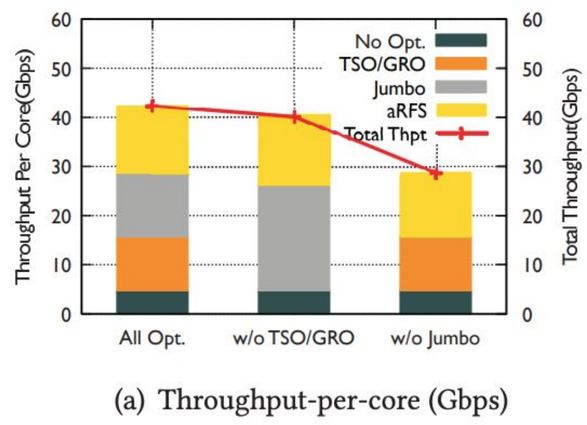
EvOS: Application-Defined OS Evolution

RS3LAB

Kumar Kartikeya Dwivedi Rishabh Iyer Sanidhya Kashyap

Specializing the OS for high-performance applications is hard

General-purpose applications	Kernel-bypass solutions	Current eBPF-based approaches
Suffer from overhead of the OS stack	Do not retain benefits of a general purpose OS	Too restrictive due to verifier limitations
IO completion models only optimize issue of IO requests	Maintenance burden, require application rewrite	Lack of generality
Pay the cost of features they never use	Support for sharing hardware is poor	A difficult programming model

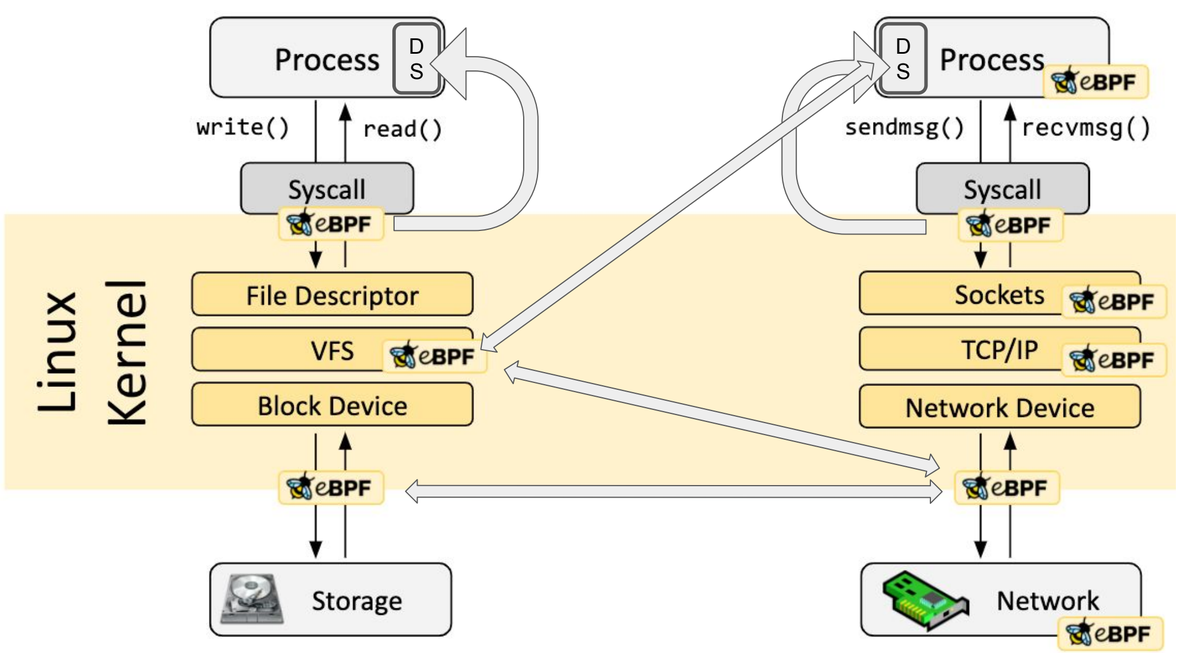


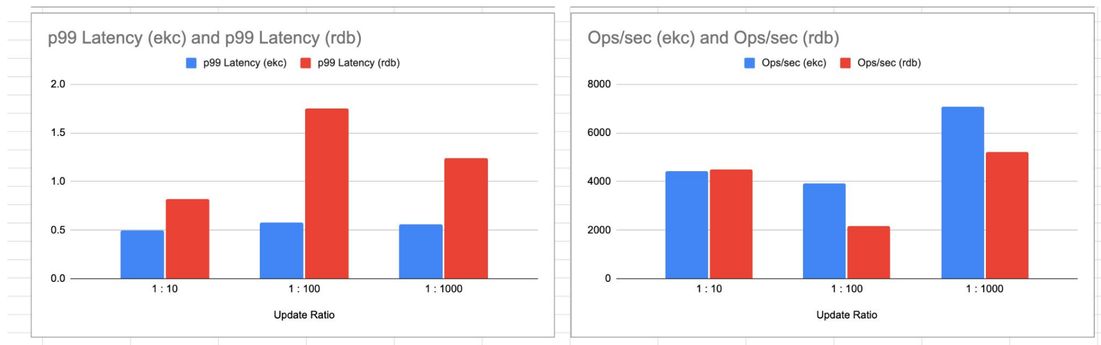
(a) Throughput-per-core (Gbps)

EvOS: Kernel-userspace co-design for high performance applications

Programmability	Safety	Data structure access/design	Use the Linux kernel
Express IO requests using a task abstraction	Assertions to simplify program writing	Read/write application data structures	Lower maintenance
Integrate user space task abstractions with eBPF	Bounded userspace data structure traversal	Transparent access to user space memory	Reuse existing OS stack, rewrite only when needed
BPF Coroutines	eBPF threads with a relaxed safety model	Control over task scheduling	Accelerate existing applications

Overview





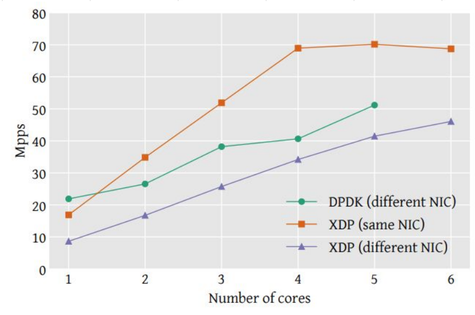


Figure 5: Packet forwarding throughput. Sending and receiving on the same interface takes up more bandwidth on the same PCI port, which means we hit the PCI bus limit at 70 Mpps.

Research questions

- How do we formally reason about the safety of extensions we add to the eBPF runtime in the Linux kernel?
- How to exhaustively identify beforehand whether an application can benefit from such a framework?
- How do we enable synchronization and concurrency control between the eBPF program and the userspace application?

Potential solutions

- Precisely define the safety properties of the verifier, and sketch a proof of correctness about the impact of extensions on the verification model.
- We map performance metrics to bottlenecks which can be used to identify use cases that will benefit from the hybrid approach of co-design through our framework.
- We relax the safety of eBPF in certain known safe contexts, while still maintaining invariants that interacting programs rely on through verification, enabling flexibility with safety.

Applications are co-designed as a kernel+userspace hybrid for better performance with the same flexibility.