Snap: a Microkernel Approach to Host Networking

Qianqian Tan

Snap: a Microkernel Approach to Host Networking

Snap: a Microkernel Approach to Host Networking

Authors

Michael Marty Marc de Kruijf Jacob
Adriaens
Christopher Alfeld Sean Bauer Carlo Contavalli
Mike Dalton Nandita Dukkipati William C. Evans
Steve Gribble Nicholas Kidd Roman
Kononov
Gautam Kumar Carl Mauer Emily Musick
Lena Olson Mike Ryan Erik Rubow
Kevin Springborn Paul Turner Valas Valancius
Xi Wang Amin Vahdat

Authors

Michael Marty Marc de Kruijf Jacob Adriaens
Christopher Alfeld Sean Bauer Carlo Contavalli
Mike Dalton Nandita Dukkipati William C. Evans
Steve Gribble Nicholas Kidd Roman Kononov
Gautam Kumar Carl Mauer Emily Musick
Lena Olson Mike Ryan Erik Rubow
Kevin Springborn Paul Turner Valas Valancius
Xi Wang Amin Vahdat

Slides largely borrowed from the SOSP talk
https://www.youtube.com/watch?v=S2YJY398oZk

Quick look

- Snap:Framework for packet processing in software
  - Goals: Performance and Deployment Velocity
  - Technique: Microkernel-inspired userspace approach

Quick look

- Snap:Framework for packet processing in software
  - Goals: Performance and Deployment Velocity
  - Technique: Microkernel-inspired userspace approach
- Supports multiple use cases
  - Andromeda: Network virtualization for Google Cloud Platform [NSDI 2018]
  - Espresso: Edge networking [SIGCOMM 2017]
  - Maglev: L4 load balancer [NSDI’16]
  - New: High-performance host communication with “Pony Express”
- 3x throughput efficiency (vs kernel TCP), 5MIOPS, and weekly releases

Part 1: Motivation

Motivation

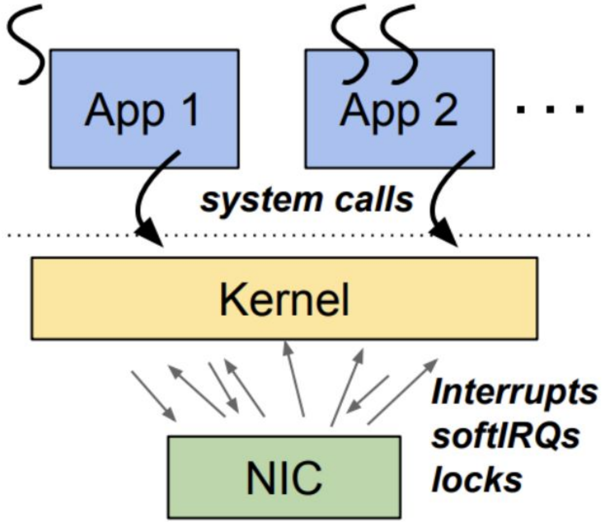
- Growing performance-demanding packet processing needs at Google

- The ability to rapidly develop and deploy new features is just as important!

Monolithic (Linux) Kernel

Deployment Velocity:

- Smaller pool of software developers
- More challenging development environment
- Must drain and reboot a machine to roll out new version
- Typically months to release new feature



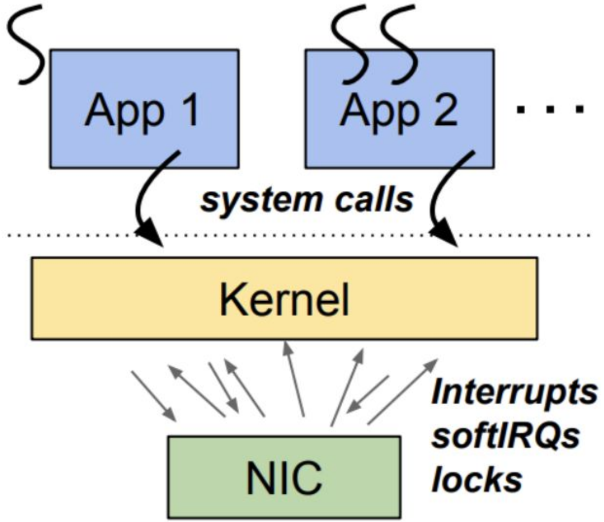
Monolithic (Linux) Kernel

Deployment Velocity:

- Smaller pool of software developers
- More challenging development environment
- Must drain and reboot a machine to roll out new version
- Typically months to release new feature

Performance:

- Overheads from system calls,fine-grained synchronization, interrupts, and more.



