IOFlow

A Software-Defined Storage Architecture

Jakub Łuczyński

October 28, 2015

So, what is the problem?

- how to enforce high level flow policies
- we want to dictate storage IO
  - performance (bandwidth/tenant guarantee)
  - routing (additional processing middlebox)

Scope

- Enterprise Data Center
- Compute Servers and Storage Servers
- virtualized (VMs, VHDs)
- small/medium Data Centers
  $\sim$ 10 SS, $\sim$ 100 CS, 8-16 VMs/CS
  $O(thousand)$ VMs
- scaling defered to future work
- assumed performance bottleneck in storage

Policies

$\{[\text{set of VMs}], [\text{set of storage shares}]\} \rightarrow \textit{Policy}$

	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

Figure : C(p) - Compute Server, S(X) - Storage Server

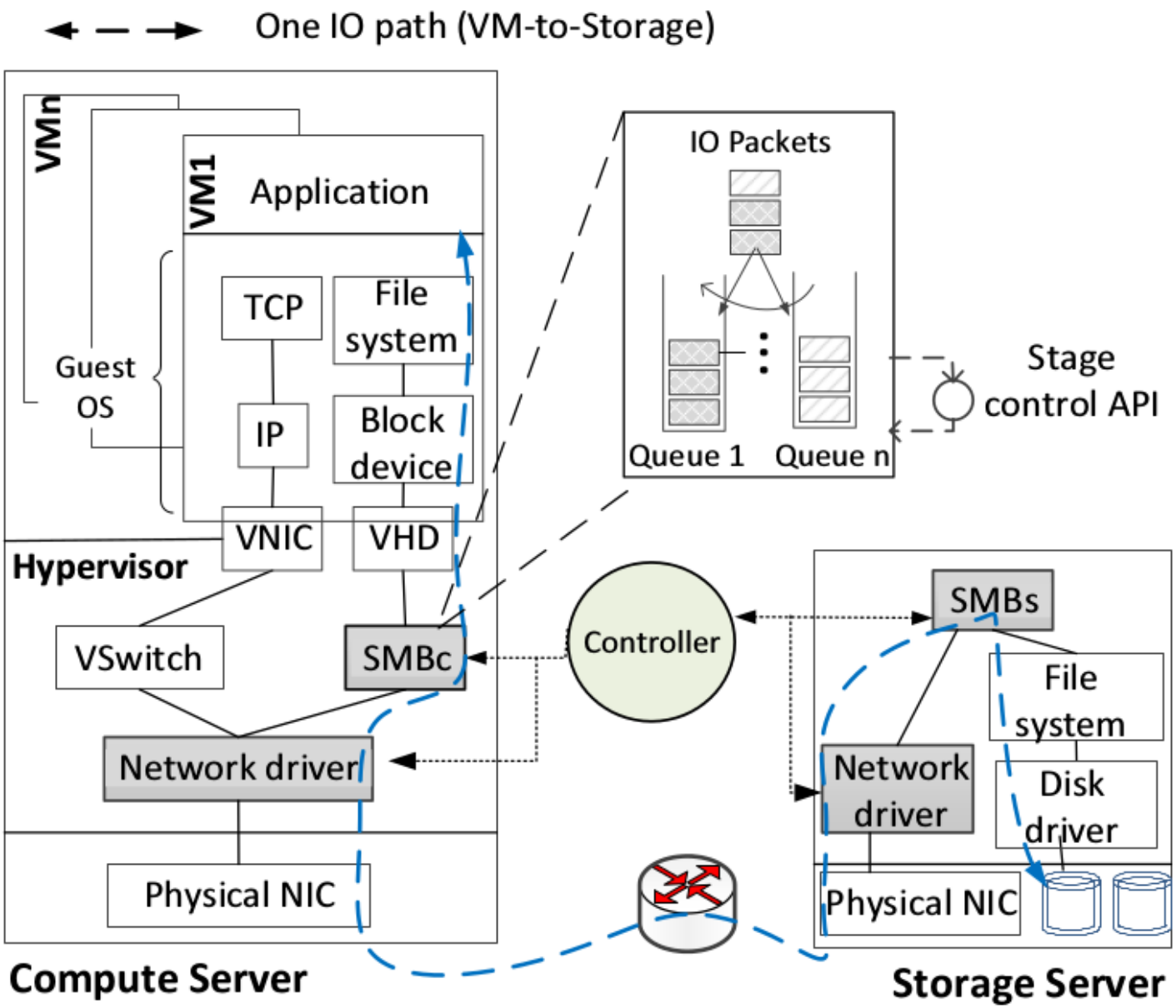
Challenges

- Differentiated treatment
- Flow name resolution
- Distributed enforcement
- Dynamic enforcement
- Admission control

