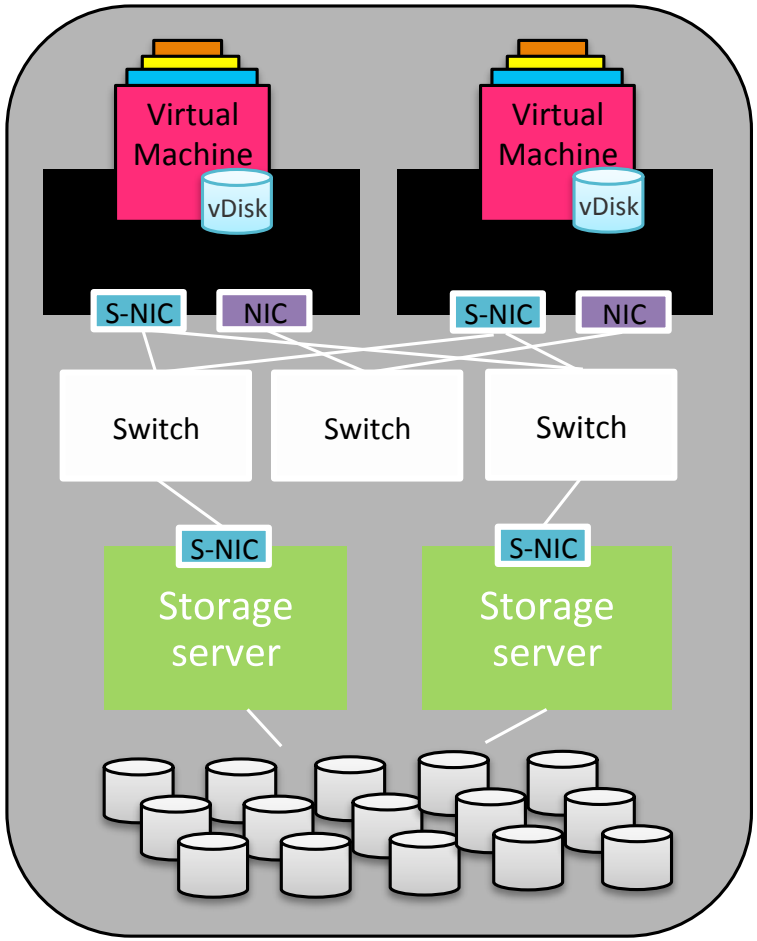
IOFlow

A Software-Defined Storage Architecture

ACM SOSP 2013
Authors: Eno Thereska, Hitesh Ballani, Greg O'Shea, Thomas Karagiannis,
Antony Rowstron, Tom Talpey, and Timothy Zhu

Presented by yifan wu, CS294, UC Berkeley, Sep 21 2015
Some material borrowed from authors' SOSP 2013 presentation

Problem



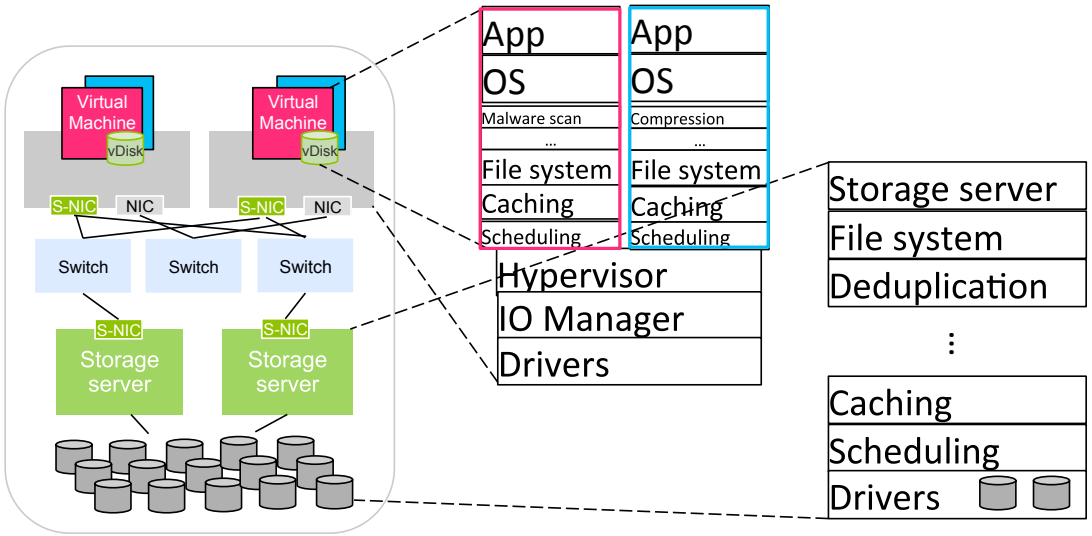
- Who: enterprise data centers (**virtualization** of servers and storage)
- What: **end-to-end** policies (*performance* and *routing*) are hard to enforce

