eXpress Data Path (XDP)

Programmable and high performance networking data path

Tom Herbert <tom@herbertland.com>

Agenda

- Background: userspace stacks
- Building a foundation: BPF, and eBPF
- A solution: eXpress Data Path
- Looking forward

Userland frameworks

- Library, SDK, or other framework in userpsace
- Sits on top of a device access facility such as DPDK
- Completely bypasses the kernel
- Completely segregated from the kernel
- Useful in niches such as HFT

Why bother?

- Performance*
- “Safety” (isolation prevents crashing system)
- Developing in kernel is “difficult”
- There are a lot of userland programmers
- Reboot and upgrades are invasive to whole system

On the other hand...

- Userspace frameworks are difficult to use use for generic solutions
- Full TCP/IP stacks in userspace are hard
- Hybrid path solutions are ugly
- *Performance numbers tend to be overstated
- Maintaining separate stacks is a pain in deployment
- More proprietary solutions, less use of open source
- Extra constraints, e.g. huge pages in DPDK

Need programmable policy in kernel

- Long lead times until deployment. Lots of cruft...
- Without programmable policies, the kernel is in limbo forever
- New ABIs constantly being added to the kernel
- Because we cannot predict future policy needs
- Old ABIs fall into disuse, and can’t be removed
- Programmable policy ends this cycle for good

So what we really want...

A method integrated with the kernel stack that provides a means for users to program the networking stack on the fly that is safe, flexible, yields high performance, and encourages an ecosystem of solutions driven by the community!

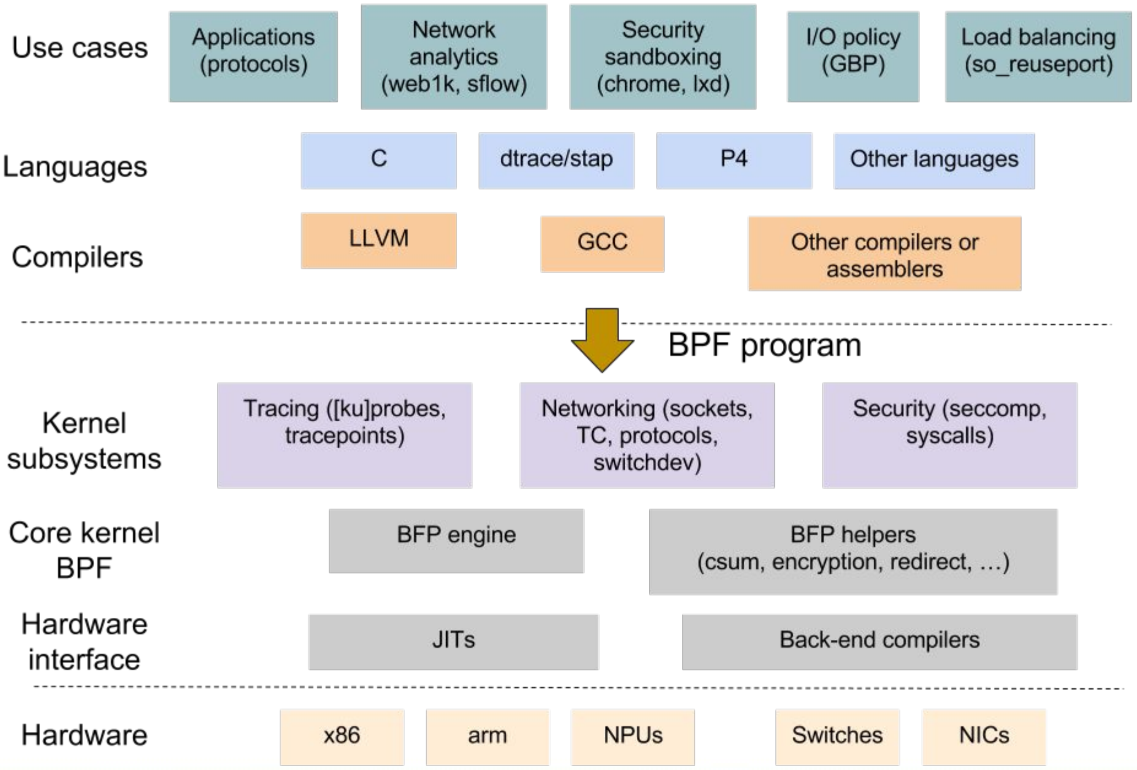
The beginnings of a solution

- In 1992 the first step of a long journey was taken at Lawrence Berkeley Labs
- Van Jacobson and Steven McCane saw that one part of kernel should be programmable
- And this led to Berkeley Packet Filter or just BPF
- Limited in scope to sockets, and mainly used for packet sniffing applications
- Limited capabilities, only two registers, byte code
- The full potential of BPF remained hidden for 24 years...

