XDP and BPF insights

Programmable Runtime Extending Linux Kernel for Packet Processing

Jesper Dangaard Brouer
Senior Principal Kernel Engineer

Technical BPF introduction talk
November 2021

Overview: What will you learn?

What is BPF really ?

- How this technology fundamentally changes existing OS-model

Taming BPF superpower - is not easy

- BPF “user experience” - could be better!

What is XDP?

- and what pain points have recently been resolved

What is AF_XDP?

- How is this connected with XDP and deep-dive into tech details

What is BPF ?

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

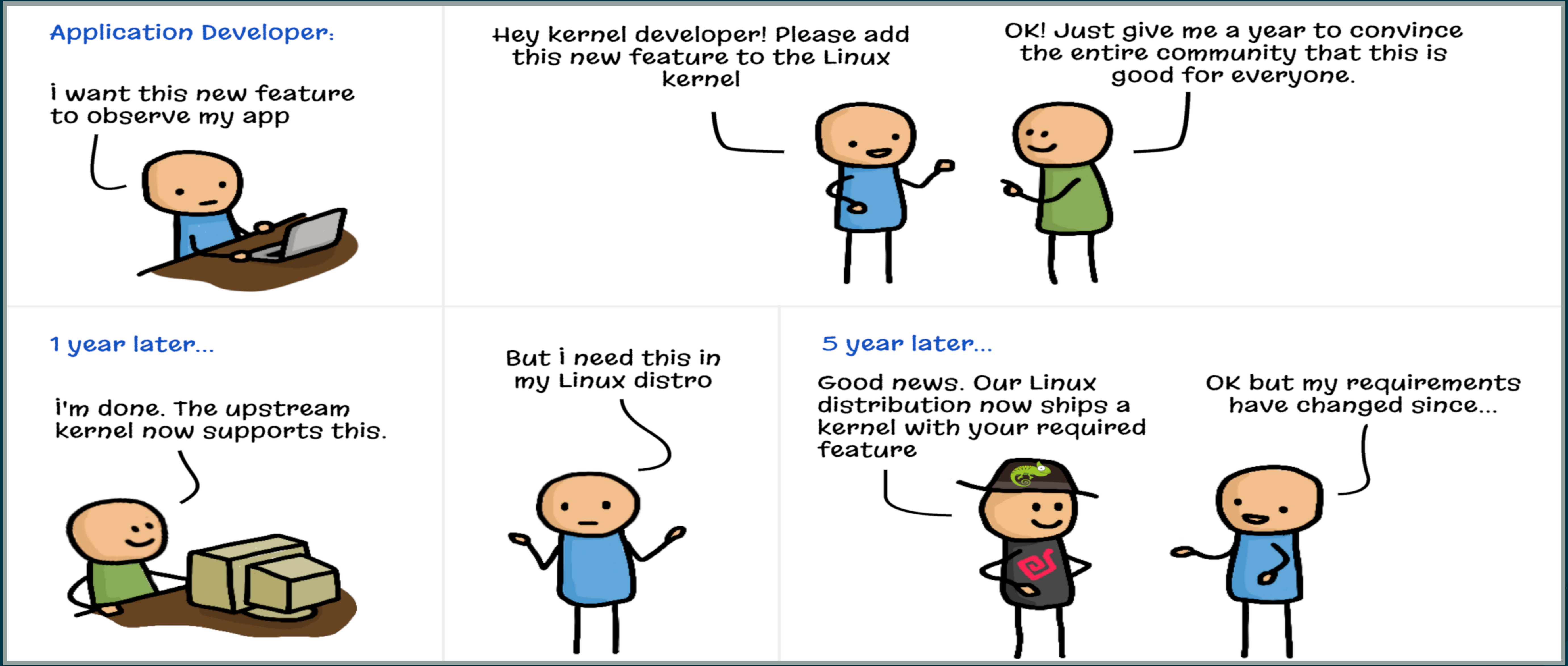
> *eBPF is a revolutionary technology that can run sandboxed programs in the Linux kernel without changing kernel source code or loading a kernel module*

