MorphOS

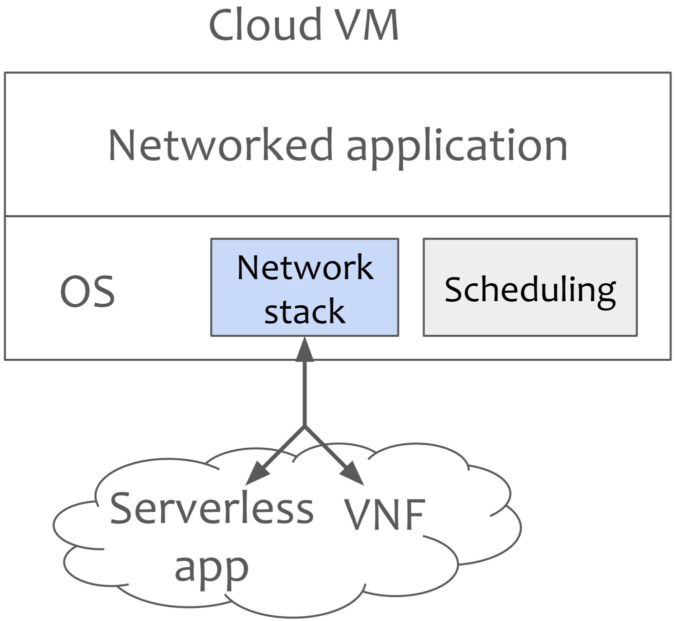
An Extensible Networked Operating System

Peter Okelmann, Ilya Meignan--Masson, Masanori Misono,
Pramod Bhatotia

CoNEXT 2025, Hong Kong

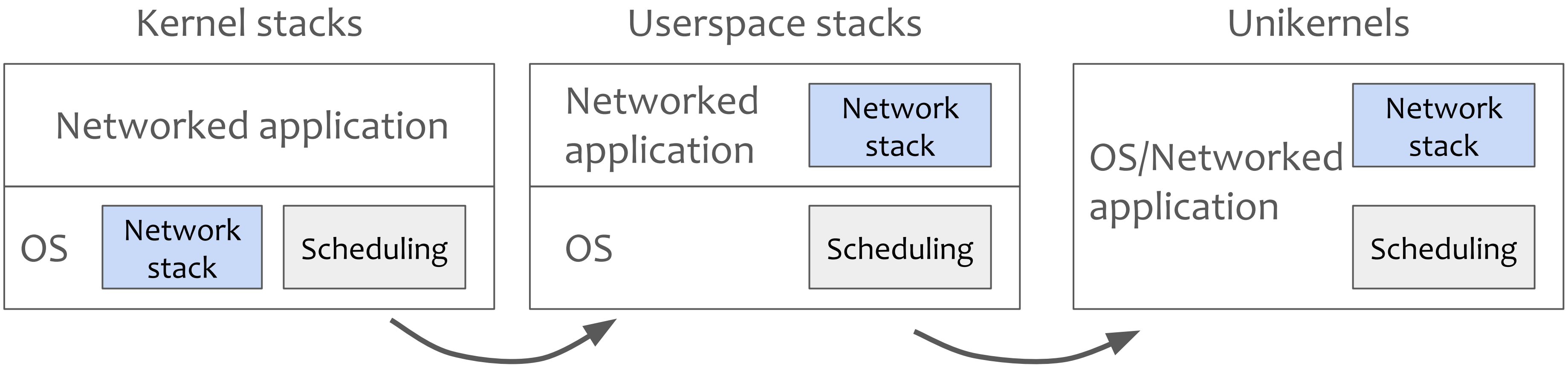
High-Performance Networking for the Cloud

- Fast, networked applications
  - Serverless apps
  - Virtual Network Functions (VNFs)
- App-OS optimizations and codesign
  - Network stack
  - Scheduling
  - VM instance chaining, …



OS design is critical for fast, networked applications

OS Architectures for Network Stacks



- Reduce context switches
- Leverage compile-time optimizations

- Extend optimization to all kernel functionality
- Fast boot times

Unikernels are ideal for cloud VMs and networked applications

Unikernel Characteristics

Pros

- High performance
- Small image size
- Minimized Trusted Computing Base (TCB)

Cons

- Lacking **extensibility**
- Lack of address space isolation
- No multi-tenant safety

Can we get unikernel benefits while mitigating the shortcomings?