Summarizing

- Traffic differentiation and global visibility at control plane.
- This motivates controller base design

Design


One IO path (VM-to-Storage)
VMn
VM1
Application
TCP
File system
Guest OS
IP
Block device
VNIC
VHD
Hypervisor
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

- centralized controller
- control apps
  translate policies to state configurations (queueing rules)
- stages - layers implementing API (7 calls)
- stages in IOFlow prototype
  - SMBc at Hypervisor
  - SMBs at Storage Server
  - network drivers
- SMB over RDMA (client-server)

Stage

- traffic differentiation through queues
- maps each IO request to queue
- queue contol
  - how fast requests are serviced (service)
  - next stage IO requests are routed to (routing)

Controller

- discovers and interacts with the stages
- maintain topology graph
- exposes those to control applications

IOFlow's API for stage

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>)

Example

Single policy:

- $\{\text{VM 4}, \text{Share X}\} \rightarrow$ Bandwith B

Calls at SMBc stage:

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

Design goals

- stages
  - fast, cause minimal performance degradation
- control plane
  - flexible
  - responsive,
  - accurate,
  - resilient,
  - scalable
- not to require any application or VM changes

Stages

```
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
```

Figure : example SMBc quering rule

```
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
```

Figure : example SMBs quering rule

- queuing rules checked in order they were created
- default rule checked last
- if no match exists, the request is blocked and controller is requested for matching rule

Stages

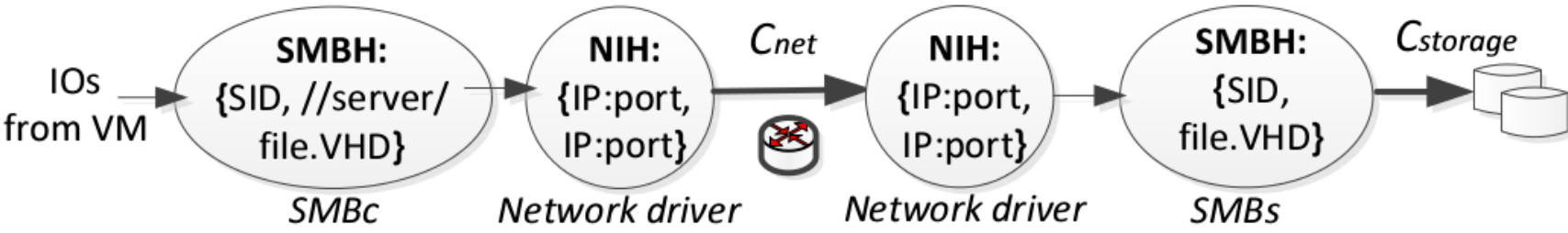
- queues and queuing rules are soft state
- throttle or treat IO requests preferentially
- queues are tocken-bucket abstraction
- properties (configureQueueService)
  <token rate, priority, queue size>
- pecularities of IO requests
  - different operation types (create, read, write)
  - processing at storaage-backand is variable
  - f(operation type, tokens) may not be linear
  - other factors: data locality, device type..

Some Solutions

- contoller discovery component
  - benchmarks the storage devices
  - done periodically
  - configureTokenBucket()
- split requests
  - reduces performance uncertainty

Controler

- present information to controll applications (discovery)



- converts policies into stage specific configurations
- flow name resolution
- soft state
- keeps track of version number
- request ordering (not changed, only delayed)