BPF is a technology name: no longer an acronym

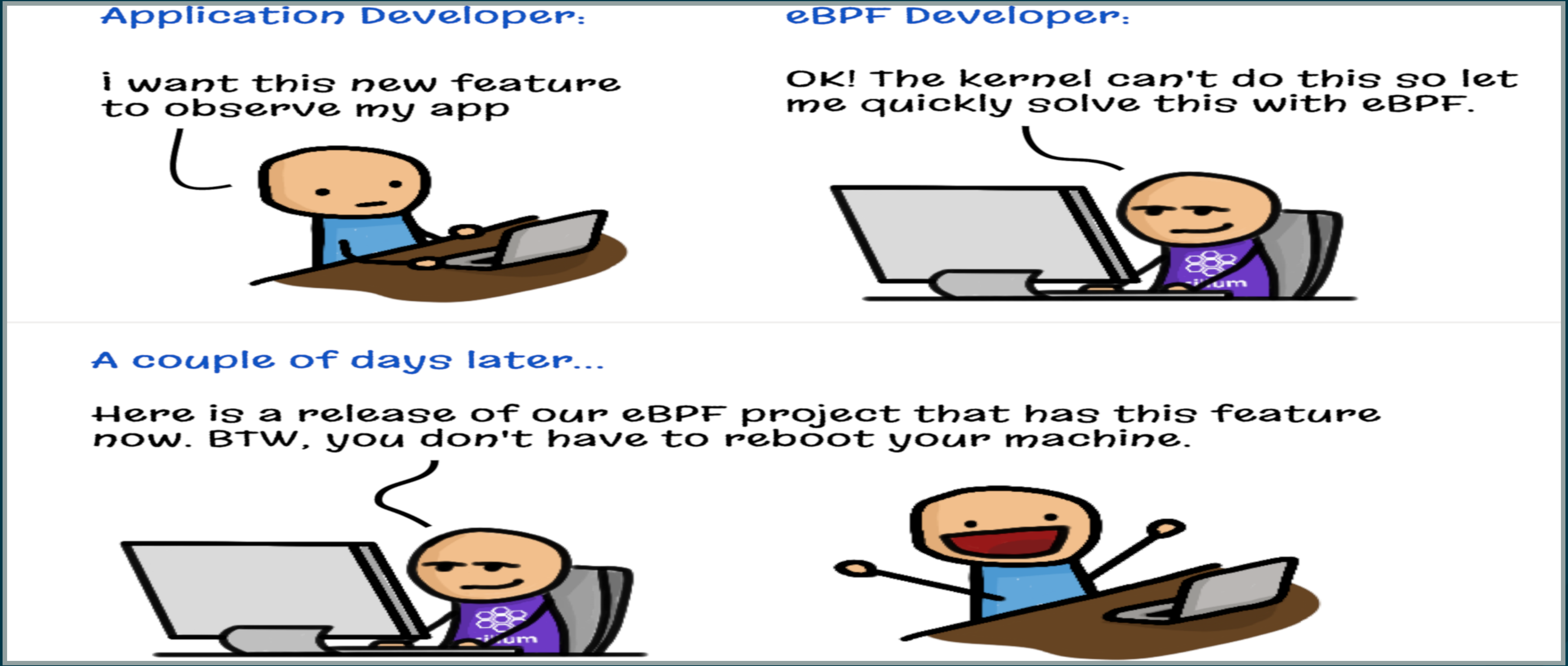
Rate of innovation at the operating system level: Traditionally slow

- BPF enables things at OS-level that were not possible before
- BPF will 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...

BPF 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.