Problem Statement

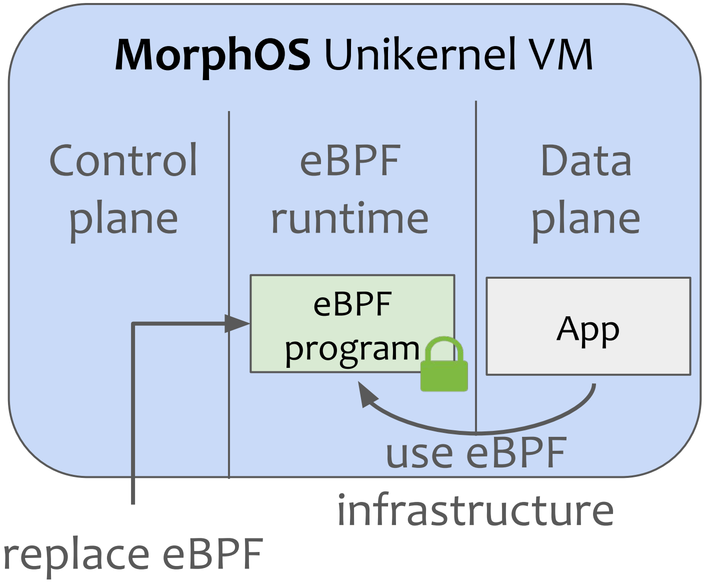
How do we achieve extensibility of fast, networked unikernel applications without compromising safety?

Design Goals

- Reconfigurability
- Safe multi-tenancy
- Performance

Our Proposal: MorphOS

- **eBPF runtime:** safe and fast extension execution
- **Control plane:** reconfigurability through eBPF updates
- **Data plane:** core application logic leveraging MorphOS hookpoints