Control Applications

2 control applications implemented

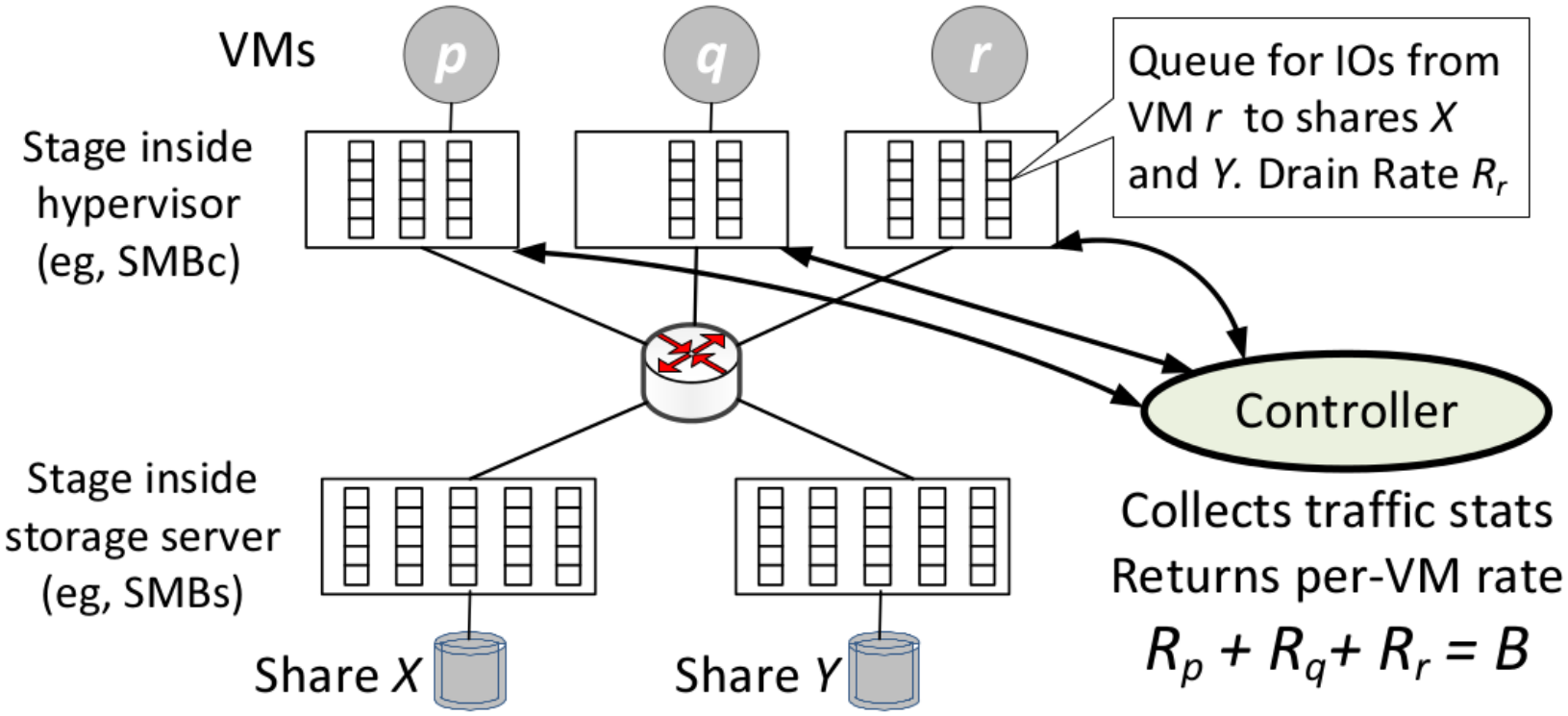
- performance contol (P2, P4, P5)
- routing contol (P3)

Performance Control

Let's take P5 as an example:

$\{[p, q, r], [X, Y]\} \rightarrow B$

- admission controll



Performance Control

- enforcement

Algorithm 4.1 Controller-based distributed rate limiting

Require: N VMs with aggregate guarantee B; D: set of VM demands sorted in ascending order ($D_i > 0$); VM i's IOs are queued at q_i

Ensure: Assign rate R_i to VM i s.t. $\sum R_i = B$

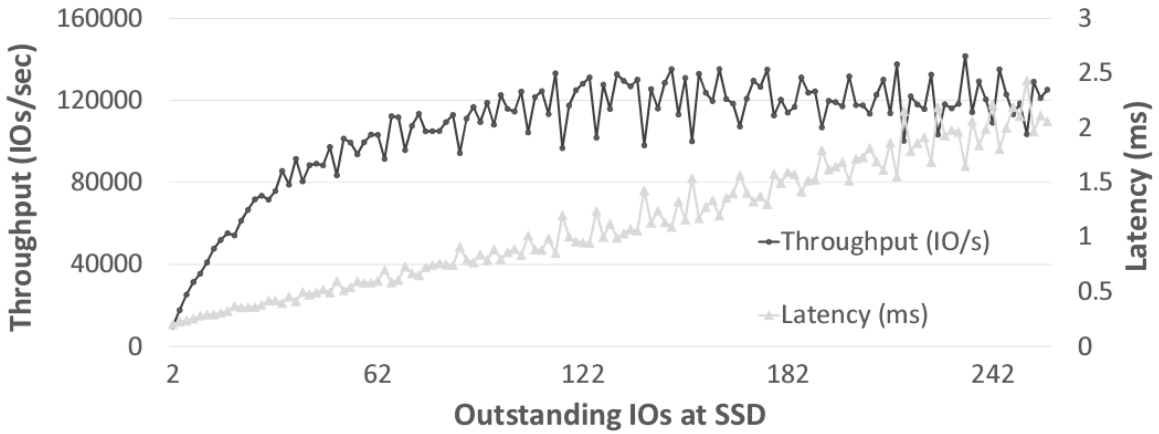
```
leftB = B    // bandwidth left to distribute
for i in [0, N - 1] do
    if D_i ≤ leftB/(N−i) then
        R_i = D_i          // VM demand is less than fair share
    else
        R_i = leftB/(N−i)  // VM demand is more than fair share
    leftB -= R_i

{share any left bandwidth and configure queues}
for i in [0, N - 1] do
    R_i += leftB / N
    configureQueueService(q_i, < R_i, 0, 1000 >)
```