LibraryOS and OS Bypass

Networking logic in application binaries

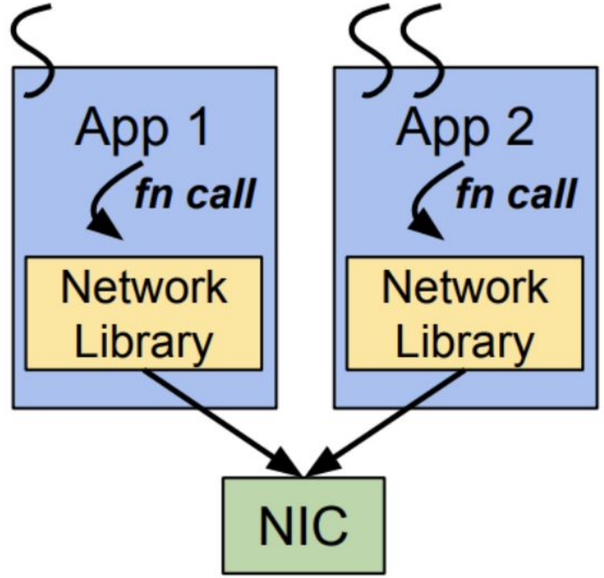
Examples: Arrakis, mTCP, IX, ZygOS, and more

Deployment Velocity:

- Difficult to release changes to the fleet
- App binaries may go months between releases

Performance:

- Can be very fast
- But typically requires spin-polling in every application
- Benefits of centralization (i.e.,scheduling) lost
- Delegates all policy to NIC



Microkernel Approach

Hoists functionality to a separate userspace process

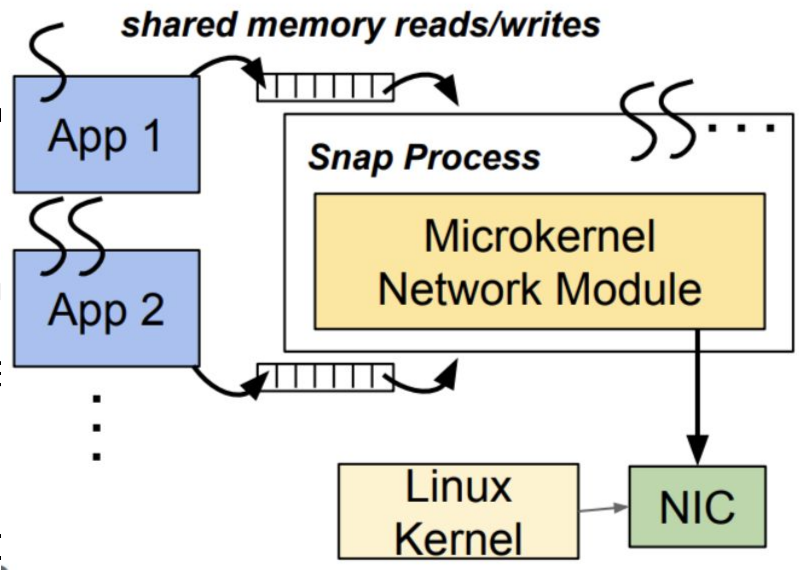
Deployment Velocity:

- Decouples release cycles from application and kernel binaries
- Transparent upgrade with iterative state transfer

Performance:

- Fast! Leverages kernel bypass and many-core CPUs
- Maintains centralization of a kernel
  - Can implement rich scheduling/multiplexing policies

Microkernel Approach



Hoists functionality to a separate userspace pr

Deployment Velocity:

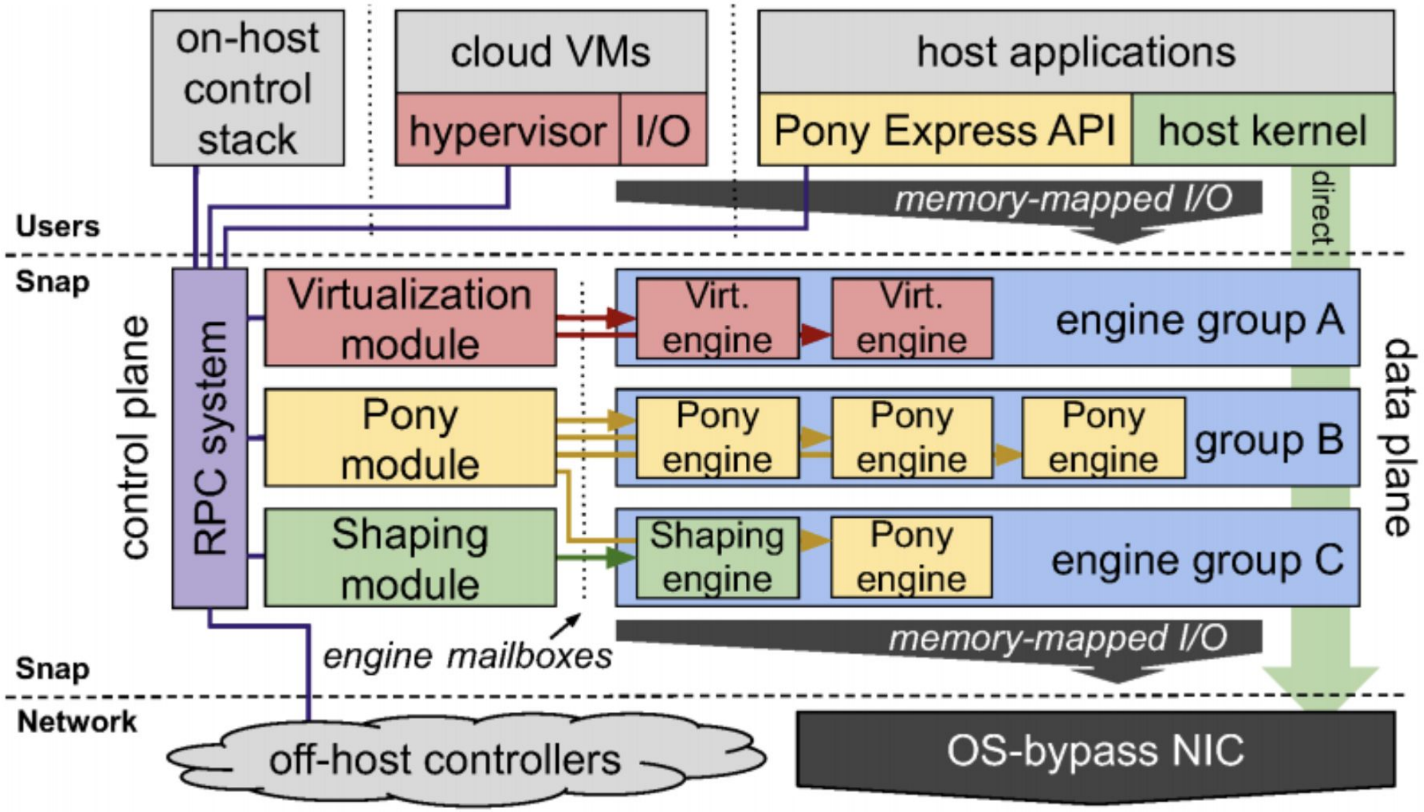
- Decouples release cycles from application
- Transparent upgrade with iterative state tra

Performance:

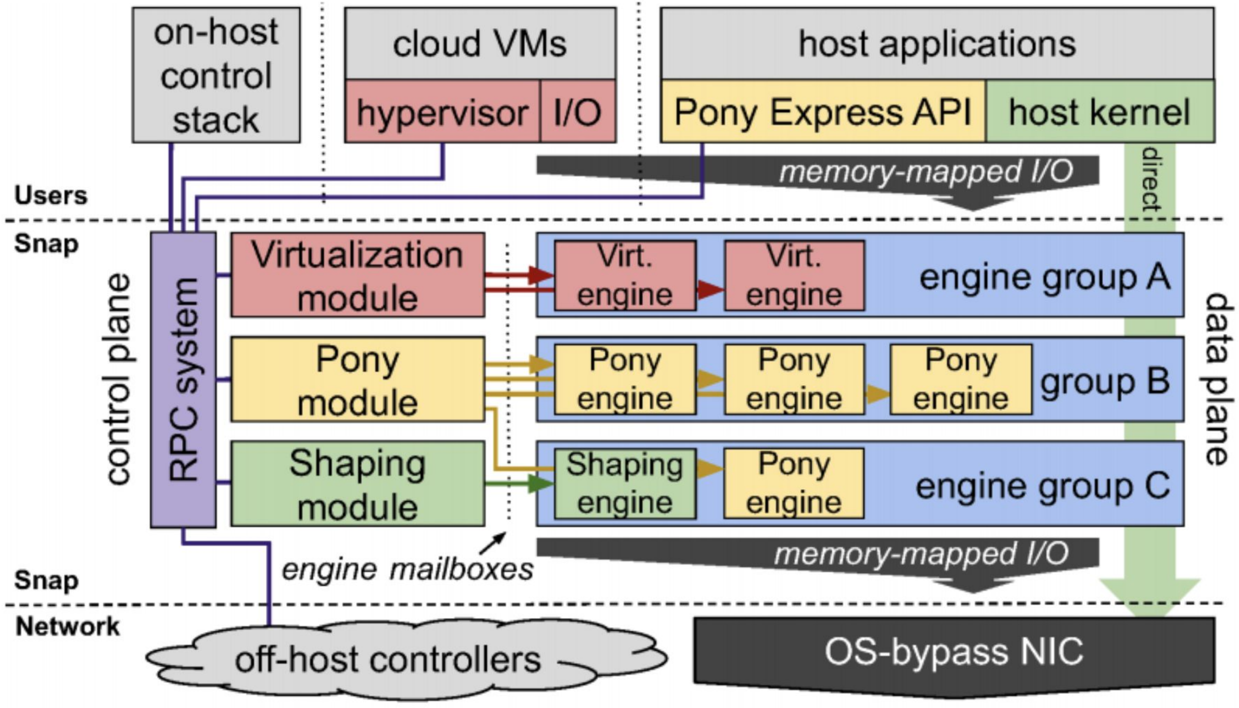
- Fast! Leverages kernel bypass and many-c
- Maintains centralization of a kernel
  - Can implement rich scheduling/multiplexing policies

