Host Networking (Google Case Study)

ECE/CS598HPN

Radhika Mittal

Snap: a Microkernel Approach to Host Networking

SOSP'19

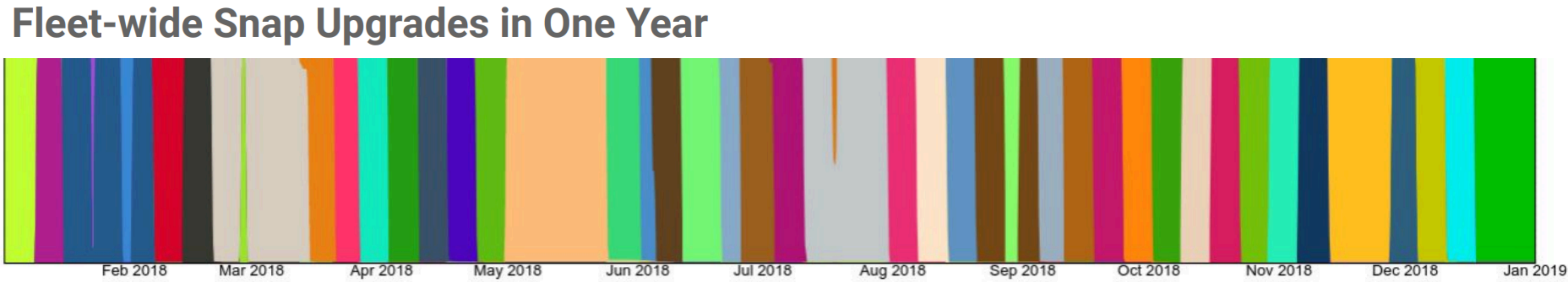
Slides largely borrowed from the SOSP talk

Summary

- 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), 5M IOPS, and weekly releases

Motivation

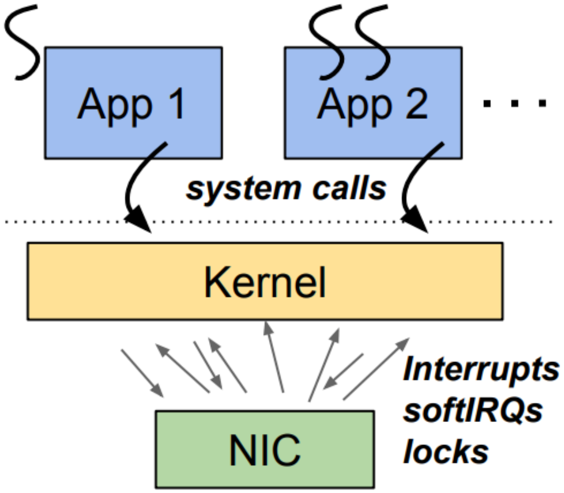
- Growing performance-demanding packet processing needs at Google
- The ability to rapidly **develop and deploy** new features is just as important!



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



Performance:

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

LibraryOS and OS Bypass

Networking logic in application binaries

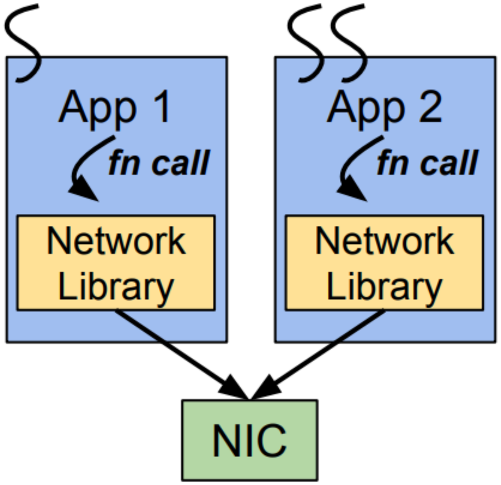
Examples: Arrakis, mTCP, IX, ZygOS, and more

Deployment Velocity:

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

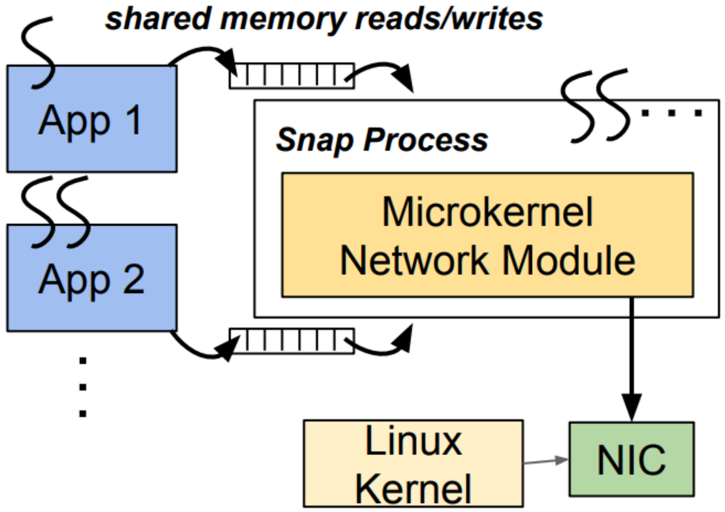
Performance:

- 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



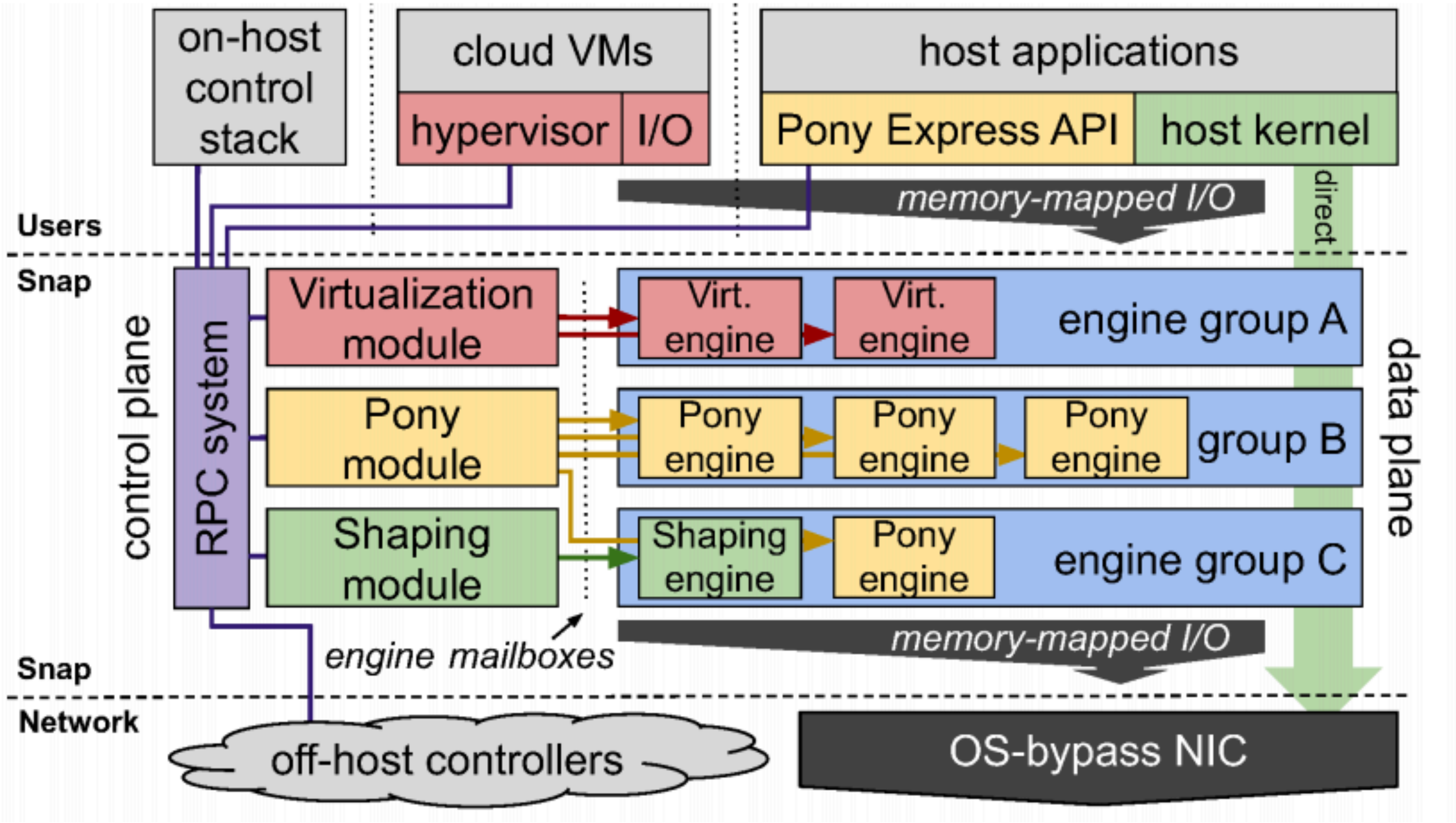
Deployment Velocity:

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

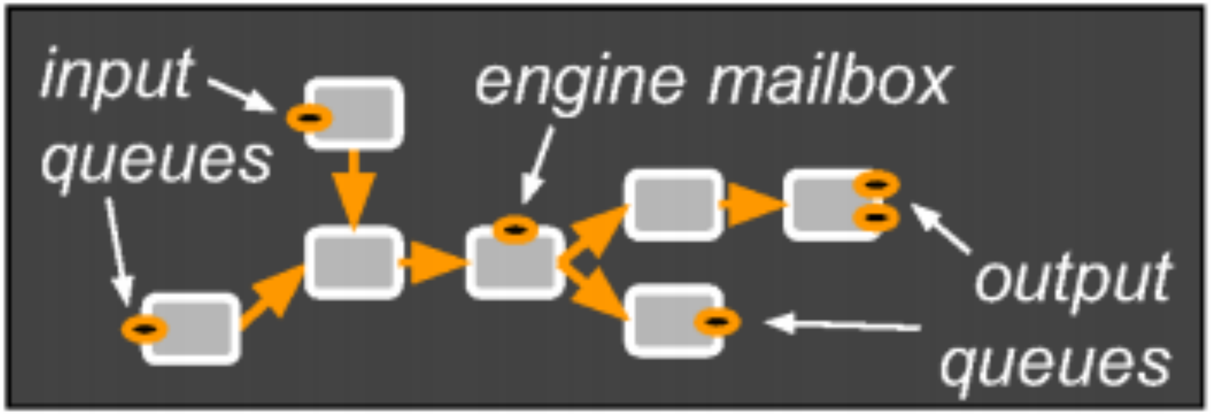
Performance:

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

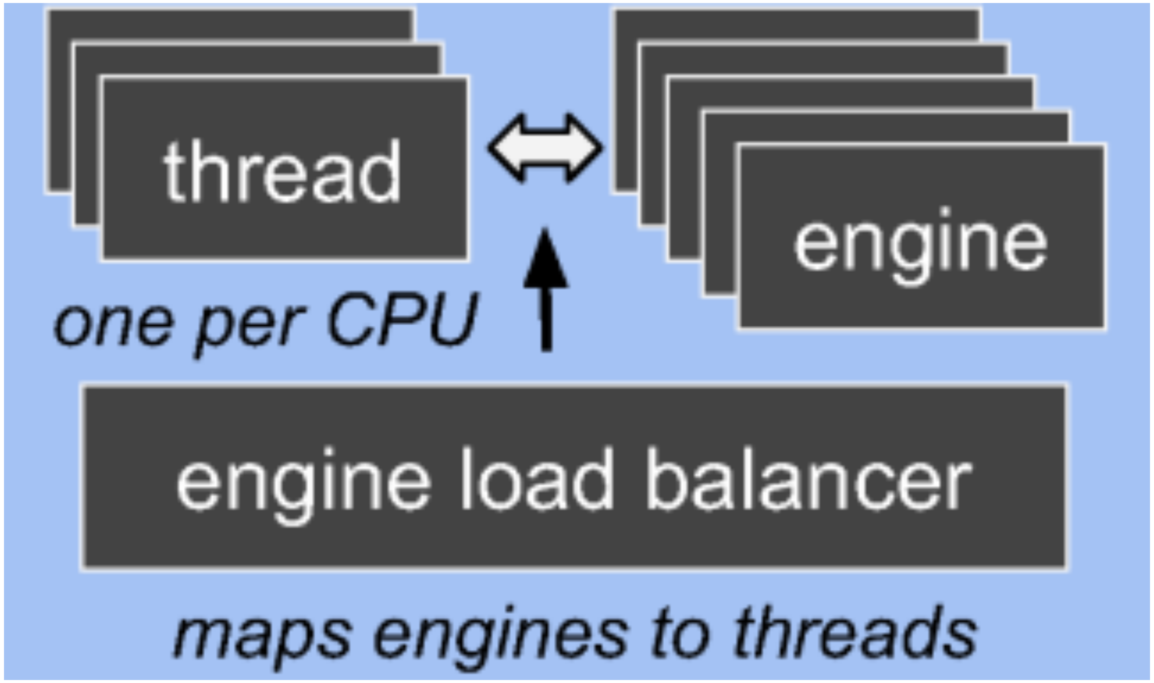
Snap Architecture



Snap Engine



Snap Engine Scheduling



thread

engine

one per CPU

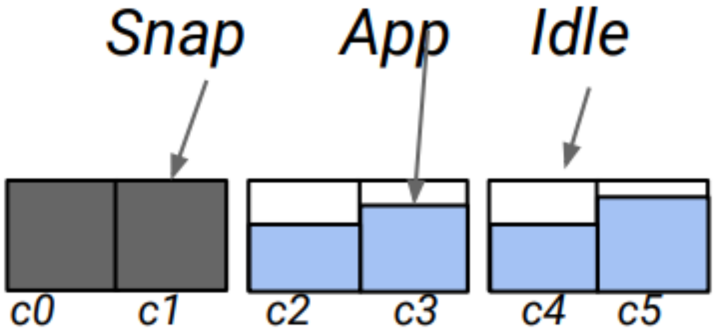
engine load balancer

maps engines to threads

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

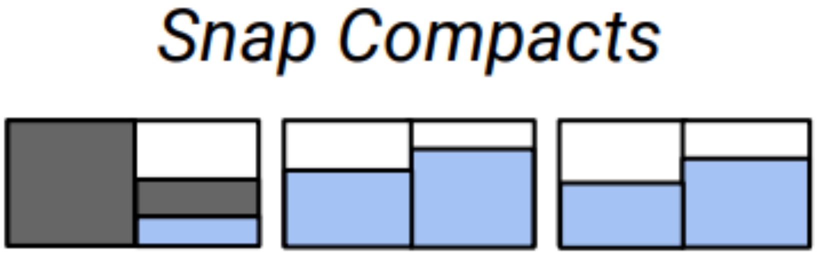
- Bind each engine to a unique thread
- Threads scheduled on-demand based on interrupts triggered from NIC or application
- Leverages new micro-quanta kernel scheduling class for tighter latency
- *Can* provide lowest tail latency
- Scheduling pathologies and overheads