Performance Control

- dynamic enforcement
- demand estimation (periodically)
- VMs with no IOs are estimated to low value (not 0)

- priority
  - requires support from all stages
  - IOFlow tolerate layers not being stages (ex SSD)



90 requests to utilize 95% throughput

Control Applications

Routing Control

- malware scanning middlebox
- scanning stage is optional (untrusted VMs)

Control Applications

Transient Controller Failure

- no impact on system correctness
- can lead to temporary degraded performance
- default policy in case the contoller is unreachable
  ex. best effort queue, no scan etc

Implementation

- Windows-based IO stack
- drivers (C, 22 kLOC)
  - 2 storage drivers (SMBc SMBs)
  - network driver
  - malware scanning device driver
  - optional guest os driver (ioctl trick)
- controller (C#, 3 kLOC), single server

Evaluation

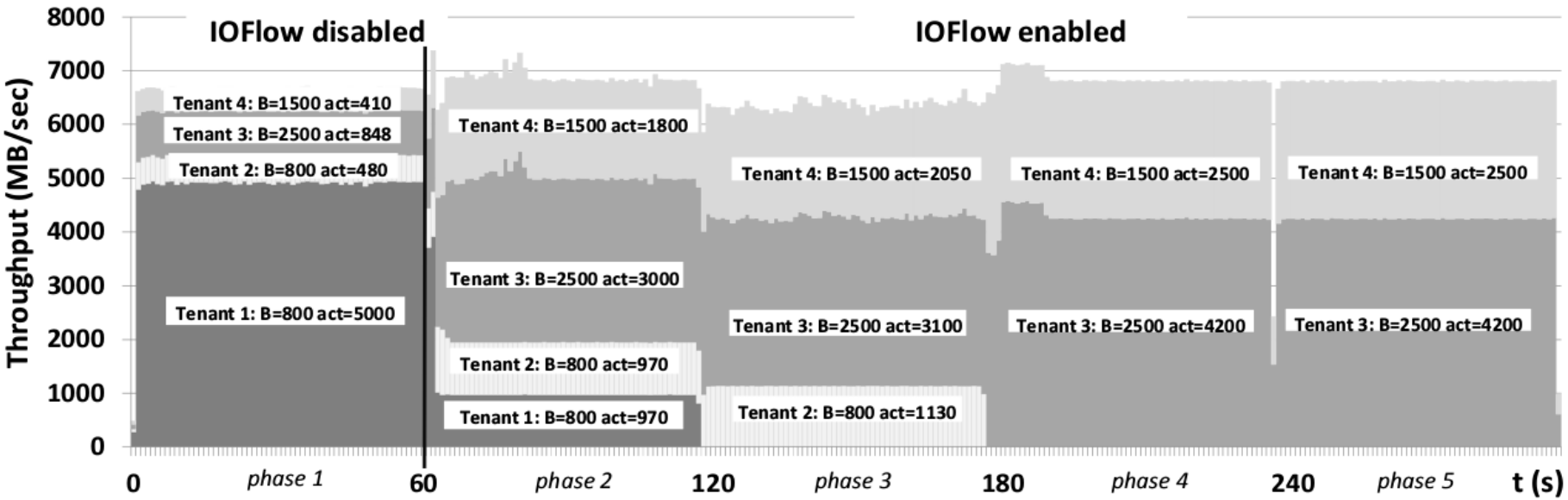


Figure : 4 tenants using 120 VMs in total across 10 hypervisors with policies. When IOFlow is dis- abled tenant policies B are not met. With IOFlow enabled tenant policies are met ("actual" $\geq$ B). Furthermore, extra capacity is assigned in proportion to the tenant minimum bandwidth