Problem Challenges

1. many operations & layers (e.g. 18?!)

2. non-linear: `f(IO type, data locality, device type, request size) -> processing time`

3. deployability (cannot change existing OSs)



Design Goals

- Enable high-level flow policies, including for multi-point
- **Declarative** interface:

```
{[Set of VMs], [Set of storage shares]} → Policy
```

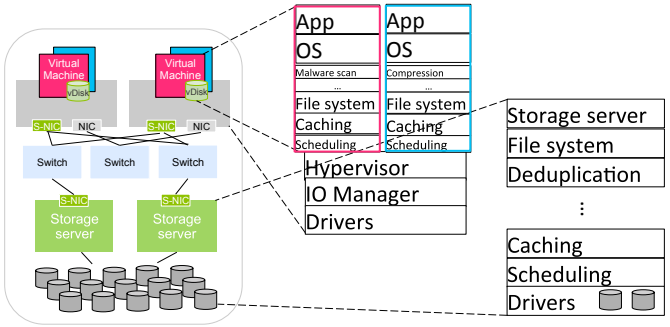
	Policy	Where to enforce?	What to enforce?
P1	$\{p, X\} \rightarrow B$	C(p) Or S(X)	Static rate limit
P2	$\{p, X\} \rightarrow$ Min B	C(p) Or S(X)	Dynamic rate limit
P3	$\{p, X\} \rightarrow$ Sanitize	C(p) Or S(X)	Static routing
P4	$\{p, X\} \rightarrow$ Priority	C(p) & S(X)	Static priority
P5	$\{[p, q, r], [X, Y]\} \rightarrow B$	C(p), C(q) & C(r) Or S(X) & S(Y)	Dynamic VM Or Server rate limits