Snap Spreads

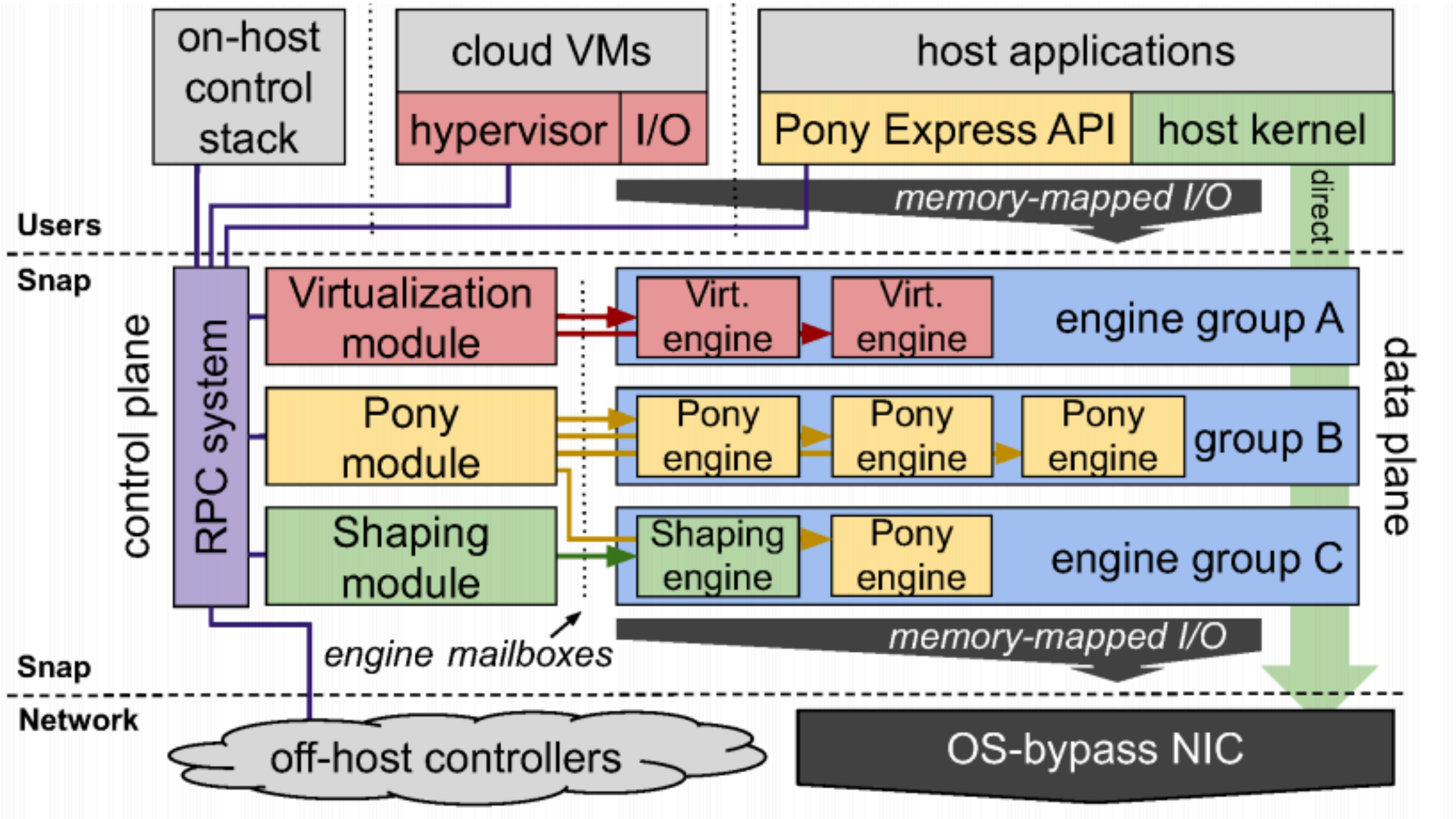
Snap Engine Scheduling Modes

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.