Part 2: Design

Snap Architecture



Snap Architecture



Snap engine

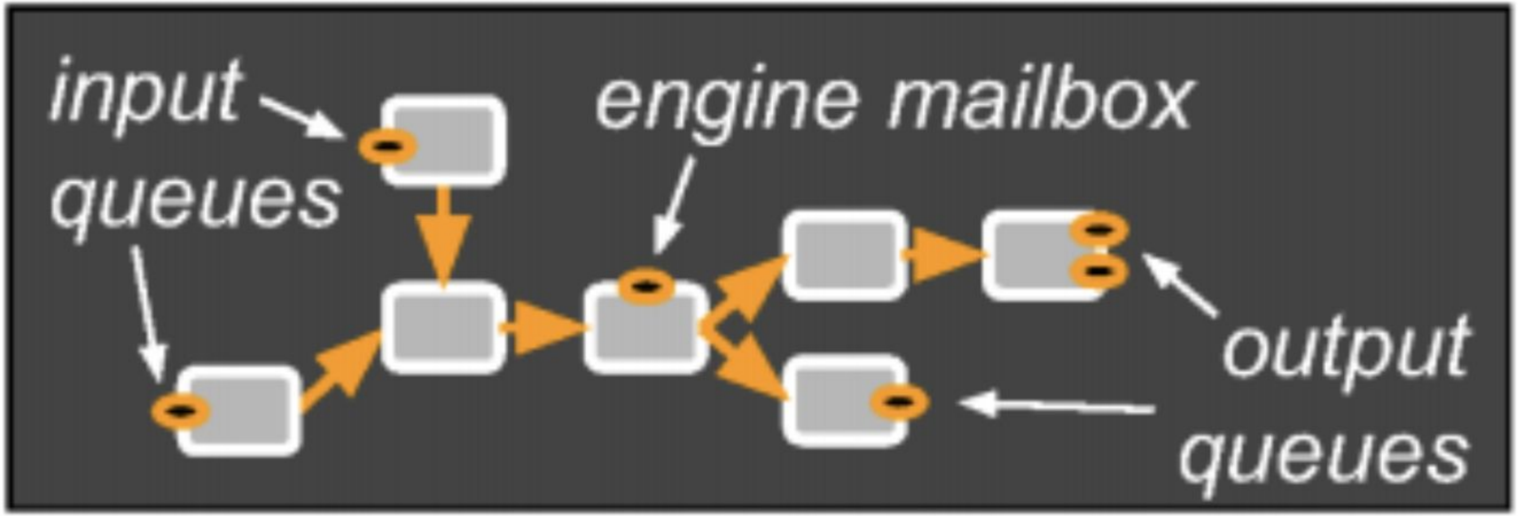
- Key dataplane element
- Implements packet processing piplines
- Unit of CPU scaling