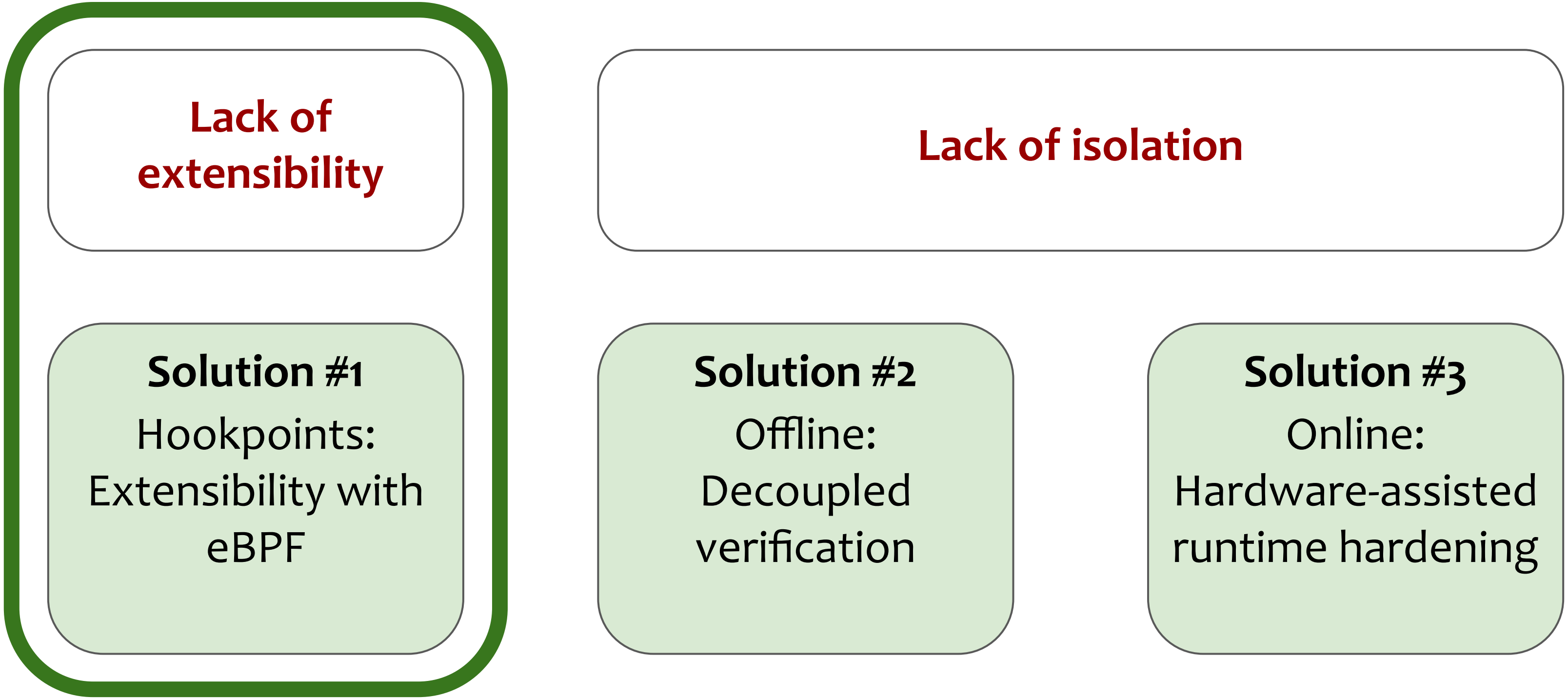


An extensible networked operating system that brings verified eBPF to unikernels

Outline

- ~~Overview~~
- Design
- Evaluation

Challenges of Unikernels



Lack of extensibility

Lack of isolation

Solution #1
Hookpoints:
Extensibility with eBPF

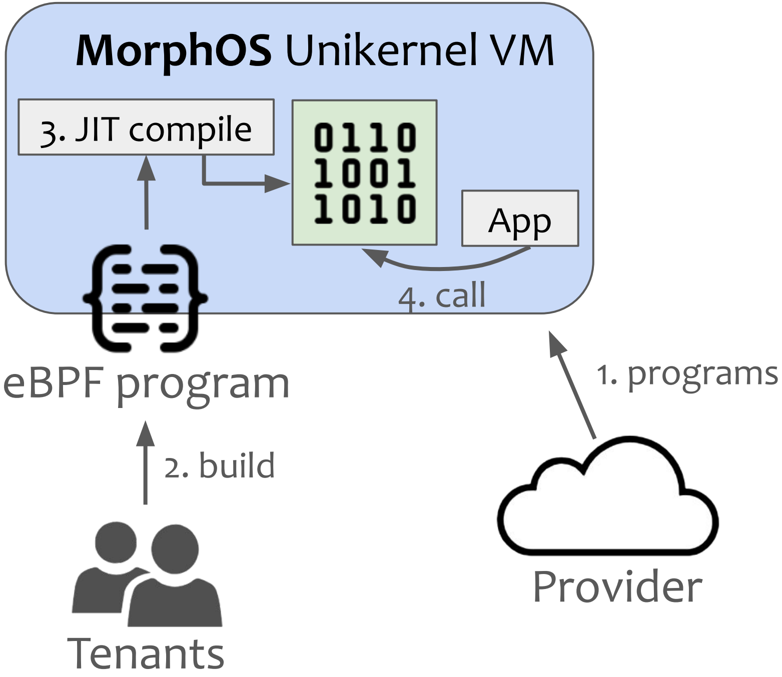
Solution #2
Offline:
Decoupled verification

Solution #3
Online:
Hardware-assisted runtime hardening

#1: Extensibility with eBPF Hookpoints

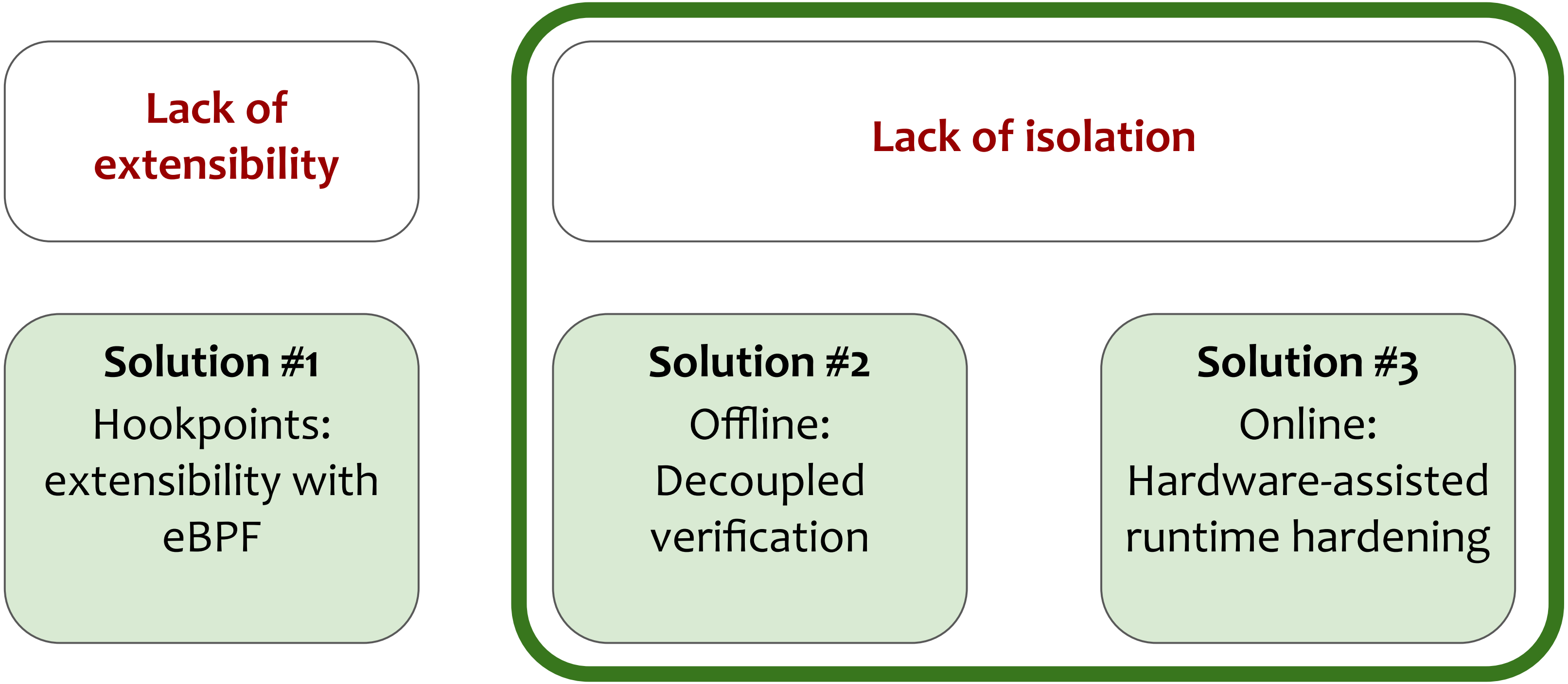
Multi-tenant VM development:

1. Provider builds optimized app
   -> **MorphOS Provider API**
2. Tenant build eBPF program
   -> **MorphOS Tenant API**
3. JIT compilation
4. Runtime execution



MorphOS APIs make eBPF available to providers and define a runtime for tenants

Challenges of Unikernels



Lack of extensibility

Lack of isolation

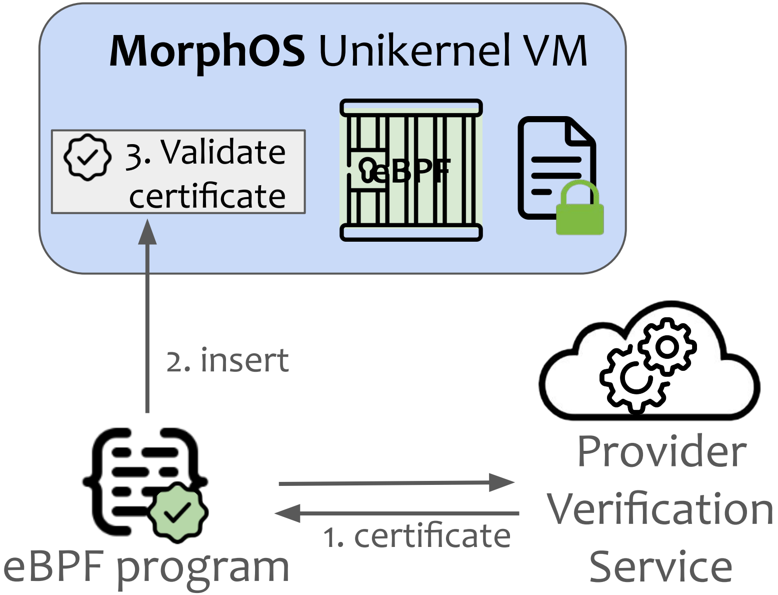
Solution #1
Hookpoints: extensibility with eBPF

Solution #2
Offline: Decoupled verification

Solution #3
Online: Hardware-assisted runtime hardening

#2 Decoupled eBPF Verification

- Protect against malicious tenants
  - Integrity, confidentiality, availability
- Verification service
  - Static control flow analysis
  - Track memory values and ranges
  - Generates cryptographic certificate
- Unikernel validates certificate



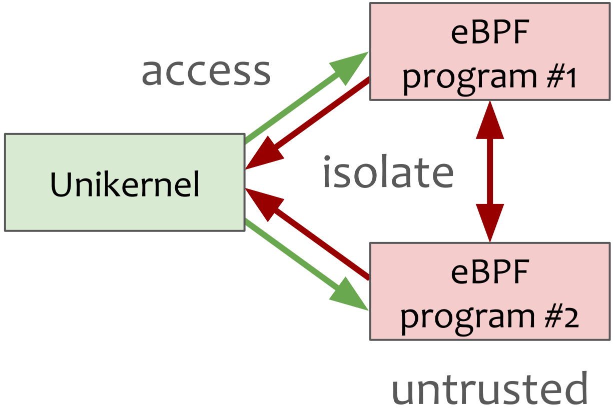
MorphOS allows providers to dedicate resources for asynchronous verification

#3 MPK: Hardening against Verifier Bugs

- Current eBPF verifiers are error prone
  - Linux verifier is unfit for security purposes
  - Correctness of verifiers remains unproven

    ⇒ Runtime hardening
- Memory Protection Keys (MPK)
  - 16 protection domains (PKeys)

    ⇒ Fast domain switches via `Wrpkru`



- Protection domains: unikernel, eBPF, packets, …
- Transport packets by changing protection domains

MorphOS hardens eBPF execution even in the presence of verifier/JIT bugs

Outline

- ~~Overview~~
- ~~Design~~
- Evaluation

Implementation and Case Study

MorphOS implementation:

