BPF Performance testing

How to compare performance of the BPF runtime across platforms

Problem statement

- BPF programs are becoming cross-platform
- BPF programs are often in performance sensitive paths
- What can developers expect in terms of performance

Runtime performance

- Does this even matter?
  - Direct impact to cycles per byte
  - Direct impact to latency and jitter
- What should be measured?
  - Runtime overhead – transition from kernel -> BPF VM
  - Helper function performance
- How should it be measured?
  - Platform neutral
  - Repeatable

github.com/microsoft/bpf_performance

- MIT Licensed project
- Collection of tiny BPF programs
  - Written to be platform agnostic
  - Compiled per platform
- Uses libbpf to be platform agnostic
  - Loads program into kernel
  - Schedules via bpf_prog_test_run_opts
  - Runs concurrently on N CPUs

What is being measured

- Baseline – Cost of an empty BPF program
- Generic map
  - bpf_map_lookup_elem
  - bpf_map_update_elem
  - bpf_map_delete_elem
- Helpers
  - bpf_get_prandom_u32
  - bpf_ktime_get_ns
  - bpf_tail_call
- LPM_TRIE and other map types

Special cases

- Longest Prefix Match
  - Prefix population built from BGP data
  - https://bgp.potaroo.net/as2.0/bgp-active.html
  - Attempts to be representative of the internet routing tables
- Least Recently Used Hash-Table
  - Random lookup/update/delete
  - Rolling lookup/update

How measurements are taken

- Test divided into two phases
  - Prep
  - Execution
- Prep phase
  - Populate maps (if needed)
  - Runs on a single CPU core
  - Not measured
- Execution
  - Runs on specified set of CPUs
  - Executes in parallel

How eBPF for Windows uses it

- Runs daily as part if CI/CD
  - Allows tracking of changes of performance over time
- Results are published to Grafana
  - Allows for easier viewing of results
- Public dashboard:
  - Grafana (bpfperformancegrafana.azurewebsites.net)
- Comparison of Windows vs Linux performance
  - Currently blocked on infrastructure
  - Linux tests running as GitHub runners (vs self-hosted for Windows)
  - Data shows too much variability
  - Windows uses AOT vs Linux JIT

Lessons from Windows

- JIT vs AOT vs Interpret
  - The ahead-of-time compilation is significantly faster
  - C Compiler can generate more optimal code than JIT
- Lack of kernel integration
  - Tracking per-thread state adds a high cost
- LPM as a hash-table (instead of a TRIE)
  - Lookup perf is close
  - Update significantly outperforms
- LRU
  - Managing global consensus on key age is expensive
  - Partitioned Generational LRU performs best