Fast forward to 2015

- Extended BPF (eBPF) was born
- Brainchild of Alexei Starovoitov, in the 3.18 Linux
- Full 64-bit engine, a dozen or so registers
- Comprehensive instruction set with atomic ops, etc.
- C code can be compiled to generate “safe” eBPF programs
- eBPF is used extensively in the kernel for all sorts of things (logging for instance)
- Some HW even adding support

BPF architecture



Use cases: Applications (protocols) | Network analytics (web1k, sflow) | Security sandboxing (chrome, lxd) | I/O policy (GBP) | Load balancing (so_reuseport)

Languages: C | dtrace/stap | P4 | Other languages

Compilers: LLVM | GCC | Other compilers or assemblers

BPF program

Kernel subsystems: Tracing ([ku]probes, tracepoints) | Networking (sockets, TC, protocols, switchdev) | Security (seccomp, syscalls)

Core kernel BPF: BFP engine | BFP helpers (csum, encryption, redirect, …)

Hardware interface: JITs | Back-end compilers

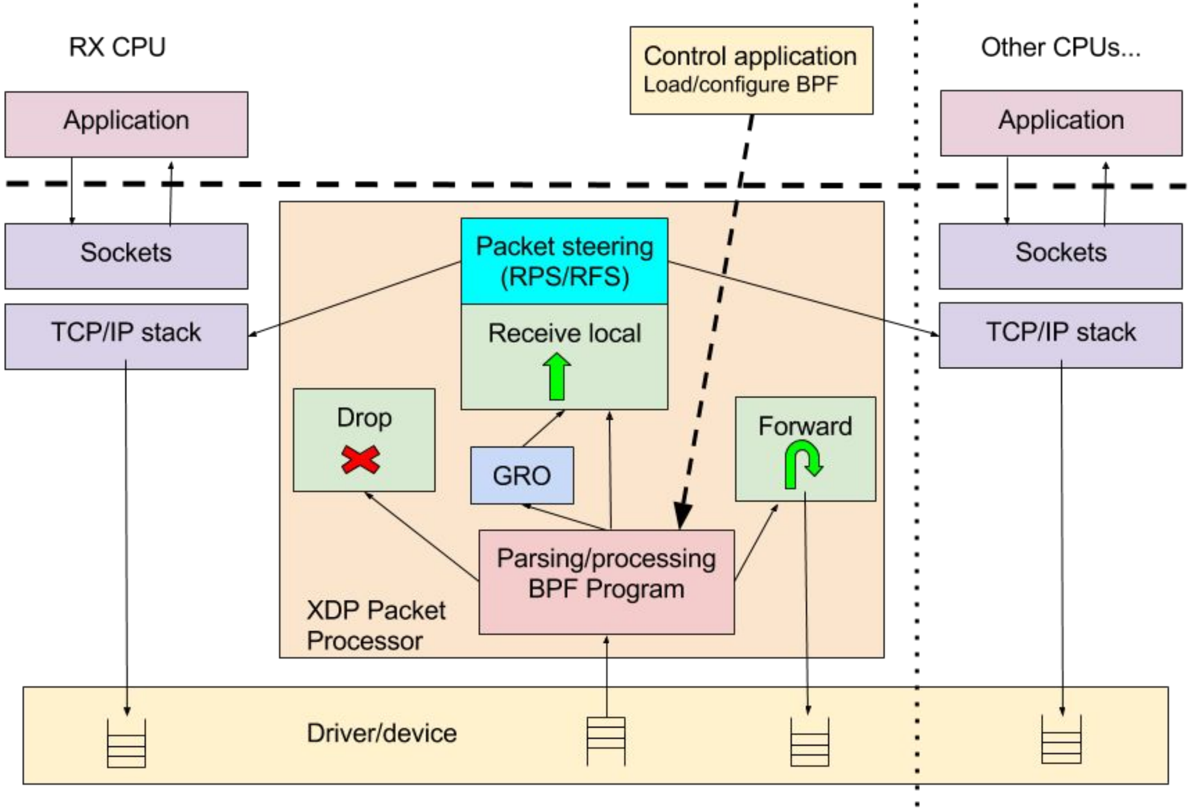
Hardware: x86 | arm | NPUs | Switches | NICs

Enter eXpress Data Path

- Conceived in 2016. Idea is to run eBPF programs at the earliest place possible in the stack
- Exactly when the device driver takes the packet from the RX ring
- XDP eBPF program returns a simple verdict:

  XDP_DROP, XDP_PASS, XDP_TX, XDP_ABORT
- XDP datapath lives in full harmony with rest of kernel networking stack

XDP Packet Procsessor



Properties

- XDP is **designed for high performance**. It uses known techniques and applies selective constraints to achieve performance goals
- XDP is also **designed for programmability**. New functionality can be implemented on the fly without kernel modification
- XDP is **not kernel bypass**. It is an integrated fast path in the kernel stack
- XDP does **not replace the TCP/IP stack**. It augments it and works in concert (*Don’t throw the baby out with the bathwater!*)
- XDP does **not require any specialized hardware**. It espouses the *less is more* principle for networking hardware

What XDP is being used for

- L2 and L3 protocol processing
- DDoS protection: e.g. filter and drop “bad IPs”
- More sophisticated DDoS protection where a an eBPF program looks for “patterns”
- Load balancing via XDP_TX verdict (e.g. replacement for IPVS)
- Switching, Routing, Tunnel termination… (e.g. ILA routing)

Some performance numbers

System: Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz, Mellanox mlx5 NIC

TC stack Drop	(1 core)	3.45Mpps (baseline)
XDP Drop	(1 core)	16.9Mpps
XDP TX	(1 core)	13.7Mpps
XDP TX	(24 threads)	45Mpps

The future

- Enable more drivers (5 currently support XDP)
- HW support (Netronome for one)
- Build out ecosystem of contributed solutions
- Packet batching for performance
- Continue to close gap with DPDK (~10%)
    - Advanced x86 instruction sets?
    - TLB advantage with huge pages?
    - Busy polling

Thank you!