Achieving lower latency with eBPF and XDP

Toke Høiland-Jørgensen

Principal Kernel Engineer, Red Hat

Understanding Latency webinar
March 7th, 2023

What is eBPF ?

From: https://ebpf.io/what-is-ebpf

> *eBPF is a revolutionary technology that can run sandboxed programs in the Linux kernel without changing kernel source code or loading a kernel module*

Rate of innovation at the operating system level: Traditionally slow

- eBPF enables things at the OS level that were not possible before
- eBPF can radically increase rate of innovation

eBPF components

Closer look at the eBPF components:

- Bytecode - Architecture independent Instruction Set
  - JIT to native machine instructions (after loading into kernel)
- Runtime environment - Linux kernel
  - Event based BPF hooks all over the kernel
  - Per hook limited access to kernel functions via helpers and kfuncs
- Sandboxed by the eBPF verifier
  - Limits and verifies memory access and instructions limit

What is XDP?

XDP (eXpress Data Path) is a Linux in-kernel fast path

- Programmable layer in front of the kernel networking stack
  - Read, modify, drop, redirect or pass
  - For L2-L3 use cases: seeing x10 performance improvements!
- Avoiding memory allocations
- Adaptive bulk processing of frames
- Very early access to frame (in driver code after DMA sync)
- Ability to skip (large parts) of kernel code

XDP performance

XDP_DROP: 100Gbit/s mlx5 max out at 108 Mpps (CPU E5-1650v4 @3.60GHz)

Lower latency with eBPF

eBPF can help improve latency in a number of ways

- Lower latency by increasing PPS performance
- Custom software RSS steering
- Passive latency monitoring
- Future: Queueing for XDP

Scaling a latency-reducing traffic shaper

Use case: ISP middlebox providing per-customer bandwidth enforcement and bufferbloat mitigation (using kernel queueing infrastructure)

Problem: Software shaping doesn't scale because of global qdisc lock

Solution: XDP can choose which CPU to start the Linux networking stack on - steer a subset of customers to each CPU, so CPUs can run independently (avoiding the lock contention)

https://github.com/xdp-project/xdp-cpumap-tc
https://github.com/LibreQoE/LibreQoS

Passive latency monitoring

Use case: Monitor TCP traffic and extract flow latency (using TCP timestamps) to passively monitor traffic flowing through a middlebox.

Problem: The existing solution in software (pping) doesn't scale to high bandwidths

Solution: eBPF can inspect every packet with very low overhead - implement the monitoring in the kernel with eBPF, only export metrics to userspace

https://github.com/xdp-project/bpf-examples/tree/master/pping

Lower latency by increasing PPS performance

Use case: Linux-based servers exposed to the internet with high-speed NICs.

Problem: Head-of-Line (HOL) blocking in NIC RX rings causes latency and packet drops if kernel can't keep up with flow rate.

Solution: Implement filtering and redirection in XDP, allowing the kernel to keep up with the incoming packet rate.

Ex: https://blog.cloudflare.com/l4drop-xdp-ebpf-based-ddos-mitigations/

Future: Queueing for XDP

Use case: High-performance forwarding path with XDP "software offload"

Problem: XDP currently has no support for packet queueing and scheduling

Solution: We're working on adding programmable queueing support to XDP

https://lpc.events/event/16/contributions/1351/

Closing remarks

eBPF allows unprecedented visibility into the OS, and safe, dynamic extensibility of core OS features, networking in particular.

eBPF unlocks the kernel's potential for innovation

- Pioneered on Linux, but exists in Windows too: https://github.com/microsoft/ebpf-for-windows
- The eBPF Foundation (working on standardisation): https://ebpf.foundation/
- More examples of applications using eBPF: https://ebpf.io/applications
- Code examples: https://github.com/xdp-project/bpf-examples
- XDP tutorial: https://github.com/xdp-project/xdp-tutorial

End: Questions?