Host Networking (Google Case Study)

ECE/CS598HPN

Radhika Mittal

Snap: a Microkernel Approach to Host Networking

SOSP'19

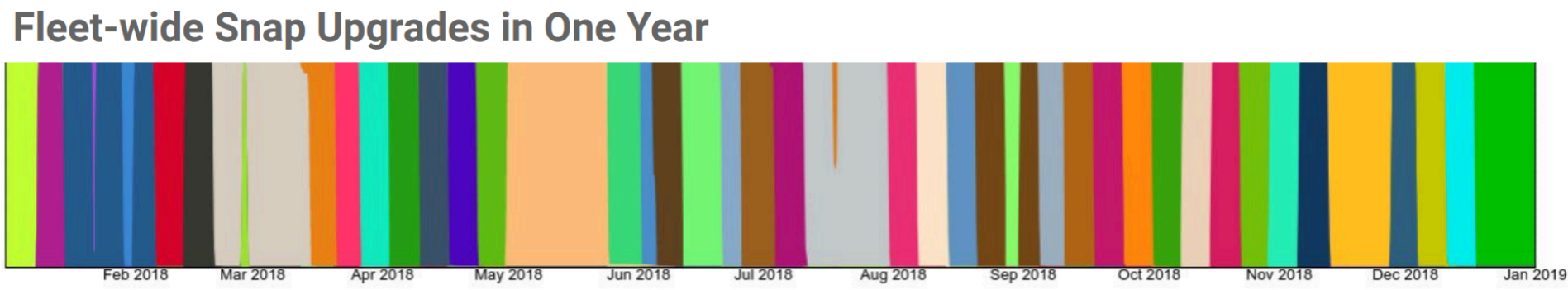
Slides largely borrowed from the SOSP talk

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!



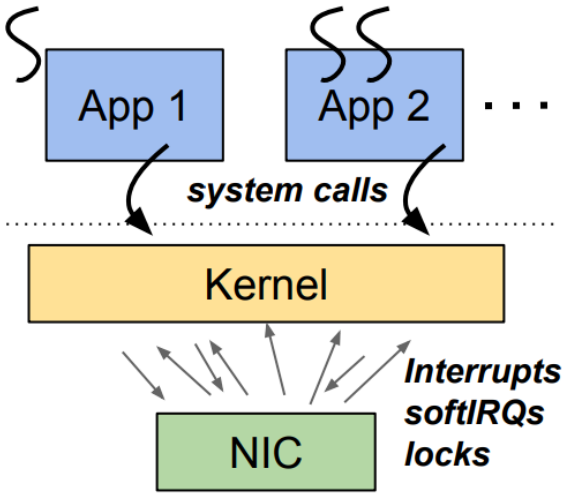
Monolithic (Linux) Kernel

Performance:

- Overheads from system calls, fine-grained synchronization, interrupts, etc.

Deployment Velocity:

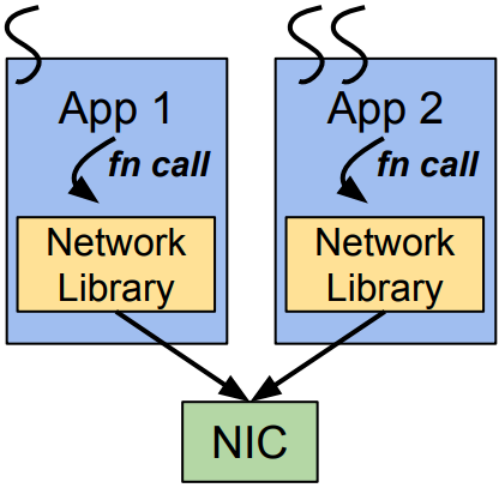
- Smaller pool of software developers
- More challenging development environment
- Must drain and reboot a machine to roll out new version
- Typically months to release new feature



LibraryOS and OS Bypass

Networking logic in application binaries

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



Performance:

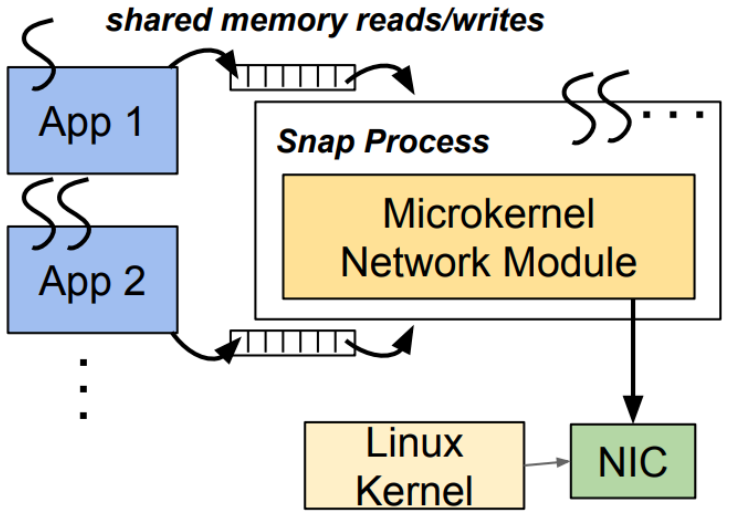
- Can be very fast
- But typically requires spin-polling in every application
- Benefits of centralization (i.e., scheduling) lost
  - Delegates all policy to NIC

