eBPF '26 Workshop @ SOSP

Unifying eBPF across Platforms: Formal Semantics, Conformance Testing, and Specifications

Yanze Li[1] (yanzeli@cs.ubc.ca), Reto Achermann[2], Ivan Beschastnikh[1], Alexander J. Summers[1] [1]University of British Columbia [2]Technical University of Munich

github.com/bpf-star

eBPF runs on Linux, Windows, in user space, on microcontrollers, and in blockchain VMs — on independently built runtimes.

Can one executable definition of eBPF serve as the reference that runtimes, verifiers, and the standard are checked against?

Motivation

- eBPF is cross-platform
  - **But** runtimes disagree: the same bytecode may produce different results
- An IETF ISA standard exists (RFC 9669)
  - **But** it is out of date: it omits instructions added since it was frozen and leaves corner cases unspecified
  - **But** its scope is narrow: it excludes the verifier, helpers, kfuncs, maps, and BTF
- Formal semantics exist (rBPF, Solana, Linux)
  - **But** each covers one platform

Gaps in RFC 9669

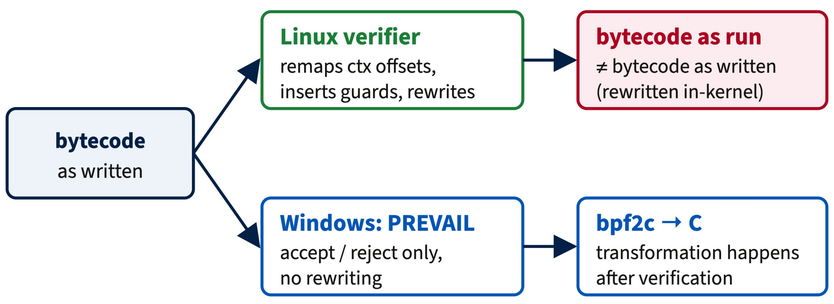
1. Unspecified by the RFC, implemented differently by runtimes.

Case	RFC	Linux	Win · uBPF · bpftime	rBPF · rbpf	sBPF
`INT_MIN s/ -1` sdiv overflow	— unspecified	`INT_MIN` verifier guard	`INT_MIN` guard†	`0` runs as `udiv`	own opcodes
load-acquire / store-release‡	—	✓ 6.15 (2025)	—	—	—
UHMUL / SHMUL high-half multiply	—	proposed MUL, off = 1 / 2	—	—	✓ PQR class, own encoding

† bpftime (llvmbpf) does not guard `smod` by −1: a constant divisor is folded away by LLVM, but a divisor that is not statically known triggers the bug.
‡ Linux added load-acquire/store-release in 6.15 (2025); RFC 9669 and the bpf_conformance suite do not include them.

2. Outside the RFC's scope: verifier, helpers, kfuncs, maps, BTF.

The RFC defines the instruction set only; each platform defines the rest.



The verifier determines what runs. Helpers, maps, and BTF determine what a program can call and access. All are platform-specific:

Helpers & kfuncs
Linux: 200+ helpers and kfuncs.
Windows, bpftime: a Linux-compatible subset.
uBPF: host-registered.

Maps
Linux: 30+ map types.
Other runtimes: a subset, with their own memory model and stack size.

BTF & CO-RE
Type information for kfunc calls, map definitions, and relocations.
Defined by Linux; partial or absent elsewhere.

Comparing runtimes alone is insufficient: verifier decisions and these interfaces must be modelled as well.