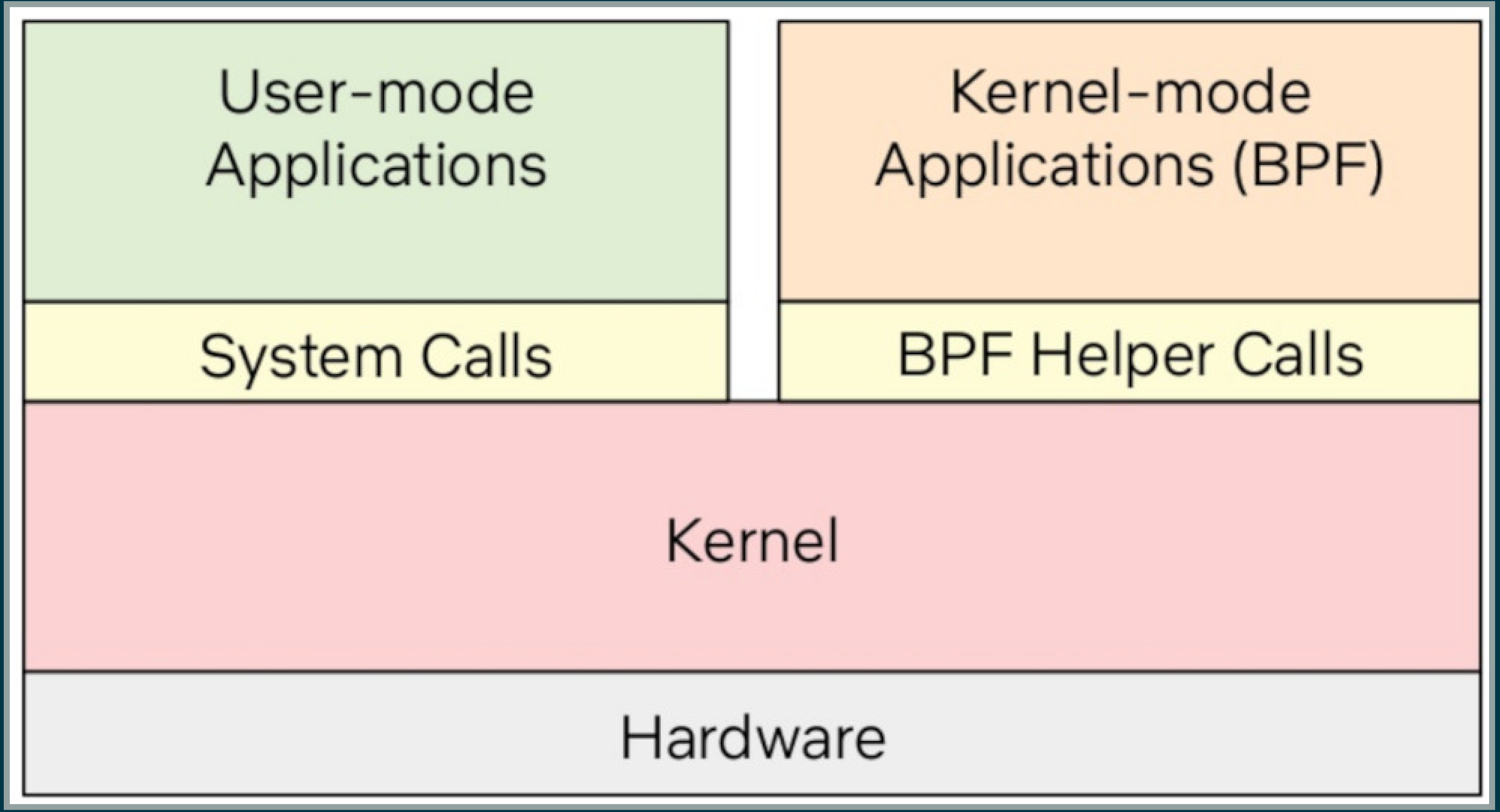
What is BPF from OS-vendor perspective

From an OS vendor (like Red Hat) perspective

- BPF is a fundamental change to a 50-year old kernel model
  - New interface for applications to make kernel requests, alongside syscalls
- BPF is “kernel-mode applications”
- See it as safer alternative to kernel modules
  - Better way to extend the kernel dynamically
  - Kernel developers: Do lose some control
  - … but positive for rate of innovation

