Measuring BPF Implementation Adherence: The bpf_conformance Project

A Presentation for the Linux Plumbers Conference
Presenter: Alan Jowett alan.jowett@microsoft.com
Microsoft Corporation
Agenda

• Introduction
• Overview of the bpf_conformance suite
• The motivation for this test suite
• How conformance is measured
• Q&A
Introduction

• BPF – a synthetic ISA (instruction set architecture) used by a software-defined virtual machine that executes in the Linux kernel and other environments.
• BPF ISA is in the process of being standardized by a IETF working group.
• BPF runtimes, both software and hardware, are proliferating rapidly.
• If programs are to be portable, then all runtimes need to agree on the ISA.
What is bpf_conformance?

• A project that measures a BPF runtime’s compliance with the draft IETF BPF specification.
• Built on the uBPF project’s test collateral and extended to cover newer ISA versions.
• Can be used either as a CLI or as a library.
• Uses the Linux kernel BPF runtime to validate the test collateral.
Why perform conformance testing?

• Validation of the draft IETF specification
  • Verify that the Linux kernel BPF runtime and the draft spec match
  • Identify gaps in the IETF draft.

• Interoperability
  • By creating a test suite for BPF ISA, the conformity of BPF runtimes can be measured.
  • Ensure that the same bytecode has the same behavior in all runtimes.

• Bug detection
  • Provide a mechanism to detect if an BPF runtime works as intended.
Why – Part 2

• Standardization Compliance
  • A specification is more useful if there is a way to test the assertion that a runtime conforms to that specification.

• Security
  • The conformance suite is being used by the PREVAIL Verifier to validate the model for the ISA.

• Documentation
  • The conformance suite provides a method to quickly test out BPF bytecode, allowing documentation authors to tease out the behavior of the Linux BPF runtime without accessing the GPL source.
BPF Instruction Set Architecture

• RISC Architecture: BPF follows a Reduced Instruction Set Computing (RISC) design for efficiency.
• Bytecode and Registers: BPF programs are expressed in bytecode and use registers for data storage.
• Packet Processing: BPF was originally used for packet filtering and manipulation in networking but is now more general purpose.
• Safety and Security: BPF is designed to run safely within a sandboxed environment, preventing unintended side effects.
• Widely Used: BPF was initially adopted in the Linux kernel and has been implemented in a variety of other platforms.
Conformance Testing Approach

- Declare an initial state for the VM
  - Currently just the context memory.
  - Pre-populated maps.

- Declare a set of BPF instructions to execute
  - Declared as a set of BPF assembly.
  - GCC style assembly (currently using handwritten parser).

- Declare an expected return code
  - BPF runtime is expected to return the contents of %r0 as a 64bit unsigned integer.
Example BPF conformance test case

# Copyright (c) Big Switch Networks, Inc
# SPDX-License-Identifier: Apache-2.0

-- asm
mov32 %r0, 0xf8
mov32 %r1, 16
lsh32 %r0, 28
# %r0 == 0x80000000
arsh32 %r0, %r1
exit
-- result
0xffff8000
The bpf_conformance suite

- Test suite consists of a runner, a set of tests, and a set of plugins.
  - Runner
    - Parses each test file
    - Generates BPF bytecode from assembly mnemonics.
    - Invokes the plugin.
    - Checks the return value.
    - Records statistics (which instructions have been tested).
  - Tests
    - A small snippet of BPF assembly.
    - Initial value in context.
    - Expected %r0 on exit.
  - Plugins
    - Platform-specific wrapper for invoking BPF runtime.
The Implementation Variability Challenge

• BPF runtimes are exposed via widely differing APIs
  • All APIs share a common behavior but vary significantly.
• OS platforms have a variety of IPC mechanisms.
• Command Line Interface
  • Input / output streams
  • List of arguments
• The bpf_conformance runner interacts with the per-runtime plugin.
• Plugin accepts bytes code and context, returning the value of %r0 register on completion.
Variability Challenge – Part 2

• Plugin design is kept as simple as possible

• Plugins exist for:
  • Linux – Loads the BPF bytecode via libbpf
  • PREVAIL
  • eBPF-for-Windows – Uses bpf2c compile bytecode to native and execute it.
  • rbpf – Rust wrapper that passes the bytecode to the rbpf runtime.
  • uBPF – Loads BPF bytecode into the uBPF VM then either interprets or JIT executes the code.
  • WASM uBPF – Permits running BPF in the browser.
Future improvements

• Pass ELF file containing BPF program
  • Passing map definitions
  • Local calls
  • Waiting on finalization of ELF BPF specification

• Enhance the existing test cases
  • Test cases derived from uBPF
  • Lack support for some newer BPF instructions (work is in progress to add support for CPU v4)

• Deprecate BPF assembler in favor of GCC assembler?

• GitHub CI/CD uses older Linux kernel
  • Makes it challenging to test newer instructions in CI/CD
Q&A

• GitHub Repo
  • https://github.com/Alan-Jowett/bpf_conformance/
Thanks

• Dr Dave Thaler
  • Primary contact for the IETF BPF ISA specification

• Dr Will Hawkins
  • Bug fixing and beta testing 😊
  • Reviewing this presentation