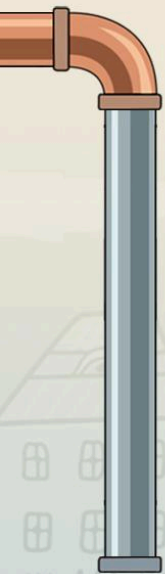


Safely Optimizing eBPF for Hardware and Workloads

Kinsn · speculative ReJIT · bpf-bench · KProg

Yusheng Zheng · Hao Sun (ETH Zurich) · Tong Yu

Mon 5 Oct 2026 · 17:00 · South Hall 1 A



**LINUX
PLUMBERS
CONFERENCE**

PRAGUE, CZECHIA / OCT. 5-7, 2026

THE PROBLEM

Verified eBPF is 1.6–2.0× slower than native



x86-64 1.57×

ARM64 1.98×

geomean, 27 microbenchmarks

One CPU, one workload. The pipeline knows neither.



LINUX
PLUMBERS CONFERENCE PRAGUE, CZECHIA / OCT. 5-7, 2026



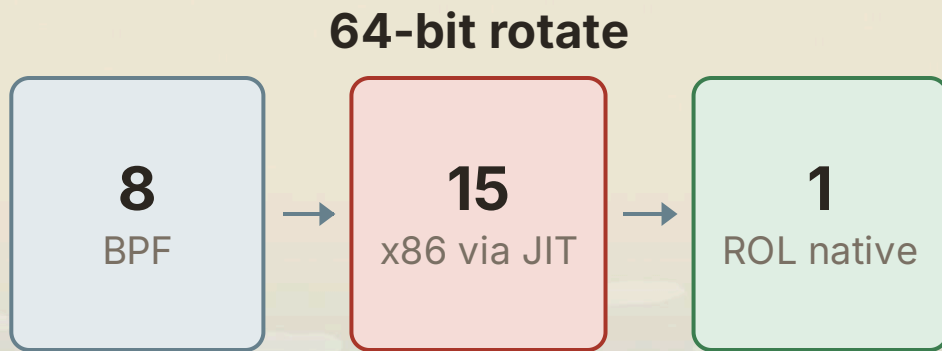
The loss is code generation

2% / 5%

cost of running in the kernel

93% / 94%

of the gap recovered by an
optimizing compiler



Don't grow the JIT

1 Slow upstream cycle

2 Verifier can't check native

3 Per-arch rewrites

Also missing: CPU, maps, branch bias.

Userspace proposes, the verifier disposes. One small interface.



Four pieces around one rule

USERSPACE

C / Rust



LLVM

**speculative ReJIT**
which rewrite?

KERNEL

KINSN

verifier



JIT



native

bpf-bench
did it help?**KProg**
how far can trust move?

One operation, two forms

kinsn operation

64-bit rotate-left

Proof sequence

shift + or, checked

=

Native emit

ROL on x86-64, ROR on
arm64

Verified as BPF. Executed as native.



A sidecar plus a kfunc call

```
; sidecar, metadata only
MOV_K
  src_reg = BPF_PSEUDO_KINSN_SIDECAR
  payload

; operation
CALL
  src_reg = BPF_PSEUDO_KFUNC_CALL
  imm      = BTF id
```

52-bit payload

dst

off

imm

No new opcode. Orphans rejected.

[REVIEW ITEM](#)

MOV src_reg



A descriptor bound to a kfunc BTF ID

```
struct bpf_kinsn {  
    max_insn_cnt;    /* proof bound */  
    max_emit_bytes;  /* native bound */  
    instantiate_insn(payload);  
    emit_x86/arm64();  
};  
btf_kfunc_id_set {  
    .kinsn_ids[], .kinsn_descs[]  
}
```

Kernel code only.

Usual kfunc filters apply.



Verifier: lower, check, restore



LENGTH BOUNDED

NO CALLS OR EXITS

JUMPS FORWARD, INSIDE REGION

Verifier never sees a kinsn.



No silent fallback

1 JIT opts in
default: no

2 Emitter
exists

3 Within
max_emit_bytes

4 Callee-
saved kept

restored pair

emit_x86 / emit_arm64

native bytes
no call

Any gate fails: the load fails.