Deployment Velocity:

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

Microkernel Approach

Hoists functionality to a separate userspace process



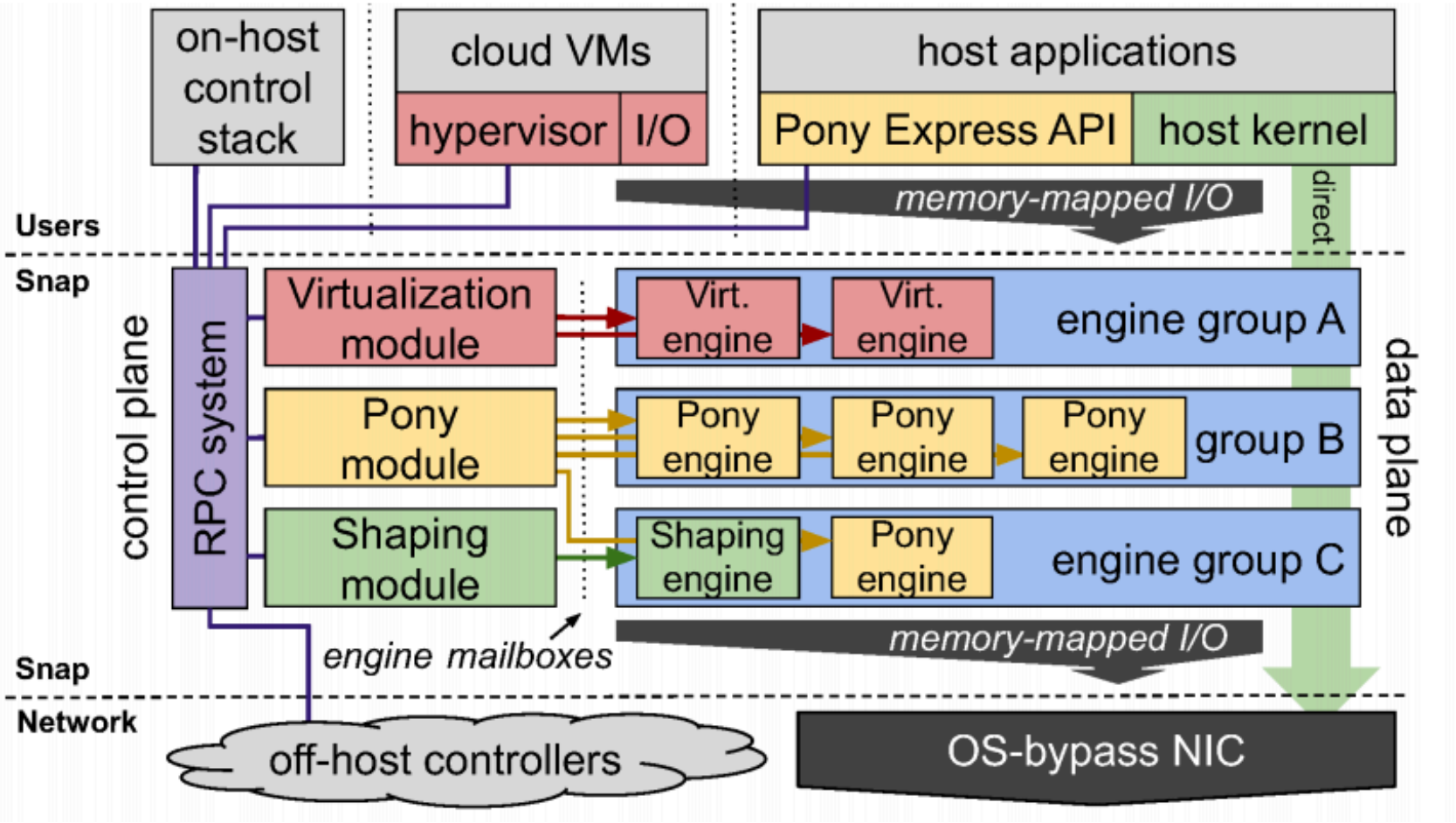
Performance:

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

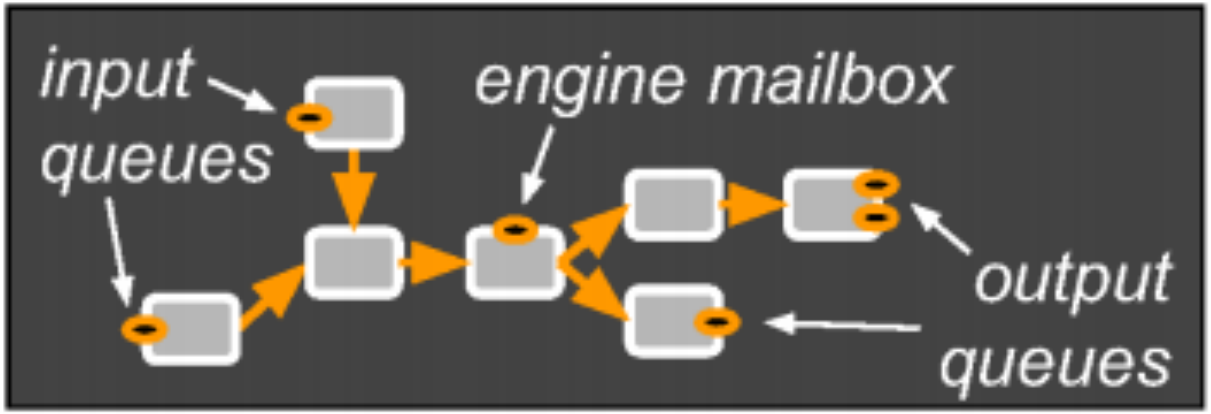
Deployment Velocity:

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

Snap Architecture



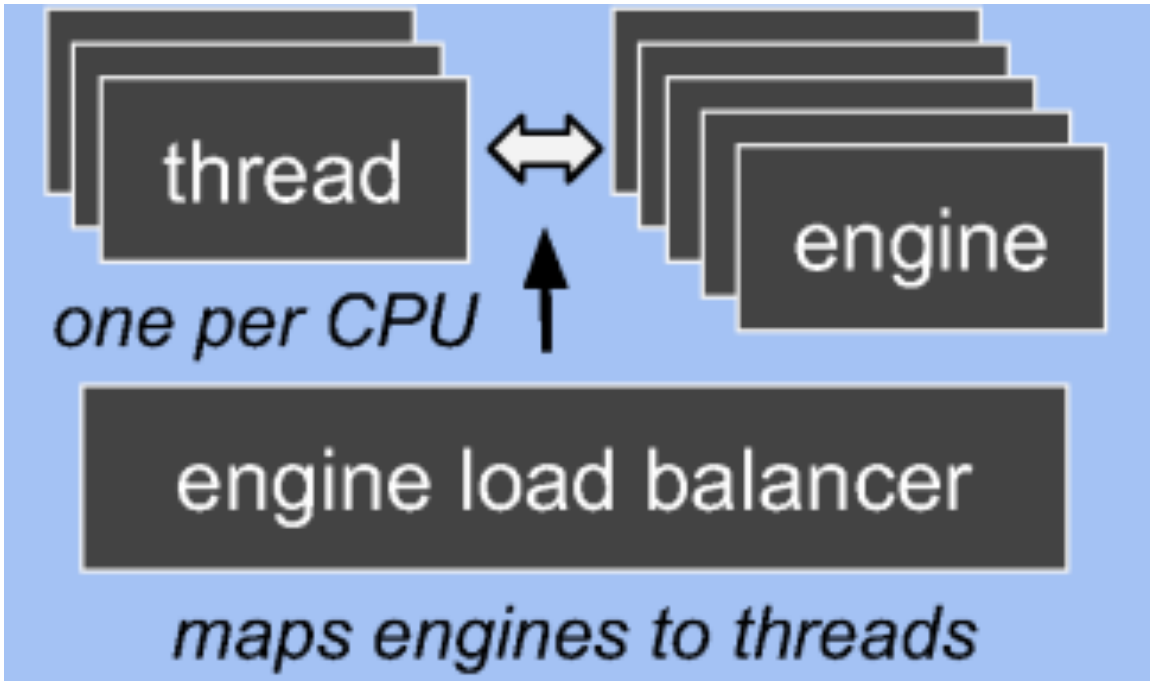
Snap Engine



- Dataplane communication:
  - Memory-mapped I/O for payload / packets.
  - Non-blocking asynchronous I/O through input and output queues for ops.

- Control plane communication: "engine mailboxes" – queue of depth one. also non-blocking with respect to the engine.