- Built on **Unikraft** unikernel (v0.16.3)
- Extend **Prevail** verifier
- Extend the **uBPF** JIT compiler

Case study app: MorphClick

- Click: flexible VNF system
- Replace native algorithms with eBPF hookpoints
    - Firewall: replace ACL tables with JIT compiler
    - DPI (string matching): frequent reconfiguration
    - NAT: retain state across reconfigurations

Evaluation

- **RQ #1:** Does MorphOS improve **live-reconfigurability**?
  - Reconfiguration time
- **RQ #2:** Does MorphOS harden against **verifier bugs**?
  - Effectiveness of hardening with MPK
- **RQ #3:** Does MorphOS maintain **performance**?
  - Throughput

More evaluation in the paper

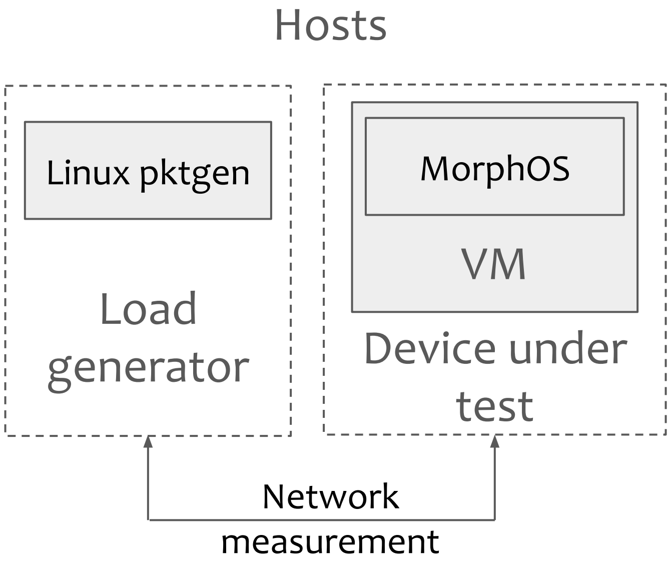
Experimental Setup and Variants

Setup:

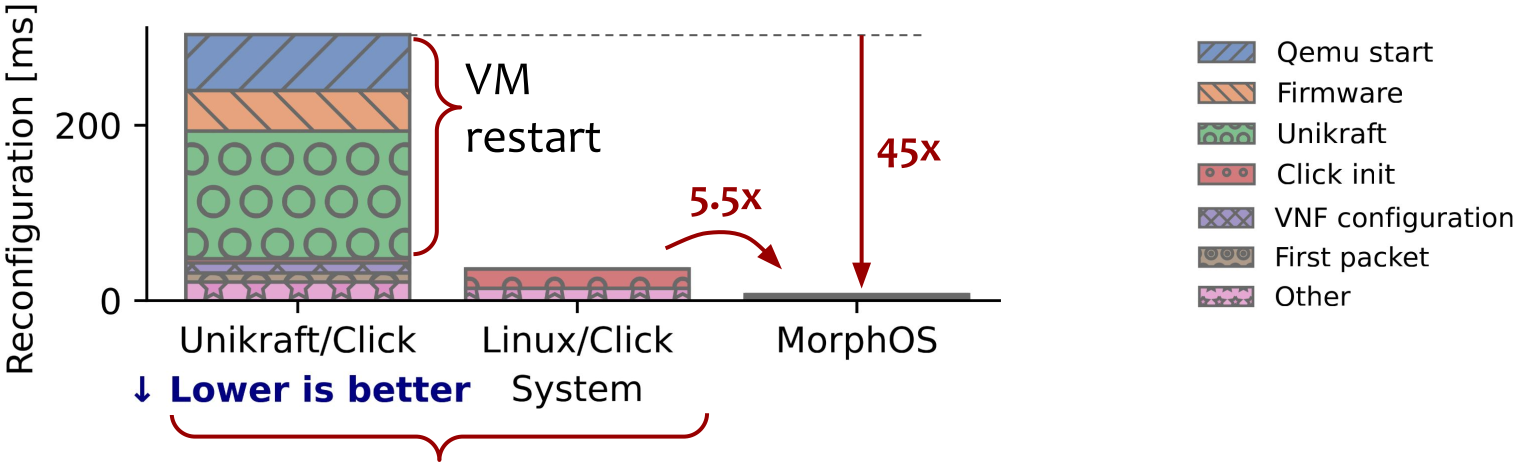
- Intel Xeon 5317, 256GB RAM
- 10G NIC: Intel X520
- Qemu 8.2.6 + VPP 24.06

Variants:

- **Linux + Click:** baseline
- **Unikraft + Click:** native VNF programs
- **MorphOS + MorphClick:** eBPF VNF programs