Verifier checks the proof, not the emit

KERNEL CHECKS

proof is safe BPF

OWNER OWES

emit equals proof, every
payload

Lean 4

7 ops

100,000

random inputs

4 of 4

bad emitters caught



RFC v1: small core, two architectures

23/23

selftests

0

new failures

4/4

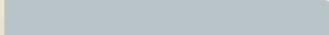
bad emitters caught

Added lines

core  525

x86  45

arm64  55

tests  588

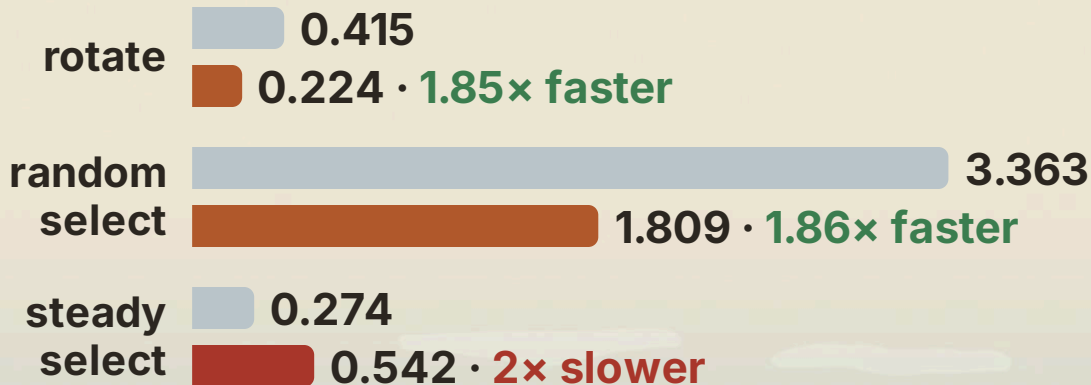
docs  78

625 kernel lines, in private review.



Wins on hard branches, loses on easy

ns/op: BPF vs kinsn



x86-64 KVM



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



On datapaths the gain shrinks to about 7%

1.24× / 1.22×

microbenchmark geomean

0.77× / 0.88×

JIT code size

1.074×

Cilium, x86-64

1.073×

Katran, ARM64

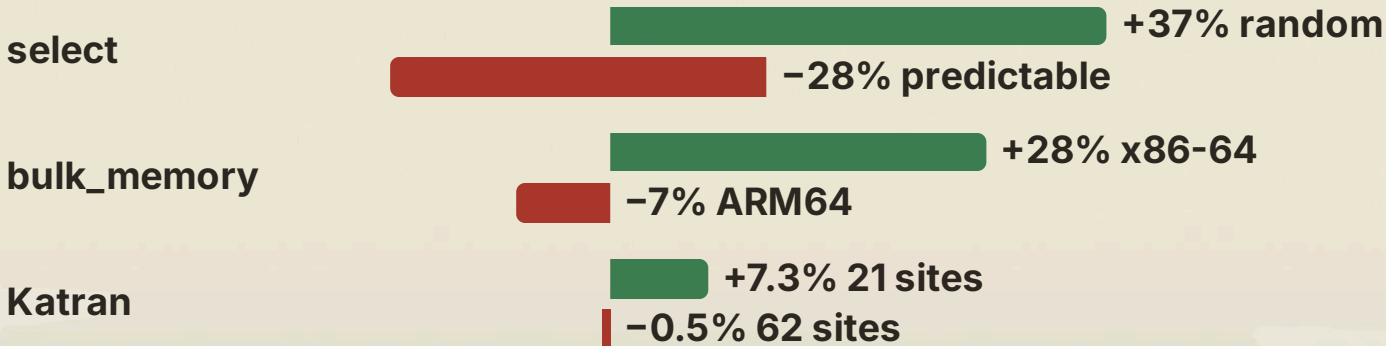
Helpers and maps dominate datapaths.



WHEN TO APPLY

More sites is not more speed

Percent change



Profit depends on CPU and workload.



Rewrite in userspace; the kernel re-verifies

runner



APPLICATION PROCESS

loader
unmodified



shim
LD_PRELOAD

bpftop

stock BPF_PROG_LOAD: verifier + JIT

No kernel patch.

Safety: verifier. Correctness: userspace.

kinsn is one more pass.



Speculation needs an explicit contract

Unconditional

Verifier-guided

Speculative

```
if (key == HOT_KEY)
    use inlined value
else
    bpf_map_lookup_elem()
```

TIER 1: DESIGN

RELOAD: UNMEASURED

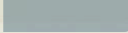
TIER 2: NOT BUILT

A guard miss takes the slow path.