Approach

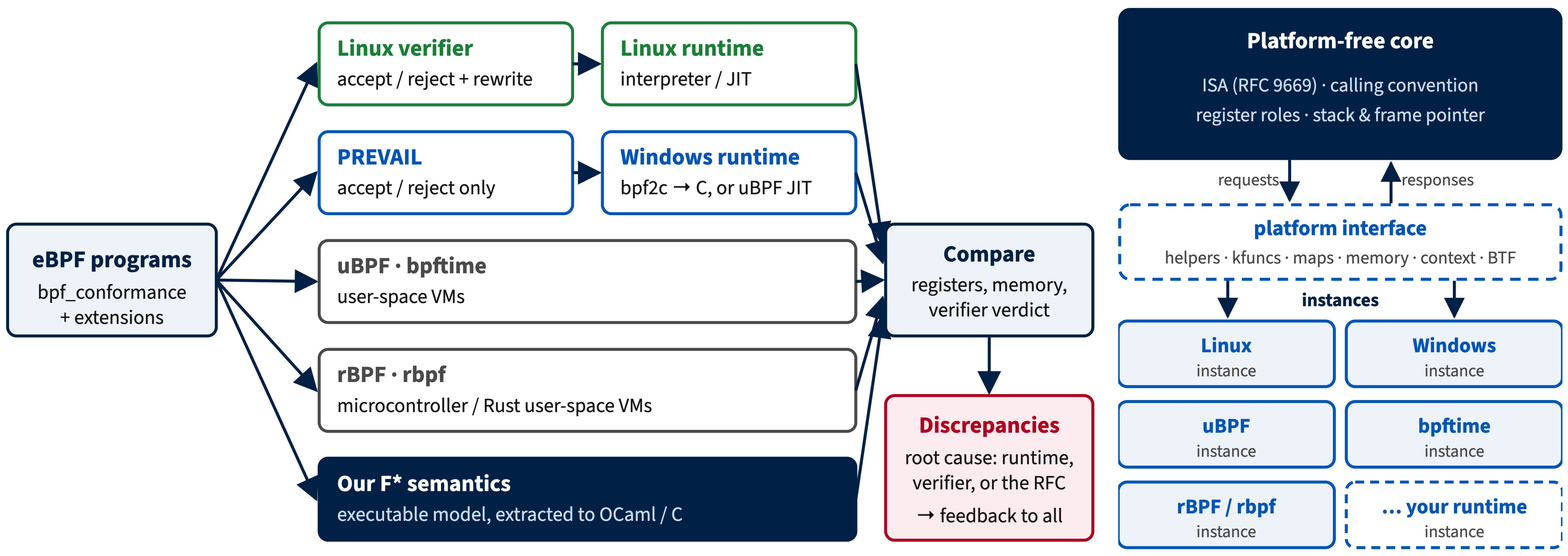
1. **F* and Pulse**: executable semantics with SMT-backed proofs, extracted to OCaml and C; with Pulse, verification of C runtimes and verifiers against the same semantics; **complementary to RFC 9669**, tracking what runtimes implement today
2. **bpf_conformance**: the ISA suite, which we are extending beyond the RFC to helpers, verifiers, and new instructions, run on every runtime and on our model: **confidence** in the model, **behavioural differences** across platforms
3. **Platform-specific semantics**: platform operations (helpers, maps, memory, context) are an F* effect over a free monad; **platform-free** core + **platform-specific** typeclass instances
4. **Meta-F***: metaprograms that generate prose specifications; **documentation with proved semantics correspondence**

Our contributions

1. **Executable cross-platform semantics** for eBPF in F*, with an explicit shared / platform-specific boundary
2. **Conformance testing** across runtimes: our semantics matches the reference runtimes on every modelled instruction; we are extending the bpf_conformance suite beyond its scope to helpers, verifiers, and new instructions
3. **Faithful specifications, in multiple prose styles**: a specification AST generated from the model renders to multiple prose forms (formal notation, RFC-style English), each certified against the semantics by a per-instruction lemma

Architecture

One executable model, written once against a platform interface, sits alongside the runtimes it is checked against.



Current status and findings

1. **Validated semantics:** 277 / 312 upstream and 27 / 27 custom tests; **all failures are due to unmodelled features**
2. **Even mature runtimes diverge from the RFC:** Linux's verifier rejects 12 RFC-valid shift tests; uBPF allows a 9-frame call chain where Linux stops at 8
3. **Extended tests exposed more bugs:** 27 tests beyond bpf_conformance found crashes, silent mis-decodes, and compile failures in bpftime, rbpf, rBPF, and uBPF's JIT
4. **Faithful prose (WIP):** formal semantics rendered into prose, with auto-generated equivalence lemmas

bpf_conformance results

Runtime	Version	Upstream (312)	Custom (27)[a]
Ours (F*)	—	**277 / 312**	**27 / 27**
Linux	`7.0.0-34, JIT`	300 / 312	27 / 27
Windows (bpf2c)	`v1.6.0`	312 / 312[b]	—[b]
uBPF interp. uBPF JIT	`4f7ef61`	312 / 312	26 / 27 25 / 27
bpftime (llvmbpf)	`bd032ec`	312 / 312	23 / 27
rbpf interp. rbpf JIT	`v0.4.1`	228 / 312	22 / 27 20 / 27
rBPF	`6987a20`	127 / 312	15 / 27