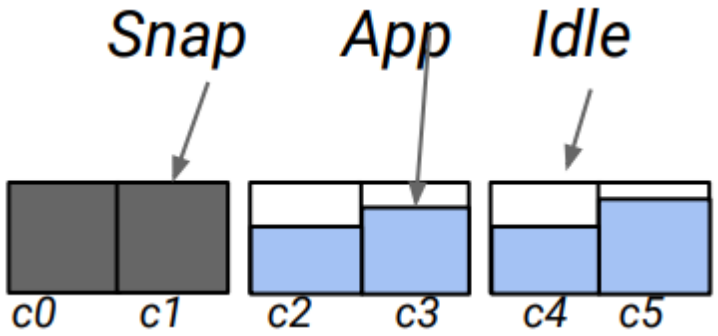
Snap Engine Scheduling



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

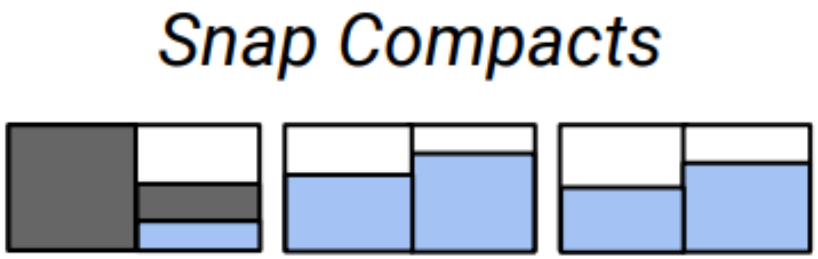
Snap Spreads



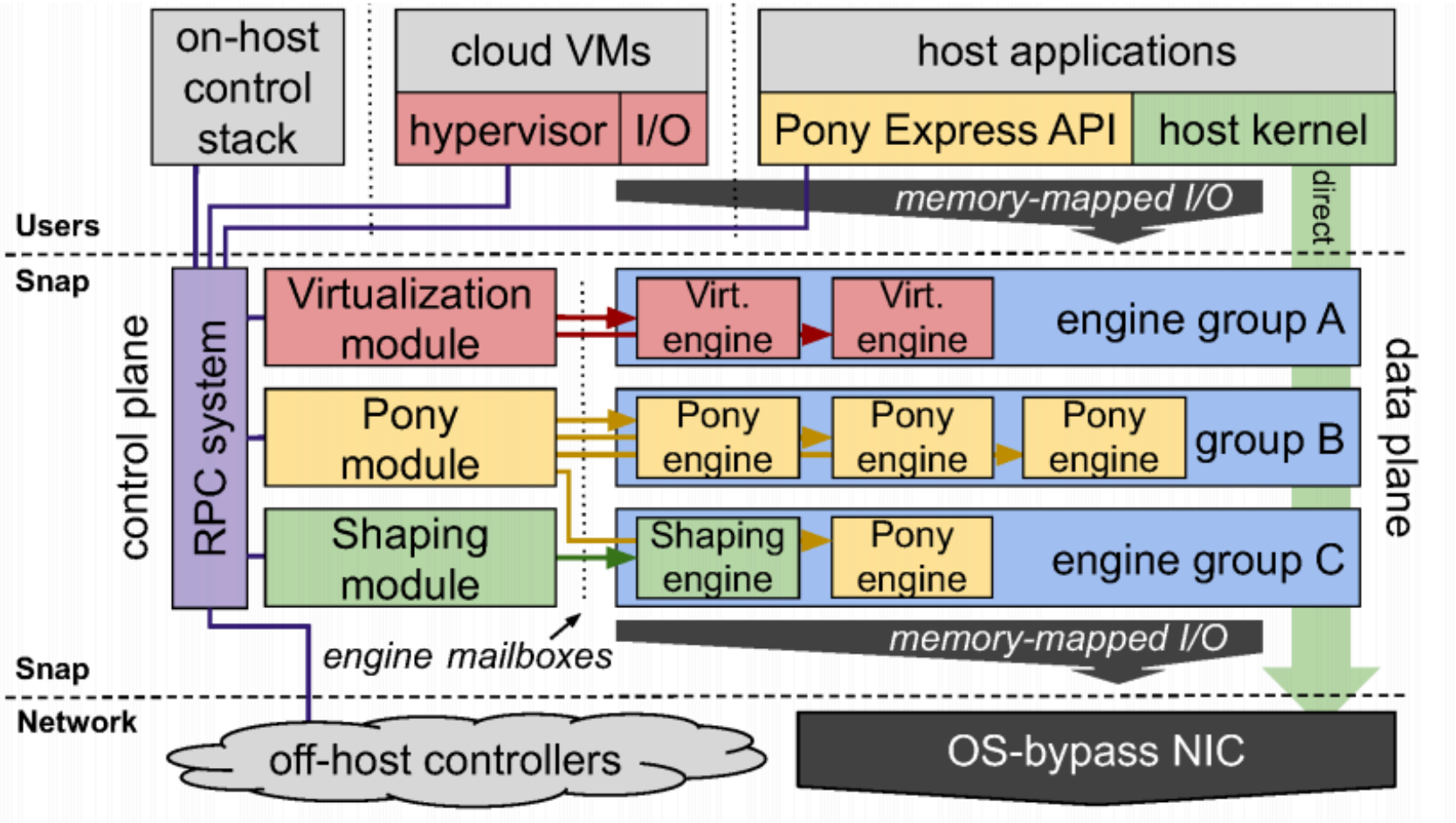
Snap Engine Scheduling Modes

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

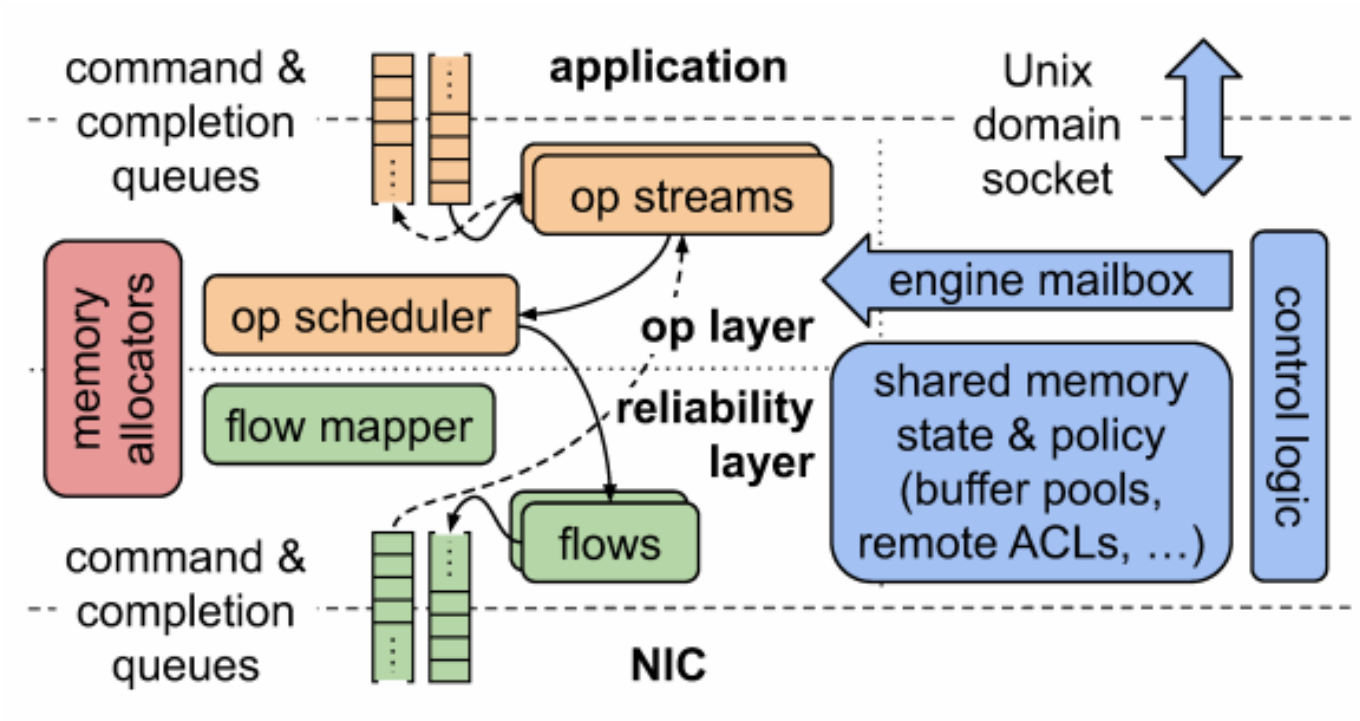


High Performance Communication

Pony Express Communication Stack

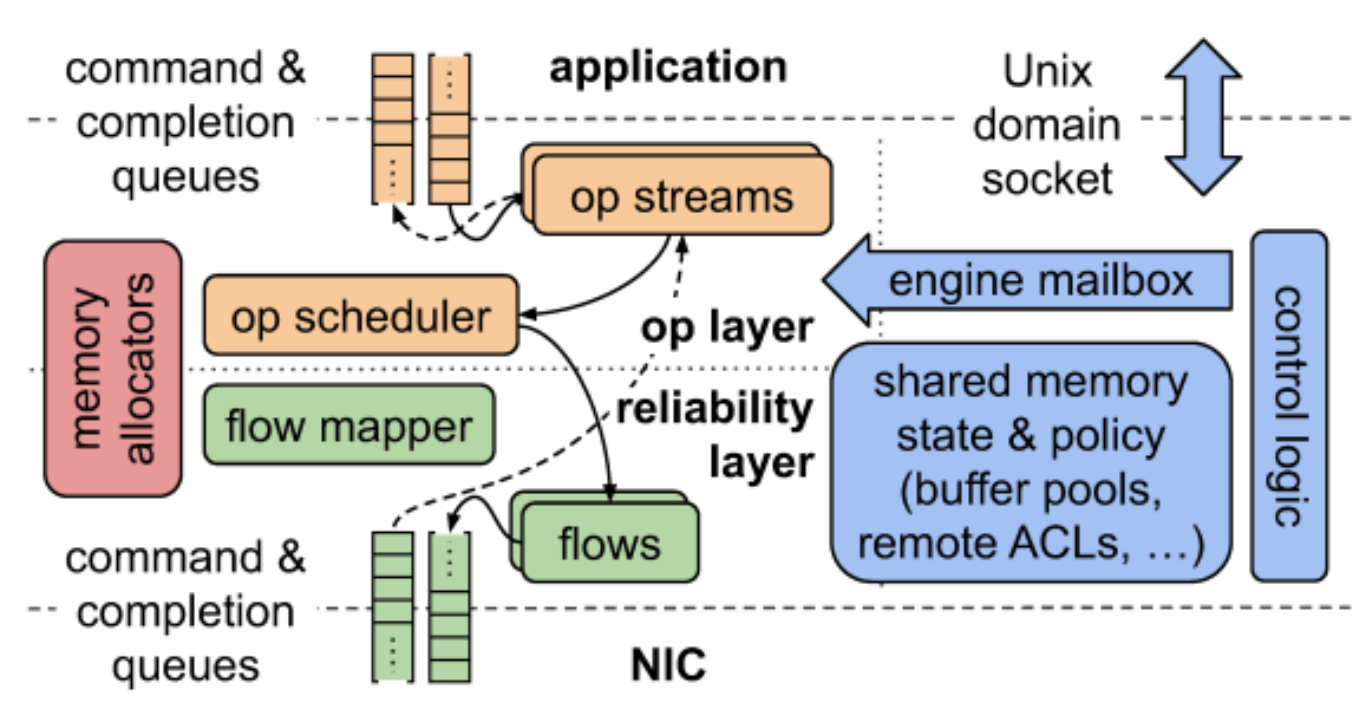
- Implement a full-fledged reliable transport and interface
  - RDMA-like operation interface to applications
  - Two-sided operations for classic RPC
  - One-sided (pseudo RDMA) operations for avoiding invocation of application thread scheduler
  - Software flexibility enables custom one-sided operations (e.g. indirect reads, scan-and-read).
  - Custom transport and delay-based congestion control (Timely/Swift)