Controls change the answer

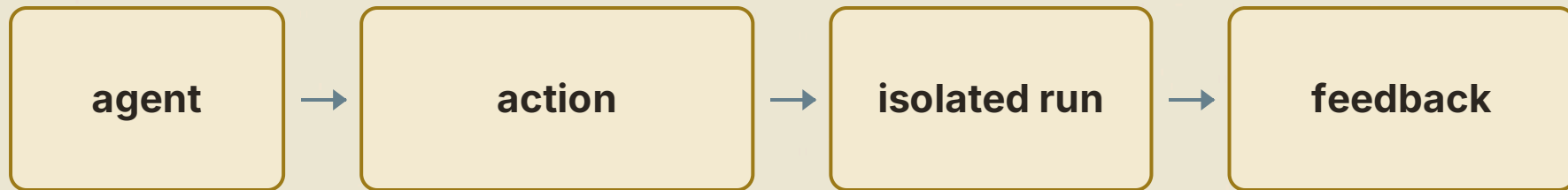
map_inline, corrected by restart control

Katran	 1.075× control 1.00
Tetragon	 1.036×
Tracee	 1.015×
BCC	 1.008×
Cilium	 0.962× control 1.07
OTel	 1.080× noisy controls

Restarting alone moved Cilium +7%.



A closed loop with a real oracle



6 apps, 146 programs, 42
microbenchmarks

Raw data only; hidden
correctness checks

Oracle: a gain beyond noise.



Most trials lose; a few win

3,197 vs 2,487

regressions vs improvements

2,299

verifier rejections

Cilium  +15.2%

Tetragon  +11.6%

OTel | 0%

Stacking five Cilium patches lost to one.

Paper draft, preliminary.

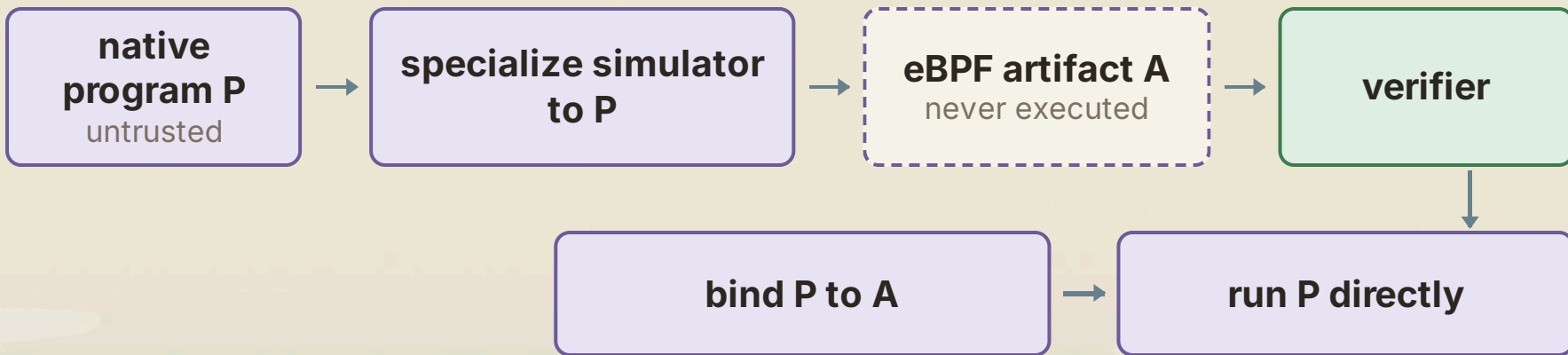


LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Native execution, eBPF-style checking



Safe only if the simulator is faithful.



Headroom is real, fidelity is open

1.47×

vs JIT, 29 microbenchmarks

54×

slower to load

Working

- 29/29 verify and run

Open

- equivalence is trusted
- ARM: QEMU, functional

A research direction, not a kernel proposal.



SUMMARY

A ladder of trust, priced in speedup

Stock eBPF	nothing	1×
Speculative ReJIT	nothing in kernel	Katran 1.075×
kinsn	one emit per op	1.24× / 1.07×
KProg	simulator fidelity	1.47×, no app

bpf-bench prices every rung.



Three decisions we need input on

1 Descriptor API

Is struct
`bpf_kinsn` right?

2 Emit equivalence

What review bar?

3 Sidecar encoding

`MOV src_reg = 3?`

Not asked of the kernel: when to apply.



TAKEAWAYS

Safe speed, priced in trust

The gap is code generation.

Kinsn: the owner owes emit equivalence.

Apply by measurement.

KProg: fidelity is open.

Paper arxiv.org/abs/2606.24213

Code github.com/eunomia-bpf/bpf-benchmark

Patches [RFC PATCH bpf-next v1 0/5]

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026