no one has done it before


CONTROL ALL THE THINGS!
hyperboleandahalf.blogspot.com

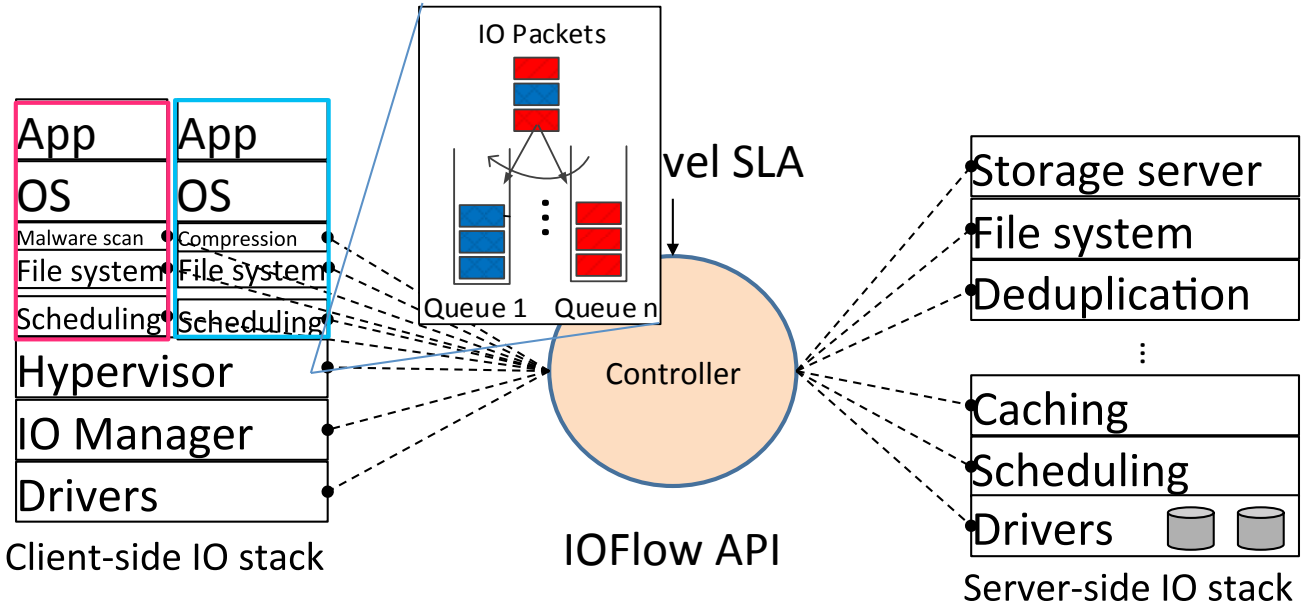
Centralized Control*

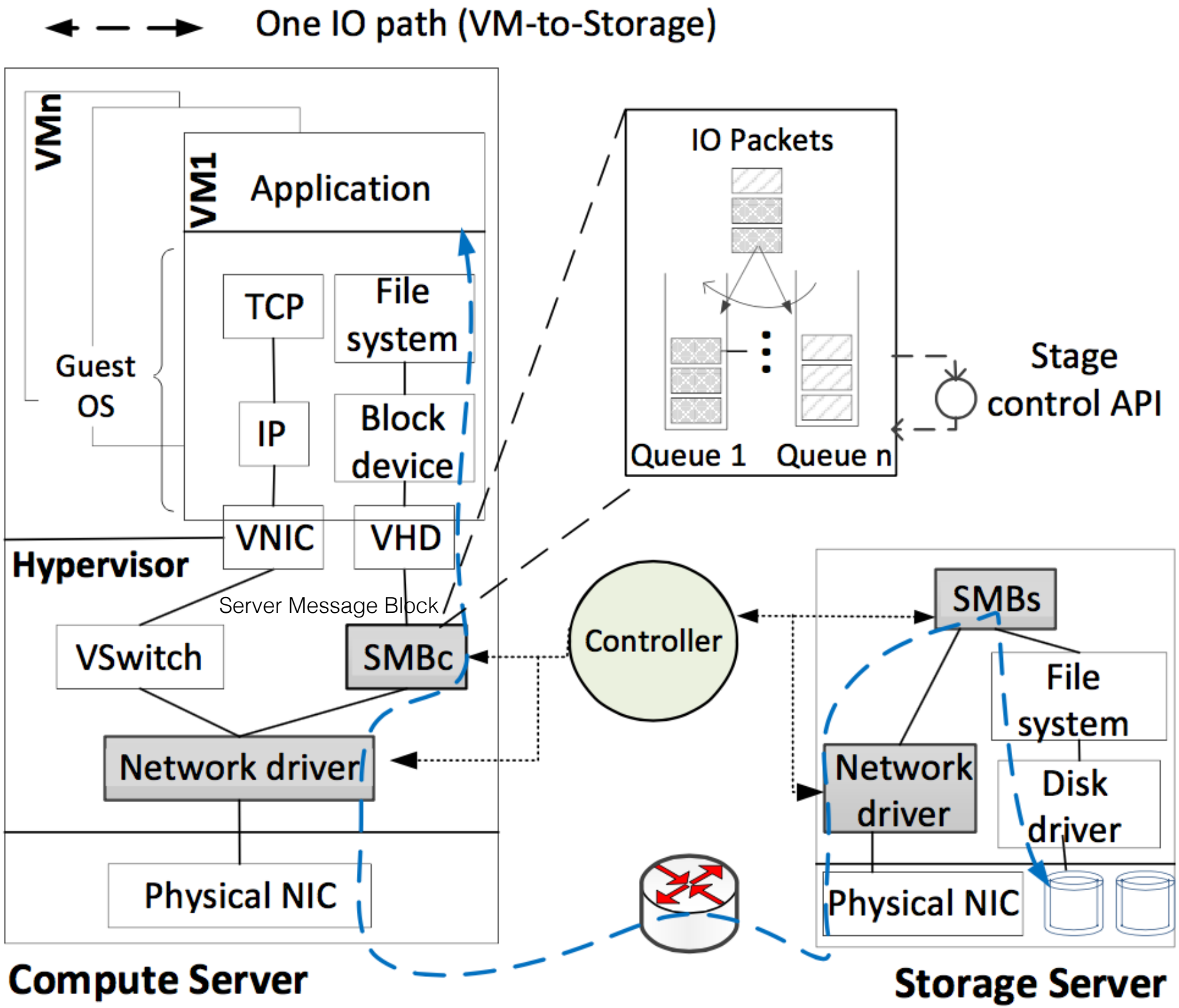
- Inspired from software-defined networking (SDN), thus “software-defined storage architecture”, but it’s much harder
- Stages
  - storage driver in hypervisor
  - storage server
- *But distributed enforcement