Snap engines implement a Run() menthod invoke by Engine Threads

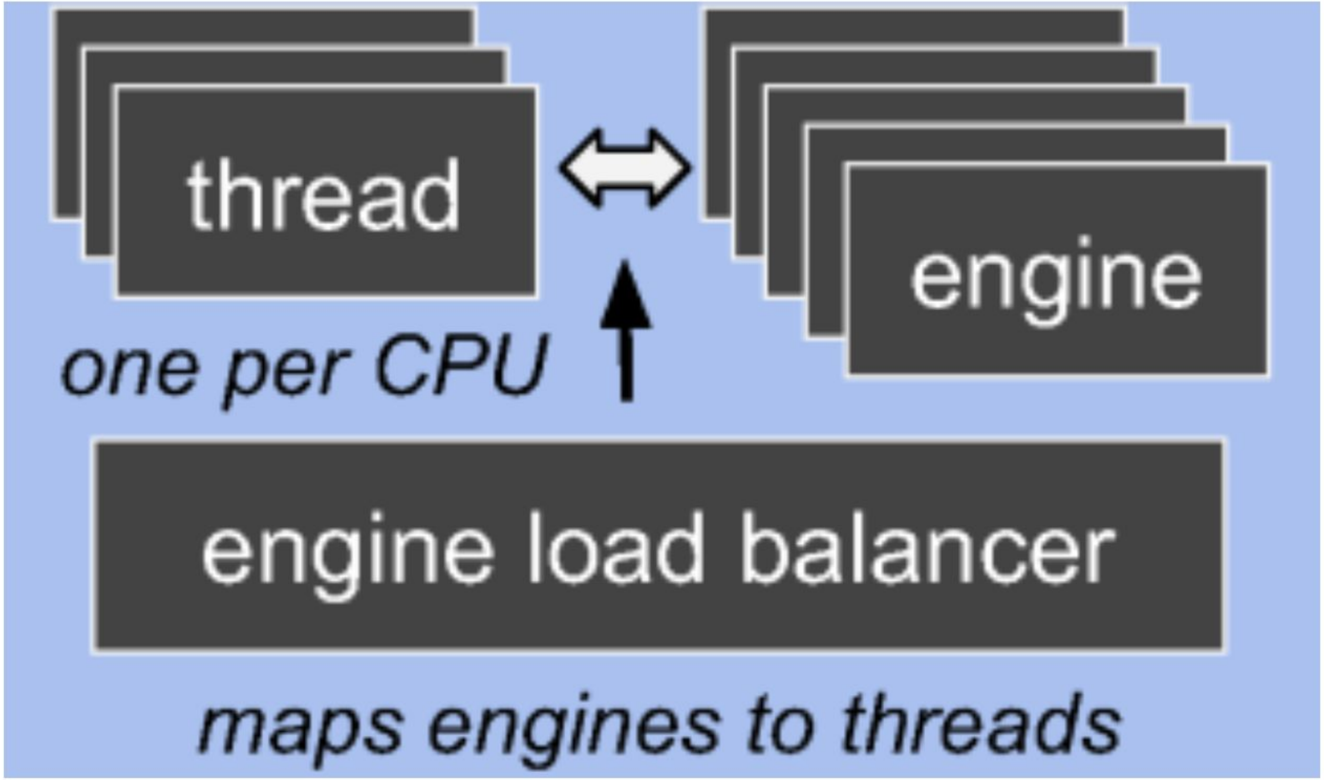
Principle synchronization:

- No blocking locks

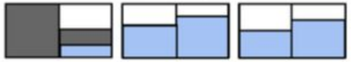
Snap Engine



Snap Engine Scheduling



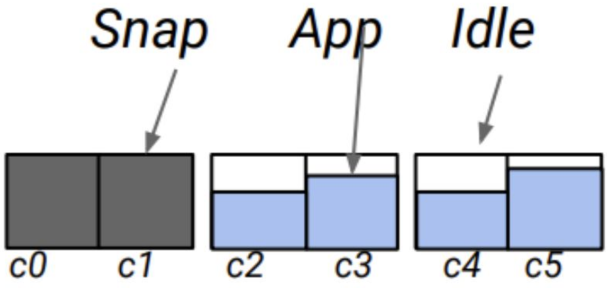
Snap Engine Scheduling Modes

scheduling mode	CPU resources	scheduling latency		CPU efficiency	visualization
		median	tail		
dedicating cores	static	0-1μs	0*-100+μs	poor	
spreading engines	dynamic	3-10μs	10-30**μs	good	
compacting engines	dynamic	0-5μs	50-100**μs	excellent	

Snap Engine Scheduling Modes

Dedicated Cores

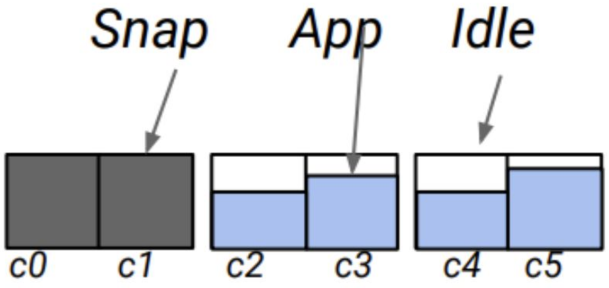
- Static provisioning of N cores to run engines.
  - Fair share these N cores across engines.
- Simple and best for some situations.



Snap Engine Scheduling Modes

Dedicated Cores

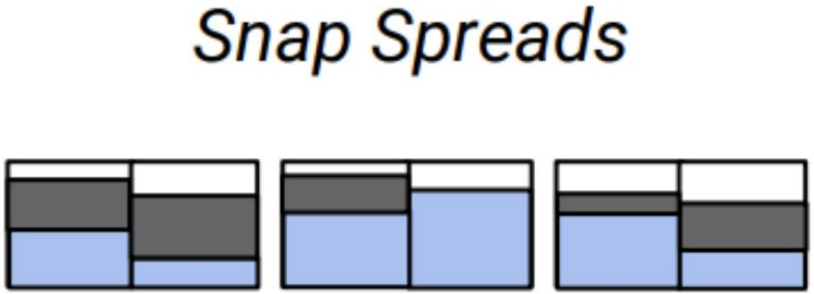
- Static provisioning of N cores to run engines.
  - Fair share these N cores across engines.
- Simple and best for some situations.
- Provisioning for the worst-case is wasteful
- Provisioning for the average case leads to high tail latency