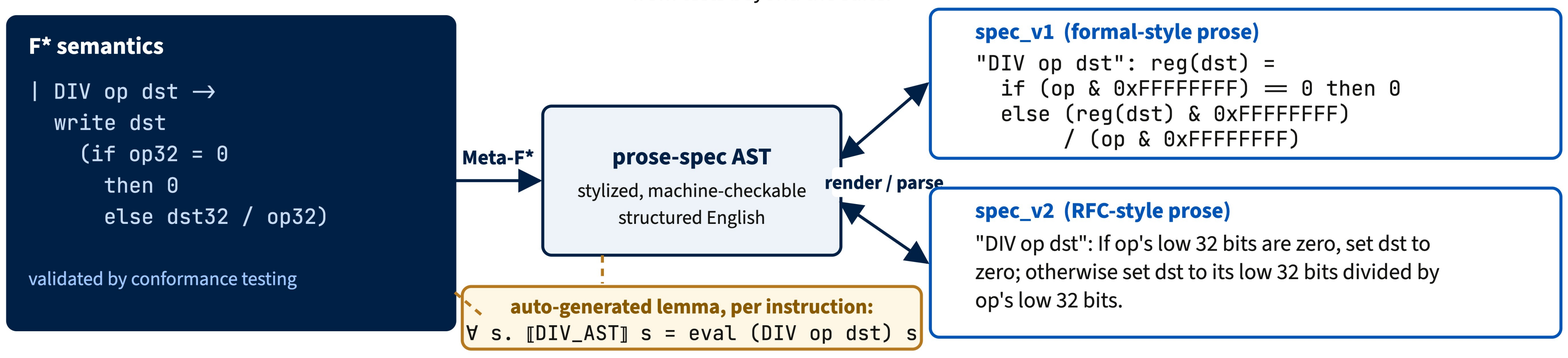
[a] 27 tests were added beyond the original bpf_conformance suite.
[b] Results taken from ebpf-for-windows CI; custom tests not run on Windows.

What the tests exposed

Runtime	Finding	Kind
Ours (F*)	34 atomics tests and one test needing a uBPF-only helper, all unmodelled; **no wrong answers**	unsupported
Linux	verifier rejects 12 shift-by-immediate tests with out-of-range amounts; RFC 9669 masks the amount	rejected
uBPF	allows a 9-frame call chain (Linux: 8); JIT cannot compile `ja32` with the maximum offset	bound rejected
bpftime (llvmbpf)	segfaults when a program starts with a local call; aborts on an out-of-bounds `ja32`	crash
rbpf	no ISA v4: `sdiv` / `smod` / `movsx` run as unsigned or unextended; sign-extension bug in 64-bit jump-imm (interp.); `mov32`, `le16`/`le32` wrong in JIT; no call-depth limit	wrong bound
rBPF	ISA v1 only; division by zero is an error; `div32` by zero crashes the host (SIGFPE); rejects call depth ≥ 7	unsupported wrong crash bound

crash the runtime crashes **wrong** the runtime returns an incorrect result **rejected** the compiler/verifier rejects a valid program **bound** a platform-specific bound, e.g. call depth **unsupported** feature not supported

Passing bpf_conformance is not enough: the discrepancies above come from verifier decisions and from tests beyond the suite.



Unlike documentation generated from ISA models (e.g. Sail), the generated prose is certified against the semantics.

Ongoing and future directions

1. Extend the semantics and the conformance suite beyond the RFC: helpers, maps, concurrency, memory models
2. Model the verifiers (Linux; PREVAIL) and formally verify their intended guarantees
3. Prose generation for the full ISA, tracked against the IETF documents

References

1. D. Thaler. *BPF Instruction Set Architecture (ISA)*. RFC 9669, IETF, 2024.
2. A. Jowett. *bpf_conformance: Tests Measuring BPF Runtime Conformance to the ISA*.
3. N. Swamy et al. *Dependent Types and Multi-Monadic Effects in F**. POPL 2016.
4. G. Martínez et al. *Meta-F*: Proof Automation with SMT, Tactics, and Metaprograms*. ESOP 2019.
5. E. Gershuni et al. *Simple and Precise Static Analysis of Untrusted Linux Kernel Extensions*. PLDI 2019.
6. S. Yuan et al. *Formalizing the Linux eBPF Core ISA*. OOPSLA 2026.

Supported by NSERC Discovery Grants RGPIN-2025-06826, RGPIN-2025-06888, and RGPIN-2025-04203. We thank Hugh Coleman, Ning Ding, Dennis Gu, and Jiayin Kralik for their contributions.