Evaluation

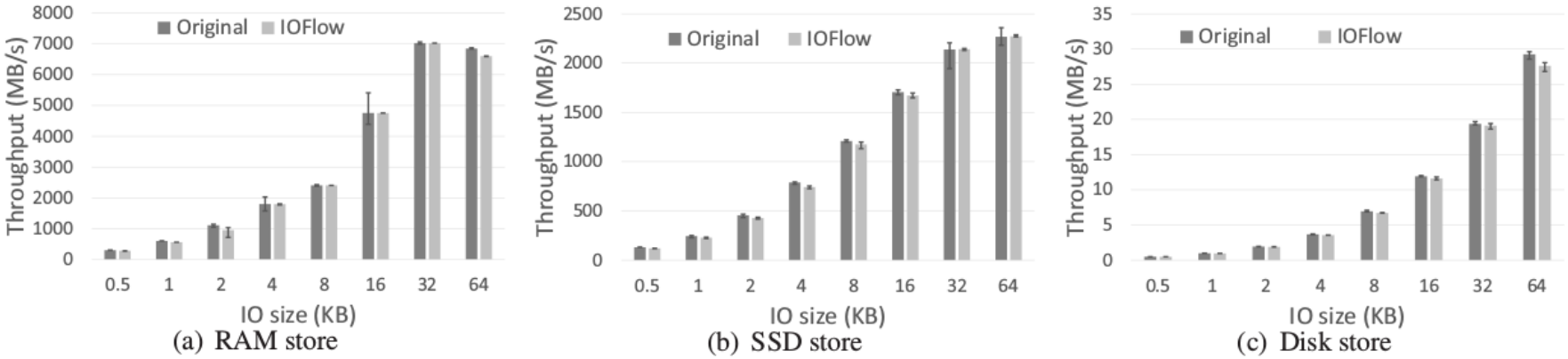


Figure : Performance overheads of IOFlow when compared to unmodified storage stack. Error bars show minimum and maximum values from 5 runs. Note that y-axis is different for each graph.