Components

1. logically centralized controller
2. data-plane queues
3. interface between the controller and control applications (*visibility*!)



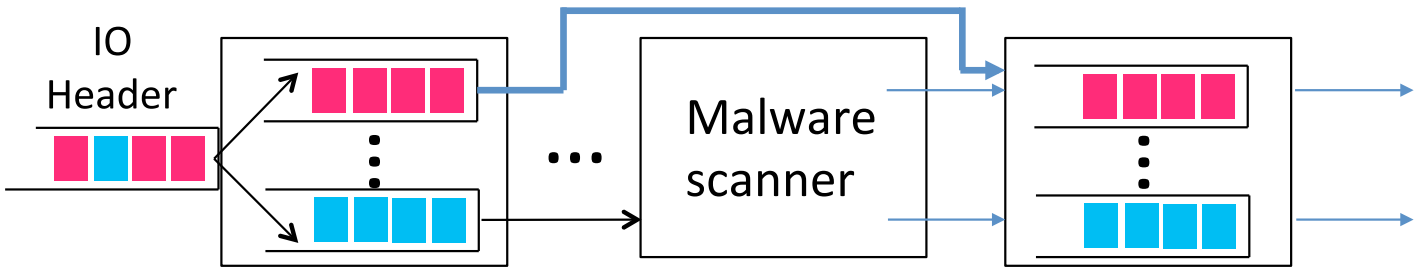

One IO path (VM-to-Storage)
VMn
VM1
Application
Guest OS
TCP
File system
IP
Block device
VNIC
VHD
Hypervisor
Server Message Block
VSwitch
SMBc
Network driver
Physical NIC
Compute Server
IO Packets
Queue 1
Queue n
Stage control API
Controller
SMBs
File system
Network driver
Disk driver
Physical NIC
Storage Server

Design Challenges

- Admission control: is the policy **possible**?
- **Distributed** enforcement: **differentiated** costs at different stages
  - access SQL data files: from guest OS > hypervisor > network switch > storage server OS > disk array
- Performance: queues at stages must be fast
- Incremental deployability/resilience
- Dynamic control: e.g. min bandwidth

Data Plane Queues

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



Controller API

A0	**`getQueueInfo`** () returns kind of IO header stage uses for queuing, the queue properties that are configurable, and possible next-hop stages
A1	**`getQueueStats`** (Queue-id *q*) returns queue statistics
A2	**`createQueueRule`** (IO Header *i*, Queue-id *q*) creates queuing rule $i \rightarrow q$
A3	**`removeQueueRule`** (IO Header *i*, Queue-id *q*)
A4	**`configureQueueService`** (Queue-id *q*, <token rate,priority, queue size>)
A5	**`configureQueueRouting`** (Queue-id *q*, Next-hop stage *s*)
A6	**`configureTokenBucket`** (Queue-id *q*, <benchmark-results>)