RQ#1: Lightweight Reconfigurability

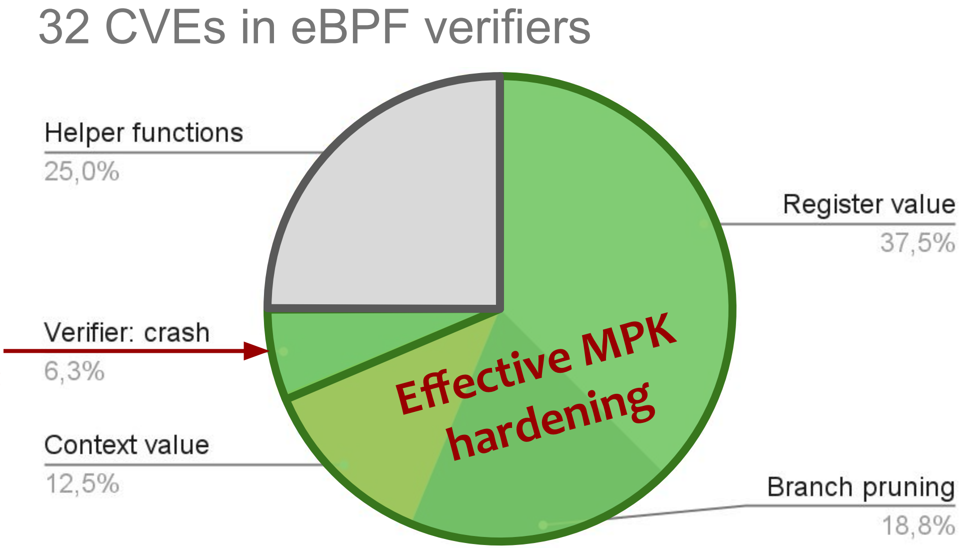


➔ Restart application to reconfigure

☠️ **loose runtime state**

MorphOS reduces reconfiguration time and preserves state

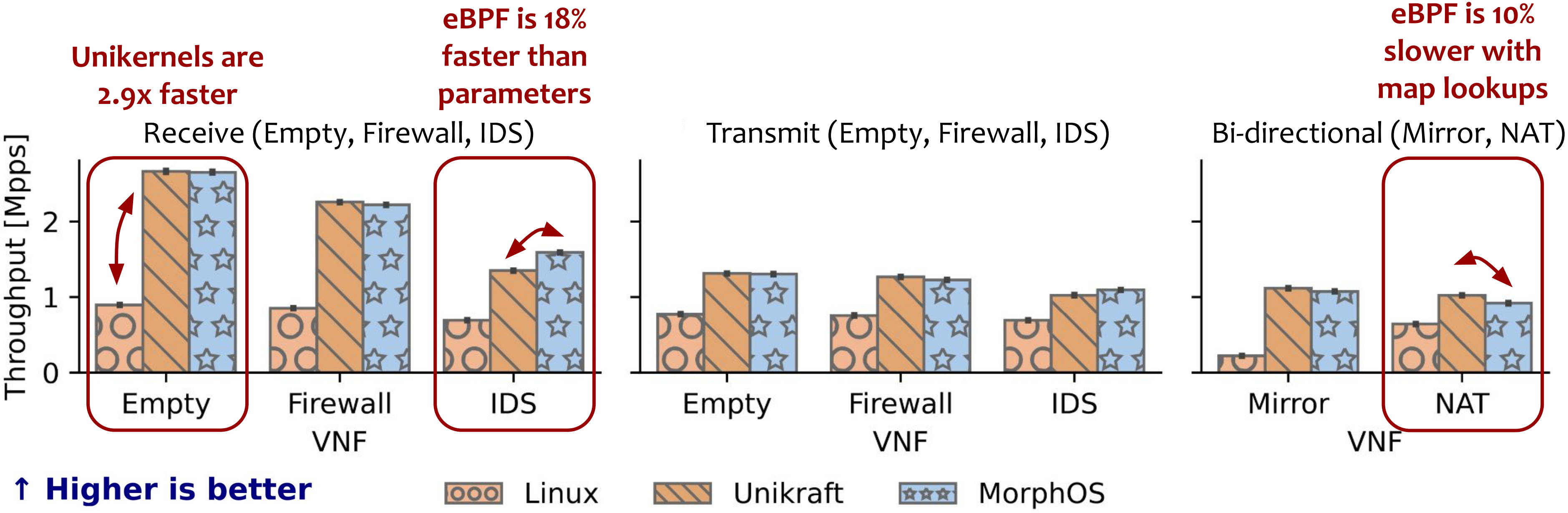
RQ#2: Safety Hardening



Mitigate with decoupled verification

MorphOS safety hardening protects against verifier vulnerabilities

RQ#3: Performance — Throughput



MorphOS is 3% faster than Unikraft without hardening and 3% slower with MPK

Conclusion

How do we achieve safe extensibility of networked unikernel applications?

