IOs [Sambasivan et al. NSDI'11], [Mesnier et al. SOSP'11], etc
- Tenant performance isolation
  - For storage [Wachs et al. FAST'07], [Gulati et al. OSDI'10], [Shue et al. OSDI'12], etc.
  - For networks [Ballani et al. SIGCOMM'11], [Popa et al. SIGCOMM'12]
  - Distributed rate limiting [Raghavan et al. SIGCOMM'07]

IOFlow API

returns kind of IO header layer uses for queuing, the queue properties that are configurable, and possible next hops
getQueueInfo ()

returns queue statistics
getQueueStats (Queue-id *q*)

creates or removes queuing rule i -> q
createQueueRule (IO Header *i*, Queue-id *q*)
removeQueueRule (IO Header *i*, Queue-id *q*)

sets queue service properties
configureQueueService (Queue-id *q*, <token rate,priority, queue size>)

sets queue routing properties
configureQueueRouting (Queue-id *q*, Next-hop stage *s*)

sets storage-specific parameters
configureTokenBucket (Queue-id *q*, <benchmark-results>)

SDS: Storage-specific challenges

Low-level primitives	Old networks	SDN	Storage today	SDS
End-to-end identifier	✓	✓	✗	✓
Data plane queues	✓	✓	✗	✓
Control plane	✓	✓	✗	✓