Snap Engine Scheduling Modes

Spreading Engines

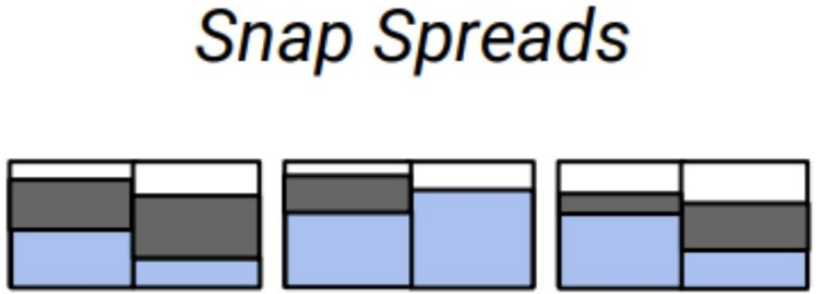
- Bind each engine to a unique thread
- Threads scheduled on-demand based on interrupts triggered from NIC or application
- Leverages new micro-quanta kernel scheduling class for tighter latency



Snap Engine Scheduling Modes

Spreading Engines

- Bind each engine to a unique thread
- Threads scheduled on-demand based on interrupts triggered from NIC or application
- Leverages new micro-quanta kernel scheduling class for tighter latency
- Can provide lowest tail latency
- Scheduling pathologies and overheads



Snap Engine Scheduling Modes

Compacting Engines

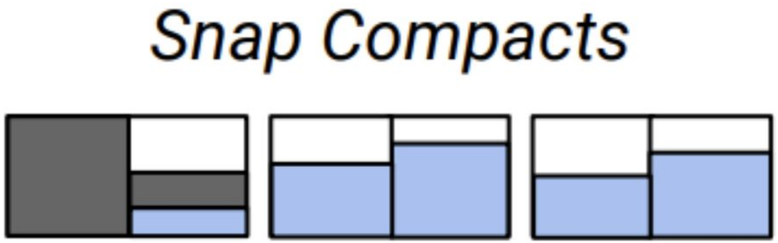
- Compacts engines to as few cores as possible
- Periodic polling of queuing delays to re-balance engines to more cores

Snap Compacts

Snap Engine Scheduling Modes

Compacting Engines

- Compacts engines to as few cores as possible
- Periodic polling of queuing delays to re-balance engines to more cores
- Can provide best CPU efficiency.
- Timely detection of queue build-up.