Pony Express Communication Stack



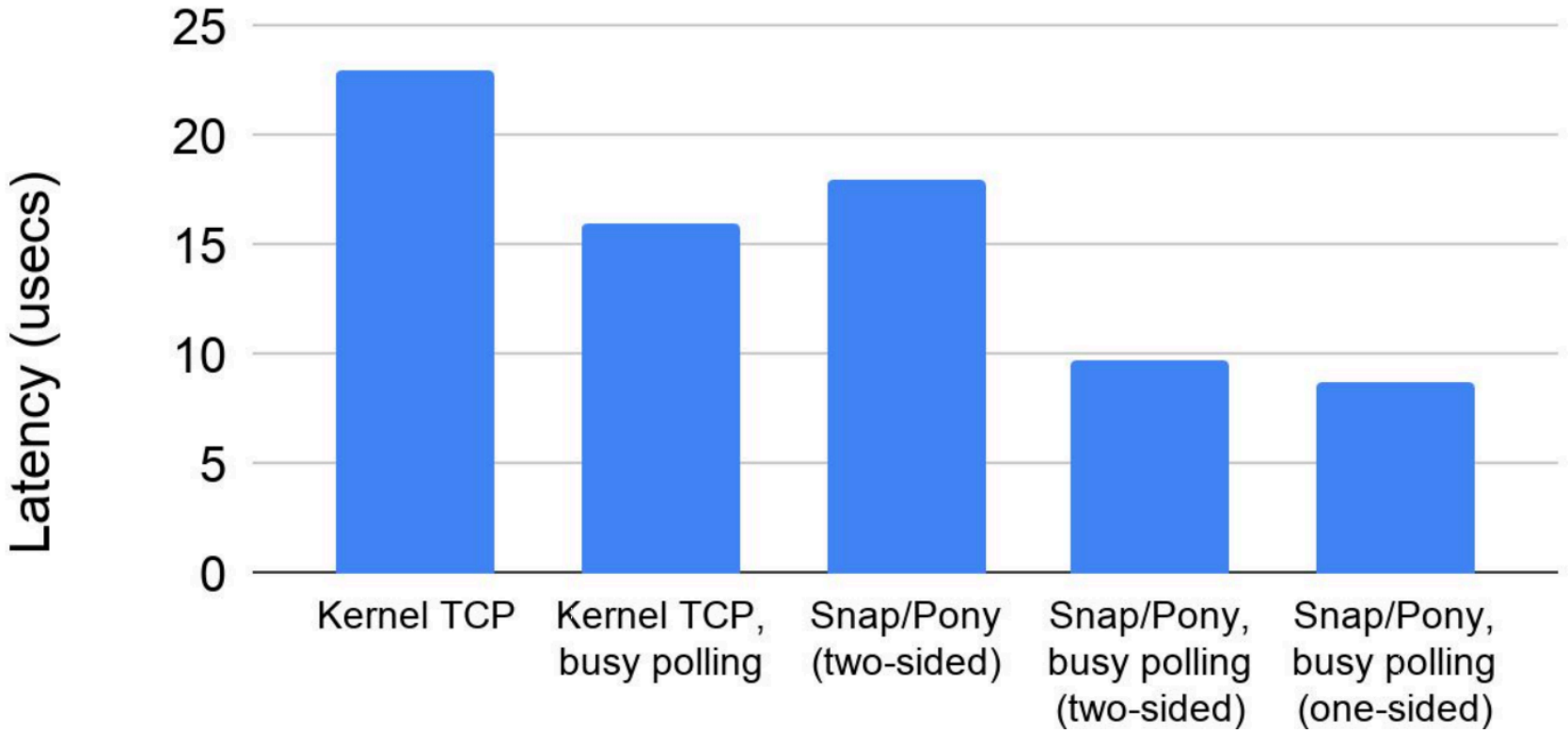
- control logic in blue.
- application threads run on a different core than Pony Express.
- apps can either have exclusive or shared Pony Express engines.