A new Operating System Model

Modern Linux: becoming Microkernel-ish (from Brendan Gregg)



Replacing kernel modules

Known examples: Replacing Kernel module with BPF

- Sysdig/Falco: Started with Kernel module, now BPF since 2019
  - Competitor: Aqua Security also BPF based
  - Proprietary: CrowdStrike intrusion detection using kmod (customer ask)
- Cumucore Telco 5G/UPF: XDP replace GTP kernel module (closed source)
  - FOSS alternative: GitHub UPF-BPF

Adjust mental model of system development

As System Developers: Adjust your mental model

- No need to bring everything into userspace
- Can task be solved via kernel-side inline processing and manipulation?

Utilize BPF superpowers

- Linux Kernel has become a flexible tool at your disposal
- Run your own BPF kernel-mode application inside the kernel

BPF components

Closer look at the BPF 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 BPF Verifier
  - Limits and verifies memory access and instructions limit

BPF concepts: context, maps and helpers

Each BPF runtime hook gets a pointer to a context struct

- BPF bytecode has access to context (read/write limited)
  - Verifier adjusts bytecode when accessing struct members (for safety)

The BPF program itself is stateless

- BPF maps can be used to create state
- Maps are basically key = value containers
- Maps can hide complex features implemented on Kernel side

BPF helpers are used for

- Calling Kernel functions, to obtain info/state from kernel

BPF functional areas

BPF has hooks ALL over the kernel

- More interesting: What functional areas are these being used?

What (current) areas are BPF being used within?

- Networking (ahead as BPF started in networking)
  - Network control and data plane implemented in BPF (TC-BPF/XDP)
  - TCP Congestion Control implemented in BPF
- Security
- Observability (tracing)

Happy 7th Birthday BPF

Happy birthday BPF!

- 7 years old (See Alexei’s post Sunday 26 September 2021)
- XDP initial commit is approx 5 years + 2 months

Exciting things ahead

- eBPF Foundation (ebpf.io/charter) working towards standardisation
- Microsoft Windows introduce BPF in their kernel

Digital age and out-of-date documentation

BPF + LLVM features evolved over time

- Google search results: Many but out-dated articles
- See outdated approaches are used as best-practices :-(

Some quick advice to follow

- Use latest LLVM compiler (and `-target bpf`)
- Install latest pahole tool (used for BTF generation)
- Get Kernel with BTF (BPF Type Format) support
- Use new BPF-maps definitions (“.maps” section) with BTF support

Taming BPF superpowers - not easy

BPF superpowers - not easy to use - sorry

- Gain kernel level building block, that can be safely updated runtime
- Taming and learning-curve is challenging

BPF makes extending Kernel easier than Kernel modules

- Don’t confuse this with “easy to use”

BPF development is hard

Know this: BPF development is hard

- Mental model mind-shift: Coding “kernel-mode applications”
- Requires understanding internal kernel functions
- Available BPF features depend on LLVM compiler versions
- Developers will experience: Verfier rejecting programs
- Coding in Restricted C and passing verifier is frustrating
  - Corner-cases due to LLVM can be the issue
- Troubleshooting event based BPF-prog running kernel side is challenging

BPF user experience - could be better

BPF is great revolutionary technology!

- BUT end-user deployment experience can be rough

Recommend watching recent LPC 2021 talk by CloudFlare

- Talk: BPF user experience rough edges
- Covers 9 common pitfalls (with sad pandas)

BPF communities

Remember to reach out to BPF communities when stuck

- BPF Kernel developers: mailto:bpf@vger.kernel.org
- Slack channel: https://ebpf.io/slack
- LLVM compiler questions: mailto:iovisor-dev@lists.iovisor.org

XDP communities

- XDP-newbies: mailto:xdp-newbies@vger.kernel.org
- GitHub project: https://github.com/xdp-project
- IRC on oftc.net channel #xdp

BPF example code

Best documentation is BPF example code

Under XDP-project: github.com/xdp-project/

- bpf-examples - Practical BPF examples and build environment
- xdp-tutorial - Tutorial with assignments (Warning: uses old BPF-maps)
- xdp-tools - Tools (xdpdump) + libxdp for multiple XDP-progs on interface
- xdp-cpumap-tc - Show XDP + TC-BPF solving Qdisc lock scaling

BPF networking

Focus on BPF for networking

- XDP (eXpress Data Path) is our focus
- TC-BPF hooks are equally important for practical use-cases
- BPF hooks for cgroups can also be useful for containers

Why was an eXpress Data Path (XDP) needed?

