eBPF: Unlocking the potential of the Linux kernel

Jesper Dangaard Brouer

Sr. Principal Kernel Engineer, Red Hat

Driving IT
November 4th, 2022

What is eBPF ?

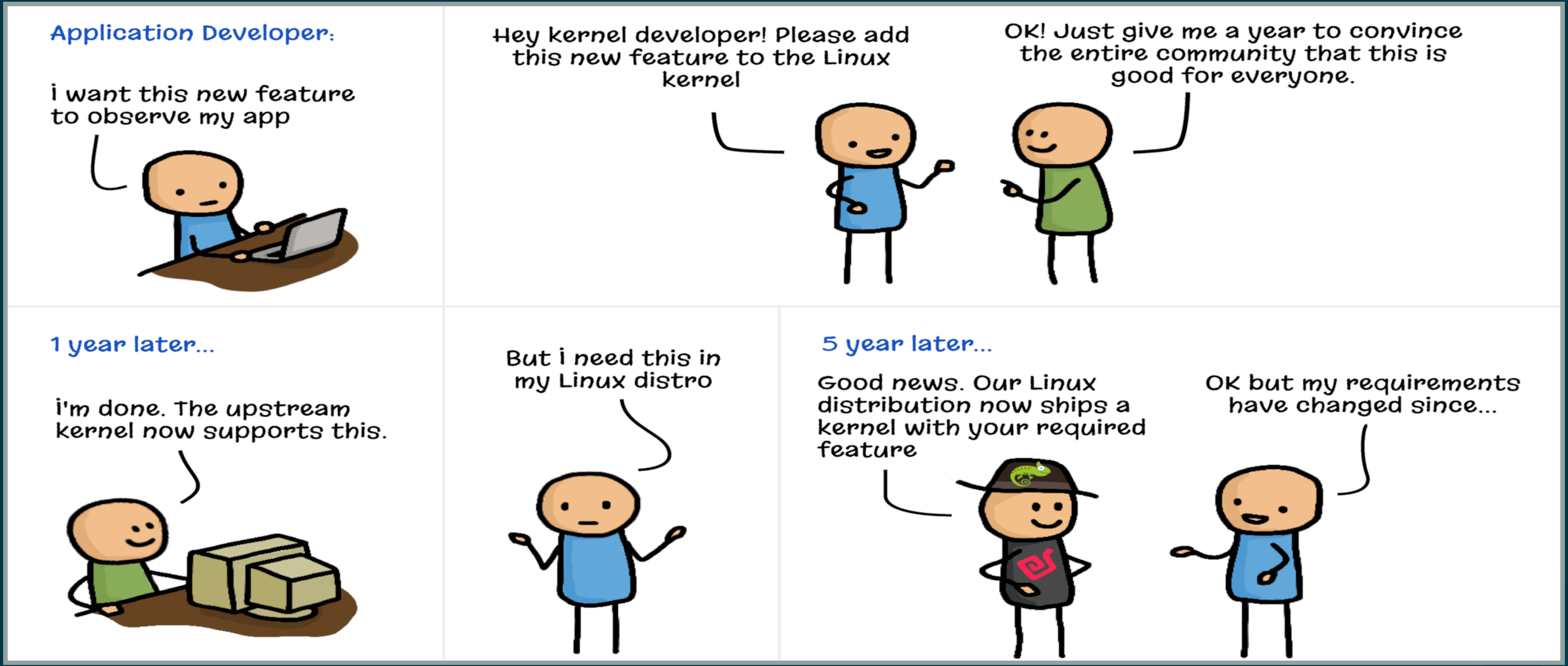
From: https://ebpf.io/what-is-ebpf

> *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

Traditional Kernel development process



Application Developer:
i want this new feature to observe my app
Hey kernel developer! Please add this new feature to the Linux kernel
OK! Just give me a year to convince the entire community that this is good for everyone.
1 year later...
i'm done. The upstream kernel now supports this.
But i need this in my Linux distro
5 year later...
Good news. Our Linux distribution now ships a kernel with your required feature
OK but my requirements have changed since...

eBPF development process

Application Developer:
i want this new feature to observe my app
eBPF Developer:
OK! The kernel can't do this so let me quickly solve this with eBPF.
A couple of days later...
Here is a release of our eBPF project that has this feature now. BTW, you don't have to reboot your machine.