Snap Architecture



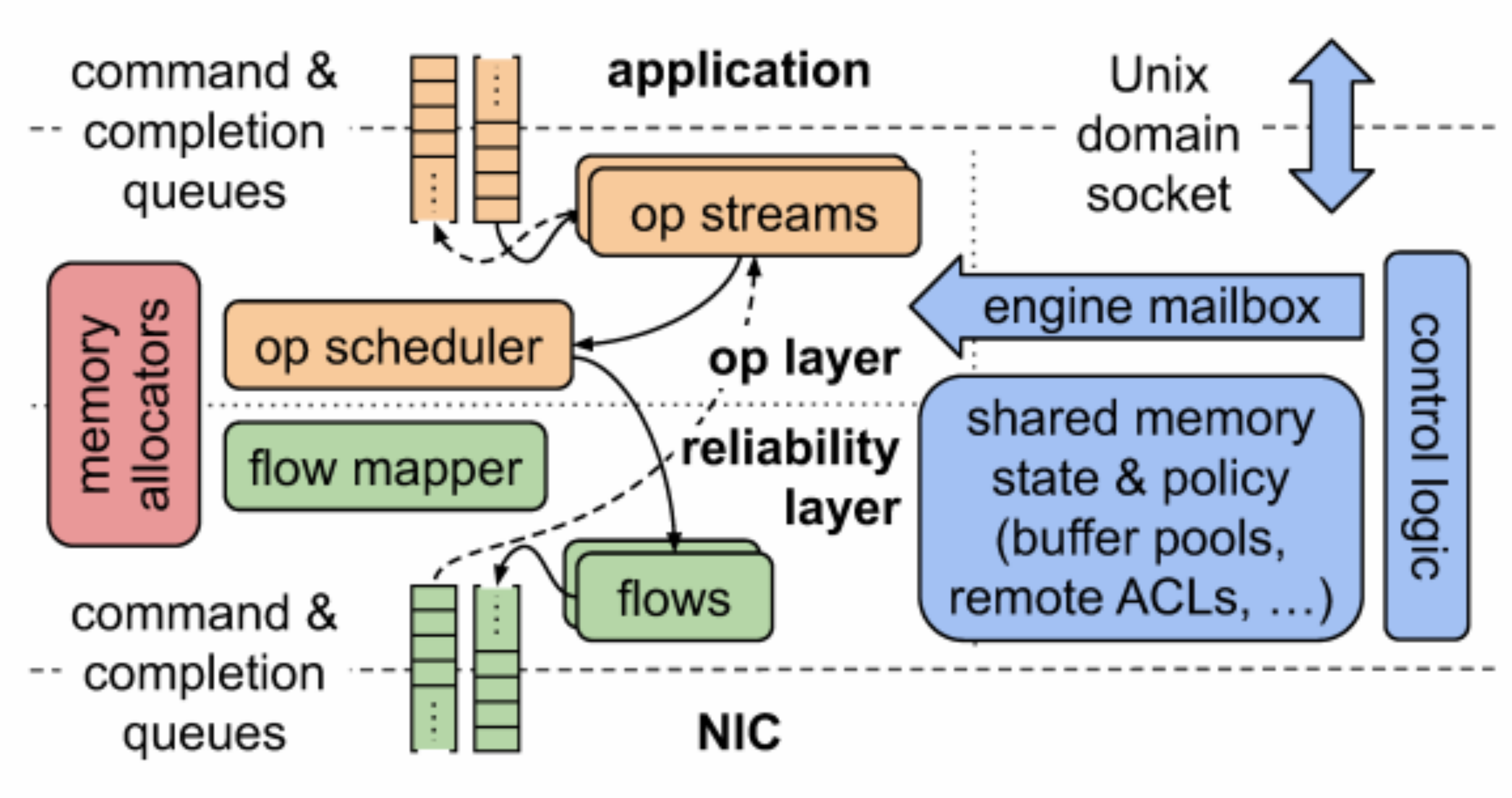
High Performance Communication

Pony Express Communication Stack

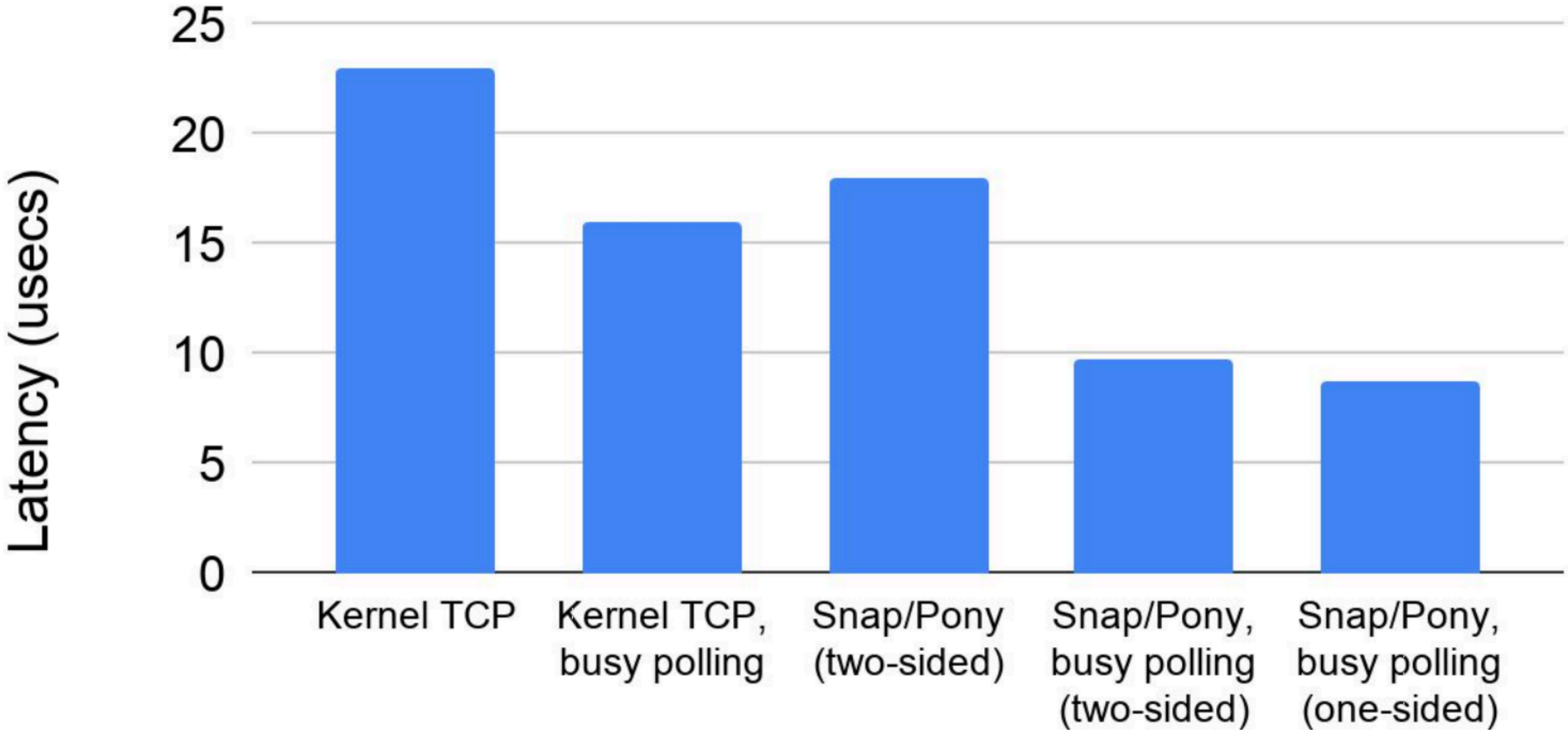
- Implement a full-fledged reliable transport and interface
  - RDMA-like operation interface to applications
  - Two-sided operations for classic RPC
  - One-sided (pseudo RDMA) operations for avoiding invocation of application thread scheduler
  - Custom one-sided operations to avoid shortcomings of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