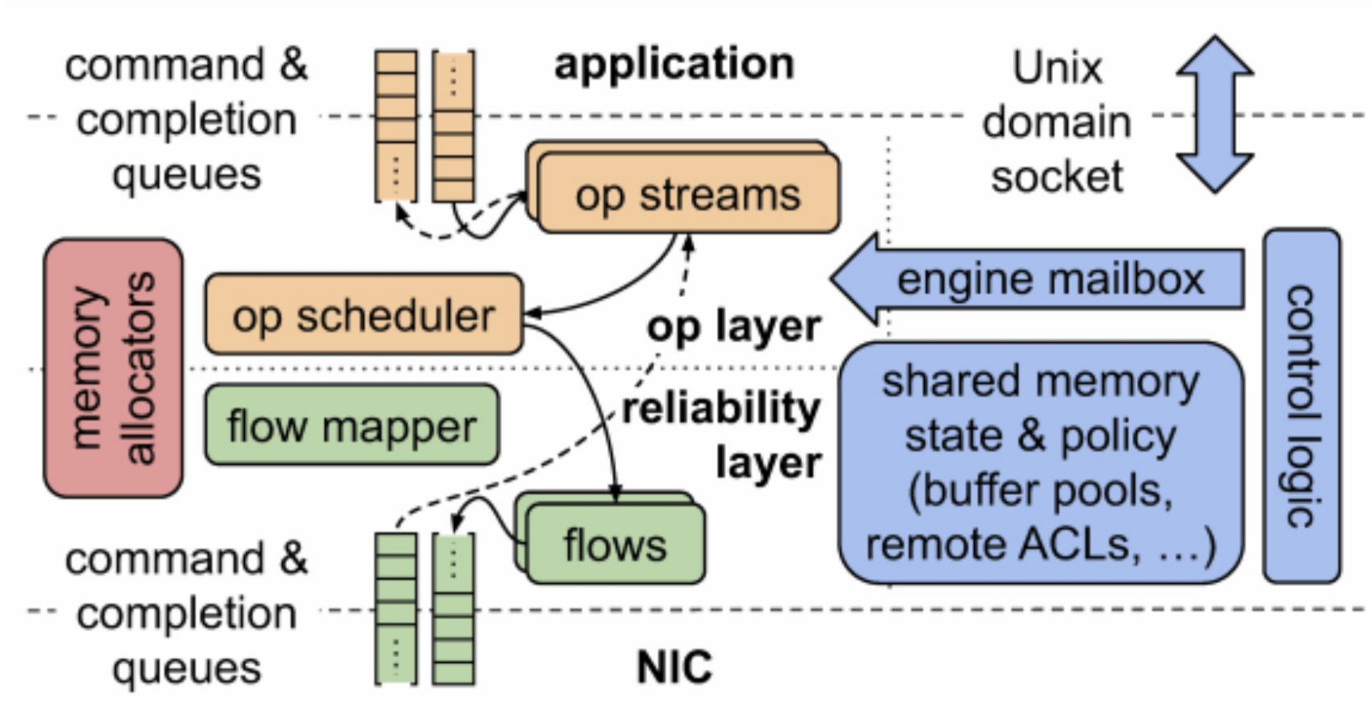
High Performance Communication

Pony Express Communication Stack

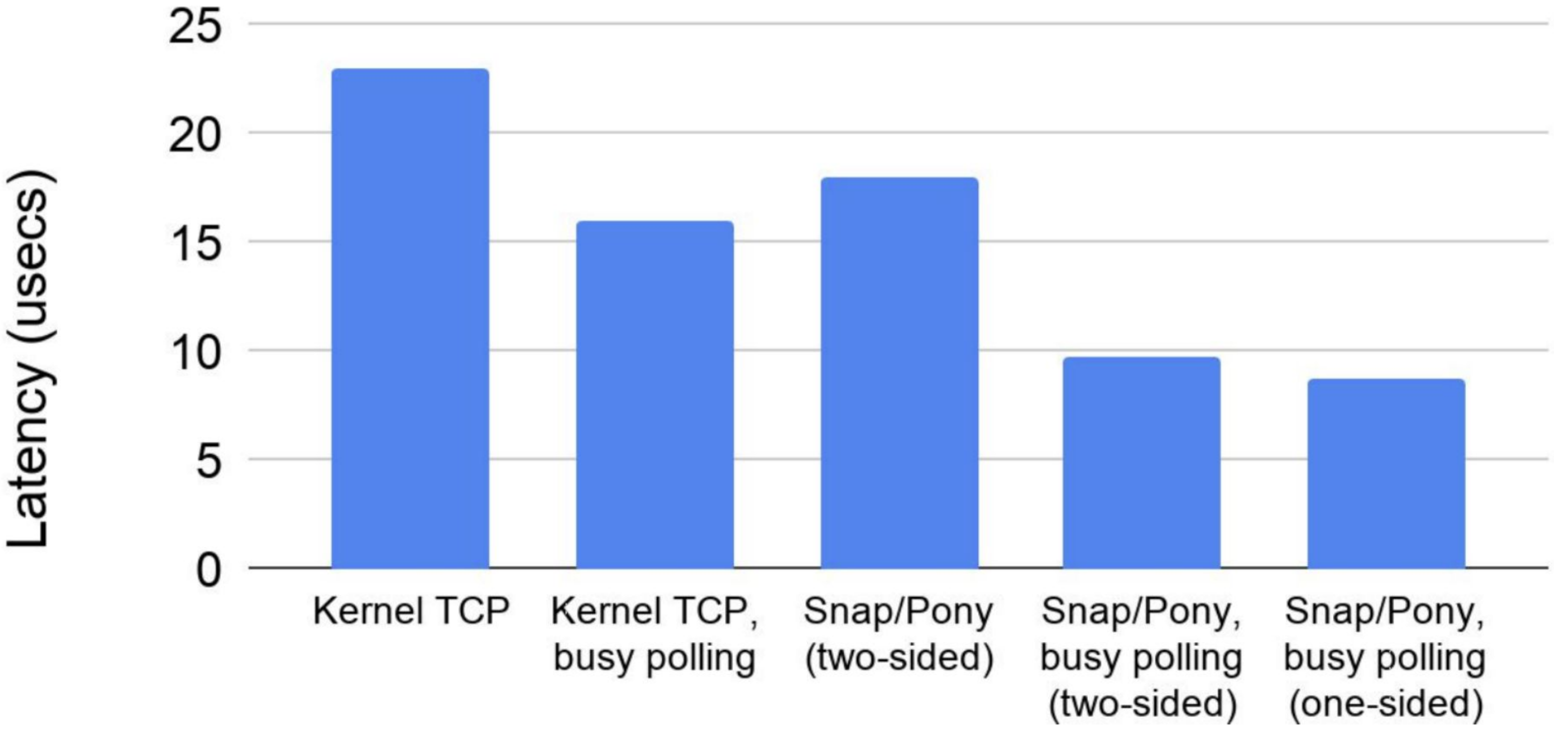
- Implement a full-fledged reliable transport and interface
  - RDMA-like operation interface to applications
  - Two-sided operations for classic RPC
  - One-sided (pseudo RDMA) operations for avoiding invocation of application thread scheduler
  - Custom one-sided operations to avoid shortcomings of RDMA (i.e., pointer chase over fabric)
  - Custom transport and delay-based congestion control (Timely/Swift)