Linux networking stack assumes layers L4-L7 are needed for every packet

- Root-cause of slowdown: (relative) high initial RX cost per packet

Needed to stay relevant as NIC speeds increase (time between packet small)

- New faster and earlier networking layer was needed to keep up.

XDP operate at layers L2-L3

- L4 load-balancer possible when no IP-fragmentation occurs

If you forgot OSI model:

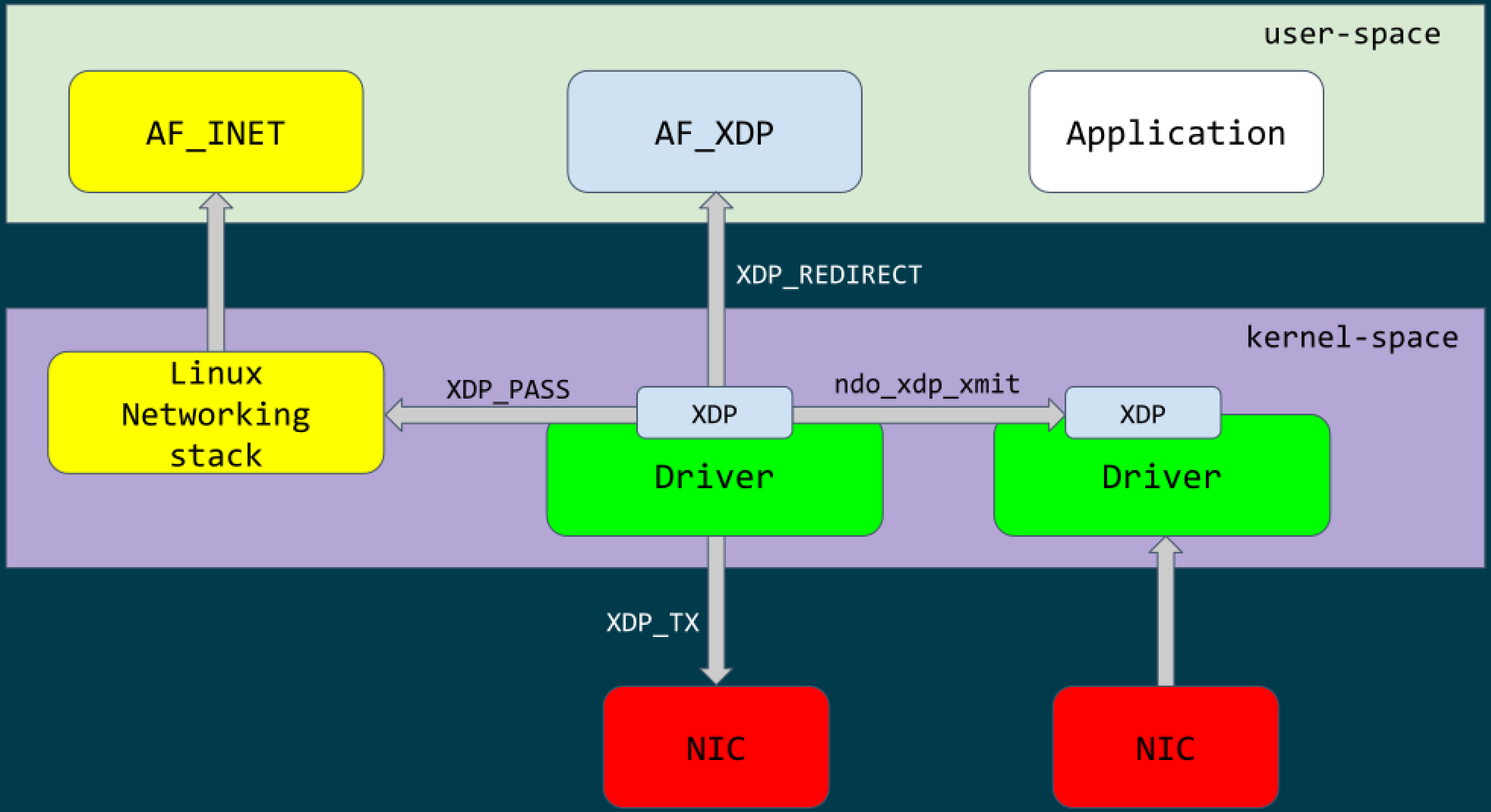
- L2=Ethernet
- L3=IPv4/IPv6
- L4=TCP/UDP
- L7=Applications