High Performance Communication

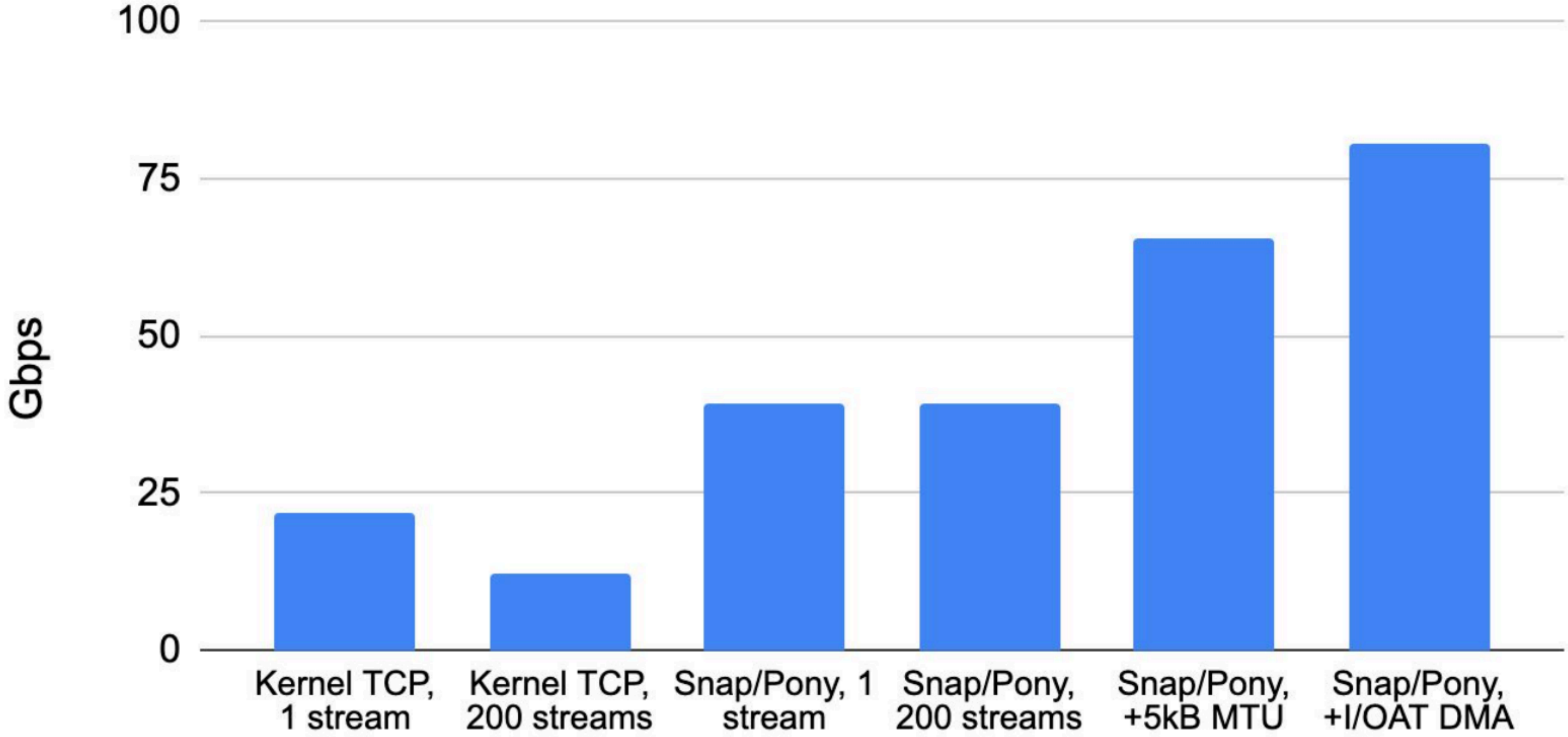
Pony Express Communication Stack



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.

Your thoughts?

- What did you like about the work?
- What are its limitations?
- What are some alternative design choices?

Logistics

- Second progress report due today!
- Please identify the key delta from first report
  - either tag new/heavily-edited sections and paragraphs
  - or include a paragraph at the end that describes the key changes.