High Performance Communication

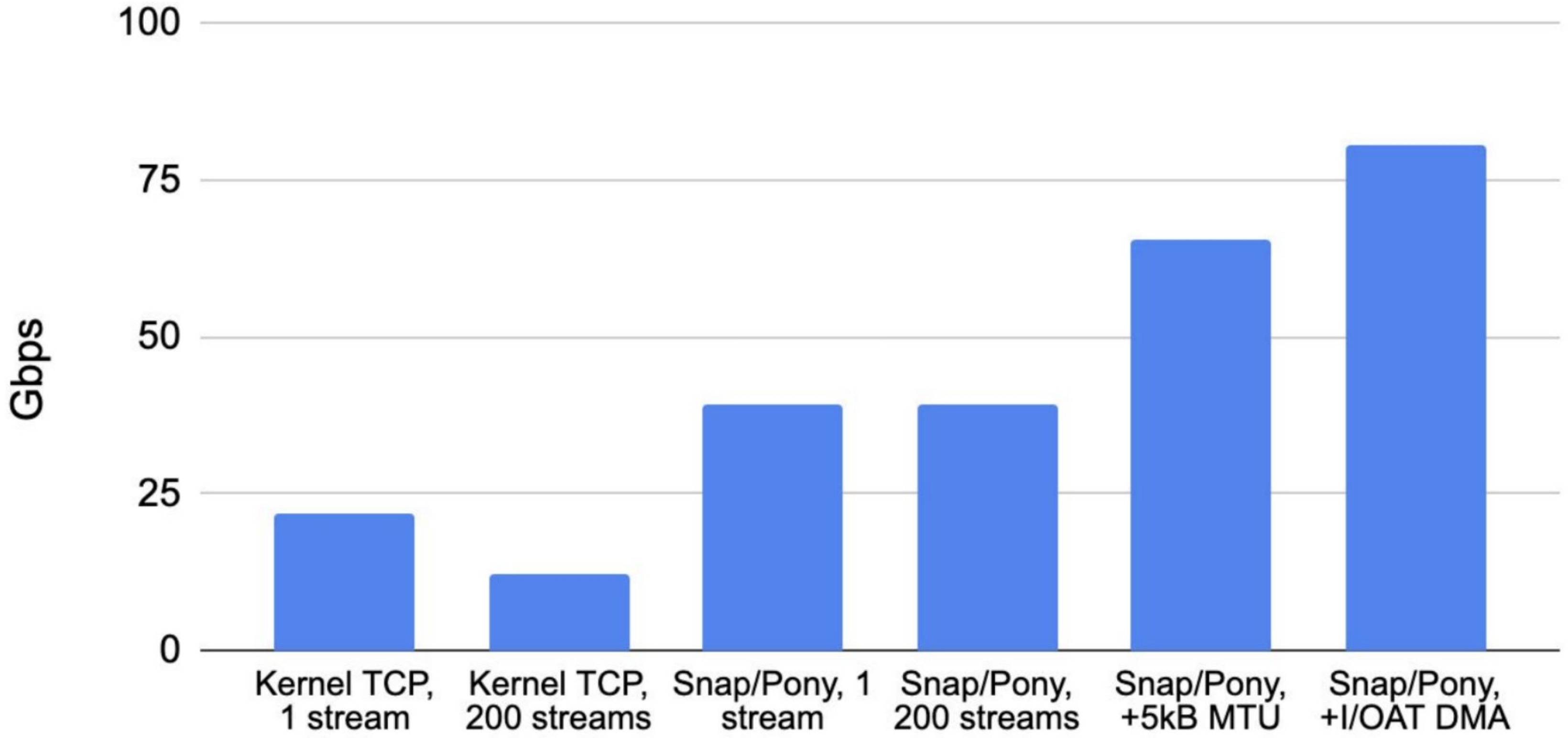
Pony Express Communication Stack



Evaluation: Ping-pong latency



Evaluation: Throughput



Evaluation: Comparison with RDMA

- Switching to Pony Express “doubled the production performance of the data analytics service”.
- Stringent RDMA rate limits applied to prevent NIC cache overflow, and ensuing PFCs.
- Could be disabled with Pony Express.

Thank you!

Q & A