Pony Express Communication Stack

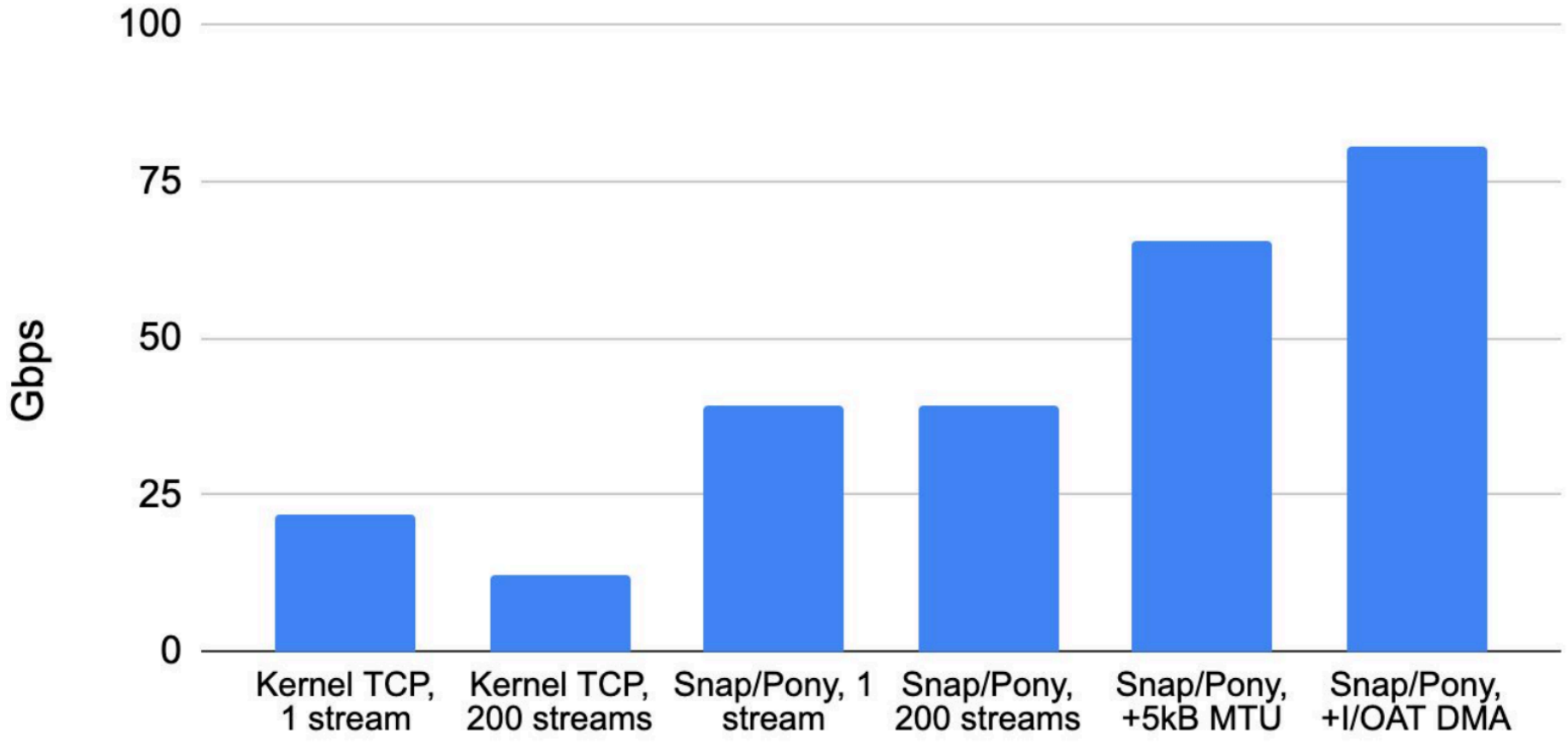


- transport operates in two layers: app-level ops and flow-level reliability and congestion-control.
- handle packets from NIC and commands from app in batches.
- just-in-time creation of packets based on NIX tx slot availability.