What is XDP?

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

- New 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 architecture



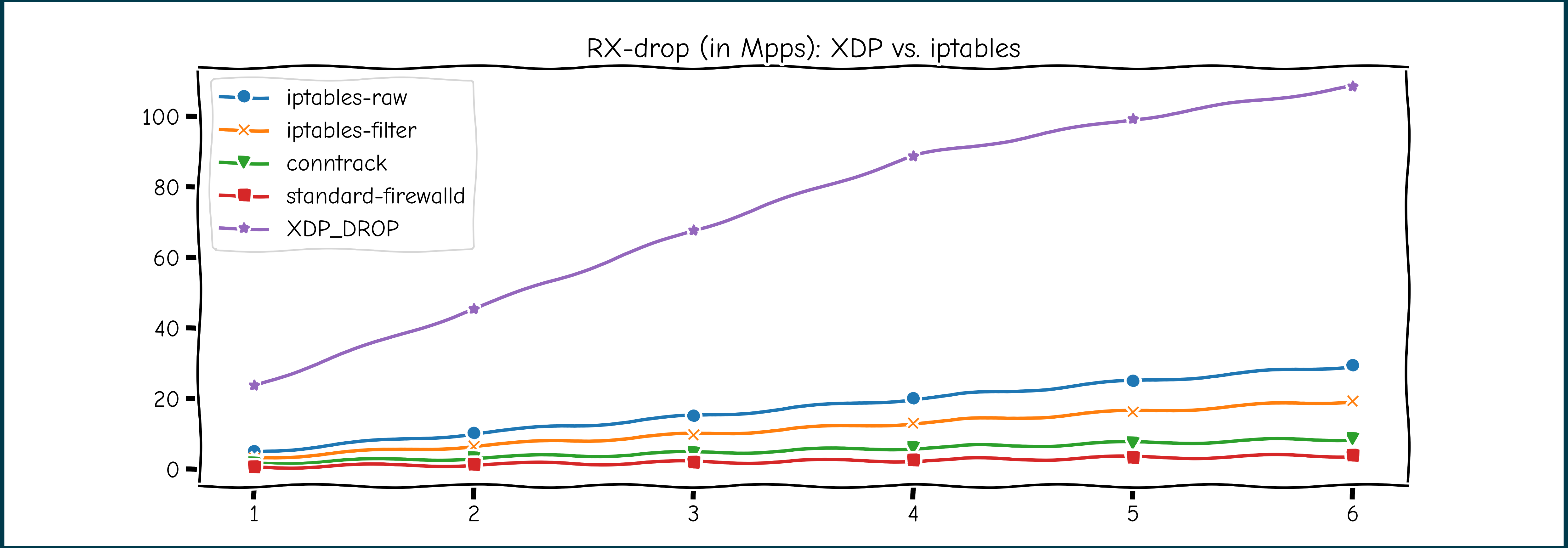
Performance graphs

Performance measurements taken from our XDP-paper

System used for testing

- Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz
- NIC driver mlx5: Mellanox ConnectX-5 Ex (MT28800)