flexible

responsive

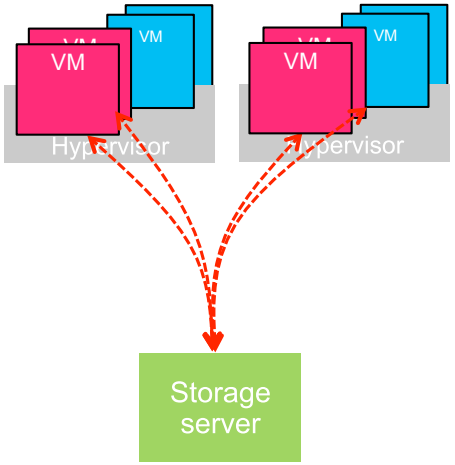
accurate

resilient

scalable

Rate Limiting

- Service request: token-bucket abstraction (simple)
- Storage request: **benchmarks** the storage devices to measure the cost of IO requests as a function of their type and size, *IoMeter.* *(RAM, SSDs, disks: read/write ratio, request size)*
- Max-min fair sharing



Tricks

- Split IO requests into smaller buckets
- Zero-copying of requests moving from stage to stage
- Min-heap-based selection of which queue to service next within a stage
- Heuristics for deciding enforcement layer

```
1: getQueueInfo (); returns "File IO"
2: createQueueRule (<VM 4, //server X/*>, Q1)
3: createQueueRule (<*, *>, Q0)
4: configureQueueService (Q1, <B,0,1000>)
5: configureQueueService (Q0, <C-B,0,1000>)
```

Controller

Blocking, but it doesn't really matter

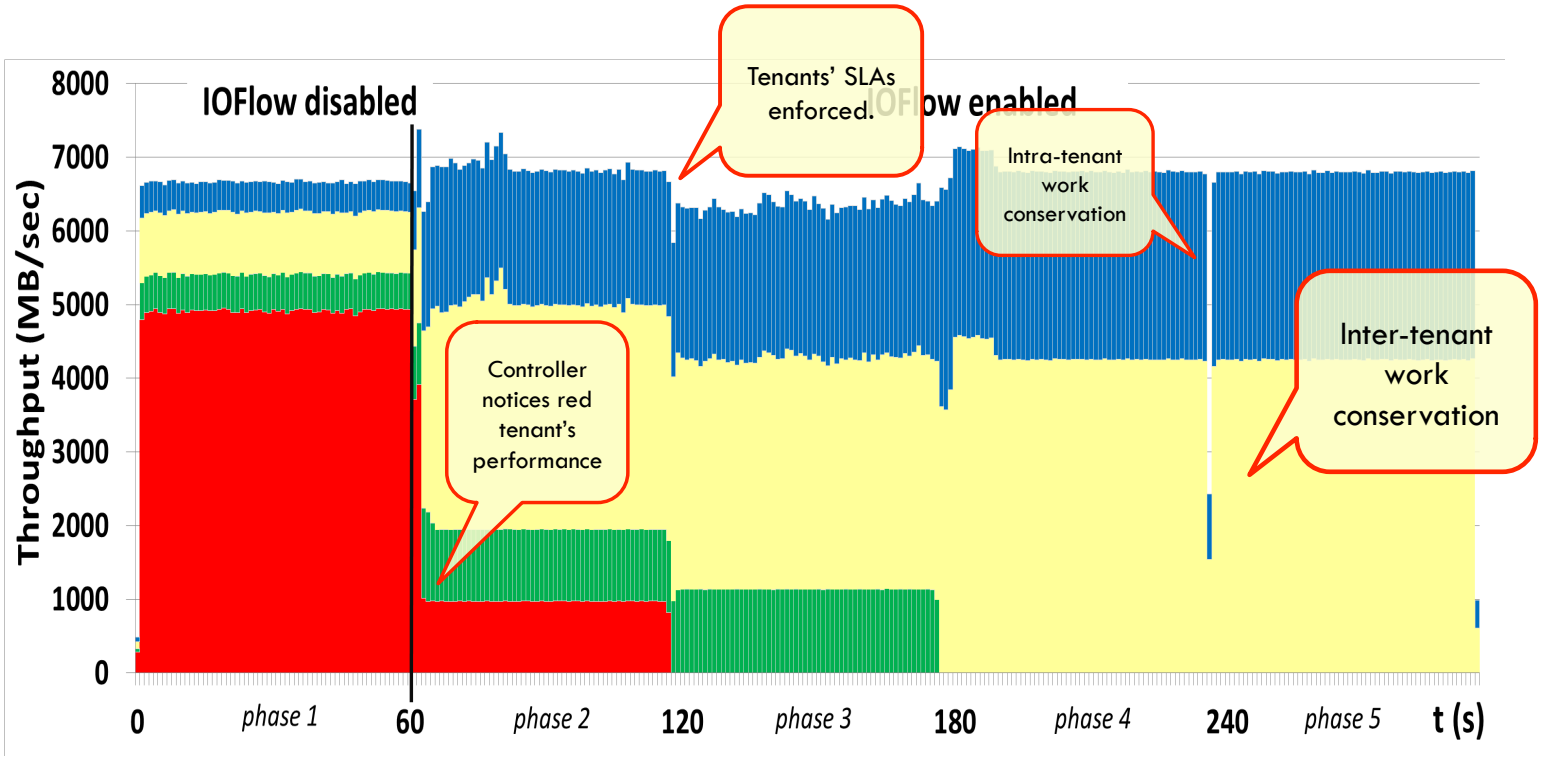
```
1: IO Header <VM1, //server X/file F> → Queue Q1
2: IO Header <VM2, //server Y/*> → Queue Q2
3: IO Header <VM3, *> → Queue Q4
4: <*, *> → Queue Q3
```