Evaluation

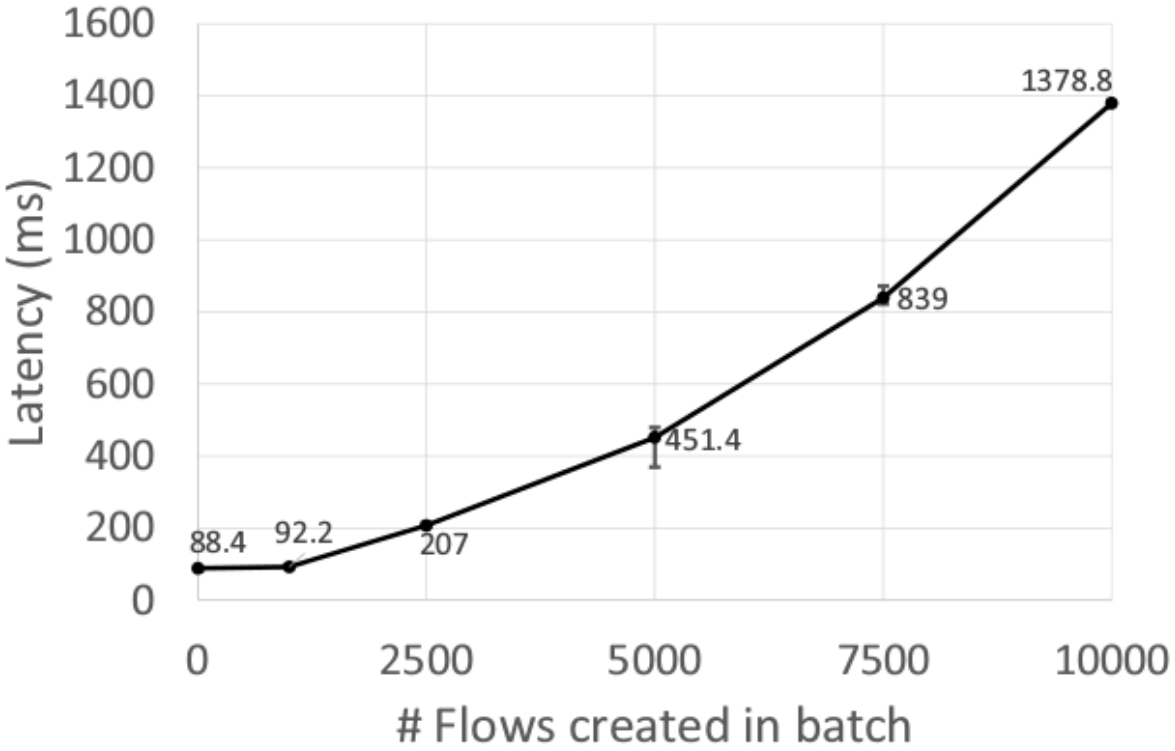


Figure : Average flow creation latency. Error bars show minimum and maximum values from 5 runs.

Evaluation

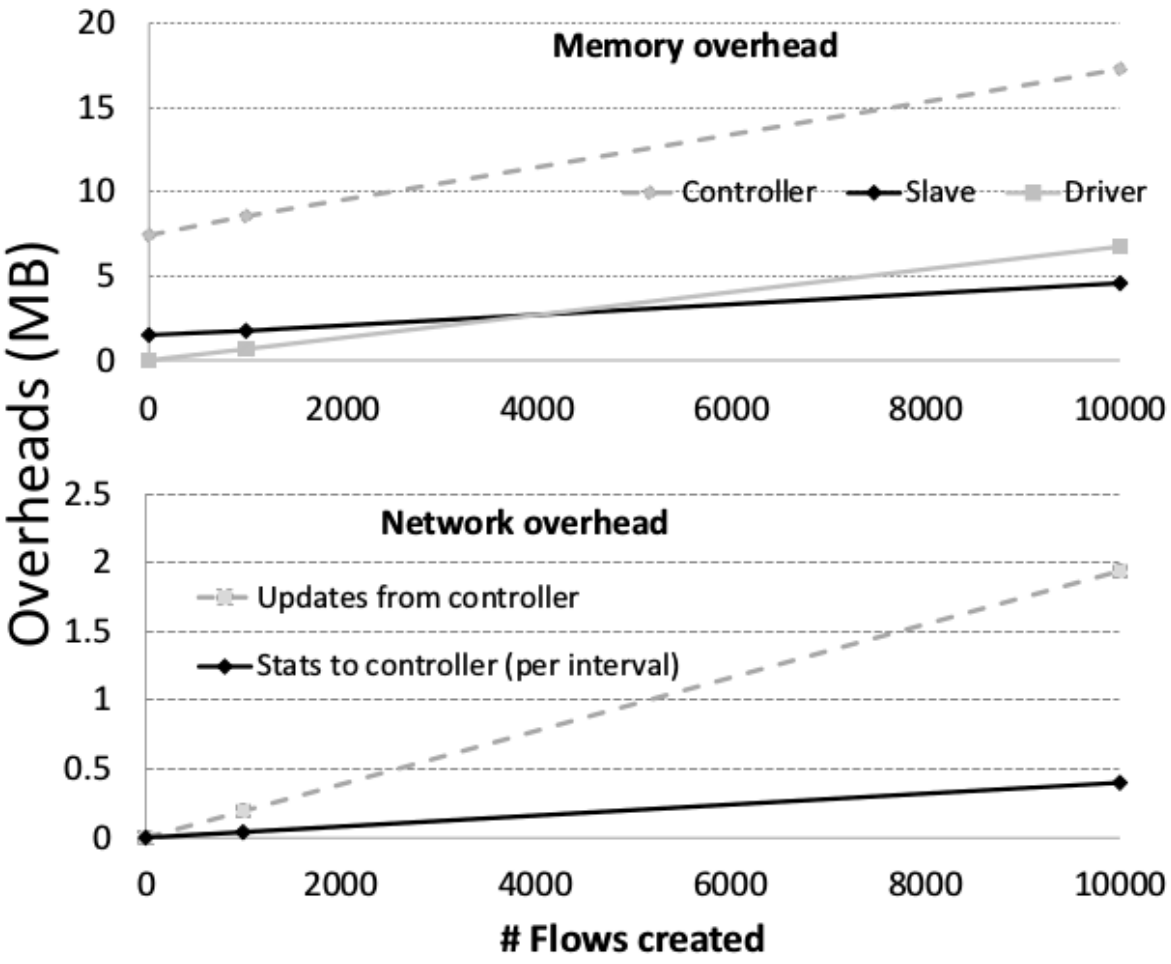


Figure : Memory and network overheads associ- ated with creating flows and getting flow statistics.