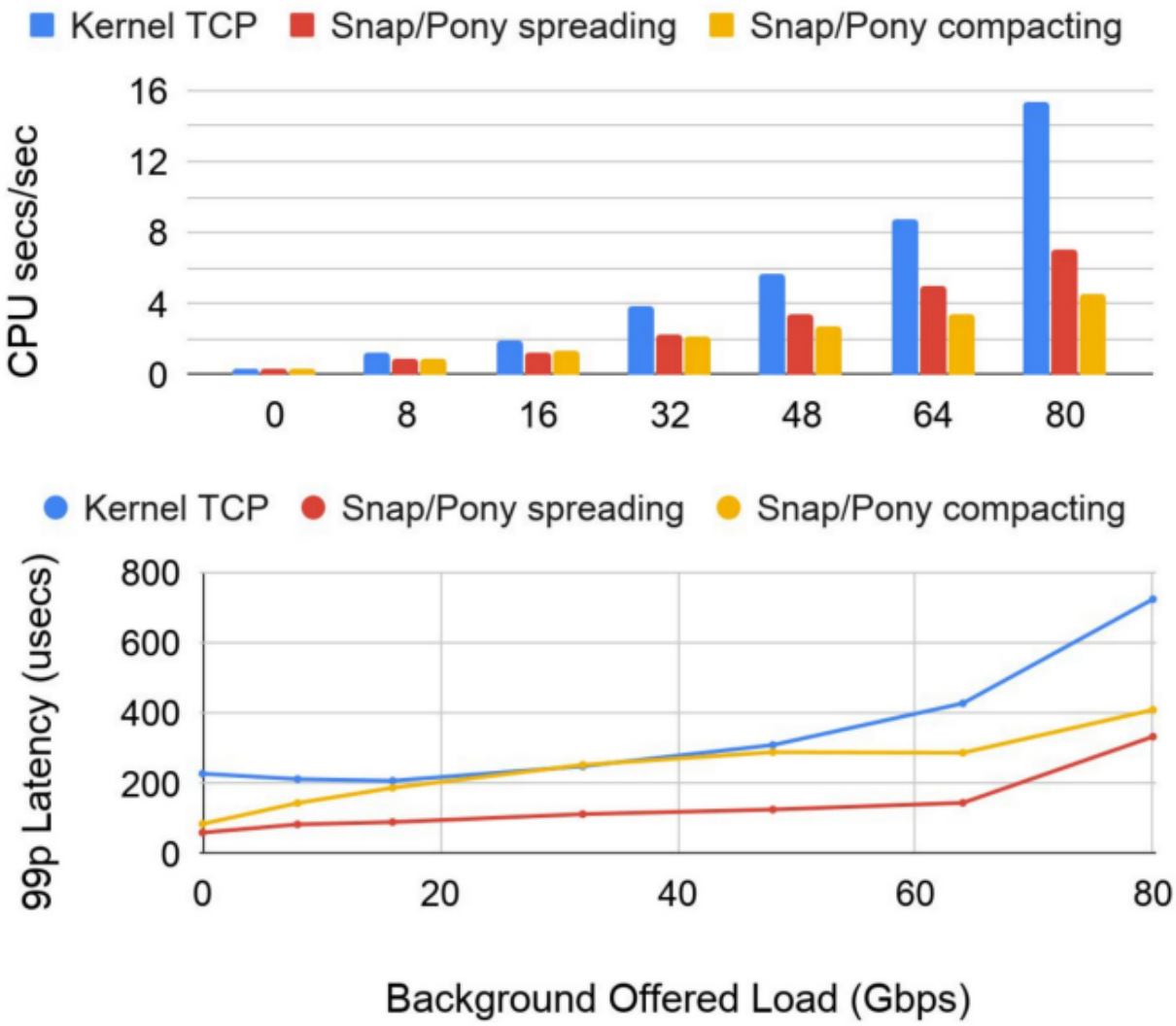
Evaluation: Ping-pong latency



Evaluation: Throughput



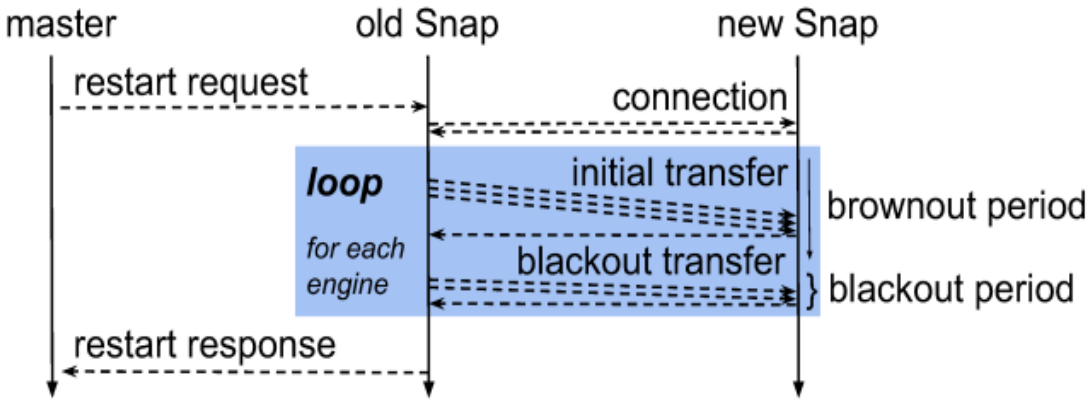
Evaluation: CPU efficiency and tail latency

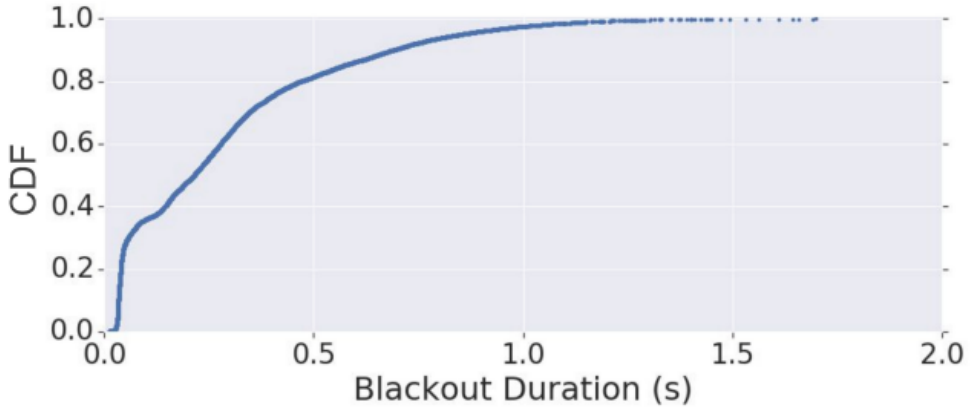


Evaluation: Comparison with RDMA

- 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.

Transparent Updates





Your thoughts?

- What did you like about the work?
- What are its limitations?

Logistics

- Second progress report due on Friday!

- 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.

Logistics

- Assigned reading for the class on Nov 20th has been updated on the website.

Student Presentations

- Classes on Dec 2 and Dec 4 (post thanksgiving break):
  - Students will present a paper of their choice related to topics discussed in class.
    - Must be outside of course syllabus (but could be one of the optional papers). Prefer papers that weren't discussed in class.
  - **3 mins per paper.**
    - **what's the key idea and the key result.**
  - Goal is breadth and not depth.
  - Slots for 25 students on Dec 2, and 9 students on Dec 4.
  - Watch out for an email with sign-up sheet.
    - First-come-first-serve. Must list the paper title to sign up.

Other things on Dec 4

- Final project report due.

- A small (simple) quiz in class for bonus participation points.
  - No preparation needed.