Hypervisor

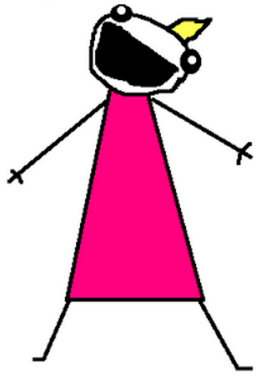
```
1: IO Header <SID S1, H:/File F> → Queue Q1
2: IO Header <SID S2, H:/File G> → Queue Q1
3: IO Header <SID S2, H:/Directory A/*> → Queue Q2
4: <*, *> → Queue Q3
```

Storage Server

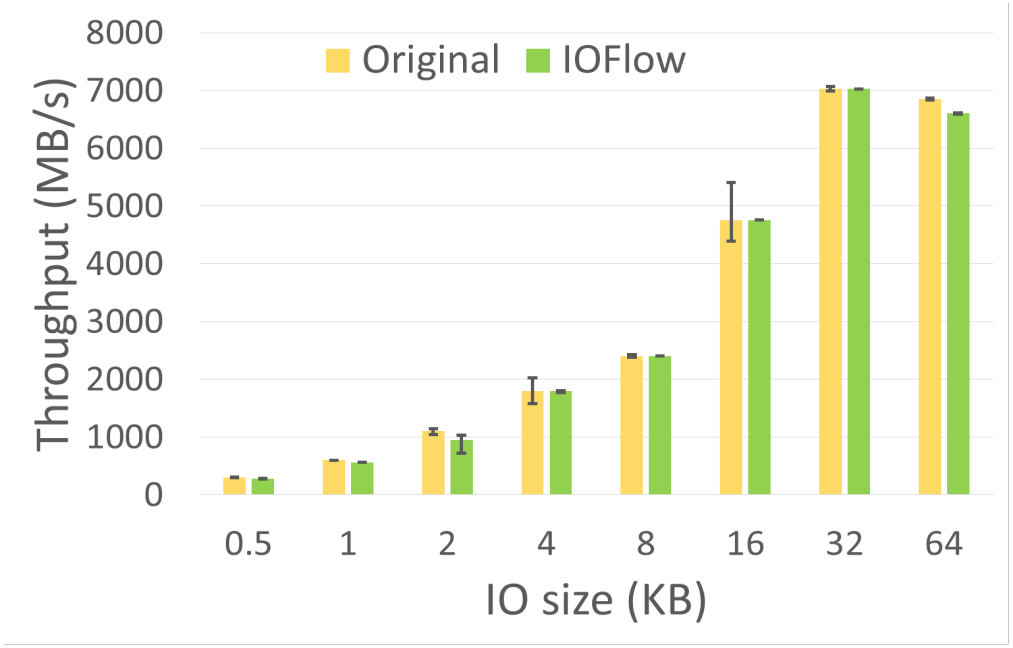
Performance