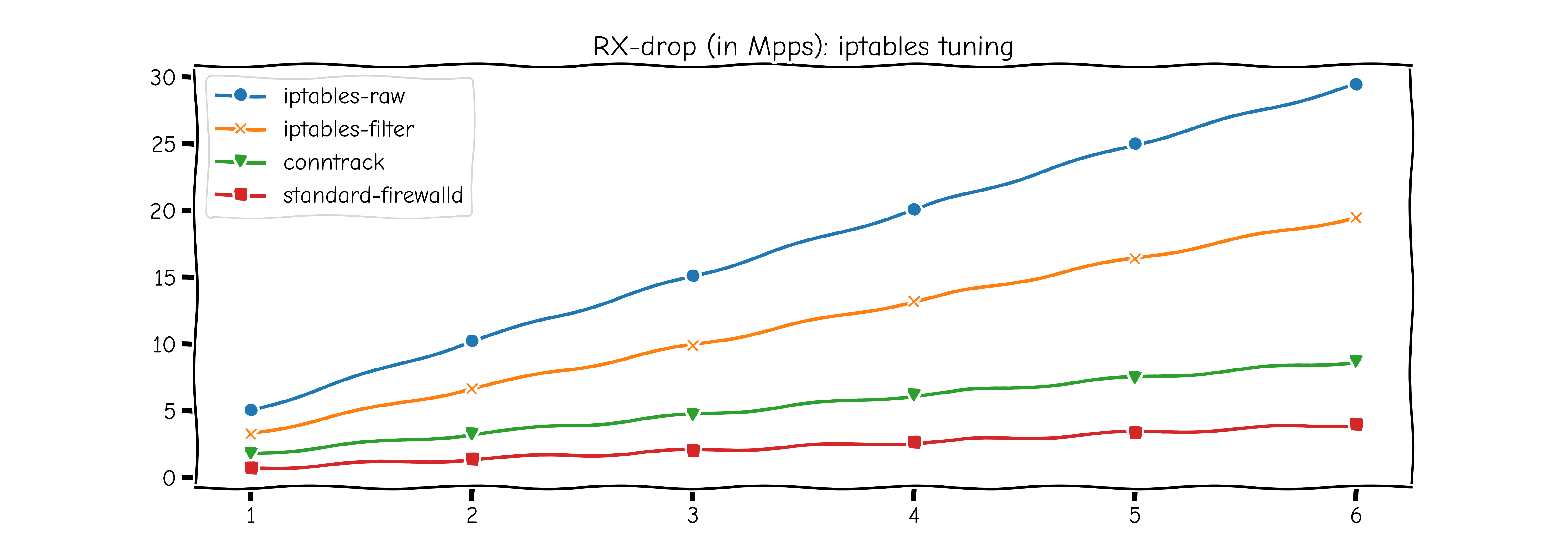
XDP performance



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`)

Zoom-in: on iptables performance tuning



iptables can be tuned to perform and scale well

- Especially if avoiding involving conntrack

XDP performance: both latency and throughput

XDP throughput and packet-per-second (PPS) performance super

- Real design goal is improving latency and throughput at same time

Designed with adaptive bulking

- Run as part of NAPI-poll (softirq) processing (max 64 frame budget)
- Pickup frames from RX-ring if available (no waiting for bulks)
- End of drivers NAPI-poll function flush any pending xdp_frames
- Another driver egress REDIRECT (devmap) flush every 16 frames
- REDIRECT to another CPU (cpumap) flush every 8 frames
- As load increase, bulking opportunities happen, system scale to load

XDP scaling across CPUs

XDP redirect to another CPU (via BPF cpumap type)

- Scalability mechanism: let XDP control on that CPU netstack runs

Allow to combine fast DDoS handing and slower netstack on same hardware

- Some CPUs run dedicated fast-path packet processing
- Delicate to other CPUs via XDP-redirect
- Remote CPUs receive raw xdp_frames, next steps:
  - ▪ Can (optionally) run XDP-prog for further filtering
  - ▪ Bulk allocate SKBs and start network stack on this CPU
- Attacker hitting netstack/app slow-path cannot influence fast-path

What is AF_XDP?