MorphOS

- Introduces live-reconfigurability with verified eBPF to unikernels
- Provides strong correctness and safety guarantees
  - Verification, MPK hardening

Impact

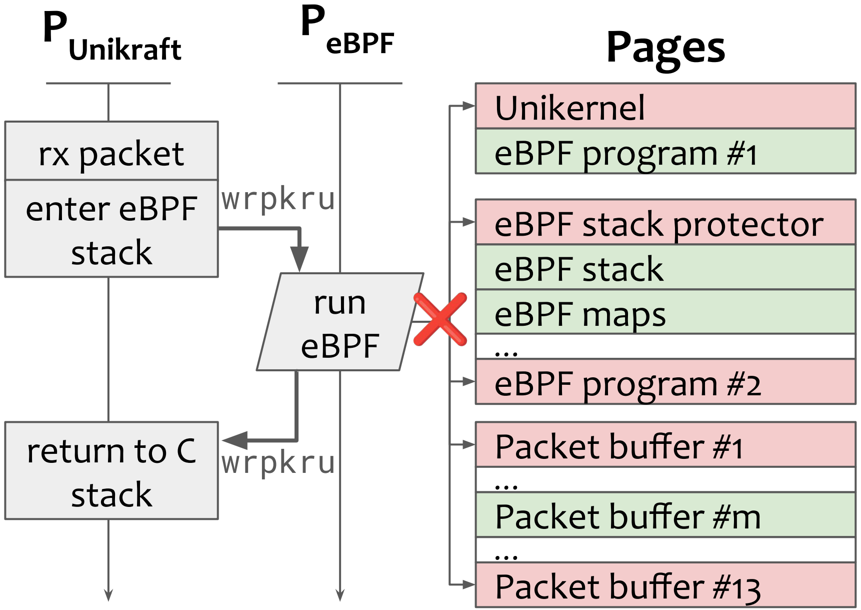
- Runtime-extensibility for unikernels
- Broaden the applicability of unikernels to real-world systems

TUM-DSE/MorphOS

Backup

Memory Safety with MPK

- Separate to different pages
    - eBPF program, stack, maps
    - Packet buffers
- Restrict memory permissions based on current context
- Selectively transport packet buffers into eBPF domain

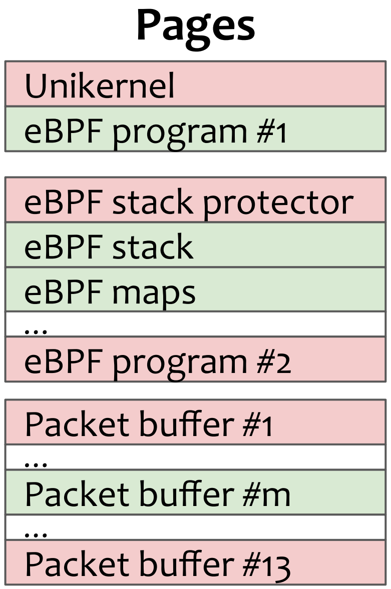


MPK scalability and probabilistic isolation

- MPK is limited to 16 protection domains
  - Exhausted with per-program and per-buffer domains
  - Existing eBPF hardening with MPK ignores IO buffers
- Naive approach: MPKey virtualization
  - libMPK: fall back to paging for cold set of domains
  - VNFs typically have >16 hot domains

-> Probabilistic isolation

  - Re-use protection domains
  - Accept permission overlap
  - Catch cloud tenants that bypass the verifier



Pages

Unikernel

eBPF program #1

eBPF stack protector

eBPF stack

eBPF maps

...