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

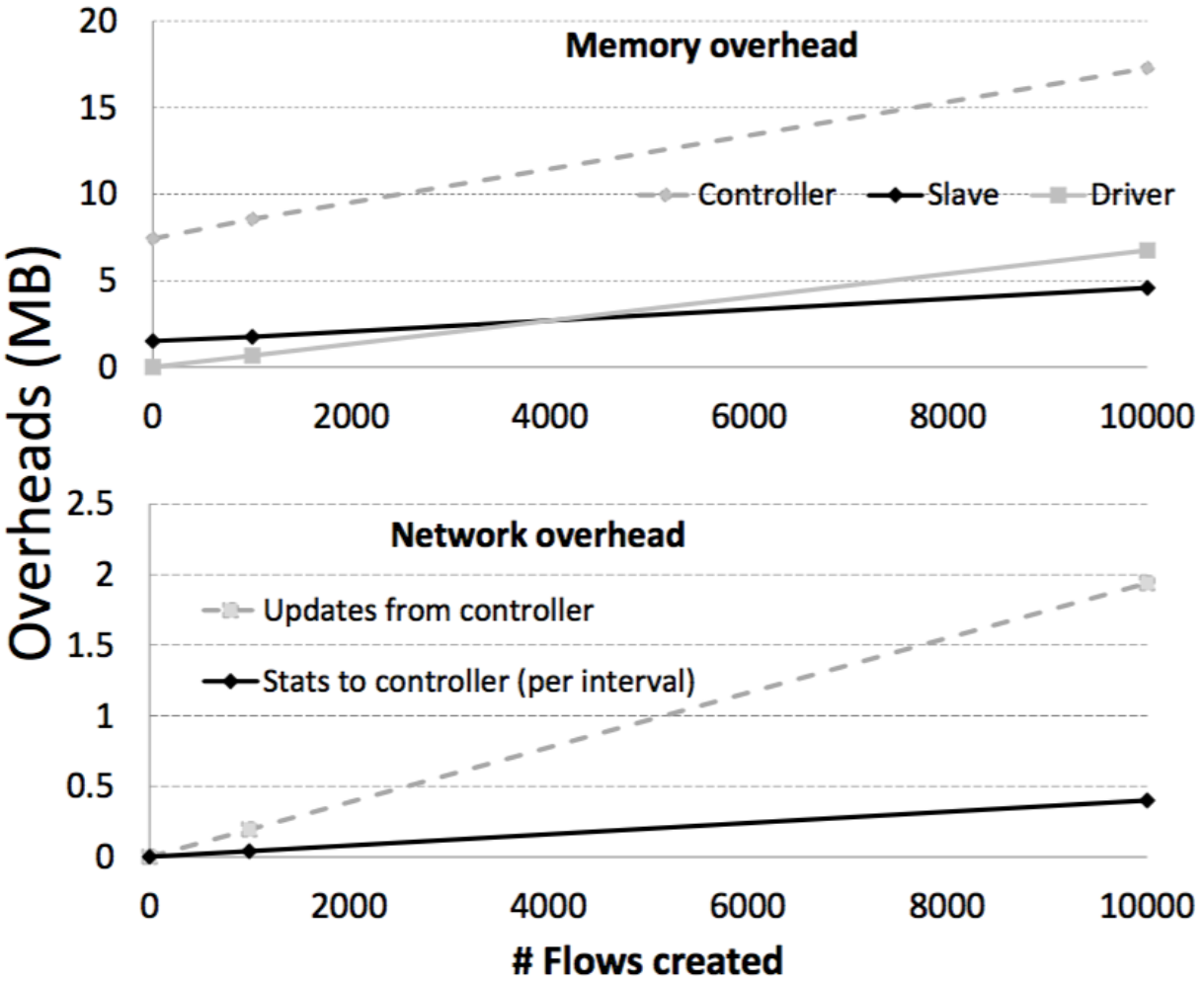


Data Plane Overhead



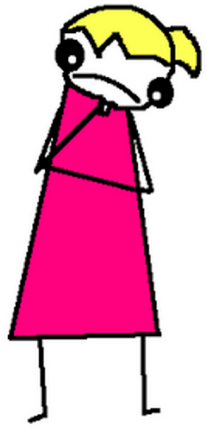
- Worst case CPU consumption at hypervisor is less than 5%
- Worst case reduction in throughput:
  RAM – 14%
  SSD – 9%
  Disk – 5%