What is AF_XDP? (the Address Family XDP socket)

- Hybrid kernel-bypass facility, selectively move frames out of kernel
- XDP/BPF-prog filters packets using REDIRECT into AF_XDP socket
- Delivers raw L2 frames into userspace (via memory mapped ring buffer)

Realize: in-kernel XDP BPF-prog step opens opportunities

- Can augment/modify packets prior to AF_XDP delivery
  - E.g. record a timestamp at this early stage
- Use CPUMAP redirect: Move netstack traffic to other CPUs

WARNING: Next slides: Deep dive into AF_XDP details

- Most casual readers of slide deck can skip these details

Where does AF_XDP performance come from?

Lock-free channel directly from driver RX-queue into AF_XDP socket

- Single-Producer/Single-Consumer (SPSC) descriptor ring queues
- Single-Producer (SP) via bind to specific RX-queue id
  - NAPI-softirq assures only 1-CPU process 1-RX-queue id (per sched)
- Single-Consumer (SC) via 1-Application
- Bounded buffer pool (UMEM) allocated by userspace (register with kernel)
  - Descriptor(s) in ring(s) point into UMEM
  - No memory allocation, but return frames to UMEM in timely manner
- Transport signature Van Jacobson talked about
  - Replaced by XDP/eBPF program choosing to XDP_REDIRECT

Details: Actually four SPSC ring queues

AF_XDP socket: Has two rings: RX and TX

- Descriptor(s) in ring points into UMEM

UMEM consists of a number of equally sized chunks

- Has two rings: FILL ring and COMPLETION ring
- FILL ring: application gives kernel area to RX fill
- COMPLETION ring: kernel tells app TX is done for area (can be reused)

Gotcha by RX-queue id binding

AF_XDP bound to single RX-queue id (for SPSC performance reasons)

- NIC by default spreads flows with RSS-hashing over RX-queues
  - Traffic likely not hitting queue you expect
- You MUST configure NIC HW filters to steer to RX-queue id
  - Out of scope for XDP setup
  - Use ethtool or TC HW offloading for filter setup
- Alternative work-around
  - Create as many AF_XDP sockets as RXQs
  - Have userspace poll()/select on all sockets

XDP pain points resolved

Followup to Linux Plumber 2019: XDP the distro view

- Some of the pain points have been resolved

Multiple XDP programs on a single interface

Followup to Linux Plumber 2019: XDP the distro view

The library libxdp (available via xdp-tools)

- Have option of loading multiple XDP programs on a single interface
- See dispatcher API (`xdp_multiprog__*`) in README
- Depend on kernel feature `freplace` (read as: function replace)

XDP “tcpdump” packet capture

Tool ’tcpdump’ does not see all packets anymore.

- E.g XDP_DROP and XDP_REDIRECT etc.

New tool ’xdpdump’ (available via xdp-tools)

- Debug XDP programs already loaded on an interface
- Packets can be dumped/inspected:
  - Before on entry to XDP program
  - After at exit from an XDP program
    - Furthermore: at exit the XDP action is also captured
    - Can inspect XDP_DROP packets!
- Use Kernel features `fentry` + `fexit`
  - Also works with multi-prog dispatcher API

XDP future development

XDP multi-buff

- Allowing larger MTUs, Jumbo-frames and GRO/GSO compatibility

XDP-hints

- Extracting NIC hardware hints (from RX-descriptor)
- Traditional hints: RX-hash, RX-checksum, VLAN, RX-timestamp

Drivers without SKB knowledge

- based only on `xdp_frame`
- Depend on both XDP-hints + XDP multi-buff

End: Questions?

Resources:

- XDP-project - GitHub.com/xdp-project
  - Get an easy start with xdp-project/bpf-examples