eBPF program #2

Packet buffer #1

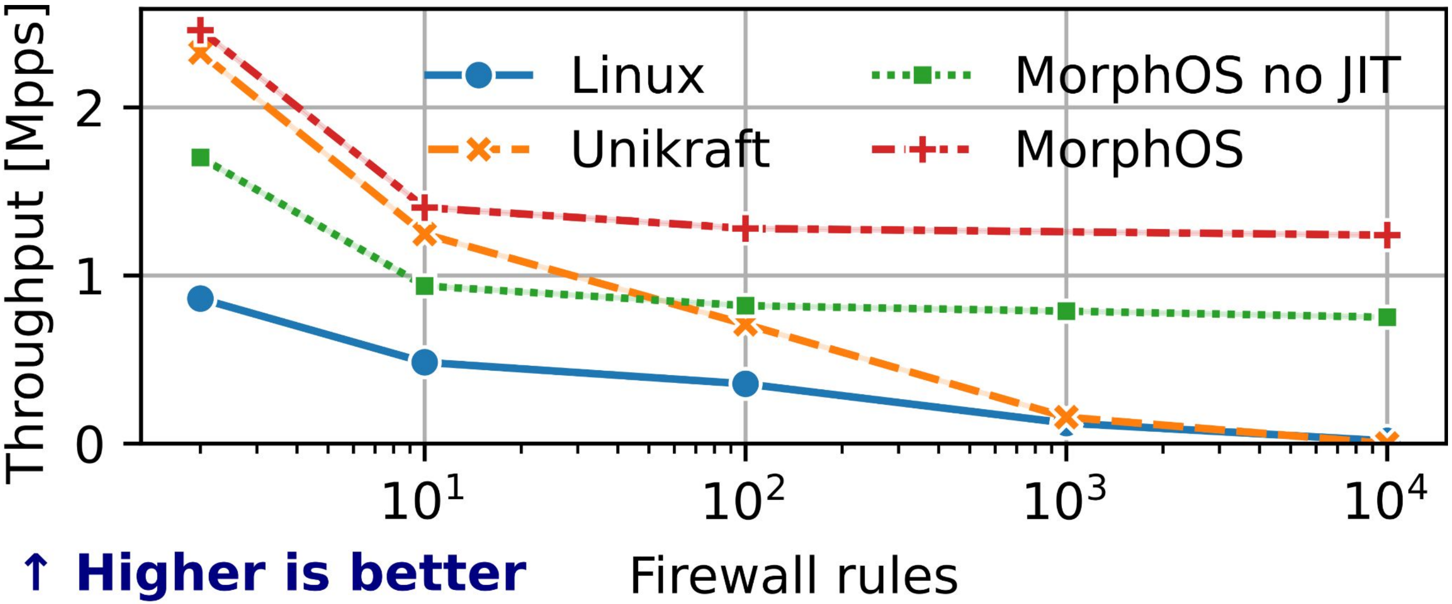
...

Packet buffer #m

...

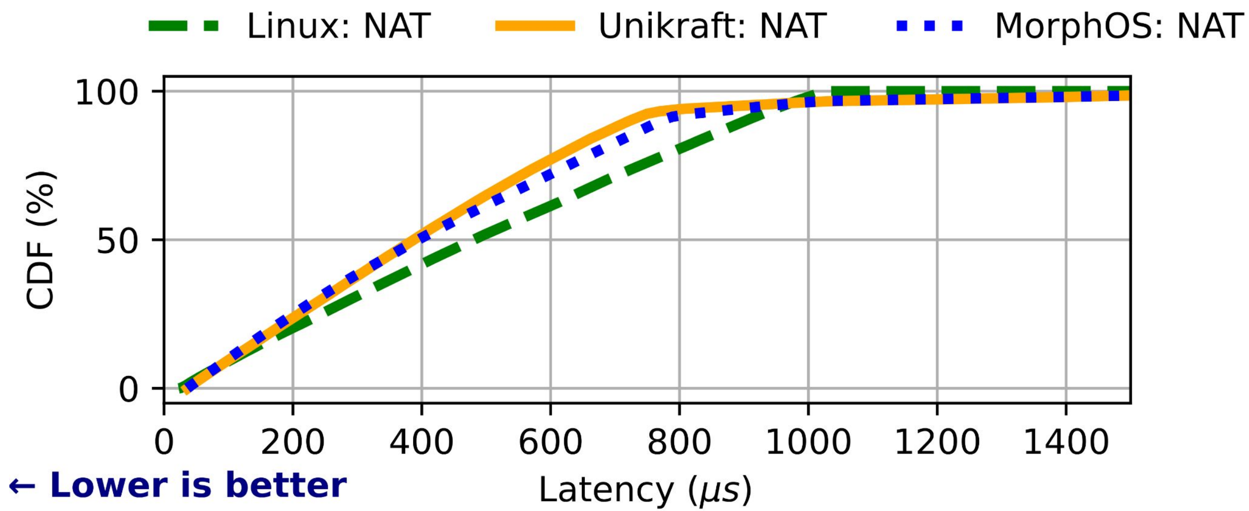
Packet buffer #13

Performance: Firewall



RQ#3: Performance — Latency

LoadGen: MoonGen (sw timestamps at 100kpps, 64B)



MorphOS and Unikraft reduce median latencies by 22% over Linux

Network Stack

- Central buffer management
  - Shared memory pool between app, OS, and eBPF
  - Zero-copy data transfers
  - Hardware-assisted isolation with MPK
- Event model
  - Run-to-completion model
  - Direct callback invocation
  - Optimized for cloud-native single-core VMs
  - Lock-free parallelism
- Network stack bypass
  - Direct driver access like DPDK
  - Preserves interrupt-based processing unlike DPDK

MorphOS combines kernel-stack bypass with the efficiency of unikernels

- Performance
  - Codesign: Compile app into unikernel
  - Changing the app requires recompilation
- Single-address space
  - Single-address space: reduce context switches
  - Lack of isolation
- Isolating extensions
  - Isolation through static verification
  - Verification is error prone