Control Plane Overhead



Checking Assumptions

- performance bottleneck is at the storage servers
- small IO requests are typically interrupt limited
- large requests are limited by the bandwidth of the storage back-end or the server's network link

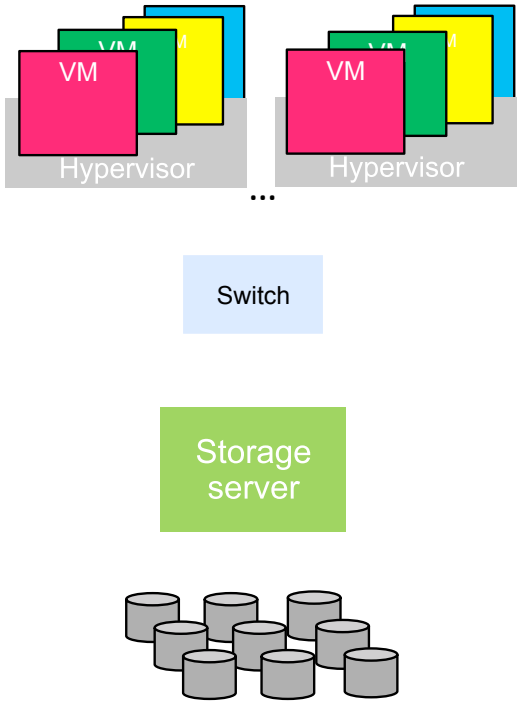


Questions/Ideas

- What more does it take to **scale**?
  - Controller bottleneck?
  - Ad hoc access patterns (bad prediction)
- Too many nobs, hard to tune?
- Extend to optimize for co-requests?

IOFlow

Experiment Setup



Clients:10 hypervisor servers, 12 VMs each
4 tenants (Red, Green, Yellow, Blue)
30 VMs/tenant, 3 VMs/tenant/server

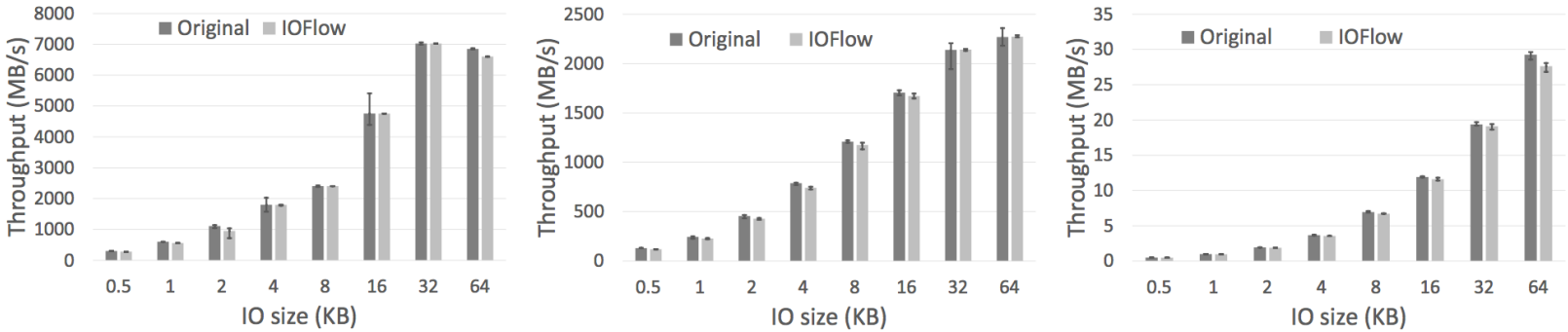
Storage network:
Mellanox 40Gbps RDMA RoCE full-duplex

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)

Resilience/Consitency

- When controller is unreachable, use default policy.
- Allow for inconsistencies



(a) RAM store (b) SSD store (c) Disk store