eBPF components

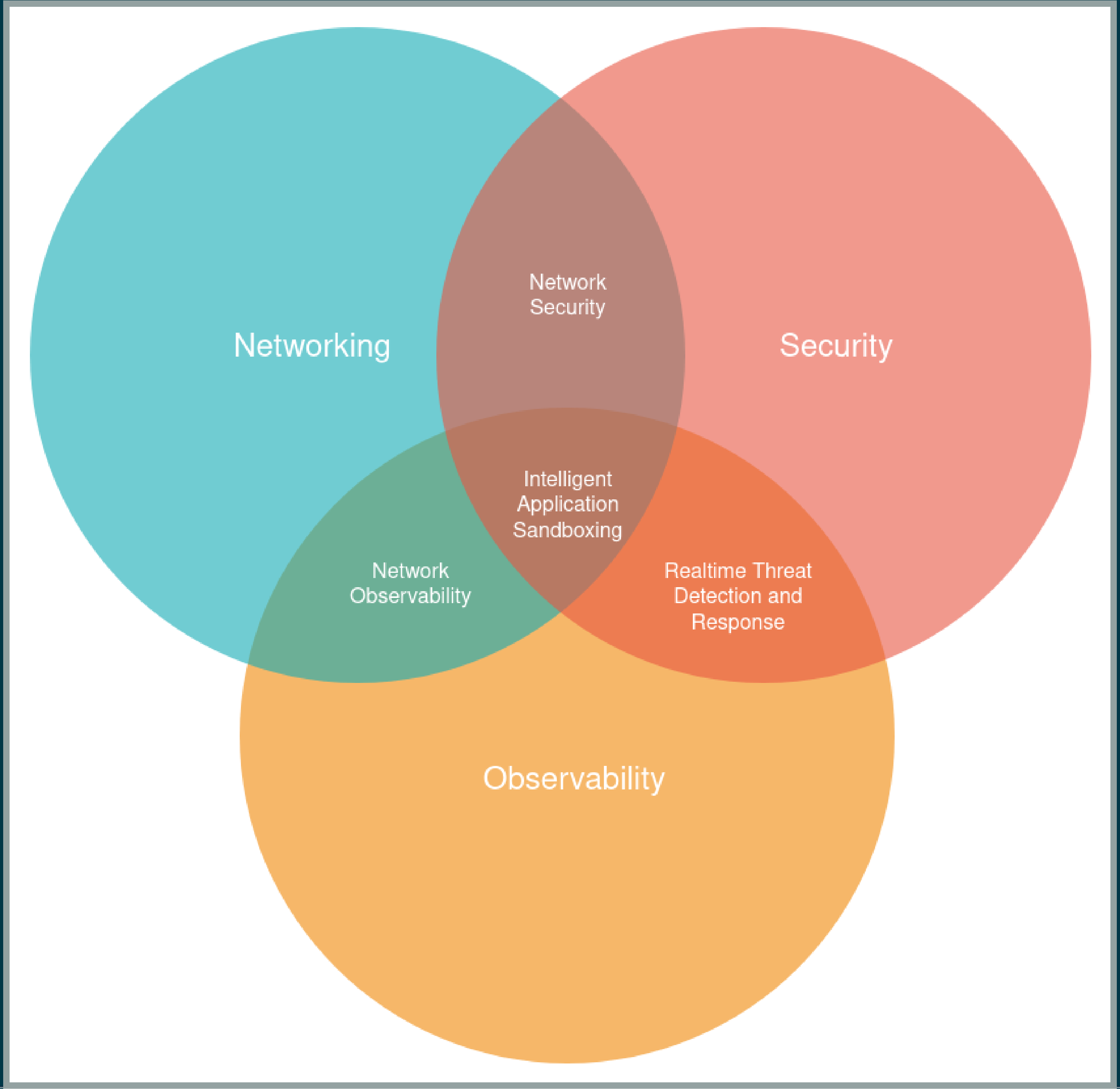
Closer look at the eBPF components:

- 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 BPF helpers
- Sandboxed by the eBPF verifier
  - Limits and verifies memory access and instructions limit

eBPF use cases

- Networking
  - Use eBPF to amend the data path with new features
  - CPU efficiency: Use XDP to keep up with high packet rates
    - Accelerate firewall, load balancing, forwarding
- Monitoring
  - Low overhead performance monitoring
  - Application resource usage reporting
- Security
  - Firewalling and DDoS protection
  - Application isolation
  - Custom security monitoring and enforcement

eBPF application areas


Networking
Network Security
Security
Intelligent Application Sandboxing
Network Observability
Realtime Threat Detection and Response
Observability

eBPF networking

Focus on eBPF for networking

- XDP (eXpress Data Path) for fast processing at ingress
- TC-BPF hooks inside the regular stack
- eBPF hooks for cgroups can also be useful for containers

What is XDP?

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

- Programmable layer in front of traditional network stack
  - Read, modify, drop, redirect or pass
  - For L2-L3 use cases: seeing x10 performance improvements!
- Avoiding memory allocations
  - No SKB allocations and no init (SKB zeroes 4 cache-lines per pkt)
- Adaptive bulk processing of frames
- Very early access to frame (in driver code after DMA sync)
- Ability to skip (large parts) of kernel code
  - Evolve XDP via BPF helpers

XDP performance

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

- PCIe tuning needed - NIC compress RX-descriptors (`rx_cqe_compress on`)

eBPF networking use cases

- DDOS filtering
- Custom software RSS steering
- Passive latency monitoring
- NAT64 gateway

DDOS filtering

Use case: Filtering DDoS attack traffic at line rate on servers instead of on a dedicated firewall.

Problem: The kernel firewall (iptables/netfilter) doesn't scale to high line rates (10-100Gbps)

Solution: Implement the filtering in XDP, allowing it to scale to line rate with low overhead.

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

Custom software RSS steering

Use case: ISP middlebox providing per-customer bandwidth enforcement (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

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

NAT64 gateway

Use case: NAT64 gateway for IPv4-IPv6 transition

Problem: Existing open source implementation (Tayga) routes packets through user space, causing bad performance and bufferbloat.

Solution: Implement the translation inband in the kernel path using eBPF - adding a new feature to the networking stack without changing kernel code

https://github.com/xdp-project/bpf-examples/tree/master/nat64-bpf

eBPF and Red Hat

We support eBPF on RHEL:

- Full kernel eBPF backports (RHEL 8.7: kernel 5.14, RHEL 9.1: kernel 5.16)
- Support for eBPF kernel features, bcc-tools and bpftrace

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/9.0_release_notes/new-features#BZ-2070506

We develop eBPF:

- Upstream kernel contributions (networking, tracing, HID)
- Userspace libraries and tools (libxdp, Aya)
- Code examples and docs (xdp-tutorial, bpf-examples)

We are a platinum member of the eBPF foundation.

Closing remarks

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

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?

Red Hat

❤

eBPF